Probabilistic Machine Learning for Pattern Recognition and Design Exploration

Von der Fakultät für Georessourcen und Materialtechnik der
Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von

M. Sc. Kevin Cremanns

aus Kleve

Berichter: Herr Univ.-Prof. Dr.-Ing. Stefan Reh
Herr Univ.-Prof. Dr.-Ing. Sebastian Münstermann
Herr Prof. Dr.-Ing. Dirk Roos

Tag der mündlichen Prüfung: 10.02.2021

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar

I dedicate this thesis to my family.

DANKSAGUNG

Mein besonderer Dank gilt Herrn Professor Dirk Roos für die Förderung dieser Arbeit und dessen hervorragende Betreuung. Des Weiteren möchte ich mich bei Herrn Professor Stefan Reh für die Übernahme des Erstgutachtens und die ebenfalls sehr förderlichen fachlichen Gespräche und Hinweise bedanken, die zur Qualität dieser Arbeit maßgeblich beigetragen haben. Ich danke auch Herrn Professor Stefan Münstermann für die Übernahme des Zweitgutachtens. Mein Dank gilt ebenfalls Can Bogoclu, der mit mir zusammen am Institut für Modellbildung und Hochleistungsrechnen der Hochschule Niederrhein gearbeitet hat. Die fachlichen Diskussionen und Hinweise haben mich oft inspiriert. Ich möchte mich bei allen Kollegen der Siemens AG, die ich während dieser Zeit kennenlernen durfte, bedanken. Die fachliche Unterstützung sowie die vielen hilfreichen Anregungen und Diskussionen haben einen großen Beitrag zu dieser Arbeit geleistet. Insbesondere gilt hier mein Dank meinen Betreuern Ralf Voß und Peter Dumstorff. Ganz besonders möchte ich mich bei meiner lieben Martyna für ihre Motivation und für ihr Verständnis bedanken. Zuletzt gilt mein größter Dank meiner Familie, deren Unterstützung mir erlaubt hat diesen Weg zu bestreiten.

ABSTRACT

The following thesis deals with the development of new probabilistic machine learning models with a focus on efficiency, flexibility, applicability, scalability and high prediction accuracy. Therefore, learning tasks from the areas of regression, classification, image classification, time series as well as the representation of spatial and / or temporally correlated quantities are examined. In order to achieve these goals and in addition to the development of the models themselves, their interaction to the field of data generation is also investigated. The calculable uncertainty of the probabilistic models is used to obtain the maximum information gain with as few data points as possible. This methodology is also used for the efficient solving of optimization problems. Besides obtaining an accurate approximation model of the data, a deeper understanding of the correlations between input and output is desired, therefore the topic of sensitivity analysis is a further part of this work. The developed probabilistic models are used to evaluate the number of calculations needed to estimate the sensitivities as efficiently as possible, even for correlated inputs.

ZUSAMMENFASSUNG

Die folgende Arbeit befasst sich mit der Entwicklung neuer probabilistischer Modelle auf Basis des maschinellen Lernens, mit dem Fokus auf Effizienz, Flexibilität, Anwendbarkeit, Skalierbarkeit und einer möglichst hohen Prognosegüte. Dabei werden Problemstellungen aus den Bereichen: Regression, Klassifizierung, Bilderkennung, Zeitreihen und die Abbildung von räumlich und / oder zeitlich korrelierten Größen behandelt. Um diese Ziele zu erreichen, wird neben der Entwicklung der Modelle, auch deren Zusammenspiel mit dem Themenfeld der Datenerzeugung untersucht. Dabei wird die berechenbare Unsicherheit der probabilistischen Modelle genutzt, um mit möglichst wenig Datenpunkten den maximalen Informationsgewinn zu erlangen. Diese Methodik wird ebenfalls für die effiziente Lösung von Optimierungsfragestellungen genutzt. Um nicht nur ein möglichst genaues Modell der Daten zu erhalten, sondern auch um ein tieferes Verständnis der Zusammenhänge zu erlangen, sind Methoden der Sensitivitätsanalyse ein weiterer Bestandteil dieser Arbeit. Die entwickelten probabilistischen Modelle werden dazu verwendet, die bei der Sensitivitätsanalyse notwendigen Berechnungen so effizient wie möglich durchzuführen, unter anderem auch für korrelierte Eingangsgrößen.

CONTENTS

List of Figures viii

List of Tables ix

Nomenclature x

1 INTRODUCTION 1
1.1 Motivation . . . 1
1.2 Outline of the Thesis . . . 3
1.2.1 Objectives . . . 3
1.2.2 Organization of the Text . . . 3
1.2.3 Associated Publications and Software . . . 4

2 SUPERVISED MACHINE LEARNING 7
2.1 Introduction . . . 7
2.2 Model Selection . . . 13
2.2.1 Cross-Validation . . . 14
2.2.2 Maximum Likelihood Estimation . . . 26
2.3 Heteroscedasticity in Regression . . . 30
2.4 Curse of Dimensionality . . . 32
2.5 Polynomials . . . 33
2.6 Automatic Relevance Determination . . . 38
2.7 Moving Least Squares . . . 40
2.7.1 Anisotropic Moving Least Squares . . . 42
2.7.2 Numerical Examples . . . 43
2.7.3 Heteroscedasticity . . . 48
2.8 Gaussian Processes . . . 50
2.8.1 Noise-free Gaussian Processes . . . 51
2.8.2 Noisy Observations . . . 57
2.8.3 Free Parameter Estimation . . . 59
2.8.4 Classification . . . 61
2.8.5 Stationary Covariance Functions . . . 62
2.8.6 Non-Stationary Covariance Functions . . . 65
2.8.7 Anisotropic Gaussian Processes . . . 67
2.8.8 Heteroscedastic Noise Estimation . . . 67
2.8.9 Multi-Output . . . 72
2.8.10 Further Extensions to Gaussian Processes . . . 72
2.9 Neural Networks . . . 75
2.9.1 Introduction and Notation . . . 75
2.9.2 Stochastic Gradient Descent . . . 79
2.9.3 Backpropagation . . . 81

2.9.4 Advanced Methods . . . 82
2.10 Deep Gaussian Covariance Network . . . 87
2.10.1 Review of Algorithms . . . 87
2.10.2 Mathematical Formulation . . . 90
2.10.3 Implementation Details and Scalability . . . 104
2.10.4 Sequential Problems . . . 106
2.10.5 Multi-Output . . . 109
2.10.6 Classification . . . 122
2.10.7 Image Classification . . . 126
2.10.8 Further Test Cases . . . 128
2.10.9 Conclusion . . . 145
2.11 Model Validation . . . 147
2.11.1 Test Data Validation . . . 147
2.11.2 Confidence Intervals . . . 153
2.11.3 Confusion Matrix . . . 155
2.12 Conclusion . . . 157

3 DESIGN OF EXPERIMENTS 159
3.1 Introduction . . . 159
3.2 Overview of Methods . . . 160
3.2.1 Systematic Plans . . . 160
3.2.2 Stochastic Plans . . . 161
3.3 Optimization of Latin Hypercube Sampling . . . 166
3.3.1 Distance-based Criteria . . . 166
3.3.2 Correlation-based Criteria . . . 168
3.3.3 Optimization Algorithm . . . 174
3.3.4 Influence on Surrogate Model Training . . . 178
3.4 Adaptive Sampling & Bayesian Optimization . . . 184
3.4.1 Sequential Latin Hypercube Sampling . . . 184
3.4.2 One-by-One Adaptation . . . 193
3.5 Sampling Reconstruction . . . 208
3.5.1 Inverse Transform Sampling . . . 208
3.5.2 Novelty-Detection-based Rejection Sampling . . . 211
3.6 Conclusion . . . 220

4 SENSITIVITY ANALYSIS 223
4.1 Introduction . . . 223
4.2 Types of Correlation Coefficients . . . 225
4.2.1 Overview . . . 225
4.2.2 Comparison . . . 226
4.3 Coefficient of Importance . . . 232
4.3.1 Overview . . . 232
4.3.2 Numerical Example . . . 233
4.4 Global-Variance-based Analysis . . . 235
4.4.1 Overview . . . 235
4.4.2 Numerical Computation of Sobol Indices . . . 238

4.4.3 Sobol Indices for Dependent Parameters . . . 239
4.4.4 Numerical Examples . . . 243
4.5 Derivative-based Global Sensitivity Measures . . . 249
4.5.1 Overview . . . 249
4.5.2 Numerical Computation . . . 250
4.5.3 Example . . . 251
4.6 Conclusion . . . 252

5 CONCLUSION 255
5.1 Summary of Contributions . . . 255
5.2 Future Work . . . 258

BIBLIOGRAPHY 261

Appendix A IMAN AND CONOVER RANK CORRELATION METHOD 285
A.1 Mathematical Formulation . . . 285
A.1.1 Declaration of Notation . . . 285
A.1.2 Mathematical Formulation of the Rank Correlation Method . . . 285

LIST OF FIGURES

Figure 2.1 Traditional programming vs. machine learning. Source: Barpaga [2019]. 7
Figure 2.2 Industries targeted by machine learning application developers. Source: Google [2016a], 2016, $N = 1,441$ respondents. 8
Figure 2.3 Benefits of machine learning. Source: Google [2016b], 2016, $N \approx 68$ companies were asked what they expect to gain from the use of machine learning. 8
Figure 2.4 Overview of supervised machine learning. 10
Figure 2.5 Overview of machine learning related learning and data types. . . . 11
Figure 2.6 Overview of model selection. 12
Figure 2.7 Illustration of k-fold cross-validation with $k = 5$ and sample size $N = 10$. 14
Figure 2.8 Stratified k-fold cross-validation with $k = 5$ and $N = 25$ for symmetric regression problems. 16
Figure 2.9 k-fold ($k = 6$) cross-validation for a training data set ($N = 30$) with sorted parameter x_1 without subsequent shuffling. 17
Figure 2.10 k-fold ($k = 6$) cross-validation for a training data set ($N = 30$) with sorted parameter x_1 with subsequent shuffling. 18
Figure 2.11 k-fold ($k = 6$) stratified cross-validation for a training data set ($N = 30$) with sorted parameter x_1. 19
Figure 2.12 Modified Branin-Hoo function. 19
Figure 2.13 Himmelblau function. 20
Figure 2.14 Eggholder function. 20
Figure 2.15 Gaussian process model with free parameters estimated on the full data set. Two points are left out for a CV test subset and should be predicted by the model. The model in the middle is updated and the model at the bottom is retrained on the subset. The test point annotated with 1^* is used to show the covariance values (illustrated via color) to the other training points as an indicator for the correlation length-scale of the retrained and updated $\mathcal{GP}$ models (see section 2.8). 21
Figure 2.16 Examples of fitting a Gaussian distribution via MLE on $N = 3$ and $N = 50$ training points. In average the mean is correct, but the variance is underestimated for $N = 3$. 28
Figure 2.17 Example of fitting two $\mathcal{DGCN}$ models via MLE on $N = 70$ training points. The upper model has a fixed noise parameter $\sigma^2 = 0$ (so no noise for the training) and the lower one was allowed to train its noise parameter. This results in a different fit of the models and also in a different prediction of the standard deviation $\hat{\sigma}$ of the prediction. The true function is $f(x) = 3\sin(2x)$, $x \in \mathbb{R}^{[0,10]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. $\hat{f}(x)$ is the approximation and $f(x)$ the true function. 29

Figure 2.18 Two RSM models trained on a sinusoidal function with constant (top) and dynamic (bottom) amplitude. Furthermore, the residual vs. prediction is plotted to show the homoscedasticity and heteroscedasticity. . . . 31
Figure 2.19 RSM trained for $m = 6$, 12, 16 (top $\rightarrow$ down) and $N = 120$. Evaluated the best m value via 10-fold CV and the $\hat{R}^2$ metric. The true function is $f(x) = 3\sin(2x)$, $x \in \mathbb{R}^{[0,10]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. . . . 37
Figure 2.20 RSM models trained with a full linear model according to eq. 2.43 and a reduced model according to eq. 2.44 next to the true model function. x_3 is set to be 0 for this plot. . . . 39
Figure 2.21 Gaussian weighting function depending on r and scaled distance $\|\boldsymbol{x}_* - \boldsymbol{x}_i\|, \in^{[0,1]}$. . . . 42
Figure 2.22 Test function 1, $N = 50$, $x1, ..., x5 \in \mathbb{R}^{[0,1]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data. . . . 44
Figure 2.23 Test function 2, $N = 50$, $x1, ..., x5 \in \mathbb{R}^{[0,1]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data. . . . 45
Figure 2.24 Test function 3, $N = 150$, $x1, ..., x5 \in \mathbb{R}^{[-\pi,\pi]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data. . . . 45
Figure 2.25 Test function 4, $N = 100$, $x1, ..., x5 \in \mathbb{R}^{[-\pi,\pi]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data. . . . 46
Figure 2.26 Surrogate model comparison (top $AMLS$, bottom MLS) for test function 3, 3D view for parameter x_1, x_2. All variables not shown are set constant to their mean value estimated on the training data. 47
Figure 2.27 Surrogate model comparison (top $AMLS$, bottom MLS) for test function 4, 3D view for parameter x_2, x_3. All variables not shown are set constant to their mean value estimated on the training data. 48
Figure 2.28 Two MLS trained on a sinusoidal function with constant (top) and dynamic (bottom) amplitude. Furthermore, the residual vs. prediction is plotted to show the homoscedasticity and heteroscedasticity of the error. . . . 49
Figure 2.29 The residual vs. prediction plot for test function 2 (left) and 3 (right) of the $AMLS$ model. . . . 49
Figure 2.30 Top: independent bivariate Gaussian distribution. Bottom: dependent bivariate Gaussian distribution. . . . 53
Figure 2.31 One dimensional $\mathcal{GP}$ model with zero mean and variance of one. Top: shows 5 functions drawn from a $\mathcal{GP}$ prior without any observations. Middle and bottom show 5 functions drawn from the $\mathcal{GP}$ posterior distribution over functions, so the prior conditioned on $N = 10$ and $N = 50$ noise-free observations. The shaded area illustrate the 95% confidence region of the $\mathcal{GP}$. . . . 54

Figure 2.32 $\mathcal{GP}$ model with a squared exponential covariance function. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$. Number of training points $N = 20$. The top picture shows the trained model on the noise-free data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l which should be minimized in order to fit the data. 56
Figure 2.33 $\mathcal{GP}$ model with a squared exponential covariance function. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. The number of training points $N = 20$. The top picture shows the trained model on the noisy data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l which should be minimized in order to fit the data. 57
Figure 2.34 $\mathcal{GP}$ model with a squared exponential covariance function and additional noise parameter σ_n^2. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. Number of training points $N = 20$. The top picture shows the trained model on the noisy data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l with a fixed valued $\sigma_n^2 = 0.09$ which should be minimized in order to fit the data. 59
Figure 2.35 The top picture shows a one dimensional Gaussian process $\hat{f}(x)$ and the bottom picture shows the result of using the logistic function $\lambda(z) = (1 + \exp(-z))^{-1}$ to obtain the class probability $\pi(x) = \lambda(\hat{f}(x))$. 61
Figure 2.36 Comparison of a $\mathcal{GP}$ model with different covariance functions trained on $N = 200$ samples. 65
Figure 2.37 $\mathcal{GP}$ model with a summation of four different kernels to address different trends in the measured CO_2 data from 1959-1998 + 30 years future prediction. The 545 observations and the fitted model together with the 95% confidence interval of its prediction are shown. It can be seen that the extrapolation into the future results in increasing uncertainty of the model. 66
Figure 2.38 Comparison of two $\mathcal{GP}$ models with a stationary and non-stationary covariance function. Both use a squared exponential kernel. The second level $\mathcal{GP}$, to estimate local length-scales, has also a squared exponential covariance function. 68
Figure 2.39 Comparison of two $\mathcal{GP}$ covariance matrices of a stationary and non-stationary covariance function. The color represents the covariance value. 69
Figure 2.40 Isotropic (left) and anisotropic (right) $\mathcal{GP}$ models with squared exponential covariance function for the test function of section 2.7, eq. 2.49 to 2.52 named as testfunction 1-4. 70
Figure 2.41 Comparison for a simple linear function $y = x + \mathcal{N}(0, x^2)$ with some heteroscedastic noise dependent on x of a homoscedastic and heteroscedastic $\mathcal{GP}$ noise model. 71
Figure 2.42 Neural network with 4 layers, 1 input, 2 hidden and 1 output layer. 75

Figure 2.43 Neural network activation function types. 76
Figure 2.44 Single neuron network (perceptron). 77
Figure 2.45 Deep neural network with 4 layers and two outputs. 77
Figure 2.46 Deep neural network with 5 layers and two outputs. 78
Figure 2.47 Comparison of different learning rates for the training of a neural network. 80
Figure 2.48 Loss over multiple training steps with different learning rates indicated through the arrow length. The numbers stand for the training step and the ball shows the current loss at the corresponding training step. 82
Figure 2.49 Recurrent neural network. 84
Figure 2.50 $LSTM$ network. The activation function is a complex system of gates (sigmoid functions) and mathematical operations. These special activation functions can be chained over the number of sequences which should be considered. 84
Figure 2.51 Dropout sets random activation functions results to zero. 85
Figure 2.52 Dropconnect sets random weights to zero and provides even more possible network configurations compared to dropout. 85
Figure 2.53 Small neural network containing 2 convolutional layers and 1 pooling layer for the MNIST data set of handwritten digits. Accuracy is 91.5% on a 10,000 images test set. 86
Figure 2.54 Illustration of a CNN which learned from handwritten digits (MNIST data set LeCun [1998]). The image is taken from the visualization tool of Harley [2015]. 86
Figure 2.55 $\mathcal{DGCN}$ model with non-stationary noise variance estimation. 93
Figure 2.56 $\mathcal{DGCN}$ model without noise variance estimation. 94
Figure 2.57 $\mathcal{DGCN}$ model for training data with 3 outliers for a sine test function: $y = \sin(x)$, $x \in \mathbb{R}^{[0,15]}$. 94
Figure 2.58 $\mathcal{DGCN}$ model for training data with a random noise $y_{noisy} = y + 3(0.5 - \mathcal{U}[0,1])$ for a sine test function: $y = \sin(x)$, $x \in \mathbb{R}^{[0,15]}$. . . 95
Figure 2.59 Comparison of the non-stationary $\mathcal{DGCN}$ model and a stationary $\mathcal{GP}$ model with squared exponential covariance function. 97
Figure 2.60 Plot of the length-scale parameters of 5 different covariance functions. It can be seen that all of them, predicted by the neural network, change with the value of x. 97
Figure 2.61 Non-stationary $\mathcal{DGCN}$ covariance matrix. The red dashed lines mark where the function makes a step. 98
Figure 2.62 Stationary $\mathcal{GP}$ covariance matrix with squared exponential covariance function. The red dashed lines mark where the function makes a step. 98
Figure 2.63 Comparison of $\mathcal{DGCN}$ model surface over x_1, x_2 (black wireframe) and the true function (red wireframe). Black points indicate training points which consists of a global and local sampling. 100

Figure 2.64 Comparison of $\mathcal{GP}$ model function surface with squared exponential covariance over x_1, x_2 (black wireframe) and the true function (red wireframe). Black points indicate training points which consists of a global and local sampling. 100
Figure 2.65 Covariance matrix of the non-stationary $\mathcal{DGCN}$ model for 50 training points (30 globally sampled (0-30 on the x-axis) and 20 locally sampled (30-50 on the x-axis)) and 40 test points (20 globally sampled (0-20 on the y-axis) and 20 locally sampled (20-40 on the y-axis)).101
Figure 2.66 Covariance matrix of the stationary $\mathcal{GP}$ model for 50 training points (30 globally sampled (0-30 on the x-axis) and 20 locally sampled (30-50 on the x-axis)) and 40 test points (20 globally sampled (0-20 on the y-axis) and 20 locally sampled (20-40 on the y-axis)). 101
Figure 2.67 Length-scales of 5 different covariance functions used in the $\mathcal{DGCN}$ model over the 40 test points. Unwanted correlation functions: absolute exponential and Matérn $v = 3/2$ are deselected by setting the correlation length to zero. Since this function is very smooth, it can be better approximated by smooth covariance functions. . . 102
Figure 2.68 $\mathcal{DGCN}$ model for the CO_2 example with automatically build covariance matrix. 103
Figure 2.69 Schematic $\mathcal{DGCN}$ model overview with only one input parameter (for simplicity). The solid lines illustrate connections with trainable $W, \boldsymbol{b}$ parameters and the dashed lines are intended to illustrate the flow of processing the data. 103
Figure 2.70 Training time comparison for different batch sizes N_b on CPU and GPU. 107
Figure 2.71 Narendra and Parthasarathy [1990] time dependent test function with a $\mathcal{DGCN}$ model and a shift of input and output of 3. 109
Figure 2.72 For comparison the Narendra and Parthasarathy [1990] time dependent test function with a $\mathcal{DGCN}$ model and a shift of input and output of 2 and 1. 110
Figure 2.73 For comparison the Narendra and Parthasarathy [1990] time dependent test function (only first 100 points) with a $\mathcal{DGCN}$ model with and without start points for the prediction. 110
Figure 2.74 Air compressor storage system. 111
Figure 2.75 Comparison of a recurrent neural network and $\mathcal{DGCN}$ for the electrical power consumption (top) and the storage pressure (bottom) of an air storage system. 112
Figure 2.76 Illustration of an autoencoder with one hidden layer for the encoding from 4 inputs to 2 and back to 4 for the decoding. 113
Figure 2.77 Examples of training signals of test function 2.138. 114
Figure 2.78 Examples of training signals transformed via PCA of test function 2.138. 115
Figure 2.79 Comparison of different algorithms without PCA on test function 2.138, first 5 test signals are shown. 116
Figure 2.80 Comparison of different algorithms with PCA on test function 2.138, first 5 test signals are shown. 117

Figure 2.81 Comparison of different algorithms without PCA on test function 2.138, one model per output parameter was trained, first 5 test signals are shown. . . . 120
Figure 2.82 Comparison of different algorithms with PCA on test function 2.138, one model per output parameter was trained, first 5 test signals are shown. . . . 121
Figure 2.83 Comparison of different algorithms without PCA on test function 2.138, where the output parameters are aligned as a vector, first 5 test signals are shown. . . . 122
Figure 2.84 Rotor model for the finite element analysis multi-output example. Schematic boundary conditions are visualized. . . . 123
Figure 2.85 Comparison of predicted and true FEM node results for different pressure values. . . . 123
Figure 2.86 Turbine-generator train with simple model of rotor support. . . . 126
Figure 2.87 Example of plotting the class probabilities for the turbine-generator train example. Points illustrate the $1,000$ test points. The contour plot represents the class probability from 0 (blue) no resonance to 1 (red) resonance. . . . 127
Figure 2.88 Generic illustration of how $\mathcal{DGCN}$ is coupled with a convolutional neural network. . . . 127
Figure 2.89 Examples of the MNIST handwritten digits data set. . . . 129
Figure 2.90 Examples of the CIFAR10 data set. . . . 129
Figure 2.91 CATS benchmark $4,900$ data points and 100 missing points. . . . 130
Figure 2.92 CATS benchmark 5 blocks comparison of prediction and true result for the missing 100 points. . . . 132
Figure 2.93 Auto-correlation for all blocks with shift= 20, 50, 70, 100, 150, 200, 250, 300, 400, 500. . . . 133
Figure 2.94 Visualization of the rotor to be analyzed. . . . 137
Figure 2.95 Creep strain over time and space. . . . 137
Figure 2.96 Comparison of different algorithms for the creep strain approximation over space and time with PCA pre-processing. . . . 138
Figure 2.97 Comparison of different algorithms for the creep strain approximation over space and time without PCA pre-processing. . . . 138
Figure 2.98 Illustration of the small punch test. A stamp is pressed on a material sample until it fails. . . . 139
Figure 2.99 Force vs. displacement diagram of a small punch test. . . . 139
Figure 2.100 Reconstruction of the autoencoder compression to the full $1,000$ steps. 140
Figure 2.101 Comparison of the calibrated FEM curve vs. experimental curve. . 140
Figure 2.102 Overview of the different departments involved in the design development of a steam turbine rotor. The red marked text gives a generic overview of the input parameters used. . . . 142
Figure 2.103 Overview of the different departments involved in the design development of a steam turbine rotor. The red marked text gives a generic overview of the input parameters used. Surrogate models should directly learn the multi physical relationships from the input parameters used to the final rotor relevant output parameters. . . . 143

Figure 2.104 3D and all 2D Pareto-frontier combinations of the rotor optimization. Darker points are more close than gray points to visualize the third dimension. . . . 144
Figure 2.105 Residual plot without anomalies. . . . 149
Figure 2.106 Residual plot with unbalanced y-axis values. . . . 149
Figure 2.107 Heteroscedastic residuals. . . . 150
Figure 2.108 Non-linear data residual plot. . . . 150
Figure 2.109 Residual plot with outlier. . . . 151
Figure 2.110 Residual plot with large y data. . . . 151
Figure 2.111 Residual plot with unbalanced x-axis values. . . . 152
Figure 2.112 Example of a quadratic function transformed via Box-Cox power transformation for different values of λ. . . . 153
Figure 2.113 Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained with a start DoE of 30 samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence interval. Black points show training points. . . . 154
Figure 2.114 Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained with an extended local refined DoE of 35 samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence bounds. Here only the additional training samples are plotted (black points). . . . 154
Figure 2.115 Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained three times with $N = 10, 20, 40$ samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence bounds. Training samples are plotted (black points). 155
Figure 2.116 Confusion matrix for the MNIST example. Non-diagonal elements show the amount of miss-classified digits. . . . 156
Figure 3.1 Overview of the application of design of experiment and possible interference factors. . . . 159
Figure 3.2 Example of a inadequate quality data set. . . . 160
Figure 3.3 Full factorial design plan with $n = 2$, $n = 3$ and $l = 2$, $l = 3$. . . . 162
Figure 3.4 Central composite design plan with $n = 2$, $n = 3$. . . . 162
Figure 3.5 Experimental costs for a full factorial design plan with $l = 2$, $l = 3$ and different values for n. . . . 162
Figure 3.6 Monte Carlo design plan with $n = 2$, $n = 3$ and $N = 4, 2$. The black histograms should illustrate the empirical distribution for the parameters. $\sim \mathcal{U}$ should indicate the target distribution. . . . 163
Figure 3.7 Latin hypercube design plan with $n = 2$, $n = 3$ and $N = 4, 2$. The black histograms should illustrate the empirical distribution for the parameters. $\sim \mathcal{U}$ should indicate the target distribution. . . . 164
Figure 3.8 Correlated and optimized Latin hypercube design plans. . . . 164
Figure 3.9 Comparison of the reduced data set of a LHS and full factorial design plan if one input parameter (x_3) is removed. . . . 165
Figure 3.10 Comparison of distance-based objective for $N = 20$ and $n = 2$. . . . 169

Figure 3.11 Scatter matrix plot and pairwise linear correlation coefficients for the E^{AE} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 170
Figure 3.12 Scatter matrix plot and pairwise linear correlation coefficients for the E^{EMM} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 170
Figure 3.13 Scatter matrix plot and pairwise linear correlation coefficients for the E^{Φ_p} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 171
Figure 3.14 Scatter matrix plot and pairwise linear correlation coefficients for the E^{CL_2} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 171
Figure 3.15 Scatter matrix plot and pairwise linear correlation coefficients for the E^{ME} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 172
Figure 3.16 Scatter matrix plot and pairwise linear correlation coefficients for the E^{CN} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 175
Figure 3.17 Scatter matrix plot and pairwise linear correlation coefficients for the E^{PCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 175
Figure 3.18 Scatter matrix plot and pairwise linear correlation coefficients for the E^{MAPCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 176
Figure 3.19 Scatter matrix plot and pairwise linear correlation coefficients for the E^{MADCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 176
Figure 3.20 Scatter matrix plot and pairwise linear correlation coefficients for the E^{CD} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given. . . . 177
Figure 3.21 Illustration of the design points permutation. . . . 178
Figure 3.22 Illustration of the sequential LHS approach for $N_1 = 2$, $N_2 = 4$ and $N_3 = 8$. The green fields show the possible free bins for the sequential extension. . . . 185
Figure 3.23 Comparison of a sequential LHS vs. a one-shot LHS up to $N = 16$ samples, optimized via the E^{ME} criterion. . . . 186
Figure 3.24 Comparison of different distance-based criteria for the sequential LHS. The magenta colored points show the previous DoE and the white points the next adaptation. The contour plots illustrate the variance (uncertainty) of the $\mathcal{DGCN}$ model based on the magenta colored points. The DoE sizes are $N = 10, 20, 40$. The left column shows the first adaptation the right column the second adaptation. 188
Figure 3.25 Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20, 40$ optimized via the $E^{ME_{cond}}$ objective with $n = 2$. 1,000 test points are shown. . . . 190

Figure 3.26 Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20, 40$ optimized via the $E^{ME_{cond}}$ objective with $n = 5$. 1,000 test points are shown. 191
Figure 3.27 Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20, 40$ optimized via the E^{ME} objective with $n = 5$. 1,000 test points are shown. 192
Figure 3.28 Illustration of the expected improvement based on a Gaussian process model prediction. 195
Figure 3.29 Exemplary Pareto-front for two objectives which should be minimized. 197
Figure 3.30 Expected-improvement-based optimization of the modified Branin-Hoo function. The global optimum is at $x_1 \approx -3.69$, $x_2 \approx 13.63$ with $y \approx -16.64$. A start DoE optimized via the E^{CL_2} criterion of $N = 10$ samples was used. The numbers indicate the order of the samples during the optimization. 199
Figure 3.31 Expected-improvement-based optimization of the Eggholder function. The global optimum is at $x_1 = 512$, $x_2 \approx 404.23$ with $y \approx -959.64$. A start DoE optimized via the E^{CL_2} criterion of $N = 50$ samples was used. The numbers indicate the order of the samples during the optimization. 200
Figure 3.32 Overview of test function and the Pareto-frontiers estimated on the numerical model via optimization. The gray area illustrates the feasible domain. For the WeldedBeam example, the Pareto-front makes up only a very small part of the feasible domain. 203
Figure 3.33 Overview of the 6 objectives WATER test function. 204
Figure 3.34 Surrogate model standard-error-based one-by-one sampling for the Eggholder test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 190 points. 204
Figure 3.35 Surrogate model standard-error-based one-by-one sampling for the Himmelblau test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 30 points. 205
Figure 3.36 Surrogate model standard-error-based one-by-one sampling for the modified Branin-Hoo test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 30 points. 205
Figure 3.37 Surrogate model standard-error-based one by one sampling for the combined Eggholder, Himmelblau and modified Branin-Hoo test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 50 points. 207
Figure 3.38 Schematic overview of the uniform and exponential distribution type. 209
Figure 3.39 Illustration of the inverse transform sampling method. 209
Figure 3.40 Illustration of the inverse transform sampling method for a LHS from uniform to normal distribution. 209
Figure 3.41 Stepwise inverse transform sampling for unknown CDF and PPF. 233 measured samples and $1,000$ reconstructed samples. 210

Figure 3.42 Measured operational data from a steam turbine (top) and 500 new samples with the same multivariate distribution (bottom). 211
Figure 3.43 Demonstration of novelty detection on new observations and the learned decision boundaries for two parameters x_1, x_2. The used algorithm is a one class support vector machine. 213
Figure 3.44 Influence of the parameter changes θ and ρ on the $LSAD$ algorithm. 214
Figure 3.45 Results on using the $LSAD$ novelty detection algorithm to reconstruct the multivariate correlations between all parameters. $N = 260$ samples are shown. 215
Figure 3.46 Results on using the $LSAD$ novelty detection algorithm to reconstruct the multivariate correlations between all parameters with the usage of alg. 2. $N = 534$ samples are shown. 217
Figure 3.47 Further example of the reconstruction algorithm on an industrial example of measured data. Top shows the measured data with $N = 200$ and bottom the reconstructed data with $N = 425$. 217
Figure 3.48 KDE used to resample the two examples of industrial data presented. The reconstructed data has $N = 500$ samples. 219
Figure 4.1 Illustration of the different univariate relationship types between a single input and a single output. 224
Figure 4.2 Illustration of the different multivariate relationship types between two input parameters and a single output. 224
Figure 4.3 Test examples of data points for two variables x, y (the used data set is inspired by Wikipedia contributors [2020]). 227
Figure 4.4 Visualization of the one dimensional correlations between y, x_1 and x_2 of the Himmelblau function. 228
Figure 4.5 Overview of PCC and $SRCC$ correlation values for all 10 parameters for different sample sizes. x-axis shows sample sizes and y-axis the correlation value. The red dashed line marks the 0.1 correlation value. Blue bars are based on the uniform random sampling and orange transparent bars are based on an optimize LHS. Error bars indicates the standard deviation of 30 repetitions for the uniform random sampling. Blue bars are the mean value. 230
Figure 4.6 Overview of MI_{norm} and DCC correlation values for all 10 parameters for different sample sizes. x-axis shows sample sizes and y-axis the correlation value. The red dashed line marks the 0.1 correlation value. Blue bars are based on the uniform random sampling and orange transparent bars are based on an optimize LHS. Error bars indicates the standard deviation of 30 repetitions for the uniform random sampling. Blue bars are the mean value. 231
Figure 4.7 Visualization of the test function to be analyzed via the CoI. . . . 234
Figure 4.8 Visualization univariate dependencies for x_1, x_2, x_3. 234
Figure 4.9 Barplot of the CoI results for the parameters x_1, x_2, x_3, x_2x_3. . . . 235
Figure 4.10 Visualization of the first order dependencies between y and x_2, x_3. 245
Figure 4.11 Example of the minimum / maximum pressure on the rotor structure and the resulting FEM node stresses. 251

Figure 4.12 *DGSM* results for the *FEM* nodes. 252

Table 2.1 Overview of CV results for the Branin-Hoo function (top) and the Himmelblau function (bottom). The results show the difference $\epsilon_{cv} = \hat{R}^2_{val} - \hat{R}^2_{cv}$. Bold marked results show the best results for the specific sample count N. The green values with zero difference have a perfect $\hat{R}^2_{cv}$ value of one. The red values are results with a $\hat{R}^2_{cv}$ of zero. 23

Table 2.2 Overview of CV results for the Eggholder function. The results show the difference $\epsilon_{cv} = \hat{R}^2_{val} - \hat{R}^2_{cv}$. Bold marked results show the best results for the specific sample count N. The green values with zero difference have a perfect $\hat{R}^2_{cv}$ value of one. The red values are results with a $\hat{R}^2_{cv}$ of zero. 24

Table 2.3 Overview of parameters r_j and prognosis measurements $\hat{R}^2_{iso}$, $\hat{R}^2_{aniso}$ for test functions (TF) 2.49 to 2.52. 46

Table 2.4 Overview of length-scales l_i, $i = 1, ..., n$ and prognosis measurements $\hat{R}^2_{iso}$, $\hat{R}^2_{aniso}$ for test functions (TF) 2.49 to 2.52. 69

Table 2.5 Overview of algorithms capabilities. 89

Table 2.6 $RMSE$ for the adaptive sampling example for the global, local and combined test points. 99

Table 2.7 Shifted time series for supervised learning. 108

Table 2.8 Overview of the mean $RMSE$ $(MRMSE)$ values for 100 test signals of test function 2.138 for different multi-output algorithms. The bold values indicate the best result between using PCA or not. . . 116

Table 2.9 Example for a multi-output problem with three input parameters and $n_o = 5$. 118

Table 2.10 Example for a multi-output problem, where one model is built for each output parameter. 118

Table 2.11 Example for a multi-output problem, where the data is transformed to a vector. An additional parameter x_t is introduced to distinguish the signals regarding the the input. 118

Table 2.12 Overview mean $RMSE$ $(MRMSE)$ values for 100 test signals of test function 2.138 for different multi-output algorithms. One model per output parameter was trained. 119

Table 2.13 Overview mean $RMSE$ $(MRMSE)$ values for 100 test signals of test function 2.138 for different multi-output algorithms. Data was transformed to a vector. 120

Table 2.14 Overview $MAPE$ values in [%] for 10 test designs of the test FEM analysis for different multi-output algorithms. 124

Table 2.15 Overview of prediction accuracy in [%] for $1,000$ test designs for different classification algorithms. 126

Table 2.16 Overview of prediction accuracy in [%] of different published works for $10,000$ test images of the MNIST data set. 129

Table 2.17 Overview of prediction accuracy in [%] of different published works for 10,000 test images of the CIFAR10 data set. 130
Table 2.18 Results for the CATS benchmark. 131
Table 2.19 Comparison of the Boston housing example against other Gaussian-process-based methods. $RMSE$ min, mean, max values are given. For the last example the standard deviation is used instead. 134
Table 2.20 Comparison of deep Gaussian process, drop out neural network and deep Gaussian covariance network based on mean and standard deviation $RMSE$ values for the UCI database examples. 135
Table 2.21 Comparison of $\mathcal{DGCN}$, neural networks (NN), random forest (RF), XGboost (XGB), decision tree (DTR) with PCA and without PCA for 21 test cases of the creep strain approximation. The mean $MAPE$ ($MMAPE$) metric in [%] over all 21 multi-output sequences are estimated. Bold values indicate the best result between using PCA and not using PCA. 136
Table 2.22 The $MAPE$ metric in [%] between 7 calibrated FEM curves to the experimental ones. 140
Table 2.23 The $\hat{R}^2$ metric for the 21 rotor optimization relevant outputs. . . . 142
Table 2.24 Validated best objective values for each objective (not in one design). 143
Table 2.25 Validated objective values of one design (one point of the Pareto-frontier). 143
Table 2.26 Mean and standard deviation of the relative error estimated via 25 designs for validation. 144
Table 2.27 Overview of algorithms capabilities. 145
Table 3.1 Overview of published works about LHS optimization and their chosen optimization algorithm / criteria. 180
Table 3.2 Comparison of distance-based LHS objectives and the E^{MADCC} criterion for surrogate model training based on different sample sizes N. The bold numbers indicate the best results considering the mean and the standard deviation. 181
Table 3.3 Comparison of combined LHS objectives for surrogate model training based on different sample sizes N. The bold numbers indicate the best results considering the mean and the standard deviation. . 183
Table 3.4 Comparison of distance-based sequential LHS objectives and the E^{MADCC} criterion for surrogate model training for the modified Branin-Hoo test function with 2 input parameters. The bold numbers indicate the best results considering the mean and the standard deviation. 189
Table 3.5 Comparison of distance-based sequential LHS objectives and the E^{MADCC} criterion for surrogate model training for the modified Branin-Hoo test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation. 189

Table 3.6 Comparison of E^{ME} and the $E^{ME_{cond}}$ criterion for surrogate model training for the Himmelblau test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation. . 193
Table 3.7 Comparison of E^{ME} and the $E^{ME_{cond}}$ criterion for surrogate model training for the Eggholder test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation. . . . 194
Table 3.8 Overview of the constrained multi-objective test functions. 201
Table 3.9 Overview of the constrained multi-objective test functions results of the different approaches and the mean and standard deviation of needed samples over 30 iterations. 202
Table 3.10 Overview of the $RSME$ and $\hat{R}^2$ metric for $1,000$ test points for three different test functions. 205
Table 4.1 Results for the different correlation coefficients on 21 different test sets shown in fig. 4.3 with $N = 1,000$ samples. 227
Table 4.2 Overview of the mean $\pm$ standard deviation of the $p-$values for all correlation coefficients and for all sample sizes, estimated over 30 repetitions. The null hypothesis is that there is no correlation. Bold values indicate significant correlation estimates with $p-$values < 0.05 based on the mean value. 229
Table 4.3 Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.28. For this example the analytical solution is also given for S_i^C. . 246
Table 4.4 Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.30. 247
Table 4.5 Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.32. 247
Table 4.6 Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Mara et al. [2015] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.28. 248
Table 4.7 Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.32. This time a $\mathcal{DGCN}$ model is used to evaluate the design matrices. 248

NOMENCLATURE

NOTATION

The following notation is used throughout the work:

- Upper case letters X denote matrices if not stated otherwise, some coefficients and scalars are also denoted by an upper case letter.
- Bold lower case letters $\boldsymbol{x}$ denote vectors.
- Standard weight letters x denote scalar quantities.
- Subscripts can denote either entire rows / columns with bold letters ($\boldsymbol{x}_i$) or specific elements (x_{ij}).
- A subscript asterisk, such as in X_*, indicates reference to a test set quantity.
- A superscript like $[0, 1]$ in combination with a quantity symbol like $\mathbb{R}$ denotes the limits of the variable(s).
- A scalar or multiplication of scalars like 2×2 superscript in combination with a quantity symbol like $\mathbb{R}$ denotes the dimension of the vector or matrix.
- Vectors are assumed to be column vectors unless noted otherwise.

The following listed symbols are used multiple times throughout this work and therefore listed here to give the reader a reference. Symbols used to temporary describe an algorithm or only listed once are not listed here, since every notation is also explained in the corresponding paragraph.

ROMAN SYMBOLS

Matrices:

B	Matrix of bias terms
C	Correlation matrix
D	Distance matrix
I	Identity matrix, Rank correlation matrix for the Iman and Conover rank correlation method
K	$K(X, X)$, the $N \times N$ covariance (or Gram) matrix between training points
K_*	$K(X, X_*)$, the $N \times N_*$ covariance matrix between training and test points
L	Lower triangular matrix of the Cholesky decomposition
P	Vandermonde matrix (RSM coefficients matrix), Lower triangular matrix of the target rank correlation matrix C

Q	Lower triangular matrix of the rank correlation matrix I
S	Transformation matrix, which maps X to X^* for the Iman and Conover rank correlation method
W	Weighting matrix
X, X_*	Training / test input matrix (N rows, n columns)
$X_{\sim i}$	Training input matrix without the i-th parameter (column)
Y	Training output matrix (N rows, q columns)
$\hat{Y}$	Approximated output matrix
Y_P	Matrix of the current Pareto-frontiers

Vectors:

$\mathbf{0}$	Vector of all 0's
$\mathbf{1}$	Vector of all 1's
$\boldsymbol{\theta}$	Vector of free parameters
$\boldsymbol{\mu}$	Mean vector
$\boldsymbol{b}$	Vector of bias terms
$\boldsymbol{f}$	Vector of function values
$\boldsymbol{k}$	Vector of covariance values between X and $\boldsymbol{x}$ or X_* and $\boldsymbol{x}_*$
$\boldsymbol{k}_*$	Vector of covariance values between X and $\boldsymbol{x}_*$
$\boldsymbol{o}^{[l]}$	Output vector of layer l of a neural network
$\boldsymbol{p}$	Vector of free parameters of a neural network
$\boldsymbol{p}_*$	Vector of polynomial regressors for one point
$\boldsymbol{w}$	Vector of weights
$\boldsymbol{x}$, $\boldsymbol{x}_*$	A single (training / test) point vector (n columns) or vector of random variables
$\boldsymbol{x}_i$	i-th training data point
$\boldsymbol{y}$, $\boldsymbol{y}_*$	Training / test output vector
$\boldsymbol{y}^T$	The transpose of vector $\boldsymbol{y}$
$\boldsymbol{y}_0^{ref}$	Reference point for expected-improvement-based optimization
$\boldsymbol{y}_i$	i-th training output data point
$\hat{\boldsymbol{y}}_i$	Model output vector on $\boldsymbol{x}_i$
$\hat{\boldsymbol{y}}$	Model output vector on X
$\boldsymbol{z}^{[l]}$	Weighted input vector for all neurons at layer l

Scalars:

b	Bias term
c_{ij}	Pearson correlation coefficient between input parameter i and j
c_{ij}^T	Target correlation coefficient between input parameter i and j
c_{ij}^X	Current correlation coefficient between input parameter i and j of design matrix X
CoI	Coefficient of importance
d_i	i-th distance
d_{ij}	Inter-sited distances between point i and j
d_{max}	Maximum pairwise distance
dc_{ij}	Distance correlation coefficient between input parameter i and j
e	Euler constant

E^{Φ_p}	Morris maximin distance criterion
E^{AE}	Audze-Eglais criterion
E^{CD}	Correlation-difference-based criterion
E^{CL_2}	Centered discrepancy criterion
E^{CN}	Condition-number-based criterion
E^{DCC}	Distance-correlation-based criterion
E^{EMM}	Maximin distance criterion
E^{MADCC}	Maximum-absolute-distance-correlation-based criterion
E^{MAPCC}	Maximal-absolute-Pearson-correlation-based criterion
E^{ME}	Maximum entropy criterion
$E^{ME_{cond}}$	Maximum entropy criterion with conditioned covariance matrix
E^{PCC}	Pearson-correlation-based criterion
E^{s}	Standard-error-based optimization criterion
f_0^2	Mean value for the Sobol indice calculation
h	Bandwidth parameter for the kernel density estimation algorithm
J_i	i-th index of inter-sited distances
k	Number of folds in cross-validation
l	Correlation length-scale, Number of levels in a systematic design plan
m	Polynomial order, Number of inducing points
N and N_*	Number of training and test data points / samples
N_b	Batch size of samples
n	Number of input parameters
n_{bin}	Number of bins / classes for a histogram
n_c	Number of covariance functions
n_{cl}	Number of classes in a classification problem
n_{con}	Number of constraints
n_{hl}	Number of hidden layers in a neural network
n_l	Number of layers in a neural network
$n_n^{[l]}$	Number of neurons per layer l
n_o	Number of output parameters
n_{obj}	Number of objectives
n_t	Number of previous time steps used to train a $\mathcal{DGCN}$ model
$o_j^{[l]}$	Denote the output from neuron j at layer l of a neural network
p	Pressure
$p-$value	Used to estimate the significance of correlation coefficients (Probability of no correlation)
q	Number of output parameters
R^2	Coefficient of determination
$\hat{R}^2$	Predictive coefficient of determination
R^2_{adj}	Adjusted coefficient of determination
r	Influence radius of the MLS method
S_i^C	i-th first order correlated Sobol indice
S_{Ti}^C	i-th total order correlated Sobol indice
S_i^U	i-th first order uncorrelated Sobol indice

S_{Ti}^{U}	i-th total order uncorrelated Sobol indice
S_i	i-th first order Sobol indice
S_{Ti}	i-th total order Sobol indice
s	Number of pairwise distances
$\hat{s}_*$	Standard error of a $\mathcal{DGCN}$-based prediction
t	Time step
v	Parameter for smoothness control of the Matérn class of covariance functions
w	Scalar weight
x	Scalar variable
x_i^l	Lower parameter limit of parameter x_i
x_i^u	Upper parameter limit of parameter x_i
y	Scalar output
$\hat{y}$	Scalar approximated output
$\bar{y}$	Mean value of the vector $\boldsymbol{y}$
$z_j^{[l]}$	Weighted input for neuron j at layer l

Quantities:

$\mathbb{N}$	Natural numbers
$\mathbb{R}$	Real numbers
$\mathbb{Z}$	Integer numbers

Distributions:

$\mathcal{E}(\lambda)$	Exponential distribution
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$\mathcal{U}(a,\, b)$	Uniform distribution with lower limit a and upper limit b

Norms:

$\lvert K \rvert$	Determinant of K matrix
$\lvert\lvert$	Absolute norm if not used with a matrix
$\lVert \boldsymbol{y} \rVert$	Euclidean norm of vector $\boldsymbol{y}$, i.e. $(\sum_i y_i^2)^{1/2}$
$\lVert \boldsymbol{x}, \boldsymbol{x}_* \rVert$	Euclidean distance between two points $\boldsymbol{x}, \boldsymbol{x}_*$

Mathematical symbols / operations:

$\cdot$	Scalar product between two vectors
$\circ$	Component-wise product of two vectors
$\sim$	Distributed according to; example: $x \sim \mathcal{N}(\mu, \sigma^2)$
$\approx$	Approximately
$\cup$	Union of two sets
∂, ∇ or ∇_x	Partial derivatives (w.r.t x)
$=$	Equal to
$\triangleq$	Equal by definition to
$\frac{d}{dx}$	Derivative w.r.t. x
$\backslash$	Left matrix divide: $A\backslash \boldsymbol{b}$ is the vector $\boldsymbol{x}$ which solves $A\boldsymbol{x} = \boldsymbol{b}$
$\times$	Indicator for the dimensions of a matrix (e.g. $N \times N$) or the amount of neurons

$\|$	Conditioned on
∇^2	The Hessian matrix of second derivatives
$\propto$	Proportional to
$\in$	Member of
$\notin$	Not member of
$\mathbb{E}$	Expectation
$\mathcal{O}$	Complexity
$\mathbb{V}$	Variance
∞	Infinity

Functions:

$a()$	Activation function of a neural network
$\Gamma()$	Euclidean Gamma function
$\lambda()$	Logistic function
$\mu(\boldsymbol{x})$	The mean function of a Gaussian process
$\Psi()$	Objective function
$\phi()$	Mapping function
$chol(X)$	Cholesky decomposition: L is a lower triangular matrix such that $LL^T = X$
$cond()$	condition number of
$cos()$	Cosine function
$cov(\boldsymbol{x}_i, \boldsymbol{x}_j)$	Covariance (or kernel) function evaluated at $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
$dCov()$	Distance covariance for the distance correlation coefficient
$diag(\boldsymbol{w})$	(Vector argument) a diagonal matrix containing the elements of $\boldsymbol{w}$
$diag(W)$	(Matrix argument) a diagonal vector containing the diagonal elements of W
$dVar()$	Distance variance for the distance correlation coefficient
$E_i(\boldsymbol{x}_*)$	The limit version of a finite differencing scheme where the finite difference for x_i tends to zero
$EI(\boldsymbol{x}_*)$	Expected improvement for a test point $\boldsymbol{x}_*$
$exp(x)$	Exponential function of x
$F(x)$	Cumulative density function
$F^{-1}(x)$	Quantile function
$f(x)$, $f'(x)$	Function of x and its first derivative
$f_{pdf}(x)$	Probability density function of x
$\hat{f}_{pdf}(x)$	Approximation of the probability density function of x
$\hat{f}()$	Surrogate model / approximation of function $f()$
$I(\boldsymbol{x}_*)$	Improvement of a objective in the context of expected improvement
$k(\boldsymbol{x}, \boldsymbol{x})$	Covariance (or kernel) function evaluated at $\boldsymbol{x}$ and $\boldsymbol{x}$
$k_*(\boldsymbol{x}, \boldsymbol{x}_*)$	Covariance (or kernel) function evaluated at $\boldsymbol{x}$ and $\boldsymbol{x}_*$
$K_v()$	Modified Bessel function
$L()$	Likelihood function, Loss function of a neural network
$ln()$	Natural logarithm with base exp

$max()$, $min()$	Function to estimate the maximum / minimum
$p(x)$	(Marginal) probability distribution of x
$p(x, y)$	Joint probability distribution of x, y
$pol()$	Polynomial function
$y\|x$ and $p(y\|x)$	Conditional random variable y given x and its probability distribution
$PoI(\boldsymbol{x}_*)$	Probability of improvement for a test point $\boldsymbol{x}_*$
$q()$	Approximation of a probability p()
$RankCorr()$	Rank correlation
$rk(x_i)$	Rank of the input parameter value x_i
$sin()$	Sine function
$t(\frac{\alpha}{2}\|N-(mn+1))$	Student-t-distribution percentile
$tan()$	Tangent function
$tanh()$	Hyperbolic tangent function
$tr(A)$	Trace of (square) matrix A
$z\left(\frac{\alpha}{2}\right)$	z-score / critical value of the standard Gaussian distribution

GREEK SYMBOLS

α	Confidence interval percentile, Constant in the MLS approach, Parameter of the rational quadratic covariance function
Γ	Gamma function, Ratio (in percents) of the volume of the feasible region to that of the search space
γ	Parameter of the γ-exponential covariance function
Δ	Gradient
δ	Error, Kronecker delta
$\delta_j^{[l]}$	Error in the j-th neuron at layer l
ϵ	Error / residual
$\hat{\epsilon}$	Model residuals
η	Learning rate of a neural network
θ	Free parameters in general, Kernel length-scale parameter
λ	Regularization coefficient, Box-Cox power transformation coefficient, Free parameter of the exponential distribution, Eigenvalue
μ	Mean value
ν_i	The $DGSM$-based sensitivity measure for parameter x_i
π	Mathematical constant ≈ 3.14159
ρ	Regularization parameter of the $LSAD$ novelty detection algorithm
σ	Standard deviation
σ^2	Variance

σ_f^2	Signal variance hyperparameter of a $\mathcal{GP}$
σ_n^2	Noise variance hyperparameter of a $\mathcal{GP}$
$\hat{\sigma}$	Standard deviation of the prediction of a model $\hat{f}$
$\hat{\sigma}^2$	Variance of the prediction of a model $\hat{f}$
τ	Euclidean distance between two points $\boldsymbol{x}_i - \boldsymbol{x}_j$, Kendall's τ correlation coefficient
Φ	Cumulative density function of the Gaussian distribution
ϕ	Probability density function of the Gaussian distribution
Ψ	Objective
Ω^n	The n-dimensional unit input parameter cube $[0,1]^n$

SUPERSCRIPTS

$'$	First derivative
$\geq x$	All numbers greater than x
$\bar{}$	Mean of
$\tilde{}$	Possible candidate for the polynomial coefficients, Augmented input for the neural network covariance function
$[x, y]$	Lower and upper limit x, y In conjunction with a quantity symbol, it can also indicate the possible values allowed of this quantity
Φ_p	Morris maximin distance criterion
AE	Audze-Eglais criterion
C	Indicator for the correlated Sobol indice
CD	Correlation-difference-based criterion
CL_2	Centered discrepancy criterion
CN	Condition-number-based criterion
cv	Cross-validation estimated metric
DCC	Distance-correlation-based criterion
EMM	Maximin distance criterion
(j)	j-th sample
l	Indicates the lower parameter limit
$[l]$	Layer number of a neural network
(N)	Indicates the N-th sample for the Sobol indice estimation
$MADCC$	Maximum-absolute-distance-correlation-based criterion
$MAPCC$	Maximal-absolute-Pearson-correlation-based criterion
ME	Maximum entropy criterion
ME_{cond}	Maximum entropy criterion with conditioned covariance matrix
p	Exponent for the Morris maximin distance criterion
PCC	Pearson-correlation-based criterion
ref	Reference point
s	Number of free parameters of a neural network, Standard-error-based optimization criterion
T	Transpose of a vector / matrix
Tar	Target, used to indicate the target correlation matrix

U	Indicator for the uncorrelated Sobol indice
u	Indicates the upper parameter limit
X	Current design matrix, used to indicate the current design correlation matrix

SUBSCRIPTS

$\sim i$	Without parameter i
λ	Indicates a Box-Cox-power-transformation-based metric
$*$	Indicates a test set / point
adj	Adjusted to the number of samples
$aniso$	Anisotropic
b	Batch
$best$	Current best version of a scalar / vector / matrix
bin	Bins of a histogram
c	Covariance
$cand$	Current candidate version of a scalar / vector / matrix
CI	Confidence interval
cl	Classes
con	Constraint
cv	Estimated metric on cross-validation
DNN	Indicator for a neural-network-based result
$DNN_{\sigma_n^2}$	Indicator for the neural network used to learn the noise variance parameters of the $\mathcal{DGCN}$ algorithm
DNN_l	Indicator for the neural network used to learn the length-scale parameters of the $\mathcal{DGCN}$ algorithm
hyp	Hypervolume indicator
i, k	i-th / k-th row of a vector (e.g. x_i) or row / column of a matrix (e.g. $\boldsymbol{x}_i$)
ij	i-th row, j-th column of a matrix (e.g. x_{ij})
iso	Isotropic
L	Number of network layers
Mat	Matérn kernel
max/min	Maximum / minimum of a scalar variable
MLE	Solution of the maximum likelihood estimation
n	Neurons
$neighbor$	Indicates the set of training points in the neighborhood of the point to predict
$noisy$	Indicator for a noisy quantity
$norm$	Normalized
o	Output
obj	Objective
OLS	Solution of the ordinary least square estimation
P	Parteo-frontier indicator
Per	Periodic kernel
RQ	Rational quadratic kernel

s	Indicates the summed covariance matrix of a $\mathcal{DGCN}$ model
SE	Squared exponential kernel
std	Indicates a standardized metric
t	Time step or time dependent result
$t-i$	Indicator for the i-th previous time step
up	Updated model without retrain
v	Matérn kernel parameter
val	Validated metric

ACRONYMS

Italic acronyms denote a mathematical function, coefficient, method, model or are somehow else mathematical related. Others are acronyms for preambles or simply not mathematical.

1D/3D	1, 3 dimensional
AB	AdaBoost
a.e.	Almost everywhere
alg.	Algorithm
$AMLS$	Anisotropic moving least squares
ARD	Automatic relevance determination
C	Correlation
CD	Correlation difference
CDF	Cumulative density function
CFD	Computational fluid dynamics
CI	Confidence interval
CNN	Convolutional neural network
CoI	Coefficient of importance
CPU	Central processing unit
CV	Cross-validation
DCC	Distance correlation coefficient
$\mathcal{DGCN}$	Deep Gaussian covariance network
$\mathcal{DGP}$	Deep Gaussian process
$DGSM$	Derivative-based global sensitivity measures
DNN	Deep neural network
DoE	Design of Experiment
DTR	Decision trees
e.g.	Exempli gratia ("for the sake of an example")
EGO	Efficient global optimization
EI	Expected improvement
eq.	Equation
FEM	Finite element analysis
fig.	Figure
GB	Gigabyte
$\mathcal{GP}$	Gaussian process: $f \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x_i}, \boldsymbol{x_j}))$, the function f is distributed as a Gaussian process with mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x_i}, \boldsymbol{x_j})$

GPU	Graphics processing unit
HV	Dominated hypervolume
$\overline{HV}$	Non-dominated hypervolume
i.e.	Id est ("it is")
KDE	Kernel density estimation
LHS	Latin hypercube sampling
log	Logarithm
LOO	Leave-one-out
$LSAD$	Least squares anomaly detection
$LSTM$	Long short-term memory units
MAP	Maximum a posteriori
$MAPCC$	Mean absolute percentage error
$MAPE$	Mean absolute percentage error
$MaxAE$	Maximum absolute error
MC	Monte Carlo
$MCMC$	Marcov chain Monte Carlo
MCS	Monte Carlo sampling
MI	Mutual information
MLE	Maximum likelihood estimation
MLS	Moving least squares
MNIST	Modified National Institute of Standards and Technology database
$MMAPE$	Mean mean absolute percentage error for multi-output problems
$MRMSE$	Mean root mean squared error for multi-output problems
MSE	Mean squared error
MW	Megawatt
NN	Neural network
OLS	Ordinary least square
Pa	Pascal - SI derived unit of pressure
PCA	Principle component analysis
PCC	Pearson correlation coefficient
PDF	Probability density function
PoI	Probability of improvement
PPF	Percent point function
ppm	Parts per million
RAM	Random-Access Memory
RDO	Robust design optimization
$ReLU$	Rectified linear unit
RF	Random forest
$RMSE$	Root mean squared error
RNN	Recurrent neural network
RSM	Response surface model
$RSSE$	Regularized sum of squared error
SA	Sensitivity analysis
$SLHS$	Sequential Latin hypercube sampling
$SRCC$	Spearman's rank correlation
SSE	Sum of squared error

s.t.	Such that
SVM	Support vector machines
tab.	Table
$TanH$	hyperbolic tangent
TF	Test function
TPU	Tensor processing unit
V	Volume
w.r.t	With respect to
XGB	XGBoost

1 INTRODUCTION

1.1 MOTIVATION

The field of modern artificial intelligence has made significant advances in the past decades. The objective of this field is to seek solutions that allow machines to "understand" and "learn", so that a generalization beyond the observations is possible and allows the machine to "reason" about novel data. The use of machine learning is not only of interest in computer science but has also recently become very popular in fields such as health-care, mechanical engineering, finance, aerospace, chemistry and many more. It will be one of the key technologies in the coming years to improve many different aspects in those areas. It will also help to meaningfully process the enormous amounts of data created by the digital transformation and industry 4.0. A big challenge in using such algorithms is to know how certain their predictions are. For example, in autonomous driving, a crucial question would be how reliable is the algorithm in order to let the car cross an intersection without endangering the lives of the passengers? In machine learning one of the promising solutions to solve these challenging tasks are so-called probabilistic models. In a probabilistic framework, data understanding is achieved by separating the true information from noise and by removing redundancies in complex and noisy data (Damianou [2015]). It also incorporates the possible uncertainty of a prediction. The concepts of uncertainty and noise are fundamental in probabilistic modeling. So instead of only having the information that the model believes the car can cross the intersection without a crash, it also gives the information about how certain it is that there will be no crash. This makes a huge difference in the acceptance of this decision.

This thesis focuses on probabilistic approaches that solve the data understanding and generalization challenges so that the uncertainty estimation of those models is always a part of the results. The unknowns included in the probabilistic models of interest are treated in a Bayesian manner, according to which priors are employed to weight all possible outcomes. The resulting posterior distributions encode the uncertainty in the resulting estimates. In order to provide such models, aspects of how to generate data for those models in the most efficient way will also be part of this thesis. In many applications it is very expensive to generate data, for example in the area of mechanical engineering or chemistry. Physical experiments or complex numerical simulations take considerable time, computer hardware and / or material resources. For example, machines for automatic chemical formulations are very expensive and therefore usage time should be kept as short as possible in order to be able to deal with more projects. Additionally, the more experiments are needed, the more costs are incurred for waste disposal in addition to the already existing material costs. A low number of experiments (also generally referred to as designs or samples) in such a case reduces both the time on the expensive machines and the material costs and helps the environment because less harmful waste is produced. Therefore, methods which can reduce the number of designs needed in order to train a probabilistic machine learning model, are very necessary and beneficial. The use of probabilistic models in the data generation phase

will be advantageous over deterministic models because knowing about the uncertainty of the model can help in deciding where to place the next experiment in order to reduce it most efficiently. This will quickly lead to a globally accurate surrogate model. On the other hand, it also gives information about when to stop the generation of further samples. This strategy can also be used in combination with the intention to optimize one or more objectives very efficiently and is known as Bayesian optimization which should also be part of this thesis.

Another important aspect of data analysis is to get a deeper understanding of the importance of the input parameters explaining the observations. The answer to the question: "Which are the main drivers for my output of interest?". Regarding the example of finding a new chemical formulation, the answer to this question may reveal that a certain ingredient which might also be very expensive, has no influence on the performance and can be left out in future productions. Another aspect would be the investigation of production-relevant parameters whose manufacturing tolerances have a large influence on the robustness and reliability of a product. If these tolerances are too large, it could be that certain properties are no longer met in the use of the product or the rejection rate rises considerably. On the other hand, very small manufacturing tolerances can be very costly to produce. An analysis of these parameters could reveal that these small tolerances are not necessary to maintain a certain minimum failure probability of the product. An example in the turbomachinery area would be to find out which parameters have the highest influence on the lifetime of a turbine. This would give information on how to increase the reliability of the product and possibly which parameters must be controlled more closely, and also which parameters must be carefully checked during service maintenance. Unfortunately, this type of analysis usually takes a lot of samples and therefore it might be efficient to perform these on a machine learning model. The advantage of using a probabilistic model is also given in this case, since it gives information about how certain the resulting importance measures are. Additionally, the connection of machine learning and sensitivity analysis is beneficial, because those methods can be used as pre-variable selection in order to decrease the number of parameters the model has to handle. The curse of dimensionality (Keogh and Mueen [2011]) makes it difficult to handle an arbitrary number of input parameters with a limited supply of samples.

In most literature related to machine learning, the link between these different fields and methods it not investigated. Therefore, one main motivation of this work is to show that these topics in combination reveal a much higher potential of machine learning and its efficient practical application. Besides this, the efficiency and scalability of these algorithms is an important aspect in this work so that the application to real world problems is feasible. Furthermore, scientists and engineers from many different areas want to use these algorithms but usually do not have the time to become an expert. Typically, in addition to the proper choice of algorithm (not every algorithm is suitable for both regression and classification, for example), the user must also know the peculiarities of the algorithm and its so-called hyperparameters[1] which can decide on good or bad prediction accuracy. In order to set these hyperparameters correctly, the user usually has to have an understanding of the mathematical background of the algorithms. If the user wants to solve a multitude

1 Hyperparameters are parameters which are typically not learned during the training phase and must be set based on the user's experience or expensive optimization. In this work, therefore, the differentiation is made between hyperparameters and free model parameters which are learned during training.

of different tasks with the help of machine learning algorithms, it requires a sound and extensive knowledge in a variety of methods. Therefore, another focus of this thesis is the simple use of the algorithms without losing performance. This is achieved, for example, by decreasing the number of hyperparameters as far as possible and making them applicable to wide range of different tasks. Making machine learning algorithms easier to use, helps to spread this topic even further and to solve important challenges of the future in non-computer science related fields like healthcare, chemistry or mechanical engineering.

1.2 OUTLINE OF THE THESIS

1.2.1 *Objectives*

The goal of this work is the development of a new, efficient and easy to use probabilistic machine learning algorithm which can solve several different learning tasks - namely regression, classification, image-classification, sequential dependent and multi-output problems within one single algorithm. Furthermore, it should be capable of handling arbitrary data set sizes and so be applicable for big data problems which might become standard in the near future. As mentioned before, a further requirement should be a high automation of the algorithm without the need for the user to set several different hyperparameters. Many algorithms have various hyperparameters, the optimal setting of which requires either some experience and a mathematical understanding of how the algorithms work, or time through testing. An algorithm that has no hyperparameters and which can be used for a variety of applications saves the user a lot of time. Additionally, the capabilities and efficiency of this model should further be enhanced by using its properties in the other connected fields of data generation and sensitivity analysis mentioned, to obtain a fully-automatic, efficient and complete data analysis experience. This should be demonstrated on a variety of academic and industrial examples. To establish the efficiency of the proposed methods, it will be benchmarked with other established methods taken from literature. Besides developing new or extending existing algorithms, the reader will get a comprehensive overview of state of the art methods to solve common machine learning problems. Furthermore, recommendations are given in numerous sections about when and why certain algorithms are best suited to solve a particular problem. These recommendations are always given with the focus on efficiency.

1.2.2 *Organization of the Text*

To achieve these goals the text is organized as follows:

- Chapter 2 gives an overview of several aspects of supervised machine learning. It starts with the general topic of model selection which explains how a machine learning model can be trained or selected in general. This topic is dealt with first because it explains some general ideas used for many algorithms presented in this work. Afterwards, several machine learning algorithms will be presented with increasing complexity up to the newly developed method called deep Gaussian covariance network ($\mathcal{DGCN}$), including another new further development of the moving least squares method (section 2.7), so that it can automatically reduce the influence of unimpor-

tant parameters. Each of the algorithms presented will introduce some properties or ideas which will be reused in the $\mathcal{DGCN}$ model. Section 2.11 will give an overview of methods to validate a trained model and finally in section 2.12 a conclusion of this chapter is given. There are three additional sections given namely: 2.3, 2.4, 2.6 which should give some insight in the advanced machine learning aspects: heteroscedasticity in regression, curse of dimensionality and automatic relevance determination which play a important role for some of the presented algorithms.

- Chapter 3 presents several methods for data generation which is also known as design of experiments. A comparison is given for their effectiveness in combination with machine learning model training, especially with the new $\mathcal{DGCN}$ model. Section 3.4 will show some advance techniques for adaptive sampling and also includes Bayesian optimization. It will be shown that the non-stationarity [2] of the $\mathcal{DGCN}$ model allows a much higher synergy [3] with the adaptive sampling methods compared to standard Gaussian processes. Section 3.5 deals with the problem of extrapolation, if for example optimization or sensitivity analysis is performed on surrogate models. It shows a new rejection-sampling-based algorithm to resample from measured data with the same multivariate distribution and correlation. This algorithm is based on a specific type of unsupervised machine learning called novelty detection.

- Chapter 4 will introduce methods for sensitivity analysis. It gives an overview of different methods, starting with correlation coefficients over surrogate-model-based estimates to global variance and derivative-based sensitivity measures. Since most of the methods presented need a lot of design evaluations, they are predestined to be evaluated on surrogate models. This chapter will also provide an insight of which method should be used to detect specific types of dependencies. In section 4.5, derivative-based sensitivity measures are presented. Here, the sensitivity analysis for multi-output problems will be illustrated, for example to visualize sensitivities on field models, e.g. used in finite element or finite volume discretization methods.

- Chapter 5 summarizes the key contributions of the thesis and discusses ideas for future work.

Each section is structured in such a way that, first of all, the mathematical foundations are explained, then possible newly developed extensions are presented and finally the algorithms are tested on examples.

1.2.3 *Associated Publications and Software*

Several of the presented algorithms of this thesis have also been applied for real world applications which have been published in papers and are listed in the following with their link to this thesis. These publications focus on the application of the algorithms and not on the mathematical background which are presented for the first time (except for the

2 Non-stationarity will be explained more in detail in section 2.8. In this context, it means that the algorithm is able to adjust its behavior in the design space. In the case of $\mathcal{DGCN}$, this is achieved by the fact that the free parameters can vary depending on where in the design space the model should make a prediction.

3 In this context it means a lower number of needed samples to obtain the global optimum or a global accurate model.

moving least squares extension) in full detail in this thesis. To protect the idea of $\mathcal{DGCN}$, a pre-print has been uploaded to arXiv (Cremanns and Roos [2017]). It does not contain as many details and extensive examples as presented in this work. Furthermore, section 5.1 give an overview of the new contributions this thesis provides and have not been published yet.

- Cremanns et al. [2016] presents the efficient multi-objective optimization of labyrinth seal leakage of a steam turbine based on hybrid surrogate models. In this work, an ensemble of Gaussian processes and an enhanced moving least squares models have been used. It shows also the usage of an optimized design plan as well as the usage of the global variance-based sensitivity analysis to detect the most important parameters. The ideas applied in this paper are presented in chapter 2: section 2.7, 2.8 as well as the methods presented in chapter 3: section 3.3 and chapter 4: section 4.4. As mentioned, does this paper only show the application of the algorithms and not the mathematical background.

- Cremanns and Roos [2016] demonstrate a self-developed extension to the moving least squares algorithm in order to automatically learn the importance of input parameters and enables the algorithm to reduce or even remove the influence of unimportant parameters during the training phase. This extends the predictive power of the algorithm greatly compared to the standard version. This work is mainly presented in chapter 2: section 2.7 in combination with one of the model selection methods of chapter 2: section 2.2.

- Lueckemeyer et al. [2017] is a co-authored work which gives an overview of different classification algorithms to investigate the critical bearing and pedestal properties of a rotor train alignment under the assumption that these properties are not available or only available with a limited degree of confidence. The data of this example is reused in this thesis in chapter 2: section 2.10 to show the capabilities of $\mathcal{DGCN}$ for classification tasks. In this paper $\mathcal{DGCN}$ was not used. This paper also includes the topic of probabilistic design assessments on surrogate models which is not covered in this thesis.

- Musch et al. [2017] is also a co-authored work. It presents the combined optimization of the last-stage and diffuser of a steam turbine based on surrogate models. It utilizes the sequential LHS method which is explained in chapter 3: section 3.4. As a surrogate model again, an ensemble of a Gaussian process and the enhanced moving least squares algorithm was used. The sequential LHS method was not explained in this paper.

- Cremanns and Roos [2017] is the pre-print version of the $\mathcal{DGCN}$ algorithm, therefore it includes parts of chapter 2: section 2.10. As mentioned does this pre-print give only hints at the functionality of DGCN to protect the core novelty of this thesis. The extensions, like the application for image recognition or the combination with Bayesian Optimization, are not mentioned there and are published for the first time in this thesis. Also the structure of the used neural networks or the used regularization are for example not mentioned in the pre-print.

- Cremanns et al. [2018a] showed once again the efficient multi-objective optimization of labyrinth seal leakage of a steam turbine based on surrogate models. This time the focus is to perform a robust design optimization which includes the influence of possible scattering parameters into the optimization. The objectives were the mean and standard deviation of the original target values instead of the deterministic ones. An efficient, multi-objective optimization algorithm for this purpose is also presented. As mentioned, the topic of robust design optimization is left out of this thesis, since it does not fit with the thesis focus. The optimization is based on a $\mathcal{DGCN}$ model and it will be explained how statistics are done on surrogate models and how to perform the design of experiment for stochastic and non-stochastic parameters in one combined design space. This paper also presents the application of variance-based sensitivity analysis of chapter 4: section 4.4 in order to find the most important parameters for the robustness of the design. This paper provides more information on the topic of robust design optimization based on surrogate models.

- Cremanns et al. [2018b] gives a further example of an efficient multi-objective optimization based on a $\mathcal{DGCN}$ model including sensitivity analysis and adaptive sampling (the sequential LHS sampling of section 3.4). In this work, a steam turbine exhaust is optimized. Again no detailed mathematical background of the used methods is given in this paper.

All algorithms created for this thesis were written in Python and are compatible with the versions 2.7 / 3.7. If third party software is used for some specific tasks, like for example optimization algorithms, it will be referenced in the corresponding section. Every non-industrial example in this work can be reproduced with open source software and the correspondingly developed algorithms. Industrial examples may require licensed simulation software to generate the design samples.

2

SUPERVISED MACHINE LEARNING

2.1 INTRODUCTION

Over the past two decades, machine learning (Bishop [2006], Smola and Vishwanathan [2008]) has become one of the pillars of computational intelligence technology and with that, a rather central, albeit usually hidden, part of everyone's life (Marasigan [2016]). Machine learning is the general term for algorithms and methods which allow computers (machines) to automatically discover patterns in data and use that pattern to perform predictions e.g. for optimization and stochastic analysis or other types of analysis (J. Forrester et al. [2006]). Worthy of mention is the fact that the algorithm does not only learn the relationships of the data but is also able to generalize the information (Murphy [2012], Alpaydin [2014]). Therefore, it is a unique property of machine learning that these relationships do not have to be explicitly programmed (Koller et al. [2007]). Instead of writing a program, the data is fed into a generic algorithm which builds a logic based on the given data, as shown in fig. 2.1 (Barpaga [2019]). After a machine learning algorithm is trained on the data and an approximation model is obtained, it is often called a surrogate model or a digital twin (Gorissen et al. [2010]).

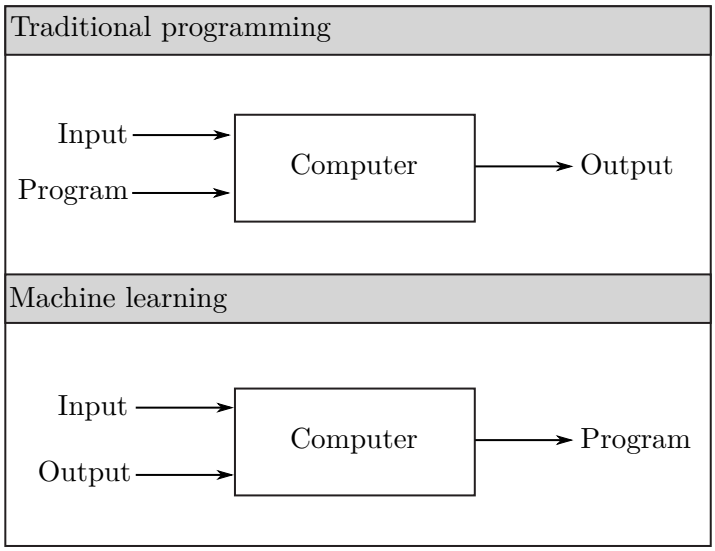


Figure 2.1: Traditional programming vs. machine learning. Source: Barpaga [2019].

Machine learning is a tool which gives users the opportunity to handle the ever-increasing amount of data produced in the digital era. It enables products to become smart and interact with their environments like, for example, autonomous driving. Particularly, the large amount of data which will be produced in the coming years will be impossible for the human brain to process and analyze it in an acceptable amount of time. Therefore, machine learning has become a part of many different industries as shown in fig. 2.2 (Google [2016a]). The benefits of machine learning are versatile. According to a survey preformed by Google

in 2016 (Google [2016b]), they asked 68 companies what they expect to gain from the use of machine learning. The result is shown in fig. 2.3. Efficiency, cost reduction and deeper insights in the data were some of the important aspects cited.

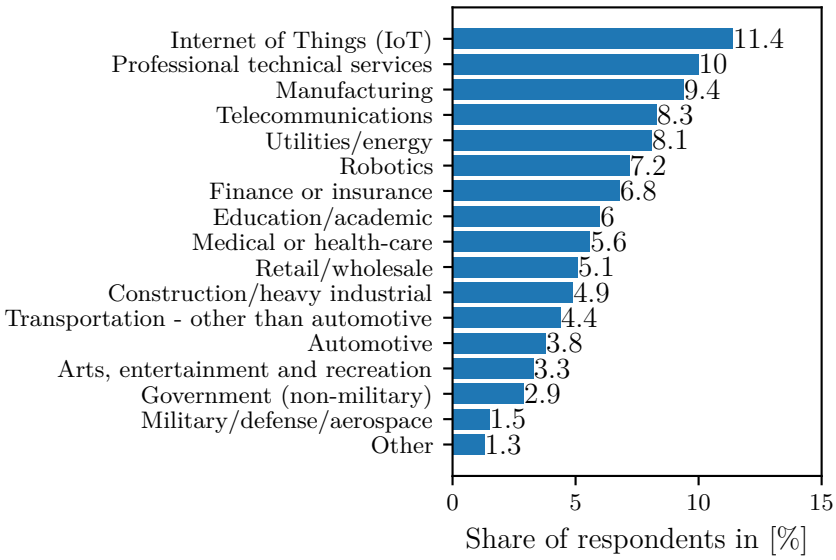


Figure 2.2: Industries targeted by machine learning application developers. Source: Google [2016a], 2016, $N = 1,441$ respondents.

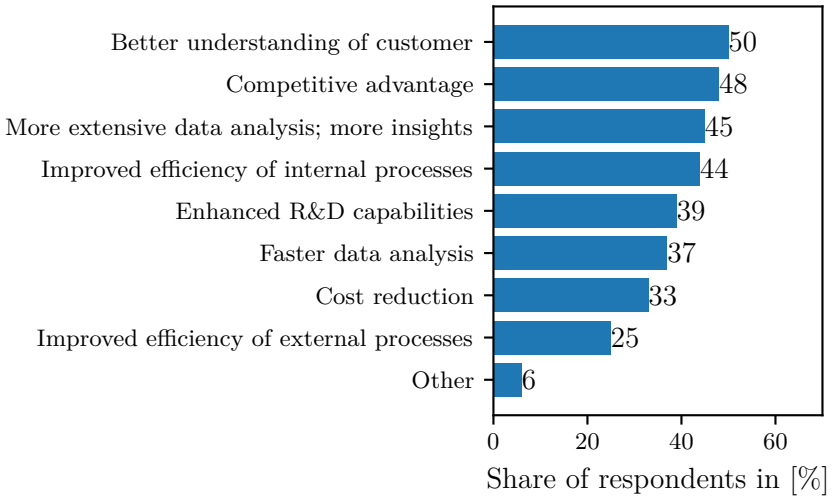


Figure 2.3: Benefits of machine learning. Source: Google [2016b], 2016, $N \approx 68$ companies were asked what they expect to gain from the use of machine learning.

Since machine learning is just the generic term for several different algorithms and learning types, it can be split up into further subcategories. There are three main groups of learning types:

- Supervised learning (e.g. Kotsiantis et al. [2007]):

  - In the case of supervised learning, input and output data exists and the machine learning algorithm is supposed to learn the relationship between them. A function $y(\boldsymbol{x}) = x_1 + x_2$ would be an example, where the model should learn the relationship between the output y and its input parameters x_1, x_2. There are further subgroups of supervised learning, such as:
    * semi-supervised learning (e.g. Chapelle et al. [2009]): the output is not available for all data points.
    * active learning (e.g. Echard et al. [2011]): the algorithm needs not only to learn from the data but also needs to decide which training points, based on a limited budget, should be generated for learning. Adaptive or sequential data generation approaches would fit this kind of learning type which are presented in chapter 3.

- Unsupervised learning (e.g. Barlow [1989]):
  - For the unsupervised learning there is only one data set, without the differentiation of input and output. Typical examples for this kind of learning are cluster estimation or outlier / novelty detection.

- Reinforcement learning (e.g. Sutton et al. [1998]):
  - The output of the learning algorithm is in the form of rewards or punishments for its actions in a dynamic environment. Self-driving cars use this kind of learning. Other examples are computers which are able to play games like chess (Newborn [2012]), GO (Silver et al. [2016]) and other complex computer games (Vinyals et al. [2017]). This kind of learning is also used in the robotic area so that, for example, robot arms learn by themselves how to move instead of being programmed explicitly (Kormushev et al. [2010]).

Supervised machine learning will be focused on in this thesis. It is the task of learning a function that maps an input to an output based on example input-output pairs. These examples come typically from numerical simulations or experiments. The data is in most cases specified as a design or sample matrix:

$$X = \begin{Bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,n} \end{Bmatrix}$$

where its columns represent the n input parameters or features and its rows the N samples (designs / experiments / simulations). The output is a vector $\boldsymbol{y} = [y_1, y_2, ..., y_N]^T$ of the corresponding observations. Fig. 2.4 demonstrates how supervised machine learning works. It always starts with the data generation step to have some data the algorithm can learn from. For this, the user provides some input and the experiments / simulations return the corresponding outputs. These input and output can be used in a next step to train a machine learning algorithm on a computer. This is also referred as the training step. After the model has been trained, it can be utilized to predict the output for new input. This would be the prediction step.

Apart of different types of learning there are also different types of data, especially depending on the output data type, there are also further subgroups of algorithms. If

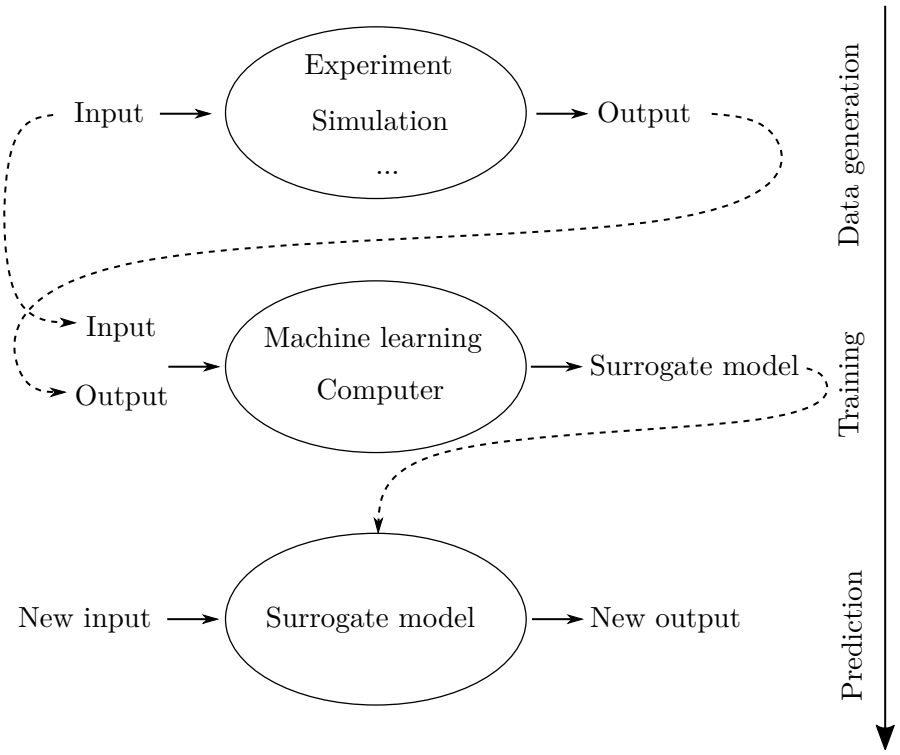


Figure 2.4: Overview of supervised machine learning.

the output data is real valued and continuous, the task of building a model is called regression. In contrast, if the output consists of discrete labels like a, b, c, it is called a classification task. The output could also be an image or video, in which case it is called image classification. For all these kinds of data types there are unique algorithms. Typical examples for regression methods are the response surface method (using polynomial models e.g. Marjavaara et al. [2007]) or moving least squares (Levin [1998]). For classification tasks, boosted decision trees like XGboost (Chen and Guestrin [2016]) or neural networks (Cybenko [1989]) are common choices and for image classification, convolutional neural networks (Krizhevsky et al. [2012]) are the state of the art. Fig. 2.5 gives an overview of the different learning and data types (without reinforcement learning) and some algorithms belonging to these categories. Most of them will also be covered in this work. The figure shows additional aspects like data generation, pre-processing and sensitivity analysis to reduce the amount of input parameters which are also important aspects of obtaining approximation models with a high prognosis quality. Prognosis quality means in this context that the predictions of the models are close or even equal to the true values. The aspects of data generation as well as sensitivity analysis are also covered in this work in the chapters 3 and 4.

As shown, there are plenty of algorithms for several different tasks. Each of these algorithms has some so-called hyperparameters and additional free parameters which will be optimized during the learning phase. The difference between these parameter definitions is that hyperparameters are typically set by the user and the additional free parameters will be estimated by an automatic learning (optimization) procedure based on the training data. To give an example: if a polynomial should be trained on a data set (see section 2.5) then the user needs to decide what the polynomial should look like and more precisely

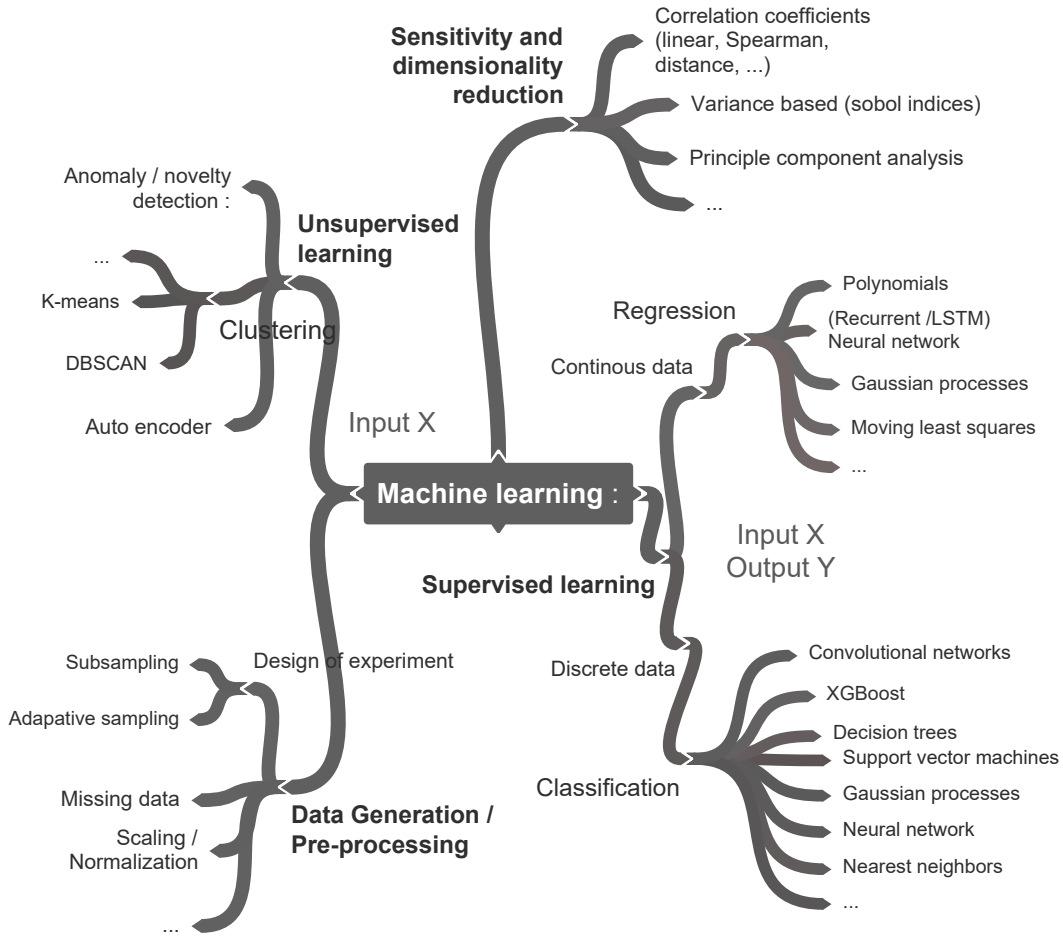


Figure 2.5: Overview of machine learning related learning and data types.

which regressors (terms in the model equation contributing to the model prediction) should be included. As an example, a linear polynomial for a problem with two input parameters x_1, x_2 can be built like

$$\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 \tag{2.1}$$

or a quadratic polynomial

$$\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 + \theta_4 x_1^2 + \theta_5 x_2^2 + \theta_6 x_1 x_2 \tag{2.2}$$

where θ_1, ..., θ_6 are the free coefficients. In this example the number of regressors is a hyperparameter set by the user and the coefficients θ_1, ..., θ_6 are the additional free parameters which will be estimated during the learning phase on the data. Nevertheless, it is also possible to use an optimization or brute force method to estimate these hyperparameters based on the data. For example, several different combinations of regressors can be tested automatically for the polynomial example and the best one is then used for the final model. There are further algorithms, like for example neural networks (see section 2.9)

which have even more hyperparameters and therefore makes them difficult to be applied on new tasks without spending a lot of time estimating these hyperparameters. In most cases, experience with the algorithms on different data sets is required to know how to set the hyperparameters correctly. Additionally, for many algorithms proper pre-processing of the data is required, like for example normalization or scaling of $\boldsymbol{y}$ and / or X, due to numerical issues. Also, a variable selection might increase the accuracy of a model. This makes it especially difficult for non-experienced users to get satisfying results if they apply machine learning on their data. Besides the right choice of the machine learning algorithm and the correct setting of the hyperparameters, a further choice that the user is faced with, is how to train the free parameters. This procedure is also known as model selection (see section 2.2). One machine learning algorithm can sometimes be trained via different optimization algorithms and strategies which deliver different models on the same training data. An overview is given in fig. 2.6.

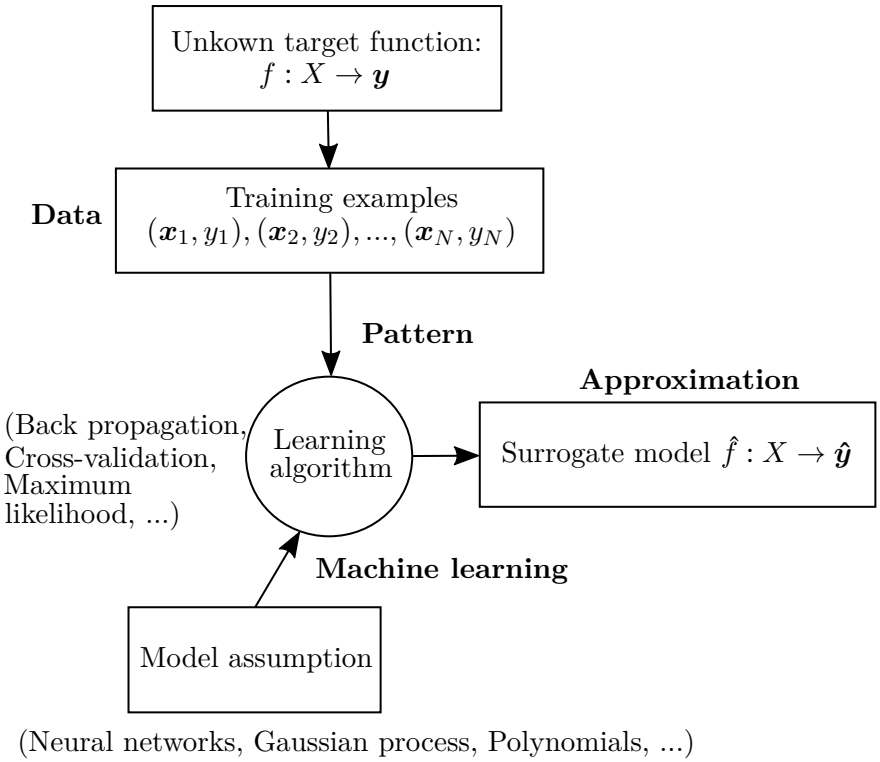


Figure 2.6: Overview of model selection.

Finally, after training a model, a last important step is necessary, namely the model validation. It gives the user information about the model prognosis quality and how it might behave on new test data. A model can for example have a very low error on the training set but a high error on new test points which might be an indicator for over-fitting (Hawkins [2004]). Model validation might also in some cases give helpful hints on how to improve the model if necessary.

To summarize the challenges to apply machine learning; it requires:

- **Design of experiment** (optional if no data is available): The usage of smart data generation plans which is covered in chapter 3.
- **Sensitivity analysis**: Variable selection and proper pre-processing of the data. This will be explained in the algorithm sections and in chapter 4.
- **Model selection**: The correct choice of machine learning algorithm dependent on the task to be solved and the data types. Model selection covers not only the choice of the right algorithm but also the training of their free and hyperparameters and will be explained in section 2.2 and partially in the corresponding algorithm sections for algorithm specific details. The chosen model selection method should be able to train the machine learning algorithm efficiently and should also find the global optimum of the free trainable parameters.
- **Model validation**: A reliable model validation to be sure of how the model performs on new test points. This is covered in section 2.11.

Therefore, the objective of this chapter is to give an overview of how to handle these challenges. The chapter will start with the topic of model selection, where some general methods will be presented on how to train machine learning models. Then several machine learning algorithms will be presented in detail which are all ingredients for the main new contribution, the deep Gaussian covariance network ($\mathcal{DGCN}$) which will be presented in section 2.10. The order of the algorithm sections should give the reader an idea how the $\mathcal{DGCN}$ algorithm was developed. The claim of the $\mathcal{DGCN}$ algorithm is to simplify many of the challenges mentioned so that it will be an easy-to-use algorithm without selecting any hyperparameters but still be on par with other complex algorithms concerning the accuracy. Additionally, it should be applicable for many different learning tasks, where different algorithms would usually be needed. Finally, there will also be a section about model validation which should give some hints on how to analyze a model correctly. There is no separate section about data pre-processing, since it will be mentioned in the algorithm sections if necessary. The topics of variable selection and data generation have their own chapters. As already mentioned in section 1.2.2, the above sequence of steps has been deliberately changed. The data generation was placed behind the explanation of the algorithms so that the influence of the data generation on the already described algorithms can be better represented. The influence of a sensitivity analysis on the models can also be better described if the models and their training methods have already been explained. Thus the chosen order of the topics is designed to show the connections and mutual influence of the methods as well as possible.

2.2 MODEL SELECTION

This section gives an overview of possible methods to estimate the free parameters (and hyperparameters) of machine learning algorithms in general which is also known as model selection. Besides the parameter estimation, it can also be used to choose between several different algorithm types based on their performance. Since the following methods can be used to train a wide range of algorithms, this chapter is placed before the detailed expla-

nation of some chosen machine learning algorithms so that only method-specific details of the parameter training need to be explained in those sections.

2.2.1 *Cross-Validation*

One way to train the parameters of a surrogate model $\hat{f}$ is to use cross-validation (CV) (Kohavi et al. [1995]). In cross-validation, the training data X and $\boldsymbol{y}$ is commonly split randomly into k folds (therefore known as k-fold cross-validation), where $k-1$ folds are used to train the model and the left fold with X_* and $\boldsymbol{y}_*$ (subscript $*$ indicates test points), is used to test the model. This procedure is repeated until all k folds are used once as a test fold which is illustrated in fig. 2.7. The predicted results $\hat{\boldsymbol{y}}$ of all test folds (after each is evaluated) can then be used to evaluate an optimization criteria in order to fit the model to the data. The free parameters (or hyperparameters) are changed in that way by the optimizer so that they improve the CV estimated metric.

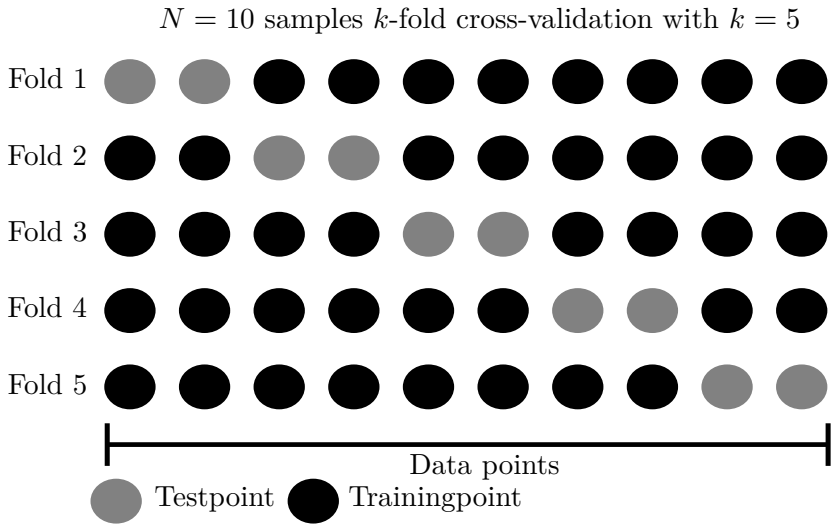


Figure 2.7: Illustration of k-fold cross-validation with $k = 5$ and sample size $N = 10$.

Different possible error metrics are available to train the parameters of a surrogate model $\hat{f}$ in connection with CV. One common choice is the predictive coefficient of determination $\hat{R}^2$ (e.g. used in Most and Will [2008]) which has the advantage over other metrics in that it also delivers an interpretable and comparable measure of the prognosis quality of the surrogate model. This metric estimates the ratio of the explained variance of y by the surrogate model divided by the true overall variance of y with

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}, \; \hat{R}^2 \in \mathbb{R}^{[0,1]} \tag{2.3}$$

so, if a model can explain the full variance of y the ratio would be 1,0. Other possible metrics are the root mean squared error ($RMSE$, Willmott and Matsuura [2005], eq. 2.4) which is the root of sum of squares of the computed residuals, the maximum absolute error ($MaxAE$, Rennie and Srebro [2005], eq. 2.5) or the mean absolute percentage error ($MAPE$, Lam et al. [2001], eq. 2.6). These measures are not very comparable for different

training sets, since they all rely on the scale of the observations $\boldsymbol{y}$ and prediction $\hat{\boldsymbol{y}}$. The corresponding equations are

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{N}} \tag{2.4}$$

$$MaxAE = max(|\boldsymbol{y} - \hat{\boldsymbol{y}}|) \tag{2.5}$$

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{2.6}$$

The right choice of k and the distribution of the data points inside the fold are important to obtain an unbiased estimation of the used metric (Bengio and Grandvalet [2004]). If k is set equal to N, which is also known as leave-one-out (LOO) CV (Vehtari et al. [2017]), then the metric used is in most cases less conservative (or more biased (Bengio and Grandvalet [2004])) than choosing a k of 10 to 15 which is mentioned and recommended by Friedman et al. [2001]. Furthermore, depending on the chosen machine learning algorithm and the number of training samples N, the computational costs to train the surrogate model k times can be numerically expensive.

As mentioned, a further important aspect which is rarely mentioned in literature, is the distribution of the data points in the folds. For example, it might be that the training points X are somehow sorted based on one of its parameters and so a biased CV scheme will be the outcome. A typical solution would be to shuffle the data, before using it for CV. There is also a third option called stratified k-fold CV (Zeng and Martinez [2000]). The idea of stratified cross-validation comes from classification tasks where each test fold should cover the same proportion of classes available. In the context of regression tasks, "stratified" means that the $\boldsymbol{y}_*$ values in each test fold should represent an equal range of values as in the whole data set. This can be accomplished by sorting the input and output after the ranks of $\boldsymbol{y}$ and then every $(i \times k) + j$-th element with $i \in \mathbb{Z}^{[0,\dots,N/k]}$, $j \in \mathbb{Z}^{[0,\dots,k-1]}$ is picked to build the j-th test fold (starting with 0 as the first fold). Points selected once are not available for the next test sets. This concept also works for symmetric functions as shown in fig. 2.8.

The difference between the three possible CV schemes can also be illustrated for an arbitrary two-dimensional test function with input parameters x_1, x_2. Fig. 2.9 shows a possible cross-validation procedure with $N = 30$ and $k = 6$. The input parameter x_1 is sorted by its rank and the folds are built without shuffling the data. Especially fold 1, 4, 6 show that the test points lie in an area without training points nearby which might result in an inaccurate prediction. In fold 1 and 6 the surrogate model even needs to extrapolate its predictions outside the known training space, since the x_1-range of the test points is outside the x_1-range of the training points. The second CV scheme is shown in fig. 2.10, where the training points are randomly shuffled before building the folds. It shows that the test points are more uniform distributed over the whole design space in most of the folds but randomly, like in fold 5, it can also look quite similar to the sorted scheme with the same drawbacks. The stratified CV scheme is shown in fig. 2.11. The test points are mostly uniformly distributed over the whole design space for each test fold and surrounded by other training points which results in a lower bias of the used error metric.

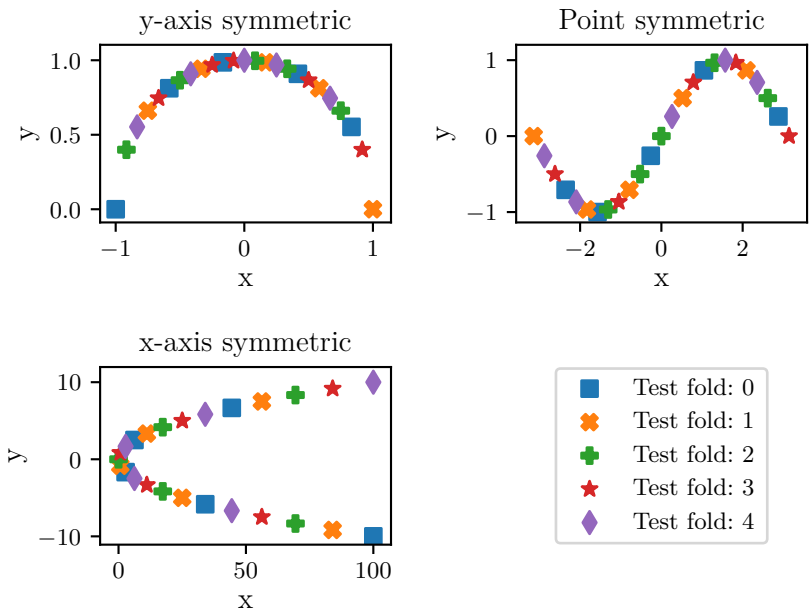


Figure 2.8: Stratified k-fold cross-validation with $k = 5$ and $N = 25$ for symmetric regression problems.

To investigate the influence of the CV scheme on the model accuracy further, a set of three two-dimensional test functions will be analyzed. The first test function is a modified version of the Branin-Hoo function (Goel et al. [2007]) with two local minima and one global minimum taken from Forrester et al. [2008] (see fig. 2.12). The modification compared to the original version is the addition of $5x_1$ to the response which results in two local minima and one global minimum instead of three equivalent global minima:

$$f(\boldsymbol{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left[\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 1\right] + 5x_1, \quad x_1 \in \mathbb{R}^{[-5,10]},\ x_2 \in \mathbb{R}^{[0,15]} \tag{2.7}$$

The second one is the Himmelblau function (Himmelblau [1978]) which has 4 global minima (see fig. 2.13):

$$f(\boldsymbol{x}) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2,\ x_1 \in \mathbb{R}^{[-5,5]},\ x_2 \in \mathbb{R}^{[-5,5]} \tag{2.8}$$

The third one is the highly non-linear Eggholder test function (Sekaj [2004]) with one single global optimum (see fig. 2.14):

$$f(\boldsymbol{x}) = -(x_2 + 47)\sin\left(\sqrt{|0.5x_1 + (x_2 + 47)|}\right) - x_1\sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right), \quad x_1 \in \mathbb{R}^{[-512,512]},\ x_2 \in \mathbb{R}^{[-512,512]} \tag{2.9}$$

The task is to build approximation models which represent the test functions as well as possible. Instead of using the CV directly to optimize the free parameters of a model, it should be used to validate the model prognosis quality on these test functions. The models

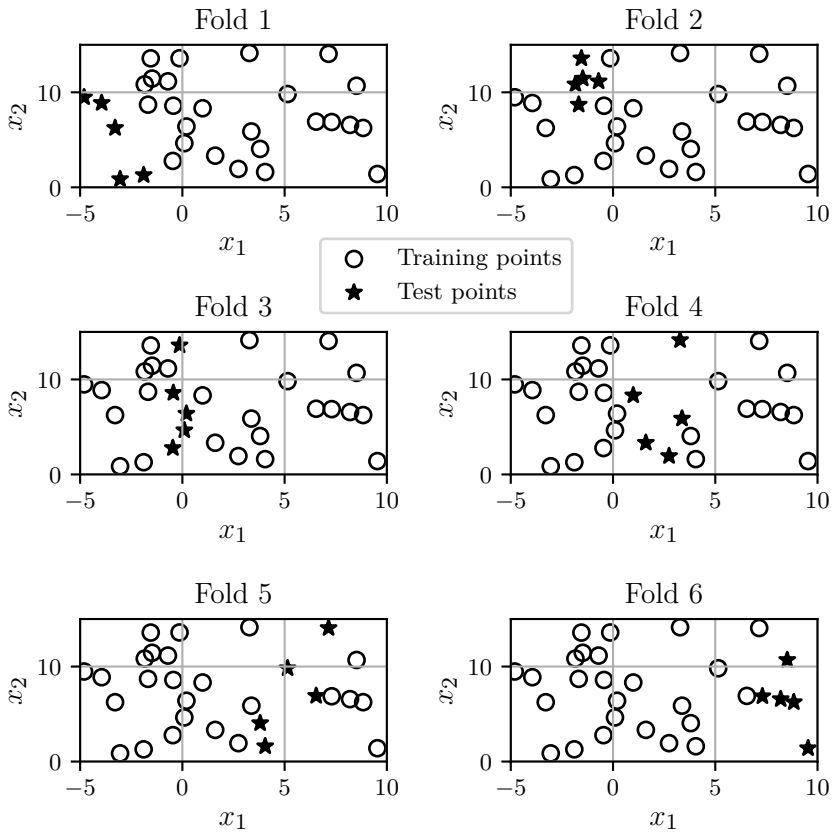


Figure 2.9: k-fold ($k = 6$) cross-validation for a training data set ($N = 30$) with sorted parameter x_1 without subsequent shuffling.

are trained internally (during each fold evaluation) via its specific training algorithm (see the corresponding sections in this chapter) This means, if for example the fold size is $k = 5$, the model trains five times with the given training points of the CV scheme and predicts five times the test points so that in the end all available points have been predicted once. It could be argued that this test should then be made in the model-validation section 2.11 but this test should show how the estimated metric is influenced by the number of samples, fold number k and package scheme. This is important for both ways CV is used, either model selection or model validation. Each test should be repeated 25 times to estimate the variance of the shuffled-CV-scheme-based metric. These repetitions make this investigation numerically very expensive. If the model parameters had also been trained via CV, several nested CV loops for the parameter optimization would have been required. Furthermore, three different fold sizes should be tested with $k = 5$, $k = 10$ and $k = 15$ and three different training sample sizes with $N = 30$, $N = 120$ and $N = 300$ uniformly distributed between the parameter limits, so in total 9 combinations were tested, each repeated 25 times for three different CV schemes. For the sorted scheme the data is sorted after x_1. To check if the results are model dependent, the following algorithms should be tested: quadratic polynomials (RSM) (section 2.5), deep neural networks (DNN) with one hidden layer and 100 neurons (section 2.9), Gaussian processes ($\mathcal{GP}$) with the absolute exponential covariance function (section 2.8), deep Gaussian covariance networks ($\mathcal{DGCN}$) (section

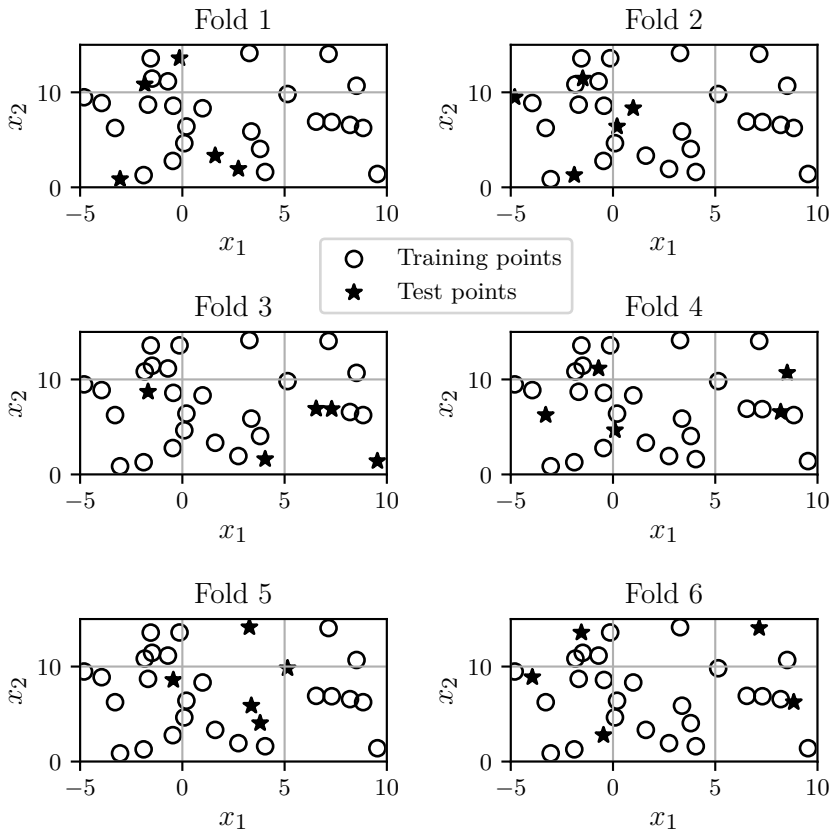


Figure 2.10: k-fold ($k = 6$) cross-validation for a training data set ($N = 30$) with sorted parameter x_1 with subsequent shuffling.

2.10). The test metrics should be the $\hat{R}^2$ metric of eq. 2.3. To validate the CV estimated metric, a test set with $N = 100,000$ points is used. For the validation, the models are retrained on the complete data set which is recommended, if the CV is only used for model validation. A fixed seed is set to exclude the influence through the model optimizers on the results

As an additional analysis, an estimation of the prognosis quality of Gaussian-process-based algorithms via CV should be tested without retraining the model for each fold. In this approach the Gaussian process (and the $\mathcal{DGCN}$ model) are trained on the full data without CV to determine the free parameters. Then, to estimate the prognosis quality, the free parameters are kept fixed and only the input / output database is updated with the corresponding CV set of training points in each fold. Since the Gaussian process uses a weighted linear combination of the training observations for its predictions, it has a huge influence if the training database is changed (see section 2.8). The coefficients of these weighted linear combination are based on a so-called correlation length-scale parameter which determines how strong the points around the point to be predicted should influences the result. Therefore, if points are excluded from the database it does not make a huge difference if the correlation length-scale was trained on the full data, since the information from the omitted points is still lost. It can even lead to over optimistic results under certain circumstances, if the model would be retrained as illustrated in fig 2.15. In this

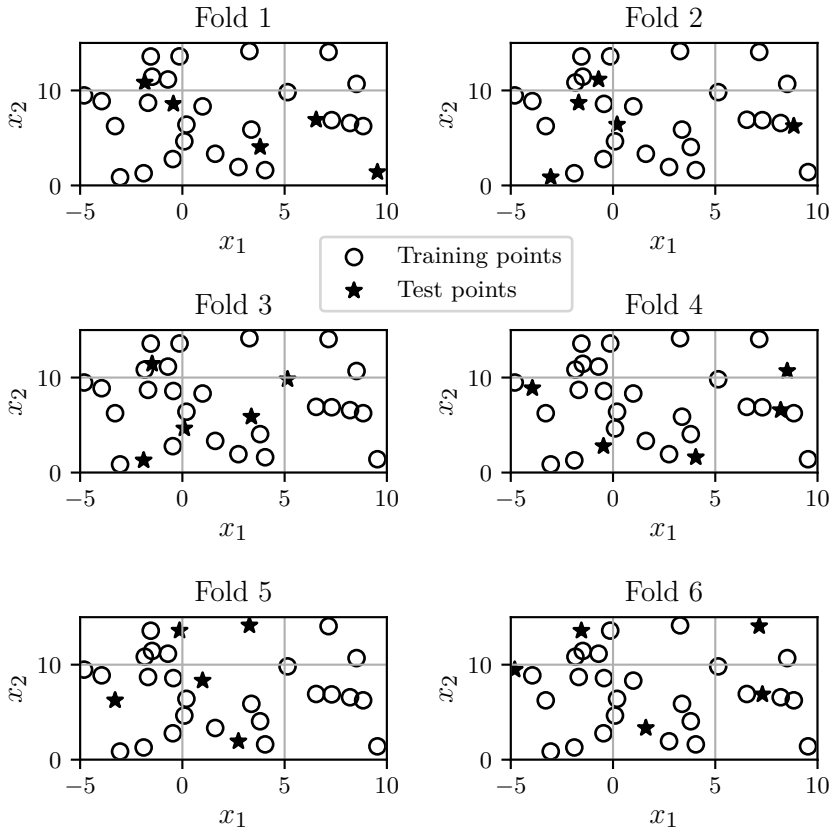


Figure 2.11: k-fold ($k = 6$) stratified cross-validation for a training data set ($N = 30$) with sorted parameter x_1.

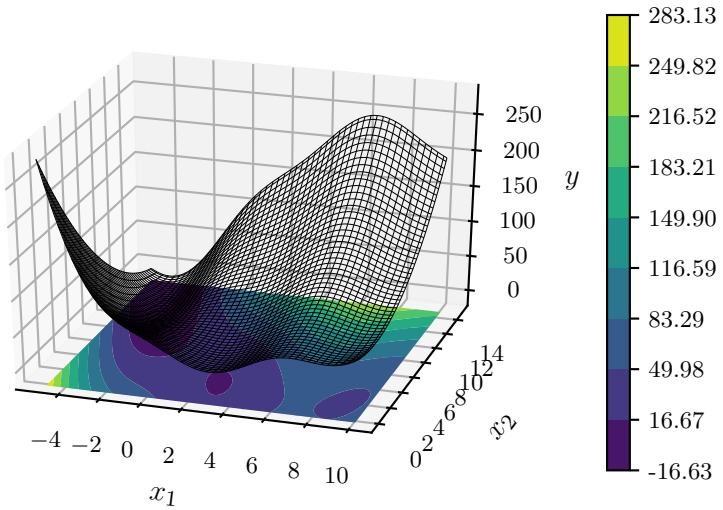


Figure 2.12: Modified Branin-Hoo function.

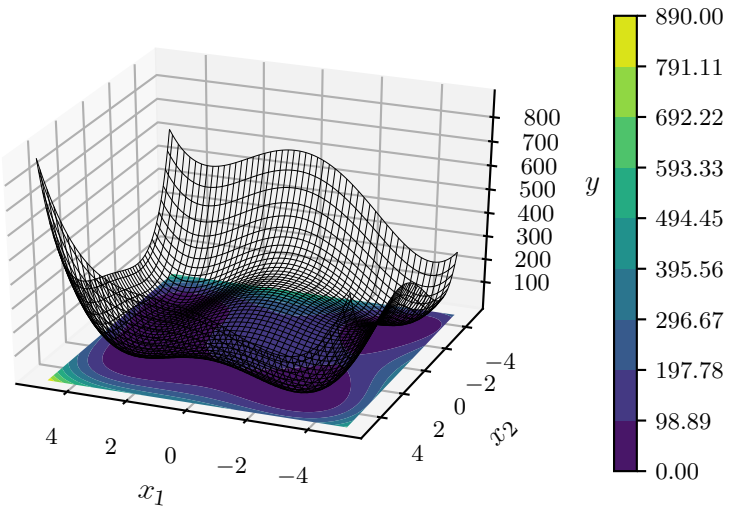


Figure 2.13: Himmelblau function.

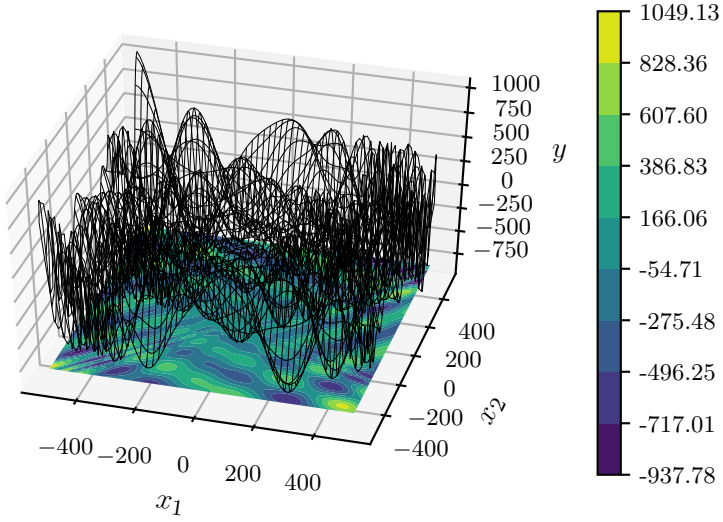


Figure 2.14: Eggholder function.

figure, the difference is demonstrated between a retrained and only updated Gaussian process model on the prediction of two test points of a *CV* subset. The retrained model increases its correlation length-scale during the retraining which can be seen by the color of the covariance values of the training points compared to the colors based on the full set so that the remaining training points are considered more strongly when predicting the missing points. This leads to a much better prediction compared to the updated version, where the correlation length-scale stays constant, and which in this case is too small to make a good prediction of the test points. So, based on this demonstration, the results of the retrained-model-based *CV* prognosis quality would be too optimistic. Since the free parameters of the model trained on the full data set would be used after the *CV* and therefore the updated model shows the more conservative behavior of the model for later

predictions. This also shows that if an algorithm is not regularized very well, the retraining might hide an over-fitted model.

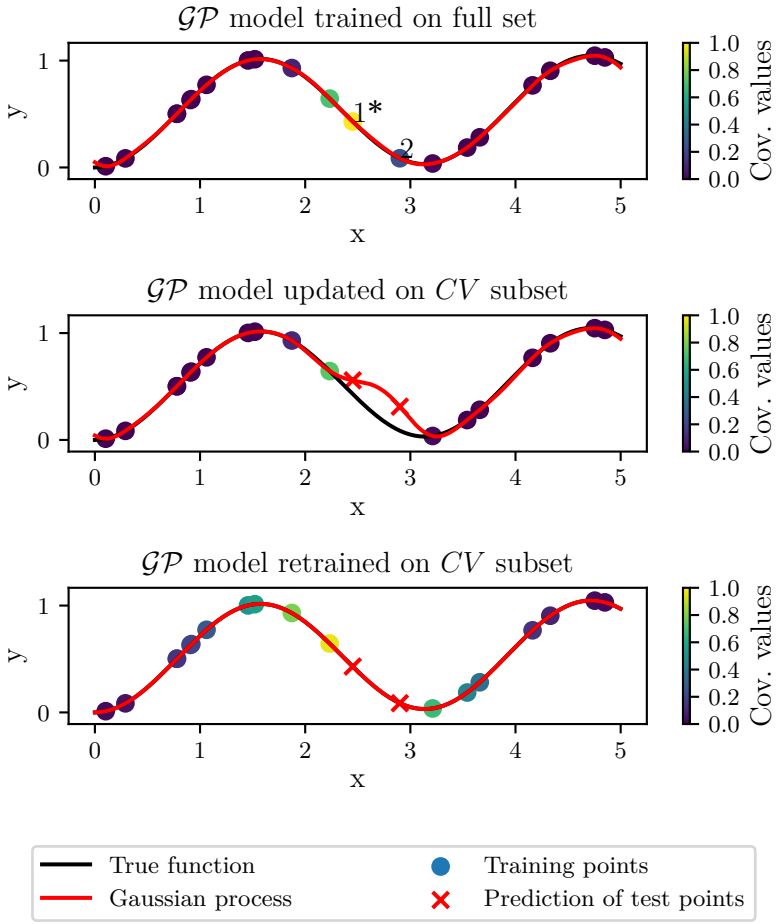


Figure 2.15: Gaussian process model with free parameters estimated on the full data set. Two points are left out for a CV test subset and should be predicted by the model. The model in the middle is updated and the model at the bottom is retrained on the subset. The test point annotated with 1^* is used to show the covariance values (illustrated via color) to the other training points as an indicator for the correlation length-scale of the retrained and updated $\mathcal{GP}$ models (see section 2.8).

For the Gaussian process model the training points information is stored in a $N \times N$ covariance matrix which describes the covariances between each training point. The removal of test points during the CV iterations is equal to removing the corresponding columns and rows of the full covariance matrix. This procedure was also suggested in Rasmussen and Williams [2006] (section 5.4) and Sundararajan and Keerthi [2000] for the leave-one-out CV scheme. For the LOO CV scheme, the difference is marginal removing one data point from the covariance matrix compared to using the full matrix. Thus, it is interesting if the removal of many more points in the k-fold scheme is also feasible. Usually, other methods like polynomials or neural networks do not use the training database for predictions and therefore this approach is generally not applicable to them. This investigation will analyze

if this approach does not over- or underestimate the validated metric. If it delivers reliable results, it would speed up the CV usage for model validation with Gaussian-process-based algorithms, drastically. These only updated models are named in the result as $\mathcal{GP}_{up}$ and $\mathcal{DGCN}_{up}$.

In tab. 2.1 and 2.2 the results of this investigation are summarized. These tables show the difference between the CV estimated $\hat{R}^2$ ($\hat{R}^2_{cv}$) based on the training data and the validated $\hat{R}^2$ based on the $100,000$ test points not used for training ($\hat{R}^2_{val}$) with

$$\epsilon_{cv} = \hat{R}^2_{val} - \hat{R}^2_{cv},\ \epsilon_{cv} \in \mathbb{R}^{[0,1]} \tag{2.10}$$

so that a positive ϵ_{cv} value represents a conservative estimation of the $\hat{R}^2_{cv}$ value and a negative sign indicates an over optimistic estimation of $\hat{R}^2_{cv}$. For the shuffled CV scheme, the mean value and standard deviation is given. The values marked in bold show the best result between the three CV schemes for the given number of training samples. For the comparison of the best scheme, the maximum absolute value of the mean value $\pm$ standard deviation of the shuffled scheme is used. The green values with zero difference have a perfect $\hat{R}^2$ value of one. The red values are results, where the model was not able to learn anything and therefore had a $\hat{R}^2$ of zero. The intention behind the more complex Eggholder function was to have no perfect $\hat{R}^2$ values for any model, especially for the $\mathcal{GP}$ and $\mathcal{DGCN}$ models because if the model is perfect, the difference to the validation is zero in all cases. The same holds for models which have a $\hat{R}^2$ of zero which for the Eggholder function mostly concerns the RSM and DNN model. The most interesting test cases to compare the $\hat{R}^2$ values against the validation are where the model is neither perfect nor totally wrong.

	N=30			N=120			N=300		
$k = 5$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.707	0.048±0.14	**-0.023**	0.781	0.06±0.03	**0.052**	0.783	0.001±0.005	**-0.004**
DNN	0.347	-0.014±0.023	**-0.032**	0.471	0.064±0.025	**0.055**	0.467	0.01±0.005	**-0.006**
$\mathcal{GP}$	0.591	0.016±0.028	**-0.003**	0.24	0.008±0.0013	**-0.003**	0.182	**0.001±0**	**0.001**
$\mathcal{GP}_{up}$	0.513	0.009±0.023	**-0.007**	0.22	0.008±0.014	**-0.003**	0.176	**0.001±0**	**0.001**
$\mathcal{DGCN}$	0.86	**0.057±0.073**	0.181	0.227	0.035±0.042	**0.002**	0.058	0.008±0.01	**-0.003**
$\mathcal{DGCN}_{up}$	0.133	-0.035±0.016	**-0.049**	0.068	**0±0**	**0**	0.053	**0±0**	**0**
$k = 10$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.707	0.029±0.036	**-0.005**	0.351	0.035±0.015	**0.041**	0.318	**0±0.002**	-0.003
DNN	0.134	-0.003±0.017	**-0.016**	0.175	0.028±0.018	**0.019**	0.165	**0.006±0.002**	**0.008**
$\mathcal{GP}$	0.069	-0.004±0.023	**-0.016**	0.088	0.001±0.009	**-0.004**	-0.039	**0±0**	**0**
$\mathcal{GP}_{up}$	0.041	-0.008±0.021	**-0.019**	0.087	0.001±0.009	**-0.004**	0.038	**0±0**	**0**
$\mathcal{DGCN}$	0.263	0.026±0.074	**0.051**	0.128	0.02±0.012	**-0.003**	0.007	0.011±0.014	**-0.005**
$\mathcal{DGCN}_{up}$	-0.02	-0.043±0.009	**-0.049**	0.003	**0±0**	**0**	0.003	**0±0**	**0**
$k = 15$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.097	0.02±0.018	**0.011**	0.174	0.055±0.016	**0.04**	0.145	**0.001±0.002**	**-0.003**
DNN	0.039	-0.008±-0.009	**0.009**	0.128	0.041±0.16	**0.027**	0.082	**0.001±0.002**	0.007
$\mathcal{GP}$	**0.001**	-0.014±0.007	-0.016	0.034	**0.001±0.01**	-0.003	0.016	**0±0**	**0**
$\mathcal{GP}_{up}$	**-0.001**	-0.017±0.006	-0.019	0.035	**0.001±0.01**	-0.003	0.016	**0±0**	**0**
$\mathcal{DGCN}$	**0.019**	0±0.061	0.079	0.073	0.017±0.012	**0.006**	0.022	**0.007±0.007**	0.021
$\mathcal{DGCN}_{up}$	-0.042	-0.047±0.005	**-0.048**	0.001	**0±0**	**0**	0.001	**0±0**	**0**

	$N = 30$			$N = 120$			$N = 300$		
$k = 5$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.199	0.192±0.026	**0.199**	0.317	0.05±0.04	**0.02**	0.331	0.036±0.013	**0.017**
DNN	0.302	**0.094±0.073**	0.194	0.467	0.047±0.034	**0.033**	0.592	0.085±0.013	**0.05**
$\mathcal{GP}$	0.556	0.077±0.043	**0.038**	0.433	-0.037±0.02	**-0.032**	0.374	0.004±0.02	**0.003**
$\mathcal{GP}_{up}$	0.541	0.053±0.043	**0.026**	0.414	-0.04±0.019	**-0.034**	0.373	0.004±0.002	**0.003**
$\mathcal{DGCN}$	0.462	0.169±0.14	**0.098**	0.438	0.021±0.037	**-0.009**	0.44	0.032±0.03	**0.036**
$\mathcal{DGCN}_{up}$	0.543	0.047±0.05	**0.009**	0.33	-0.012±0.006	**-0.012**	0.276	**0±0**	**0**
$k = 10$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.199	0.19±0.023	**0.199**	0.317	0.038±0.002	**0.001**	0.331	0.032±0.008	**0.027**
DNN	0.269	**0.026±0.043**	0.087	0.327	**-0.026±0.024**	-0.052	0.447	0.039±0.008	**0.017**
$\mathcal{GP}$	0.355	0.023±0.021	**0.029**	0.118	-0.049±0.011	**-0.059**	0.151	0.002±0.002	**0.002**
$\mathcal{GP}_{up}$	0.335	0.012±0.021	**0.025**	0.118	**-0.051±0.01**	-0.06	0.151	0.002±0.001	**0.002**
$\mathcal{DGCN}$	0.657	0.084±0.113	**0.222**	0.228	0.011±0.029	**-0.007**	0.221	0.034±0.023	**0.039**
$\mathcal{DGCN}_{up}$	0.311	-0.02±0.024	**0.011**	0.062	-0.016±0.004	**-0.019**	0.035	**0±0**	**0**
$k = 15$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0.199	0.194±0.01	**0.177**	0.204	0.034±0.015	**0.025**	0.235	0.03±0.006	**0.026**
DNN	**0.026**	-0.047±0.034	-0.064	0.141	-0.031±0.014	**-0.036**	0.179	**-0.001±0.006**	-0.018
$\mathcal{GP}$	0.052	0.012±0.015	**-0.003**	0.018	**-0.054±0.005**	**-0.059**	0.056	**0.002±0**	**0.002**
$\mathcal{GP}_{up}$	0.049	0.005±0.014	**-0.008**	0.017	-0.055±0.005	**-0.059**	0.056	0.002±0	**0.001**
$\mathcal{DGCN}$	**0.027**	0.084±0.112	-0.075	0.083	**0.008±0.024**	0.035	0.043	0.033±0.023	**0.024**
$\mathcal{DGCN}_{up}$	0.04	-0.01±0.02	**-0.026**	0.001	**-0.017±0.002**	-0.02	0.008	**0±0**	**0**

Table 2.1: Overview of CV results for the Branin-Hoo function (top) and the Himmelblau function (bottom). The results show the difference $\epsilon_{cv} = \hat{R}^2_{val} - \hat{R}^2_{cv}$. Bold marked results show the best results for the specific sample count N. The green values with zero difference have a perfect $\hat{R}^2_{cv}$ value of one. The red values are results with a $\hat{R}^2_{cv}$ of zero.

	$N = 30$			$N = 120$			$N = 300$		
$k = 5$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0	0±0	0	0	0±0	0	0	0±0	0
DNN	0	0±0	0	0	0±0	0	0	0±0	0
$\mathcal{GP}$	0	-0.052±0.056	-0.105	**0.129**	-0.084±0.072	-0.211	0.387	-0.021±0.04	**-0.005**
$\mathcal{GP}_{up}$	0	-0.051±0.056	-0.096	**0.129**	-0.084±0.072	-0.21	0.387	-0.021±0.039	**-0.005**
$\mathcal{DGCN}$	0	-0.012±0.035	0	0	-0.043±0.067	-0.009	0.541	-0.042±0.066	**0.062**
$\mathcal{DGCN}_{up}$	0	0±0	0	0.02	**-0.174±0.111**	-0.294	0.353	-0.034±0.029	**0.021**
$k = 10$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0	0±0	0	0	0±0	0	0	0±0	0
DNN	0	0±0	0	0	0±0	0	0	0±0	0
$\mathcal{GP}$	-0.159	-0.054±0.05	**-0.083**	**0.08**	-0.135±0.06	-0.188	0.387	-0.057±0.03	**-0.067**
$\mathcal{GP}_{up}$	-0.162	-0.053±0.05	**-0.08**	**0.079**	-0.135±0.06	-0.189	0.379	-0.057±0.03	**-0.067**
$\mathcal{DGCN}$	0	-0.007±0.018	0	0	-0.109±0.086	-0.068	0.541	-0.098±0.051	**-0.021**
$\mathcal{DGCN}_{up}$	-0.104	-0.001±0.03	0	-0.083	-0.263±0.072	**-0.331**	0.353	-0.061±0.03	**-0.045**
$k = 15$	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.	Sort.	Shuff.	Strat.
RSM	0	0±0	0	0	0±0	0	0	0±0	0
DNN	0	0±0	0	0	0±0	0	0	0±0	0
$\mathcal{GP}$	0	-0.042±0.03	-0.072	**-0.007**	-0.164±0.048	-0.214	0.221	-0.06±0.02	**-0.067**
$\mathcal{GP}_{up}$	0	-0.039±0.029	-0.068	**-0.007**	-0.165±0.048	-0.215	-0.221	-0.06±0.02	**-0.066**
$\mathcal{DGCN}$	0	-0.006±0.027	0	**-0.036**	-0.14±0.108	-0.15	0.481	-0.108±0.055	**-0.091**
$\mathcal{DGCN}_{up}$	0	0±0	0	-0.119	-0.283±0.066	**-0.324**	0.242	-0.064±0.02	**-0.06**

Table 2.2: Overview of CV results for the Eggholder function. The results show the difference $\epsilon_{cv} = \hat{R}^2_{val} - \hat{R}^2_{cv}$. Bold marked results show the best results for the specific sample count N. The green values with zero difference have a perfect $\hat{R}^2_{cv}$ value of one. The red values are results with a $\hat{R}^2_{cv}$ of zero.

In the following the results and conclusions of tab. 2.1 and 2.2 can be summarized as follows:

- The stratified scheme shows in average the most reliable results and the sorted scheme the worst.
- Even though the mean value of the shuffled scheme is sometimes comparable or even lower than the stratified scheme, it has partially very high standard deviations. Especially, if the number of training samples is low which makes it also very unreliable.
- The lower the number of training samples is, the higher the difference between the CV estimated metric and the validation tends to be, except for the sorted scheme. This makes it very difficult to know when the sample size is enough to be able to rely on the estimated CV metric. Especially, if the number of training samples is very low, a very good metric can be the result of over-fitting. Therefore, it is recommended to have a sufficient regularization of the model and to validate the results, if possible, with a further test set. Another approach is to use a sequential or adaptive sampling approach where the convergence of the CV estimated metric can be proofed.

- If the metric is nearly perfect (except if the model is totally over-fitted) or extremely bad, it can be assumed that the CV estimated metric is reliable.
- For the Eggholder function it can be seen that even with a higher number of samples the CV estimated metric is not reliable. This leads to the assumption that there is no rule for the number of samples which can ensure a reliable result. The complexity of the underlying data is also an important factor.
- The results show that an increased value of k also leads to more reliable results and reduces the variance of the shuffled scheme. A too high value of k can lead to a very biased result and should be avoided.
- In most cases, especially if the ϵ_{cv} value is higher, the estimated $\hat{R}^2_{cv}$ is conservative. Mostly if the ϵ_{cv} is very small, like for most of the stratified CV results, the estimated $\hat{R}^2_{cv}$ are slightly overly optimistic which is acceptable.
- The retrained and updated Gaussian process algorithm shows very similar results regarding the differences of the $\hat{R}^2$ values. Therefore, it seems to be legitimate to use this variant for the $\mathcal{GP}$ model, since it speeds up the CV estimation drastically.
- For the $\mathcal{DGCN}$ model, the difference between the retrained and updated version are more significant than for the $\mathcal{GP}$ model, even though its prediction is also based on a $\mathcal{GP}$ (see section 2.10 for its details). For the $\mathcal{DGCN}$ model, the test shows that the updated version is more reliable then the retrained version, since the differences are in average smaller and partially even much smaller.
- Based on these test results, the CV estimated metric for algorithms which use the input and output for predictions (like the $\mathcal{GP}$ model), can be estimated to be equivalently reliable without having to retrain the whole model for each subset.
- A further possible reason for the difference ϵ_{cv} is the chosen metric itself. Since it is divided by the full variance of y, the validation set of 100,000 points might have a different variance compared to the variances in the training sets. So especially, if the number of training samples N is low and only a small part of the true function is captured, the full variance might be much lower compared to the one in the validation set. If the total variance in the nominator is lower, it is easier to get a higher $\hat{R}^2$ value and this would lead to an over optimistic result compared to the validation.

Similar conclusions were also made by Kohavi et al. [1995]. In their work a Naive-Bayesian classifier algorithm was trained via k-fold CV, and they investigated the influence of the number of folds k on the accuracy of the model. Kohavi et al. [1995] conclude that for k-fold CV, moderate values of k between 10 and 20 reduces the variance of the model accuracy but increases the bias. For lower values of k $(2-5)$ and lower sample sizes N, the variance of the model accuracy increases due to the instability of the training. Kohavi et al. [1995] further conclude that the stratified CV strategy is generally a better scheme, both in terms of bias and variance compared to the regular shuffled k-fold CV scheme. Bootstrapping presented in Efron and Gong [1983] or Efron [1992] was also tested in Kohavi et al. [1995] which is a resampling method with replacement, so N samples are drawn from the training set but some might be duplicates and therefore some are also left out. This resampled set is used to train the model and the original training set without duplicates is used for

validation. Compared to CV there is an overlap between training and test points which can lead to a serious underestimation of the true prediction error. This was proven by that benchmark presented in Kohavi et al. [1995] which shows that bootstrapping has an extremely large bias on some problems and is therefore not recommended. A further actual overview of CV schemes can also be found in Arlot et al. [2010].

The model accuracy highly depends on the number of training points N, as illustrated in tab. 2.1 and 2.2. If the number of training points N is too low in combination with a over-fitted model, the resulting $\hat{R}^2$ is not reliable, regardless of the CV scheme used. This can lead to poor validation results, as presented. Therefore, regularization is a very important aspect in model selection and some techniques will be presented in the following algorithm sections. Cross-validation-based parameter training itself, is also a regularization method which can be applied to a wide range of models (Arlot et al. [2010]). However, this alone might not be enough regularization to avoid over-fitting and is, as already mentioned, a quite expensive way to train a model. A deeper analysis of model validation methods and possible further ways to estimate model performance will also be given in section 2.11, especially from the perspective of using a probabilistic model.

2.2.2 *Maximum Likelihood Estimation*

Another common method of model selection is to formulate a maximum likelihood criterion (Konishi and Kitagawa [1996]). The maximum likelihood estimation (MLE) is a method in statistics to estimate the parameters of a statistical model (for example for a Gaussian distribution) based on given observations. The objective of the MLE is to find the parameter values that maximize the likelihood that the model leads to the observations. Since the objective of the parameter training of surrogate models is also to find a model which provides the best possible representation of the data, MLE can also be used for the training of the free model parameters. In this case, the parameters of the model should be optimized to maximize the likelihood that the surrogate model predicts the observations. Note that this model selection method is only applicable to probabilistic models (Myung [2003]). In literature the likelihood function is often named as an error function, since maximizing the likelihood is equivalent to minimizing the error (see Bishop [2006] and section 2.5).

Technically, there is a difference between probabilities for discrete variables and probability densities for continuous variables. Throughout this work the term "probability" is used to refer to both because in machine learning problems the observed variables (e.g. y the measured output of a simulation or experiment) is only described by discrete values (the measurements). Therefore, in the following, the normal distribution symbol $\mathcal{N}$ is used as a synonym for the probability density function since the probability density in the continuous domain is analogous to the probability in the discrete domain (see e.g. Rasmussen and Williams [2006]). In the context of the MLE in which the probability is to be maximized for continuous variables, one would speak of the probability density for discrete variables. Furthermore, the symbol $|$ can stand for the conditional probability (e.g. in $p(y|x)$) but can also be the probability function depending on its parameters (e.g. in $p\left(\boldsymbol{y}|\mu,\sigma^2\right)$) because in Bayesian statistics the unknown parameters μ,σ^2 are also treated as random variables (see e.g. Lee [1989]) and so $p\left(\boldsymbol{y}|\mu,\sigma^2\right)$ is the conditional probability of observing $\boldsymbol{y}$ given μ,σ^2. For example, the Gaussian distribution for a single real-valued variable y which is important for section 2.8, is defined by

$$p\left(y|\mu,\sigma^2\right) = \frac{1}{\left(2\pi\sigma^2\right)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}\left(y-\mu\right)^2\right\} \tag{2.11}$$

with

$$p\left(y|\mu,\sigma^2\right) \sim \mathcal{N}(\mu,\sigma^2)$$

This function is described by two parameters; the mean μ and the variance σ^2. Furthermore, a data set of observations $\boldsymbol{y} = (y_1, ..., y_N)^T$ exists that represents the N observations of the scalar variable y. It is assumed that these observations are drawn from a Gaussian distribution (eq. 2.11) whose mean μ and variance σ^2 are unknown which means these are random variables and should be estimated via the maximum likelihood approach from the given observations $\boldsymbol{y}$. As a further assumption, the observations $\boldsymbol{y}$ are drawn independently from the same distribution and therefore called independent and identically distributed. With defining each point of the given observations $\boldsymbol{y}$ as a separate event $p(y_i)$, the probability of the data set $\boldsymbol{y}$, given μ and σ^2 can be described by the product of the probabilities of each single event $p(y_i)$ (see e.g. Rasmussen and Williams [2006] or Bishop [2006]) with

$$p\left(\boldsymbol{y}|\mu,\sigma^2\right) = \prod_{i=1}^{N} p\left(y_i|\mu,\sigma^2\right) \tag{2.12}$$

Eq. 2.12 is the likelihood function for the Gaussian distribution function given a data set $\boldsymbol{y}$ and relies on the parameters μ, σ^2. To find the best values for μ, σ^2 in order to fit the Gaussian distribution to the given data set $\boldsymbol{y}$, it is a common practice to maximize the likelihood function (this holds for every kind of distribution function). In practice, it is more useful to maximize the log-likelihood function for various reasons. The logarithm is a monotonically increasing function of its parameters and the maximization of the logarithm of a function is equivalent to maximizing the function itself. If the likelihood function contains an exponential term, in most cases the log-likelihood is mathematically simpler. For computational reasons, it is more robust to take the sum of small probabilities instead of taking the product which can lead to significant round-off errors ($log(xy) = log(x) + log(y)$). The definition of the log-likelihood for the Gaussian distribution can be given with the help of eq. 2.11 and eq. 2.12 as

$$\ln p\left(\boldsymbol{y}|\mu,\sigma^2\right) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i-\mu\right)^2 - \frac{N}{2}\ln\left(\sigma^2\right) - \frac{N}{2}\ln\left(2\pi\right) \tag{2.13}$$

Before the general likelihood function for a surrogate model can be derived, there is one important thing to mention about the likelihood estimation and the tendency to over-fit the data. In Bishop [2006] the expectation for the estimated parameters μ_{MLE}, σ^2_{MLE} of the MLE of a Gaussian distribution are given by

$$\mathbb{E}[\mu_{MLE}] = \mu \tag{2.14}$$

and

$$\mathbb{E}[\sigma^2_{MLE}] = \left(\frac{N-1}{N}\right)\sigma^2 \tag{2.15}$$

These equations show that, on average, the maximum likelihood estimate is correct for the mean but underestimates the true variance by a factor of $(N-1)/N$. In practice, this is only a problem if N is small, since for $N \to \infty$ the maximum likelihood solution results in the true variance as demonstrated in fig. 2.16.

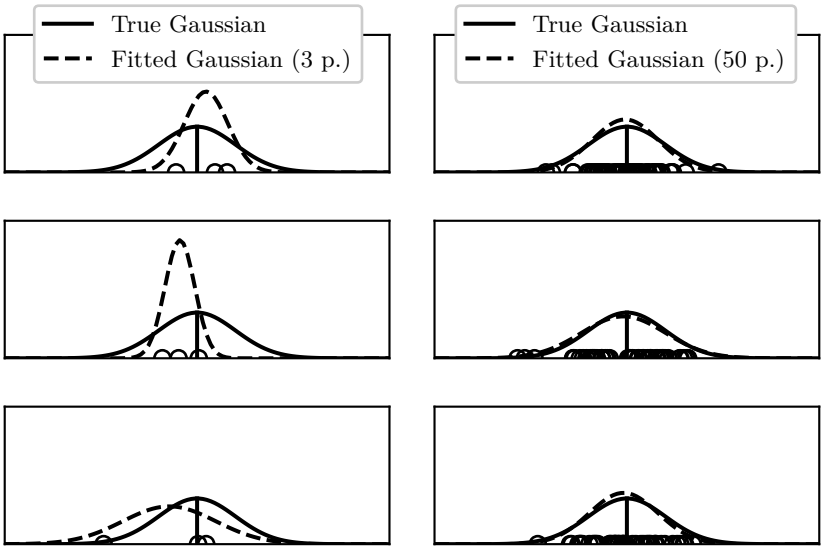


Figure 2.16: Examples of fitting a Gaussian distribution via MLE on $N = 3$ and $N = 50$ training points. In average the mean is correct, but the variance is underestimated for $N = 3$.

To transfer the MLE concept to a surrogate model $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$ with its free parameters $\boldsymbol{\theta}$ and the training data $\boldsymbol{x}$ ($n = 1$, therefore a vector instead of a matrix), once again it is necessary firstly to define a probability distribution over the uncertainty of the target values $\boldsymbol{y}$. For simplicity, it is assumed again that, for a given set of points $\boldsymbol{x} = [x_1, ..., x_N]^T$, the corresponding output values $\boldsymbol{y} = [y_1, ..., y_N]^T$ are Gaussian distributed with the mean equal to the predictions of the surrogate model $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta})$. The variance of the prediction is described by $\hat{\sigma}^2$ with

$$p\left(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \hat{\sigma}^2\right) \sim \mathcal{N}\left(\boldsymbol{y}|\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}), \hat{\sigma}^2\right) \tag{2.16}$$

If the observations $\boldsymbol{y}$ are assumed to be independently drawn from its distribution (eq. 2.16), then the likelihood can be written as

$$p\left(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \hat{\sigma}^2\right) = \prod_{i=1}^{N} p\left(y_i|\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}), \hat{\sigma}^2\right) \tag{2.17}$$

Once again it would be more useful to optimize the log-likelihood function which can be obtained from eq. 2.11 as

$$\ln p\left(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}, \hat{\sigma}^2\right) = -\frac{1}{2\hat{\sigma}^2} \sum_{i=1}^{N} \left[y_i - \hat{f}\left(x_i, \boldsymbol{\theta}\right)\right]^2 - \frac{N}{2} \ln\left(\hat{\sigma}^2\right) - \frac{N}{2} \ln(2\pi) \tag{2.18}$$

If a constant $\hat{\sigma}^2$ is assumed, all parts of the eq. 2.18, except the part containing the free parameters $\boldsymbol{\theta}$, can be omitted, since the constant parts do not change the position of the maximum regarding $\boldsymbol{\theta}$:

$$\ln p\left(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}\right) = \sum_{i=1}^{N}\left[y_i - \hat{f}\left(x_i,\boldsymbol{\theta}\right)\right]^2 \tag{2.19}$$

This is the well-known sum-of-squares error function. Note that the negative sign changed to a positive here and eq. 2.19 must be minimized, since the values of $\boldsymbol{\theta}$ that maximize eq. 2.18 are the same values that minimize the negative of eq. 2.18. Therefore, maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function for the special case of a Gaussian distribution. This function is often used as an error function in optimization, for example to find the coefficients of a polynomial model (see section 2.5). If the free parameters $\boldsymbol{\theta}$ are estimated via the MLE, the model can be used for new predictions of x. Since the model was defined as a probabilistic model, the prediction can also be seen as a predictive distribution that gives the probability distribution over y instead of a point estimation. This can be used to estimate the uncertainty of the model prediction or also to handle noisy data as shown in fig. 2.17.

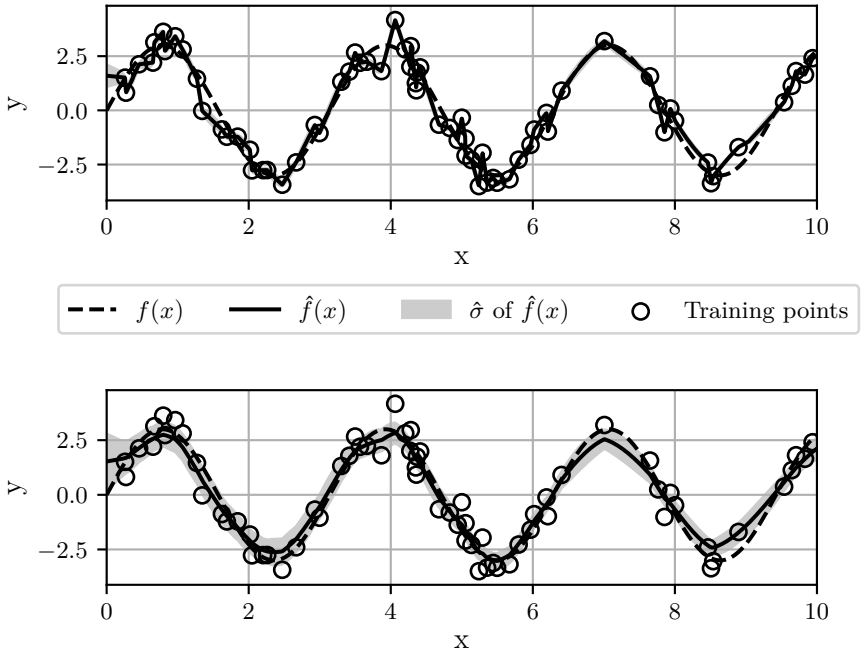


Figure 2.17: Example of fitting two $\mathcal{DGCN}$ models via MLE on $N = 70$ training points. The upper model has a fixed noise parameter $\sigma^2 = 0$ (so no noise for the training) and the lower one was allowed to train its noise parameter. This results in a different fit of the models and also in a different prediction of the standard deviation $\hat{\sigma}$ of the prediction. The true function is $f(x) = 3\sin(2x)$, $x \in \mathbb{R}^{[0,10]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. $\hat{f}(x)$ is the approximation and $f(x)$ the true function.

The main advantage of using a likelihood function over CV is the computational cost. Since no repeated evaluations of different subsets are needed, it is much faster to evaluate the likelihood. A disadvantage of using MLE for model selection, is the possible over-fitting of the model to the data which was shown in the Gaussian distribution example (see fig. 2.16 and 2.17). A further possible disadvantage of the MLE is the right choice of likelihood

function. Since it depends on assumptions made about the data and the error distribution, a wrong choice of the likelihood function can lead to a poor predictive accuracy. Therefore, some previous statistical tests on the data might be helpful.

Important notes:

- If k-fold cross-validation is used for model selection, higher values for k deliver more reliable results with a lower variance ($10 - 15$ recommended). A too high value of k can lead to biased results.
- A low number of samples might also deliver unreliable CV results, depending on the complexity of the data. A sequential or adaptive sampling approach might help to find the correct number of samples. Additionally, if a probabilistic model is used, the variance of predictions can be analyzed.
- A stratified cross-validation scheme is more reliable than a shuffled or sorted scheme and should always be favored over the others.
- If possible, a validation of the CV metric is always recommended.
- The predictive coefficient of determination $\hat{R}^2$ is independent of the scaling of the observations and therefore useful for comparisons between different output parameters and models. Nevertheless, it depends on the variance of the output.
- The maximum likelihood approach is an alternative to the CV training. It can be used for probabilistic models if the corresponding underlying assumptions for the data are fulfilled. The main advantage of this approach is its computational efficiency in the training process and the possibility to judge the prediction's uncertainty probabilistically (e.g. via confidence intervals). This might be more helpful than a deterministic prognosis coefficient.

2.3 HETEROSCEDASTICITY IN REGRESSION

A topic rarely mentioned in most machine learning literature, is the heteroscedasticity of the residuals a model can produce by predicting new points. This means the error of the prediction is not equal in all areas of the design space. Specifically, heteroscedasticity is a systematic change in the spread of the residuals over the range of observations. Heteroscedasticity is a problem for many machine learning algorithms, for example the response surface method (RSM) using polynomials (section 2.5) assumes that all residuals have a constant variance (homoscedasticity). Therefore, to satisfy the regression assumptions and to trust the results, the residuals should have a constant variance in the design space. Fig. 2.18 shows an example of two linear RSM models trained on a sinusoidal function, once with a constant amplitude and once with an amplitude dependent on the input x. In the case of the constant amplitude, the residual vs. prediction plot shows nearly homoscedastic errors over the whole range of predictions. For the dynamic amplitude, the residuals increase more in a specific range of y. This means that the model is less trustworthy in one

region compared to others. Also, this information is not gained by some prognosis quality estimation and should be analyzed as a part of model validation (see section 2.11).

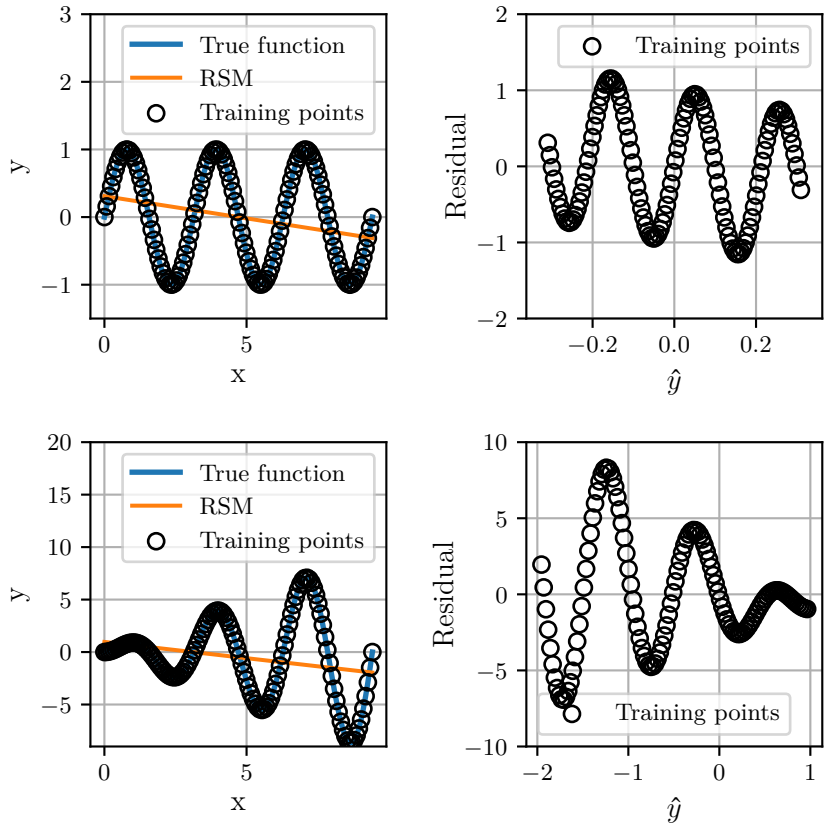


Figure 2.18: Two *RSM* models trained on a sinusoidal function with constant (top) and dynamic (bottom) amplitude. Furthermore, the residual vs. prediction is plotted to show the homoscedasticity and heteroscedasticity.

Heteroscedasticity occurs more often in data sets which have a large range between the smallest and largest observation or local changes in the smoothness of response (Muller et al. [1987]). There might also be further reasons, such as, for example, the data is based on a process which becomes more noisy or unstable in a specific region which would be heteroscedasticity of the noise. If the data becomes noisier in a specific region, it is not intended that the model fits that noise. So, the problem is that the model should have equal residuals over the whole observation range but only if the variance of the residuals is not based on more noise in some regions. In most cases, the heteroscedasticity of the residuals correlates with one or more of the input features, as shown in the sinusoidal example, where the amplitude correlates with x as do the residuals. To handle the heteroscedasticity of the residuals, the correct choice of model is necessary. In this example, a *RSM* model with a sinusoidal regressor would be beneficial to decrease the variance of the residuals. During this work, it will be shown that there are methods to fix or reduce this problem even for the case of heteroscedastic noise in the data.

Important notes:

- A possible heteroscedasticity of the residuals is an important thing to check for which belongs to the mandatory step of model validation (see section 2.11).
- If heteroscedasticity of the residuals is present, estimated metrics like for example the predictive coefficient of determination, might be not reliable over the whole design space or they under- and / or overestimate different areas of the design space.
- In most cases the reason for heteroscedasticity of the residuals are specific input parameters which might also be missing in the model or the data itself is corrupted with noise.
- It is also possible to have heteroscedasticity of the noise which means the noise level varies over the design space.

2.4 CURSE OF DIMENSIONALITY

Some of the machine learning algorithm presented in this work will utilize the distance information between the training and test data points to estimate the points with highest influence on the point to predict. The Gaussian process in section 2.8 is one example but also the moving least squares algorithm of section 2.7 uses the nearest neighbors of the point to predict, to estimate the prediction value. Therefore, both algorithms need to estimate the distances d_{ij} between points i and j (typically the test and training points) with the help of a distance function, like for example the Euclidean distance function

$$d_{ij} = \left[\sum_{k=1}^{n} \left(x_{i,k} - x_{j,k} \right)^2 \right]^{\frac{1}{2}} \tag{2.20}$$

The distance is estimated over all n dimensions of the data points. The problem with such formulations is that if n gets large, there is little difference in the distances between different pairs of samples (Aggarwal et al. [2001]):

$$\lim_{n\to\infty} \mathbb{V}\left(\frac{d_{ij}}{\mathbb{E}\left[d_{ij} \right]} \right) = \lim_{n\to\infty} \mathbb{V}\left(\frac{\left[\sum_{k=1}^{n} \left(x_{i,k} - x_{j,k} \right)^2 \right]^{\frac{1}{2}}}{\mathbb{E}\left[\left[\sum_{k=1}^{n} \left(x_{i,k} - x_{j,k} \right)^2 \right]^{\frac{1}{2}} \right]} \right) = 0 \tag{2.21}$$

which means

$$\frac{d_{\max} - d_{\min}}{d_{\min}} \xrightarrow[n\to\infty]{} 0 \tag{2.22}$$

with d_{max}, d_{min} being the maximum and minimum distance between all points. Eq. 2.22 is also known as the relative contrast. It is used to define the meaningfulness of distance metrics (Aggarwal et al. [2001]). This means that the difference between the maximum and minimum distances to a given test point does not increase as fast as the nearest distance to any point in high dimensional space. This makes it more difficult for the algorithm to find the correct training points to be used for the prediction, since all points are similar

far away of the point to predict. This problem is also known more generally as the curse of dimensionality (Bellman [1966], Keogh and Mueen [2011]). Cursed phenomena occur also in other domains, such as numerical analysis, sampling, combinatorics, data mining and databases (Theodoridis and Koutroumbas [2009]). The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse (Trunk [1979]).

To overcome this problem, it is very important for algorithms, like the Gaussian process or the moving least squares, to reduce the number of input dimensions to only important ones or to use dimensionality reduction methods, like for example the principle component analysis (PCA) (Wold et al. [1987]) (see subsection 2.10.5) or methods of sensitivity analysis presented in chapter 4 as a pre-processing step. Methods like the Gaussian process or neural networks have also the capability to reduce the influence of unimportant parameters automatically which is called automatic relevance determination (Neal [2012]) and will be presented in section 2.6. Furthermore, a new extension for the moving least squares algorithm, explained in subsection 2.7.1, allows this parameter reduction also for this model algorithm.

Important notes:

- The curse of dimensionality refers to various phenomena that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional settings.
- It is especially a problem for machine learning algorithms which utilize the distance information between training and test points for their predictions.
- Sensitivity analysis or dimensionality reduction methods like principle component analysis might help as pre-processing step.
- Some algorithm are capable to automatically reduce the influence of unimportant parameters which is called automatic relevance determination.

2.5 POLYNOMIALS

Polynomial models, also called response surface method (RSM) (see e.g. de Oliveira Filho et al. [2018]), are supervised regression models and are one of the most widely used surrogate models because of their simplicity (see e.g. Zhou et al. [2005], Marjavaara et al. [2007]). A recently published paper Cheng et al. [2018] investigated different test functions and concluded that polynomials can even outperform complex neural networks in some circumstances. They are a good choice if the relationship between input X and output $\boldsymbol{y}$ is not very complex. Furthermore, they belong to the few methods which are also sometimes used for extrapolation (Zhan et al. [2012]). They are also often part of ensembles of surrogate models as presented in Goel et al. [2007]. A very good overview of this method can be found in Box et al. [1987]. Since this method is the foundation of the moving least square method in section 2.7 and it can also be used for sensitivity analysis as shown in section

4.3 as well as it introduces the idea of a probabilistic model, an overview of this method is given in the following.

To start, a function $y = f(x)$ with a single input parameter $n = 1$, $x \in \mathbb{R}^1$ is considered. So that the approximated response of this function can be written in general as

$$y = \hat{f}(x) + \hat{\epsilon} \tag{2.23}$$

with

$$\hat{y} = \hat{f}(x) \tag{2.24}$$

where $\hat{f}(x)$ is the chosen model (e.g. the RSM) and its prediction error $\hat{\epsilon}$. $\hat{y}$ donates the approximated response value of the model. Both the response value as well as the error should be viewed as random variables to derive the MLE of the free model parameters. If the choice of the RSM model is assumed to be of order m, eq. 2.23 can be written as

$$y(x, m, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_m x^m + \hat{\epsilon} = \theta_0 + \sum_{i=1}^{m} \left(\theta_i x^i\right) + \hat{\epsilon} \tag{2.25}$$

In this equation is $\boldsymbol{\theta} = \{\theta_0, \theta_1, \ldots, \theta_m\}^T$ the vector of the free parameters to fit the RSM to the training data (note that the maximal order m is a hyperparameter typically chosen by the user). The equation can also be written in the more general matrix notation as

$$\boldsymbol{y} = P\boldsymbol{\theta} + \hat{\boldsymbol{\epsilon}} \tag{2.26}$$

where $\boldsymbol{y}$ and $\hat{\boldsymbol{\epsilon}}$ are $N \times 1$ vectors and P is a $N \times (mn + 1)$ matrix also known as the Vandermonde matrix (assuming not to use any cross-terms between the input parameters), containing the $(mn + 1)$ regressors (here for simplicity in the case for $n = 1$):

$$P = \begin{Bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \ldots & x_N^m \end{Bmatrix}$$

The $+1$ term results from the constant intercept term θ_0. In order to estimate these free parameters $\boldsymbol{\theta}$ with size $((mn + 1) \times 1)$, the sum of squared errors (SSE) can be used which is also the likelihood for these parameters under the assumption that the error has a Gaussian distribution (as already explained in subsection 2.2.2):

$$SSE = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = (\boldsymbol{y} - P\boldsymbol{\theta})^T (\boldsymbol{y} - P\boldsymbol{\theta}) \tag{2.27}$$

The values of $\boldsymbol{\theta}$ that minimize this sum are also known as the ordinary least square (OLS) solution. The global minimum of eq. 2.27 can be found by differentiating with respect to the vector $\boldsymbol{\theta}$ (assuming that the second derivatives are > 0) and setting the equation as equal to zero leads to

$$\begin{aligned} \frac{\delta}{\delta \boldsymbol{\theta}} SSE &= \frac{\delta}{\delta \boldsymbol{\theta}} \left(\boldsymbol{y}^T \boldsymbol{y} - P^T \boldsymbol{\theta}^T \boldsymbol{y} - P^T \boldsymbol{\theta}^T \boldsymbol{y} + P^T \boldsymbol{\theta}^T P \boldsymbol{\theta}\right) \\ &= -2P^T \boldsymbol{y} + 2P^T P \boldsymbol{\theta} = 0 \end{aligned} \tag{2.28}$$

The least square solution estimation for $\boldsymbol{\theta}$ then yields (under the assumption that $P^T P$ is invertible)

$$\boldsymbol{\theta} = \left(P^T P\right)^{-1} P^T \boldsymbol{y} \tag{2.29}$$

where $(P^T P)^{-1} P^T$ is the Moore–Penrose pseudo-inverse of P (inverse of non-square matrices). The final equation for the RSM prediction is defined by (eq. 2.23)

$$\hat{\boldsymbol{y}} = P\boldsymbol{\theta} \tag{2.30}$$

It is also possible to estimate the variance $\hat{\boldsymbol{\sigma}}^2$ of the predictions including the error. As already mentioned, the least square estimation is the same as the MLE if the error $\hat{\boldsymbol{\epsilon}}$ in eq 2.26 has a Gaussian distribution with $\mathcal{N}(\mathbf{0}, \sigma^2 I)$. I denotes the identity matrix with size $N \times N$ which is used here to work with a multivariate Gaussian distribution described by the covariance matrix K instead of a single variance:

$$K\left(\boldsymbol{\theta}\right) = \mathbb{V}\left(P\boldsymbol{\theta}\right)\left(P^T P\right)^{-1} \tag{2.31}$$

so that eq. 2.29 is distributed as

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\boldsymbol{\theta}, \mathbb{V}\left(P\boldsymbol{\theta}\right)\left(P^T P\right)^{-1}\right)$$

and is uncorrelated to the regression residuals $\hat{\boldsymbol{\epsilon}}$ (see Johnson et al. [2014])

$$\hat{\boldsymbol{\epsilon}} = \boldsymbol{y} - \hat{\boldsymbol{y}} = \boldsymbol{y} - P\boldsymbol{\theta} \tag{2.32}$$

$\mathbb{V}(P\boldsymbol{\theta})$ denotes the variance of the free parameter estimations (or the variance of the model prediction without the residuals) and can be determined based on the OLS solution (Johnson et al. [2014]) with

$$\mathbb{V}(P\boldsymbol{\theta})_{OLS} = \frac{\hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}}}{N - (mn+1)} \tag{2.33}$$

or also based on the MLE

$$\begin{aligned} \mathbb{V}(P\boldsymbol{\theta})_{MLE} &= \frac{N - (mn+1)}{N} \mathbb{V}(P\boldsymbol{\theta})_{OLS} \\ &= \frac{N - (mn+1)}{N} \frac{\hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}}}{N - (mn+1)} = \frac{\hat{\boldsymbol{\epsilon}}^T \hat{\boldsymbol{\epsilon}}}{N} \end{aligned} \tag{2.34}$$

The $N - (mn+1)$ term represents the statistical degree of freedom. $\mathbb{V}(P\boldsymbol{\theta})_{OLS}$ and $\mathbb{V}(P\boldsymbol{\theta})_{MLE}$ are quite similar if $N >> (mn+1)$, for the following equations the $\mathbb{V}(P\boldsymbol{\theta})_{OLS}$ will be used, since it is more unbiased. The variance estimation of either of these equations can also be used to estimate the importance of coefficients in $\boldsymbol{\theta}$. If the confidence interval (CI) based on this variance spans from a negative limit to a positive limit, i.e. includes the value 0.0 for one coefficient, it might be dropped. The variance of the full model prediction $\hat{\boldsymbol{\sigma}}^2$ can then be estimated through the help of the variance of the prediction error $\hat{\boldsymbol{\epsilon}}$ and the variance of the free parameters estimation itself $\mathbb{V}(P\boldsymbol{\theta})$ (eq. 2.33) with

$$\hat{\boldsymbol{\sigma}}^2 = \mathbb{V}(\hat{\boldsymbol{\epsilon}}) + \mathbb{V}(P\boldsymbol{\theta}) \tag{2.35}$$

The variance of the forecast error $\mathbb{V}(\hat{\boldsymbol{\epsilon}})$ can be estimated via (Johnson et al. [2014])

$$\mathbb{V}(\hat{\boldsymbol{\epsilon}}) = \boldsymbol{p}_*^T K(\boldsymbol{\theta})\, \boldsymbol{p}_* = \boldsymbol{p}_*^T \mathbb{V}(P\boldsymbol{\theta}) \left(P^T P\right)^{-1} \boldsymbol{p}_* \tag{2.36}$$

whereby $\boldsymbol{p}_*$ is the vector containing the regressors for the point $\boldsymbol{x}_*$ to predict and P is based on the training points X. Therewith, the variance of the model prediction $\hat{\boldsymbol{\sigma}}^2$ can finally be estimated through

$$\begin{aligned} \hat{\boldsymbol{\sigma}}^2 &= \boldsymbol{p}_*^T \mathbb{V}(P\boldsymbol{\theta}) \left(P^T P\right)^{-1} \boldsymbol{p}_* + \mathbb{V}(P\boldsymbol{\theta}) \\ &= \mathbb{V}(P\boldsymbol{\theta}) \left(1 + \boldsymbol{p}_*^T \left(P^T P\right)^{-1} \boldsymbol{p}_*\right) \end{aligned} \tag{2.37}$$

This equation also gives the possibility of approximating the confidence interval of the model prediction $\hat{\boldsymbol{y}}$ (see (Johnson et al. [2014])) via

$$\hat{\boldsymbol{y}}_{CI} = P\boldsymbol{\theta} \pm t\left(\frac{\alpha}{2}|N - (mn+1)\right) \sqrt{\mathbb{V}(P\boldsymbol{\theta}) \left(1 + \boldsymbol{p}_*^T \left(P^T P\right)^{-1} \boldsymbol{p}_*\right)} \tag{2.38}$$

$t\left(\frac{\alpha}{2}|N-(mn+1)\right)$ denotes the upper $100\left(\frac{\alpha}{2}|N-(mn+1)\right)$ percentile of a Student-t-distribution with $N-(mn+1)$ degrees of freedom.

To estimate the model accuracy of a RSM, the coefficient of determination is often used (the same as the predictive coefficient of determination without CV) with

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2},\ R^2 \in \mathbb{R}^{[0,1]} \tag{2.39}$$

$\bar{y}$ denotes the mean of the observations. This estimate is a biased estimate of the population R^2 and will not decrease if additional regressors are added, even if they are irrelevant. Therefore, there is also the adjusted R^2_{adj} which penalize the number of regressors which do not add to the explanatory power of the regression:

$$R^2_{adj} = 1 - \frac{N-1}{N-(mn+1)} \left(1 - R^2\right),\ R^2_{adj} \in \mathbb{R}^{[0,1]} \tag{2.40}$$

One of the advantages of RSM compared to many other machine learning methods is that it does not need any expensive optimization to train the model. The free parameters (except the hyperparameter m) can be estimated analytically via the presented approach. As shown in eq. 2.25 the polynomial approximation of order m is a Taylor series expansion truncated after $m+1$ terms (Box et al. [1987]). Therefore, more terms (a greater value of m) should lead to a more accurate model. However, a value for m which is too high makes the model very prone to over-fitting, especially if noise or outliers are present in the training data. This may lead to the situation where the model might fit very well to the training data but is very poor in generalization and therefore worse in predicting new points. Consequently, there must be a way to find the appropriate value for m without losing the generalization. k-fold cross-validation, which was already mentioned in subsection 2.2.1, is one common way to estimate the order of m (Cherkassky and Mulier [2007], Forrester et al. [2008]). Since the numerical effort to train the RSM is low, a straightforward approach would be to test different discrete values for m, $m \in \mathbb{Z}^{[\geq 0]}$ and choose the one with the lowest CV error. The maximum number of m is limited by the number of training samples N such that $(mn+1) <= N$ or it would be a underdetermined problem. Dependent on the choice of k in the CV scheme, m is limited by the size $(mn+1) <= N-(N/k)$ of

the k-fold training subset. Fig. 2.19 illustrates an example of choosing m via 10-fold CV. The best result is obtained with $m = 12$. Additionally, the 95% confidence interval of the model prediction is shown.

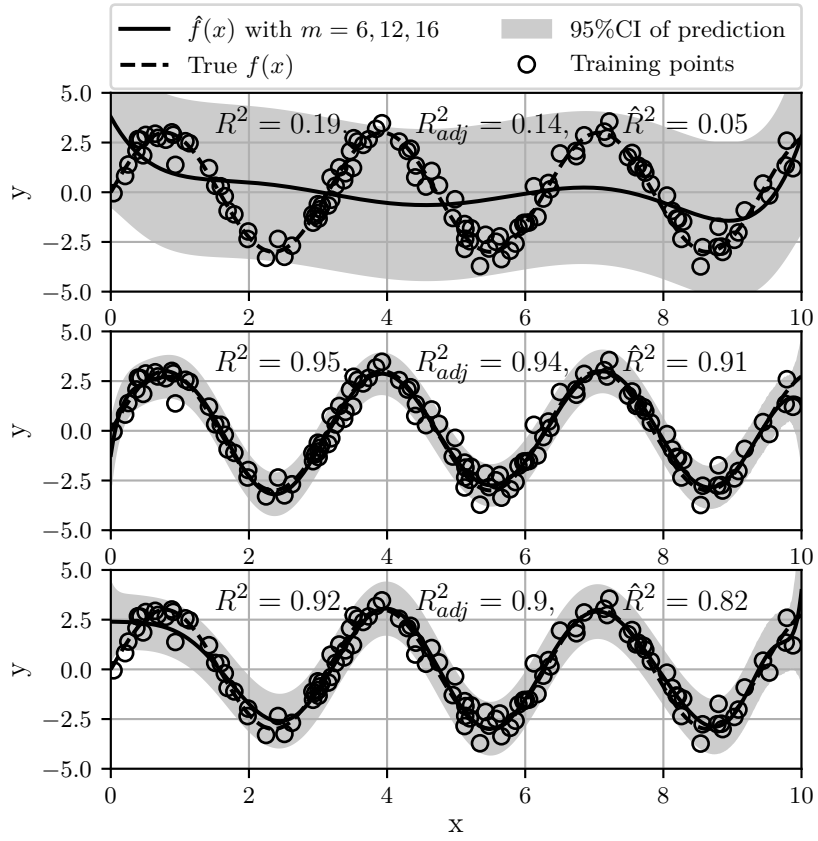


Figure 2.19: RSM trained for $m = 6$, 12, 16 (top $\rightarrow$ down) and $N = 120$. Evaluated the best m value via 10-fold CV and the $\hat{R}^2$ metric. The true function is $f(x) = 3\sin(2x)$, $x \in \mathbb{R}^{[0,10]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$.

RSM are also sometimes useable for extrapolation (if m is not too high) which is not possible (or should be avoided) with other methods such as, for example, Gaussian processes without a mean function. Nevertheless, RSM models are, in most cases, only suitable if the relationship between input and output is not too complex and it should only be used if $(mn + 1) <= N$ which might also be a problem if the number of input parameters n is very high. One disadvantage of the RSM is that it is prone to heteroscedasticity of the residuals as previously shown in section 2.3.

There are also variations of the RSM, such as ridge regression (also known as Tikhonov regularization or weight decay) (Hoerl and Kennard [1970]) which uses an additional regularization term in the OSL to stabilize the values of the free parameter $\boldsymbol{\theta}$ in the case of an underdetermined or overdetermined problem (avoids over-fitting) so that eq. 2.27 changes to

$$RSSE = \sum_{i=1}^{N} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta}\right)^2 + \lambda \sum_{j=1}^{m+1} \tilde{\theta}_j^2 = (\boldsymbol{y} - P\boldsymbol{\theta})^T (\boldsymbol{y} - P\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2 \tag{2.41}$$

This additional term avoids very high values for the free parameters $\boldsymbol{\theta}$ and therefore reduces the variance of the model. Another approach which is also based on the RSM and uses some kind of parameter shrinkage is called moving least squares (Levin [1998]) (MLS) and is presented in the next section.

Important notes:

- Polynomials are fast training supervised models which are capable of building models for more simple problems.
- They can be trained via MLE or OLS which turns out to be same under the assumption of a Gaussian error distribution.
- If trained via MLE, it is also possible to approximate the confidence interval of the prediction.
- The hyperparameters power and type of regressors may be found with the help of CV.
- RSM models are prone to heteroscedasticity of the residuals and noise, compared to other algorithm like for example Gaussian processes (see section 2.8).
- The free parameters $\boldsymbol{\theta}$ of a trained RSM model can be used for sensitivity analysis of the input parameters (see. chapter 4).

2.6 AUTOMATIC RELEVANCE DETERMINATION

A topic that has only been mentioned in the margins so far, is the characteristic of some machine learning algorithms to reduce or even completely neglect unimportant parameters without any extra preliminary sensitivity analysis (see chapter 4). For example, Gaussian processes or neural networks are capable of this. Many machine learning methods (especially distance-based methods) experience problems with high dimensional data which is also known as the curse of dimensionality (as already mentioned in section 2.4). The ability of machine learning methods to reduce the influence of unimportant parameters itself, is known in literature as automatic relevance determination (ARD) (MacKay and Neal [1994], Williams and Rasmussen [1996], Wipf and Nagarajan [2008], Neal [2012]). Sometimes models with this property are also called anisotropic models. The difference between isotropic and anisotropic models (Xiao [2009]) is that anisotropic models can control the influence of the different input parameters separately and therefore reduce it if necessary. In general ARD describes the property to reduce the influence of unimportant parameters with the help of additional free parameters in the model training if it helps to reduce the model loss function (see section 2.2). Typically, ARD is known as a Bayesian method for feature selection and sparse learning (Qi et al. [2004]) because it finds the relevance of features by optimizing the model marginal likelihood (the loss function), also known as the evidence. All models presented in this work can be viewed as a Bayesian model dependent how the loss function is defined. For RSM models, it was shown in section 2.5, how the known least squares error can be derived from a Bayesian point of view. Therefore, this also applies to

its extension the moving least squares model presented in the following section 2.7. Typically, a likelihood function is also used to train a Gaussian processes (see section 2.8) and there are also so called Bayesian neural network (Neal [2012]) which are not covered in this work.

The *ARD* property can be demonstrated for *RSM* models (see section 2.5 for more details). Assuming a problem with 3 input parameters x_1, x_2, x_3, $\in \mathbb{R}^{[0,1]}$, so $n = 3$, and one single output y. The true function should be

$$y(\boldsymbol{x}) = 5x_1 + 2x_2 \tag{2.42}$$

so that x_3 is unimportant. A sufficient linear *RSM* model for this would be

$$\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \tag{2.43}$$

To train this model, 20 sample points uniform distributed are used. Using eq. 2.29 of section 2.5 to estimate the least square solution for the coefficients θ_1 to θ_3 yields the correct results: $\theta_1 = 5$, $\theta_2 = 2$, $\theta_3 = 0$. The estimated R^2_{adj} value is 1.0. Via minimizing the least squares solution or maximizing the likelihood, like explained in subsection 2.2.2, the influence of the unimportant parameter x_3 has been removed from the model.

If the same problem should be solved with a *RSM* model like this:

$$\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_1 x_1 + \theta_2 x_2 + x_3 \tag{2.44}$$

where the free parameter θ_3 is removed from the model, the R^2_{adj} value reduces to 0.93 and the estimated coefficient values are: $\theta_1 = 4.90$, $\theta_2 = 1.16$. The model tries to compensate the missing free parameter θ_3 with the remaining two. The different models are visualized in fig. 2.20. It can bee seen that the true function and the full linear model with the removed influence of x_3 fit perfectly together. The parameter-reduced linear model with no free parameter to remove the influence of x_3, shows some deviations to the true function.

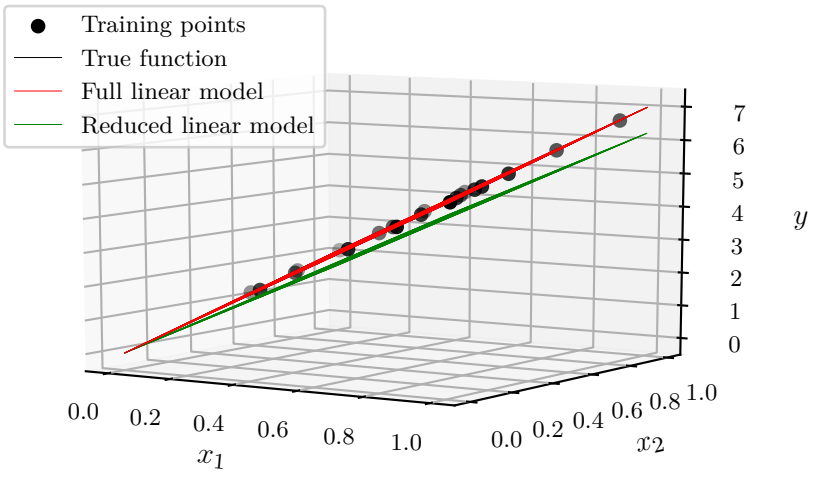


Figure 2.20: *RSM* models trained with a full linear model according to eq. 2.43 and a reduced model according to eq. 2.44 next to the true model function. x_3 is set to be 0 for this plot.

The removal of unimportant parameters in the polynomial model had a direct influence on the prediction of the model. For models like the Gaussian process, the influence is given by calculating the distances from training to test points (as already mentioned in section 2.4 and see section 2.8 for details). As shown in eq. 2.21 the difference in the distances between the points becomes more and more similar as the number of dimensions increases. Here the possibility of parameter reduction helps to minimize this effect. So that these models can still be applied to many dimensions. However, it does not completely avoid the problem if a very large number of input parameters are really relevant for an accurate model. This also becomes relevant for the moving least squares model in section 2.7 and the newly developed anisotropic extension.

Important notes:

- *ARD* is known as a Bayesian method for feature selection and sparse learning because it finds the relevance of features by optimizing the model marginal likelihood.
- It helps a model to reduce or even completely neglect unimportant parameters in order to reduce the model loss function.
- It also helps to minimize the effect of the curse of dimensionality for distance-based models like the Gaussian process or the moving least squares.
- Typically, the control of the importance of a parameter is done by adding a additional free parameter to the model which will be shown in section 2.7 for the moving least squares and in section 2.8 for the Gaussian process.

2.7 MOVING LEAST SQUARES

Moving least squares (MLS) (Levin [1998]), also known as weighted least squares, is a very simple but powerful supervised regression method. It is an extension of the RSM, whereas, as the name suggests, multiple local RSM models are built in an area around the point to predict. This allows the building of much more flexible models even with a low number of regressors (typically first or second order) compared to the RSM (Lancaster and Salkauskas [1981]). The basic idea behind this method is to modify the free parameters $\boldsymbol{\theta}$ of the RSM via a distance-based weighting matrix multiplication which gives a high weight to training points of X near the point to predict $\boldsymbol{x}_*$ and decreasing weights to points with an increasing Euclidean distance to this point. So that eq. 2.29 changes to

$$\boldsymbol{\theta} = \left(P^T W P\right)^{-1} P^T W \boldsymbol{y} \tag{2.45}$$

$W = W(X, \boldsymbol{x}_*)$ denotes the $N \times N$ weighting diagonal matrix $\text{diag}(\boldsymbol{w})$ for the prediction point $\boldsymbol{x}_*$ and is the main mathematical difference to the standard RSM. The vector $\boldsymbol{w}$ contains the weights w_i (diagonal elements) for each element of X regarding their Euclidean distance to the point to predict $\boldsymbol{x}_*$:

$$W = \begin{Bmatrix} w_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_N \end{Bmatrix}$$

This weighting matrix must be rebuilt for every point to predict which makes it numerically very inefficient in predicting and training (if for example trained via CV). Nevertheless, the MLS method also has strengths which makes it sometimes even more sophisticated then the Gaussian process model, as Kim et al. [2009] shows for non-convex functions. Especially in arbitrarily spaced data (Farwig [1986]) or in 3D surface reconstruction presented in Ahn et al. [2005], Fleishman et al. [2005] and Chang [2008], the MLS model shows good performance. Compared to methods like Gaussian processes, the MLS model has the advantage that it does not experience difficulties with numerical instabilities (singular covariance matrices, see section 2.8), if there are many dense design points to train on. Furthermore, if the training points show different local behavior such as clusters, MLS performs well. The main obstacle for local regression methods like the MLS is to determine the points to consider for the local approximation, especially if the density of the training points have a large variance in the design space. A possible solution to this problem which can be applied for every distance-based machine learning algorithm, is presented in section 2.10. To estimate these weights W, there are different types of weighting functions. Some of them can even force the MLS to interpolate through the training points as presented in Maisuradze et al. [2003]. Typically, a Gaussian weighting function is used as proposed in Levin [1998] to calculate one diagonal element w_i of $\mathrm{diag}(\boldsymbol{w})$ with

$$w_i(\boldsymbol{x}_i, \boldsymbol{x}_*, r) = \exp\left(-\left(\frac{\|\boldsymbol{x}_* - \boldsymbol{x}_i\|}{r\alpha}\right)^2\right) \tag{2.46}$$

with the constant

$$\alpha = \frac{1}{\sqrt{-\ln(0.001)}}$$

In this formulation the $\|\boldsymbol{x}_* - \boldsymbol{x}_i\|$ stands for the Euclidean distance calculated over all input parameter dimensions n between the prediction point $\boldsymbol{x}_*$ and a training point $\boldsymbol{x}_i$. There is one additional free model parameter r which represents the influence radius around the prediction point $\boldsymbol{x}_*$ and controls the weight value w_i dependent on the distance to the training point $\boldsymbol{x}_i$. As shown in fig. 2.21 the weight tends towards zero if the training points lie outside of the radius r.

To estimate the free model parameter r different kinds of possibilities are available. A straightforward solution would be to use an optimizer in combination with a stratified k-fold cross validation. The objective function can be one of the proposed metrics of subsection 2.2.1, e.g. the $\hat{R}^2$ metric (see. eq. 2.3). After W is obtained, the parameters $\boldsymbol{\theta}$ can still be estimated via MLE as for the RSM (see section 2.5).

As already mentioned, the MLS approach calculates the Euclidean distances over all input parameter dimensions n to estimate the weights w_i which are controlled by a single parameter r. Thus, all input parameters influence the weight estimation equally, hence the standard MLS method is an isotropic model (regarding the weight estimation, the local polynomial is still an anisotropic model, since it has unique coefficients for each parameter

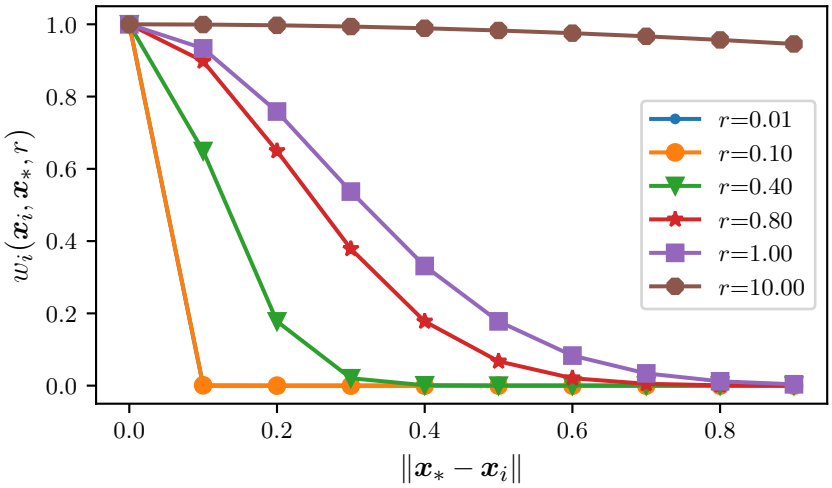


Figure 2.21: Gaussian weighting function depending on r and scaled distance $\|\boldsymbol{x}_* - \boldsymbol{x}_i\|, \in^{[0,1]}$.

like the RSM). This makes the MLS method without a separate preliminary sensitivity analysis a poor choice if the number of input parameters n is high and the training sample count N is low (see also section 2.4 and 2.6). Furthermore, if the underlying true function is very complex, one single parameter r might deliver unsatisfactory results. To overcome this problem, a newly developed extension to an anisotropic MLS ($AMLS$) is presented in the following.

2.7.1 *Anisotropic Moving Least Squares*

To give a machine learning algorithm the ability to reduce the influence of unimportant parameters, it is necessary to introduce further free parameters which can control the influence of each single input parameter in the model (as illustrated in section 2.6). Therefore, typically one additional free parameter per input parameter is necessary which is for example the case for the RSM, Gaussian processes (Snelson and Ghahramani [2006]) or neural networks (MacKay [1992], Neal [2012]) (see sections 2.8 and 2.9). In the following a new developed extension for the MLS model will be presented that allows it also to reduce the influence of unimportant parameters.

In the standard MLS formulation, the Euclidean distance is calculated over all input parameter dimensions n (see eq. 2.46). Thus, only one radius r is needed to control the corresponding weight calculation. A possible solution to extend the MLS method to an anisotropic model and reduce the influence of unimportant parameters for the weight estimation is to calculate the distances over all input parameter dimensions n separately. Therefore, it is necessary to introduce one free parameter r_j with $j \in \mathbb{Z}^{[1,2,\ldots,n]}$ per input parameter (in total n) which finally leads to the opportunity to calculate a weight vector $\boldsymbol{w}_j$ for each input parameter separately. This changes eq. 2.46 to estimate a single weight $w_{i,j}$ of a weight vector $\boldsymbol{w}_j$ to

$$w_{i,j}(x_{i,j}, x_{*,j}, r_j) = \exp\left(-\left(\frac{(x_{*,j} - x_{i,j})^2}{r_j \alpha}\right)^2\right) \tag{2.47}$$

To combine these weight vectors $\boldsymbol{w}_j$ into one weight diagonal matrix W in order to still be able to use eq. 2.45, the geometric mean overall n weight vectors $\boldsymbol{w}_j$ is calculated (element wise) via

$$W = \text{diag}\left(\sqrt[n]{\prod_{j=1}^{n} \boldsymbol{w}_j}\right) \tag{2.48}$$

As the geometric mean is used to combine the weight vectors $\boldsymbol{w}_j$, the parameters r_j of input parameters with a minor or no influence on the output $\boldsymbol{y}$ will tend to

$$r_j \to \infty$$

and $r_j \to \infty$ leads to a corresponding weight vector with $\boldsymbol{w}_j = \mathbf{1}$, meaning that all training points are considered with a weight of 1 for the non-important parameters (see fig. 2.21 with $r = 10$). Whereas input parameters with high influence will tend to

$$r_j \to 0$$

and so the weight vector $\boldsymbol{w}_j$ tends to have very small weights $w_{i,j}$ for the training points in the vicinity of the prediction point $x_{*,j}$. This means that only the r_j of the important input parameters have a significant influence on the diagonal weight matrix W. Even so, with this anisotropic approach of the MLS method, it is not necessary to scale the input parameters to equal ranges as is necessary for the isotropic MLS, but it is still recommended to perform the scaling for numerical stability. To estimate the free model parameters r_j, the Powell optimization algorithm is used in this work (Powell [1964]). This algorithm is especially suitable for this kind of optimization problem, because it does not require any derivative of the cost function to be minimized, only a start vector is needed. A line search is performed for each optimization variable separately which makes it effective in identifying unimportant parameters. The line search can, for example, be made using the Brent's method (Brent [1971]).

2.7.2 *Numerical Examples*

To demonstrate the advantage of the $AMLS$ method, numerical examples comparing the isotropic MLS method to the anisotropic version are given in the following. For the comparison, the same Powell optimization algorithm is used for both algorithms. The parameters r_j are limited between $[0, 20]$ which is high enough to ensure weights of 1 for all training points if the input parameters are scaled to $[0, 1]$ for the used amount of variables in the test functions. The optimization criterion is the already introduced $\hat{R}^2$ (eq. 2.3) evaluated via stratified k-fold CV. To train the surrogate models, $N = 50$ (test function 2.49, 2.50), $N = 150$ (test function 2.51) and $N = 100$ (test function 2.52) uniformly distributed training samples are created, via optimized Latin hypercube sampling (see section 3.3). Furthermore, each test function is approximated with a first order polynomial basis. The k-fold CV is done with $k = 10$, with the same package structure (same designs in each

package) for both methods. For the comparisons, four test functions assist in demonstrating how the $AMLS$ works and how the free parameters r_j behave. The test functions are defined as

$$y = x_1 + x_2 + x_3 + x_4 + x_5,\ x_1, ..., x_5 \in \mathbb{R}^{[0,1]} \tag{2.49}$$

$$y = x_1^6 + x_2^4 + x_3^2 + x_4 + 0.001x_5,\ x_1, ..., x_5 \in \mathbb{R}^{[0,1]} \tag{2.50}$$

$$y = x_1^4 x_2^2 + 0.01x_3x_4 + 0.001x_5,\ x_1, ..., x_5 \in \mathbb{R}^{[-\pi,\pi]} \tag{2.51}$$

$$y = 0.5x_1 + x_2 + 0.5x_1x_2 + 5sin(x_3) + 0.2x_4 + 0.1x_5,\ x_1, ..., x_5 \mathbb{R}^{\in[-\pi,\pi]} \tag{2.52}$$

These test functions intentionally include some parameters with a low influence to show that the r_j values of the $AMLS$ can identify them. In eq. 2.49 all variables have the same influence on the output y and should show equal results regarding the error metric for the isotropic MLS and $AMLS$, because it is a linear function. Furthermore, all r_j of the $AMLS$ should also show an equal value and so an equal weighting of the input variables. Eq. 2.50 includes decreasing influence from x_1, the highest, to x_5, the lowest. Here the intention is that the free parameters r_j should also show an increasing value for the corresponding input parameters. The variance contribution to y of each variable is different and so the new capabilities of the anisotropic version should result in a higher $\hat{R}^2$ value for the $AMLS$. Eq. 2.51 contains the influence of cross terms with a high variance in one corner of the data range and eq. 2.52 has one dominant variable x_3 and four less important terms. The test functions 2.49 to 2.52 are illustrated in fig. 2.22 to fig. 2.25.

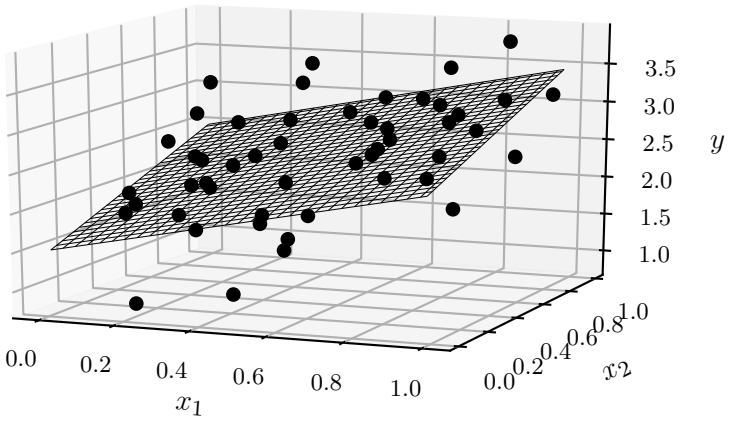


Figure 2.22: Test function 1, $N = 50$, $x1, ..., x5 \in \mathbb{R}^{[0,1]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data.

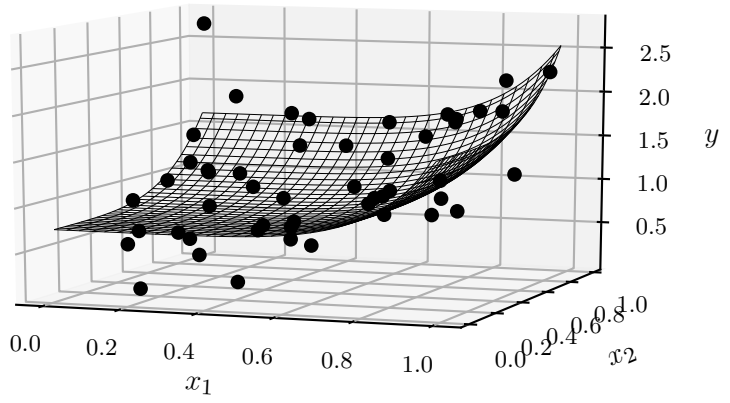


Figure 2.23: Test function 2, $N = 50$, $x1, ..., x5 \in \mathbb{R}^{[0,1]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data.

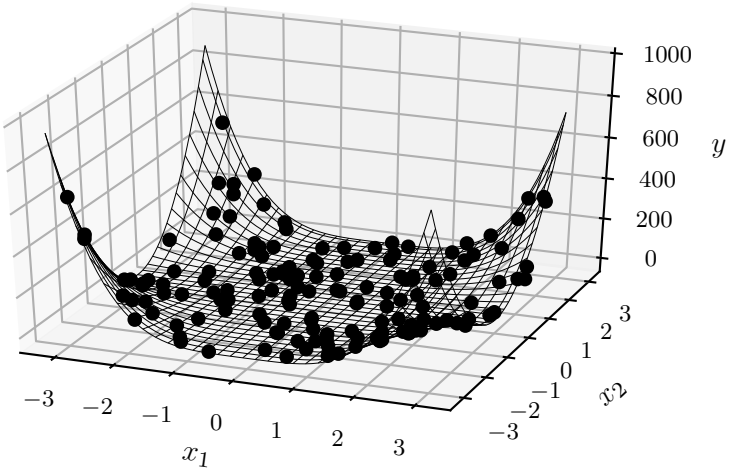


Figure 2.24: Test function 3, $N = 150$, $x1, ..., x5 \in \mathbb{R}^{[-\pi,\pi]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data.

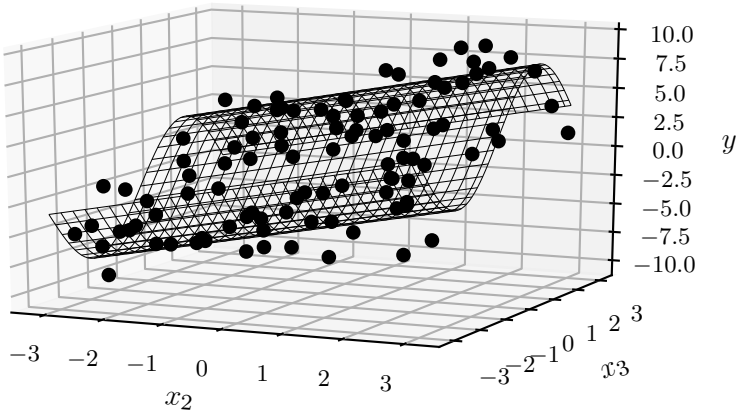


Figure 2.25: Test function 4, $N = 100$, $x1, ..., x5 \in \mathbb{R}^{[-\pi,\pi]}$, black points illustrate training points. All variables not shown are set constant to their mean value estimated on the training data.

Tab. 2.3 shows the results for the free parameters of the MLS r_{iso} and for the $AMLS$ $r_1, ..., r_5$. Additionally, the predictive coefficient of determination $\hat{R}^2_{iso}$ and $\hat{R}^2_{aniso}$ for both methods are given.

For the first test function (eq. 2.49), the parameters r_j of the $AMLS$ are all equal as expected and higher than the parameter range of $[0, 1]$ which means all points are considered in the weighting and there is no local regression effect; in other words, it works like a global RSM model. The same applies to the MLS method for this specific example with parameter ranges of $[0, 1]$ with the maximum Euclidean distance of $\sqrt{\sum_1^{n=5} 1^2} = \sqrt{5}$. The prognosis quality of both methods is maximal with $\hat{R}^2_{iso} = \hat{R}^2_{aniso} = 1$.

The results for the second test function eq. 2.50 show the first difference between the MLS and the $AMLS$ method regarding the prognosis quality $\hat{R}^2$ of 16%. The parameters r_j of the $AMLS$ are very low for the most important parameters x_1, x_2, x_3 and at maximum for the unimportant parameters x_4, x_5, so that their weighting does not influence the overall weighting in the geometric mean described in eq. 2.48. The values of the important parameters suggest that the approximation works locally. Additionally, there is a ranking of the parameters x_1, x_2, x_3 corresponding to the real ranking in the test function.

The third test function eq. 2.51 shows the highest difference between $\hat{R}^2_{iso}$ and $\hat{R}^2_{aniso}$ of all test functions. Here the $AMLS$ seems to use the x_1, x_2, x_3 parameters with a focus on the first two parameters x_1, x_2 as important parameters for the overall weighting, corresponding to the test function part with $x_1^4 x_2^2$ which clearly dominate the values of the test function. The MLS is not capable of approximating this function with one single free parameter $r = 8.99$ at all.

The last test function eq. 2.52 yields a difference for the $\hat{R}^2$ values of 42%. The dominant parameter x_3 has been identified in the $AMLS$ which results in the lowest parameter value $r_3 = 0.59$. x_1, x_2 also have a small influence and x_4, x_5 have been identified as unimportant. This ranking also follows the fourth test function parameter ranking. The MLS approach has difficulties to approximate this test function appropriately.

TF	r_{iso}	r_1	r_2	r_3	r_4	r_5	$\hat{R}^2_{iso}$	$\hat{R}^2_{aniso}$
1	12.58	2.91	2.91	2.91	2.91	2.91	**1.0**	**1.0**
2	2.28	0.16	0.25	0.63	20	20	0.75	**0.91**
3	8.99	0.66	0.81	20	2.6	20	0.00	**0.70**
4	5.75	2.54	2.49	0.59	20	20	0.54	**0.96**

Table 2.3: Overview of parameters r_j and prognosis measurements $\hat{R}^2_{iso}$, $\hat{R}^2_{aniso}$ for test functions (TF) 2.49 to 2.52.

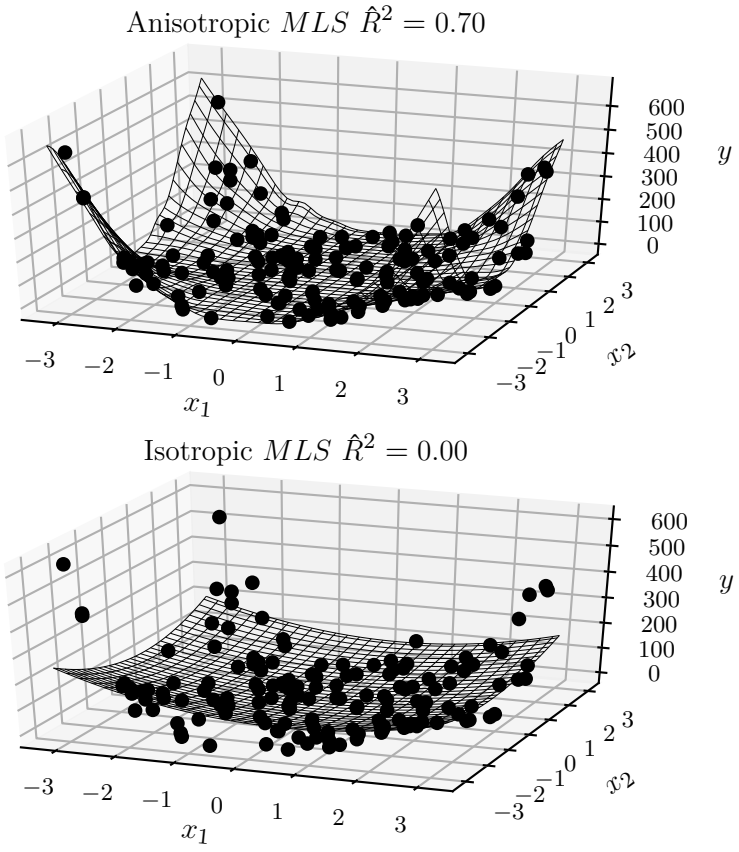


Figure 2.26: Surrogate model comparison (top $AMLS$, bottom MLS) for test function 3, 3D view for parameter x_1, x_2. All variables not shown are set constant to their mean value estimated on the training data.

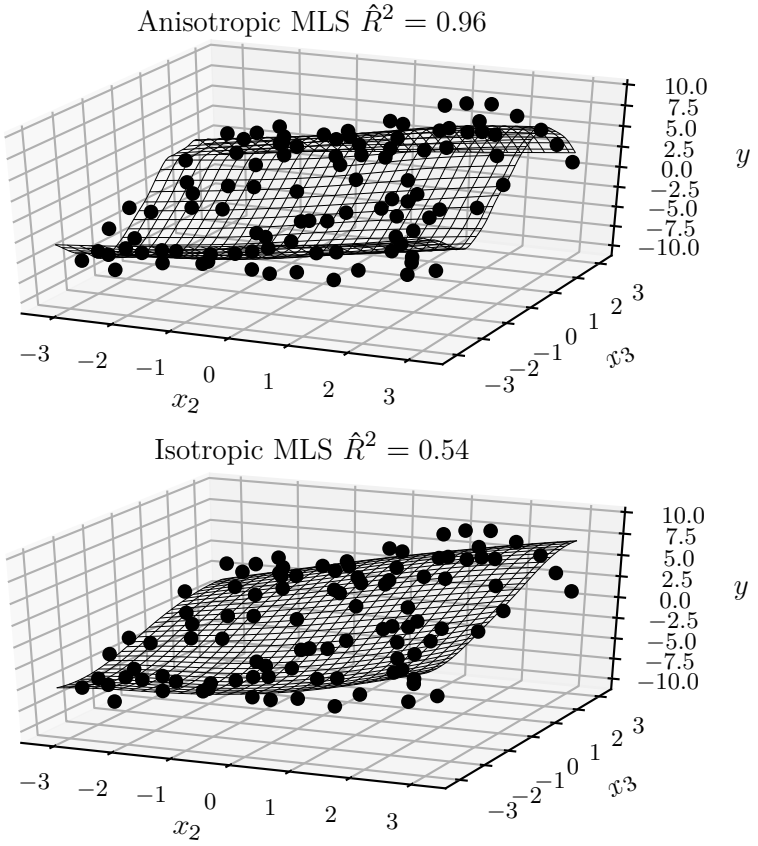


Figure 2.27: Surrogate model comparison (top $AMLS$, bottom MLS) for test function 4, 3D view for parameter x_2, x_3. All variables not shown are set constant to their mean value estimated on the training data.

In fig. 2.26 and 2.27 the 3D view of the surrogate models for the most important parameters of test function 3 and 4 are presented. In both pictures, it can be seen that the trend is captured much better by the $AMLS$ leading to the higher $\hat{R}^2$, especially for x_3 of test function 4, where it captured the $sin()$ term compared to the MLS which seems to average through the data points.

The numerical examples show that the $AMLS$ method is more accurate than the MLS method for the test functions with input parameters of different importance (eq. 2.50 to eq. 2.52). Furthermore, large differences can be seen in the test function 3 with high multivariate dependencies (eq. 2.51). This is accomplished by setting the free model parameter r_j corresponding to the influence of the input parameters. Input parameters with a high influence result in lower values for r_j and vice versa. To find the right value for r_j, a good start solution vector for the optimizer might be helpful. This can be obtained by previously performing a sensitivity analysis (see chapter 4) and set the r_j values accordingly. Otherwise, it is recommended that the mean values of the input parameters based on the training data for each r_j are used as a starting point. Additionally, if the input parameters are of different magnitudes, it is recommended that they are normalized or scaled. If no preliminary sensitivity analysis has been performed, the $AMLS$ parameters can also work as a simple kind of sensitivity analysis. Unimportant parameters can be identified

through high values for the r_j parameters (similar to high correlation lengths of Gaussian processes, see section 2.8). It also delivers a ranking of importance with the smaller r_j value corresponding to the most important parameters.

2.7.3 *Heteroscedasticity*

As already introduced in section 2.3 the problem of heteroscedasticity of the residuals for some machine learning methods exists, particularly for the RSM. The MLS method can reduce this drawback of the RSM through its local adaption to the data. Fig. 2.28 shows the same example as already shown in fig. 2.18 (section 2.3) with a MLS fitted to the data. Since the MLS is able to fit the data nearly perfectly, it is clear that the residuals are zero. Therefore, fig. 2.29 shows the residual vs. prediction plot for test function 2 and 3 of the $AMLS$ model, where the prognosis quality is not perfect. As shown for test function 2, there is no real variance of the residual but test function 3 shows a much higher residual for high values of y. As explained in section 2.3, one common reason for the heteroscedasticity of the residuals is the range between the smallest and largest observations value which is very high for test function 3. So, the MLS / $AMLS$ method is capable of reducing the heteroscedasticity of the residuals but also only if the model fits the data well.

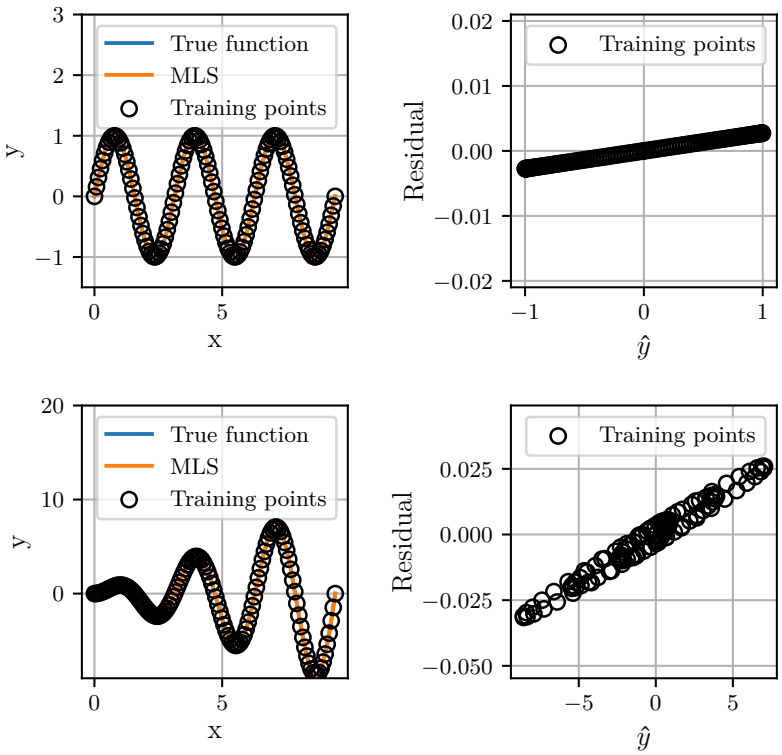


Figure 2.28: Two MLS trained on a sinusoidal function with constant (top) and dynamic (bottom) amplitude. Furthermore, the residual vs. prediction is plotted to show the homoscedasticity and heteroscedasticity of the error.

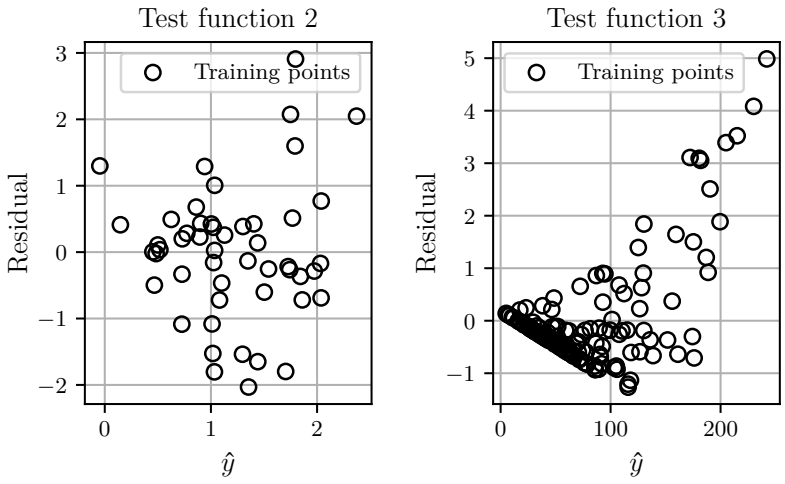


Figure 2.29: The residual vs. prediction plot for test function 2 (left) and 3 (right) of the $AMLS$ model.

Important notes:

- The moving least square method is a smart extension to the RSM which allows the building of local RSM models which makes this type of model much more flexible, even with a low order of regressors.
- An additional parameter is necessary to describe the local neighborhood around the point to predict and to build the local RSM model for.
- Since it is always necessary to calculate the local neighborhood, the algorithm is numerically not very efficient and can be very slow if N is high.
- A novel anisotropic extension was presented ($AMLS$) which allows the MLS model to reduce the influence of unimportant parameters in the definition of the local neighborhood. Note that the RSM and its free parameters itself are also capable of reducing the influence of unimportant parameters. This extension refers to the estimation of the local neighborhood which can be crucial as shown in the numerical examples.
- Due to the local nature of the approximation of the MLS / $AMLS$ methods, it is strongly recommended not to use these methods for extrapolation.
- Based on experience, the use of the $AMLS$ model is suggested if the amount of the training samples is $N <= 150$ (due to training time) and if the problem complexity is too high for a RSM. Similar to the MLS, it is not suitable for big data applications without dividing the data into smaller subgroups (Merry et al. [2013]).
- Nevertheless, compared to the model approaches presented later, benchmarks as presented in Bogoclu and Roos [2016], show that the MLS / $AMLS$ method is on average less accurate then Gaussian processes or neural networks.

2.8 GAUSSIAN PROCESSES

In statistics, a Gaussian process ($\mathcal{GP}$) (named after Carl Friedrich Gauss) is a stochastic process in which every finite subset of random variables has a multivariate Gaussian distribution. For supervised regression, the Gaussian process is a Gaussian probability distribution of the functional values (outputs) over the continuous domain of the input space. At the limit of infinitely many samples, it forms an infinite-dimensional Gaussian distribution whose samples are functions. Therefore, the Gaussian process can be seen as a distribution over functions whose mean is used for prediction. Another more numerical point of view would be that the $\mathcal{GP}$ measures the similarity between the training and test points with the help of a covariance function to estimate the values for unknown observations. The idea of the Gaussian process is used in many other algorithms. For example, the Wiener process, (also known as Brownian motion, Hitsuda [1968]) is the integral of a white noise Gaussian process which was already used in the 1940s. Around the 1970s, Gaussian processes were used in Geostatistics to estimate the most likely distribution of gold, based on samples from a few boreholes. The theoretical basis for this method was developed by the French mathematician Georges Matheron in 1960 (Matheron [1963]), based on the Master's thesis of Danie G. Krige (Krige [1951]), the pioneering plotter of distance-weighted average gold grades at the Witwatersrand reef complex in South Africa. Therefore, the Gaussian process is also often named as Kriging, especially if it is used for regression tasks. After this, in the 1990s, it became a general tool for spatial statistics (Ver Hoef and Cressie [1993]) and at the end of 1990s it became more and more interesting for machine learning purposes, especially for regression and classification tasks (Williams and Rasmussen [1996], Neal [1997]). Gaussian processes provide an alternative approach compared to the RSM to supervised regression problems with some advantages which will be explained in the following.

As already introduced in section 2.5 an approximation model in the form of (repetition of eq. 2.23)

$$y = \hat{f}(x) + \hat{\epsilon} \tag{2.53}$$

should be built. In the case of the RSM, the number of regressors (e.g. linear or quadratic) need to be defined a priori and thus also the number of free parameters $\boldsymbol{\theta}$ which make this kind of model a so-called parametric model. If the wrong assumptions are made, the model might be not adequate to represent the data. The Gaussian process instead does not need any model assumptions of this kind a priori and belongs to the group of non-parametric models (it will be shown that some assumptions are possible to increase the model accuracy). Every observation in $\boldsymbol{y} = [y_1, ..., y_N]^T$ can be imagined as a single point sampled from a multivariate (N-variate) Gaussian distribution. Therefore, in contrast to the RSM, the $\mathcal{GP}$ model is a non-parametric model in which, if no data is observed, all functions $\hat{f}$ are possible[1]. Thus, the $\mathcal{GP}$ can also be seen as a model which extends a multivariate Gaussian distribution to infinite dimensionality with infinitely many parameters (in practice it is restricted by the number of observations N). Furthermore, in contrast to the RSM model trained via MLE, the $\mathcal{GP}$ approach tries not to find distributions over parameters (as introduced in subsection 2.2.2) but over possible functions $\hat{f}$ which fit the observed output $\boldsymbol{y}$ and fulfill the prior assumption for the mean and covariance matrix. For the prediction of the $\mathcal{GP}$ model, the mean of the distribution of possible functions is taken. As with all

1 All functions which fulfill the assumptions for the mean and covariance.

Bayesian methods, it begins with a prior distribution and gets updated as data points are observed, producing the posterior distribution over functions. As a side note; a neural network with one hidden layer with an infinite number of neurons is equal to a $\mathcal{GP}$ (see Neal [2012]) which motivated the idea of replacing supervised neural networks by $\mathcal{GP}$ models, presented in many publications (Williams and Rasmussen [1996], MacKay [1997], Rasmussen [1999]). In the following, the model error $\hat{\epsilon}$ is initially left out and later in subsection 2.8.2, there is an explanation of how to introduce noise to the $\mathcal{GP}$ model.

2.8.1 *Noise-free Gaussian Processes*

To gain a better understanding of the following explanations, a short revision of the multivariate Gaussian distribution will be given. The multivariate Gaussian distribution for a random vector $\boldsymbol{x}$ has a joint probability distribution defined by

$$p\left(\boldsymbol{x}|\boldsymbol{\mu}, K\right) = \frac{1}{\sqrt{(2\pi)^N |K|}} \exp\left(-0.5(\boldsymbol{x}-\boldsymbol{\mu})^T K^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) \tag{2.54}$$

where $\boldsymbol{\mu}$ is the mean vector of length N and K is the (symmetric, positive definite) covariance matrix of size $N \times N$. A short notation would be $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, K)$.
The corresponding log-likelihood function is (Williams and Rasmussen [1996])

$$\ln\left(p\left(\boldsymbol{x}|\boldsymbol{\mu}, K\right)\right) = -0.5\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^T K^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right) - 0.5\ln\left(|K|\right) - \frac{N}{2}\ln(2\pi) \tag{2.55}$$

Furthermore, if $\boldsymbol{x}$ were to be decomposed into two consecutive sub vectors $\boldsymbol{a}, \boldsymbol{b}$, the joint Gaussian distribution can be written as

$$\begin{bmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_a\\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}A & C\\ C^T & B\end{bmatrix}\right)$$

If all off-diagonal elements of the decomposed covariance matrix

$$K = \begin{bmatrix}A & C\\ C^T & B\end{bmatrix}$$

of the joint Gaussian distribution are zero. Then $\boldsymbol{a}$ and $\boldsymbol{b}$ are independent of each other and knowing $\boldsymbol{b}$ would give no information about $\boldsymbol{a}$ and vice versa. Otherwise, if the off-diagonal elements are non-zero then both relate to each other. Fig. 2.30 demonstrates this for the more simple case that the random vector $\boldsymbol{x}$ contains only two variables (bivariate) a, b. The top part shows a covariance matrix with

$$K = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

thus a variance of 1 for each variable to itself on the diagonal and no covariance between the two variables. Seen from above, the isolines of the bell shaped function in the top part of fig. 2.30 would be perfect circles, this means that these two variables a, b have an independent Gaussian distribution. The bottom part of this figure shows the case where the covariance matrix is given as

$$K = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

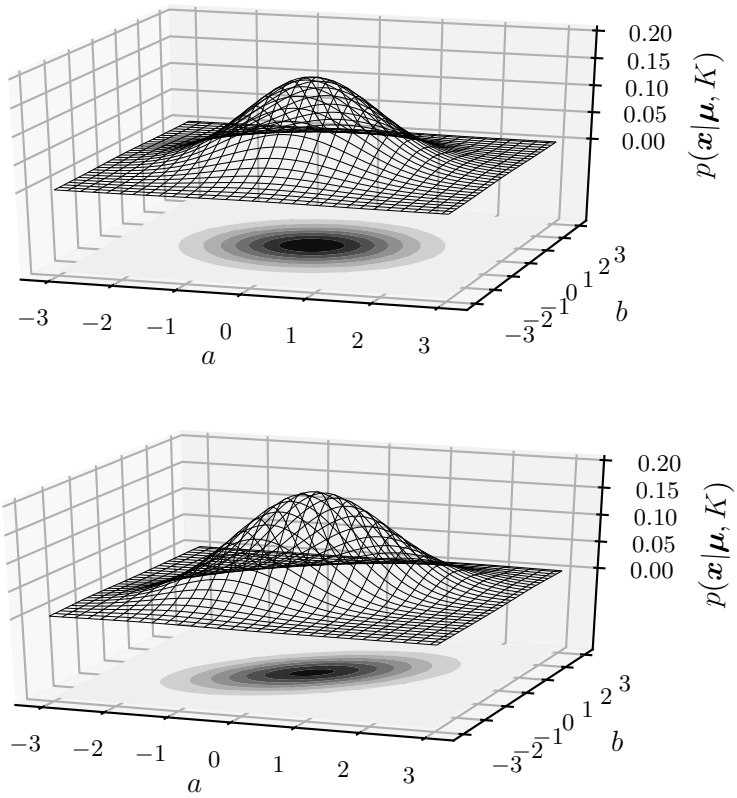


Figure 2.30: Top: independent bivariate Gaussian distribution. Bottom: dependent bivariate Gaussian distribution.

As depicted in the bottom part of fig. 2.30, the shape of the bell is now more oval, and the two variables are dependent bivariate Gaussian distributed. Therefore, if the off-diagonal elements are non-zero, with the help of Bayes' theorem (Bayes and Price [1763])

$$p(\boldsymbol{a}|\boldsymbol{b}) = \frac{p(\boldsymbol{a})p(\boldsymbol{b}|\boldsymbol{a})}{p(\boldsymbol{b})} = \frac{p(\boldsymbol{b},\boldsymbol{a})}{p(\boldsymbol{a})} \tag{2.56}$$

the distribution of $\boldsymbol{a}$ and the conditional distribution $\boldsymbol{a}$ given $\boldsymbol{b}$ can be written as (Von Mises [2014])

$$\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}_a, A) \tag{2.57}$$

and

$$\boldsymbol{a}|\boldsymbol{b} \sim \mathcal{N}\left(\boldsymbol{\mu}_a + CB^{-1}(\boldsymbol{b} - \boldsymbol{\mu}_b), A - CB^{-1}C^T\right) \tag{2.58}$$

The mean, $\boldsymbol{\mu}_a + CB^{-1}(\boldsymbol{b} - \boldsymbol{\mu}_b)$ is known as the matrix of regression coefficients. This means that if it is possible to formulate a conditional distribution $\boldsymbol{a}$ given $\boldsymbol{b}$, it can also be used

for the Gaussian process to formulate the conditional distribution for $\boldsymbol{y}_*$ given $\boldsymbol{y}$ (and the corresponding input X, X_*). By defining a prior and some observations given, it is possible to obtain a posterior, thus a $\mathcal{GP}$ which is able to predict new observations. The joint prior of the training outputs $\boldsymbol{y}$ and the unobserved outputs $\boldsymbol{y}_*$ of a $\mathcal{GP}$ with zero mean (which is typically assumed or $\boldsymbol{y}$ is normalized) is defined by

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) \tag{2.59}$$

If there are N training points and N_* test points then $K(X, X_*)$ denotes the $N \times N_*$ matrix of the covariances evaluated at all pairs of training and test points. The same holds for the other entries $K(X,X)$, $K(X_*,K_*)$ and $K(X_*,X)$. To get the posterior distribution over functions, this joint prior distribution needs to be restricted to contain only those functions which fit to the observed data points $\boldsymbol{y}$. This can be done with the help of eq. 2.58, so conditioning the joint Gaussian prior distribution of $\boldsymbol{y}$ and $\boldsymbol{y}_*$ on the observations $\boldsymbol{y}$ or in other words, the conditional distribution of $\boldsymbol{y}_*$ given X_*, X, $\boldsymbol{y}$ which yields (Rasmussen and Williams [2006])

$$\begin{aligned} \boldsymbol{y}_*|X_*, X, \boldsymbol{y} \sim &\mathcal{N}(K(X_*,X)K(X,X)^{-1}\boldsymbol{y}, \\ &K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)) \end{aligned} \tag{2.60}$$

To better understand the conditional distribution of $\boldsymbol{y}_*$ given X_*, X, $\boldsymbol{y}$, fig. 2.31 shows an example of the unconditioned prior and the prior conditioned on $N = 10$ and $N = 50$ noise-free observations of a $\mathcal{GP}$. It shows that the more observations are available to condition the prior, the less freedom is left in the posterior for a possible distribution of functions. The mean function (the black line) of all possible functions would be used for predictions and is typically assumed a priori to be zero with variance one (other assumptions are possible or normalization is used). As it can be seen, the mean function also changes with observing new data points.

Hence, a $\mathcal{GP}$ extends the multivariate Gaussian distribution to infinite dimensionality, thus it defines priors over functions which can be converted into a posterior over functions once some data has been observed (Murphy [2012]). As it seems to be difficult to define a distribution over functions, it turns out that only a distribution over the function's values at a finite set of points $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ needs to be defined. A $\mathcal{GP}$ assumes that the probability of these functions values $p(y_1, ..., y_N)$ is jointly Gaussian, with the mean function $\mu(\boldsymbol{x})$ and the covariance function (sometimes named kernel function) $k(\boldsymbol{x}, \boldsymbol{x}_*)$ which defines the elements of the covariance matrix K. Therefore, a Gaussian process is completely specified by its mean and covariance function with the resulting covariance matrix. Defining the mean function $\mu(\boldsymbol{x})$ and the covariance function $k(\boldsymbol{x}, \boldsymbol{x}_*)$ of a Gaussian process model $\hat{f}(\boldsymbol{x})$ as

$$\mu(\boldsymbol{x}) = \mathbb{E}[\hat{f}(\boldsymbol{x})] \tag{2.61}$$

$$k(y, y_*) = k(\boldsymbol{x}, \boldsymbol{x}_*) = \mathbb{E}[(\hat{f}(\boldsymbol{x}) - \mu(\boldsymbol{x}))(\hat{f}(\boldsymbol{x}_*) - \mu(\boldsymbol{x}_*))] \tag{2.62}$$

then the Gaussian process can be written as

$$\hat{f}(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}_*)) \tag{2.63}$$

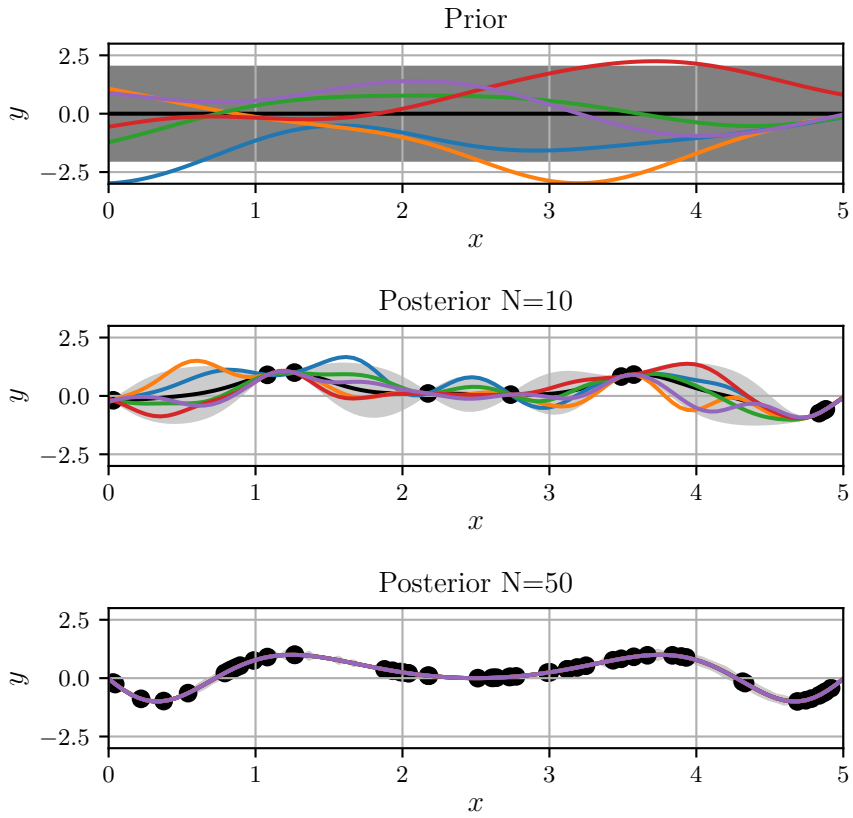


Figure 2.31: One dimensional $\mathcal{GP}$ model with zero mean and variance of one. Top: shows 5 functions drawn from a $\mathcal{GP}$ prior without any observations. Middle and bottom show 5 functions drawn from the $\mathcal{GP}$ posterior distribution over functions, so the prior conditioned on $N = 10$ and $N = 50$ noise-free observations. The shaded area illustrate the 95% confidence region of the $\mathcal{GP}$.

As already mentioned, it is typically assumed that $\mu(\boldsymbol{x})$ is zero so that the relationship between the observations $\boldsymbol{y}$, $\boldsymbol{y}_*$ is only described by the covariance function $k(\boldsymbol{x}, \boldsymbol{x}_*)$. Therefore, the observations need to be normalized before training by subtracting the mean. If the Gaussian process is used for predicting points outside the training design space (extrapolation), it will always tend to its mean value / function (e.g. zero). Therefore, if the observations were normalized, the rescaling of the normalized Gaussian process prediction will give the observations mean instead of zero. Additionally, it is common to divide the observation by its standard deviation which is called "standardization" so that the standard deviation / variance of the standardized Gaussian process is 1.0.

A popular choice of a covariance function is the squared exponential function with

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}_*) = \sigma_f^2 \exp\left(-0.5 \frac{\|\boldsymbol{x} - \boldsymbol{x}_*\|^2}{l^2}\right) \tag{2.64}$$

One single element of the covariance matrix K is based on the Euclidean distance $\tau = \|\boldsymbol{x} - \boldsymbol{x}_*\|$ between the training point $\boldsymbol{x}$ and the point to predict $\boldsymbol{x}_*$. Additionally, there are two free parameters, l (positive value) which is named the correlation length-scale and can be thought of the distance to move in input space before the function values y become

uncorrelated or, in other words, how many training observations of $\boldsymbol{y}$ are correlated to the test point value $\hat{y}$ which should be predicted. Note that this correlation is estimated in the covariance matrix over the input space, so between the training and test input $\boldsymbol{x}$, $\boldsymbol{x}_*$ and not over the output directly. This was also illustrated in fig. 2.15 of section 2.2. Therefore, the $\mathcal{GP}$ prediction can also be seen as a linear combination of the training observations and the weights of this linear combination are based on the resulting matrix of $K(X_*, X)K(X, X)^{-1}$ (note here for multiple points to predict, see eq. 2.60). σ_f^2 is called the signal variance and scales the maximum allowable covariance. It determines the variation of function values from their mean. A small value of σ_f^2 characterize functions that stay close to their mean value, larger values allow more variation. If the signal variance is too large, the model will be prone to outliers. As Heinonen et al. [2016] already mentioned, this parameter does not have a huge influence on the prediction quality, since it scales the maximum covariance to σ_f^2 times the diagonal elements of K which is in the noise-free case $\sigma_f^2 \cdot 1$ (due to normalization / prior assumption of variance equal one). Fig. 2.32 shows an example of a trained $\mathcal{GP}$ model with a squared exponential covariance function. The objective was to minimize the negative marginal log-likelihood[2] which will be introduced later in this section (see. subsection. 2.8.3). In this example the data is noise-free and the $\mathcal{GP}$ model fits the data well. In contrast to this, fig. 2.33 shows the same example for noisy training data. The $\mathcal{GP}$ model does not behave well between the observations. This can be improved by introducing an additional noise parameter to the $\mathcal{GP}$ model which will be explained in subsection 2.8.2. Additionally, in both pictures the optimal value for the signal variance is one.
It should be mentioned that the covariance between two outputs $k(y, y_*)$ (eq. 2.60) is given as a function of their corresponding inputs. The idea behind this is that similar points $\boldsymbol{x}, \boldsymbol{x}_*$ lead to similar outputs y, y_*. Therefore, the covariance function delivers a value of one if $\boldsymbol{x}$ and $\boldsymbol{x}_*$ are equal.

2.8.2 *Noisy Observations*

Until now, all equations only considered the noise-free $\mathcal{GP}$ model but in reality the training observations $\boldsymbol{y}$ are often noisy. To include the noise (assuming additive independent identically distributed Gaussian noise) in a $\mathcal{GP}$ model, the Gaussian prior (with zero mean) which only depends on the covariance function (eq. 2.64), changes to

$$k(y_i, y_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sigma_n^2 \delta_{ij} \text{ or } k(\boldsymbol{y}) = K(X, X) + \sigma_n^2 I \tag{2.65}$$

where δ_{ij} is the Kronecker delta which is one if i is equal j and zero otherwise. σ_n^2 is the noise variance and a further free parameter of the $\mathcal{GP}$ model. In practice, it describes the off-set of the model to its observations. If σ_n^2 is set to zero, the $\mathcal{GP}$ model fits its observations perfectly. Therefore, compared to the noise-free case, a diagonal matrix is added to the covariance matrix $K(X, X)$. With this additional parameter, the maximum covariance value scales to $\sigma_f^2 \cdot \sigma_n^2$. Also, introducing the noise to eq. 2.59 the joint distribution yields

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

2 Due to optimization reasons the negative log-likelihood is minimized, typically in literature the maximization of the non-negative log-likelihood is described.

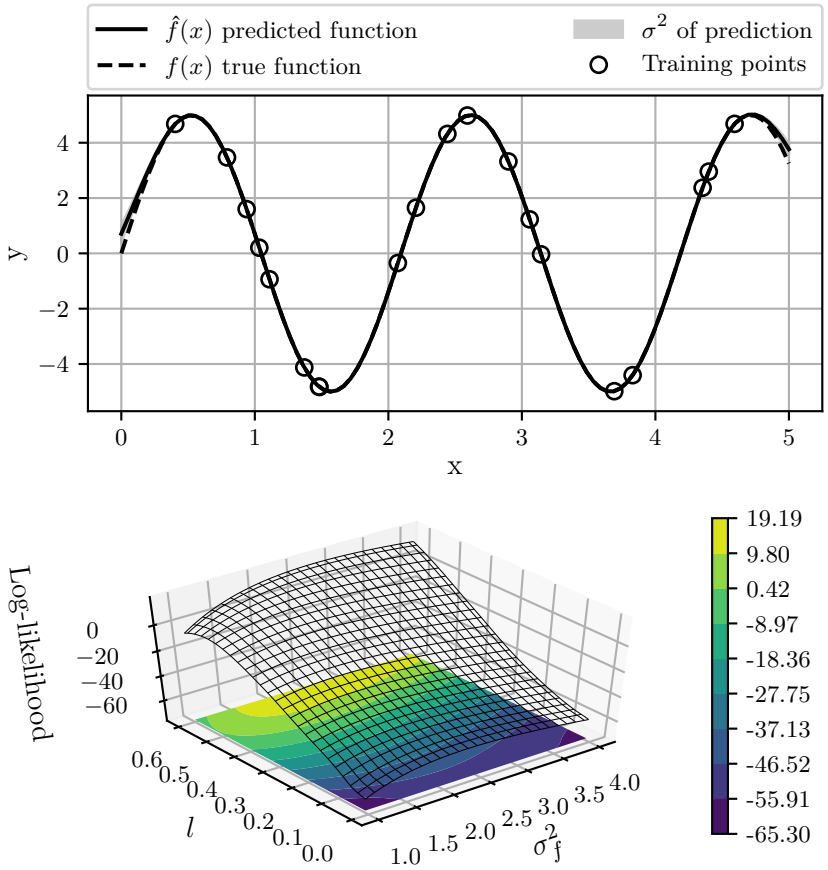


Figure 2.32: $\mathcal{GP}$ model with a squared exponential covariance function. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$. Number of training points $N = 20$. The top picture shows the trained model on the noise-free data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l which should be minimized in order to fit the data.

Deriving the conditional distribution corresponding to eq. 2.60 the key predictive equation for Gaussian process regression is

$$\begin{aligned} &p\left(\boldsymbol{y}_*|X_*, X, \boldsymbol{y}\right) \sim \mathcal{N}(\bar{y}_*, k(\boldsymbol{y}_*)), \text{ with} \\ &\bar{\boldsymbol{y}}_* \triangleq \mathbb{E}[\boldsymbol{y}_*|X_*, X, \boldsymbol{y}] = K(X_*, X)\left[K(X, X) + \sigma_n^2 I\right]^{-1}\boldsymbol{y}, \\ &k(\boldsymbol{y}_*) = K(X_*, X_*) - K(X_*, X)\left[K(X, X) + \sigma_n^2 I\right]^{-1}K(X, X_*) \end{aligned} \tag{2.66}$$

which was also already shown in eq. 2.60 for the noise-free case. In a more compact form with $K = K(X, X)$, $K_* = K(X, X_*)$, a single test point $\boldsymbol{x}_*$ and the vector of covariances between the test point and the N training points $\boldsymbol{k}_* = k(\boldsymbol{x}_*, X)$, eq. 2.66 can be written as[3]

$$\bar{y}_* = \hat{y}_* = \boldsymbol{k}_*\left(K + \sigma_n^2 I\right)^{-1}\boldsymbol{y} \tag{2.67}$$

and for the predictive variance:

3 Note that it is a single test point, therefore the mean and variance is a scalar value.

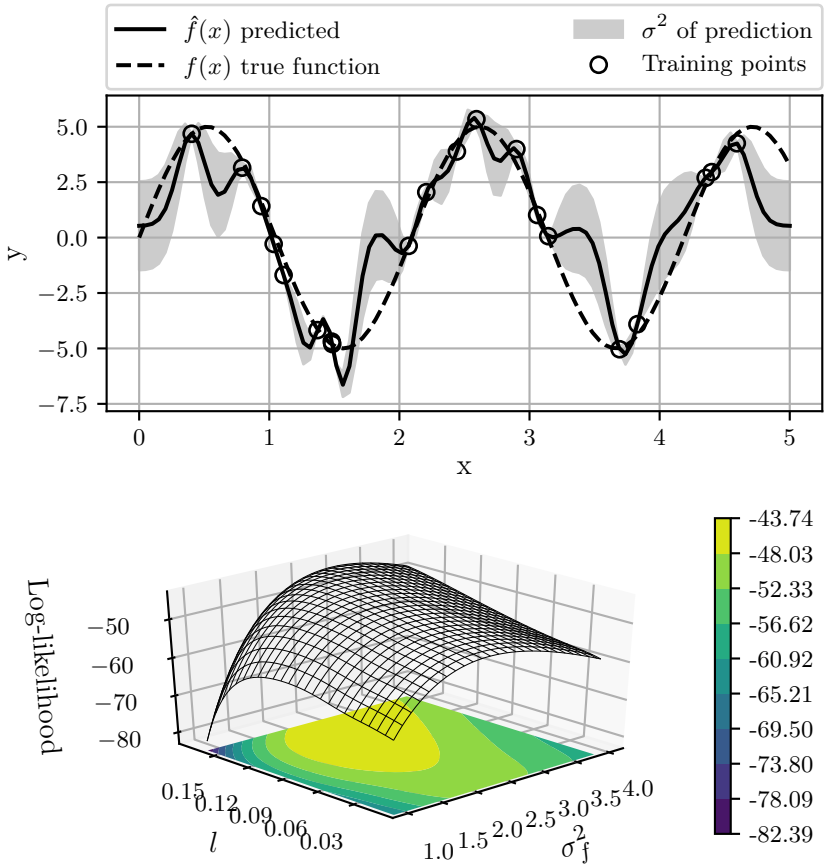


Figure 2.33: $\mathcal{GP}$ model with a squared exponential covariance function. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. The number of training points $N = 20$. The top picture shows the trained model on the noisy data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l which should be minimized in order to fit the data.

$$\mathbb{V}[y_*] = \mathbb{V}\left[\hat{y}_*\right] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T \left(K + \sigma_n^2 I\right)^{-1} \boldsymbol{k}_* \tag{2.68}$$

Eq. 2.67 gives the prediction mean $\bar{y}_*$ for a test point $\boldsymbol{x}_*$ and is used as the general prediction result $\hat{y}_*$ of a $\mathcal{GP}$ model. Therefore, the notation $\hat{y}_*$ for unknown test points and $\hat{y}$ for known points is further used. Eq. 2.68 is the variance of the prediction and can also be used to estimate the prediction confidence interval via

$$\begin{aligned} \hat{y}_{*CI} &= \hat{y}_* \pm z\left(\frac{\alpha}{2}\right)\sqrt{\mathbb{V}\left[\hat{y}_*\right]} \\ &= \boldsymbol{k}_*(K + \sigma_n^2 I)^{-1}\boldsymbol{y} \pm \left(z\left(\frac{\alpha}{2}\right)\sqrt{k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{k}_*}\right) \end{aligned} \tag{2.69}$$

$z\left(\frac{\alpha}{2}\right)$ denotes the critical value (z-score) for the standard Gaussian distribution, since the Gaussian process is assumed to be a multivariate Gaussian distribution with known mean and standard deviation. As already mentioned, another point of view on the mean prediction equation 2.67 is that it is a linear combination of the N covariance function values, each one centered on a training point $\boldsymbol{x}_i$ with $\boldsymbol{\beta} = (K + \sigma_n^2 I)^{-1}\boldsymbol{y}$:

$$\hat{y}_* = \sum_{i=1}^{N} \beta_i \boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_*) \tag{2.70}$$

This means the $\mathcal{GP}$ model is represented by N basis functions / regressors if $N \to \infty$ the $\mathcal{GP}$ model has infinite regressors as stated at the beginning of this section. Also worthy of note is that the variance in eq. 2.68 does not depend on the observed targets $\boldsymbol{y}$, but only on the inputs X and the corresponding covariance matrix values which is a property of the Gaussian distribution. So, the variance (uncertainty) of the model prediction is larger where no training points exist ($\boldsymbol{k}_*$ has very small values).

After introducing the noise to the Gaussian process, fig. 2.34 shows the difference to the example already used of a fitted $\mathcal{GP}$ model to noisy data. As shown, the model is now able to deal with the noisy data much better than without the additional noise parameter σ_n^2.

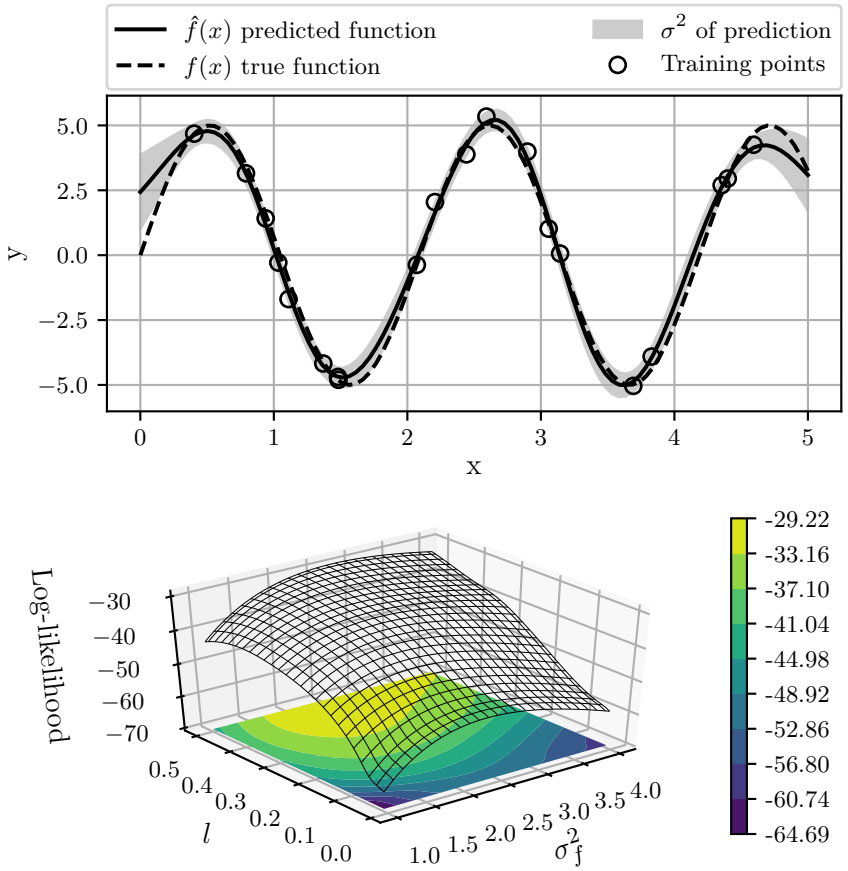


Figure 2.34: $\mathcal{GP}$ model with a squared exponential covariance function and additional noise parameter σ_n^2. The true function is $f(x) = 5\sin(3x)$, $x \in \mathbb{R}^{[0,5]}$ with some additive Gaussian noise $\mathcal{N}(0, 0.5)$. Number of training points $N = 20$. The top picture shows the trained model on the noisy data. The bottom picture shows the negative marginal log-likelihood values over the two parameters σ_f^2 and l with a fixed valued $\sigma_n^2 = 0.09$ which should be minimized in order to fit the data.

2.8.3 *Free Parameter Estimation*

There are different possibilities to train the free parameters correlation length-scale l, signal variance σ_f^2 and noise variance σ_n^2 of the $\mathcal{GP}$ model. For example, one of the methods presented in the model selection section 2.2 can be used. In most cases during the training, it would be necessary to invert the covariance matrix with complexity $\mathcal{O}(N^3)$ (see eq. 2.67). This is a numerically expensive operation, especially if N gets large. If k-fold cross-validation is used to train the model, during one optimization step the matrix inversion would be performed k times which might be very time consuming. In practice the Cholesky decomposition (Krishnamoorthy and Menon [2013]) is used in combination with solving a triangular system to estimate eq. 2.67 so that the important equations change to

$$\begin{aligned} L &= chol(K + \sigma_n^2 I) \\ \boldsymbol{\gamma} &= L^T \backslash (L \backslash \boldsymbol{y}) \\ \hat{y}_* &= \boldsymbol{k}_*^T \boldsymbol{\gamma} \\ \boldsymbol{v} &= L \backslash \boldsymbol{k}_* \\ \mathbb{V}[y_*] &= k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{v}^T \boldsymbol{v} \end{aligned} \tag{2.71}$$

whereas $\backslash$ stands for a left matrix division (see Rasmussen and Williams [2006]) with: $A \backslash \boldsymbol{b}$ is the vector $\boldsymbol{x}$ which solves $A\boldsymbol{x} = \boldsymbol{b}$. The computational complexity reduces to $\mathcal{O}(N^3/6)$ for the Cholesky decomposition compared to the standard matrix inversion. In any case, to avoid the usage of cross-validation it would also be possible to maximize the likelihood of the Gaussian process, since it is a probabilistic model. In the case of the Gaussian process, the marginal log-likelihood (or evidence) $\ln p(\boldsymbol{y}|X)$ is often used. The marginal log-likelihood is the integral of the likelihood times the prior:

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\hat{\boldsymbol{y}}, X) p(\hat{\boldsymbol{y}}|X) d\hat{\boldsymbol{y}} \tag{2.72}$$

The Gaussian process model prior is $p(\hat{\boldsymbol{y}}|X) \sim \mathcal{N}(0, K)$ or

$$\ln p(\hat{\boldsymbol{y}}|X) = -0.5 \hat{\boldsymbol{y}}^T K^{-1} \hat{\boldsymbol{y}} - 0.5 \ln |K| - \frac{N}{2} \ln 2\pi \tag{2.73}$$

and the likelihood is a factorized Gaussian $p(\boldsymbol{y}|\hat{\boldsymbol{y}}) \sim \mathcal{N}(\hat{\boldsymbol{y}}, \sigma_n^2 I)$. Finally, the integration yields (see the derivation of the integral in Rasmussen and Williams [2006]) the log-likelihood of a Gaussian process[4]:

$$\ln p(\boldsymbol{y}|X) = -0.5 \boldsymbol{y}^T \left(K + \sigma_n^2 I\right)^{-1} \boldsymbol{y} - 0.5 \ln \left(\left|K + \sigma_n^2 I\right|\right) - \frac{N}{2} \ln(2\pi) \tag{2.74}$$

The three terms of the likelihood have interpretable roles:
$-0.5 \boldsymbol{y}^T \left(K + \sigma_n^2 I\right)^{-1} \boldsymbol{y}$ is the data fit (in statistics also known as the Mahalanobis distance (De Maesschalck et al. [2000])), $-0.5 \ln \left|K + \sigma_n^2 I\right|$ is the complexity penalty depending only on the chosen covariance function $k()$ and the input X, $-\frac{N}{2} \ln(2\pi)$ is a normalization constant. Since the correlation length-scale, the signal variance and the noise variance are possible parameters of the $\mathcal{GP}$ model the marginal log-likelihood of eq. 2.74 can be also conditioned on these free parameters $\boldsymbol{\theta} = [l, \sigma_f^2, \sigma_n^2]$. with $K(\boldsymbol{\theta}) = K\left(l, \sigma_f^2\right) + \sigma_n^2 I$ to

4 See for comparison to the likelihood of the multivariate Gaussian distribution eq. 2.55.

$$\ln p(\boldsymbol{y}|X,\boldsymbol{\theta}) = -0.5\boldsymbol{y}^T(K(\boldsymbol{\theta}))^{-1}\boldsymbol{y} - 0.5\ln|K(\boldsymbol{\theta})| - \frac{N}{2}\ln(2\pi) \tag{2.75}$$

with the help of the Cholesky decomposition and $\boldsymbol{\gamma}$ from eq. 2.71, it can also be calculated through

$$\ln p(\boldsymbol{y}|X,\boldsymbol{\theta}) = -0.5\boldsymbol{y}^T\boldsymbol{\gamma}(\boldsymbol{\theta}) - \sum_i^N \ln L_{ii}(\boldsymbol{\theta}) - \frac{N}{2}\ln(2\pi) \tag{2.76}$$

The partial derivatives of the marginal log-likelihood with respect to the free parameters $\boldsymbol{\theta}$ yields

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\ln p(\boldsymbol{y}|X,\boldsymbol{\theta}) &= \frac{\partial \ln p\,(\boldsymbol{y}|X,\boldsymbol{\theta})}{\partial K(\boldsymbol{\theta})}\frac{\partial K(\boldsymbol{\theta})}{\partial\theta_j} \\ &= 0.5\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\frac{\partial}{\partial\theta_j}K(\boldsymbol{\theta})^{-1}\boldsymbol{y} - 0.5tr\left(K(\boldsymbol{\theta})^{-1}\frac{\partial K(\boldsymbol{\theta})}{\theta_j}\right)\end{aligned} \tag{2.77}$$

where tr means the trace of the resulting matrix and the subscript j stands for one of the possible free parameters. So the chosen covariance function $k(X,\boldsymbol{\theta})$ must be derived with respect to the free parameters.

2.8.4 *Classification*

Gaussian processes are also capable of handling binary classification problems. The underlying idea is to take the prediction of a $\mathcal{GP}$ model $\hat{f}(\boldsymbol{x})$ and then put it through a logistic function to obtain a prior on $\pi(\boldsymbol{x}) \triangleq p(y=+1|\boldsymbol{x}) = \sigma(\hat{f}(\boldsymbol{x}))$. This is illustrated in fig. 2.35 where a one dimensional Gaussian process prediction is transformed into the logistic function $\lambda(z) = (1+\exp(-z))^{-1}$ to obtain the class probability $\pi(x) = \lambda(\hat{f}(x))$.

The prediction of the class probability for a single new test case $\boldsymbol{x}_*$ can also be divided into two steps (Rasmussen and Williams [2006]): first computation of the distribution of the prediction corresponding to the test case:

$$p(\hat{y}_*|X,\boldsymbol{y},\boldsymbol{x}_*) = \int p(\hat{y}_*|X,\boldsymbol{x}_*,\hat{\boldsymbol{y}})p(\hat{\boldsymbol{y}}|X,\boldsymbol{y})d\hat{\boldsymbol{y}} \tag{2.78}$$

where $p(\hat{\boldsymbol{y}}|X,\boldsymbol{y}) = p(\boldsymbol{y}|\hat{\boldsymbol{y}})p(\hat{\boldsymbol{y}}|X)/p(\boldsymbol{y}|X)$ is the posterior over the $\mathcal{GP}$ prediction for X and subsequently using this distribution for the new test point prediction $\hat{y}_*$, to produce a probabilistic prediction:

$$\bar{\pi}_* \triangleq p(y_*=+1|X,\boldsymbol{y},\boldsymbol{x}_*) = \int \sigma(\hat{y}_*)p(\hat{y}_*|X,\boldsymbol{y},\boldsymbol{x}_*)d\hat{y}_* \tag{2.79}$$

Eq. 2.78 is a non-Gaussian likelihood, where the integral is analytically intractable, compared to the previous regression example, where it was straightforward to solve the integral. Similarly, eq. 2.79 can be analytically intractable for certain sigmoid functions, although in the binary case it is only a one-dimensional integral, so simple numerical techniques are generally adequate (Rasmussen and Williams [2006]). Therefore, there are options to use either analytic approximation of integrals, e.g. Laplace approximation (Williams and Barber [1998]), or solutions based on Monte Carlo sampling like Markov chain Monte Carlo ($MCMC$) approximations (Bernardo et al. [1998]). All of these methods scale as $\mathcal{O}(N^3)$,

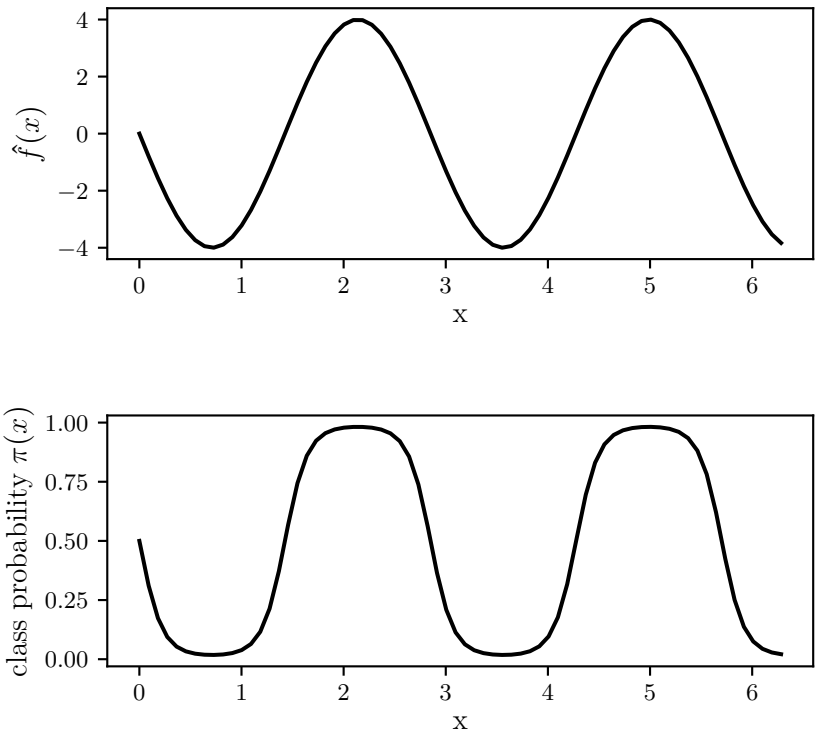


Figure 2.35: The top picture shows a one dimensional Gaussian process $\hat{f}(x)$ and the bottom picture shows the result of using the logistic function $\lambda(z) = (1 + \exp(-z))^{-1}$ to obtain the class probability $\pi(x) = \lambda(\hat{f}(x))$.

therefore for large data sets, it is numerically expensive to solve the integral. Laplace's method utilizes a Gaussian approximation $q(\hat{\boldsymbol{y}}|X, \boldsymbol{y})$ to the posterior $p(\hat{\boldsymbol{y}}|X, \boldsymbol{y})$ in the integral 2.78. Applying a second order Taylor expansion of $\ln p(\hat{\boldsymbol{y}}|X, \boldsymbol{y})$ around the maximum of the posterior, the Gaussian approximation

$$q(\hat{\boldsymbol{y}}|X, \boldsymbol{y}) = \mathcal{N}\left(\hat{\boldsymbol{y}}|\tilde{\boldsymbol{y}}, A^{-1}\right) \propto \exp(-0.5(\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}})^T A(\hat{\boldsymbol{y}} - \tilde{\boldsymbol{y}})) \tag{2.80}$$

can be obtained, where $\tilde{\boldsymbol{y}} = argmax\, p(\hat{\boldsymbol{y}}|X, \boldsymbol{y})$ is the maximum posterior (the mode) and $A = -\nabla\nabla \ln p(\hat{\boldsymbol{y}}|X, \boldsymbol{y})|_{\hat{\boldsymbol{y}}=\tilde{\boldsymbol{y}}}$ is the Hessian of the negative log posterior at that point. The method for how to obtain $\tilde{\boldsymbol{y}}$ and A in order to estimate $q(\hat{\boldsymbol{y}}|X, \boldsymbol{y})$, can be found in section 3.4 of Rasmussen and Williams [2006]. In order to make the class probability prediction in eq. 2.79, the posterior mean for $\hat{y}_*$ is first needed which can be calculated through

$$\mathbb{E}[\hat{y}_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = k\left(\boldsymbol{x}_*\right)^T K^{-1}\tilde{\boldsymbol{y}} = k\left(\boldsymbol{x}_*\right)^T \nabla \ln p\left(\boldsymbol{y}|\tilde{\boldsymbol{y}}\right) \tag{2.81}$$

together with the Gaussian approximation for the point to predict $q(\hat{y}_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$, the mean predictive probability can be estimated via

$$\bar{\pi}_* \approx \mathbb{E}[\pi_*|X, \boldsymbol{y}, \boldsymbol{x}_*] = \int \sigma\left(\hat{y}_*\right) q(\hat{y}_*|X, \boldsymbol{y}, \boldsymbol{x}_*) d\hat{y}_* \tag{2.82}$$

The corresponding approximation of the log-likelihood to train the classifier model is

$$\ln q(\boldsymbol{y}|X,\boldsymbol{\theta}) = -0.5\tilde{\boldsymbol{y}}^T K^{-1}\tilde{\boldsymbol{y}} + \ln p\left(\boldsymbol{y}|\tilde{\boldsymbol{y}}\right) - 0.5\ln|B| \tag{2.83}$$

with $|B| = |K||K^{-1} + W| = |I + W^{0.5}KW^{0.5}|$ and $W = -\nabla\nabla \ln p(\boldsymbol{y}|\tilde{\boldsymbol{y}})$.
$\ln p(\boldsymbol{y}|\tilde{\boldsymbol{y}})$ is the likelihood function of the logistic function $\ln p(y_i|\tilde{y}_i) = -\ln(1+\exp(-y_i\tilde{y}_i))$.
Examples of using $\mathcal{GP}$ classification will be shown in subsection 2.10.6.

2.8.5 *Stationary Covariance Functions*

As mentioned in the introduction of this section, Gaussian processes are non-parametric models. After explaining how this type of model works, some possibilities (or hyperparameters) are still available to influence the predictive power. This can be done mainly by the choice of covariance function and its parameters which can be seen as the core of a Gaussian process model.

Up to this point, only one covariance function type was presented: the squared exponential function (eq. 2.64). This covariance function is infinitely differentiable and leads to a very smooth $\mathcal{GP}$ model. Nevertheless, it might be that such strong smoothness assumptions are unrealistic for modeling many physical processes (Stein [2012]) and therefore other types of covariance functions are needed which are less smooth. In the following, some of them are described which will be used during this work. For all covariance function must hold that they are positive definite to be invertible.

The squared exponential function comes from a class of γ-exponential covariance functions:

$$k_\gamma(\tau, l, \gamma) = \exp\left(-\left(\frac{\tau}{l}\right)^\gamma\right), 0 < \gamma \leq 2 \tag{2.84}$$

with $\tau = \|\boldsymbol{x} - \boldsymbol{x}_*\|$. Besides the squared exponential function with $\gamma = 2$, the absolute exponential function with $\gamma = 1$ is a common choice. The squared exponential function can also be used to build a periodic kernel (see MacKay [1998]) which might be useful for periodic observations with

$$k_{Per}(\tau, l) = \exp\left(-\frac{2\sin^2(\pi\tau)}{l^2}\right) \tag{2.85}$$

Although this kernel gives rise to periodic functions, it is strictly positive as with all stationary kernels presented in this section. Intuitively, it is expected that periodic functions, like a sinusoid, will have negative covariances, since peaks are anti-correlated with troughs. Therefore, this kernel is mostly useful in combination with other covariance functions. The Matérn class of covariance functions (Matérn [2013]) is similar to the γ-exponential class but more flexible and described by

$$k_{Mat}(\tau, l, v) = \frac{2^{v-1}}{\Gamma(v)}\left(\frac{\sqrt{2v}\tau}{l}\right)^v K_v\left(\frac{\sqrt{2v}\tau}{l}\right) \tag{2.86}$$

with positive parameters v and l, where K_v is a modified Bessel function and $\Gamma()$ is the Euclidean gamma function (Abramowitz and Stegun [1964]). The choice of v controls the smoothness of the $\mathcal{GP}$ model. Common values for v are $v = 3/2, v = 5/2$ (Cornford et al. [2002]) for which the general equation simplifies to

$$k_{Mat,v=3/2}(\tau, l) = \left(1 + \frac{\sqrt{3}\tau}{l}\right) \exp\left(-\frac{\sqrt{3}\tau}{l}\right) \tag{2.87}$$

and

$$k_{Mat,v=5/2}(\tau, l) = \left(1 + \frac{\sqrt{5}\tau}{l} + \frac{5\tau^2}{3\tau^2}\right) \exp\left(-\frac{\sqrt{5}\tau}{l}\right) \tag{2.88}$$

The smaller v gets, the less smooth the $\mathcal{GP}$ model becomes. For $v = \infty$ it becomes equal to the squared exponential function and for $v = 0.5$ it becomes equal to the absolute exponential function. An other covariance function is the rational quadratic function:

$$k_{RQ}(\tau, l, \alpha) = \left(1 + \frac{\tau^2}{2\alpha l^2}\right)^{-\alpha}, \alpha > 0 \tag{2.89}$$

This function can be seen as a scale mixture (an infinite sum) of squared exponential covariance functions, therefore it is also very smooth.

To see the different levels of smoothness of these covariance functions, fig. 2.36 shows $\mathcal{GP}$ models with different covariance functions for the non-convex Ackley test function (Ackley [1985]):

$$\begin{aligned} f(x, y) = &-20 \exp\left[-0.2\sqrt{0.5(x_1^2 + x_2^2)}\right] - \exp[0.5(cos(2\pi x_1) \\ &+ cos(2\pi x_2))] + \mathrm{e} + 20,\ x_1,\ x_2 \in \mathbb{R}^{[-5,5]} \end{aligned} \tag{2.90}$$

Only the absolute exponential covariance function is able to handle this non-convex function properly. Also, it can be seen that the smooth covariance functions, like the squared exponential or the rational quadratic, smooth the non-convex function out which in this case leads to poorer predictive accuracy. The choice of the right covariance function, dependent on the structure of the data, is a crucial part in order to obtain an accurate $\mathcal{GP}$ model. This, besides a possible non-zero mean (e.g. a polynomial), is the only hyperparameter of a Gaussian process which the user has to define. It is also possible for the user to define a value for the noise variance without optimization if known from the data.

It is also possible to build new kernels from existing kernels in order to increase the predictive power of the model. Here is a list of simple techniques, reproduced from MacKay [1998] and Bishop [2006]:

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{x}_*) &= g(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{x}_*) g(\boldsymbol{x}_*) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= pol(k_1(\boldsymbol{x}, \boldsymbol{x}_*)) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= \exp(k_1(\boldsymbol{x}, \boldsymbol{x}_*)) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= k_1(\boldsymbol{x}, \boldsymbol{x}_*) + k_2(\boldsymbol{x}, \boldsymbol{x}_*) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= k_1(\boldsymbol{x}, \boldsymbol{x}_*) k_2(\boldsymbol{x}, \boldsymbol{x}_*) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= k_a(\boldsymbol{x}_a, \boldsymbol{x}_{a*}) + k_b(\boldsymbol{x}_b, \boldsymbol{x}_{b*}) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= k_a(\boldsymbol{x}_a, \boldsymbol{x}_{a*}) k_b(\boldsymbol{x}_b, \boldsymbol{x}_{b*}) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= k_1(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}_*)) \\ k(\boldsymbol{x}, \boldsymbol{x}_*) &= \boldsymbol{x}^T A \boldsymbol{x}_* \end{aligned}$$

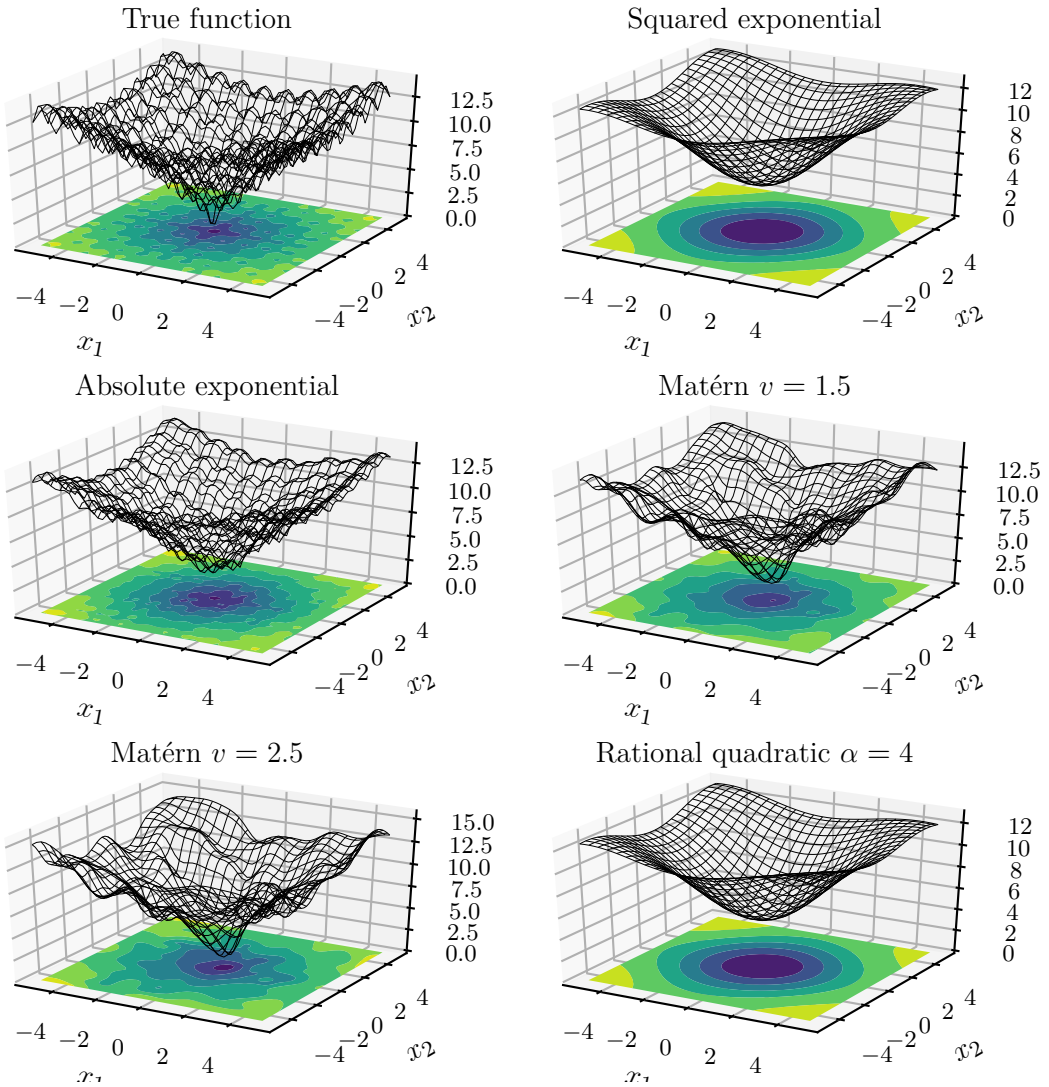


Figure 2.36: Comparison of a $\mathcal{GP}$ model with different covariance functions trained on $N = 200$ samples.

where $g()$ is any function, $pol()$ is a polynomial with non-negative coefficients, $\phi()$ is a mapping function which maps its input in a higher or lower dimensional space, A is a symmetric positive definite matrix, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are not necessarily disjoint variables with $X = (\boldsymbol{x}_a, \boldsymbol{x}_b)^T$ and k_a and k_b are valid covariance functions in their respective spaces.

A good example of combining different covariance functions can be found in Rasmussen and Williams [2006] and should be demonstrated in the following. The 545 observations of monthly averages of the atmospheric concentration of CO_2 made between 1959 and the end of 1998 derived from sites in the SIO air sampling network collected at the Mauna Loa Observatory, Hawaii (Keeling [1994]), were used to train a $\mathcal{GP}$ model with a summation of four different covariance functions. These different covariance functions should address different aspects of the data. In this example a squared exponential covariance function was used to model the smooth rising trend. Furthermore, a product of squared exponential and periodic covariance function was used to describe the seasonal variation and a ratio-

nal quadratic term to model the small medium-term irregularities. Finally, a noise term consisting of a squared exponential term and a constant component was used to address the possible measurement inaccuracies and local short-term weather phenomena. Fig. 2.37 shows the resulting fitted $\mathcal{GP}$ model, including a prediction of 30 years into the future. As demonstrated, the 95% confidence interval of the prediction into the future becomes increasingly uncertain.

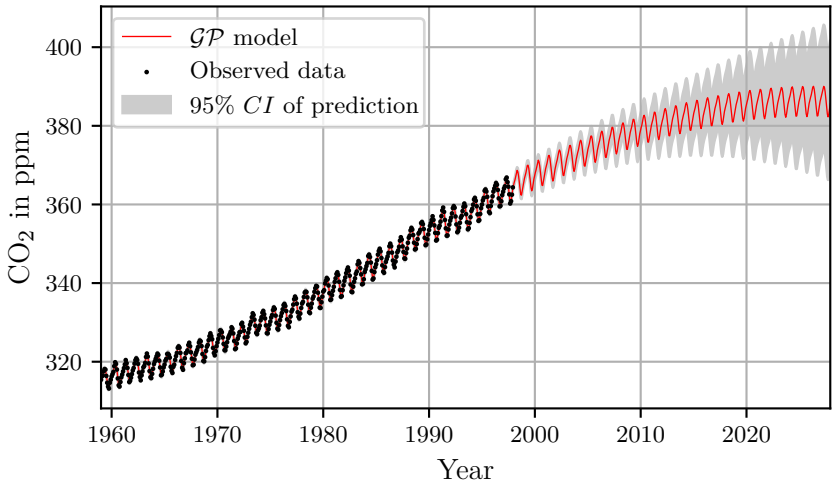


Figure 2.37: $\mathcal{GP}$ model with a summation of four different kernels to address different trends in the measured CO_2 data from 1959-1998 + 30 years future prediction. The 545 observations and the fitted model together with the 95% confidence interval of its prediction are shown. It can be seen that the extrapolation into the future results in increasing uncertainty of the model.

2.8.6 *Non-Stationary Covariance Functions*

Up to now, only stationary covariance functions have been mentioned. Stationarity in the context of covariance functions means that the covariance value $k(\boldsymbol{x}, \boldsymbol{x}_*)$ depends only on the Euclidean distance τ between the corresponding points $\boldsymbol{x}, \boldsymbol{x}_*$ and thus these kind of covariance functions are invariant to translations in the input space X. However, non-stationary covariance functions allow the model to adapt to functions whose smoothness and training point density varies within the input space. This is beneficial for example for data with discontinuities, like step functions and robotic tasks or sparse and locally adapted training samples (e.g. in optimizations). Some examples of non-stationary covariance functions are: Calandra et al. [2016] which uses a neural network to map a n dimensional input space X to a higher or lower $m \neq n$ dimensional space U as input for a $\mathcal{GP}$ model. Both, the mapping and the $\mathcal{GP}$ model, are jointly trained in order to get the best overall result. Another approach is the neural network kernel from Williams [1997]:

$$k_{NN}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}_*) = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\boldsymbol{x}}^T \Sigma \tilde{\boldsymbol{x}}_*}{\sqrt{\left(1 + 2\tilde{\boldsymbol{x}}^T \Sigma \tilde{\boldsymbol{x}}\right)\left(1 + 2\tilde{\boldsymbol{x}}_*^T \Sigma \tilde{\boldsymbol{x}}_*\right)}}\right) \tag{2.91}$$

where $\tilde{\boldsymbol{x}} = (1, x_1, ..., x_n)^T$ is an augmented input vector and Σ is a diagonal weighting matrix. There is also the class of polynomial covariance functions which are also non-stationary (Rasmussen and Williams [2006]) with

$$k_{poly}(\boldsymbol{x}, \boldsymbol{x}_*) = \left(\sigma_0^2 + \boldsymbol{x}^T \Sigma_m \boldsymbol{x}_*\right)^m \tag{2.92}$$

with m, $m \in \mathbb{Z}^{\geq 0}$ denoting the order, σ_0^2 the bias term and Σ_m is the covariance matrix on the components of $\boldsymbol{x}$.

A further type of non-stationary covariance functions can be found in Gibbs [1998], Schmidt and O'Hagan [2003], Paciorek and Schervish [2004] and Plagemann et al. [2008], all of them introduce the non-stationarity through making the correlation length-scale l dependent on $\boldsymbol{x}_*$ ($l(\boldsymbol{x}_*)$). Plagemann et al. [2008] did this by applying a second-level Gaussian Process. For this, they select several subsets of the training data and train $\mathcal{GP}$ models on these subsets. The estimated correlation lengths for these subsets are then used as observations along with the original inputs X to train the second level $\mathcal{GP}$ model. Then, this second level $\mathcal{GP}$ model can predict dependent on the prediction point $\boldsymbol{x}_*$, the corresponding length-scale l_* for the first level $\mathcal{GP}$ model which predicts the output of interest. Note that one model per length-scale would be necessary for the case of a anisotropic Gaussian process which is very inefficient. Fig. 2.38 shows this for a sinusoidal function with a discontinuity at $x = 0$. It can be seen that the length-scale reduces at the discontinuity (changing smoothness) which is not possible for the stationary covariance function. Even for this simple example, it can be seen that the non-stationary $\mathcal{GP}$ model fits the test function better. Furthermore, the uncertainty given through the prediction variance is lower compared to the stationary model. The stationary length-scales need to be very small in order to capture the discontinuity. This leads to a model which is rather over-fitted which can be seen through the wiggly behavior between the observation points. In fig. 2.39 the covariance matrix of both models is shown between 100 training and 100 test points. Here it can also be seen that the local length-scale model has a smaller area of higher covariance values at the discontinuity compared to the stationary covariance matrix which has the same covariance value bandwidth everywhere. As shown later in section 2.10, this idea of two level models will be extended to a combined one level model with several benefits.

2.8.7 *Anisotropic Gaussian Processes*

As already introduced in case of the moving least squares method (see section 2.7), Gaussian processes are also able to neglect unimportant parameters which is named automatic relevance determination (ARD, see section 2.6) (MacKay and Neal [1994]) or anisotropic Gaussian process (Paciorek and Schervish [2004]). In a similar way to the $AMLS$ presented, this is done by introducing additional parameters. In case of the $\mathcal{GP}$ model, individual correlation length-scales l_i, $i = 1, ..., n$ for each input parameter are used instead of one single length-scale. For example, the squared exponential covariance function then holds

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}_*) = \sigma_f^2 exp\left(-0.5 \sum_{i=1}^{n} \frac{(x_i - x_{i*})^2}{l_i^2}\right) \tag{2.93}$$

This allows the $\mathcal{GP}$ model to determine the relevance of each input feature to the corresponding output. If l_i gets very large, then the feature becomes irrelevant. This is applicable

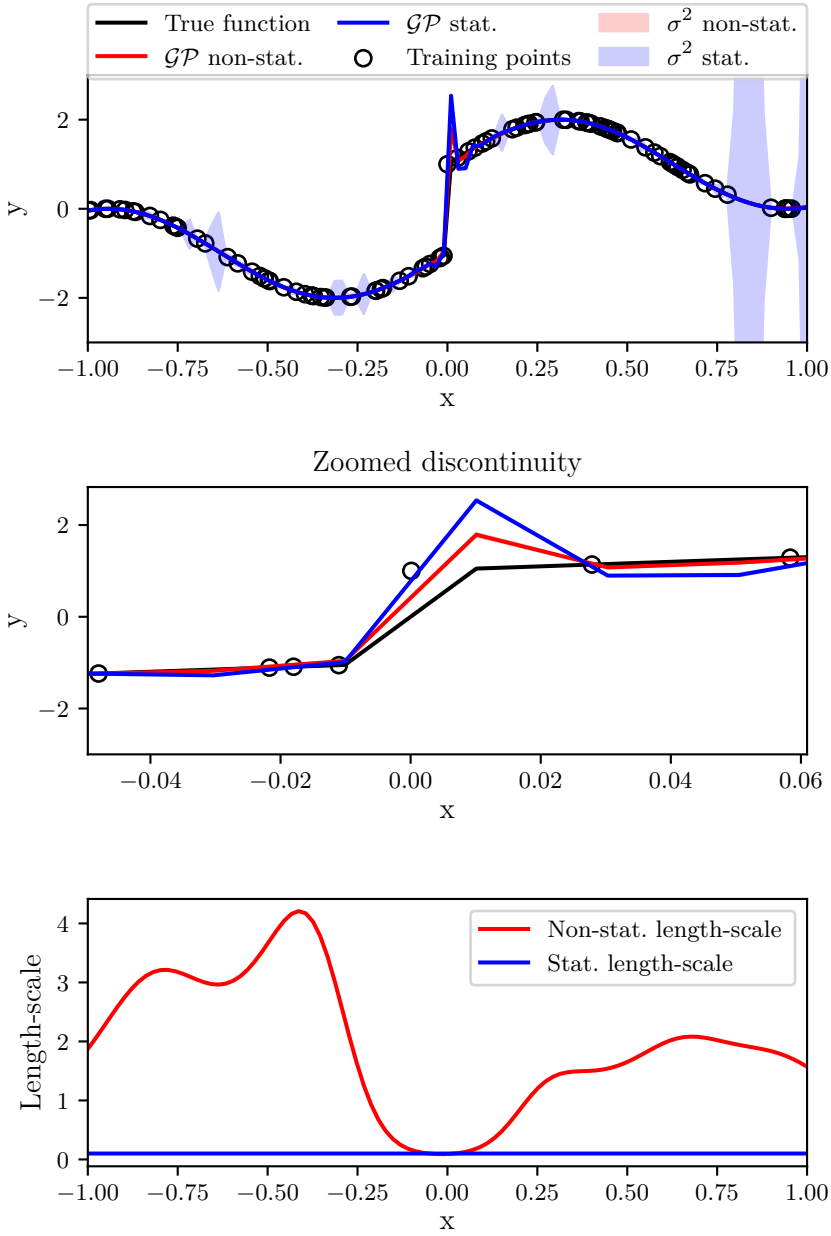


Figure 2.38: Comparison of two $\mathcal{GP}$ models with a stationary and non-stationary covariance function. Both use a squared exponential kernel. The second level $\mathcal{GP}$, to estimate local length-scales, has also a squared exponential covariance function.

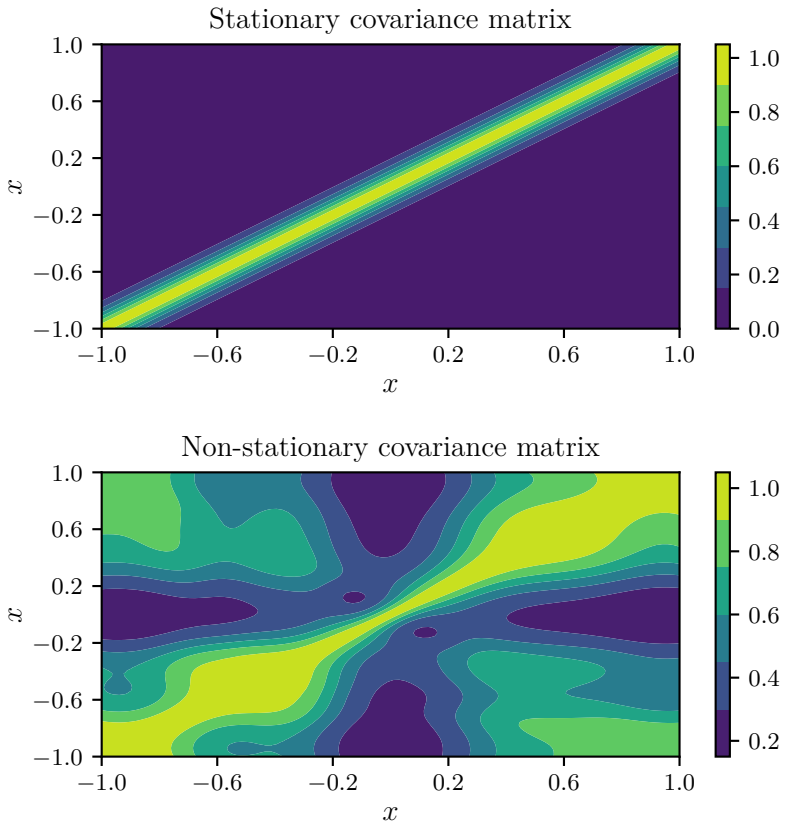


Figure 2.39: Comparison of two $\mathcal{GP}$ covariance matrices of a stationary and non-stationary covariance function. The color represents the covariance value.

to all types of covariance functions presented which include the correlation length-scale parameter. Fig. 2.40 show the difference of isotropic and anisotropic trained $\mathcal{GP}$ models with a squared exponential covariance function for the test functions eq 2.49 to 2.52 already used. The results for the values of the length-scales and the predictive coefficient of determination $\hat{R}^2$ of both models are summarized in tab. 2.4. The anisotropic $\mathcal{GP}$ model outperforms the isotropic variant for the test functions with varying importance of the input features (equivalent to the $AMLS$ version). It can also be seen that the correlation length-scales correspond to the importance of the input parameters. Compared to the $AMLS$ version of this test, the $\mathcal{GP}$ variants show superior results.

2.8.8 *Heteroscedastic Noise Estimation*

Also for the $\mathcal{GP}$ model, heteroscedasticity of the residuals (see section 2.3) plays a role. As already shown, the right choice of model (or in the case of the GP model the right covariance function) is crucial to handle this problem. Since the Gaussian process is a very flexible model, especially with a non-stationary covariance function, heteroscedasticity of the residuals is less of a problem compared to other less flexible models as has already been shown in the comparison of the MLS to the RSM (see subsection 2.7.3).

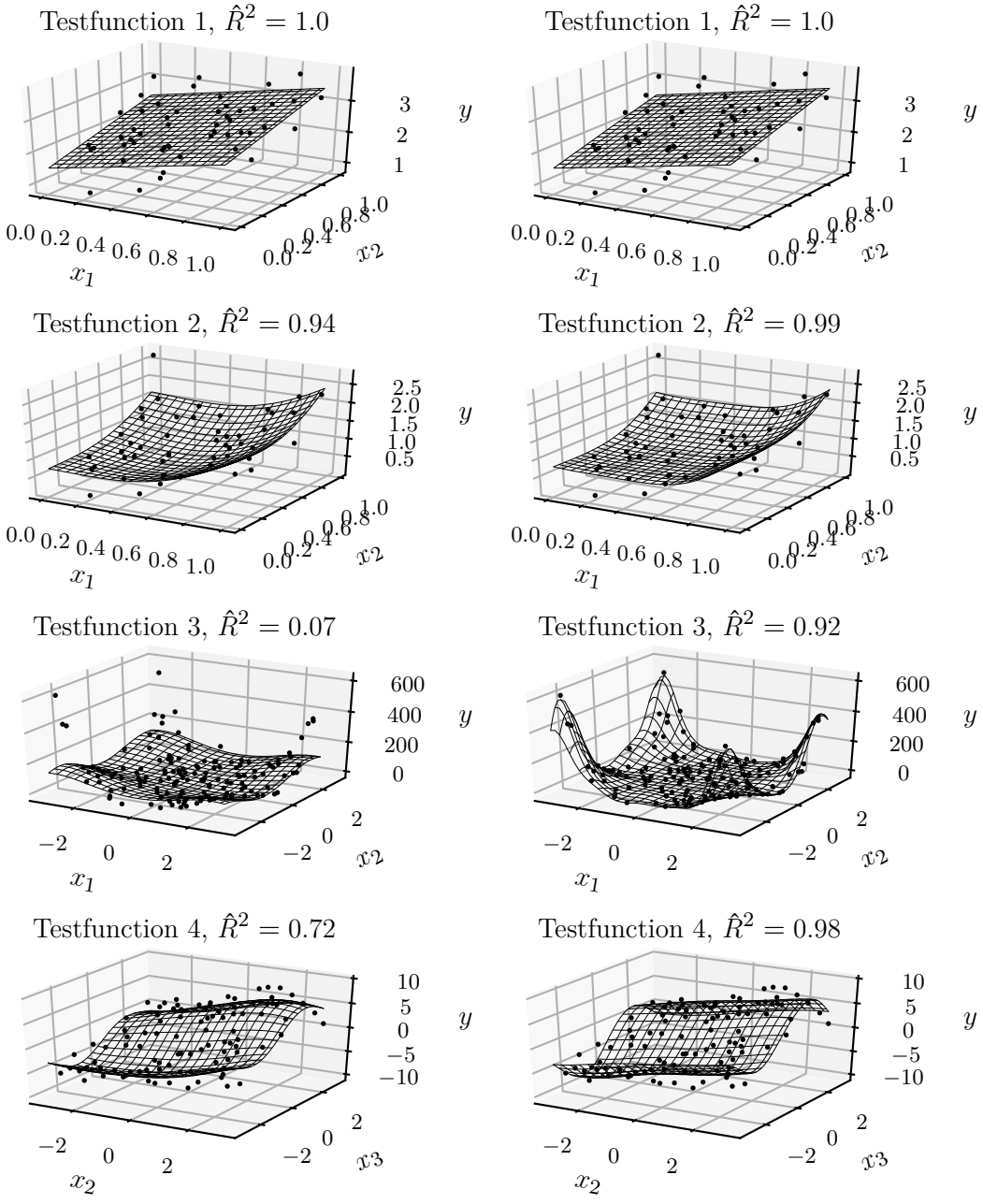


Figure 2.40: Isotropic (left) and anisotropic (right) $\mathcal{GP}$ models with squared exponential covariance function for the test function of section 2.7, eq. 2.49 to 2.52 named as testfunction 1-4.

TF	l_{iso}	l_1	l_2	l_3	l_4	l_5	$\hat{R}^2_{iso}$	$\hat{R}^2_{aniso}$
1	2.66	5.35	6.44	4.86	6.29	6.23	**1.0**	**1.0**
2	0.85	0.51	0.72	1.30	2.15	71.39	0.94	**0.99**
3	1.07	0.46	0.61	9.26	10.31	5.29	0.07	**0.92**
4	1.77	2.7	1.97	1.11	10.62	14.58	0.72	**0.98**

Table 2.4: Overview of length-scales l_i, $i = 1, ..., n$ and prognosis measurements $\hat{R}^2_{iso}$, $\hat{R}^2_{aniso}$ for test functions (TF) 2.49 to 2.52.

Another topic already briefly mentioned is the heteroscedasticity of the noise which means the noise level is not constant over the input space. An example of this is if the noise level scales with one of the input parameters or, due to other reasons, if it is higher in specific areas of the input space than in others. This can be handled by a $\mathcal{GP}$ model in a similar way as that shown for the local length-scale (see fig. 2.38). Goldberg et al. [1998], Kersting et al. [2007] and Snelson and Ghahramani [2012] all proposed the use of a second $\mathcal{GP}$ model to approximate the noise level σ_n^2 dependent on the test point $\boldsymbol{x}_*$. These works only differ in the training of the combined models. Markov chain Monte Carlo ($MCMC$) was used by Goldberg and the maximum a posteriori (MAP) approach by Kersting. $MCMC$ is very slow in large scale applications and the MAP estimation is prone to over-fitting as mentioned in Snelson and Ghahramani [2012]. Therefore, Snelson uses a variational framework by maximizing an analytically tractable lower bound on the exact marginal likelihood which still takes twice the time to train compared to a normal $\mathcal{GP}$ model. The deep Gaussian covariance network ($\mathcal{DGCN}$), newly introduced in this work in section 2.10, will show an alternative method to learn the heteroscedasticity of the noise in a much faster and more scalable way. Fig. 2.41 shows an example of taking a second $\mathcal{GP}$ model to estimate the local noise for a simple linear test function $y = x + \mathcal{N}(0, x^2)$ with heteroscedastic noise. The local noise is learned in the same way, as presented previously for the local length-scale by taking subsets. It can be seen that the heteroscedastic $\mathcal{GP}$ noise model fits much better the true noise compared to the homoscedastic one.

2.8.9 *Multi-Output*

Throughout this work, the focus has been only on the problem of predicting a single output variable y (one scalar value per $\boldsymbol{x}$ / $\boldsymbol{x}_*$). However, it might be necessary to predict multiple correlated output variables (e.g. a signal) simultaneously. A simple solution to this would be to create an independent model for each output variable separately. Still, this would be suboptimal, since the information of the cross-correlations between these outputs get lost. One way to learn and keep the multidimensional correlation for two outputs originates from Geostatistics, where correlations between the abundances of different ores, e.g. silver and lead, must be taken into account. This method is known as Co-Kriging (Myers [1982], Allard et al. [2013]). In Co-Kriging a covariance matrix is learned for each of the two correlated output variables and an additional covariance matrix for the cross-correlations, this is also known as the linear model of coregionalization. A good overview of this method

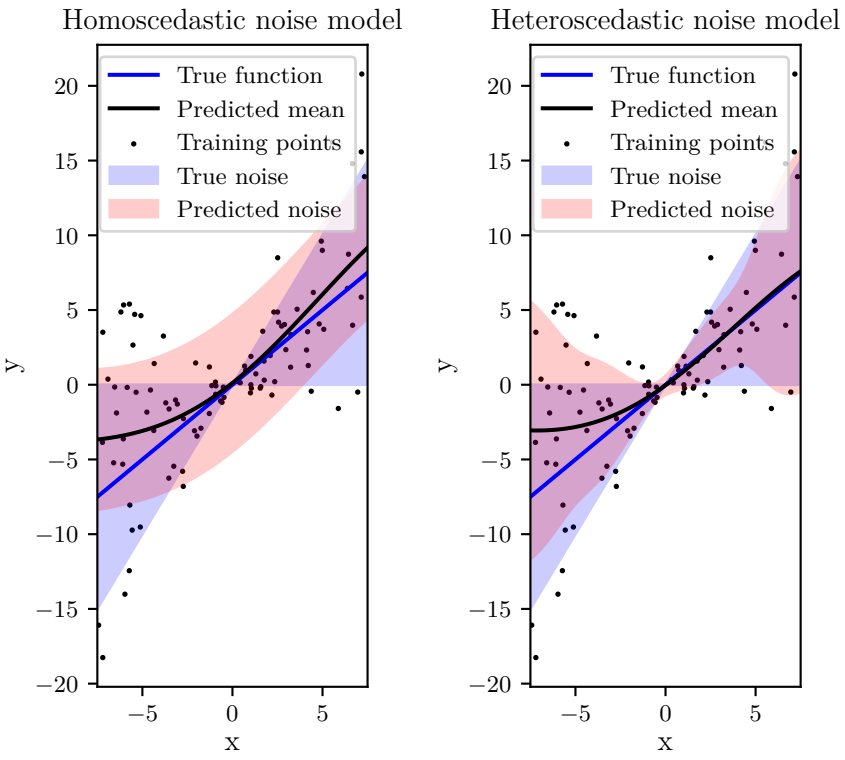


Figure 2.41: Comparison for a simple linear function $y = x + \mathcal{N}(0, x^2)$ with some heteroscedastic noise dependent on x of a homoscedastic and heteroscedastic $\mathcal{GP}$ noise model.

can be found in Cressie [1992] (section 3.2.3). In the area of mechanical engineering, this method is often used to combine low and high-fidelity results to improve the overall model prediction quality (see Han et al. [2012], Han et al. [2013]), without the need for many expensive high fidelity calculations. An example would be to combine the output of a 1D and 3D computational fluid dynamics (CFD) simulation so that only a few expensive three-dimensional simulations are needed together with many fast 1D results and the model learns the relationship between them. The benefit is that the model predictions become more accurate even though only a few 3D simulations were used for training, and it takes less time to create training samples. So, the model tries to compensate the difference between the low and high-fidelity results. This method cannot be used for more than two outputs.

Another attempt to use $\mathcal{GP}$ models for multi-output can be found in Álvarez et al. [2010] which is based on the idea of Higdon [2002] using so-called convolution processes. Convolution processes allow the integration of prior information from physical models, such as ordinary differential equations, into the covariance function. In this attempt, the convolution of several independent $\mathcal{GP}$ models is used which lead also to a covariance matrix of size $n_o \times (N \times N)$ (n_o denotes the number of output parameters) with the computational complexity of $\mathcal{O}(N^3 n_o^3)$ for training and $\mathcal{O}(N^2 n_o^2)$ for prediction. Therefore, this approach is not very useful if n_o increases to large values. The Co-Kriging approach can be seen as a particular case of the convolution processes.

The deep Gaussian covariance network $\mathcal{DGCN}$, newly introduced in this work in section 2.10, will show an improvement to this problem with a fast calculation of multi-output problems even for very high values of n_o. At this point, it should also be mentioned that neural networks (section 2.9) also offer a much more efficient way to build multi-output models and are typically used for these kinds of tasks.

2.8.10 *Further Extensions to Gaussian Processes*

In this last subsection about the $\mathcal{GP}$ model, some of the actual trends of research which are not covered in this work but might be interesting for future research, will be presented. One of the most interesting new topics is the so-called deep Gaussian processes introduced by Damianou and Lawrence [2013] and further developed by Hensman and Lawrence [2014], Dai et al. [2015], Bui et al. [2016]. The deep Gaussian process is a deep belief network based on Gaussian process mappings. The data is modeled as the output of a multivariate $\mathcal{GP}$. The inputs to that Gaussian process are then governed by another $\mathcal{GP}$. A single layer model is equivalent to a standard $\mathcal{GP}$ (Damianou and Lawrence [2013]). So, it is a neural network with $\mathcal{GP}$ models as activation functions. Even though this method will not be covered in detail here, it will be compared for benchmarks with the newly developed $\mathcal{DGCN}$ model presented in section 2.10.

Another way of using Gaussian processes is to use them for data compression, similar to a principle component analysis (PCA) (Wold et al. [1987]). Lawrence [2004] showed this for the first time and it was further extended in Titsias and Lawrence [2010] and Damianou et al. [2016]. This formulation interprets PCA as a particular Gaussian process prior on a mapping from a latent space to the observed data space (Damianou and Lawrence [2013]). This allows, compared to standard $\mathcal{GP}$ models, a non-linear mapping from higher to lower dimensions to be performed (similar to neural network based autoencoders (Vincent et al. [2010])). This method might be a useful pre-processing step for correlated data, like time series or multi-output problems, to reduce the amount of output variables that are needed to approximate.

Furthermore, the topic of handling large data sets is part of active research. Since the $\mathcal{GP}$ is limited to a smaller amount of data ($N < 5,000 - 10,000$ depending on memory resources $8 - 16$ GB of RAM) due to its complexity of $\mathcal{O}(N^3)$ (matrix inversion), it is usually not used for big data applications. Many publications deal with this topic like Lawrence [2007], Snelson and Ghahramani [2007], Banerjee et al. [2008], Titsias [2009] and Gal et al. [2014]. A common method, besides simply taking a subset of the data to approximate the sparse covariance matrix (Seeger et al. [2003]), has become the usage of so called inducing points as proposed in Snelson and Ghahramani [2006], where $m \ll N$ pseudo input points and observations were generated which should represent the same information as the full data set with N points. These points are jointly optimized with the correlation length-scales and noise variance. To be more precise, the user defines the number of pseudo points and then the optimal position of these points (which must not be equal to the real input points) will be found by maximizing the likelihood. The corresponding pseudo observations of these points will then be approximated also by a Gaussian process. The complexity then becomes $\mathcal{O}(m^2 N)$. Quiñonero-Candela and Rasmussen [2005] give a good overview of all algorithms using this method. As Hensman et al. [2013] stated, the usage of these inducing points is a trade-off between accuracy and efficiency.

Lastly, further impressive extension topics, which might not be of actual research interest, can be found in Rasmussen and Williams [2006] chapter 9; for example, non-Gaussian processes.

Important notes:

- Gaussian processes ($\mathcal{GP}$) are non-parametric models which define a distribution over functions conditioned on observations.
- Based on this distribution over functions, the mean is taken for the prediction and the variance to estimate the prediction uncertainty of the model.
- Gaussian processes can be used for supervised regression and classification tasks. The likelihood function changes between these two tasks and the prediction is passed through a logistic function for classification.
- There are three free parameters: the length-scale l, the signal variance σ_f^2 and noise variance σ_n^2 which can be optimized via maximizing the marginal log-likelihood function which underlies the assumption made on the data; namely that the error of the data has a Gaussian distribution. There are also other model assumptions, for instance the Student-t process (see Rasmussen and Williams [2006] chapter 9).
- The length-scale defines how many points around a point to predict should be considered (similar to the radius of the MLS model).
- If one length-scale parameter is defined per input parameter, it is called an anisotropic Gaussian process which is capable of reducing the influence of unimportant parameters and thus might lead to superior accuracy (similar to the $AMLS$).
- The signal variance is a scaling factor for the maximal covariance and often not considered for the optimization, as in this work.
- The noise variance is important for noisy observations and can also be optimized simultaneously with the length-scale or be set by the user.
- These three parameters are all included in most covariance function types which must be defined by the user. The covariance function controls the values of the covariance matrix which must be positive definite in order to be invertible.
- A Gaussian process is fully described by its mean and the covariance matrix or, more precisely, the covariance function. If the mean is set to zero (often assumed in published works), then the Gaussian process is only described by the covariance function. Therefore, the right choice is crucial, mainly dependent on the smoothness of the observed data.
- There are several different covariance functions and it is also possible to build new ones out of existing ones by, for example, the summation or multiplication

of the corresponding covariance matrices. Combinations of covariance function can be helpful in complex problems, since each of them can describe a different part of the underlying complexity.

- Non-stationary covariance functions can increase even further the flexibility of a $\mathcal{GP}$, for example with local length-scales or local noise variance parameters dependent on the point to predict.
- There are attempts for multi-output problems, such as Co-Kriging (two outputs) or convolution processes which, due to its numerical complexity, are only useful if the number of output parameters is not too large.
- Also for big data application some methods are available, mainly based on so-called inducing points which are pseudo inputs / outputs and lead to reduced predictive accuracy.
- Otherwise Gaussian processes are not usable for big data applications due to its complexity of $\mathcal{O}(N^3)$. The amount of usable training samples depends on memory resources (about 10,000 points for 16 GB of RAM).

2.9 NEURAL NETWORKS

2.9.1 *Introduction and Notation*

In the following, neural networks (e.g. Schmidhuber [2015]), also commonly named deep neural networks or artificial neural networks, are briefly introduced and an explanation of how to train them will be given. Neural networks are an important part of the newly developed deep Gaussian covariance network ($\mathcal{DGCN}$) introduced in this work in section 2.10. The mathematical notation and derivations used in the following are based on Higham and Higham [2018].

A standard feedforward neural network consists typically of three general types of layers (see fig. 2.42). Firstly, the input layer which contains as many neurons as input features n exist. These neurons provide the information from the training data to the network. No computation is performed in these nodes[5]. Secondly, the so-called hidden layers which perform the (non-)linear transformation of the input to the output layer. Typically, the number of hidden layers n_{hl} as well as the number of neurons $n_n^{[l]}$ (the superscript $[l]$ denotes the corresponding layer) per hidden layer must be specified by the user (hyperparameters) or can be found by some expensive grid search (Bergstra et al. [2011]), the same holds for the type of activation function in the neurons per hidden layer and output layer. The activation functions are in most cases simple mathematical functions such as, for example, the sigmoid function (Schmidhuber [2015])

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2.94}$$

which has the convenient property that its derivative takes the simple form

5 The terms node and neuron are arbitrarily interchangeable in the context of neural networks in this work.

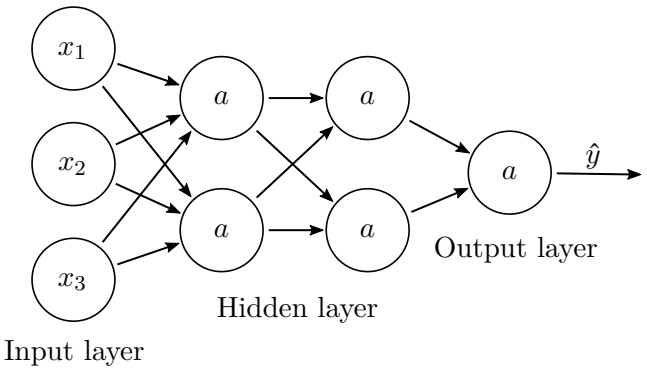


Figure 2.42: Neural network with 4 layers, 1 input, 2 hidden and 1 output layer.

$$f'(x) = f(x)(1 - f(x)) \tag{2.95}$$

Other often used activation functions are for example (also illustrated in fig. 2.43) (Glorot and Bengio [2010]):

Identity:

$$\begin{aligned} f(x) &= x \\ f'(x) &= 1 \end{aligned} \tag{2.96}$$

Hyperbolic tangent ($TanH$):

$$\begin{aligned} f(x) &= \tanh(x) \\ f'(x) &= 1 - f(x) \end{aligned} \tag{2.97}$$

Rectified linear unit ($ReLU$):

$$\begin{aligned} f(x) &= \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \\ f'(x) &= \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \end{aligned} \tag{2.98}$$

The output layer is the last layer of a neural network and contains only one neuron for a single output variable $n_o = 1$ but also a multi-output regression is performed very simply, by just setting the number of neurons in the output layer to the desired amount of output variables n_o. If the neural network should perform a classification task, then the number of neurons in the output layers is equal to the classification labels / classes and there might be also some kind of non-linear activation function used, for example a step function which is zero or one dependent on the label identified[6]. For regression tasks, the neurons of the output layer use the identity activation function, so no transformation is performed. The number of hidden layers n_{hl} should increase with the amount of complexity the model should represent. The problem is that this can also lead very quickly to an over-fitted model. If there is more than one hidden layer, it is called a deep neural network. A deep neural network (DNN) can learn any function with a sufficient number of hidden layers and

6 The sigmoid function can also be used which is able to give the class probabilities.

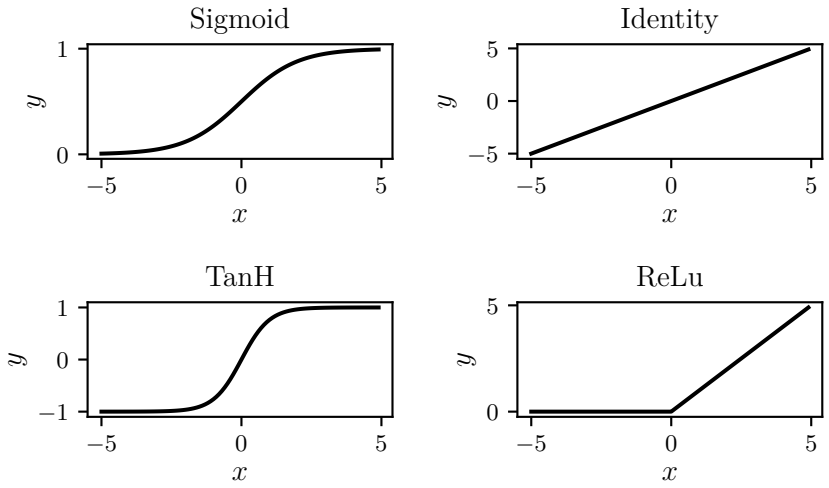


Figure 2.43: Neural network activation function types.

neurons (Cybenko [1989]). This makes it a powerful algorithm for any kind of supervised regression or classification task.

The simplest way to explain how a DNN works, is to begin with a single neuron network, also called a single layer perceptron (Rosenblatt [1958]), as shown in fig. 2.44. The prediction equation for one point $\boldsymbol{x}$[7] is

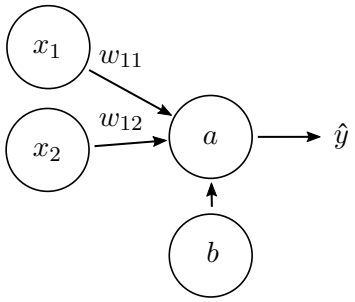


Figure 2.44: Single neuron network (perceptron).

$$\hat{y}(\boldsymbol{x}) = a\left(W\boldsymbol{x} + \boldsymbol{b}\right) = a\left(\sum_{j=1}^{n_n^{[l]}} \sum_{i=1}^{n_n^{[l-1]}} w_{ji}x_i + b_j\right) = a\left(\sum_{j=1}^{1} \sum_{i=1}^{2} w_{ji}x_i + b_j\right) \tag{2.99}$$

where $a()$ describes the activation function. W is the matrix containing the weights w_{ji}. The number of columns i in W matches the number of neurons that produces the vector $\boldsymbol{x}$ in the previous layer (here it is the input layer with $n_n^{[l]} = 2$, so W has only two columns). The number of rows j matches the number of neurons at the current layer l (in this case one row, since it has only one neuron). The vector $\boldsymbol{b}$ contains the bias terms b_j which also

7 The distinction between a known point $\boldsymbol{x}$ and a unknown point $\boldsymbol{x}_*$ is neglected here because it makes no difference for a neural network. It does not use like the $\mathcal{GP}$ model the input / output information directly for predictions.

matches the number of neurons at the current layer. These weights and the bias terms are the free parameters of a neural network which need to be estimated in order to fit the model to the training data. The next step would be to extend the network to a deep neural network with two hidden layers ($n_{hl} = 2$) and a further output layer with 2 neuron as the output (multi-output), so in total the number of layers is $n_l = 4$ which is also illustrated in fig. 2.45. Since the input data still has the form $\boldsymbol{x} \in \mathbb{R}^2$, the weights and biases for layer two are represented by a matrix $W^{[2]} \in \mathbb{R}^{2\times 2}$ and a vector $\boldsymbol{b}^{[2]} \in \mathbb{R}^2$. The output from layer two then has the form

$$a\left(W^{[2]}\boldsymbol{x} + \boldsymbol{b}^{[2]}\right) \in \mathbb{R}^2 \tag{2.100}$$

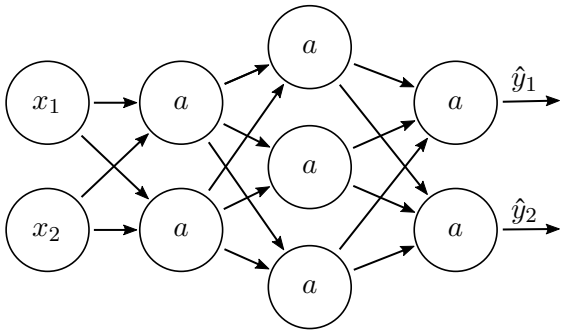


Figure 2.45: Deep neural network with 4 layers and two outputs.

Layer three has three neurons ($n_n^{[3]} = 3$), each receiving input in $\mathbb{R}^2$. So the weights and biases for layer three can be represented by a matrix $W^{[3]} \in \mathbb{R}^{3\times 2}$ and a vector $\boldsymbol{b}^{[3]} \in \mathbb{R}^3$, respectively. The output from layer three then has the form

$$a\left(W^{[3]}a\left(W^{[2]}\boldsymbol{x} + \boldsymbol{b}^{[2]}\right) + \boldsymbol{b}^{[3]}\right) \in \mathbb{R}^3 \tag{2.101}$$

Finally, the fourth (output) layer has two neurons, each receiving input in $\mathbb{R}^3$. So the weights and biases for this layer may be represented by a matrix $W^{[4]} \in \mathbb{R}^{2\times 3}$ and a vector $\boldsymbol{b}^{[4]} \in \mathbb{R}^2$, respectively. The output from layer four, and hence from the overall network, has the form

$$\hat{\boldsymbol{y}}_{DNN} = a\left(W^{[4]}a\left(W^{[3]}a\left(W^{[2]}\boldsymbol{x} + \boldsymbol{b}^{[2]}\right) + \boldsymbol{b}^{[3]}\right) + \boldsymbol{b}^{[4]}\right) \in \mathbb{R}^2 \tag{2.102}$$

Eq. 2.102 defines a function $f : \mathbb{R}^2 \to \mathbb{R}^2$ in terms of its 23 free parameters (weights and biases). This introduction of the mathematical notation can be generalized to an arbitrary sized neural network. A network with n_l layers, with layers 1 and n_l being the input and output layers. Each of the n_l layers has $n_n^{[l]}$ neurons. So, for example, $n_n^{[1]}$ is the dimension of the input data. Overall, the network maps from $\mathbb{R}^{n_n^{[1]}}$ to $\mathbb{R}^{n_n^{[n_l]}}$. The weight matrix $W^{[l]}$ has the form $\mathbb{R}^{n_n^{[l]} \times n_n^{[l-1]}}$. More precisely, $w_{ji}^{[l]}$ is the weight that neuron j at layer l applies to the output from neuron i at layer $l-1$. Similarly, $\boldsymbol{b}^{[l]} \in \mathbb{R}^{n_n^{[n_l]}}$ is the vector of biases for layer l, so neuron j at layer l uses the bias $b_j^{[l]}$. Fig. 2.46 shows an example with $n_l = 5$ layers. Here, $n_n^{[1]} = 4$, $n_n^{[2]} = 3$, $n_n^{[3]} = 4$, $n_n^{[4]} = 5$ and $n_n^{[5]} = 2$, so $W^{[2]} \in \mathbb{R}^{3\times 4}$, $W^{[3]} \in \mathbb{R}^{4\times 3}$, $W^{[4]} \in \mathbb{R}^{5\times 4}$, $W^{[5]} \in \mathbb{R}^{2\times 5}$, $\boldsymbol{b}^{[2]} \in \mathbb{R}^3$, $\boldsymbol{b}^{[3]} \in \mathbb{R}^4$, $\boldsymbol{b}^{[4]} \in \mathbb{R}^5$, $\boldsymbol{b}^{[5]} \in \mathbb{R}^2$. Given an input

$\boldsymbol{x} \in \mathbb{R}^{n_n^{[1]}}$, the output of the network can be summarized by letting $o_j^{[l]}$ denote the output, or activation, from neuron j at layer l. So that the output of each layer can be written as

$$\boldsymbol{o}^{[1]} = \boldsymbol{x} \in \mathbb{R}^{n_n^{[1]}} \tag{2.103}$$

$$\boldsymbol{o}^{[l]} = a\left(W^{[l]}\boldsymbol{o}^{[l-1]} + \boldsymbol{b}^{[l]}\right) \in \mathbb{R}^{n_n^{[l]}}, \text{ for } l = 2, 3, ..., n_l \tag{2.104}$$

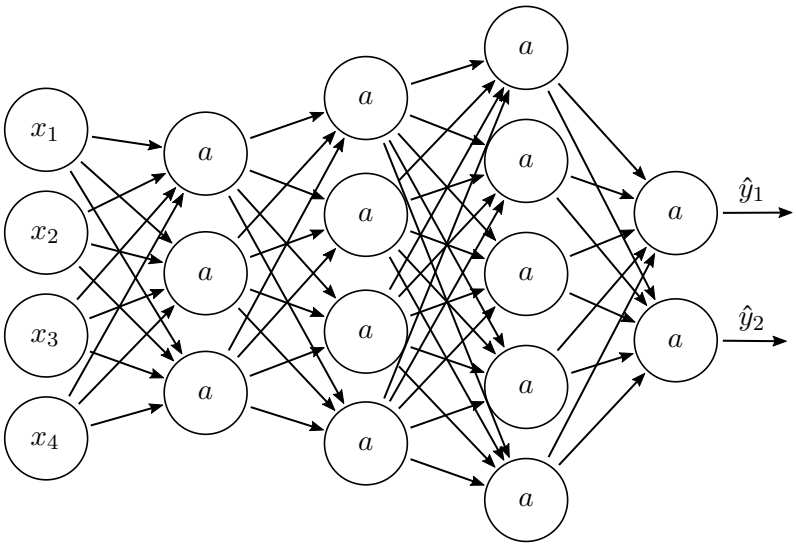


Figure 2.46: Deep neural network with 5 layers and two outputs.

2.9.2 *Stochastic Gradient Descent*

After introducing the general topology and its notation for a deep neural network, the next step is to explain how such a network can be trained. To train such a network with N training samples, there are different loss functions dependent on the learning task (regression / classification). For the regression task, the quadratic loss function is typically used which should be minimized with respect to all free parameters (weights and biases):

$$L = \frac{1}{N}\sum_{i=1}^{N} 0.5 \left\| y(\boldsymbol{x}_i) - o^{[n_l]}(\boldsymbol{x}_i) \right\|^2 \tag{2.105}$$

To simplify the following equations, all free parameters (which are matrices and vectors) should be stored in a single vector $\boldsymbol{p}$. So, for the first example with 23 free parameters, $\boldsymbol{p} \in \mathbb{R}^{23}$. Generally, it is supposed that $\boldsymbol{p} \in \mathbb{R}^s$, and write the loss function as $L(\boldsymbol{p})$ to emphasize its dependence on these parameters. A classical method in optimization that vector $\boldsymbol{p}$ is often named as steepest descent or gradient descent (Bottou [2010]) and will be introduced in the following. This method proceeds iteratively, computing a sequence of vectors in $\mathbb{R}^s$ with the aim of converging to a vector that minimizes the loss function. So, how to choose a perturbation, $\Delta\boldsymbol{p}$, so that the next vector $\boldsymbol{p} + \Delta\boldsymbol{p}$ represents an improvement? If $\Delta\boldsymbol{p}$ is small, then ignoring terms of order $|\Delta\boldsymbol{p}|^2$, a Taylor series expansion gives

$$L(\boldsymbol{p}+\Delta\boldsymbol{p}) \approx L(\boldsymbol{p}) + \sum_{r=1}^{s} \frac{\partial L(\boldsymbol{p})}{\partial p_r} \tag{2.106}$$

Here $\partial L(\boldsymbol{p})/\partial p_r$ denotes the partial derivative of the cost function with respect to the r-th parameter. Therefore, $\nabla L(\boldsymbol{p}) \in \mathbb{R}^s$ denotes the vector of partial derivatives, known as the gradient so that

$$(\nabla L(\boldsymbol{p}))_r = \frac{\partial L(\boldsymbol{p})}{\partial p_r} \tag{2.107}$$

Then eq. 2.106 becomes

$$L\left(\boldsymbol{p}+\Delta\boldsymbol{p}\right) \approx L(\boldsymbol{p}) + \nabla L(\boldsymbol{p})^T \Delta\boldsymbol{p} \tag{2.108}$$

To reduce the loss function, a $\Delta\boldsymbol{p}$ has to be chosen such that $\nabla L(\boldsymbol{p})^T\Delta\boldsymbol{p}$ becomes as negative as possible. This problem can be addressed via the Cauchy-Schwarz inequality (Steele [2004]) which states that for any $f, g \in \mathbb{R}^s$, it yields that $\left\|f^T g\right\| \leq \|f\|\,\|g\|$. So, the most negative that $f^T g$ can be, is $-\|f\|\,\|g\|$ which happens when $f = -g$. Hence, based on eq. 2.108 $\Delta\boldsymbol{p}$ should lie in the direction $-\nabla L(\boldsymbol{p})$. Keeping in mind that eq. 2.108 is an approximation that is relevant only for small $\Delta\boldsymbol{p}$, this limitation can be introduced by the update rule

$$\boldsymbol{p} \rightarrow \boldsymbol{p} - \eta \nabla L(\boldsymbol{p}) \tag{2.109}$$

Here η is a small step size that is known as the learning rate in the context of neural networks. This equation defines the gradient descent method. In practice, this will be used by choosing an initial vector $\boldsymbol{p}$ and then iterating with eq. 2.109 until some stopping criterion has been met, or until the number of iterations (sometimes called epochs) has exceeded the computational budget. The learning rate is a crucial parameter which has a strong influence on the predictive power of a neural network. If it is set too high, it could be that the training is stuck between two local minima. If it is too small, it might take a large amount of iterations to reasonably reduce the loss function. Fig. 2.47 shows an example of different learning rates used for a neural network training. It can be seen that the learning rate of $\eta = 0.1$ reduces the error very fast but then cannot get as low as the lower learning rates. In subsection 2.9.4 some possibilities are mentioned to overcome this problem.

The cost function of eq. 2.105 involves a sum of individual terms that runs over the training data. It follows that the partial derivative $\nabla L(\boldsymbol{p})$ is a sum over the training data of individual partial derivatives. More precisely, the loss function for a single point yields

$$L_{\boldsymbol{x_i}} = 0.5 \left\|y(\boldsymbol{x}_i) - o^{[n_l]}\left(\boldsymbol{x}_i\right)\right\|^2 \tag{2.110}$$

then from eq. 2.105 follows

$$\nabla L(\boldsymbol{p}) = \frac{1}{N} \sum_{i=1}^{N} \nabla L_{\boldsymbol{x_i}}(\boldsymbol{p}) \tag{2.111}$$

If the number of parameters $\boldsymbol{p}$ and the number of training samples N is large, the computing of the gradient vector $\nabla L(\boldsymbol{p})$ can be prohibitively expensive. A much cheaper alternative is to replace the mean of the individual gradients over all training points by the gradient at a single, randomly chosen, training point. This leads to the simplest form of what is called the

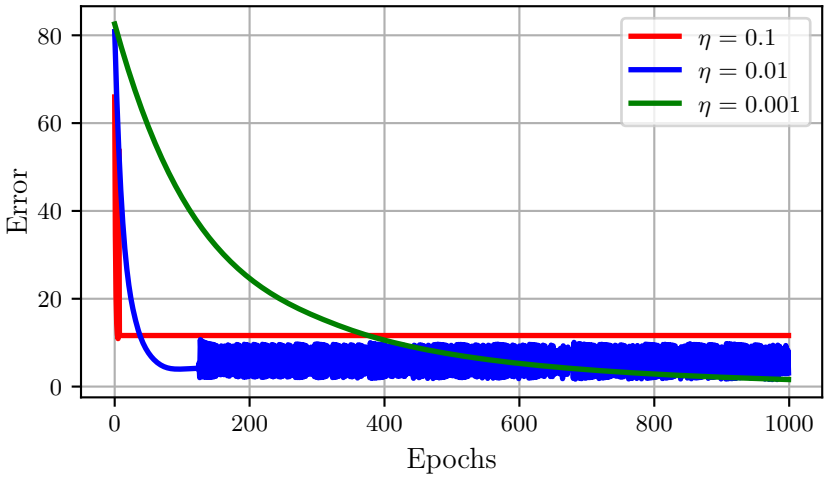


Figure 2.47: Comparison of different learning rates for the training of a neural network.

stochastic gradient descent method. Typically, the points are chosen without replacement so that each point has an equal impact on the training and if all N points have been used once to update the free parameters of the network, a so-called epoch is finished. So each training point contributed once to the update of the free parameters within one epoch. As a further extension to this, it is natural to consider not only one sample at a time to approximate the mean over all training points but a smaller subset $N_b \ll N$ which changes the update rule to

$$\boldsymbol{p} \to \boldsymbol{p} - \eta \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla L_{\boldsymbol{x_i}}(\boldsymbol{p}) \tag{2.112}$$

This subset is also known as a mini-batch (Konečnỳ et al. [2015]). This kind of training is the usual way of handling a large amount of data in an efficient way. Therefore, neural networks are also one of the best choices for big data problems. The training with one single input point is also known as online learning, because the model can learn directly by updating its parameters with the income of every new training point.

2.9.3 *Backpropagation*

With the explanation of the stochastic gradient descent method, it is now possible to apply this method to a neural network. Therefore, the vector $\boldsymbol{p}$ will be switched to the entries in the weight matrices and bias vectors. The task is to compute the partial derivatives of the loss function w.r.t. each $w_{ji}^{[l]}$ and $b_j^{[l]}$. Since eq. 2.105 is a linear combination of individual terms that runs over the training data, the same is true for its partial derivatives. Therefore, it is necessary to derive the individual partial derivatives. Hence, for a fixed training point, eq. 2.110 can be generalized to[8]

$$L = 0.5 \left\| \boldsymbol{y} - \boldsymbol{o}^{[n_l]} \right\|^2 \tag{2.113}$$

8 Note that the outputs are marked as vectors because it includes the case of multiple output variables.

The dependence of L on the weights and biases arises only through $\boldsymbol{o}^{[n_l]}$. To derive the partial derivatives, it is useful to introduce two further sets of variables. Firstly:

$$\boldsymbol{z}^{[l]} = W^{[l]}\boldsymbol{o}^{[l-1]} + \boldsymbol{b}^{[l]} \in \mathbb{R}^{n_n^{[l]}}, \text{ for } l = 2, 3, ..., n_l \tag{2.114}$$

In the following $z_j^{[l]}$ is used to describe the weighted input for neuron j at layer l. The fundamental relation of eq. 2.104 that propagates information through the network may then be written as

$$\boldsymbol{o}^{[l]} = a\left(\boldsymbol{z}^{[l]}\right), \text{ for } l = 2, 3, ..., n_l \tag{2.115}$$

Secondly, let $\delta^{[l]} \in \mathbb{R}^{n_n^{[l]}}$ be defined by

$$\delta_j^{[l]} = \frac{\partial L}{\partial z_j^{[l]}}, \text{ for } 1 \leq j \leq n_n^{[l]} \text{ and } 2 \leq l \leq n_l \tag{2.116}$$

This expression, which is often called the error in the j-th neuron at layer l, is an intermediate quantity that is useful both for analysis and computation. The idea of referring to $\delta_j^{[l]}$ as an error, comes from the fact that the loss function can only be at a minimum if all partial derivatives are zero, so $\delta_j^{[l]} = 0$ is a useful goal. At this stage, is it is also necessary to define the component-wise product of two vectors. If $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, then $\boldsymbol{x} \circ \boldsymbol{y} \in \mathbb{R}^n$ is defined by $(\boldsymbol{x} \circ \boldsymbol{y})_i = x_i / y_i$. With this, the following results are the consequence of the chain rule:

$$\delta^{[n_l]} = a'\left(\boldsymbol{z}^{[n_l]}\right) \circ \left(\boldsymbol{o}^{[n_l]} - \boldsymbol{y}\right) \tag{2.117}$$

$$\delta^{[l]} = a'\left(\boldsymbol{z}^{[n_l]}\right) \circ \left(W^{[l+1]}\right)^T \delta^{[l+1]}, \text{ for } 2 \leq l \leq n_{l-1} \tag{2.118}$$

$$\frac{\partial L}{\partial b_j^{[l]}} = \delta_j^{[l]}, \text{ for } 2 \leq l \leq n_l \tag{2.119}$$

$$\frac{\partial L}{\partial w_{ji}^{[l]}} = \delta_j^{[l]} o_i^{[l-1]}, \text{ for } 2 \leq l \leq n_l \tag{2.120}$$

The proof of these equations can, for example, be found in Higham and Higham [2018], where the used notation is also taken from. The first eq. 2.117 is the error of the network output which is used for prediction. This can be obtained by the forward pass through the network, computing $\boldsymbol{o}^{[1]}$, $\boldsymbol{z}^{[2]}$, $\boldsymbol{o}^{[2]}$, $\boldsymbol{z}^{[3]}$, ..., $\boldsymbol{o}^{[n_l]}$ in order. Then the partial errors $\delta^{[n_l-1]}$, $\delta^{[n_l-2]}$, ..., $\delta^{[2]}$ in eq. 2.118 may be computed in a backward pass. With these results of eq. 2.117 and eq. 2.118, it is possible to calculate the partial derivatives in eq. 2.119 and in eq. 2.120. Computing gradients in this way is known as back propagation (Werbos [1974]), because the error is back propagated through the network to the corresponding free parameters.

2.9.4 *Advanced Methods*

After explaining the basic training of a neural network, some advanced methods will be mentioned in this section along with references, without mathematical detailed explanations, since neural networks are not the focus of this work but a tool for the deep Gaussian covariance network which is explained later in section 2.10. Some of the advanced methods mentioned in the following will also be used for the $\mathcal{DGCN}$ model.

The first important thing is the right choice of the optimization algorithm or, more precisely, the right choice of learning rate. As already mentioned, the learning rate parameter η is very important to obtain low loss values in the gradient descent method. If it is too high, the global optimum might not be found. If it is too low, optimization takes a long time, or if the number of epochs is set too small, the global optimum will not be found. Fig. 2.48 should illustrate that.

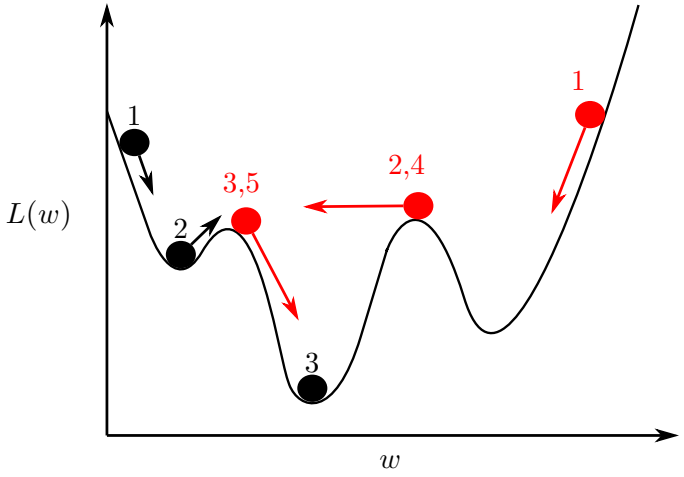


Figure 2.48: Loss over multiple training steps with different learning rates indicated through the arrow length. The numbers stand for the training step and the ball shows the current loss at the corresponding training step.

In a very simple approach the value of η remains constant over all epochs. But there are also more sophisticated methods which change the learning rate dynamically dependent on the history of the update steps. A good overview can be found in Ruder [2016].

One common method is momentum (Qian [1999]) which is a method that helps to accelerate stochastic gradient descent in the relevant direction and dampens oscillations. It does this by adding a fraction term of the update vector of the past time step to the current update vector. Therefore, the momentum term increases for free parameters whose gradients point in the same directions and reduces updates for parameters whose gradients change directions. As a result, a faster convergence and reduced oscillation can be obtained (Ruder [2016]).

Another method is called Nesterov accelerated gradients (Nesterov [1983]) which is a extension to momentum. Since momentum may increase the speed, it follows a direction until the direction changes and so maybe also does not find the global optimum. A better way would be to slow down the update at the right moment to avoid skipping the global optimum. Nesterov accelerated gradients provide this ability by using the current gradients

to approximate the future gradients in the next step. According to this result, it modifies the momentum term before calculating the real next update.

Adagrad (Duchi et al. [2011]) adapts the learning rate differently to each of the the free parameters, decreasing the learning rate faster for parameters with a large gradient, and slower for parameters with a small gradient. For this reason, it is well-suited for dealing with sparse data. So, it provides unique learning rates for all free parameters instead of using the learning rate for all parameters. There are some extensions of this algorithm to solve some numerical issues of Adagrad (radically diminishing learning rates) namely Adadelta (Zeiler [2012]) and RMSprop (Tieleman and Hinton [2012]).

"Adam" (Kingma and Ba [2014]) is a further method that computes adaptive learning rates for each parameter which can be seen as a further development of Adadelta and RMSprop. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction which thus prefers flat minima in the error surface (Ruder [2016]). Kingma and Ba [2014] shows empirically that Adam works well in practice and compares it favorably to other adaptive learning-method algorithms. This algorithm has also two extended versions called AdaMax (Kingma and Ba [2014]) and Nadam (Dozat [2016]). Adamax also concerns itself with some numerical issues with Adam. Since Adam can be seen as a combination of RMSprop and momentum, Nadam incorporates the Nesterov momentum.

Besides the right choice of the optimizer, the right choice of activation functions and net topology is also important. The basic approach would be to use a grid search (Bergstra et al. [2011]), where different combinations of n_{hl} and activation function types are tested and the best performing one is used for the final network. Furthermore, there are genetic algorithms used to evolve the neural network topology (Stanley and Miikkulainen [2002a]) during an optimization process.

Until now all methods have been introduced under the aspect that the input and output is sequential independent from previous steps. Nevertheless, there are sequential problems such as, for example, time series or natural language processing. In this type of problems, the previous input and or output is important to predict the next sequence. Specific types of neural networks can solve this kind of problem, called recurrent neural networks (RNN, Mikolov et al. [2010]) and long short-term memory units ($LSTM$) networks (Hochreiter and Schmidhuber [1997]) which use previous sequences to predict future ones. Recurrent neural networks work in such a way that they introduce additional weights and biases by reusing previous information. Every output from every node can be reused from arbitrary past sequences to predict the future steps. Therefore, these types of networks have a kind of memory. $LSTM$ networks are even more advanced, since information can be stored in, written to, or read from a node, much like data in a computer memory. The node makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplications by sigmoid functions which are all in the range of [0, 1]. Analog data processing has the advantage over digital data processing of being differentiable, and therefore suitable for backpropagation. Those gates act on the data they receive, and similarly to the neural network nodes, they block or pass on information based on its strength and import which they filter with their own sets of weights. These weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process (Nicholson [2019]). Therefore, $LSTM$

networks are especially good if the number of past steps to consider is very high. Recurrent neural networks have a problem with this type of problem due to vanishing gradients. Fig. 2.49 and 2.50 should give an generic idea how they work.

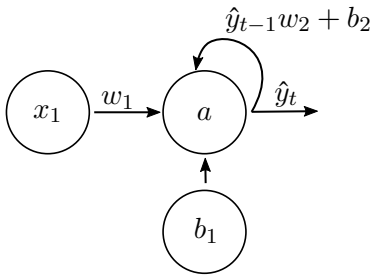


Figure 2.49: Recurrent neural network.

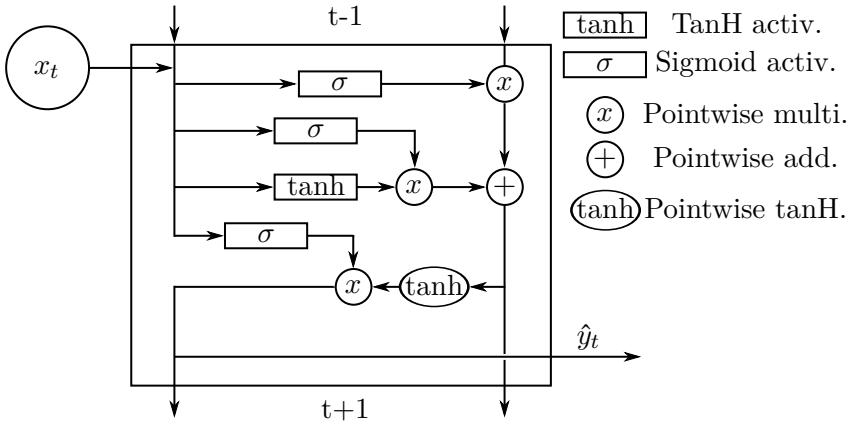


Figure 2.50: *LSTM* network. The activation function is a complex system of gates (sigmoid functions) and mathematical operations. These special activation functions can be chained over the number of sequences which should be considered.

The topic of regularization is also very important and is part of current research. Neural networks tend to over-fit very quickly, especially if the amount of data is small compared to the number of free parameters (complex and large networks). Therefore, there are a lot of different methods to regularize neural networks, like regularization of the weights or the usage of dropout layers (Srivastava et al. [2014]) or dropconnect layers (Wan et al. [2013]). The concept behind these layers is to set a random subset of activation functions output (dropout) or a random subset of weights to zero (dropconnect). These methods both work, because they effectively train several models at the same time and then average across them for testing. Dropconnect is a generalization of dropout because it produces even more possible models, since there are almost always more connections than units. Both methods are intended to prevent "co-adaptation" of units in a neural network. In other words, units should independently extract features from their inputs instead of relying on other neurons to do so. Both methods are depicted in fig. 2.51 and 2.52. A good overview about some classical neural network regularization methods can be found in Orr and Müller [2003].

The last aspect of this subsection is a further problem class, where neural networks are one of the best performing algorithms for, namely image classification. At the moment, so-called convolutional neural networks (CNN) deliver the most promising results on this

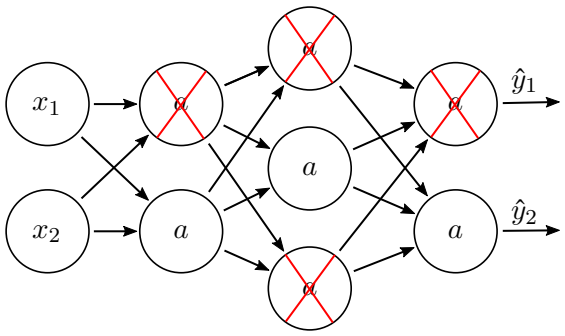


Figure 2.51: Dropout sets random activation functions results to zero.

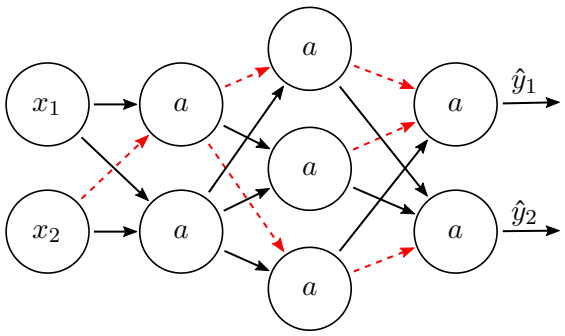


Figure 2.52: Dropconnect sets random weights to zero and provides even more possible network configurations compared to dropout.

type of problems (Krizhevsky et al. [2012]). They are also used for example for video classification (Karpathy et al. [2014]), autonomous driving (Maturana and Scherer [2015]) and visual document analysis (Simard et al. [2003]). CNN use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that were manually-engineered in traditional algorithms. This independence from prior knowledge and human effort in feature design is a major advantage (Banupriya et al. [2020]). So, these types of layers filter the important parts of a picture (like edges and object shapes) and afterwards the filtered image is classified by a standard neuronal network for classification. Both the convolutional layers and the classification layers are learned simultaneously under the same loss function. Fig. 2.53 shows a small neural network for the MNIST (Modified National Institute of Standards and Technology database) (LeCun [1998]) image classification task of handwritten digits with 91.5% accuracy on a test set of 10,000 images. The inputs are the images or more precise the pixel values (here gray-scaled). The first hidden layer is a convolutional layer which performs the first filtering step as shown. Afterwards follows a pooling layer which summarizes quadrants of the previously filtered images to reduce the pixel size and learn only important shapes. Then follows a second convolutional layer to filter even more, and finally follows a standard output layer containing sigmoid activation function for each digit. The result of each activation function of this layer can be seen as the probability for each of these digits. Fig. 2.54 shows another illustration of the filter process for the same example.

Some advantages of deep neural networks are the fast computation which makes them useful for big data applications. In addition to the use of flexible training batch sizes, it

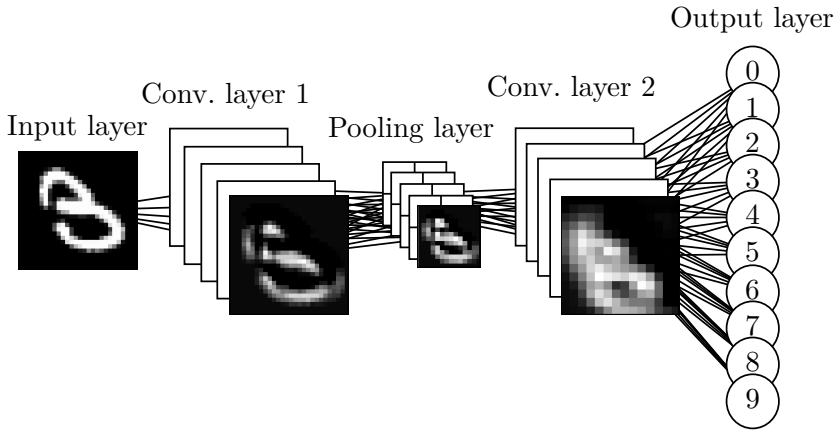


Figure 2.53: Small neural network containing 2 convolutional layers and 1 pooling layer for the MNIST data set of handwritten digits. Accuracy is 91.5% on a 10,000 images test set.

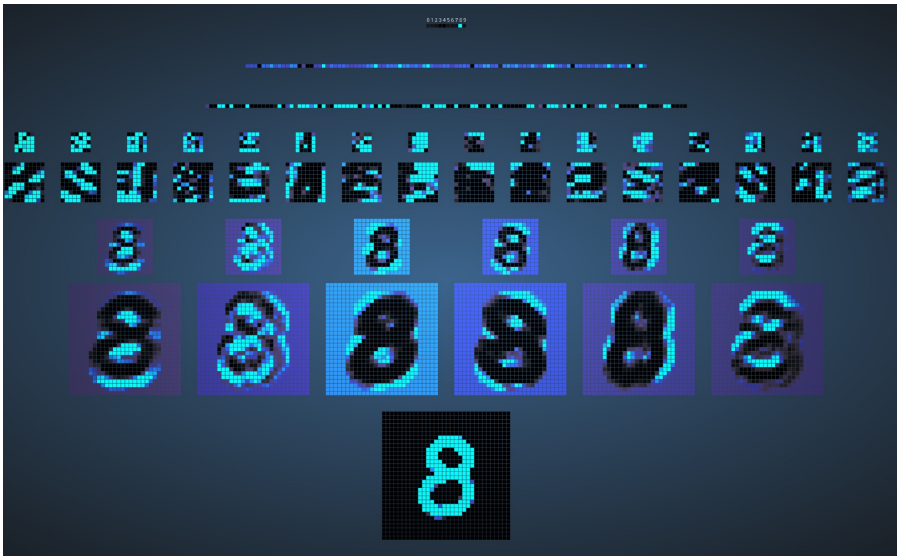


Figure 2.54: Illustration of a CNN which learned from handwritten digits (MNIST data set LeCun [1998]). The image is taken from the visualization tool of Harley [2015].

can be easily used for distributed learning. Drawbacks are the manual choice of the network topology and the type of activation functions. Furthermore, DNN tend to over-fit the data due to the high number of free parameters if no regularization methods are used, like dropout layers or to add additional noise to the data during training (Bishop [2008]). Compared to $\mathcal{GP}$ they tend to have a lower prognosis quality, especially for small number of trainings samples N ($N < 100$) (Bin and Wenlai [2013], Bogoclu and Roos [2016]). Nevertheless, they are one of the most frequently used algorithms on a wide range of problems and also the topic of many recent publications. The hardware industry also develops special computation units, so called tensor processing units (TPUs), for these kinds of algorithms to perform fast tensor algebra (nearly two times faster than actual graphics processing units).

Important notes:

- Deep neural networks are very flexible models which are theoretically able to approximate every function regardless of its complexity.
- Feedforward neural networks have three kinds of layers: input / hidden / output. The hidden layers perform the (non-)linear mapping from the input to the output. The number of hidden layers is a hyperparameter and can be crucial for the data fit.
- Additional hyperparameters are the number of neurons in the hidden layers as well as the type of activation functions. These parameters might also have a huge influence on the data fit.
- Besides the choice of these hyperparameters, it is also important to find the right balance of regularization. Dropout or dropconnect are common methods for regularization of modern neural networks.
- Also the optimization algorithm can have an influence on finding the global minima or not. It is recommended to choose an algorithm like Adam. If no algorithm is used that can adjust its learning rate during the training, then it is suggested that different learning rates are tested.
- Neural networks have several application areas and extensions. They are state of the art for image classification (convolutional neural networks) as well as for sequential dependent problems (recurrent / $LSTM$ neural networks) like time series.

2.10 DEEP GAUSSIAN COVARIANCE NETWORK

2.10.1 *Review of Algorithms*

Before this section starts to describe a novel probabilistic machine learning method called deep Gaussian covariance network ($\mathcal{DGCN}$), the pros and cons of the methods introduced so far should be reviewed. Starting with RSM (polynomials), it was shown that this type of algorithm works quite well if the pattern to be learned is not too complex (linear or slightly non-linear). They are also somewhat robust against outliers (if high order terms are not included), since they average through the observations and are able to provide confidence intervals of the predictions. They also allow extrapolation (assuming no overfitting). Some of their drawbacks are that the user needs to decide about the order of the regressors and which types should be included (e.g. mixed-terms, sine, exponential, ...). The maximum useable number of regressors is dependent on the number of sample points. To overcome some of these drawbacks, the moving least square method was introduced which enhances the flexibility of the RSM by building local models in a smaller subarea defined by an additional radius parameter, without the need to increase the order of the regressors. This flexibility was exchanged for increased training time, since the radius parameter has to be trained via expensive optimization (e.g. via cross-validation) and the extrapolation properties are no longer available. Therefore, compared to the RSM models, the moving

least squares method is advantageous for more complex problems. Nevertheless, they are in many cases less accurate then Gaussian processes (see Krishnamurthy [2005], Bogoclu and Roos [2016]) and also tend to be less robust to outliers then the RSM does. Additionally, a problem would occur if the density of the input points strongly varies without the radius parameter being a function of the input points. Therefore, the Gaussian process seems to be an even better choice, since it has smaller training times compared to moving least squares because no cross-validation is needed. It can achieve a even or higher accuracy (dependent on the problem, Bogoclu and Roos [2016]) and is one of the most flexible models possible. Furthermore, it is a probabilistic model which is also able to provide confidence intervals of its predictions. If the training data is noisy, it is possible to train the $\mathcal{GP}$ model to account for this which is also a unique property. Through the automatic relevance determination, this algorithm can reduce the influence of unimportant parameters (similar to the anisotropic moving least squares). But the Gaussian process also has some limitations. First of all, its complexity $\mathcal{O}(N^3)$ (matrix inversion) makes this algorithm not usable for big data sets (>5,000 - 10,000 depending on memory) without some special methods which most likely lead to decreased accuracy of the model (as explained in subsection 2.8.10). Secondly, the choice of covariance function is crucial to achieve an accurate model. It might even be necessary to build a more complex covariance matrix by using multiple covariance functions as demonstrated in subsection 2.8.5. Furthermore, it was shown that the standard $\mathcal{GP}$ model can be even more flexible by using non-stationary covariance functions which are in most cases difficult to train (e.g. estimating the right subspaces to train local length-scales). One major drawback of the Gaussian process is handled very well by neural networks. The scalability to large data sets via mini-batch training combined with the option of being able to learn online with the income of new training data without having to retrain the whole model. Besides that, the extensions of neural networks like recurrent / $LSTM$ networks and convolutional neural networks extend the field of applications to sequential dependent problems and image classification. Also, multi-class classification is easily possible, whereas Gaussian process is only capable of doing binary classification or needs a work-around like one vs. rest or pairwise comparison (see subsection 2.10.6). Apart from this, neural networks are not favorable over Gaussian process with regard to accuracy, especially for a low amount of training data (Bogoclu and Roos [2016]). Furthermore, finding an optimal net topology in combination with the right activation functions types is even more complex and equally crucial, like choosing the right covariance function for a $\mathcal{GP}$ model. A non-scientific aspect which also favors neural networks is that there are libraries like Tensorflow from Google (Abadi et al. [2015]) that make fast scale distributed training on CPU / GPU with neural networks very easy.

The following table summarizes the capabilities of the mentioned algorithms. Some points might be a question of definition. For example, which algorithm can perform classification. One can argue that it is possible to handle the classification task as a regression problem and round the results, thus all algorithms can do classification. This argument will not be accepted here as having the capability to solve classification tasks. Therefore, some of the criteria have a footnote to give more explanations. Aspects like training speed which is also dependent on the implementation or accuracy are also left out, since they are problem specific. For accuracy comparisons, some benchmark papers were already referenced. All criteria are formulated so that $+$ stands for a positive property the algorithm is capable of, and $-$ means the algorithm does not have this property.

Criteria	RSM	MLS	$\mathcal{GP}$	DNN
Regression	+	+	+	+
Classification[9]	-	-	+	+
Image classification	-	-	-	+
Sequential problems	-	-	-	+
Multi-output (signals)[10]	-	-	+	+
Big data (number of samples)[11]	+	-	-	+
Online learning[12]	-	-	-	+
Confidence intervals	+	+	+	-
Automatic variable selection	+	+	+	+
Trainable noise	-	-	+	-
Robust against outlier	+	-	+	-
No need of uniform distributed sampling	+	-	-	+
Avoid tendency to over-fit[13]	+	-	-	-
Non-stationarity[14]	-	+	+	-
Discrete input parameters[15]	-	-	-	+
No user-defined hyperparameters[16]	-	-	-	-
Reduced heteroscedasticity[17]	-	+	+	-

Table 2.5: Overview of algorithms capabilities.

It should also be mentioned that this table does not claim to be complete, since there might be algorithms that perform better on one or more of these criteria than the investigated ones. It is outside the scope of this work to give an overview for all existing algorithms.

9 Not handled as regression task.
10 The Gaussian process is only able to handle multi-output with a high amount of computational costs as mentioned in subsection 2.8.9.
11 It is a question of definition when big data begins, here it is assumed to have more than 100,000 data points. Gaussian processes are set to − because the existing methods do not work without loss of accuracy due to sparse input matrices (see subsection 2.8.10).
12 Meaning that it is possible to add information e.g. a single data point to the model, without having to retrain the whole model.
13 Every algorithm can over-fit, here it is meant how easy it is to control this behavior.
14 The free model parameters can change dependent on the points to predict.
15 It is possible to use all of them with discrete parameters but especially distance-based algorithms have a problem with discrete parameters.
16 Meaning parameters like the choice of the net topology of a neural network.
17 Automatic reduction of heteroscedasticity of error and / or noise (see 2.3).

Nevertheless, in some of the experiments and benchmarks, results of not explained methods will also be shown. At this point, it would be beneficial if an algorithm would be able to fulfill all these criteria. As the $\mathcal{GP}$ and the DNN have together nearly for all aspects a +, it might be useful to combine their strengths. How this can be achieved will be introduced in the following by presenting the deep Gaussian covariance network ($\mathcal{DGCN}$).

2.10.2 *Mathematical Formulation*

The basic idea of $\mathcal{DGCN}$ is to let a DNN learn the free parameters as well as the covariance function types of a Gaussian process dependent on the point to predict. As it will be demonstrated, this will add several new advantages to the GP. Therefore, first some of the $\mathcal{GP}$ equations should be reviewed again.

For training the $\mathcal{GP}$, the marginal log-likelihood function is

$$\ln p(\boldsymbol{y}|X,\boldsymbol{\theta}) = -0.5\boldsymbol{y}^T(K(\boldsymbol{\theta}))^{-1}\boldsymbol{y} - 0.5\ln|K(\boldsymbol{\theta})| - \frac{N}{2}\ln(2\pi) \tag{2.121}$$

where the covariance matrix $K(\boldsymbol{\theta}) = K(\boldsymbol{l},\sigma_f^2) + \sigma_n^2 I$ is the only term which is dependent on the free parameters $\boldsymbol{l},\sigma_f^2,\sigma_n^2$[18]. Every off-diagonal entry of this matrix is defined by a user-specified covariance function e.g. the squared exponential function (anisotropic case):

$$k_{SE}(\boldsymbol{x},\boldsymbol{x}_*) = \sigma_f^2 exp\left(-0.5\sum_{i=1}^{n}\frac{(x_i - x_{i*})^2}{l_i^2}\right) \tag{2.122}$$

Now for the $\mathcal{DGCN}$ algorithm, the vector of free parameters $\boldsymbol{\theta}$ should be learned by a deep neuronal network. Therefore, eq. 2.121 can be rewritten as

$$\begin{aligned}\ln p(\boldsymbol{y}|X,\hat{Y}_{DNN}(X,W,\boldsymbol{b})) = &-0.5\boldsymbol{y}^T\left(K\left(\hat{Y}_{DNN}(X,W,\boldsymbol{b})\right) + \sigma_n^2 I\right)^{-1}\boldsymbol{y} \\ &- 0.5\ln\left|K\left(\hat{Y}_{DNN}(X,W,\boldsymbol{b})\right) + \sigma_n^2 I\right| - \frac{N}{2}\ln(2\pi)\end{aligned} \tag{2.123}$$

with $\hat{Y}_{DNN}(X,W,\boldsymbol{b})$ being the output of a deep neural network, replacing $\boldsymbol{\theta}$. At the moment, only the learning of the length-scale parameters $\boldsymbol{l}$ will be considered. The optimization of the other parameters will be explained later. Since the output of the DNN depends on the point to predict $\boldsymbol{x}_*$, the free parameters of the $\mathcal{GP}$ now have the same dependency. Therefore, $\hat{Y}_{DNN}(X,W,\boldsymbol{b})$ is a matrix of size $N_b \times n$ ($N_b \in \mathbb{Z}^{[1,\dots,N]}$ batch size, for training) because each input parameter has its own length-scale (automatic relevance determination). Furthermore, the calculation of the covariance functions must be modified because the length-scale is no longer fixed over the number of training samples and is also not equal to the number of cross-distances $(x_i - x_{i*})^2$, $\binom{N_b}{2} \neq N_b$. In the case of the squared exponential function the suggested change is

$$k_{SE}(\boldsymbol{x},\boldsymbol{x}_*) = \sigma_f^2 exp\left(-0.5\sum_{i=1}^{n}(l_i x_i - l_{i*}x_{i*})^2\right) \tag{2.124}$$

The length-scale parameter is now directly incorporated into the input data matrix. Therefore, the role of the length-scale can now be seen as the reciprocal of the previous formu-

18 $\boldsymbol{l}$ is a vector because it contains one l_i per input parameter as explained in subsection 2.8.7.

lation which is dependent on the input x_i and test point x_{i*} (here shown for the squared exponential covariance function):

$$\begin{aligned} \frac{(x_i - x_{i*})^2}{l_i^2} &= (z_i x_i - z_i x_{i*})^2 \\ \frac{(x_i - x_{i*})^2}{l_i^2} &= (x_i - x_{i*})^2 z_i^2 \\ (x_i - x_{i*})^2 &= (x_i - x_{i*})^2 z_i^2 l_i^2 \\ \rightarrow \frac{1}{z_i^2} &= l_i^2 \end{aligned} \tag{2.125}$$

setting the new length-scale formulation as z_i and an equal z_i for x_i and x_{i*}. This means the length-scale must now be small for unimportant parameters and large for important parameters (thus the other way round to the previous description of the $\mathcal{GP}$) and, for example, at discontinuities or non-smooth functions areas, it must also become larger.

The training of this model can also use the stochastic gradient descent algorithm in combination with backpropagation which were introduced in subsection 2.9.2 and subsection 2.9.3. Therefore, the partial derivatives of the likelihood function w.r.t to $W, \boldsymbol{b}$ are needed. Reusing eq. 2.77 from subsection 2.8.3 and replace $\ln p\left(\boldsymbol{y}|X, \hat{Y}_{DNN}(X, W, \boldsymbol{b})\right)$ with $\ln p\left(\boldsymbol{y}|X, W, \boldsymbol{b}\right)$ and $K\left(\hat{Y}_{DNN}(X, W, \boldsymbol{b})\right)$ with $K(W, \boldsymbol{b})$ yields

$$\begin{aligned} &\frac{\partial}{\partial w_{ji}^{[l]}} \ln p(\boldsymbol{y}|X, W, \boldsymbol{b}) = \frac{\partial \ln p\left(\boldsymbol{y}|X, W, \boldsymbol{b}\right)}{\partial K(W, \boldsymbol{b})} \frac{\partial K(W, \boldsymbol{b})}{\partial w_{ji}^{[l]}} \\ &= 0.5 \boldsymbol{y}^T K(W, \boldsymbol{b})^{-1} \frac{\partial}{\partial w_{ji}^{[l]}} K(W, \boldsymbol{b})^{-1} \boldsymbol{y} - 0.5 tr\left(K(W, \boldsymbol{b})^{-1} \frac{\partial K(W, \boldsymbol{b})}{\partial w_{ji}^{[l]}}\right) \end{aligned} \tag{2.126}$$

$$\begin{aligned} &\frac{\partial}{\partial b_{j}^{[l]}} \ln p(\boldsymbol{y}|X, W, \boldsymbol{b}) = \frac{\partial \ln p\left(\boldsymbol{y}|X, W, \boldsymbol{b}\right)}{\partial K(W, \boldsymbol{b})} \frac{\partial K(W, \boldsymbol{b})}{\partial b_{j}^{[l]}} \\ &= 0.5 \boldsymbol{y}^T K(W, \boldsymbol{b})^{-1} \frac{\partial}{\partial b_{j}^{[l]}} K(W, \boldsymbol{b})^{-1} \boldsymbol{y} - 0.5 tr\left(K(W, \boldsymbol{b})^{-1} \frac{\partial K(W, \boldsymbol{b})}{\partial b_{j}^{[l]}}\right) \end{aligned} \tag{2.127}$$

The weights W and biases $\boldsymbol{b}$ are trained in order to maximize the likelihood of the $\mathcal{GP}$.

To use a DNN to estimate the free parameter of the $\mathcal{GP}$ means that the learning part is shifted from the $\mathcal{GP}$ to the DNN. This offers the possibility of using all positive properties of a DNN regarding scalability for large data sets, since mini-batch training is now also applicable for the $\mathcal{GP}$. As mentioned by using a DNN for the parameter learning, the free parameters of the $\mathcal{GP}$ are dependent on the point to predict. This results in a non-stationary $\mathcal{GP}$ without the need to provide a special covariance function for this. Every kind of covariance function automatically becomes non-stationary. This method also has the advantage that compared to the presented method of Plagemann et al. [2008], no subset estimation is necessary, where a second-level Gaussian Process was trained on a subset of the training points to estimate the length-scales. Also, there is no separation between different levels of $\mathcal{GP}$ models and it is easily possible to do this for multiple length-scales

(ARD) without to train several different second level models. As mentioned, can a DNN handle multi-output problems very efficiently.

The next step is to include the noise variance σ_n^2 into the optimization. Therefore, a second independent $DNN_{\sigma_n^2}$ is used to learn this parameter. To distinguish between the neural networks, the network for the length-scale is now called DNN_l. The reason for choosing a second network is that the estimation of the noise variance might need completely different weights and biases $W, \boldsymbol{b}$ and it might be a drawback if both parameters share the same network parameters. As already mentioned for the length-scale parameters, the neural network estimates the free parameter for each point exclusively. Therefore, the noise variance will also be estimated dependent on the point to predict. This allows the model to handle different noise levels within the same data set. With this additional neural network, eq. 2.123 can be written as (with the replacement of $\hat{Y}_{DNN_l}(X, W, \boldsymbol{b}) = \hat{Y}_{DNN_l}$ and $\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X, W, \boldsymbol{b}) = \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}$)

$$\begin{aligned}\ln p\left(\boldsymbol{y}|X, \hat{Y}_{DNN_l}, \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}\right) = &-0.5\boldsymbol{y}^T\left(K\left(\hat{Y}_{DNN_l}\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}} I\right)^{-1}\boldsymbol{y} \\ &- 0.5\ln\left|K\left(\hat{Y}_{DNN_l}\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}} I\right| - \frac{N}{2}\ln(2\pi)\end{aligned} \tag{2.128}$$

The output of the $DNN_{\sigma_n^2}$, $\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}$ is a vector, since only one value per input point is needed as they were added to the diagonal of the covariance matrix. For optimization, eq. 2.126 and eq. 2.127 need to be repeated for each network. For completeness, the two equations 2.67, 2.68 for the prediction of the mean and the variance of a Gaussian process can be rewritten in a more extended form (indicating for which points the neural network parts has to be evaluated for) as

$$\hat{y}_* = \boldsymbol{k}_*\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(X)\right)\left(K\left(\hat{Y}_{DNN_l}(X)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X) I\right)^{-1}\boldsymbol{y} \tag{2.129}$$

$$\begin{aligned}\mathbb{V}\left[\hat{y}_*\right] = &k\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(\boldsymbol{x}_*)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(\boldsymbol{x}_*) - \\ &\boldsymbol{k}_*\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(X)\right)^T \\ &\left(K\left(\hat{Y}_{DNN_l}(X), \hat{Y}_{DNN_l}(X)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X) I\right)^{-1} \\ &\boldsymbol{k}_*\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(X)\right)\end{aligned} \tag{2.130}$$

Note that $\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(\boldsymbol{x}_*)$ has to be added to $k\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(\boldsymbol{x}_*)\right)$ in order to predict the variance correctly. This allows the point-wise noise estimation for new test points.

To show the effect of input dependent noise estimations, the example of heteroscedastic noise of subsection 2.8.8 will be reused. Fig. 2.55 shows the results of the $\mathcal{DGCN}$ model representing the non-stationary noise very well. The 95% confidence interval covers most of the data points and the model is nearly identical to the true function. Compared to the already existing methods (mentioned in subsection 2.8.8) the $\mathcal{DGCN}$ model does not need any subset estimation or expensive extra training of subset models. The noise is just learned along with the length-scales parameter in one single optimization with maximal flexibility. For comparison, fig. 2.56 shows the results without any noise variance training. It can be seen that the model totally over-fits and goes through every noisy training point. Another

example for the non-stationary noise parameter is shown in fig. 2.57. In this example, only a few outliers are present in the training data for a sine test function. It can be seen that the model avoids them automatically and the predicted confidence bounds are very tight in the area without noise. This makes the $\mathcal{DGCN}$ model very robust to outliers in the data. Fig. 2.58 shows a last example for the noise estimation, where the training data is generally polluted by a random noise $y_{noisy} = y + 3(0.5 - \mathcal{U}[0,1])$ again for a sine test function. The $\mathcal{DGCN}$ model is capable to learn this random noise and to cover the true underlying function.

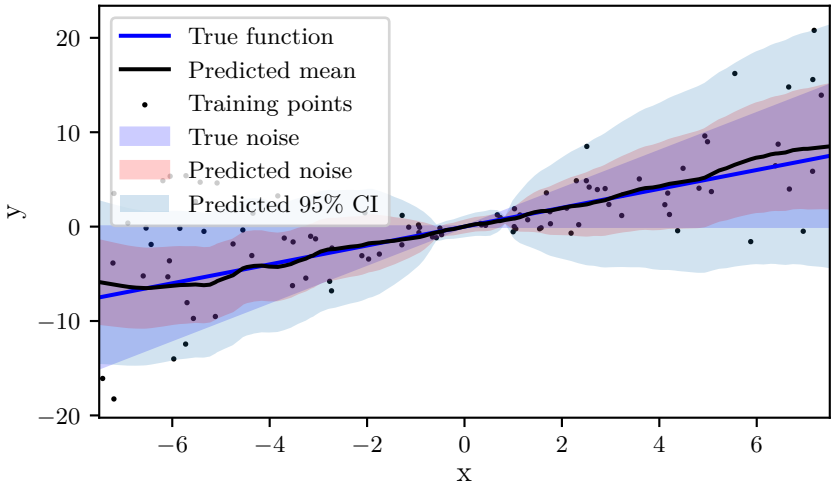


Figure 2.55: $\mathcal{DGCN}$ model with non-stationary noise variance estimation.

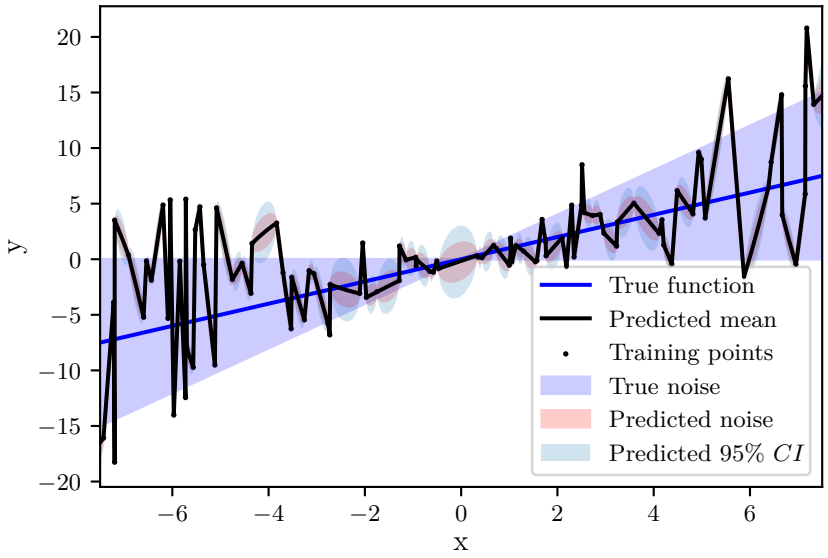


Figure 2.56: $\mathcal{DGCN}$ model without noise variance estimation.

The last parameter to optimize is the signal variance σ_f^2 which proved to be problematic to optimize within this framework. This parameter can not be point dependent, because it

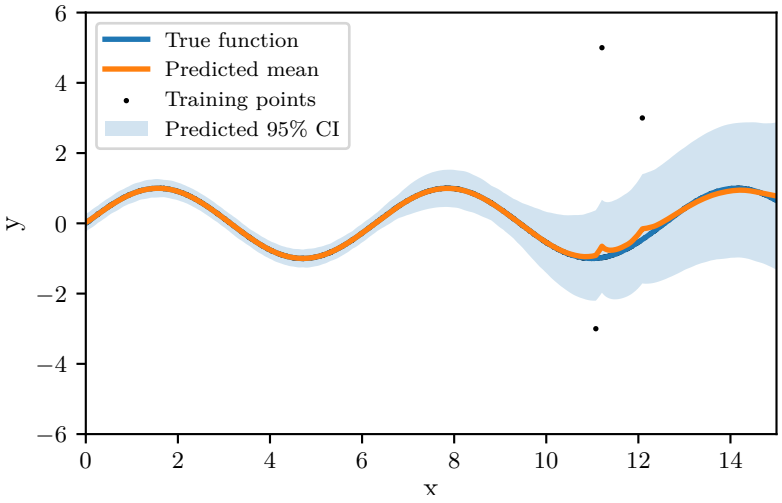


Figure 2.57: $\mathcal{DGCN}$ model for training data with 3 outliers for a sine test function: $y = \sin(x)$, $x \in \mathbb{R}^{[0,15]}$.

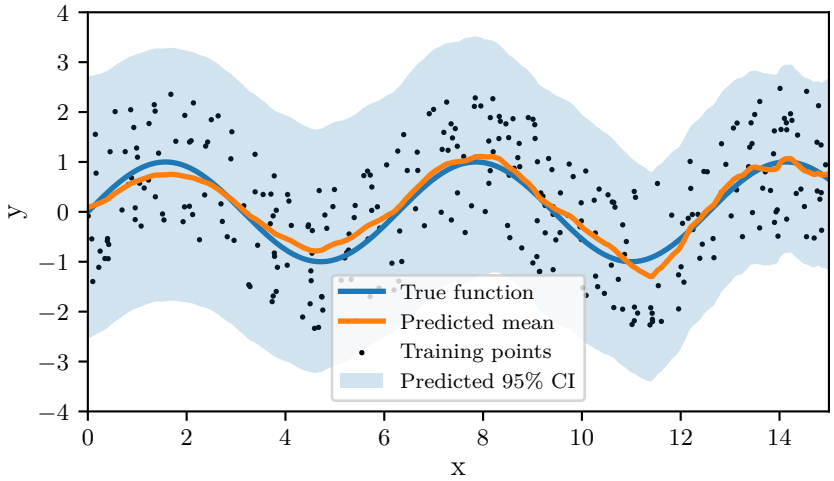


Figure 2.58: $\mathcal{DGCN}$ model for training data with a random noise $y_{noisy} = y + 3(0.5 - \mathcal{U}[0,1])$ for a sine test function: $y = \sin(x)$, $x \in \mathbb{R}^{[0,15]}$.

is multiplied with the covariance matrix of all points and therefore it will be set constant to one in this implementation. This parameter only increases the possible extrapolation properties of the model, since it increases the maximum value of the possible covariance. Therefore, prediction points can be more far away from known training points until the covariance drops to zero and the model predicts only its mean assumption (e.g. zero). If it is desired to have a specific mean, also outside the vicinity of training points, for example, a polynomial mean function for the $\mathcal{GP}$ part can also be used. Then if the covariances drop to zero, the polynomial prediction will be used as the mean of the $\mathcal{GP}$ part. Furthermore, like illustrated previously in fig. 2.32 or fig. 2.33, if the observation are scaled to have unit variance, the optimum for σ_f^2 regarding the log-likelihood is typically also one.

As already introduced in subsection 2.8.5, it is sometimes beneficial to design a covariance matrix of different covariance functions to approximate complex problems. Furthermore, it was also mentioned that the right choice of covariance function is crucial to achieving the best results. Therefore, the $\mathcal{DGCN}$ model is extended to include not only one user chosen covariance function but multiple different ones. In all results of this work the following 5 covariance functions are used (unless stated otherwise): squared exponential, absolute exponential, two Matérn function with $v = 3/2$ and $v = 5/2$ and a rational quadratic function (see subsection 2.8.5 for the corresponding equations). Since the optimization via the neural network is very fast and scalable, a large amount of numerical calculations is not needed to include multiple covariance functions. The resulting covariance matrices are superimposed at the end via a summation. Other operators to handle covariance matrices would also be possible (see subsection 2.8.5), but in this work the summation operator will be used, as it shows good results. So, the choice of the right covariance function or maybe a summation of multiple functions is done by the DNN_l network. By setting the length-scale parameters to zero, it can deactivate certain covariance functions. It would also be possible to activate and deactivate covariance functions over the input space, since the correlation length-scales are non-stationary. Therefore, the size of the output of this network extends to $N_b \times n \times n_c$, with n_c representing the number of covariance functions used ($n \times n_c$ neurons in the output layer of the DNN_l). So, the inclusion of multiple covariance function is made by adding additional nodes in the output layer of the DNN_l network, where each covariance function has its own free length-scale parameters. The possible optimized noise variance is added to the final superimposed covariance matrix diagonal. Therefore, eq. 2.128, 2.129 and 2.130 change to

$$\begin{aligned}\ln p\left(\boldsymbol{y}|X,\hat{Y}_{DNN_l},\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}\right) = &-0.5\boldsymbol{y}^T\left(\sum_{i=1}^{n_c} K_i\left(\hat{Y}_{DNN_l}\right)+\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}I\right)^{-1}\boldsymbol{y}\\ &-0.5\ln\left|\sum_{i=1}^{n_c} K_i\left(\hat{Y}_{DNN_l}\right)+\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}I\right|-\frac{N}{2}\ln(2\pi)\end{aligned} \tag{2.131}$$

$$\begin{aligned}\hat{y}_* = &\sum_{i=1}^{n_c}\boldsymbol{k}_{i*}\left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*),\hat{Y}_{DNN_l}(X)\right)\\ &\left(\sum_{i=1}^{n_c} K_i\left(\hat{Y}_{DNN_l}(X),\hat{Y}_{DNN_l}(X)\right)+\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X)I\right)^{-1}\boldsymbol{y}\end{aligned} \tag{2.132}$$

$$\begin{aligned}\mathbb{V}\left[\hat{y}_*\right] = &\sum_{i=1}^{n_c} k_i\left(\hat{Y}_{DNN_l}\left(\boldsymbol{x}_*\right), \hat{Y}_{DNN_l}\left(\boldsymbol{x}_*\right)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}\left(\boldsymbol{x}_*\right) \\ &- \left(\sum_{i=1}^{n_c} \boldsymbol{k}_{i*}\left(\hat{Y}_{DNN_l}\left(\boldsymbol{x}_*\right), \hat{Y}_{DNN_l}\left(X\right)\right)\right)^T \\ &\left(\sum_{i=1}^{n_c} K_i\left(\hat{Y}_{DNN_l}\left(X\right), \hat{Y}_{DNN_l}\left(X\right)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X) I\right)^{-1} \\ &\sum_{i=1}^{n_c} \boldsymbol{k}_{i*}\left(\hat{Y}_{DNN_l}\left(\boldsymbol{x}_*\right), \hat{Y}_{DNN_l}\left(X\right)\right)\end{aligned} \tag{2.133}$$

As a first example of the local length-scale estimation of multiple covariance functions, a similar test as shown previously in fig. 2.38 is carried out for a sinusoidal function with four discontinuities at $x = -0.6$, $x = -0.3$, $x = 0.3$, $x = 0.6$ depicted in fig 2.59. It shows the $\mathcal{DGCN}$ model and a standard stationary $\mathcal{GP}$ model trained on the test-function mentioned. Since the length-scales are not as intuitive as before, because of the usage of multiple covariance functions, it is more difficult to interpret their values. In fig. 2.60 are the values of the mentioned 5 different covariance functions length-scales shown from the $\mathcal{DGCN}$ model (corresponding to the model of fig 2.59) which are all trained simultaneously resulting in 5 different covariance matrices. It can be seen that the length-scale values get higher in the area of discontinuities which agrees with the explained reciprocal value of the length-scale in the $\mathcal{DGCN}$ method (see eq. 2.125). Additionally, in fig. 2.61 and 2.62 the covariance matrices between 100 training and 100 test points are shown, to demonstrate the effect of changing length-scales on the covariance matrix. It can be seen that the $\mathcal{DGCN}$ model has a smaller bandwidth of high covariance values for the areas with the discontinuities and a larger one for the middle part. The standard GP on the other hand has, overall, very small covariance values in order to capture the discontinuities as well. This results in much higher confidence intervals for this model due to over-fitting which can also be seen from the fidgety course of the approximation curve.

The local estimation of length-scales is not only beneficial for discontinuities. If the training data is more densely sampled (e.g. during a local adaptation of the training database for optimization), it can be shown that the local length-scale estimation allows the $\mathcal{DGCN}$ model to be superior to a standard $\mathcal{GP}$ model with a stationary covariance function. To demonstrate this, the modified Branin-Hoo test function (eq. 2.12) of subsection 2.2.1 is used. For the training, 30 samples were globally sampled via random uniform sampling in the input space of $x_1 \in \mathbb{R}^{[-5,10]}$, $x_2 \in \mathbb{R}^{[0,15]}$ and additional 20 points were sampled in the area of the global optimum $x_1 \in \mathbb{R}^{[-5,-3]}$, $x_2 \in \mathbb{R}^{[13,15]}$ also via random uniform sampling. Globally means in this context that the distance between the sample points is in the global area higher than in the local area. Then a $\mathcal{DGCN}$ and $\mathcal{GP}$ (with squared exponential covariance function) model are trained on the data. Additional 20 global and 20 local test points in the same bounds as the training samples were created to evaluate these models. In fig. 2.63 and fig. 2.64 the surfaces of both trained models are shown. It can be seen that the $\mathcal{DGCN}$ model fits the true function much better. The reason for this can be found in fig. 2.65 and 2.66 which present the covariance matrices for the test points of both models. They show that the area of the local sample points (between test and training points) have high covariance values. The $\mathcal{GP}$ model has, therefore, a very small correlation length-scale to capture this area, but this also results in low covariance values in the global area which

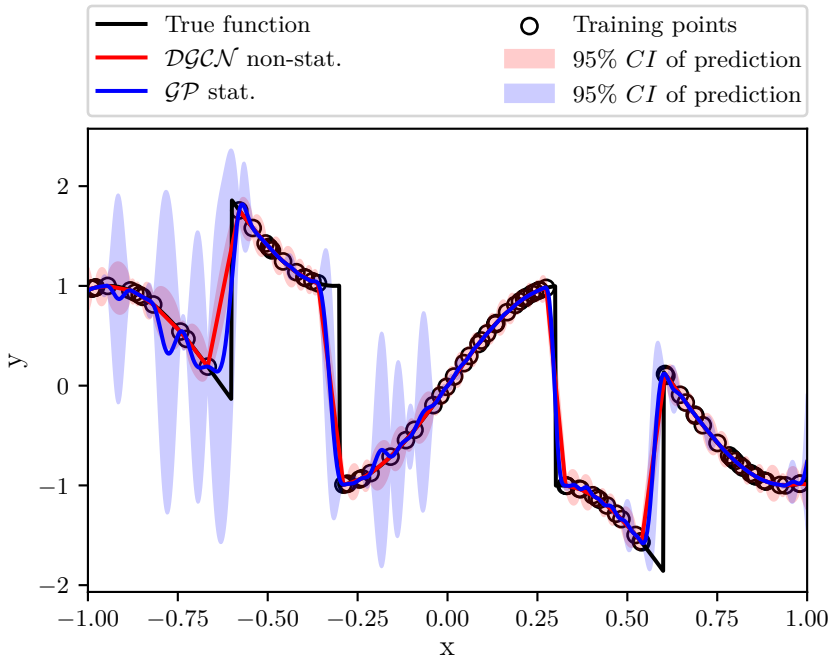


Figure 2.59: Comparison of the non-stationary $\mathcal{DGCN}$ model and a stationary $\mathcal{GP}$ model with squared exponential covariance function.

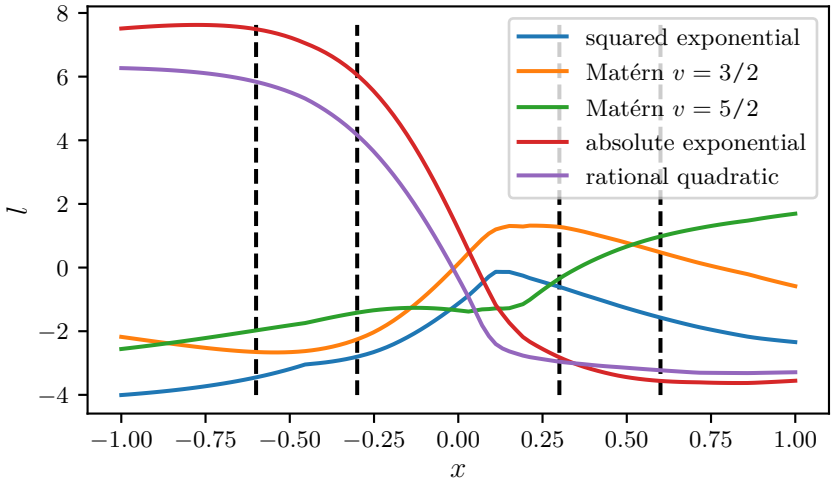


Figure 2.60: Plot of the length-scale parameters of 5 different covariance functions. It can be seen that all of them, predicted by the neural network, change with the value of x.

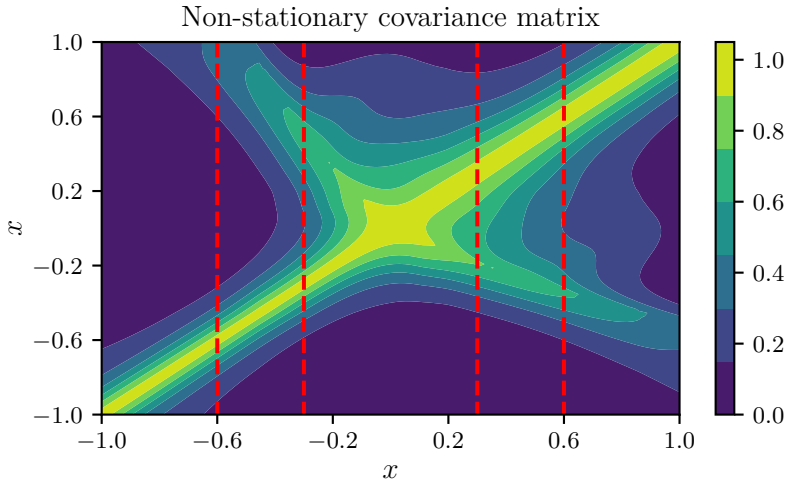


Figure 2.61: Non-stationary $\mathcal{DGCN}$ covariance matrix. The red dashed lines mark where the function makes a step.

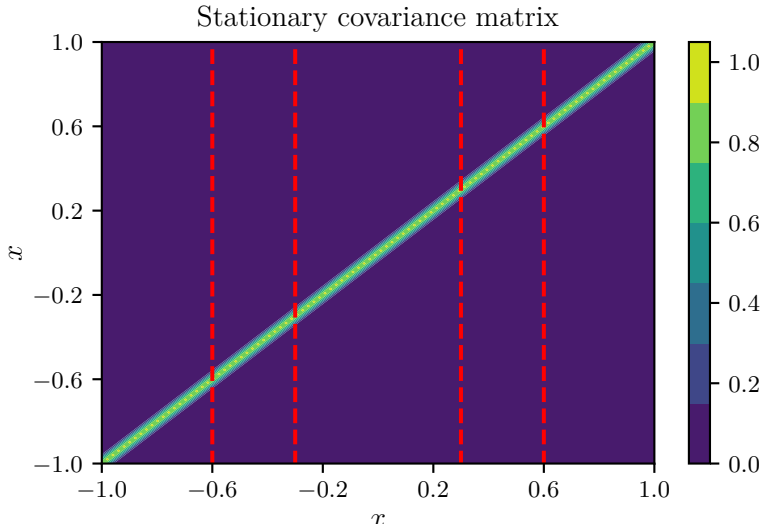


Figure 2.62: Stationary $\mathcal{GP}$ covariance matrix with squared exponential covariance function. The red dashed lines mark where the function makes a step.

is not the case for the $\mathcal{DGCN}$ model. For the $\mathcal{DGCN}$ model the correlation length-scales for both variables x_1, x_2 are plotted in fig. 2.67. It shows that the length-scales vary for the test points globally sampled and then become nearly constantly high for the local points. Also, it can be seen that the correlation length-scales of the absolute exponential and the Matérn $v = 3/2$ covariance functions are set to zero for both variables (also fitting to the reciprocal meaning of the length-scale) which demonstrate that the model can decide by itself which covariance functions to use. In this example, the $\mathcal{DGCN}$ model decided to use only smooth covariance functions, which makes sense, since the function itself is also very smooth. Furthermore, it can be seen that for the input parameter x_2 one more covariance function is used compared to x_1. Additionally, the $RMSE$ values for the test points are listed in tab. 2.6. The results are divided into three groups, namely the global, local and combined points for the error on each group. The values show that the $\mathcal{GP}$ model has low error values on the test points in the locally adapted area, but high error values for the points in the remaining global area. The reason for this is due to the small correlation length-scales in the global area. The $\mathcal{DGCN}$ model has low error values for both areas because of the non-stationary length-scale parameters.

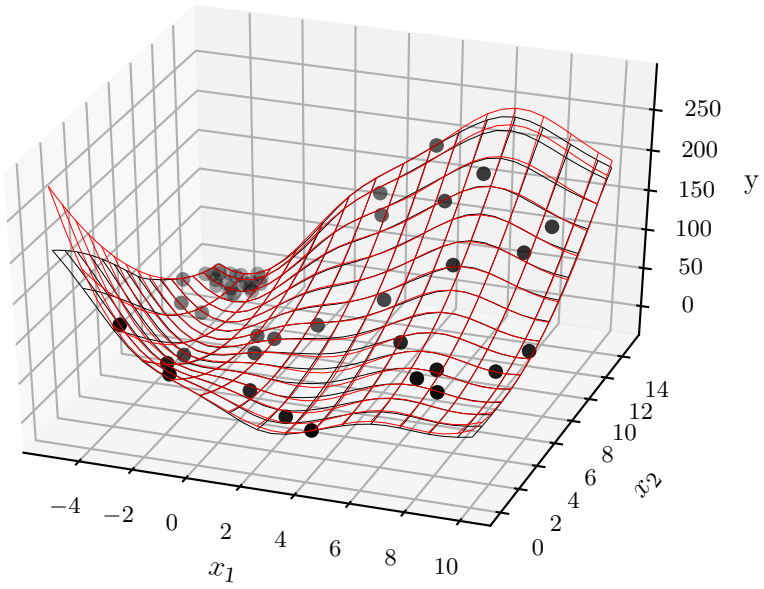


Figure 2.63: Comparison of $\mathcal{DGCN}$ model surface over x_1, x_2 (black wireframe) and the true function (red wireframe). Black points indicate training points which consists of a global and local sampling.

As a further example to show the automatic generation of a superimposed covariance matrix and also the influence of the missing optimization of the signal variance σ_f^2, the example of subsection 2.8.5, the atmospheric concentration of CO_2 collected at the Mauna Loa observatory, Hawaii Keeling [1994] should be repeated with the $\mathcal{DGCN}$ model. The previous results shown in fig. 2.37 were gained by manually constructing a covariance matrix with the help of physical considerations, so they were not automatically derived. In comparison to that, fig. 2.68 shows the result of the $\mathcal{DGCN}$ model given the choice of 5 covariance functions, namely: squared exponential, absolute exponential, two Matérn

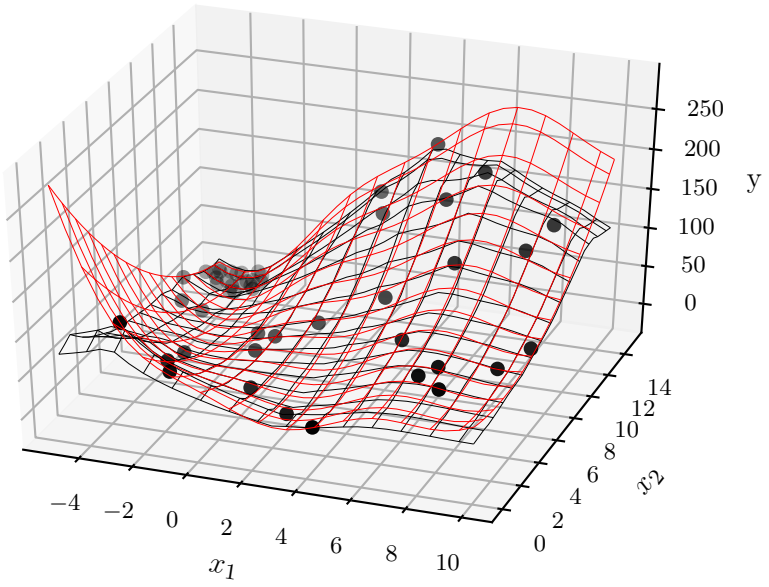


Figure 2.64: Comparison of $\mathcal{GP}$ model function surface with squared exponential covariance over x_1, x_2 (black wireframe) and the true function (red wireframe). Black points indicate training points which consists of a global and local sampling.

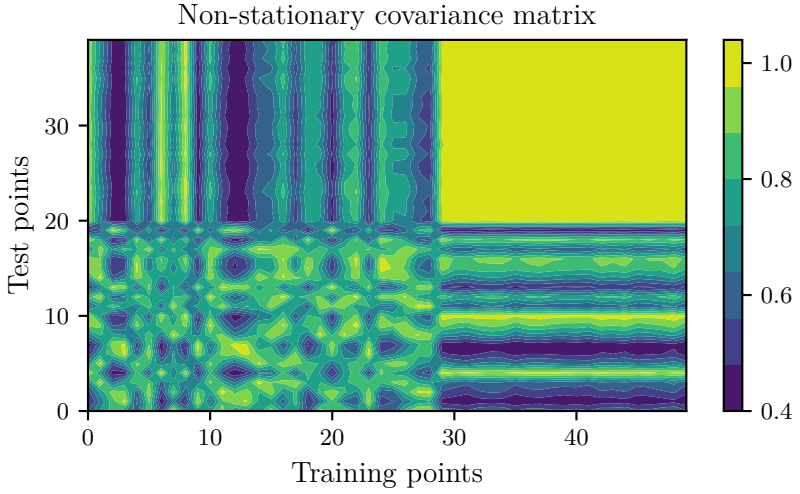


Figure 2.65: Covariance matrix of the non-stationary $\mathcal{DGCN}$ model for 50 training points (30 globally sampled (0-30 on the x-axis) and 20 locally sampled (30-50 on the x-axis)) and 40 test points (20 globally sampled (0-20 on the y-axis) and 20 locally sampled (20-40 on the y-axis)).

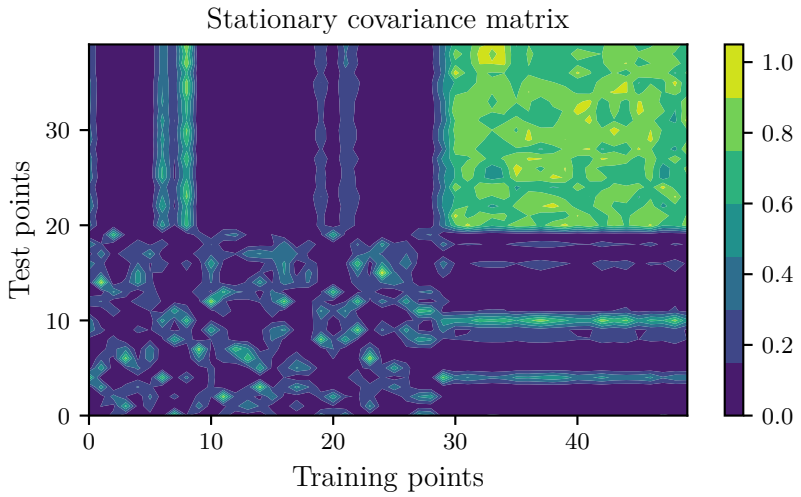


Figure 2.66: Covariance matrix of the stationary $\mathcal{GP}$ model for 50 training points (30 globally sampled (0-30 on the x-axis) and 20 locally sampled (30-50 on the x-axis)) and 40 test points (20 globally sampled (0-20 on the y-axis) and 20 locally sampled (20-40 on the y-axis)).

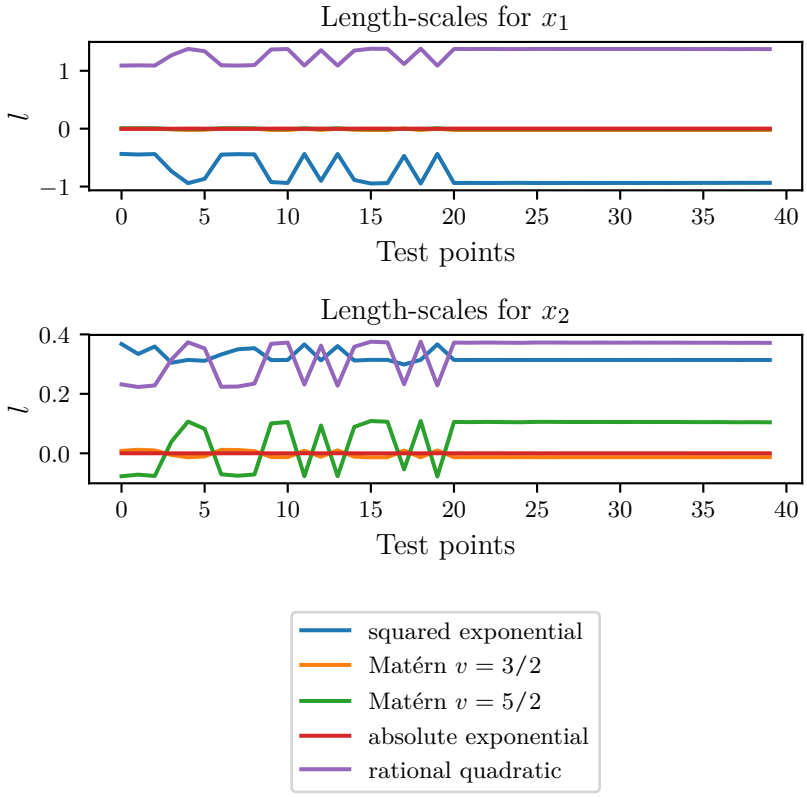


Figure 2.67: Length-scales of 5 different covariance functions used in the $\mathcal{DGCN}$ model over the 40 test points. Unwanted correlation functions: absolute exponential and Matérn $v = 3/2$ are deselected by setting the correlation length to zero. Since this function is very smooth, it can be better approximated by smooth covariance functions.

Sample group	$\mathcal{DGCN}$ $RMSE$	$\mathcal{GP}$ $RMSE$
Local	0.01	0.56
Global	0.70	13.69
Combined	0.49	9.69

Table 2.6: $RMSE$ for the adaptive sampling example for the global, local and combined test points.

function with $v = 3/2$ and $v = 5/2$ and one periodic function instead of the rational quadratic function. It is obvious that the signal begins to decrease if it extrapolates over its maximum training year of 1998. This is the effect of the missing signal variance σ_f^2 optimization. Without setting a higher signal variance, the Gaussian process prediction tends to predict the mean value of the training observations much faster. If the prediction of the model shown in fig. 2.37 would be extrapolated further into the future, it would also decrease to its mean at some point. Nevertheless, the signal itself, despite the fact that it is not using a manually introduced physical input, looks quite similar to the manually created one shown before in fig. 2.37.

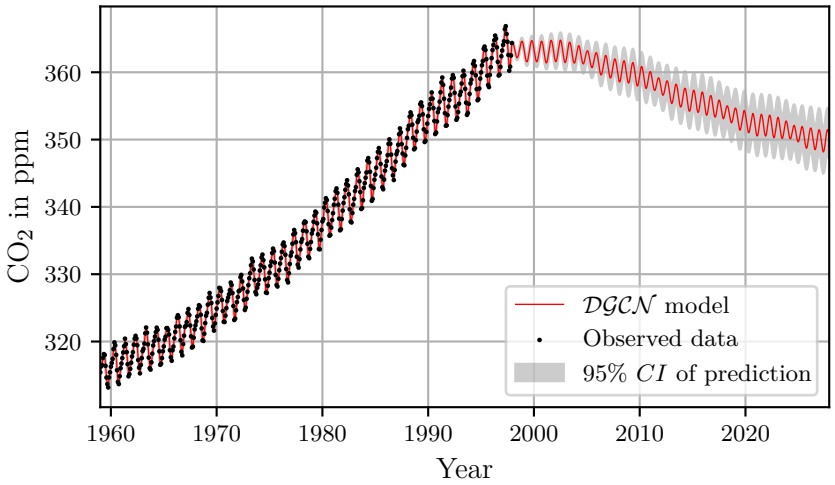


Figure 2.68: $\mathcal{DGCN}$ model for the CO_2 example with automatically build covariance matrix.

After explaining the full framework of the $\mathcal{DGCN}$ model, fig. 2.69 shows a schematic overview of the networks used DNN_l and $DNN_{\sigma_n^2}$ in combination with the multiple covariance functions and the final $\mathcal{GP}$ part. Since the networks only deliver the free parameters, there are also connections directly between the input and the covariance functions which are needed to build the covariance matrices (for the distance calculation).

Lastly, also a drawback of the current implementation of $\mathcal{DGCN}$ should be mentioned. If mini-batch training is used to handle large data sets. The prediction of new points become more time consuming, because the inverted covariance matrix between the training points as well as the inverted covariance matrix between training points and test points is needed (see e.g. eq. 2.132). The calculation of the inverted covariance matrix is numerically too demanding if the data size is too big. To solve this problem, a nearest neighbor algorithm

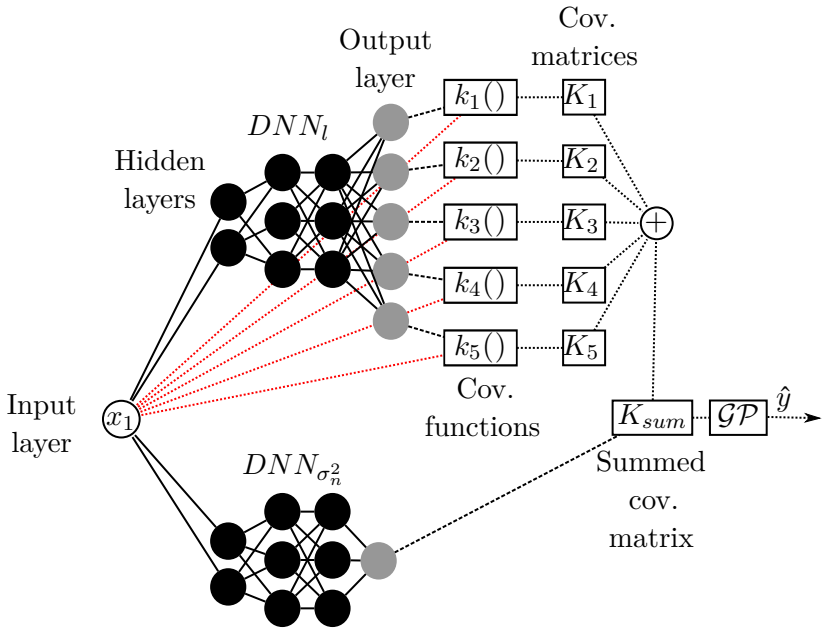


Figure 2.69: Schematic $\mathcal{DGCN}$ model overview with only one input parameter (for simplicity). The solid lines illustrate connections with trainable $W, \boldsymbol{b}$ parameters and the dashed lines are intended to illustrate the flow of processing the data.

(Cover and Hart [1967]) is utilized, where the N_b nearest points to $\boldsymbol{x}_*$ of X are searched, to predict a test point $\boldsymbol{x}_*$ instead of the full size of X. This means also that the prediction of multiple test points X_* can not be performed in parallel via matrix / vector multiplications. This increases the prediction time drastically if billions of points are predicted at the same time. To reduce this time increase, a parallel prediction is implemented which utilizes (optionally) all CPU cores. The idea behind this approach is that typically the point to predict is only influenced by a small number of observed points in the region around it. For the training, it is not necessary that the points within a batch are coming all from the same neighborhood, since the neural network part only needs to learn the prediction of the length-scales and the noise parameter and no prediction result is needed. Another solution would be to also make the prediction via batches, so that X is divided to N_b batches (where N_b for the prediction must no be the same as for the training) and all N_b batches of X are used to predict the test point $\boldsymbol{x}_*$ individually. Then instead of having one prediction result $\hat{y}_*$ for one single test point $\boldsymbol{x}_*$, there are N_b prediction results. To get a final result, the N_b batches might be averaged for example. This method would allow also to have a much faster prediction but might also result in lower prediction accuracy because the training points used inside these batches might not be optimal like in the nearest neighbor approach for given chosen test point.

2.10.3 *Implementation Details and Scalability*

Some more details about the implementation, the setup of the deep neuronal network and the covariance function shall be given in the following. This setup will be used for all following comparisons and was also used for most of the previous ones.

The network topology for the DNN_l is:

- Input layer
- Gaussian noise layer which adds additive zero-centered Gaussian noise for regularization (only active during training).
- Hidden layer with 20 neurons with sigmoid activation function.
- Batch normalization layer (Ioffe and Szegedy [2015]) which normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to zero and the activation standard deviation close to one. Additionally, there are two extra parameters in these types of layers which are learned along with the other free parameters of the neural network. These parameters are a scaling and offset parameter which control a possible useful normalization of the data. Therefore, even if a batch normalization layer is used, the neural network can decide if this normalization should be done or if information will get lost through this operation.
- Alpha dropout layer: Alpha dropout is a "dropout" that keeps mean and variance of inputs to their original values, in order to ensure the self-normalizing property even after this dropout (only active during training).
- Hidden layer with 20 neurons with sigmoid activation function.
- Batch normalization layer.
- Alpha dropout layer.
- Hidden layer with 20 neurons with $ReLU$ activation function.
- Batch normalization layer.
- Output layer with linear activation function.

The network topology for the $DNN_{\sigma_n^2}$ is:

- Input layer
- Batch normalization layer.
- Hidden layer with 20 neurons with $ReLU$ activation function. Here only $ReLU$ functions are used, to ensure positive values for the noise variance.
- Batch normalization layer.
- Hidden layer with 20 neurons with $ReLU$ activation function.
- Batch normalization layer.
- Output layer with $ReLU$ activation function.

These network topologies appeared to work very well on many problems tested. Nevertheless, the implementation also contains a grid search (Bergstra et al. [2011]) to improve the network topologies if required.

The networks contain several methods for regularization. This turned out to be a crucial part in obtaining good results, since this framework is very powerful and tends to over-fit very quickly, due to the high number of free parameters. Additionally, there is a stopping criterion implemented that stops the training if a specified improvement of the likelihood function is not gained over a specific number of epochs. Early stoppage is a method frequently used to regularize neural networks too (Prechelt [1998], Caruana et al. [2001]), since it prevents the neural network from memorizing or perfectly fitting the training data which sometimes leads to poor generalization which is reflected in poor predictions of new test data.

The optimization algorithm used, is the Nadam algorithm mentioned in subsection 2.9.4 (Dozat [2016]) because it obtains many beneficial properties of modern neural network optimization algorithms as explained in subsection 2.9.4. To improve its optimization capabilities further, a control mechanism is used during the optimization which reduces the learning rate to 50% of its current value if the result does not improve over 10 epochs.

For the construction of the superimposed covariance matrix, the already mentioned 5 covariance functions are used:

- Squared exponential
- Absolute exponential
- Two Matérn kernels with $v = 3/2$ and $v = 5/2$
- Rational quadratic

At this stage there are no hyperparameters to be defined by the user, except if noise should be trained or not. This makes this algorithm very attractive for people with lower expertise in machine learning. Some of the extensions presented in the following, have some additional options which might have an influence on the predictive accuracy of the model but are not needed to be set by the user to obtain good results.

One objective of the development was to combine the scalability of a neural network with a Gaussian process, therefore the training speed and scalability should be compared for a batch training size with $N_b = N$ and $N_b = 400$. Hence, different training sample sizes are used from 400 up to $52,428,800$ points. The batch training size with $N_b = N$ will only range from 400 to $12,800$ samples, since the memory demand exceeds 16 GB (64-bit) and on GPU only up to $6,400$ samples, since it has only 11 GB ram which also demonstrate the limitations of a standard Gaussian process model. The batch training with $N_b = 400$ will start from 400 training samples. A second test scenario is used which represents an image classifications task (see subsection 2.10.7) for a picture set with 28×28 pixel images, so 764 input parameters. For this kind of task, an additional network is conducted as explained in subsection 2.10.7 which increases the numerical effort. The number of input parameters for the standard regression task is set to $n = 10$. Furthermore, the number of training epochs is fixed to 100 which is sufficient to get a good model in most cases, especially as the more training samples are available the less epochs are needed. The test was performed on an Intel Core i7 8700K CPU with 6 cores at 5.0 GHz. For the GPU, a Nvidia GTX 1080 Ti graphics card was used with $3,584$ CUDA cores which represents modern hardware specs. The implementation was programmed in Python 3.7 with the libraries Keras 2.1.4 (Chollet et al. [2015]) and Tensorflow 1.14 (Abadi et al. [2015]) as its backend. The results are shown in fig. 2.70. Depicted here is the fact that the use of lower batch sizes N_b is faster to

calculate at a specific point compared to setting $N_b = N$. Even more important is the fact that the lower batch size enables the $\mathcal{GP}$ part of the $\mathcal{DGCN}$ to handle any size of training samples N. The smaller sample sizes have almost no training time but the compiling time of Tensorflow[19] is also included and therefore all have nearly the same value. After a specific number of samples, the training time doubles together with the sample size (log-log scale). It can also be seen that if the numerical effort increases, as demonstrated with the image classification task, the GPU calculations are much faster compared to the CPU. So, the use of a GPU is only useful if the numerical effort is high (under the assumptions of using a modern CPU and GPU). Additionally, compared to all the methods of making Gaussian processes able to handle big data already mentioned (see subsection 2.8.10), this method works without discarding any sample information. Therefore, there is no trade-off between performance and accuracy.

The next subsections demonstrates the extension capabilities of the $\mathcal{DGCN}$ algorithm.

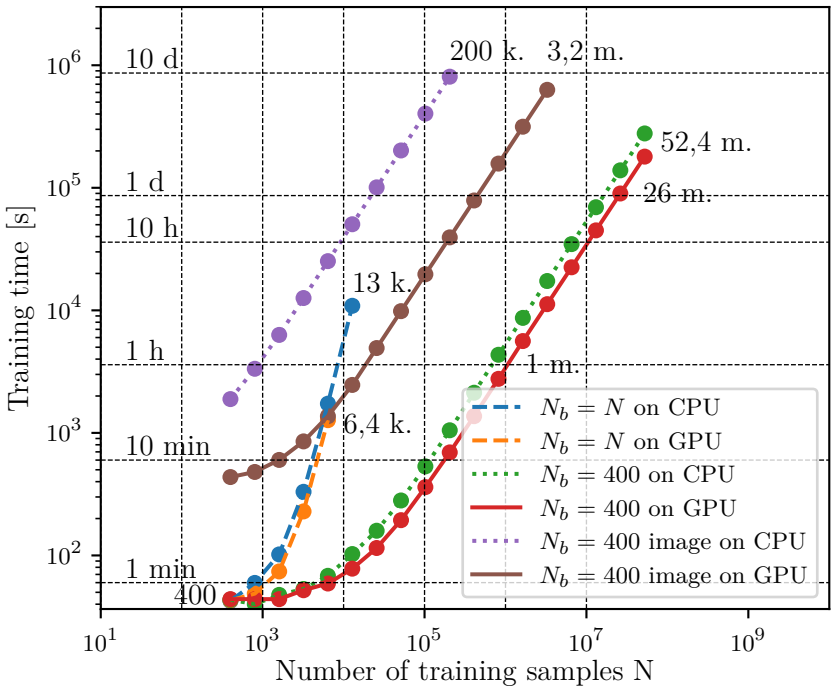


Figure 2.70: Training time comparison for different batch sizes N_b on CPU and GPU.

2.10.4 *Sequential Problems*

Recurrent neural networks or long short-term memory ($LSTM$) networks are common approaches to building models for sequential dependent outputs, as mentioned in subsection 2.9.4. One advantage of such networks is the internal use of additional parameters to train the sequential dependencies. A simple approach to emulating this behavior, is to create artificial input parameters which represent the input or output of previous sequential steps. For example, for a time series the last two output results y_{t-1}, y_{t-2} can be used as

19 Tensorflow builds the computational graph and uses the programming language C++ to boost the calculation performance.

additional input parameters with their own free parameters in the $\mathcal{DGCN}$ model. Therefore, an additional pre-processing step of the training data is necessary which shifts the data appropriately, as shown in tab. 2.7. This shifting allows the model to learn from previous sequential steps as well as predicting multiple steps into the future. Subsection 2.10.8 will also show some comparisons of recurrent networks to the presented framework. With the definition of additional input parameters, the DNN_l and the $DNN_{\sigma_n^2}$ network will learn the sequential behavior over time, similar to a recurrent network, but the approximation is still based on the $\mathcal{GP}$ part. Furthermore, this time dependency is non-stationary dependent on $\boldsymbol{x}_*$ which might result in a significant improvement compared to stationary sequential dependencies, since the algorithm can decide if it takes the fully specified past sequential steps as information (dependent for example on the length-scale) or only a part of them dependent on the point to predict. For this approach, the user has to define the number of previous sequential steps from input and / or output to use for training. This is a kind of hyperparameter, since the number of previous steps has an influence of the predictive power of the model. A possible automatic approach to estimate this parameter would be the use of an auto-correlation analysis (Dunn and Davis [2017]) which analyzes the correlation of a variable to itself over previous steps.

Time series			Input		Output		
t	y_t		x_{t-2}	x_{t-1}	y_t	y_{t+1}	y_{t+2}
1	2		-	-	-	-	-
2	3		-	-	-	-	-
3	1		2	3	1	6	7
4	6	→	3	1	6	7	3
5	7		1	6	7	3	9
6	3		6	7	3	9	1
7	9		-	-	-	-	-
8	1		-	-	-	-	-

Table 2.7: Shifted time series for supervised learning.

If a model is trained for sequential dependent data, e.g. a time series, where two previous output time steps are also used as input, it is important to provide the model with these two previous outputs to enable the model to predict the next time steps as accurate as possible. If they are not provided, the model must make assumptions of the missing parameters until it can fill them with its own predictions. This possibly results in a poor prediction of the first few steps. For the $\mathcal{DGCN}$ model, missing previous observation values are assumed to be zero which is after scaling the signal mean value. To show this behavior and to give some general examples of this extension, the first test function taken from Narendra and Parthasarathy [1990] is

$$y_{t+1} = \frac{y_t y_{t-1} y_{t-2} x_{t-1}(y_{t-2} - 1) + x_t}{1 + y_{t-1}^2 + y_{t-2}^2} \tag{2.134}$$

which represents a dynamic system, where the output for time step $t+1$ depends on the actual and last two output values y_t, y_{t-1}, y_{t-2} and the actual as well as the delayed input x_t, x_{t-1}. To create the input vector $\boldsymbol{x}$ for training and testing, an amplitude modulated pseudo random binary sequence (Fang and Shenton [2014]) is used. For training and testing $N = 500$ samples are used. The $\mathcal{DGCN}$ model is trained with a shift of 3 for the input and output data[20] corresponding to the test function. Fig. 2.71 depicts the results. Since the Gaussian process part makes the prediction and no noise variance parameter is trained, the interpolation capabilities remain also for sequential dependent problems. It can also be seen that the test set is covered very well. For the prediction of the test points, the model gets the values of x but not the values of y. For this, the model uses its own predictions of the past steps as input for the future ones. This is like a continuously rolling input, where the oldest values are dropped, and the last prediction is appended during each prediction step. Additionally, in fig. 2.72 the same example is given with a shift of 2 and 1 for the training of the $\mathcal{DGCN}$ model. The results get worse if the shift number decreases as the $RMSE$ value indicates. In fig. 2.73, the influence of giving the model not the the start points corresponding to the shift used, is demonstrated. In this case if not the 3 first results for x_t, x_{t-1}, x_{t-2} and y_t, y_{t-1}, y_{t-2} are given as start points, the model assumes that they are the signal / input mean and therefore it takes some time until it fits the test data well.

For another example, data from a compressed air storage system is used, taken from Atabay [2017]. Fig. 2.74 illustrates the system. If the binary input charge is 1, the booster is running and charges the storage to a maximum pressure of 38 bar. If the binary input discharge is 1, the booster and the compressor are off and the air demand is covered by the air in the storage which is discharged. If the storage is neither charged nor discharged, the compressor covers the air demand. The model is trained to calculate the state of the pressure in the storage and the electric power consumption of the booster and the compressor. This example compares the $\mathcal{DGCN}$ model against a recurrent neural network. For both models only 1 time step back of the input and output is used as additional input and both models are trained for 100 epochs. The recurrent neural network has two hidden layers with 5 neurons each which was supposed by the original example (Atabay [2017]). The comparison for both outputs are depicted in fig. 2.75. There are 480 training and 480 test samples, where the $RMSE$ is calculated for. For the storage pressure and for the electrical power consumption, $\mathcal{DGCN}$ has nearly a 50% lower $RMSE$ compared to the recurrent neural network. These examples give a first impression that the simple shifting method enables $\mathcal{DGCN}$ to handle sequential problems equally or better than recurrent neural networks. More challenging examples will be shown in the subsection 2.10.8.

2.10.5 *Multi-Output*

This subsection will describe how $\mathcal{DGCN}$ can handle multiple correlated outputs within one model. This topic was already briefly mentioned for the Gaussian process in subsection 2.8.9. The idea is to use a similar approach neural networks use for multi-output problems. For each additional output, the output layer of the neural network part of $\mathcal{DGCN}$

20 The implementation only allows equal shifts for input and output or one of them at this moment.

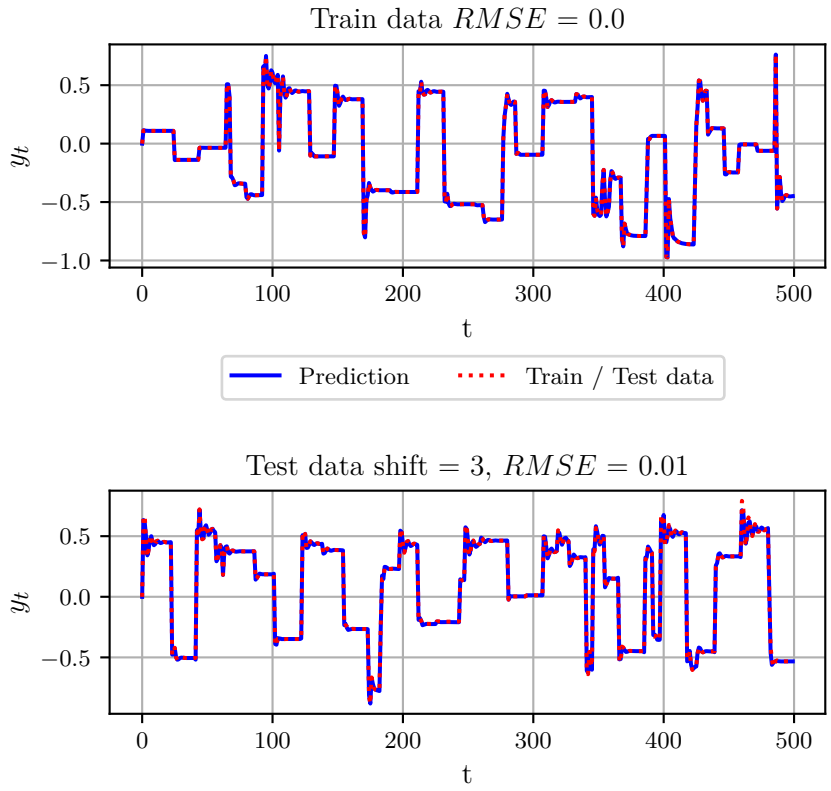


Figure 2.71: Narendra and Parthasarathy [1990] time dependent test function with a $\mathcal{DGCN}$ model and a shift of input and output of 3.

gets additional neurons as for one single output. This leads to the number of neurons to extend to $n \times n_c \times n_o$ for the DNN_l and n_o for the $DNN_{\sigma_n^2}$, with n_o standing for the number of output parameters. The output of the DNN_l would be $N_b \times n \times n_c \times n_o$ and for the $DNN_{\sigma_n^2}$ $N_b \times n_o$. This also means that the Gaussian process part with several covariance function / matrices is calculated separately for each output parameter, but all share the same neural network to estimate the free parameters. So that a separate Gaussian process is built for each output parameter and the correlation between the covariance matrices is learned by the DNN_l and $DNN_{\sigma_n^2}$, in contrast to using additional cross-correlation covariance matrices such as the Co-Kriging approach mentioned earlier in subsection 2.8.9. The two DNNs can be seen as a correlation function between the covariance matrices, describing the relationship between them. The model is trained by maximizing the summed log-likelihood over all n_o output parameters. For more clarity, before writing the new equations for eq. 2.131 to 2.133, $\sum_{i=1}^{n_c} K_i \left(\hat{Y}_{DNN_l}\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}} I$ should be replaced with K_s, $\sum_{i=1}^{n_c} \boldsymbol{k}_{i*} \left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(X)\right)$ with $\boldsymbol{k}_{s*}(\boldsymbol{x}_*, X)$, $\sum_{i=1}^{n_c} K_i \left(\hat{Y}_{DNN_l}(X), \hat{Y}_{DNN_l}(X)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(X) I$ with $K_s(X, X)$ and $\sum_{i=1}^{n_c} k_i \left(\hat{Y}_{DNN_l}(\boldsymbol{x}_*), \hat{Y}_{DNN_l}(\boldsymbol{x}_*)\right) + \hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}(\boldsymbol{x}_*)$ with $k_s(\boldsymbol{x}_*, \boldsymbol{x}_*)$. Then eq. 2.131 to 2.133 change for a multi-output model to

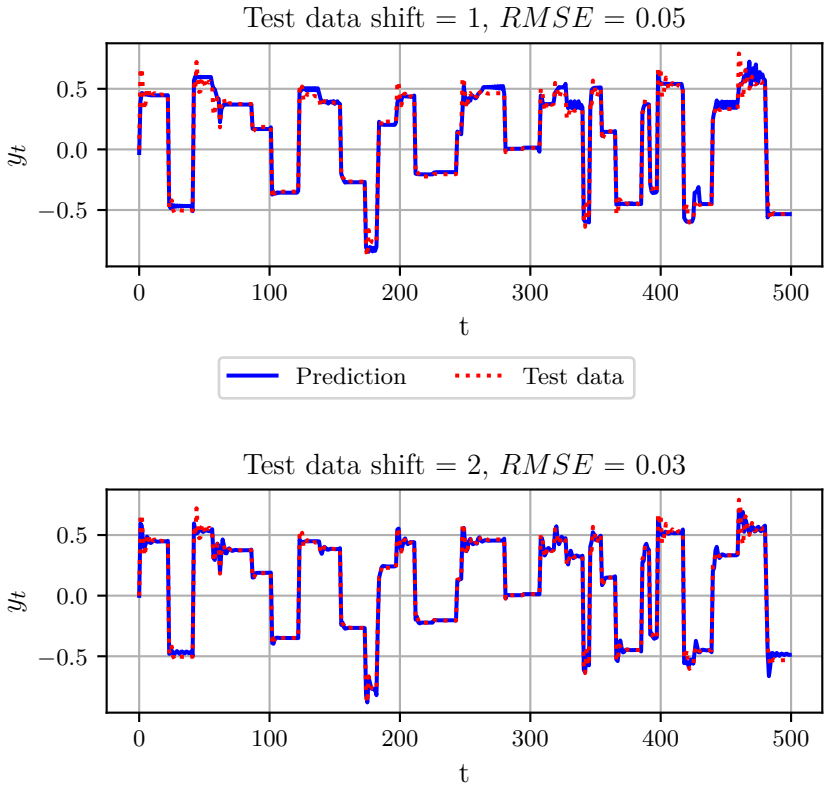


Figure 2.72: For comparison the Narendra and Parthasarathy [1990] time dependent test function with a $\mathcal{DGCN}$ model and a shift of input and output of 2 and 1.

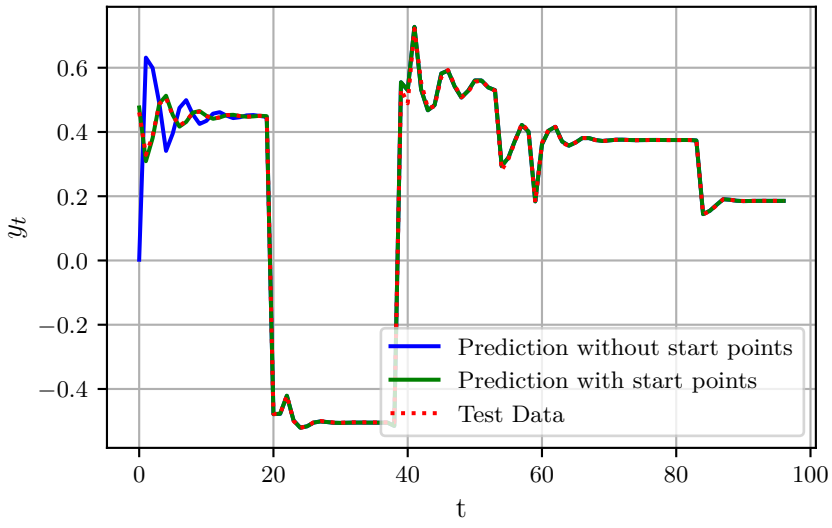


Figure 2.73: For comparison the Narendra and Parthasarathy [1990] time dependent test function (only first 100 points) with a $\mathcal{DGCN}$ model with and without start points for the prediction.

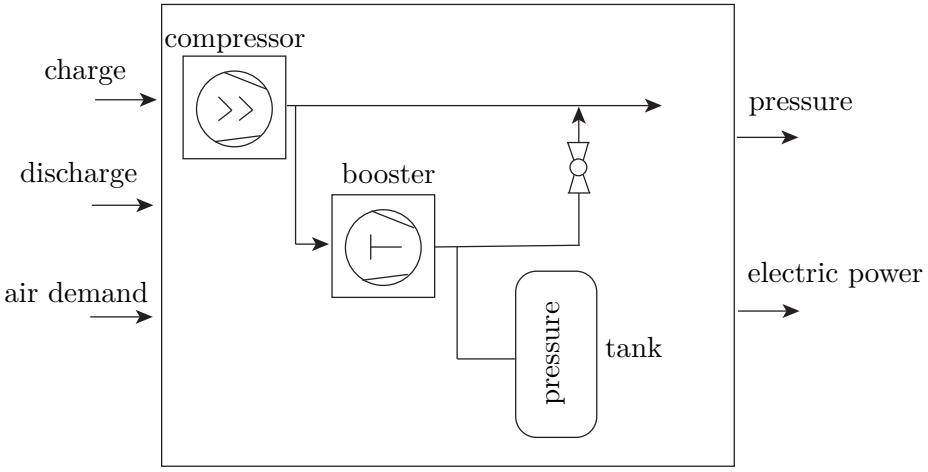


Figure 2.74: Air compressor storage system.

$$\ln p\left(\boldsymbol{y}|X,\hat{Y}_{DNN_l},\hat{\boldsymbol{y}}_{DNN_{\sigma_n^2}}\right)=\sum_{j=1}^{n_o}\left[-0.5\boldsymbol{y}_j^T\left(K_{sj}\right)^{-1}\boldsymbol{y}_j-0.5\ln|K_{sj}|-\frac{N}{2}\ln(2\pi)\right] \tag{2.135}$$

$$\hat{y}_{j*}=\boldsymbol{k}_{sj*}(\boldsymbol{x}_*,X)\left(K_{sj}(X,X)\right)^{-1}\boldsymbol{y}_j \tag{2.136}$$

$$\mathbb{V}\left[\hat{y}_{j*}\right]=k_{sj}(\boldsymbol{x}_*,\boldsymbol{x}_*)-\left(\boldsymbol{k}_{sj*}(\boldsymbol{x}_*,X)\right)^T\left(K_{sj}(X,X)\right)^{-1}\boldsymbol{k}_{sj*}(\boldsymbol{x}_*,X) \tag{2.137}$$

with subscript j for $j=1,...,n_o$ indicating the j-th output.

Apart from the neural network learning the correlations between the output parameters, a further pre-processing step can be included which allows for the correlations between the output parameters to be learned and the number of n_o even to be reduced based on their importance. The principle component analysis (PCA), also known as Karhunen Loève expansion (Wold et al. [1987]), is typically used for data compression and learns the correlations between the input parameters by performing an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of linearly uncorrelated variables called principal components. After this pre-processing step, the $\mathcal{DGCN}$ model learns the remaining non-linear correlations between the principal components. The important feature of this data compression method is that it can transform the principle components back to its original space. So if the $\mathcal{DGCN}$ model gets the principal components as output parameters, it would also only predict the principal components. With the help of the mentioned back transformation property, it is possible to transform the $\mathcal{DGCN}$ prediction back to the original space. The number of principal components to be used can be estimated by the variance they explain. In most cases it is unavoidable that some information is lost through the transformation. This may lead to a reduced prognosis quality of the model. There are other methods which are able to perform a non-linear transformation of the data, for instance kernel PCA (Mika et al. [1999]), where kernel functions are used to perform a projection in a higher dimensional space and from this higher dimensional space a lower projection is calculated. This method has the advantage of reducing non-linear correlated data and therefore allows data compression where the standard PCA fails. The disadvantage of the kernel PCA is that an approximation model also needs to be

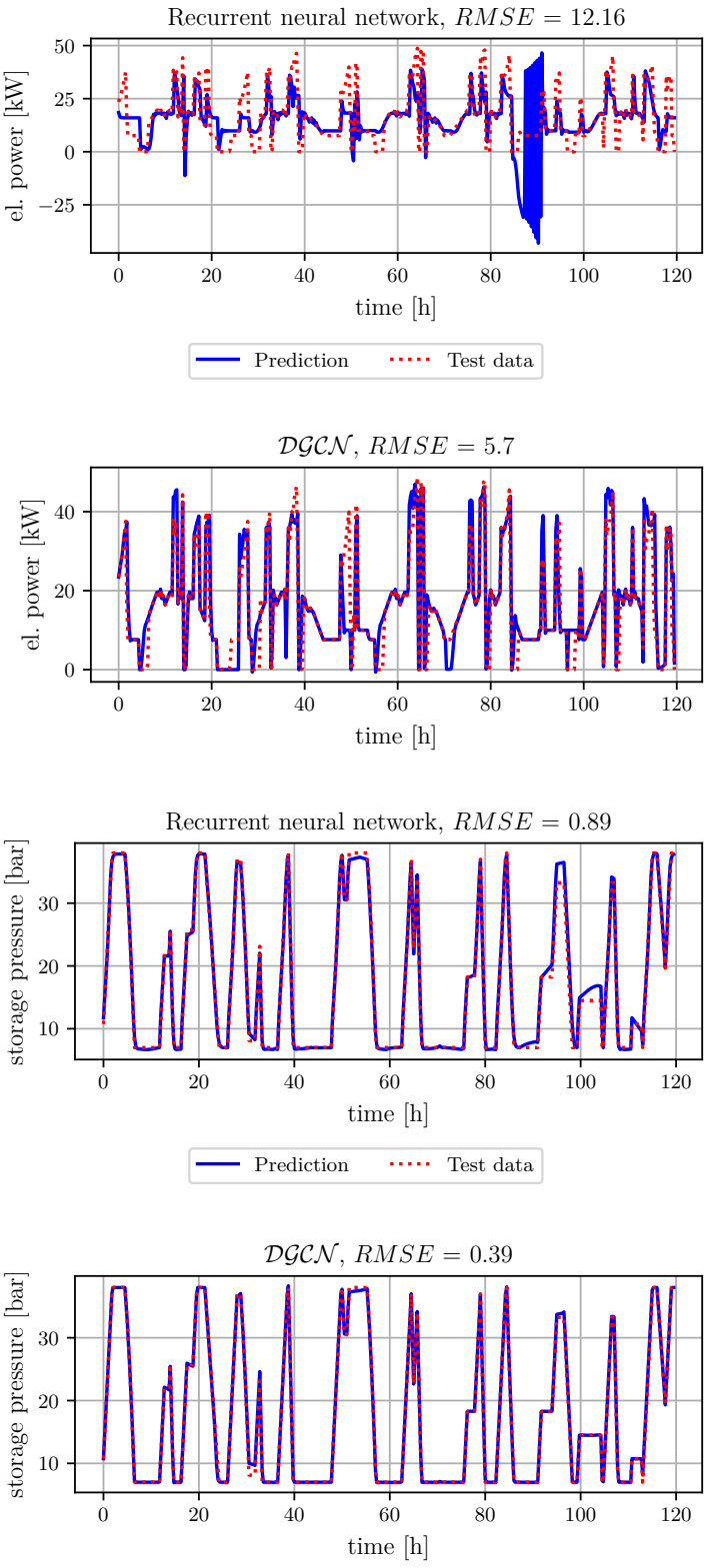


Figure 2.75: Comparison of a recurrent neural network and $\mathcal{DGCN}$ for the electrical power consumption (top) and the storage pressure (bottom) of an air storage system.

fitted as the kernel function, for example radial basis functions or polynomials. These may introduce information loss due to the approximation error. Another method for non-linear data reduction are so-called autoencoders (Masci et al. [2011]). These are feed forward neural networks, with the input and output layer having the same number of neurons with several hidden layers. In these hidden layers which should typically have a lower number of neurons compared to the input and output layer, the transformed data can be extracted as a compressed version of its input. To allow the back transformation, the model is trained so that the input should be equal to the output. The drawbacks of this method are, besides the possible approximation error, the complex training and the search for an appropriate net topology. Despite that, it can also be trained via mini-batch which allows this method to handle arbitrary data set sizes. An example of an autoencoder is depicted in fig. 2.76.

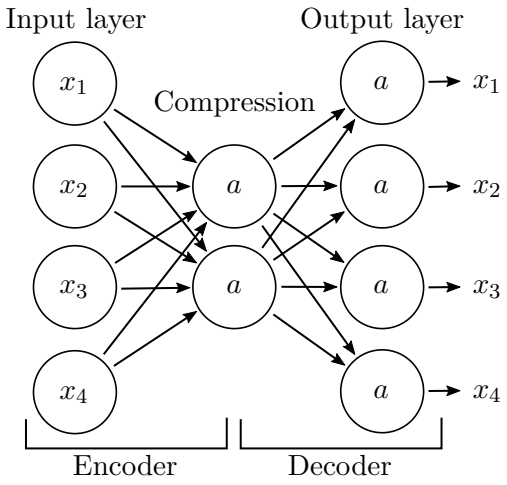


Figure 2.76: Illustration of an autoencoder with one hidden layer for the encoding from 4 inputs to 2 and back to 4 for the decoding.

It turns out (based on own experience) that in many mechanical engineering applications, the standard principle component analysis is sufficient. It has the advantage of being very fast, even for big data applications. In the implementation of the $\mathcal{DGCN}$ method, the user can define whether or not the PCA transformation should be performed and how many components should maximally be used. Internally, the algorithm increases the number of components in a stepwise fashion and calculates the explained variance. If the explained variance does not increase and the principle components used are lower than the defined maximum components, the algorithm uses this number of components. A further effect of using a data compression method is a smoothing of the predicted values. To demonstrate this effect and to show the performance of this approach compared to other algorithms which are also capable of handling multiple correlated outputs, the following test function should be used:

$$\begin{aligned} y_t =& a_i \exp\left(-b_i(t-c_i)^2\right), \\ & a \in \mathbb{R}^{[1,5]},\ b \in \mathbb{R}^{[1,5]},\ c \in \mathbb{R}^{[1,5]},\ t \in \mathbb{R}^{[0,2\pi]},\ i = 1, ..., N \end{aligned} \tag{2.138}$$

a_i, b_i, c_i are parameters modifying the signal and t is the time step discretized in 70 equal steps. Overall 100 signals are used for training and 100 signals for testing the trained models. Fig. 2.77 illustrate some of the training signals. Fig. 2.78 shows the same signals transformed into 35 principle components (reduced from 70 original time steps) which

already explain 100% of the variance. This example will be used in the following to also demonstrate the difference between using the PCA transformed and the original signal.

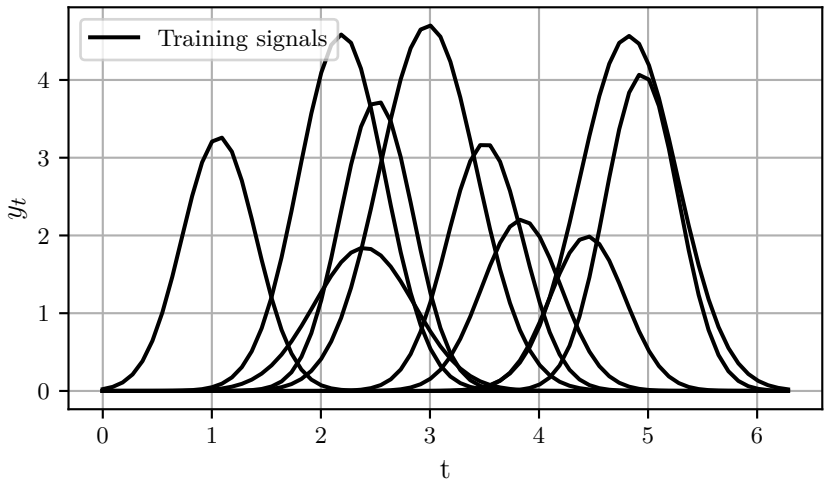


Figure 2.77: Examples of training signals of test function 2.138.

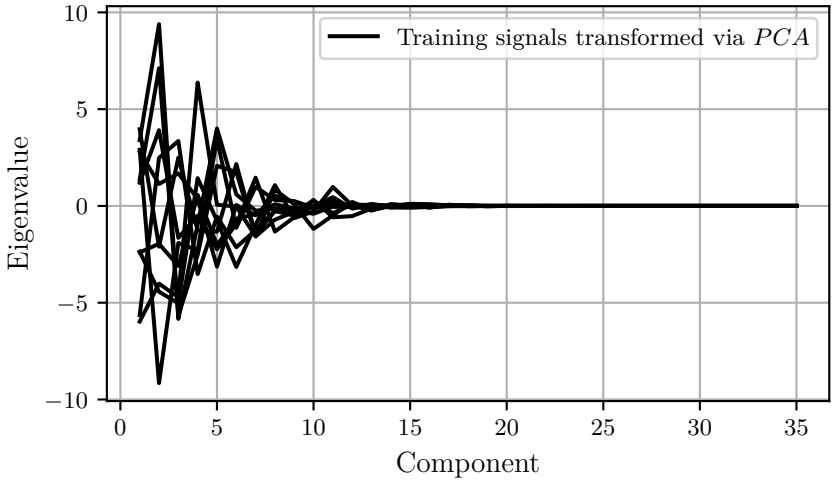


Figure 2.78: Examples of training signals transformed via PCA of test function 2.138.

For comparison with the $\mathcal{DGCN}$ model, a deep neural network with 3 hidden layers and 20 neurons each with $ReLU$ activation functions, decision tree (Quinlan [1986]) and random forest model (Svetnik et al. [2003]) are used. How neural network can handle multi-output was already explained in subsection 2.9.1. The implementations of the algorithms were used with default settings (except the neural network, where the net topology was changed according to the $\mathcal{DGCN}$ model), from the Python scientific library sci-kit learn v. 0.19.1 (Pedregosa et al. [2011]). Decision tree is an algorithm, based on a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. This algorithm defines, for example a decision, if the value for a is greater than a specific threshold and then performs a local regression. For values lower then this

threshold, a local regression with other trained parameters is performed. Typically, low order polynomials are used for the local regression. This algorithm handles multi-output by making decisions over all output parameters connected through the same decision tree. Random forest is an extension of this algorithm which uses an ensemble of multiple decision trees trained on subsets of the data. The prediction of this ensemble is made by averaging all the single predictions in the ensemble. This should avoid over-fitting and improve the predictive accuracy compared to a single decision trees model.

For the first comparison, all models should be trained without a PCA as a pre-processing step. To measure the error of the predictions of the different models, the root mean squared error ($RMSE$) is estimated for each test signal individually. Then the mean of the 100 $RMSE$ values for the 100 test signals of each model is calculated so that one final mean $RMSE$ ($MRMSE$) value is estimated for each algorithm. Fig. 2.79 shows the results for the first five test signals for each algorithm. Fig. 2.80 shows the same results with the PCA. Tab. 2.8 gives the corresponding $MRMSE$ values for all 100 test signals for both cases. As can be seen, the neural network performs the worst and $\mathcal{DGCN}$ by far the best. Using PCA has a beneficial effect for the random forest, the decision tree and the neural network. Interpreting the slight increase of the $MRMSE$ for $\mathcal{DGCN}$ when PCA is used appears to be inconclusive, because the $MRMSE$ for that method is close to zero anyway. This experiment demonstrates that the usage of a pre-processing step allows data to be compressed, while similar results can still be obtained. Nevertheless, it is important to mention that the current implementation of $\mathcal{DGCN}$ only allows a restricted amount of output parameters due to the limited RAM available of the computer system. For a system with 32GB of RAM about $1,000$ output parameters can be handled. Therefore, it is often necessary to work with a data compression pre-processing step. This first example shows also that even though a data compression step is involved, the $\mathcal{DGCN}$ model performs better than other algorithms without it.

Algorithm	$MRMSE$ without PCA	$MRMSE$ with PCA
$\mathcal{DGCN}$	**0.03**	0.04
Random forest	0.26	**0.25**
Decision tree	0.33	**0.30**
Neural network	0.64	**0.57**

Table 2.8: Overview of the mean $RMSE$ ($MRMSE$) values for 100 test signals of test function 2.138 for different multi-output algorithms. The bold values indicate the best result between using PCA or not.

Not mentioned yet, there are other generic possibilities to work with multi-output problems. The first approach would be to create a single model for each output parameter which has the drawbacks that it can be very time intensive dependent on the model algorithm and it does not capture the possible correlations between the output parameters. The second approach would be to create a vector of the aligned output data by introducing a new input parameter to distinguish the different output parameters. This also means that the input parameters need to be constant in the training samples over the length of n_o. Therefore, the number of training samples also increases to $N \times n_o$. To illustrate this, tab.

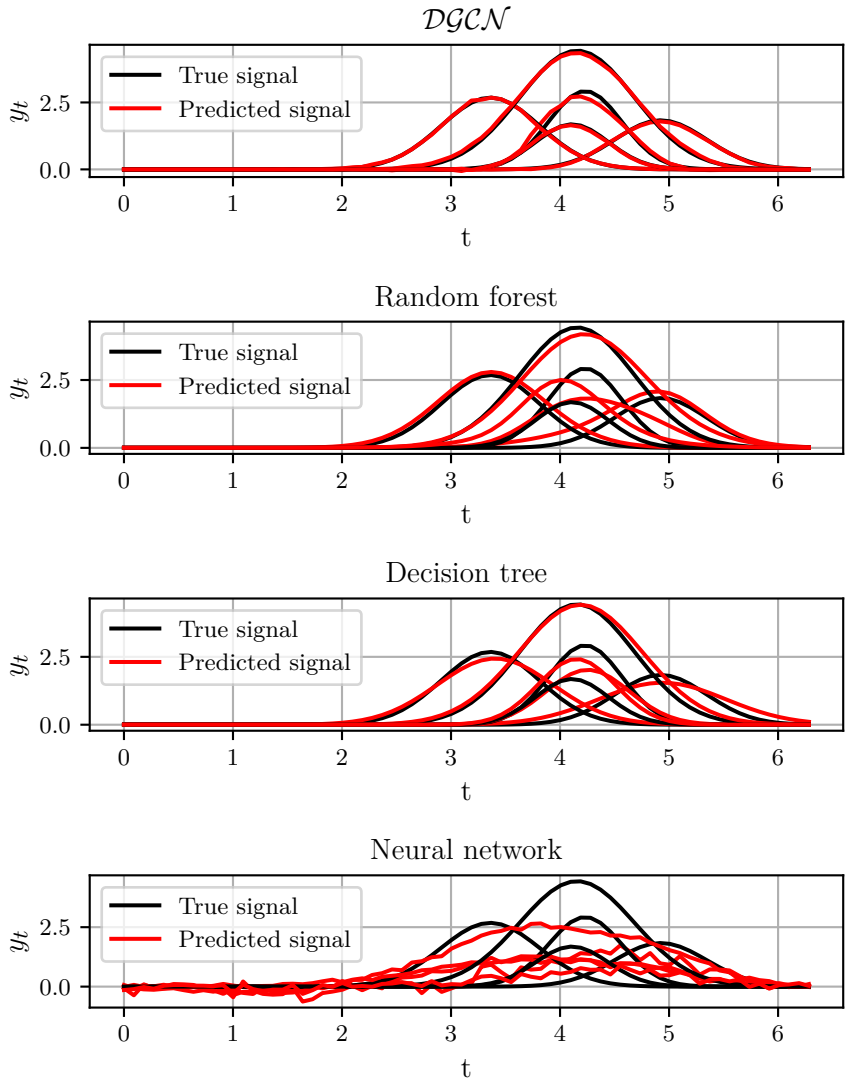


Figure 2.79: Comparison of different algorithms without PCA on test function 2.138, first 5 test signals are shown.

2.9 to tab. 2.11 show these transformations. The vector approach has the advantage that it also considers the possible correlations between the output parameters, but it increases artificially the number of samples which might be a problem for some algorithms. Also, the partially constant input parameters might lead to numerical problems (tab. 2.11).
Furthermore, the test function given in eq. 2.138 will be evaluated under the same conditions, except that the two new approaches to handle multi-output data will be tested, so to build one model per output and to align the multiple outputs to a single vector. Fig. 2.81 (without PCA) and fig. 2.82 (with PCA) show the results if one model is trained per output parameter, so for this case without PCA 70 models were trained and for the case with PCA 35 models were trained. For the case without PCA, the signals look much more notchy, since no correlation between the output parameters is learned. For the case with PCA the signals look much better, similar to the previous multi-output results. Tab. 2.12

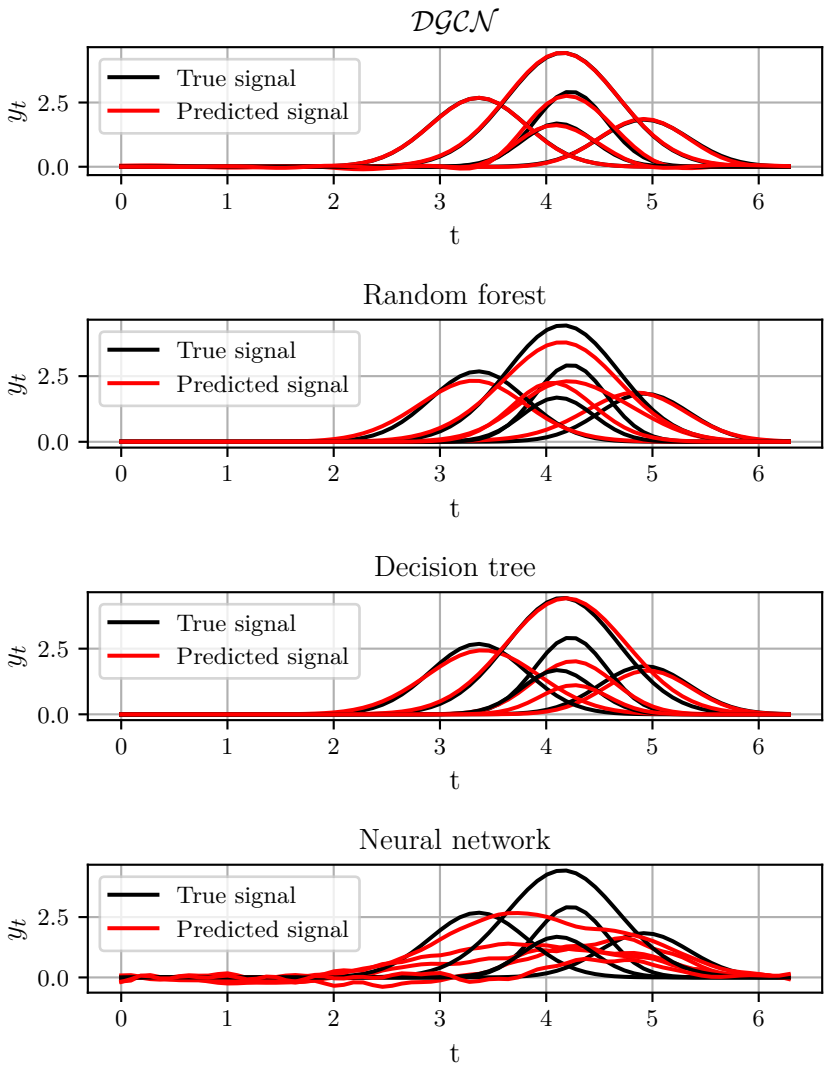


Figure 2.80: Comparison of different algorithms with PCA on test function 2.138, first 5 test signals are shown.

x_1	x_2	x_3	$y_{t=0}$	$y_{t=1}$	$y_{t=2}$	$y_{t=3}$	$y_{t=4}$	$y_{t=5}$	
0.4	0.1	0.5	2.2	2.7	3.3	5.5	6.5	4.3	...
0.1	0.3	0.6	4.4	3.3	2.2	3.5	5.4	2.2	...
0.5	0.4	0.1	2.9	3.5	3.7	4.4	5.5	4.2	...
...	...	...	...	...	...	...	...	...	...

Table 2.9: Example for a multi-output problem with three input parameters and $n_o = 5$.

x_1	x_2	x_3	$y_{t=0}$	$y_{t=1}$	$y_{t=2}$	$y_{t=3}$	$y_{t=4}$	$y_{t=5}$	
0.4	0.1	0.5	2.2	2.7	3.3	5.5	6.5	4.3	...
0.1	0.3	0.6	4.4	3.3	2.2	3.5	5.4	2.2	...
0.5	0.4	0.1	2.9	3.5	3.7	4.4	5.5	4.2	...
...	...	...	...	...	...	...	...	...	...
			m_1	m_2	m_3	m_4	m_5	m_6	...

Table 2.10: Example for a multi-output problem, where one model is built for each output parameter.

x_t	x_1	x_2	x_3	y
1	0.4	0.1	0.5	2.2
2	0.4	0.1	0.5	2.7
3	0.4	0.1	0.5	3.3
4	0.4	0.1	0.5	5.5
5	0.4	0.1	0.5	6.5
6	0.4	0.1	0.5	4.3
1	0.1	0.3	0.6	4.4
2	...	...	...	...

Table 2.11: Example for a multi-output problem, where the data is transformed to a vector. An additional parameter x_t is introduced to distinguish the signals regarding the the input.

gives the $MRMSE$ results for both cases. Compared to the multi-output approach with one single model the results get slightly worse except for the neural network. It can also be seen that the results with PCA smooth out the notchy signals. Surprisingly, the results for the decision tree method are equal for both cases, even though visually the signals look better with the PCA pre-processing. For the other algorithms, the PCA improves the results.

In fig. 2.83 and tab. 2.13 the results are shown for the vector approach. Here only one case is analyzed, since for this approach no PCA pre-processing is needed. The transformed data set has $7,000$ samples. For the $\mathcal{DGCN}$ model this approach also works very well, but because of the increased sample size, mini-batch training was used. Most interesting is the strong improvement of the neural network results. This might be due to the newly created input and artificial increase of the sample size. It was already mentioned that neural networks typically only work well if a larger number of samples is given. Now through this transformation the problem is like a single output problem with many samples which might be beneficial for the neural network. The slightly poorer results of the random forest and

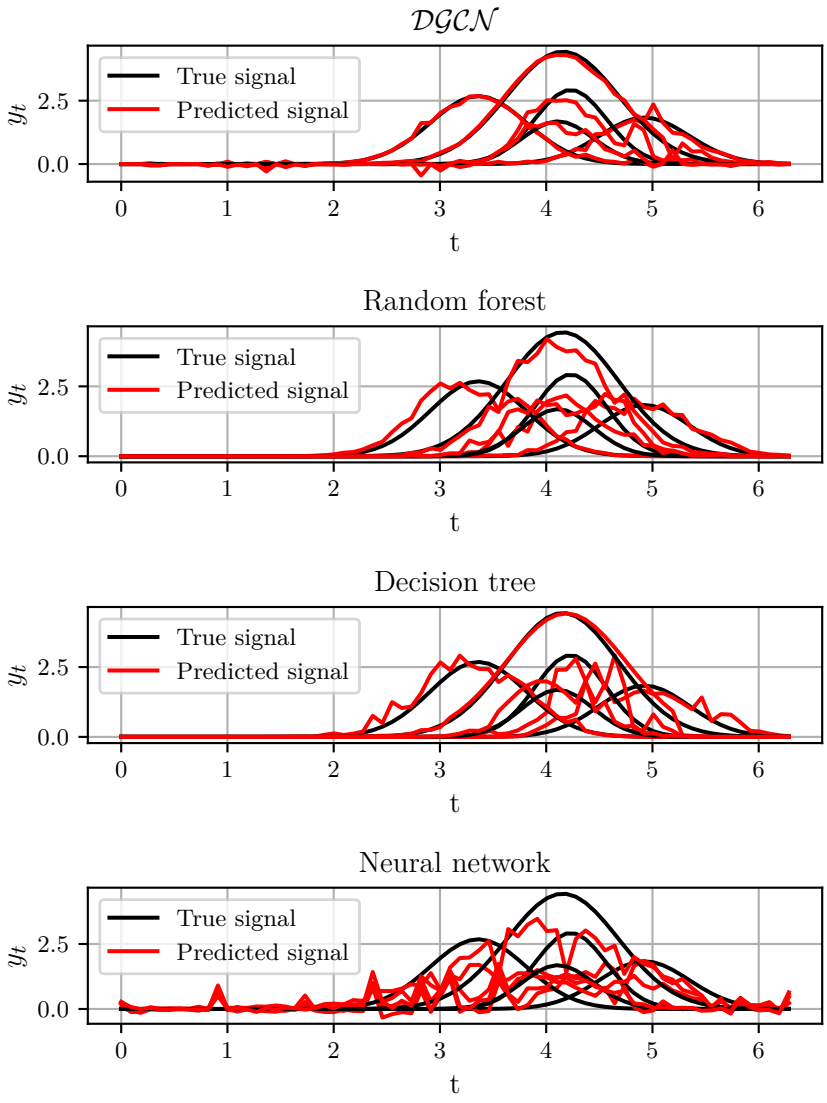


Figure 2.81: Comparison of different algorithms without PCA on test function 2.138, one model per output parameter was trained, first 5 test signals are shown.

Algorithm	$MRMSE$ without PCA	$MRMSE$ with PCA
$\mathcal{DGCN}$	0.18	**0.16**
Random forest	0.27	**0.23**
Decision tree	0.33	0.33
Neural network	0.58	**0.51**

Table 2.12: Overview mean $RMSE$ ($MRMSE$) values for 100 test signals of test function 2.138 for different multi-output algorithms. One model per output parameter was trained.

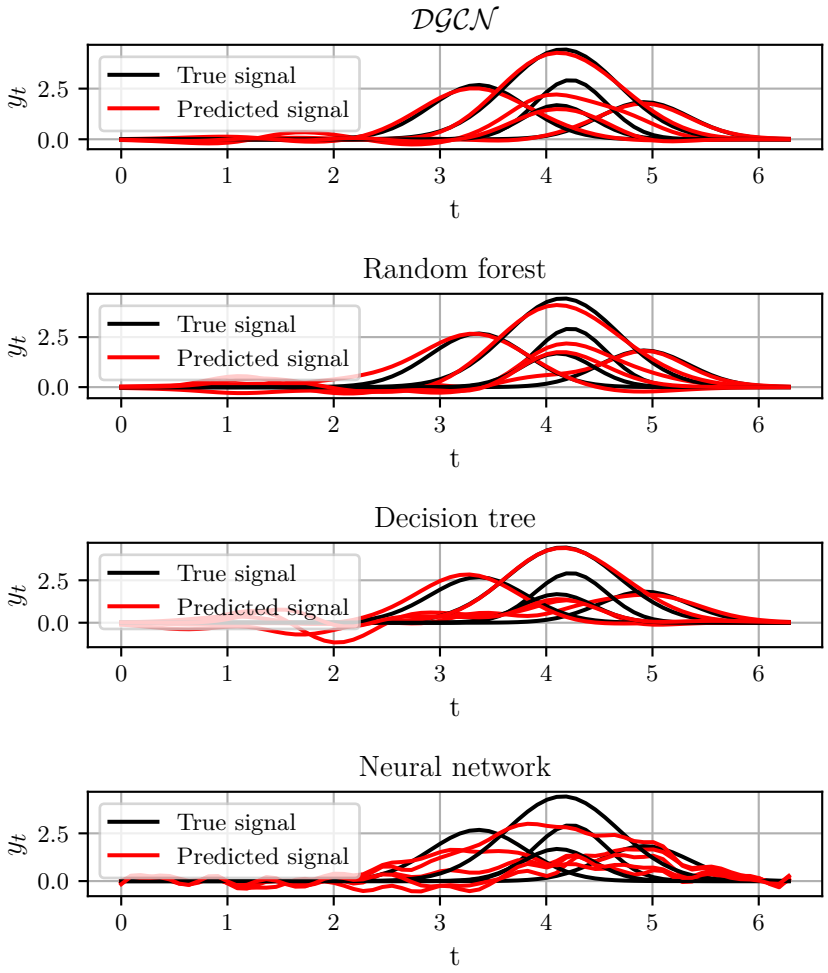


Figure 2.82: Comparison of different algorithms with PCA on test function 2.138, one model per output parameter was trained, first 5 test signals are shown.

decision tree algorithm might be due to the constant input parameters over many samples which makes it difficult to create the correct decision splits. As expected, this approach is also usable to solve multi-output problems dependent on the machine learning algorithm. Overall for this example, $\mathcal{DGCN}$ showed the best results in all variants. The most efficient way is the usage of one single model in combination with PCA. Nevertheless, the overall best result was achieved without the use of a PCA but this might also be dependent on the specific test problem.

To show the capabilities of the multi-output approach for another example, the node stress results of a finite element analysis (FEM) (Szabó and Babuška [1991]) should be learned by one single model. For this example, the model should learn the correlations over space instead over time. The model used is depicted in fig. 2.84. This model represents the geometry of a steam turbine rotor. The model is used to assess the notch factors in the blade grooves and to find an optimal geometry of the stress relief grooves. For this test, the model is discretized by $60,000$ nodes. A variable pressure parameter is set on the left side, whose amplitude also has a constant variation over the height. For this test 40

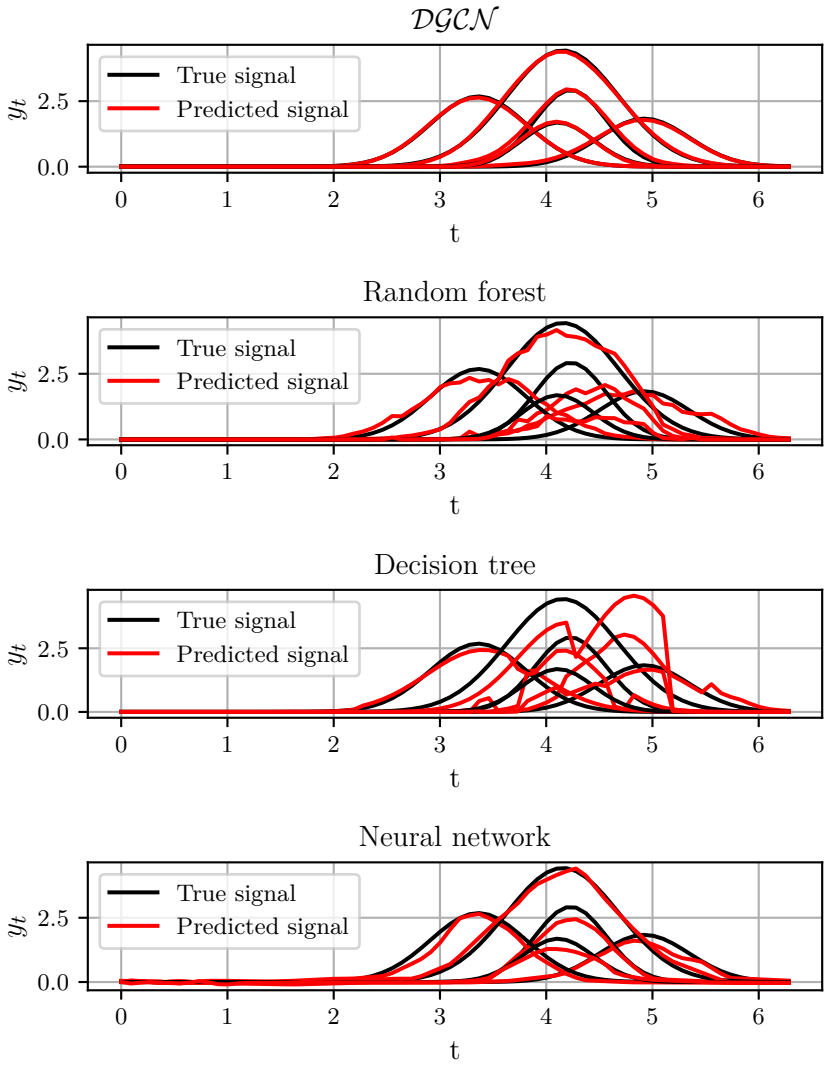


Figure 2.83: Comparison of different algorithms without PCA on test function 2.138, where the output parameters are aligned as a vector, first 5 test signals are shown.

Algorithm	$MRMSE$ without PCA
$\mathcal{DGCN}$	**0.06**
Random forest	0.27
Decision tree	0.37
Neural network	0.15

Table 2.13: Overview mean $RMSE$ ($MRMSE$) values for 100 test signals of test function 2.138 for different multi-output algorithms. Data was transformed to a vector.

designs were calculated with the pressure varying between -12 and -200 Pa to train the models and 10 further designs to test the model. This time the mean absolute percentage error ($MAPE$) over all $60,000$ nodes is estimated for each of the 10 test designs. Once again, the same algorithms with the same settings as in the last test were used. Only the variant using a PCA as a pre-processing step is used, since $\mathcal{DGCN}$ is not able to handle $60,000$ output parameters. With the PCA the output parameters are reduced to one single principle component and nearly 0.99999 of the explained variance remain. Since, the results of the FEM analysis are only linearly dependent on the pressure, it is logical that the $60,000$ nodes can be reduced to one single component. In subsection 2.10.8 more complex examples will be given. Nevertheless, this example demonstrates the power of using data compression methods like the PCA. Also, the displacement of the rotor would easily be possible to estimate if the node coordinates were used as additional outputs. Fig. 2.85 shows a comparison for two of the test designs with a low and high pressure value compared to the true values. Furthermore, the results for the 10 test designs are summarized in tab. 2.14. The $MAPE$ is much lower for the $\mathcal{DGCN}$ model compared to the other algorithms even though the data compression via PCA is used. This example gives a first impression of the possible application areas of the multi-output $\mathcal{DGCN}$ model.

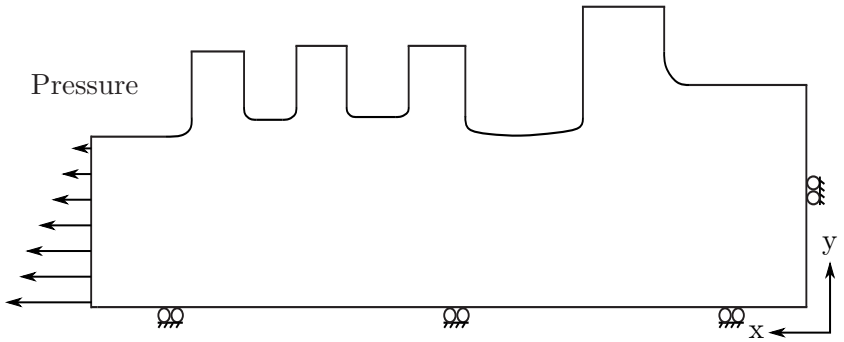


Figure 2.84: Rotor model for the finite element analysis multi-output example. Schematic boundary conditions are visualized.

2.10.6 *Classification*

The next extension of the $\mathcal{DGCN}$ should cover classification tasks. As already shortly referenced in subsection 2.8.4, it is also possible to perform binary classification with Gaussian processes. Since many problems have more than two classes, there are two common methods to overcome this drawback. First, if a multi-class problem with n_{cl} classes should be solved, one binary model per class can be trained and the model with the highest class probability is used for predictions. This strategy is also known as one-vs.-rest strategy (Perronnin et al. [2012]) and is used, for example, if support vector machines should predict multiple classes (Fung and Mangasarian [2005]). In addition to its computational efficiency (only n_{cl} classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one classifier only and it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multi-class classification. The other possibility is called one-vs.-one strategy (Galar et al. [2011]). For n_{cl} different classes, $n_{cl}(n_{cl}-1)/2$ binary classifiers are trained. So, one classifier per class pair. At prediction time, the class which received the most votes, is

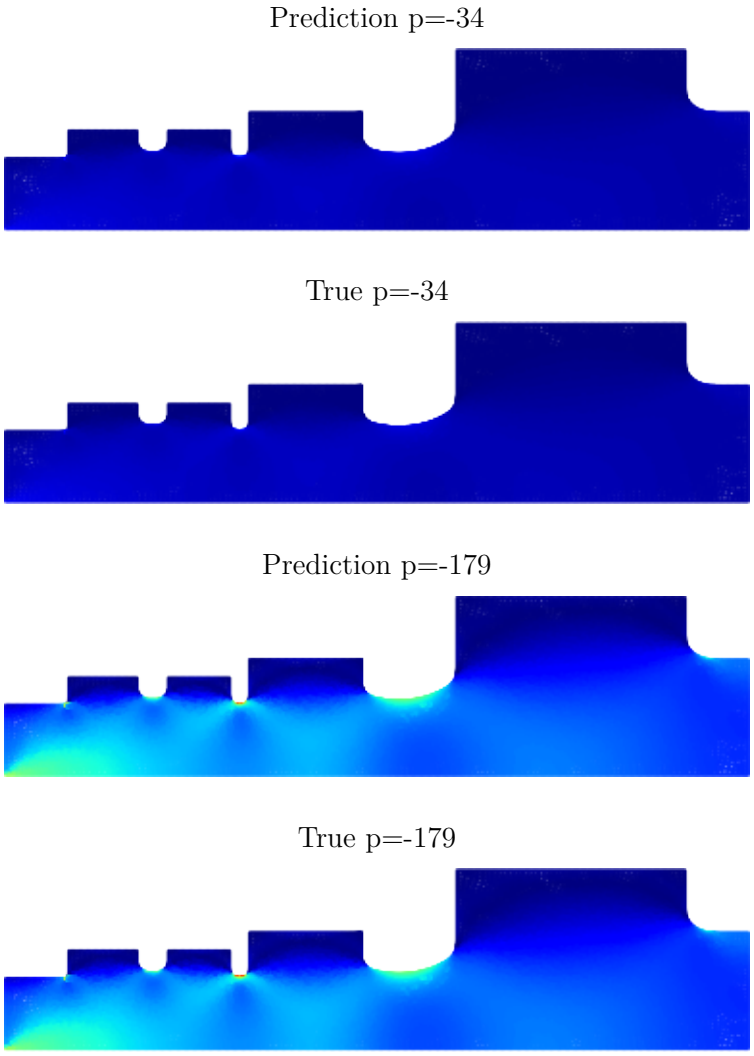


Figure 2.85: Comparison of predicted and true FEM node results for different pressure values.

selected. Since it requires to fit $n_{cl}(n_{cl}-1)/2$ classifiers, this method is usually slower than the one-vs.-rest strategy, due to its $\mathcal{O}(n_{cl}^2)$ complexity.

As the multi-output prediction can be seen like a multi-class classification, the same algorithms are naturally capable to perform this kind of task. Like for multi-output, algorithms which are naturally capable to handle multiple classes perform in most cases better and more efficient than the one-vs.-rest or one-vs.-one approach. Methods like the already mentioned neural networks or decision trees are typically used for such tasks. For the $\mathcal{DGCN}$ method, the approach similar to multi-output is used with the difference that the likelihood for classification (see eq. 2.83) is used and the prediction is a binary sequence with 1 for the corresponding class and 0 for the others. Therefore, the prediction is always a matrix of size $N_* \times n_{cl}$ (N_* is the number of test points). It should be noted that for most algorithms multi-class problems with an output vector of e.g. $\boldsymbol{y} = [1, 2, 3]$ needs to be transformed into a matrix:

Design no.	$\mathcal{DGCN}$	Random forest	Decision tree	Neural network
1	**0.026**	1.591	3.979	10.350
2	**0.013**	2.190	10.951	203.693
3	**0.026**	0.906	2.265	37.157
4	**0.009**	2.090	5.227	44.956
5	**0.028**	3.444	3.826	6.127
6	0.026	**0.007**	3.018	16.294
7	**0.037**	4.626	6.608	83.274
8	0.025	**0.001**	2.121	41.156
9	**0.088**	12.576	8.983	149.13
10	**0.027**	1.424	2.373	34.174

Table 2.14: Overview $MAPE$ values in [%] for 10 test designs of the test FEM analysis for different multi-output algorithms.

$$Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This transformation is called one-hot encoding (Rodríguez et al. [2018]). In the $\mathcal{DGCN}$ implementation, the result of the prediction corresponds to how the training data was passed. Internally always the data is one-hot encoded if necessary. Additionally, the class probabilities can be predicted. Similar to the one-vs.-rest approach, algorithms which are able to handle multi-class problems, make their prediction of the class based on the class probabilities of all possible classes. The sum of this class probabilities must not be equal to one, since each class is handled like a binary problem if the data is one-hot encoded. So the class probabilities for each class represent the probability being this class or one of the other possible classes. In the $\mathcal{DGCN}$ implementation the results are normalized by the sum of all class probabilities. All these approaches suffer from ambiguities because some regions of the input space may receive the same probability for each class.

To show the classification capabilities, an example should be given, where $\mathcal{DGCN}$ should be compared to other classification algorithms for a binary classification problem. The example was already published in Lueckemeyer et al. [2017]. The classification task is to determine whether a turbine-generator train (see fig. 2.86) has critical Eigenfrequencies or not if the stiffness of the bearings varies. Overall 8 input parameters are used which describe the horizontal and vertical stiffnesses of the bearings. The training sample size will be stepwise increased, to analyze the increase of the prediction quality. The steps are 10, 20, 50, 100, 250, 500, 1,000, 2,000 samples. For testing, additional 1,000 samples are used. As error metric, the percentage of correct classified samples will be used. Next to

the $\mathcal{DGCN}$ model, the following algorithms are used: neural network (NN), random forest (RF), support vector machine (SVM) (Hearst et al. [1998]), AdaBoost (AB) (Hastie et al. [2009]), XGboost (XGB) (Chen and Guestrin [2016]) and Gaussian process ($\mathcal{GP}$). XGBoost is an algorithm that has recently been dominating applied machine learning competitions. It is an implementation of gradient boosted decision trees designed for speed and performance. Boosting is a method for creating an ensemble of models. It starts with one model and then based on the error the first model makes, a further model will be trained to reduce this error. This can be repeated for several times. The same holds for the Adaboost (adaptive boosting algorithm) that begins by fitting a decision tree on the original data set and then fits additional copies of the classifier on the same data set but the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. All algorithms are taken from the Python scientific library sci-kit learn v. 0.19.1 (Pedregosa et al. [2011]), except the XGboost algorithm which is used from the official public available GIT repository (Chen and Guestrin [2020]) in version 0.71. A cross validation grid search (Bergstra et al. [2011]) is performed to find the best hyperparameters for all algorithms except XGboost, $\mathcal{GP}$ and $\mathcal{DGCN}$. This means hyperparameter combinations are tested within a 5-fold CV and the best performing parameters are then used for the final classification. This step is performed for every training sample step size. For more details on the used grid search (Bergstra et al. [2011]) see Lueckemeyer et al. [2017]. For the $\mathcal{GP}$, a squared exponential covariance function is used. The results for the different tested algorithms are shown in tab. 2.15. The $\mathcal{DGCN}$ model works quite well compared to the other classification algorithms, especially if $N >= 250$.

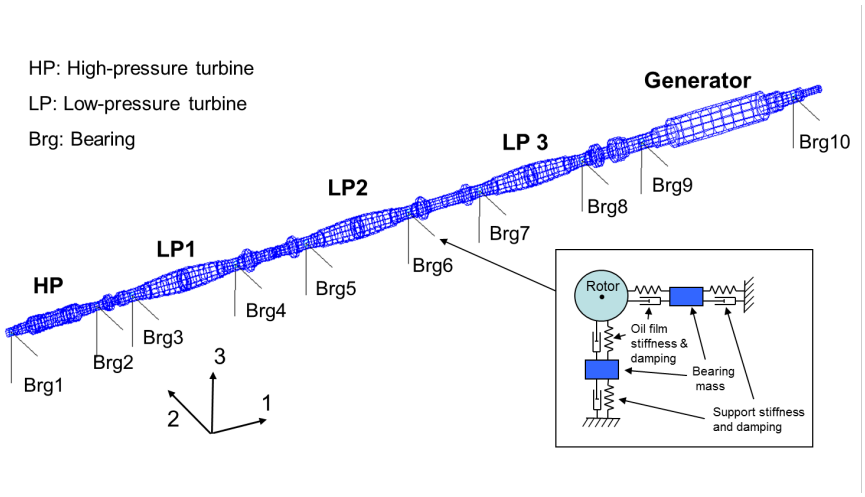


Figure 2.86: Turbine-generator train with simple model of rotor support.

It is also possible to use the class probabilities to plot the critical bearing stiffness values. In fig. 2.87 the two most important bearing stiffness parameters (vertical and horizontal stiffness of the low pressure turbine) are illustrated and the color of the contour plot demonstrate the class probability from 0 (blue) no resonance to 1 (red) resonance. Additionally, the $1,000$ test points are shown. It can be seen that the classification of the test points corresponds very well with the contour plot of the class probabilities. The probabilities are calculated with the $\mathcal{DGCN}$ model trained with $N = 2,000$ samples.

N	$\mathcal{DGCN}$	AB	RF	SVM	NN	$\mathcal{GP}$	XGB
10	89.6	**93**	88.1	87.9	84.1	87.9	87.9
20	88	85.6	**88.4**	87.9	87.9	85.9	87.9
50	90.9	92.5	**93.1**	87.9	89.6	90.5	87.9
100	94.8	91.5	93.9	87.9	90	**95.1**	90.7
250	**96.8**	93	95.8	95.1	96.4	96.5	96
500	**97.9**	94.7	96.6	94.7	95.5	97.3	94.8
1,000	**98**	95	**98**	95.8	96.3	97.9	96.7
2,000	**99.4**	96.5	98.4	97.9	99	98.7	97.8

Table 2.15: Overview of prediction accuracy in [%] for 1,000 test designs for different classification algorithms.

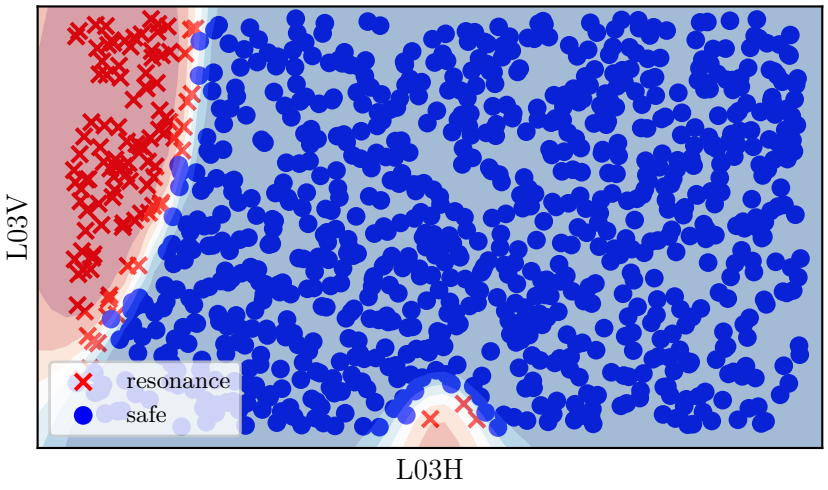


Figure 2.87: Example of plotting the class probabilities for the turbine-generator train example. Points illustrate the 1,000 test points. The contour plot represents the class probability from 0 (blue) no resonance to 1 (red) resonance.

2.10.7 *Image Classification*

The last extension utilizes the $\mathcal{DGCN}$ to perform image classification. For this, a convolutional neural network (CNN) is coupled with the $\mathcal{DGCN}$ model so that it pre-processes the images as explained in subsection 2.9.4 and then the output of the convolutional network is used as input for the $\mathcal{DGCN}$ model (see fig. 2.88). The weights and biases of the convolutional network are trained together with the neural network part of the $\mathcal{DGCN}$ model so that the pre-processing is optimal for the $\mathcal{DGCN}$ classification part. Since it is a classification task, the $\mathcal{DGCN}$ model is trained as explained in the previous subsection 2.10.6. For the convolutional network the following topology is repeated 3 times with different filter sizes:

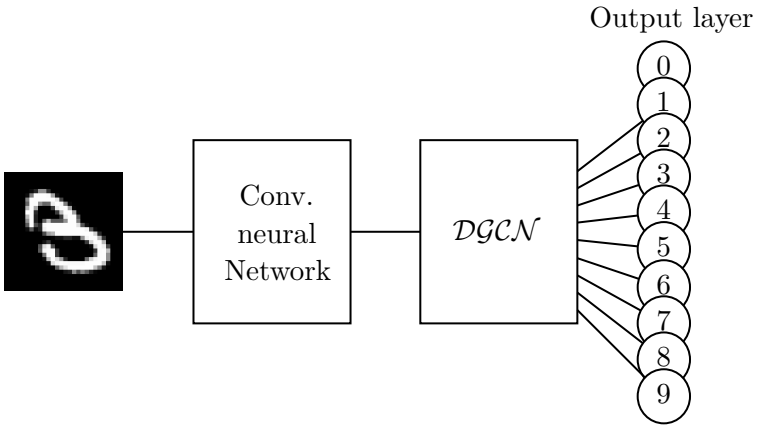


Figure 2.88: Generic illustration of how $\mathcal{DGCN}$ is coupled with a convolutional neural network.

- Convolutional layer with 64 / 256 / 128 filters, a kernel size of 3×3 and *ReLU* activation function type.
- Batch normalization layer.
- Convolutional layer with 128 / 256 / 64 filters, a kernel size of 3×3 and *ReLU* activation function type
- Batch normalization layer.
- MaxPooling layer with pool size 2×2.
- Dropout layer with 20% drop out.

It turned out that this topology worked well for most test cases analyzed so far. The topology of the convolutional network could also be a further possible hyperparameter. To show the capabilities of this extension, two famous test cases will be analyzed. The first one is the MNIST image data set of gray scaled handwritten digits $0-9$ (LeCun [1998]). For this example, there are $60,000$ training images and $10,000$ test images. The picture size is 28×28 pixels. Fig. 2.89 shows some examples. The second test case is the CIFAR10 image data set (Krizhevsky et al. [2014]), it consists of $60,000$ 32×32 color images of 10 different classes, with $6,000$ images per class. There are $50,000$ training images and $10,000$

test images. The classes are airplane, car, bird, cat, deer, dog, frog, horse, ship and truck. Fig. 2.90 also shows some examples. Compared to the examples shown in subsection 2.10.6, these two cases are multi-class classification problems. Tab. 2.16 and tab. 2.17 present the prediction accuracy of $\mathcal{DGCN}$ for both test cases in comparison to state of the art results of other published works. It is worth mentioning that no data augmentation (like rotating or flipping the images) was used for the $\mathcal{DGCN}$ model which is a common tool to increase the accuracy and also used by many of these published works. The training was performed only for 10 epochs (MNIST) and 50 epochs (CIFAR10), since so many training images were available. The training was done via mini-batch with $N_b = 100$. Both cases were trained within a few minutes on a single GPU (Nvidia Geforce GTX 1080 Ti). It can be seen that the $\mathcal{DGCN}$ performs very well compared to other state of the art methods. Most other works use much bigger convolutional neural network topologies in combination with data augmentation to achieve these results. Since Gaussian processes are typically not able to handle such big data sizes, only very few publications were available which also uses Gaussian processes for these two test cases. All publications found used inducing points to reduce the amount of data (see subsection 2.8.10). In Krauth et al. [2016] an accuracy of 55.05% was achieved for the CIFAR10 test case. The training took several hours. Ozdemir and Davis [2016] achieved 95% accuracy on MNIST and 45% on CIFAR10. Hensman et al. [2015] got an accuracy of 97.8% on MNIST but they turn the problem into a one class classification problem, by predicting only if the digit is odd or even. Gal et al. [2014] got 94.05% on MNIST. So compared to other Gaussian process approaches which are also able to handle big data set sizes, $\mathcal{DGCN}$ outperforms them. One reason for this might be that no information is lost within the presented approach compared to the inducing point concept. Another reason might be the unique combination of a CNN together with a $\mathcal{GP}$.

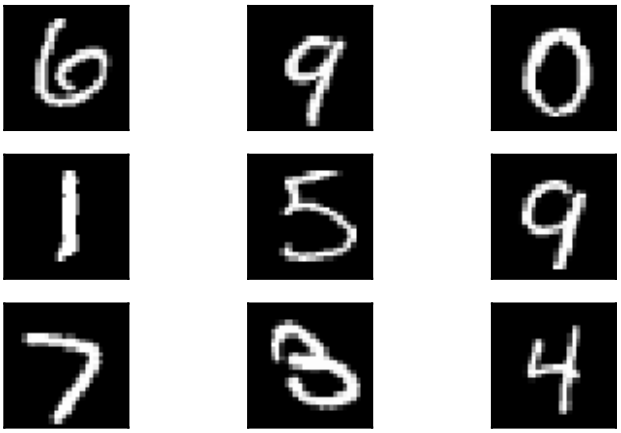


Figure 2.89: Examples of the MNIST handwritten digits data set.

2.10.8 *Further Test Cases*

Besides the experiments already presented, some further experiments are presented in the following. These especially show comparisons to other published non-stationary Gaussian

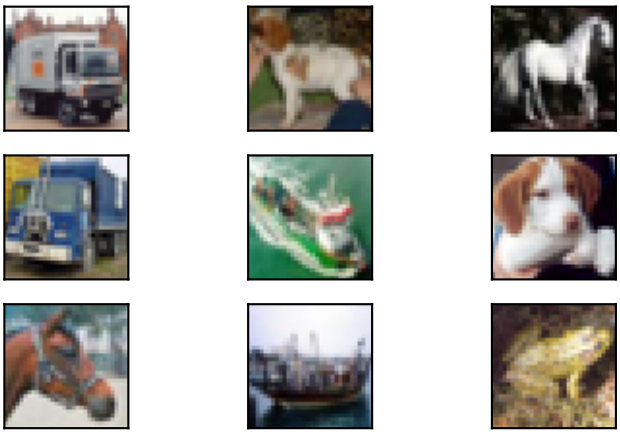


Figure 2.90: Examples of the CIFAR10 data set.

Method	Accuracy
$\mathcal{DGCN}$	99.62
Wan et al. [2013]	99.79
Lee et al. [2016]	99.71
Liang and Hu [2015]	99.69
Poultney et al. [2007]	99.61

Table 2.16: Overview of prediction accuracy in [%] of different published works for 10,000 test images of the MNIST data set.

Method	Accuracy
$\mathcal{DGCN}$	90.56
Graham [2014]	96.53
Clevert et al. [2015]	93.45
Stollenga et al. [2014]	90.78
Sato et al. [2015]	89.67

Table 2.17: Overview of prediction accuracy in [%] of different published works for 10,000 test images of the CIFAR10 data set.

processes. The first experiment uses the presented framework for a time dependent problem, namely the CATS benchmark (Lendasse et al. [2004]) which was a competition for a time series prediction problem. In this benchmark, the objective was to predict 100 missing values of a 5,000 point data set (see fig. 2.91). There is no specification about the meaning of the values. These missing values were divided in 5 blocks where the missing elements are:

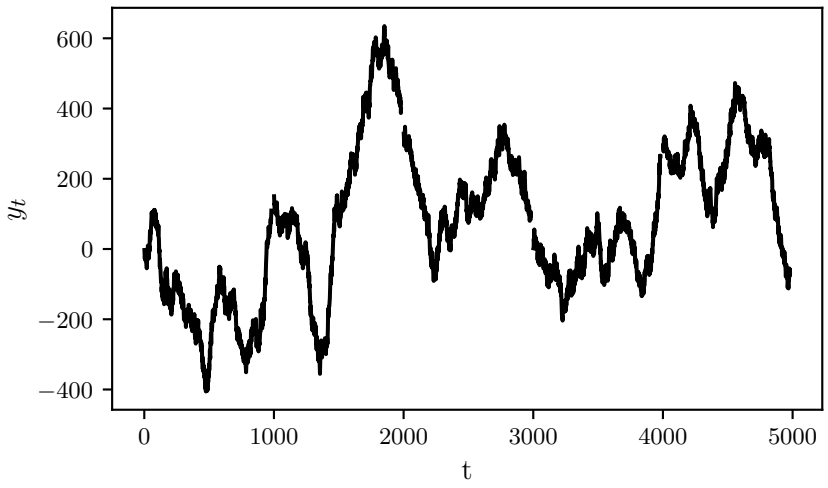


Figure 2.91: CATS benchmark 4,900 data points and 100 missing points.

- element 981 to 1,000
- element 1,981 to 2,000
- element 2,981 to 3,000
- element 3,981 to 4,000
- element 4,981 to 5,000

The competition objective was to minimize the mean square error MSE which is calculated by

$$MSE = \frac{\sum_{t=981}^{1000}(y_t - \hat{y}_t)^2}{100} + \frac{\sum_{t=1981}^{2000}(y_t - \hat{y}_t)^2}{100} + \frac{\sum_{t=2981}^{3000}(y_t - \hat{y}_t)^2}{100} + \frac{\sum_{t=3981}^{4000}(y_t - \hat{y}_t)^2}{100} + \frac{\sum_{t=4981}^{5000}(y_t - \hat{y}_t)^2}{100} \tag{2.139}$$

There was also a second criterion E_2, where only the first 80 of the 100 missing values are considered, for those who worked with the data before and after the missing values. This will not be considered here. Five models were built for this task, one for each block, where each of the models use a different number of previous time-steps as artificial input parameters. The number of previous time-steps was estimated with the help of an auto-correlation analysis shown in fig 2.93. Auto-correlation (Priestley [1981]) is the linear correlation of a signal with a shifted copy of itself. Therefore, it measures the similarity between observations as a

function of the shift between them. Here it is used to estimate the shift value when the data begins to be uncorrelated to its previous steps, to train the model for the corresponding block. Different shift values were tested. The models are trained in such a way that the $t - n_t$ previous time-steps (n_t denotes the number of previously used time-steps) are used to predict the current time-step t. So only the data before the missing values are used for training to predict the last 20 elements of each block. To specify this, for the first block 100, for the second block 200, for the third block 500, for the fourth block 70 and for the fifth block 200 previous steps were used to predict the next one. Then the own prediction of the model is aligned to the previous n_t time steps until all 20 missing steps were predicted. Overall 24 works / methods were used in this competition, with MSE ranging from 408 to $1,714$. In tab. 2.18, the result of $\mathcal{DGCN}$ with $MSE = 247.89$ is shown compared to only the best 3 of the competition. As demonstrated, the $\mathcal{DGCN}$ model gets a much better result for example compared to a deep recurrent neural network (Juang [2004]). Additionally, the prediction vs. the true function values are plotted in fig. 2.92.

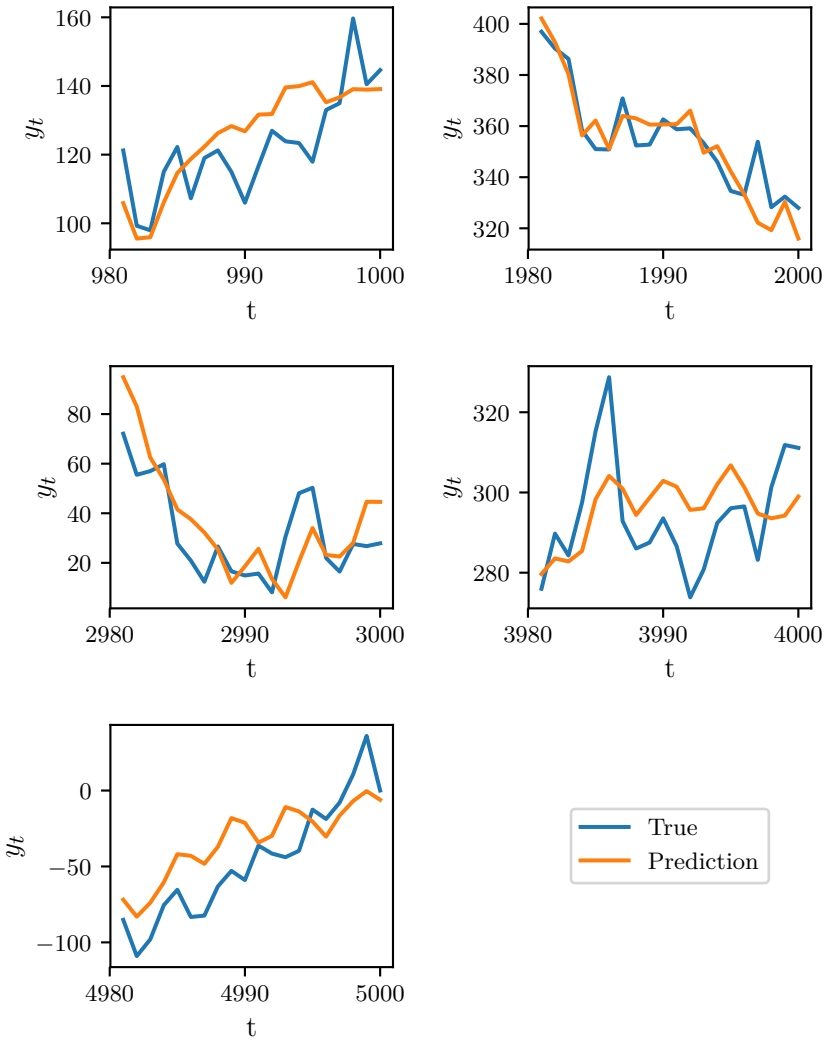


Figure 2.92: CATS benchmark 5 blocks comparison of prediction and true result for the missing 100 points.

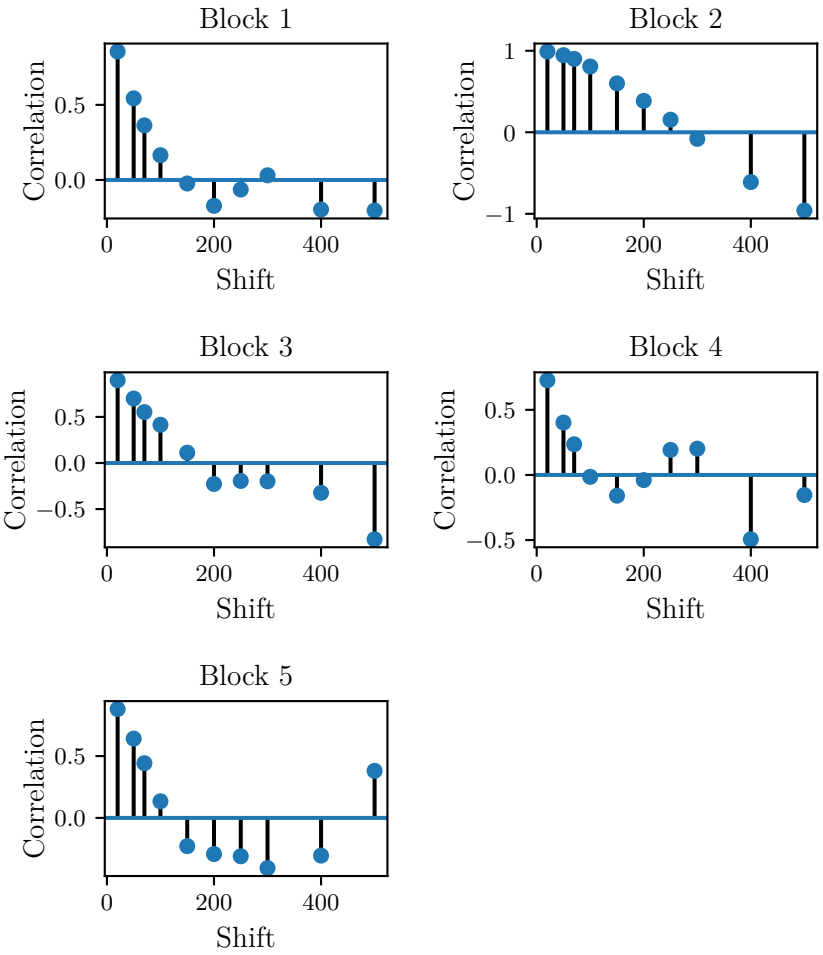


Figure 2.93: Auto-correlation for all blocks with shift= 20, 50, 70, 100, 150, 200, 250, 300, 400, 500.

Author	MSE	Model
-	248	$\mathcal{DGCN}$
Sarkka et al. [2004]	408	Kalman Smoother
Juang [2004]	441	Recurrent Neural Networks
Kurogi et al. [2004]	502	Competitive Associative Net

Table 2.18: Results for the CATS benchmark.

The second experiment shows the capabilities for regression tasks. Therefore, the often-presented Boston housing example (Harrison and Rubinfeld [1978]), which summarizes the median house prices in the Boston metropolitan area, will be used. The data was collected in 1978 and each of the 506 entries represent aggregated data about 13 inputs (including discrete parameters) and 1 output (median house price) for homes from various suburbs in Boston, Massachusetts. The results will be compared to 5 different publications

(Gramacy and Lee [2008], Mair and Brefeld [2015], Vanhatalo et al. [2009], Franz and Gehler [2006], Bui et al. [2015]) that also used this example as a benchmark for their Gaussian-process-based methods. Further results on this benchmark can be found in Chipman et al. [2002]. For the first comparison to Gramacy and Lee [2008], where non-stationary Gaussian processes were also used, the root mean squared error ($RMSE$) via 10-fold cross-validation is estimated. This is repeated 20 times, where the data is shuffled randomly for the cross-validation estimation. The data in this publication was transformed by taking the logarithm of the output to estimate the error. The second comparison is made to Mair and Brefeld [2015]. This work also uses a Gaussian process with a Student-t likelihood and a mixture of local experts (ensemble of multiple models) to provide a more robust approach. In this case, the first 455 samples were used for training and the last 51 samples for testing. The process was repeated 25 times with shuffled input and output. Also, the data was transformed through normalization. Since the results are only depicted visually and the exact values are not shown, no exact comparison is possible. For comparison once again the $RMSE$ will be used. The visual results show a value of 0.44 as the lowest number on the y-axis with the exact value being between 0.44 and 0.46. The third comparison is made to Vanhatalo et al. [2009]. Similar to the second comparison, a Gaussian process is used with a Student-t likelihood, to get a more robust result. Once again, the data is normalized and the $RMSE$ is estimated via 10-fold cross-validation. This time no repetition was made in this work. Nevertheless, as for the first comparison, 20 repetitions are carried out for $\mathcal{DGCN}$ results. Since it is not known if their value was picked as the best result of multiple tries, it is difficult to compare the results but the highest $RMSE$ out of 20 repetitions is still a bit lower than the compared method value. The fourth publication is Franz and Gehler [2006]. They use also Gaussian processes, with a special approach to choose the covariance function. A 10-fold cross-validation to estimate the mean squared error (MSE) is used. To make their results comparable with the others, the square root is taken from the results given in their paper. Also, no repetitions were made by them. The output data was not transformed in this case. The last and fifth comparison is made to Bui et al. [2015]. They use a very new and interesting approach called deep Gaussian processes (Damianou and Lawrence [2013]) using stochastic expectation propagation and probabilistic backpropagation. In this approach, Gaussian processes are used as activation functions of a deep neural network. Similar to the first comparison, 20 repetitions of a 10-fold cross-validation were used to estimate the $RMSE$. Instead of the min / max value, the standard deviation is given.

The result obtained by comparing the methods explained above are summarized in tab. 2.19. There is always the mean and min / max value over all repetitions given. Only for the Bui et al. [2015] comparison, the standard deviation is used instead. Overall, compared to these 5 works, the $\mathcal{DGCN}$ was able to achieve better results in all cases. One thing to point out is the fact that all comparison were done under the same settings for the $\mathcal{DGCN}$ given in section 2.10.2. Most of the methods compared tested different setups in their publications, from which the best results are taken for this comparison. This demonstrates that the presented approach is an easy to use method without the need for finding the right covariance functions of the $\mathcal{GP}$ or the right net topology of the DNN, even if the kind of problem switches from time series forecasting to a regression task.

Since the deep Gaussian process $\mathcal{DGP}$ (Damianou and Lawrence [2013]) is a recent promising topic in the machine learning community, combining also neural networks with

Method	$RMSE$ method	$RMSE$ $\mathcal{DGCN}$
Gramacy and Lee [2008]	0.1321 / 0.1346 / 0.1389	**0.1184 / 0.1257 / 0.1309**
Mair and Brefeld [2015]	>0.44 / >0.44 / <0.48	**0.204 / 0.293 / 0.408**
Vanhatalo et al. [2009]	- / 0.287 / -	**0.259 / 0.2707 / 0.285**
Franz and Gehler [2006]	- / 2.89 / -	**2.34 / 2.34 / 2.43**
Bui et al. [2015]	- / 2.47±0.49 / -	**- / 2.20±0.037 / -**

Table 2.19: Comparison of the Boston housing example against other Gaussian-process-based methods. $RMSE$ min, mean, max values are given. For the last example the standard deviation is used instead.

Gaussian processes, it should be used for further comparisons of other regression tasks. The work that is used for the previous comparison Bui et al. [2015] includes further regression and classification examples taken from the UCI database (Lichman [2013]) (a large machine learning repository), for which the $\mathcal{DGCN}$ method should also be tested. Additionally, a recently published work about dropout and uncertainty in deep learning also used most of the same data sets. Therefore, this is also taken for comparison, named $drop_{DNN}$. In this work a neural network with two hidden layers and 50 neurons each is used. For all examples shown in tab. 2.20, 20 repetitions of 10-fold cross-validation were used to estimate the mean and standard deviation of the $RMSE$. The results were rounded to two decimals in their works. The $\mathcal{DGCN}$ results are better in 5 out of 7 examples compared to the other methods. For the examples kin8nm, power and white wine, mini-batch training with a batch size $N_b = 500, 3,000, 500$ was used. The red and white wine data set is a multi-class classification task with 5 different classes for the red wine and 7 different classes for the white wine data set. The data sets should be described briefly:

- Concrete data set:
  - The task is to approximate the concrete compressive strength which is a highly non-linear function of age and ingredients. First published in Yeh [1998].
  - Inputs: cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate, age.
  - Output: concrete compressive strength
- energy 1, energy 2:
  - This data set looked into assessing the heating load and cooling load requirements of buildings (that is, energy efficiency) as a function of building parameters. First published in Tsanas and Xifara [2012].
  - Inputs: relative compactness, surface area, wall area, roof area, overall height, orientation, glazing area, glazing area distribution.
  - Outputs: heating load (energy 1), cooling load (energy 2).
- kin8nm:
  - This data set is concerned with the forward kinematics of an 8 link all-revolute robot arm. Among the existing variants of this data set, the variant 8nm which

is known to be highly non-linear and moderately noisy is used. The goal is to predict the distance of the end-effector from a target, given the twist angles of the 8 links as features. There is no known first publication of this data set, see also Vanschoren et al. [2014].

  - Inputs: 8 twist angles of the 8 links.
  - Output: distance of the end-effector from a target.

- power:
  - The data set contains 9,568 data points collected from a combined cycle power plant over 6 years (2006 - 2011), when the power plant was set to work with full load. First published in Tüfekci [2014].
  - Inputs: (hourly average ambient variables) temperature, ambient pressure, relative humidity and exhaust vacuum.
  - Output: net hourly electrical energy output of the plant.

- red & and white wine:
  - Two data sets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model the wine quality based on physicochemical tests. First published in Cortez et al. [2009].
  - Inputs: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol.
  - Output: quality (a score between 0 and 10)

Example	N	n	$\mathcal{DGCN}$	$\mathcal{DGP}$	$drop_{DNN}$
concrete	1,030	8	$\mathbf{3.67 \pm 0.13}$	5.21 ± 0.9	4.50 ± 0.18
energy 1	768	8	$\mathbf{0.33 \pm 0.01}$	0.48 ± 0.05	-
energy 2	768	8	$\mathbf{0.57 \pm 0.01}$	1.37 ± 0.23	-
kin8nm	8,192	8	0.06 ± 0.00	$\mathbf{0.02 \pm 0.00}$	0.08 ± 0.00
power	9,568	4	$\mathbf{2.91 \pm 0.00}$	2.95 ± 0.3	3.63 ± 0.04
red wine	1,588	11	$\mathbf{0.50 \pm 0.00}$	0.54 ± 0.11	0.60 ± 0.01
white wine	4,898	11	0.67 ± 0.00	$\mathbf{0.34 \pm 0.07}$	-

Table 2.20: Comparison of deep Gaussian process, drop out neural network and deep Gaussian covariance network based on mean and standard deviation $RMSE$ values for the UCI database examples.

The third experiment is a multi-output example for steam turbines. It has already been shown that the multi-output extension can be used to predict the values of a finite element analysis, where the outputs are correlated in space. There are also structural problems which are not only dependent on space but also on time. The estimation of creep strain is

such an example, where the strain of a material depends on the temperature and pressure history over time. This kind of analysis takes a lot of time to simulate. Therefore, a $\mathcal{DGCN}$ model was trained on 300 simulations to be used for further calculations of the creep strain. In this case, a probabilistic analysis was performed where about $14,000$ calculations were needed. To be more specific, the results of 31 finite element nodes in the rotor center (see fig. 2.94 orange / red area) over 34 time steps, so $1,054$ outputs were used. To handle the correlation of time and space, the results were all aligned to a single output matrix (300 rows $1,054$ columns), like the previous multi-output examples. Since the order of the outputs is not relevant for the $\mathcal{DGCN}$ model to learn the output correlations, this was a valid approach for this problem. The $1,054$ outputs were reduced via PCA to 50 principle components with an explained variance of 0.999999. The relationship of the creep strain over space and time is shown exemplary for one combination of the 4 varied input parameters describing the temperature and pressure on the rotor in fig. 2.95. Additional 21 test samples were created to validate the model. The result of this validation in comparison to other algorithms (neural networks (NN), random forest (RF), XGboost (XGB), decision tree (DTR)) is shown in tab 2.21. For the XGboost algorithm, which is not able to learn multi-output correlations, one model per output was built. The comparison is made between the case where PCA was used as a pre-processing step and those case where PCA was not use as a pre-processing. The $\mathcal{DGCN}$ is not applicable without PCA on this problem because the number of outputs is too high (with the available hardware). As a validation metric, the mean $MAPE$ ($MMAPE$) value (see eq. 2.6) is used, so the relative absolute percentage error over all time and space combinations, averaged over all 21 test cases.

The results show that $\mathcal{DGCN}$ outperforms every algorithm by far with or without PCA pre-processing. For one test example the contour plot of the creep strain over space and time is shown in fig. 2.96 with PCA and in fig. 2.97 without PCA for the algorithms tested. Especially the neural network has huge problems with this example. All contour plots were normalized on the same range for comparison reasons, therefore some show nearly no different colors. In general, due to confidentiality reasons, no values are allowed to be shown on the axes.

Algorithm	$MMAPE$ with PCA	$MMAPE$ without PCA
$\mathcal{DGCN}$	**1.71**	-
XGB	36	**31**
RF	289	**253**
DTR	**396**	1265
NN	218065	**121321**

Table 2.21: Comparison of $\mathcal{DGCN}$, neural networks (NN), random forest (RF), XGboost (XGB), decision tree (DTR) with PCA and without PCA for 21 test cases of the creep strain approximation. The mean $MAPE$ ($MMAPE$) metric in [%] over all 21 multi-output sequences are estimated. Bold values indicate the best result between using PCA and not using PCA.

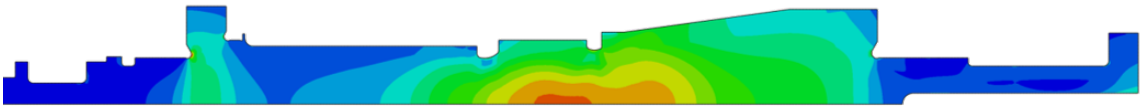


Figure 2.94: Visualization of the rotor to be analyzed.

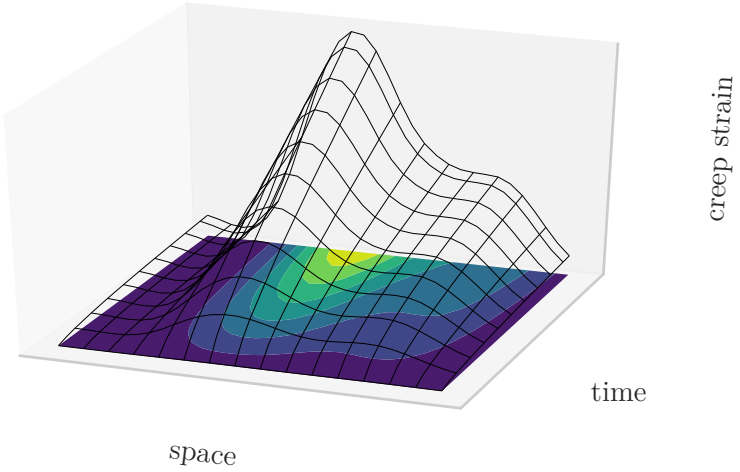


Figure 2.95: Creep strain over time and space.

A second multi-output example for the steam turbines application area is the usage of $\mathcal{DGCN}$ to predict calibration parameters for a material FEM simulation model. The objective is to simulate the material strength properties based on experimental results of a so-called small-punch test using FEM (see fig. 2.98). In this test, a stamp is pressed on a material sample until it fails. The result of such a test is a force vs. displacement diagram, as shown in fig. 2.99. The calibrated FEM model should be able to give the same curve as recorded in the experiments so that the real experiment no longer needs to be carried out. One way to do this, is to optimize the calibration parameters on the FEM model until the residuals between FEM and experiment reach a target difference. This is a very time-consuming process and must be repeated for every new material type. Therefore, the FEM-based curve calculation should be replaced by a multi-output $\mathcal{DGCN}$ model. So that the time consuming optimization of the FEM model to find the optimal set of parameters can be done on the $\mathcal{DGCN}$ model instead. The parameters from the optimization found on the $\mathcal{DGCN}$ model can then also be used for the FEM simulation. The $DGCN$ model takes as input the calibration parameters and should predict the curve shown in fig. 2.99. The optimization task is to find those parameters that the predicted curve of the $\mathcal{DGCN}$ model is as close as possible to the experimental curves. The curve in fig. 2.99 is discretized by $1,000$ steps. Overall there are 7 parameters to calibrate the FEM model and to be used as input for the $\mathcal{DGCN}$ model. The parameters are forces and material parameters like the yield strength or the strain hardening exponent. To train the $\mathcal{DGCN}$ model $1,000$ samples were created via the FEM within the maximal calibration parameter range. The $1,000$ outputs are compressed to 8 components via an autoencoder (see fig. 2.76). After

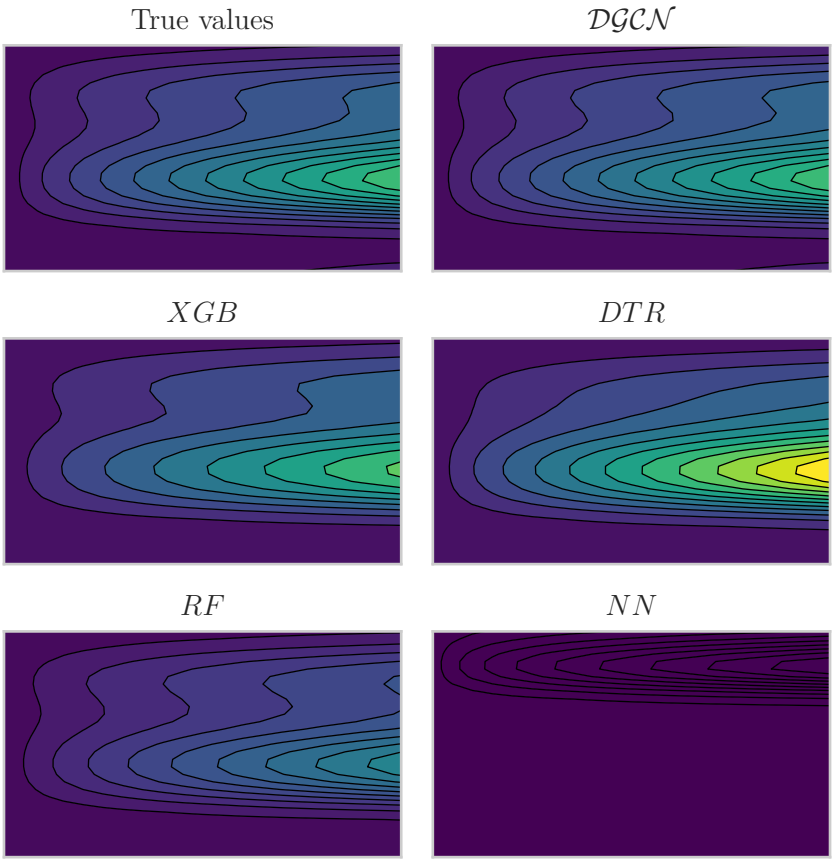


Figure 2.96: Comparison of different algorithms for the creep strain approximation over space and time with PCA pre-processing.

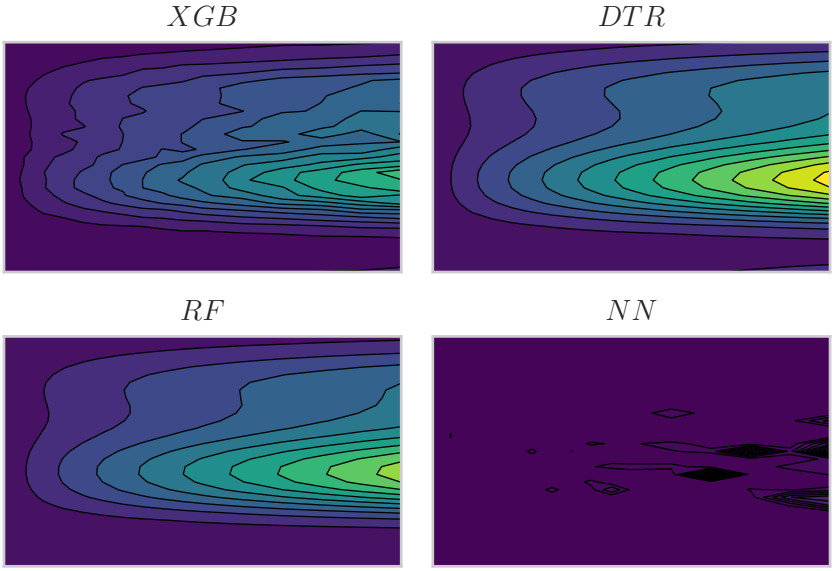


Figure 2.97: Comparison of different algorithms for the creep strain approximation over space and time without PCA pre-processing.

the model is trained, it is used to estimate the optimal calibration parameters based on the experimental curve via optimization. To validate the calibration, the FEM curve based on the optimized parameters, and the experimental curve are compared. To give a visual validation, fig. 2.100 visualizes the reconstruction from the autoencoder compression back to the full $1,000$ steps and fig. 2.101 shows the comparison of a FEM calibrated curve to the experimental curves. Furthermore, tab. 2.22 shows the $MAPE$ error for 6 experimental test curves and the difference to the calibrated FEM curves. The error for every calibration is around 4%. As can be seen in fig. 2.101, the main error contribution comes from the last part of the curve, where the material fails. Since the material fails there, the curve is probably more or less random and therefore difficult to model. The remaining, and also most important, parts are fitted very well. The huge benefit of this approach is that the $\mathcal{DGCN}$ model can be reused for a wide range of different materials without the need to retrain it, as long as the parameter range of the initial training set of the calibration parameters are not exceeded. This allows for a very fast FEM model calibration.

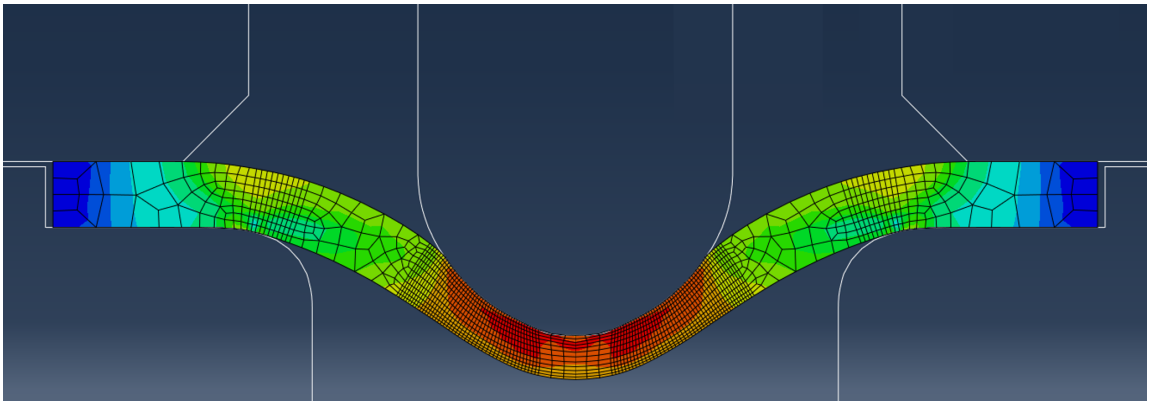


Figure 2.98: Illustration of the small punch test. A stamp is pressed on a material sample until it fails.

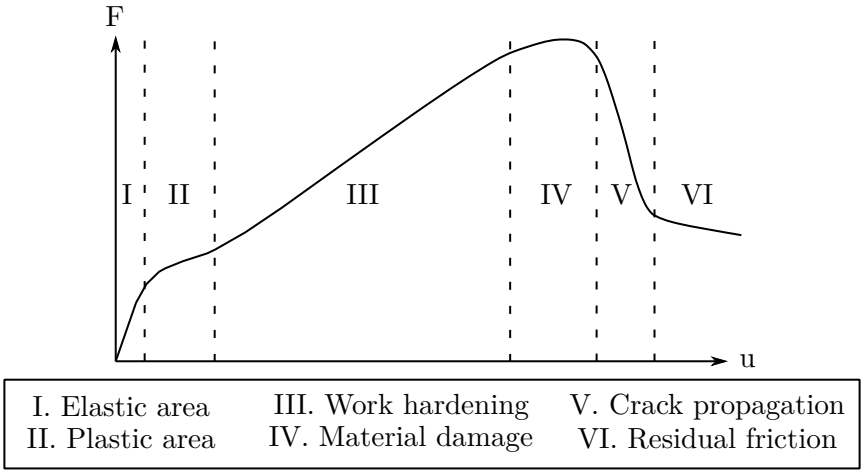


Figure 2.99: Force vs. displacement diagram of a small punch test.

Besides the experiments presented in this work, additional turbomachinery examples (where $\mathcal{DGCN}$ was used) can be found in Cremanns et al. [2018a] and Cremanns et al. [2018b], see also section 1.2. In addition to these, another unpublished turbo machinery

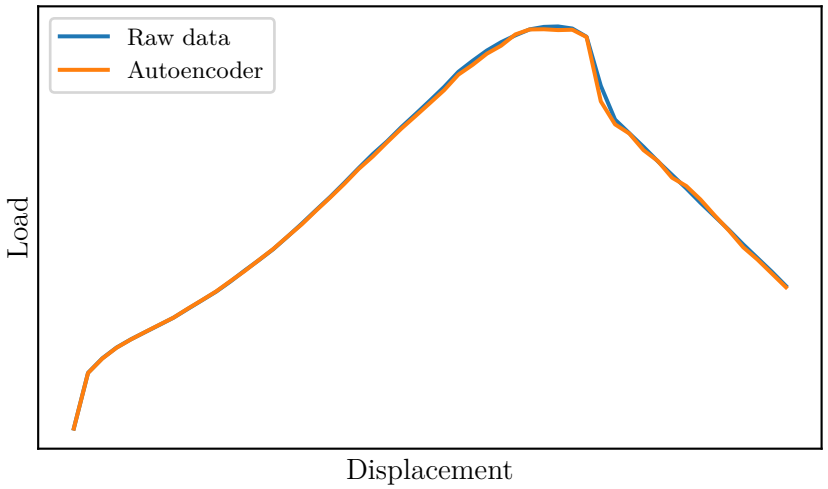


Figure 2.100: Reconstruction of the autoencoder compression to the full 1,000 steps.

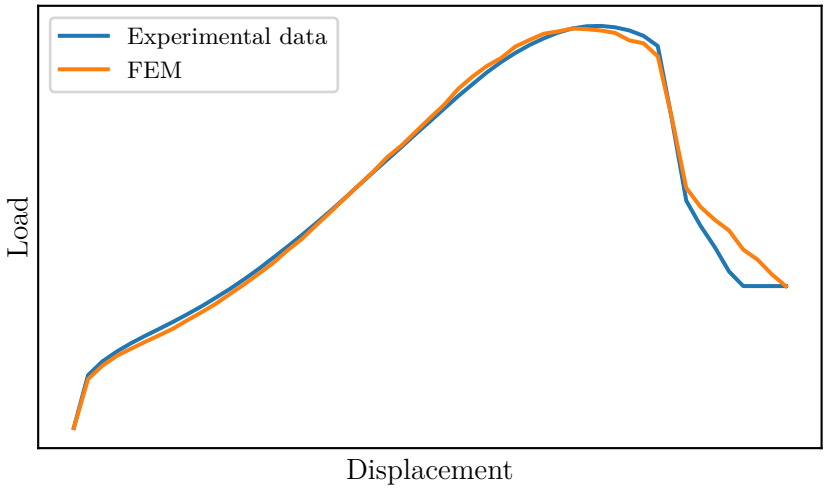


Figure 2.101: Comparison of the calibrated FEM curve vs. experimental curve.

Experiment	$MAPE$
1	5.22
2	4.26
3	4.31
4	3.94
5	3.38
6	4.60

Table 2.22: The $MAPE$ metric in [%] between 7 calibrated FEM curves to the experimental ones.

example is presented in the following to illustrate the strength of surrogate models for product-based optimization. The objective of this example is the optimization of a high-intermediate pressure rotor of a steam turbine used in power plants. Several departments are involved in the development of such a product as depicted in fig 2.102. So, an optimization of its cost and performance is very complex, since every change to the product must be typically analyzed and verified by all departments again. This means an exhaustive optimization is not possible in an acceptable amount of time. For this example, all department specific outputs which are relevant for the rotor optimization, should be replaced by surrogate models. This means that some model outputs are also input for other models. Thus, if the first model predictions have some error, this would be transferred to the following models and maybe even more amplified. Therefore, another approach was used as demonstrated in fig. 2.103. The idea was to let the models learn directly the relationships from the start to the final results describing the rotor. So for example, the mechanical integrity related outputs are modeled by $\mathcal{DGCN}$ models which takes the input of the thermodynamic department instead of taking the output of the thermodynamic department as input. So the intermediate results of the thermodynamics are omitted. This means the models must learn the complex physical relationships over multiple department combinations. For this application all models are $\mathcal{DGCN}$ models. $N = 200$ samples have been created from which only 123 could have been successfully simulated to train the models. Overall, 25 input parameters and 22 output parameters were used (so in total 22 $\mathcal{DGCN}$ models have been trained). The generic input parameter description can also be seen in fig. 2.102 (red marked text). 5 simulation / departments tools have been replaced. The rotor-dynamic tool, which is typically used to evaluate the natural frequencies of the rotor, showed non-critical frequencies for all training samples in the design space of interest and was therefore left out for the optimization. Maybe a larger design space would have lead to critical designs. The objectives of the optimization were to minimize the costs and start times and to maximize the power output of the rotary system. These three objectives are all in contrary, therefore a Pareto-frontier (see e.g. Abbass et al. [2001] or subsection 3.4.2) will be the result of the optimization. Furthermore, 14 constraints needed to be considered, mainly for the mechanical integrity. Tab. 2.23 lists the predictive coefficient of determination $\hat{R}^2$ for every output. All were $> 90\%$ which is quite good for such a complex task. For the optimization the NSGA III (Yuan et al. [2014]) algorithm was used. The resulting 3D Pareto-frontier is given in fig. 2.104. The best percentage improvements for each objective are given in tab. 2.24 (note that these results are not all in one design). One exemplary result of the percentage improvements achievable in one design is shown in tab. 2.25. These values are already the validated ones based on 25 designs which have been picked from the Pareto-frontier to validate the models via the department simulation tools. Therefore, these results are not validated on real physical experiments because the optimized designs have not been built. For this simulation-based validation, the mean and standard deviation of the relative error were estimated over all 25 designs and are given in tab. 2.26. The errors are all quite low and show that with the $\mathcal{DGCN}$ algorithm, it is possible to approximate even complex multi physical relationships very accurately with a low number of training samples.

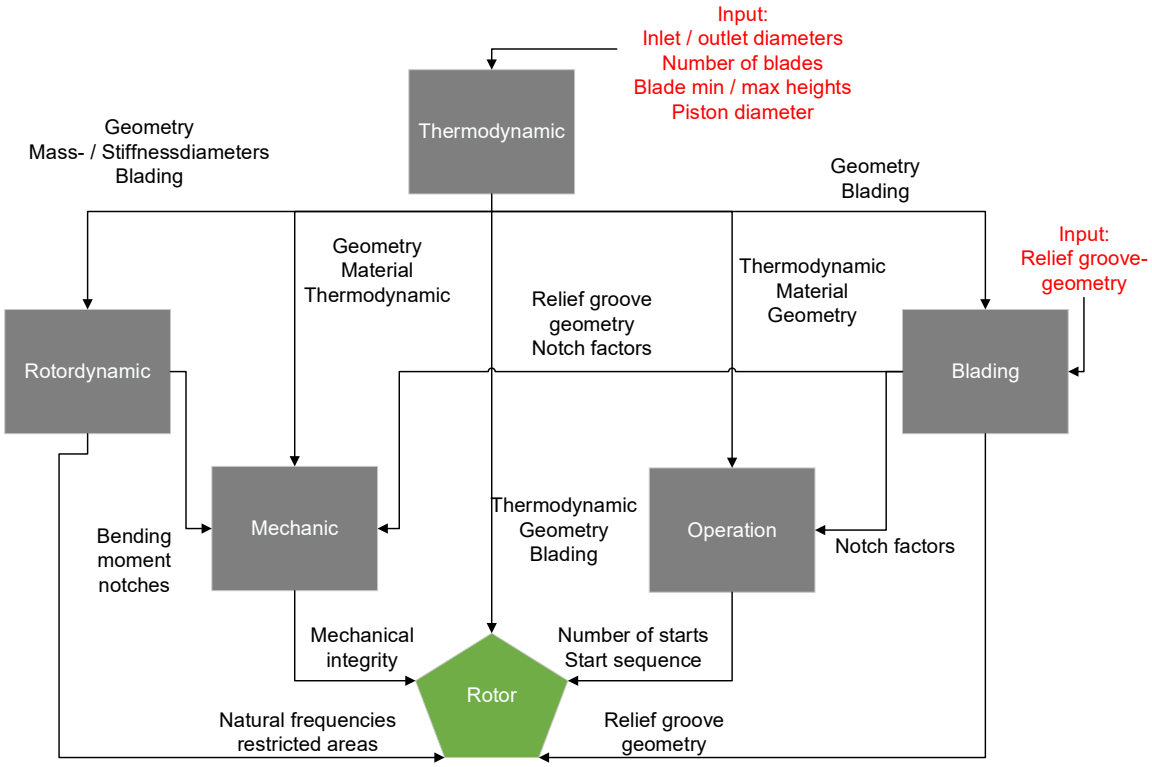


Figure 2.102: Overview of the different departments involved in the design development of a steam turbine rotor. The red marked text gives a generic overview of the input parameters used.

Name	$\hat{R}^2$	Name	$\hat{R}^2$
Cost_1	0.99	NF_HP_6	0.95
Cost_2	0.94	NF_HP_7	0.96
Exergy_loss_1	0.99	NF_IP_4	0.91
Exergy_loss_2	0.96	NF_IP_5	0.92
Exergy_loss_3	0.99	NF_IP_6	0.97
Eta_1	0.99	NF_IP_7	0.93
Eta_2	0.98	MEC_C_1	0.94
Power_1	0.99	MEC_C_2	0.97
Power_2	0.92	MEC_C_3	0.90
NF_HP_4	0.98	MEC_C_4	0.96
NF_HP_5	0.94	Start time	0.97

Table 2.23: The $\hat{R}^2$ metric for the 21 rotor optimization relevant outputs.

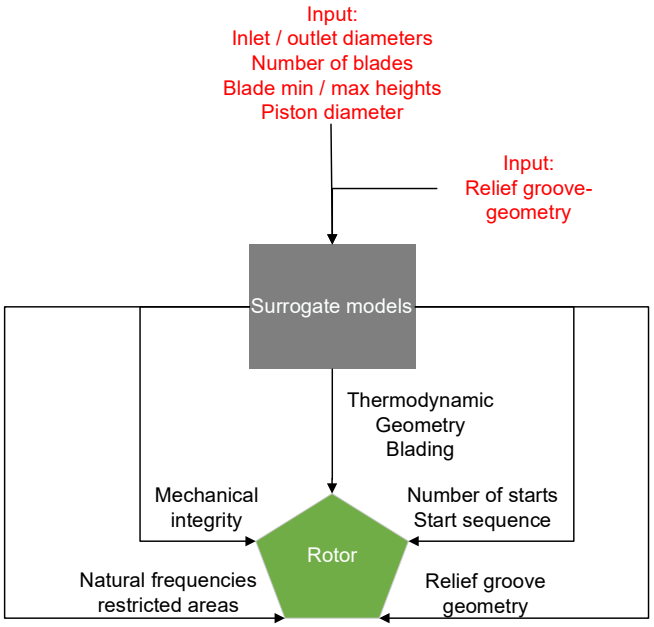


Figure 2.103: Overview of the different departments involved in the design development of a steam turbine rotor. The red marked text gives a generic overview of the input parameters used. Surrogate models should directly learn the multi physical relationships from the input parameters used to the final rotor relevant output parameters.

Name	Improvement [%]
Costs	-14.9%
Power	0.07% ($\approx$ 1.3 MW)
Start time	-23%

Table 2.24: Validated best objective values for each objective (not in one design).

Name	Improvement [%]
Costs	-3%
Power	0.05% ($\approx$ 0.92 MW)
Start time	-4%

Table 2.25: Validated objective values of one design (one point of the Pareto-frontier).

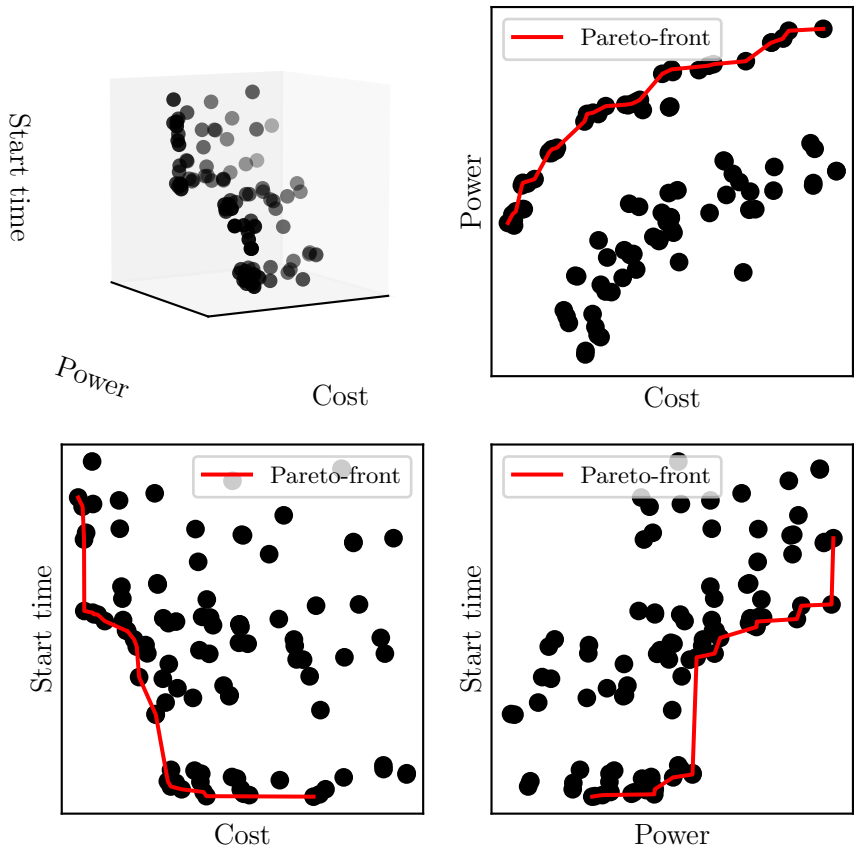


Figure 2.104: 3D and all 2D Pareto-frontier combinations of the rotor optimization. Darker points are more close than gray points to visualize the third dimension.

Name	Mean rel. error[%]	Std. rel. error[%]	Name	Mean rel. error[%]	Std. rel. error [%]
Cost_1	3.70	0.89	NF_HP_6	4.43	1.38
Cost_2	3.30	2.14	NF_HP_7	2.17	1.19
Exergy_loss_1	1.80	0.51	NF_IP_4	6.94	2.36
Exergy_loss_2	4.88	2.37	NF_IP_5	5.09	2.45
Exergy_loss_3	2.55	0.99	NF_IP_6	2.29	0.90
Eta_1	0.22	0.11	NF_IP_7	7.11	1.19
Eta_2	0.52	0.31	MEC_C_1	9.76	1.54
Power_1	0.22	0.10	MEC_C_2	1.69	1.19
Power_2	0.43	0.24	MEC_C_3	7.15	2.39
NF_HP_4	1.05	0.83	MEC_C_4	3.99	1.92
NF_HP_5	3.22	2.63	Start time	2.71	1.67

Table 2.26: Mean and standard deviation of the relative error estimated via 25 designs for validation.

2.10.9 *Conclusion*

For the conclusion, tab. 2.5 should be recapped and extended by the $\mathcal{DGCN}$ algorithm in tab. 2.27.

Criteria	RSM	MLS	$\mathcal{GP}$	DNN	$\mathcal{DGCN}$
Regression	+	+	+	+	+
Classification	-	-	+	+	+
Image classification	-	-	-	+	+
Sequential problems	-	-	-	+	+
Multi-output (signals)	-	-	+	+	+
Big data (number of samples)	+	-	-	+	+
Online learning	-	-	-	+	+
Confidence intervals	+	+	+	-	+
Automatic variable selection	+	+	+	+	+
Trainable noise	-	-	+	-	+
Robust against outlier	+	-	+	-	+
No need of uniform distributed sampling	+	-	-	+	+
Avoid tendency to over-fit	+	-	-	-	+
Non-stationarity	-	+	+	-	+
Discrete input parameters	-	-	-	+	+
No user defined hyperparameters	-	-	-	-	+
Reduced heteroscedasticity	-	+	+	-	+

Table 2.27: Overview of algorithms capabilities.

Since the $\mathcal{DGCN}$ algorithm has the combined strengths of the DNN and $\mathcal{GP}$ algorithm, it takes always a + if one or both of the them has one. Most of these points were supported by examples given in the previous subsections. One important point to mention is that no hyperparameters need to be set for the $\mathcal{DGCN}$ model in order to provide good results which is typically not the case for most machine learning algorithms. Only options which define the task, like regression, classification, image classification or sequential dependent problems are user input. Additionally, the number of sequential previous steps is a user-defined parameter for sequential problems and whether noise should be trained or not. Apart from that, all experiments where done without any special configuration of covariance functions or network topology changes of the DNN part. Still, the model performs very well in comparison to other state of the art methods. where expert knowledge is needed to obtain good results. In many cases $\mathcal{DGCN}$ even outperforms the other algorithms. Furthermore, for methods other than $\mathcal{DGCN}$, it is not enough to have expert knowledge in one algorithm but is needed in several different ones, dependent on the task which should be performed.

All this makes $\mathcal{DGCN}$ not only a very easy to use algorithm for people who have no expert knowledge in several different algorithms, but also a good choice if a high prognosis quality is crucial. It also allows the use of a $\mathcal{GP}$ model for arbitrary data sizes, without loss of information compared to the often-used inducing points method for Gaussian processes. Therefore, it is very well able to handle the increasing demand for bigger data sizes in the future.

Important notes:

- This section presented a novel probabilistic supervised machine learning algorithm called deep Gaussian covariance network $\mathcal{DGCN}$.
- It utilizes two deep neural networks for the free parameters estimation of a $\mathcal{GP}$. Through this, the free parameters can change dependent on the point to be predicted. This makes the $\mathcal{GP}$ non-stationary independent of the type of covariance function used.
- There are two different networks, one for the correlation length-scale parameters (one per input parameter - automatic relevance determination) and one for the noise variance parameter.
- The point-wise noise variance estimation allows the model to handle data with different noise levels.
- The non-stationarity of the length-scales is beneficial for training data with different point densities (e.g. locally adapted samplings) or highly non-linear functions (e.g. with discrete steps).
- Since the right choice of covariance function is important for the accuracy of a $\mathcal{GP}$ model, in $\mathcal{DGCN}$, 5 different covariance functions are trained simultaneously and the resulting covariance matrices are summed up to a final matrix used for training and prediction. The model can learn the optimal combination of covariance function by itself and if needed can exclude unnecessary covariance functions by setting the corresponding length-scales to zero.
- The use of neural networks for the free parameter training allows the model also to use mini-batch training. This enables the model to handle arbitrary big data sets. A current drawback is the prediction time if mini-batch training is used. With the help of a nearest neighbor algorithm, temporary training covariance matrices are built. This is accelerated with the help of parallel computing.
- The implementation in Tensorflow allows the model to be trained very quickly and easily on GPU and CPU. Also distributed learning on several GPUs / CPUs is possible.
- The base algorithm has several extensions to perform different learning tasks. It can handle regression, (image-)classification, multi-output, sequential dependent problems.

- In several examples these capabilities were demonstrated, also that the $\mathcal{DGCN}$ algorithm is on par or even better than other state of the art methods.
- Another advantage of that algorithm is that is has no hyperparameter to be set by the user. Therefore, it is a good algorithm for people with less knowledge in machine learning. Through the capabilities of performing several different learning tasks, it reduces also the amount of needed algorithm knowledge in general.

2.11 MODEL VALIDATION

The last technical section of the supervised machine learning chapter should, for completeness, show some further methods on how to validate trained models. In machine learning, model validation is typically referred to as the process where a trained model is evaluated with a testing data set. The testing data set is a separate portion of the same data set from which the training set is derived. The main purpose of using the testing data set is to test the generalization ability of a trained model (Alpaydin [2014]). Sometimes additional test data is not available and other ways presented in the following can be used to get an impression of the generalization ability of a trained model. Some possible methods, like cross-validation (which is mostly used if no test data is available) and prognosis metrics, were already presented in section 2.2. These metrics of course, can also be used after a model has been trained for validation. It was also shown in section 2.2 that the cross-validated metrics are not very reliable if the number of samples is low or a non-stratified package scheme is used.

2.11.1 *Test Data Validation*

Besides the usage of cross-validation, there is also the possibility to split the data into a separate training and test set or even to create an extra test set. In the following, some prerequisites and rules for a model validation via a test set based on experience and recommendations of Ng [2017] will be given:

- The test set should cover the whole design space and not only a part of it. If it is a classification task, all classes should be represented equally.
- The test set should be drawn from the same distribution as the training set.
- If scaling was used for the training data like, for example, normalization, the test set needs to be scaled with the mean and standard deviation obtained from the training set. Therefore, the scaling parameters need to be determined on the training set and only applied to the test set.
- All error metrics always need to be evaluated in the original space and not in the transformed space.
- The bigger the test sample size is, the more reliable is the validation of the model. Typically test set sizes range from $1,000$ to $10,000$. If test sets are too small, possible

differences between models cannot be detected. For example, if a classifier A has 90% accuracy and classifier B has 90.1% then a test set of 100 samples would not be able find the 0.1% difference.

- The test set should reflect the data which should be predicted with model.
- The test set should not contain outliers.
- If the training data has some noise, the test data should have a similar noise level.

Visualization of the test results is also very helpful. Assuming a test set is predicted, a residual plot is a common choice to visualize the results. In the following, the standardized residual should be used which is defined by:

$$\epsilon_{istd} = \frac{\epsilon_i}{\sqrt{\frac{1}{N_*-1}\sum_{i=1}^{N_*}\epsilon_i^2}} \tag{2.140}$$

with N_* denoting the number of test points and ϵ_i denotes the i-th residual:

$$\epsilon_i = \hat{y}_i - y_i \tag{2.141}$$

i.e. the difference between the prediction of a model and the true observation. From these kinds of plots several things can be diagnosed which might lead to improvements for the model retrain (Larsen and McCleary [1972], Wang [1987], Fernandez [1992]). In fig. 2.105 an ideal residual plot is shown. It is ideal because:

- It is symmetrically distributed on the y-axis with the tendency to cluster around $\epsilon_{std} = 0$. If no residuals were present, the points would lie on a horizontal line at $\epsilon_{std} = 0$.
- The standardized residuals acc. to eq. 2.141 are generally close to zero. Values between -2 and 2 are typically acceptable. Values smaller / larger than -4 / 4 indicate that this sample might be an outlier (Chaloner and Brant [1988]).
- There is no clear pattern or tendency visible (heteroscedastic error).

If the residual plot looks like this, the remaining residuals are due to the machine learning algorithm choice or maybe due to noisy training data and the error cannot be easily further decreased. Nevertheless, residual plots might also look different, for which some examples will be given in the following.

In fig. 2.106 the residuals are distributed asymmetrically in the direction of the y-axis. The reason for this might be that some observations in the training data have very high / low values compared to the majority of the values. Therefore, the trained model might fit through the majority of points and miss the very high / low values. An easy and likely solution to fix this, is to transform the observations, for example by taking the $log()$ of the values before the model is be trained. Another reason might be an inadequate model for example with missing input parameters which explain the behavior of possible high / low values.

Fig. 2.107 exhibit heteroscedasticity (which was already mentioned in section 2.3), meaning that the residuals get larger as the prediction moves from small to large (or from large

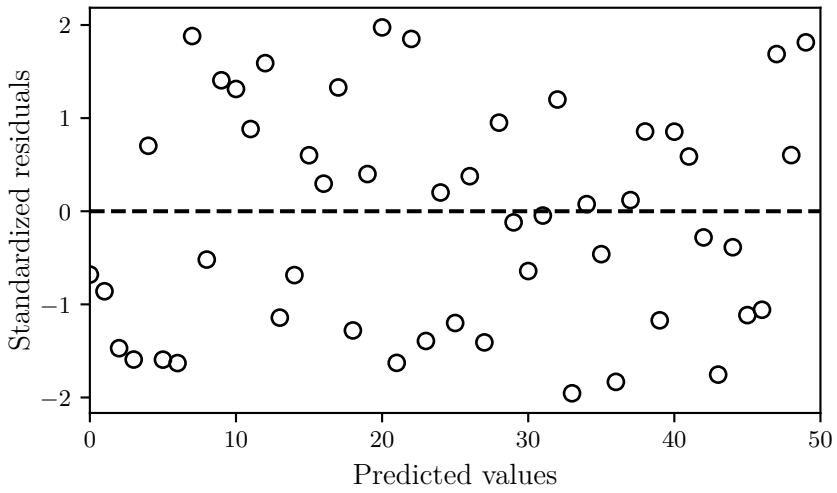


Figure 2.105: Residual plot without anomalies.

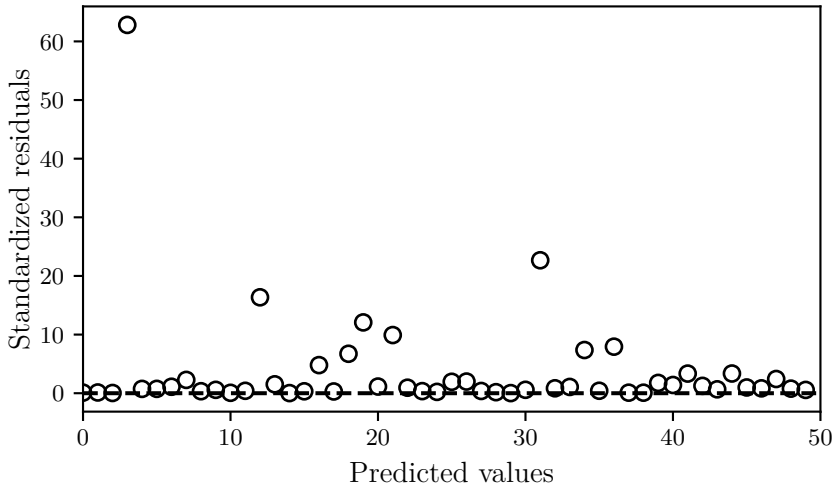


Figure 2.106: Residual plot with unbalanced y-axis values.

to small). If this effect is not due to an inadequacy of the underlying model, then the reason for this is often that for certain parameter values the output variables suddenly change (rise / fall) much more strongly than they do for the most part in the rest of the data set. To reduce the residuals, the solution again is to transform the observations, or by adding a possibly missing input parameter.

The next plot 2.108 indicates that there is some non-linearity not captured by the model. Hence, one solution would be to use another model which is capable of approximating the non-linearity better or, once again, a transformation could be useful. For example, in this case it looks like a quadratic relationship which can be transformed to a linear relationship by taking the square-root of the observations. Then it would also be possible to use a linear model to fit the data.

Outliers as shown in fig. 2.109 might drastically influence the training of a model. If these points are physically real outliers, then obviously the model did not over-fit the training

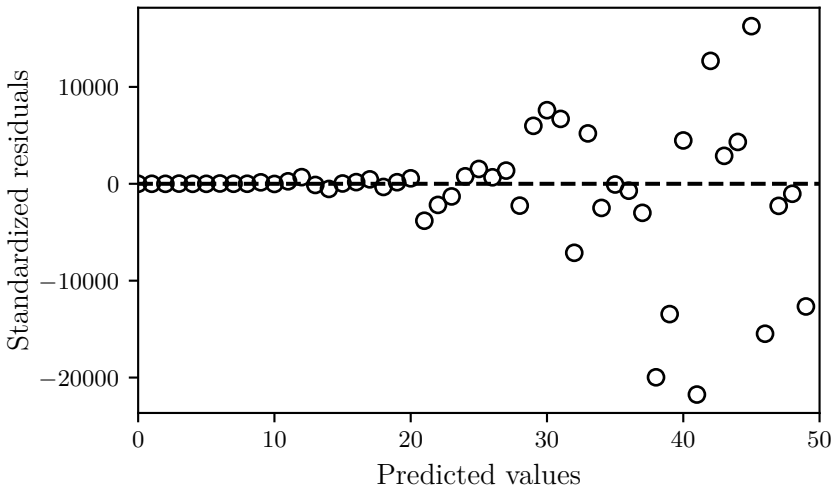


Figure 2.107: Heteroscedastic residuals.

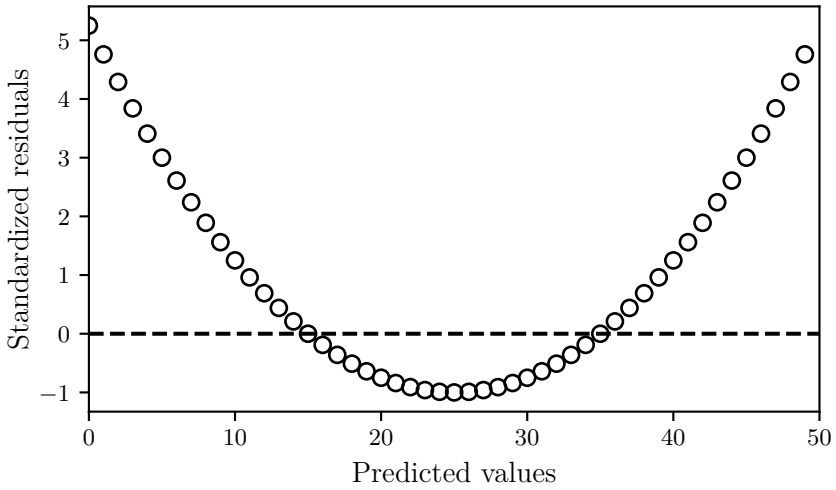


Figure 2.108: Non-linear data residual plot.

data. Nevertheless, these points should be inspected and removed before training if it makes sense. There are algorithms like the local outlier factor (Breunig et al. [2000]) which can be used to identify outliers automatically or give the user hints. Another possibility is that what appears to be just a couple outliers is, in fact, due to an exponential non-linearity. Therefore, transformation might help if one of the variables has an asymmetric distribution.

The next example in fig. 2.110 shows a residual plot where the observations have some small / large values dependent on the input parameters. These can be seen as some discrete or even binary on / off parameters which in some combinations deliver very high y values and in other combinations very low values but most of the time a value in-between. The reason that the model does not fit these high / low values would be very likely that the model has parameters entirely or partly missing from the model. Therefore, a solution would be to include them.

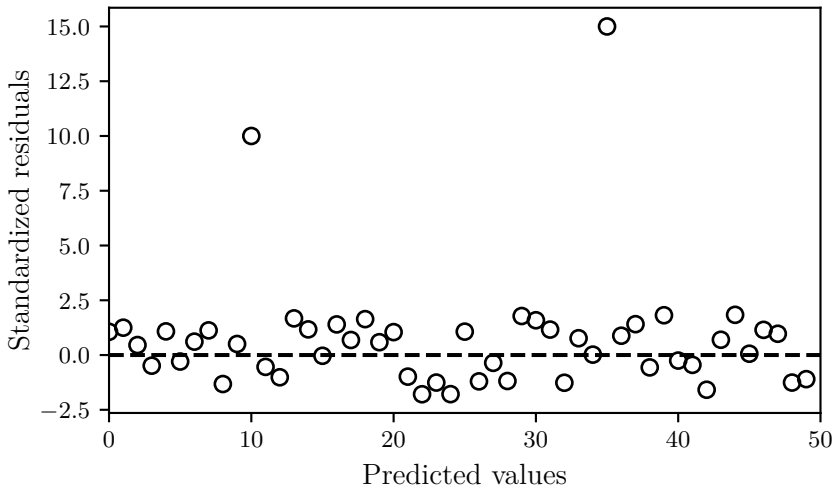


Figure 2.109: Residual plot with outlier.

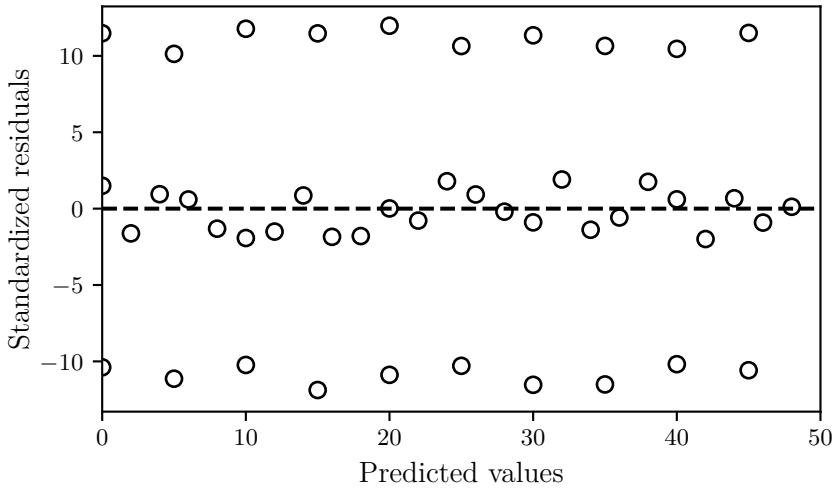


Figure 2.110: Residual plot with large y data.

The last example is shown in fig. 2.111, where this time the data points are located asymmetrically along the x-axis. The reason for such a behavior might be that most training samples have similar y values, only some few samples have a very different one. This might be due to a non-uniform sampling or, in other words, very similar training data points. A solution would be either to create more different sample variants or to exclude those samples with a very different y-value which might lead to reduced parameter limits.

Overall, it is always good to check the residual plots for anomalies. In most cases transformation of the output parameter can help to improve the results. These plots can also give some hints if input parameters are missing. For the transformation of the output, the Box and Cox [1964] power transformation is often used in statistics to map data from any distribution to a Gaussian distribution in order to stabilize variance and minimize skewness:

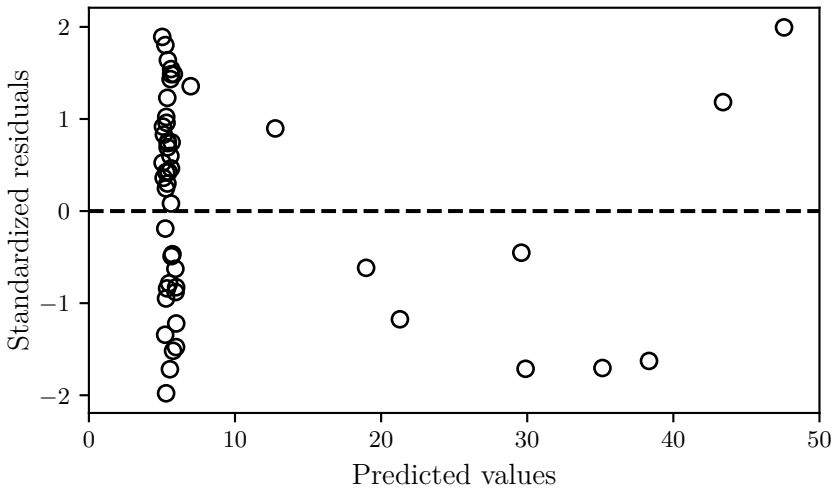


Figure 2.111: Residual plot with unbalanced x-axis values.

$$y_\lambda = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln(y) & \lambda = 0 \end{cases} \tag{2.142}$$

The back-transformation is performed via

$$y = \begin{cases} (y_\lambda \lambda + 1)^{\frac{1}{\lambda}} & \lambda \neq 0 \\ \exp(y_\lambda) & \lambda = 0 \end{cases} \tag{2.143}$$

It is worth mentioning that the data must be positive to perform the transformation and might therefore be shifted by an additional constant term. However, this is a purely mathematical solution which may not be physically permissible. The parameter λ can be estimated for example via cross-validation and a specific metric or also via maximization of the log-likelihood:

$$\ln p(\boldsymbol{y}|\lambda) = -\frac{N}{2} ln(\sigma_\lambda^2) + (\lambda - 1) \sum_{i=1}^{N} ln(y_{\lambda i}) \tag{2.144}$$

with σ_λ^2 being the population variance of the transformed $y_{\lambda i}$ values. The maximization of the log-likelihood can be used in case a Gaussian distribution of the data is desired. See Box and Cox [1964] for more details. Fig. 2.112 shows a quadratic function transformed for different values of λ. It can be seen that a $\lambda = 0.5$ (square root) transforms the quadratic function to a linear function which would be easier to approximate with a linear model. Alternatively, if the data includes negative values, the Yeo-Johnson power transformation (Yeo and Johnson [2000]) can be used:

$$y_\lambda = \begin{cases} ((y+1)^\lambda - 1)/\lambda & \lambda \neq 0,\ y \geq 0 \\ \ln(y) + 1 & \lambda = 0,\ y \geq 0 \\ -((-y+1)^{2-\lambda} - 1)/(2-\lambda) & \lambda \neq 2,\ y < 0 \\ -\ln(-y+1) & \lambda = 2,\ y < 0 \end{cases} \tag{2.145}$$

with its back-transformation:

$$y = \begin{cases} (y_\lambda \lambda + 1)^{1/\lambda} - 1 & \lambda \neq 0,\ y \geq 0 \\ \exp(y_\lambda) - 1 & \lambda = 0,\ y \geq 0 \\ 1 - (-(2-\lambda)y_\lambda + 1)^{\frac{1}{2-\lambda}} & \lambda \neq 2,\ y < 0 \\ 1 - \exp(-y_\lambda) & \lambda = 2,\ y < 0 \end{cases} \tag{2.146}$$

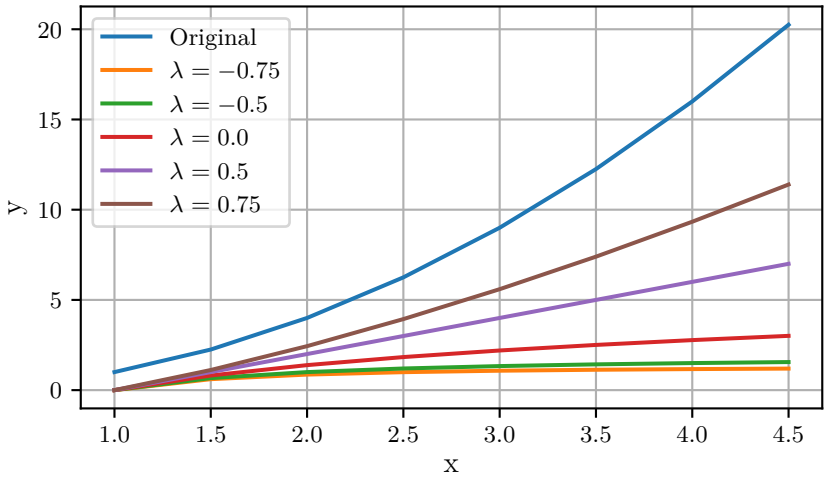


Figure 2.112: Example of a quadratic function transformed via Box-Cox power transformation for different values of λ.

2.11.2 *Confidence Intervals*

Another method which is not often used for model validation, is the usage of confidence intervals. The reason why this method is not often used is that not all algorithms are able to give a confidence interval as, for example, Gaussian processes or response surface method can do. Since the calculation of confidence intervals was already explained in section 2.5 for RSM, section 2.8 for $\mathcal{GP}$ and section 2.10 for $\mathcal{DGCN}$, only two examples of their usage for model validation will be given. For this, the modified Branin-Hoo function of subsection 2.2.1 is used. In a first step 30 samples are created uniformly distributed in the original parameter limits of $x_1 \in \mathbb{R}^{[-5,10]}$, $x_2 \in \mathbb{R}^{[0,15]}$. In a second step, a local refinement with 5 additional samples is made in the area with the lowest y values found on the model. The bounds for this local sampling are in $x_1 \in \mathbb{R}^{[-5,-3]}$, $x_2 \in \mathbb{R}^{[13,15]}$. Fig. 2.113 and

fig. 2.114 illustrate the results of the $\mathcal{DGCN}$ model and the corresponding 95% confidence. Also, the predictive coefficient of determination $\hat{R}^2$ is estimated via 10-fold cross-validation. Fig. 2.114 shows the same example with 5 additional samples included. The additional 5 samples are located in an area, where the confidence of the model was not very high (see fig. 2.113). It can be seen that adding these 5 samples reduces the confidence intervals in that area. If the intention is to obtain an approximation with as few samples as possible, adding additional sample points iteratively in the area of interest and calculating the change of the confidence interval provides an indication if the approximation is accurate enough and the sampling process has converged. It can also be seen that, with only 5 additional samples, the $\hat{R}^2$ does not change because it is a global prognosis metric and does not capture those local refinements of the model very well.

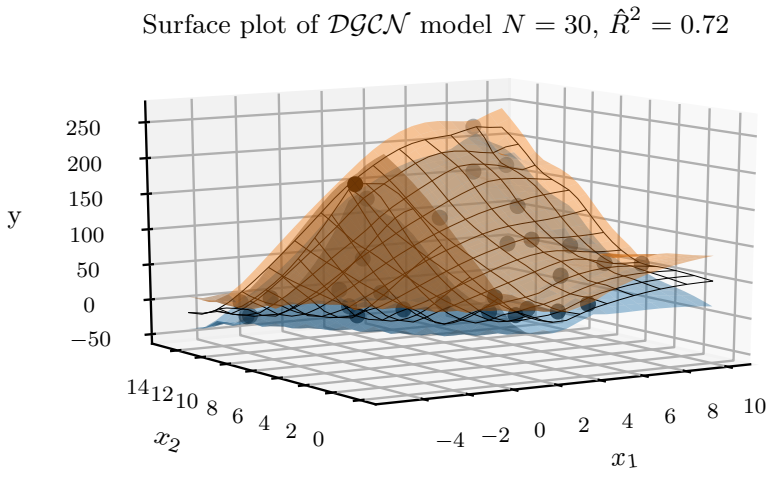


Figure 2.113: Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained with a start *DoE* of 30 samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence interval. Black points show training points.

For the second example, the training sample size will be doubled three times in order to create a more globally accurate model for the modified Branin-Hoo function (eq. 2.7). The sampling method used is described in section 3.4 and called sequential Latin hypercube sampling. For each of the sample sizes, the model surface as well as the 95% confidence interval will be visualized in order to measure the generalization ability of the model. The result is illustrated in fig. 2.115. Additionally, the predictive coefficient of determination is given. The figure demonstrates that the confidence intervals as well as the predictive coefficient of determination show a good model for $N = 40$ samples. Even though a $\hat{R}^2$ value of 99% is given, there are some areas with a higher model uncertainty which is unknown by only using the $\hat{R}^2$ value.

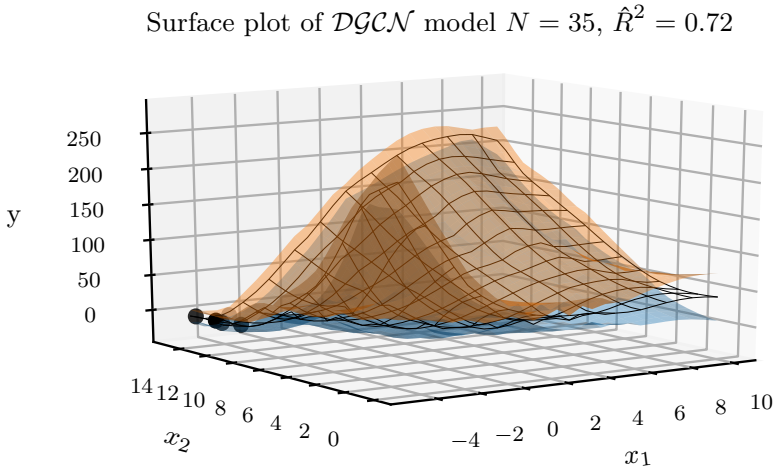


Figure 2.114: Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained with an extended local refined *DoE* of 35 samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence bounds. Here only the additional training samples are plotted (black points).

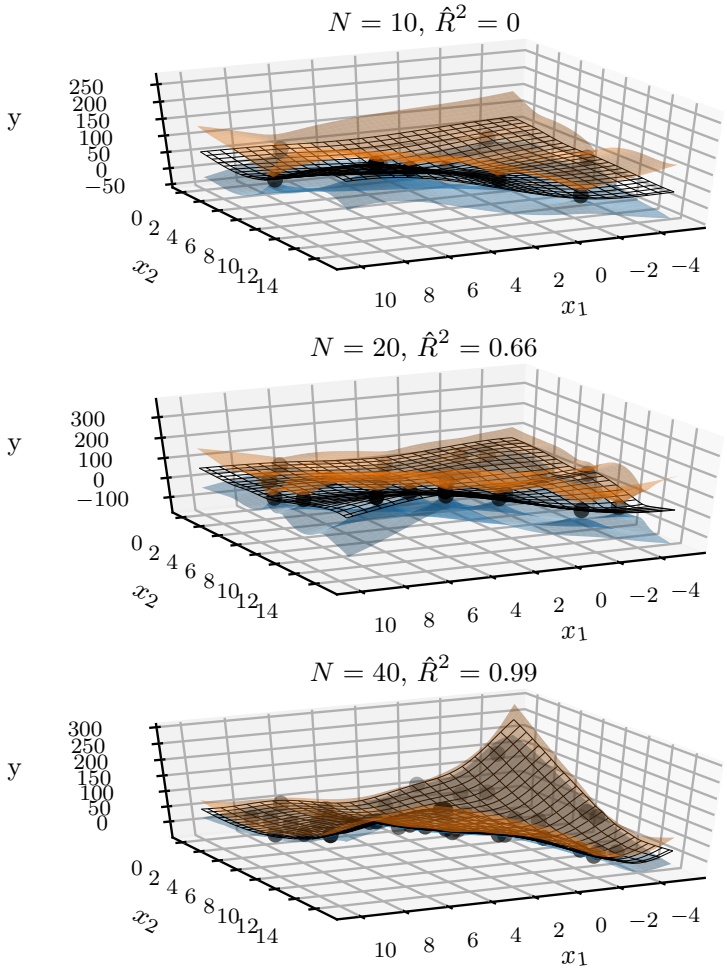


Figure 2.115: Confidence intervals for model validation. A $\mathcal{DGCN}$ model trained three times with $N = 10, 20, 40$ samples. The wireframe shows actual prediction and the colored surface shows upper and lower 95% confidence bounds. Training samples are plotted (black points).

2.11.3 *Confusion Matrix*

Similar to the residual plot which is only applicable for regression tasks (continuous output), a common method also exists for classification tasks to visualize the validation results which is called confusion matrix (error matrix or in unsupervised learning also known as matching matrix) (Godbole and Sarawagi [2004]). In a confusion matrix each row represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes. Fig. 2.116 shows the confusion matrix for the MNIST image classification example (hand-written digits) of subsection 2.10.7. It can be seen, for example, that the digit 1 and 7 or 5 and 3 are often mistaken. In a next step to solve this issue, one way would be to provide more examples of these digits e.g. via data augmentation (Frid-Adar et al. [2018]). The confusion matrix can also be normalized so that instead of the absolute amount of misclassifications the percentage values are given.

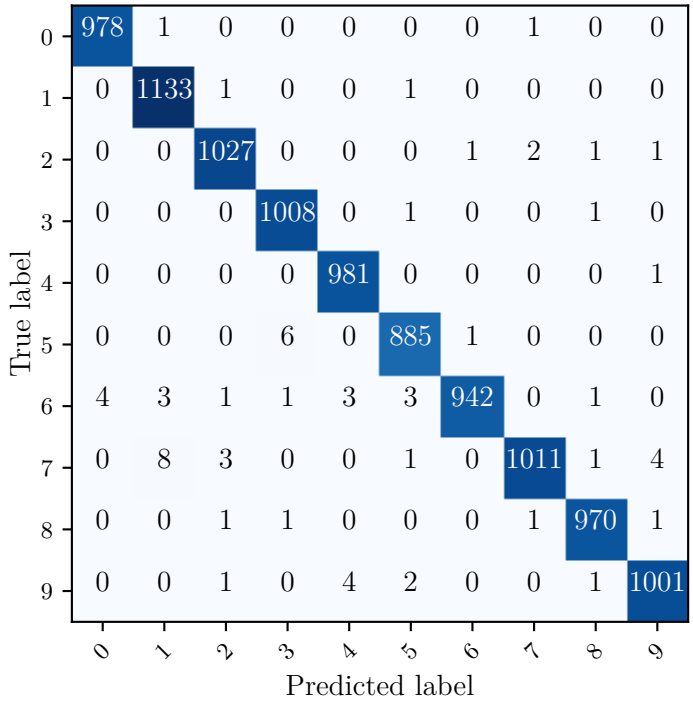


Figure 2.116: Confusion matrix for the MNIST example. Non-diagonal elements show the amount of miss-classified digits.

Important notes:

- Model validation is a very important step in the process of creating surrogate models.
- One of the most reliable methods is to use a secondary test set (if a high enough number of points is given) which must not be used for training. *CV* might also help.

- Residual plots and confidence intervals are helpful tools to visualize the reliability of the model. From a residual plot, it is possible to see which actions might help to improve the model's accuracy, e.g. through data transformation or by including further missing input parameters.
- Confusion matrices are helpful for classification tasks to find possible misclassifications.

2.12 CONCLUSION

As demonstrated in this chapter, there are many different algorithms for supervised machine learning as well as methods to train them and to quantify the generalization ability of the resulting model. The chapter started with a general overview on model selection methods like cross-validation or maximum likelihood estimation for probabilistic models which are applicable for many algorithms. Then, some common model algorithms were explained in detail. The first model type was the RSM (polynomial models) which is useful for linear or quadratic problems but it also introduces the calculation of confidence bounds if the least squares solution is derived of a probabilistic point of view. Since its flexibility is dependent on the regressor types and how many regressor should be included, which is typically a user decision, the MLS method was presented with the novel extension of automatic variable selection. The MLS algorithm introduced the advantages of a local approximating model. It could be demonstrated that this approach makes the RSM much more powerful without the need for a high number of regressors in order to get a higher flexibility of the model (this idea should be reused later for the $\mathcal{DGCN}$ model). In addition to the MLS approach, the Gaussian process was presented which is the state of the art for many machine learning tasks. A detailed explanation, with a probabilistic point of view, was given for this algorithm. The Gaussian process also represents one part of the core of the $\mathcal{DGCN}$ model. Also, the Gaussian process limitations like handling big data were illustrated. Furthermore, deep neural networks were also presented, because they represent the second part of the $\mathcal{DGCN}$ core. They are also state of the art for many tasks, especially for cases where the number of samples are exceedingly large (big data) and computer vision tasks like image classification. However, every algorithm has its advantages and disadvantages. Deep neural networks can solve tasks that $\mathcal{GP}$ models are not capable of solving, and vice versa. Therefore, it was a natural conclusion to combine the strengths of these different algorithms. The product of this process is the deep Gaussian covariance network algorithm. It was shown that this algorithm can combine all positive properties of a $\mathcal{GP}$ and a deep neural network in combination with the idea of local adaptation similar to the MLS algorithm. It could also be demonstrated that this combination eliminates many of the drawbacks of both algorithms. Apart of the aspect of providing a new model with high prognosis quality and usability for many different learning tasks, it was also important that this model is as easy to use as possible. As shown, it is a difficult task to make the right choices for algorithms and hyperparameters which will deliver a model with high accuracy. Using traditional methods, it requires a lot of experience to know which algorithm should be used for which kind of problem and how to train it. Therefore, it was the goal for the $\mathcal{DGCN}$ algorithm to have as few hyperparameters as possible. It was also shown that it is not necessary to find a new optimal net topology of the neural network part

for every problem. It seems that with the proposed net topology, many problems can be solved satisfactorily (shown by the examples in this thesis). Furthermore, it is not necessary to make a decision about the right correlation function, since the model decides this by itself. To prove this, all examples were made under the same configuration to show that it is possible to have state of the art results without investing much time in hyperparameter tuning. Nevertheless, it might be possible that the results can be improved with some different settings. Finally, a short review about model validation methods was given which could prove useful in finding the reason for low prognosis quality of a trained model. It transpired that, in many cases, missing variables or transformation of the output might bring an improvement.

Even though the current state of the $\mathcal{DGCN}$ model works quite well, there are plenty of further aspects to investigate (see also section 5.2). One significant drawback is still the prediction time in combination with the nearest neighbor search if the model is trained via mini batch. In future, this problem might be reduced due to the increasing number of cores in CPUs, since algorithms like the nearest neighbor search can be parallelized, or it might be better to use GPUs for this type of calculation. Another interesting point for future research might be the multi-output extension in combination with recurrent neural networks so that each signal curve is handled like a classical sequentially dependent problem, where only one signal is typically available (see subsection 2.10.4). It would also be interesting to see if a method exists to find the optimal net topology of a deep neural network very efficiently which would then also be useable for the DNN part of the $\mathcal{DGCN}$ model. A first step in this direction is the Neuroevolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen [2002b]) algorithm. This method is based on evolving new types of neural networks using genetic algorithms. So, the neural network topology is found through optimization. Apart from these ideas, every improvement for DNN or $\mathcal{GP}$ models, for example a new covariance function, could also be improvements for the $\mathcal{DGCN}$ model. Since the implementation is based on Tensorflow (Abadi et al. [2015]) all new innovations implemented there, are also easy to implement in the $\mathcal{DGCN}$ model.

3

DESIGN OF EXPERIMENTS

3.1 INTRODUCTION

The correct choice of machine learning algorithm and its settings is only one half of the necessary building blocks to obtain an accurate surrogate model. Maybe even more important is the quality of the training data. It contains the relationship the surrogate model should approximate and might be corrupted by noise and outliers or might be very sparse due to failed design evaluations as illustrated in fig 3.1. Therefore, even though a huge amount of data is available, it might not be useful if the quality is low. So, not only the quantity of the data but also its quality is very important. In most cases it is better to have a lower amount of data with high quality than a huge quantity with inadequate quality. Some quality characteristics for a training data set are:

- Good coverage of the feasible design space.
- Uniform distance between data points (no clusters). Especially for distance-based machine learning algorithms, it is difficult to handle different density levels of the data if they are not non-stationary.
- No unwanted correlations between input parameters. This might lead to a wrong choice of parameters the machine learning algorithm relies on.
- Low noise level.
- No outliers.

These criteria become more important the fewer the number of available samples. An example of a very low quality two dimensional data set is shown in fig 3.2. This data set is correlated, has outliers, sparse areas and different point densities.

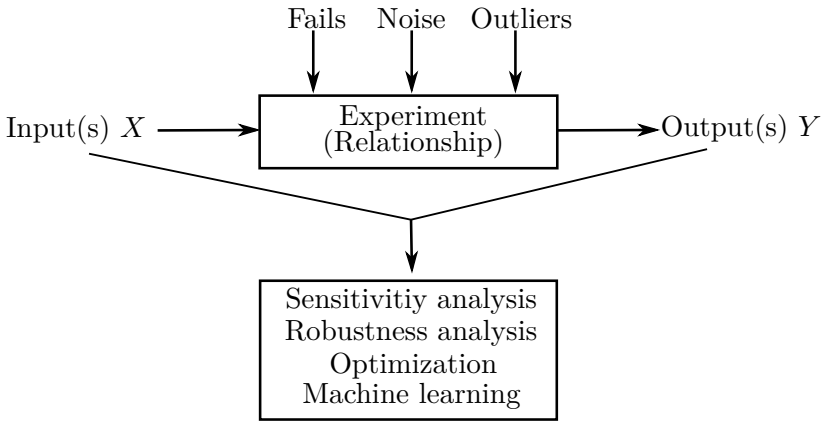


Figure 3.1: Overview of the application of design of experiment and possible interference factors.

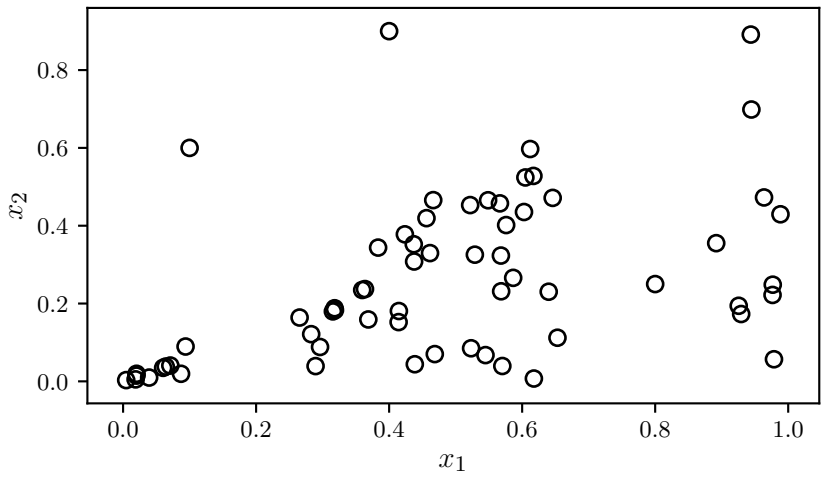


Figure 3.2: Example of a inadequate quality data set.

Typically, there are two situations of how data can be obtained. The first one is data that comes from historical measurements, where the user was not (or only in limited way) able to control how the data was generated. For example, operational data from a steam turbine or the historical course of a stock price. These data might have problems with one or more of the above-mentioned quality characteristics and therefore, it might be much more difficult to create surrogate models for it. A proper pre-processing of the data will be necessary in most cases. The other situation is that the user is able to create a design plan, also known as design of experiment (DoE), in advance, and controls which data points should be evaluated. Having the control over the design generation also allows the control of most of the quality characteristics mentioned except those related to observations like noise or outliers. There are several methods on how to create design plans which will be presented in section 3.2. The Latin hypercube sampling (LHS) is one of them and the method of choice in this work for non-adaptive solutions. This type of design plan can be improved for surrogate model training by optimizing it, as demonstrated in section 3.3. In most cases, it better to use an iterative approach starting with a few design evaluations and adjusting the design space dependent on the user's objectives (e.g. optimization or global model quality). This topic will be covered in section 3.4. As mentioned, sometimes it is only possible to work with measured data which might be correlated, sparse and not uniformly distributed. For sensitivity analysis (section 4) or optimization tasks performed on surrogate models, it might be beneficial to create new samples which have the same distribution and correlation like the historical data inter alia to avoid extrapolation. The corresponding methods are presented in section 3.5. Finally, a conclusion finishes this chapter in section 3.6.

3.2 OVERVIEW OF METHODS

3.2.1 *Systematic Plans*

Traditionally, design of experiments (DoE) are based on systematic plans (Ross and Ross [1988], Box et al. [2005], Montgomery [2017]). This means that the positions of the design

points are not based on a probability or randomly placed but are dependent on the number of input parameters n and sometimes also on the levels l per design parameter. The levels are the points along the axis of a design parameter at which the samples are located. Based on n and l the position of the design points are determined. For example a parameter $x_1 \in \mathbb{R}^{[0,1]}$ and $l = 3$ could have the test values 0, 0.5, 1. Typically, polynomial models are trained in connection with this kind of design plan so that the number of necessary levels depends on the order of the polynomial model. A linear model can be trained with a design plan with $l = 2$, a quadratic model needs a minimum of $l = 3$. Some commonly used methods are the full factorial (fig 3.3)(Youssef et al. [1994]) and central composite design (fig. 3.4)(Ahmadi et al. [2005]) plans. For the full factorial design plan, the number of design evaluations is $N = l^n$, which quickly becomes too large and therefore numerically too expensive to use if n is high. Fig 3.5 illustrates the number of resulting design points N as a function of the number of input parameters n for a two level ($l = 2$) and a three level ($l = 3$) design plan. To reduce the necessary number of evaluations, fractional factorial plans are typically used (Gunst and Mason [2009]). The central composite design can be based on a two-level full factorial plan, like illustrated, (black points) (or a fractional factorial plan), extended by 2 additional points positioned along the axis of each input parameter and one center point (red / yellow points). With the extension of the center point and the axis points fully evaluated, each parameter has 5 levels to test. For a full factorial plan as base, the number of samples needed is $N = 2^n + 2n + 1$. If a fractional factorial plan is used, the number of needed samples is $N = 2^{n-fraction} + 2n + 1$. There are further systematic design plans like:

- Box-Behnken plans (Box and Behnken [1960], Ferreira et al. [2007]), which are derived from reduced full factorial plans with $l = 3$. This type of plan remains three levels for each factor. However, some experimental points are dropped. The omitted test points are selected via the overdetermination of the regression polynomial. The orthogonality of the experimental design remains unaffected. Some numbers of required samples N with one central point are (taken from Box and Behnken [1960]): $n = 3$, $N = 13$; $n = 4$, $N = 25$; $n = 5$, $N = 41$.

- D-optimal plan (Cook and Nachtrheim [1980]) with $N = 1.5(1 + n)$ for linear models and $N = 1.5(1 + n + 0.5n(n + 1))$ for quadratic models. A D-optimal plan is not a stand-alone design plan, but a way to determine the free parameters of a design plan. Here the determinant of the (XX^T)-matrix is optimized (see section 3.3).

- Koshal plan (Box and Lucas [1959]) with $N = 1 + n$ for linear models and $N = 1 + n + 0.5n(n + 1)$ for quadratic models.

All of whose application can be limited because of the input parameter dependency. Since, higher order surrogate models are usually necessary, these kind of design plans are not followed any further in this work. They are mentioned only for completeness and more advanced design plans exist, as demonstrated in the next subsections.

3.2.2 *Stochastic Plans*

Stochastic design plans have, compared to the systematic design plans, typically no predefined design point positions and are therefore placed with some probability or randomness.

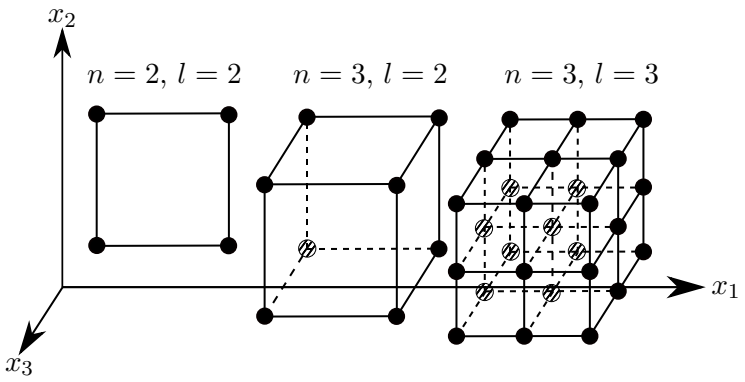


Figure 3.3: Full factorial design plan with $n = 2$, $n = 3$ and $l = 2$, $l = 3$.

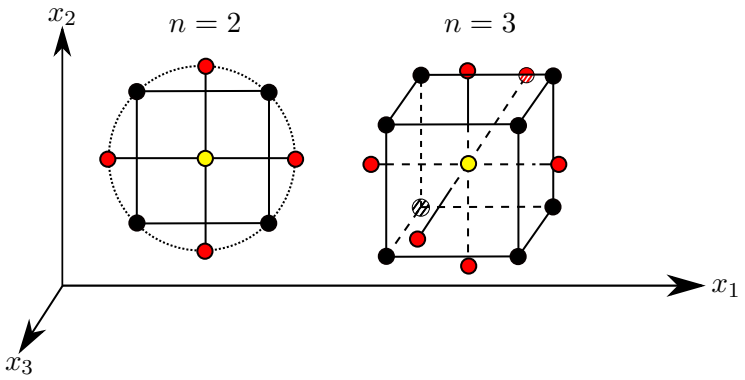


Figure 3.4: Central composite design plan with $n = 2$, $n = 3$.

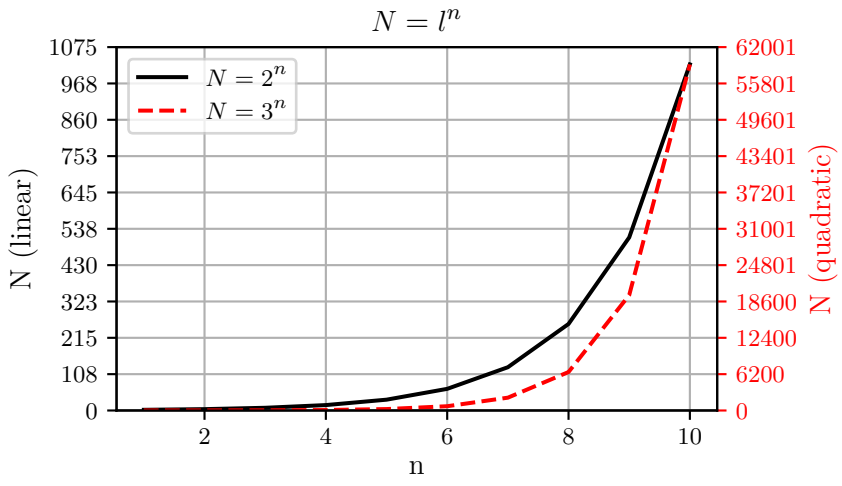


Figure 3.5: Experimental costs for a full factorial design plan with $l = 2$, $l = 3$ and different values for n.

The most famous method which is also used for different types of statistical analysis is the Monte Carlo sampling (MCS) method (Binder et al. [1993]). Monte Carlo methods are mainly used in three problem classes (Kroese et al. [2014]): optimization, numerical integration and generating samples from a probability distribution. The last problem class allows it also to be used for design plan creation. If no specific probability distributions for the input parameters are defined, then a uniform distribution is typically assumed to cover the whole design space homogeneously. So if a design plan with N samples should be generated with the limits x_i^l and x_i^u with $i \in \mathbb{Z}^{[1,n]}$ for the parameters $x_1, ..., x_n$ then N values for each parameter are sampled with the specified distribution. A design point $\boldsymbol{x}_i = [x_1, ..., x_n]$ is generated by randomly taking one value for each parameter of the sampled values without replacement. Thus, the combination of the input parameter values within one design point is also random. The advantage of this method is that it is not dependent on the number of input parameters n or levels l prescribed through the design plan method itself. Every parameter has N levels which allows much more flexible surrogate models to be trained. The number of necessary samples still depends on what kind of surrogate model is used and how many regressors should be included. These are again dependent on the number of input parameters to be used in the model. This method is illustrated in fig. 3.6 for three uniformly distributed parameters.

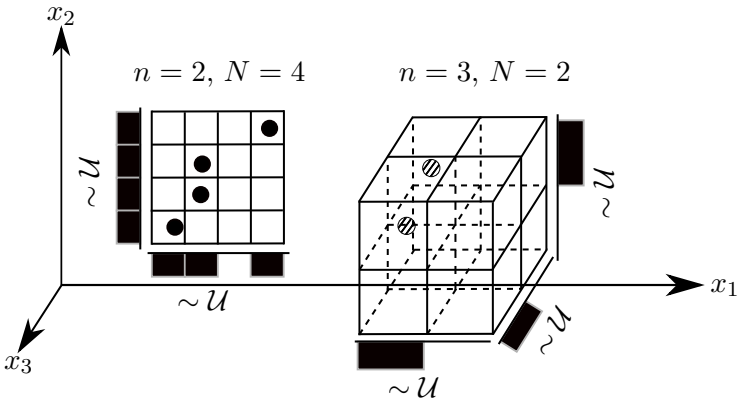


Figure 3.6: Monte Carlo design plan with $n = 2$, $n = 3$ and $N = 4$, 2. The black histograms should illustrate the empirical distribution for the parameters. $\sim \mathcal{U}$ should indicate the target distribution.

An extension of this method is the so-called Latin hypercube sampling (LHS) (McKay et al. [1979], Stein [1987]). It works in a similar way to the MCS with the only difference that for each input variable the axis of each input variable is subdivided into N intervals. The size of the intervals is determined such that each interval has equal probability of occurrence of $1/N$ according to distribution of the input parameter. For 3 input parameters the pattern of intervals form cubes and for more than 3 input parameter one speaks of hypercubes. For each input parameter each interval is populated by only one parameter value. This value can be randomly set within the bin or also be set e.g. to be always in the middle of the bin which is known as Mid-LHS. For the traditional MCS it would be possible that multiple sampling points fall within one hypercube. This is demonstrated in fig. 3.6. The same setup is also illustrated for the LHS in fig. 3.7. Hence, for the LHS, it is typically necessary to know a priori the number of points to be sampled, whereas for the MCS it is

not needed. Thus, for LHS the already sampled points must be considered. In literature, it is often recommended to use LHS over MCS, since it leads to a better coverage of the design space with fewer samples. Also, for robustness analysis, it converges much faster to the true statistical moments (Iman and Helton [1988]). For the LHS, it is also possible to control the correlations between variables as shown in section 3.3. The drawbacks of Monte Carlo methods are possible unwanted correlations between input parameters and a not fully covered design space, due to the random composition of parameter values x_i within one design point as depicted in fig. 3.8. These drawbacks can be eliminated for the LHS if an optimization procedure is coupled to the data point generation which will be investigated in the following section.

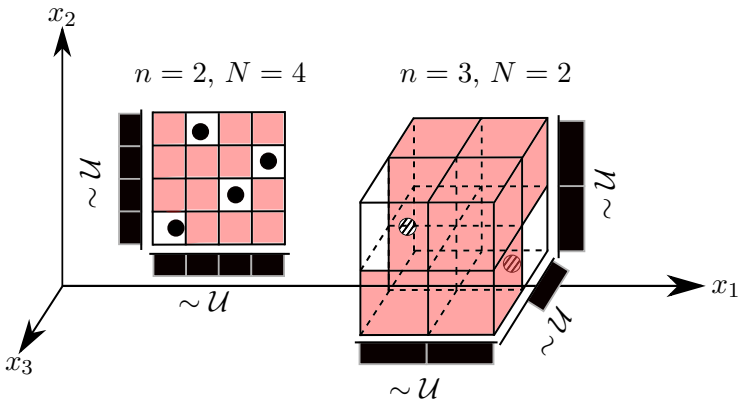


Figure 3.7: Latin hypercube design plan with $n = 2$, $n = 3$ and $N = 4, 2$. The black histograms should illustrate the empirical distribution for the parameters. $\sim \mathcal{U}$ should indicate the target distribution.

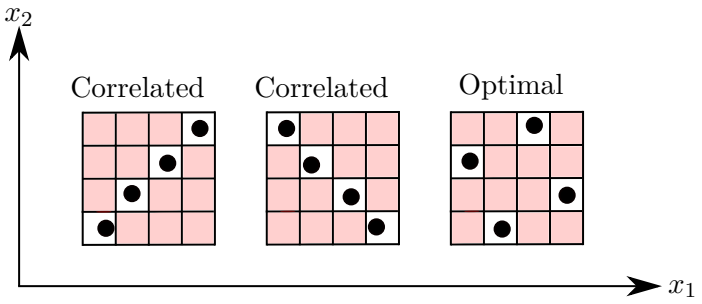


Figure 3.8: Correlated and optimized Latin hypercube design plans.

Another very important property of the LHS is its non-collapsing behavior (orthogonality of the sampling points, Simpson et al. [2001], Kleijnen et al. [2003]) if input parameters are removed from the data set, for example due to a sensitivity analysis. If parameters are removed, the sampling is still a LHS (possibly not as well covered as before). Thus, if it is not possible to create another set of data properly designed for the smaller subspace, the existing data can be reused without a reduction in the number of sampled points. This is not the case if factorial or central composite design plans are used. For those design plans, once input parameters are removed, points collapse one into another which results in a lower number of useful sampling points N (duplicated points). This is illustrated in fig. 3.9.

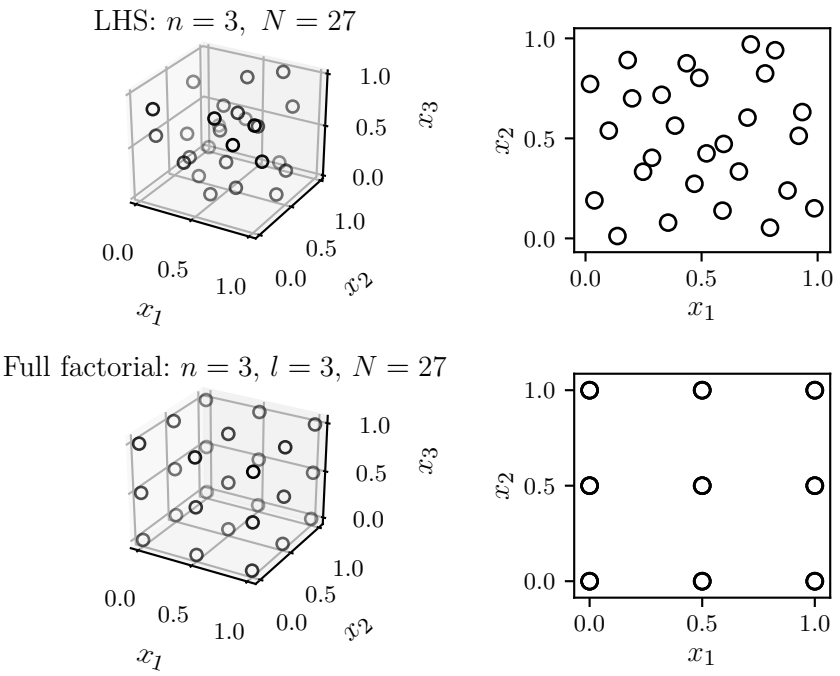


Figure 3.9: Comparison of the reduced data set of a LHS and full factorial design plan if one input parameter (x_3) is removed.

Important notes:

- Systematic design plans provide a space-filling design plan with an exponentially increasing number of samples dependent on input parameter count n and levels l.
- Therefore, they are very expensive if n is high or a high order polynomial model should be used, and they do not allow any kind of correlation or distribution type to be defined for the input parameters.
- If input parameters are removed from a systematic design plan, for example due to a sensitivity analysis, points collapse one into another which reduces the effective number of samples. This is not the case for the MCS / LHS and an important property for machine learning applications. Due to properties like automatic relevance detection (see subsection 2.8.7) of a $\mathcal{GP}$ this might also happen during the training.
- Stochastic design plans like the MCS / LHS are independent of the number of input parameters n.
- For the MCS / LHS, the number of levels for each input parameter is always equal N. This allows much more flexible surrogate models to be trained without the need for a huge number of samples.
- The MCS / LHS can handle different distribution types.

- Latin hypercube sampling is an extension of the Monte Carlo method which is in most cases recommended in preference to MCS except when the number of samples N is unknown a priori. This drawback can also be solved by using a sequential LHS presented in section 3.4.
- LHS can be optimized in order to increase e.g. space-filling properties or decrease unwanted correlations which are typical drawbacks of these kind of methods.

3.3 OPTIMIZATION OF LATIN HYPERCUBE SAMPLING

There are many published works about how to optimize a LHS, for example in Park [1994], Liefvendahl and Stocki [2006], Xiong et al. [2009] and Janouchová and Kučerová [2013] comparisons are made for different objective functions. Typically, the objectives can be categorized into two different classes. Firstly, the distance-based criteria which measure the space-filling properties of the design plan. These objective functions try to cover the available design space as good as possible so that every design point provides a maximal gain of information. Secondly, the correlation-based criteria, sometimes also called orthogonality criteria which try to avoid unwanted correlations as explained in the previous section. Apart of these two categories, there are also criteria which directly involve the surrogate model prediction quality in the design plan generation but mostly to create test design plans as proposed in Sacks et al. [1989] which introduces the integrated mean squared error criterion. The prediction error on test points is used to determine the optimal position of new sampling points for a test DoE or an adaption of the existing design plan. The topic of how to adapt existing design plans in order to increase, for example, the prognosis quality of a surrogate model will be covered in section 3.4. In the following, objective functions not based on surrogate predictions will be considered. These are suitable to create a first start DoE without the need for a surrogate model and are also sometimes called one-shot design plans (Keane and Nair [2005]).

3.3.1 *Distance-based Criteria*

Audze-Eglais objective function, proposed by Audze [1977], uses a potential energy approach. The potential energy E^{AE} is minimized if the sampling points are distributed as uniformly as possible. The relationship between the potential energy and the Euclidean distance between the points is described by

$$E^{AE} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{1}{d_{ij}^2} \tag{3.1}$$

with d_{ij} being the Euclidean distance between points i and j:

$$d_{ij} = \left[\sum_{k=1}^{n} (x_{ik} - x_{jk})^2 \right]^{\frac{1}{2}} \tag{3.2}$$

The number of pairwise distances d_{ij} in X is given as $s = \binom{N}{2}$.

Maximin distance criterion and E^{Φ_p} criterion are two of the most commonly used space-filling objective functions. According to Johnson et al. [1990] is X called a Euclidean maximin design if it maximizes the minimum Euclidean inter-site distance or for optimization reasons, minimizes the negative minimum distance

$$E^{EMM} = -min(..., d_{ij}, ...),\ i = 1, ..., N-1,\ j = i+1, ..., N \tag{3.3}$$

Morris and Mitchell [1995] proposed an intuitively appealing extension of the maximin distance criterion. For a given X, by sorting all the inter-sited distances d_{ij} $(i = 1, ..., N-1,\ j = i+1,\ ...,\ N)$, a distance list $(d_1,\ d_2,\ ...,\ d_s)$ and an index list $(J_1,\ J_2,\ ...,\ J_s)$ can be obtained, where the entries of the distance list are distinct distance values with $d_1 < d_2 < ... < d_s$ and J_i is the number of pairs of sites in X separated by d_i. A design plan X is called E^{Φ_p} optimal if it minimizes

$$E^{\Phi_p} = \left[\sum_{i=1}^{s} J_i d_i^{-p}\right]^{1/p} \tag{3.4}$$

where p is a positive integer. With a very large p, the minimum distance d_1 will dominate all subsequent d_i. In that case, the E^{Φ_p} criterion is equivalent to the E^{EMM} criterion (Jin et al. [2003]). This criterion provides very large values and is numerically expensive to optimize which is a drawback compared to the E^{EMM}.

Centered CL_2 discrepancy is a measure of the difference between the empirical cumulative distribution function of X and the uniform cumulative distribution function. In other words, it measures the non-uniformity of X. One advantage of this criterion is that it not only optimizes the uniformity in the overall design space X but also in all its lower projections (see Fang et al. [2002]). This makes it a favorable criterion if unimportant parameters are included for the surrogate training. Hickernell [1998] derived an analytical expression, in which the designs are normalized in each dimension to the interval $[0, 1]$ and then the value of CL_2 can be computed according to

$$\begin{aligned} E^{CL_2} = &\left(\frac{13}{12}\right)^n - \frac{2}{N}\sum_{i=1}^{N}\prod_{k=1}^{n}\left(1 + 0.5\,|x_{ik} - 0.5| - 0.5\,|x_{ik} - 0.5|^2\right) \\ &+ \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\prod_{k=1}^{n}\left[1 + 0.5\,|x_{ik} - 0.5| + 0.5\,|x_{jk} - 0.5| - 0.5\,|x_{ik} - x_{jk}|\right] \end{aligned} \tag{3.5}$$

x_{ik} and x_{jk} are the k-th coordinates of the i-th and j-th points, respectively. To achieve the best space-filling property of the DoE, the value of CL_2 should be minimized. One drawback of this measure is its computational complexity which can make it slow to calculate for large values of N.

Maximum entropy is a measure which was introduced by Lindley [1956], based upon Shannon's entropy (Shannon [1948]), of the amount of information provided by an experiment. This Bayesian measure uses the expected reduction in entropy as a design criterion. This criterion has been used in Box and Hill [1967] and Borth [1975] for model discrimination. Shewry and Wynn [1987] showed that if the design space is discrete (e.g. a lattice in $[0, 1]^n$), then minimizing the expected posterior entropy is equivalent to maximizing the prior entropy. From the Bayesian point of view, the lower the posterior entropy, the smaller is the uncertainty in the prediction of the response at unobserved sites (Shewry and Wynn

[1987]). Minimizing the posterior entropy is equivalent to finding a set of design points on which the least knowledge is given or, in other words, maximizing the design plan entropy results in the maximization of information a design point can provide. It has been further shown in Koehler and Owen [1996] that the entropy criterion, in the case of a Gaussian process, is equivalent to minimize the following equation:

$$E^{ME} = -\ln|K| \tag{3.6}$$

where K is the covariance matrix of the design matrix X. Since this covariance matrix is only dependent on X and not on any observations $\boldsymbol{y}$ it can even be used for an unconditioned Gaussian process model or also for the $\mathcal{DGCN}$ model. Looking back at the log-likelihood of a $\mathcal{GP}$ model, eq. 2.74 reveals that E^{ME} corresponds to the mid-term $-0.5\ln\left(|K+\sigma_n^2 I|\right)$. So minimizing E^{ME} is equivalent to minimizing the unconditioned log-likelihood of a $\mathcal{GP}$, $\mathcal{DGCN}$ model which is the opposite of what is done for the training of the model (minimizing the *negative* log-likelihood). For all following tests with this criterion, the combination of the summarized covariance matrix of the $\mathcal{DGCN}$ model, explained in section 2.10, will be used if nothing else is mentioned.

In fig. 3.10 a comparison is given, for all distance-based criteria mentioned above for a two dimensional design plan with $N = 20$. All criteria provide a visually good space-filling design plan. It also looks like the E^{ME} and E^{CL_2} prefer to set the points closer to the parameter bounds compared to the other criteria. Furthermore, in fig. 3.11 to 3.15 pairwise scatter plots are illustrated together with the linear correlation coefficients and the corresponding histograms of each variable for a design plan with $n = 5$ and $N = 20$. This should demonstrate, how the lower projections of the design plan look and how these criteria might introduce unwanted correlations between the input parameters. The low sample size is chosen because it is more difficult to reduce unwanted correlations for a small set of design points. The optimization is done as described in section 3.3.3. As the figures demonstrate, the E^{EMM} and the E^{Φ_p} have high correlation values introduced to the design plan. The other criteria seem to behave equally well. The uniformity is for all criteria quite good which is difficult to judge with only $N = 20$. It seems also that all criteria have a similar good space-filling subspace coverage, with the E^{CL_2} criterion looking to have the lowest number of clusters.

3.3.2 *Correlation-based Criteria*

There are two well-known approaches to evaluate the orthogonality or correlation of a *DoE*. The most popular one is based on correlations among the input parameters specified by a correlation coefficient, the other one is the condition number of $X^T X$.

The **Condition number** is commonly used in numerical linear algebra to examine the sensitivities of a linear system (Cioppa and Lucas [2007]) but it can also be used to maximize the orthogonality of a *DoE*. Therefore, the condition number of $X^T X$ is used:

$$E^{CN} = cond(X^T X) = \frac{\lambda_1}{\lambda_N} \tag{3.7}$$

where λ_1 and λ_N are the largest and smallest Eigenvalues of $X^T X$, respectively, therefore the E^{CN} is greater or equal to 1. Values closer to 1 correspond to a more orthogonal *DoE*,

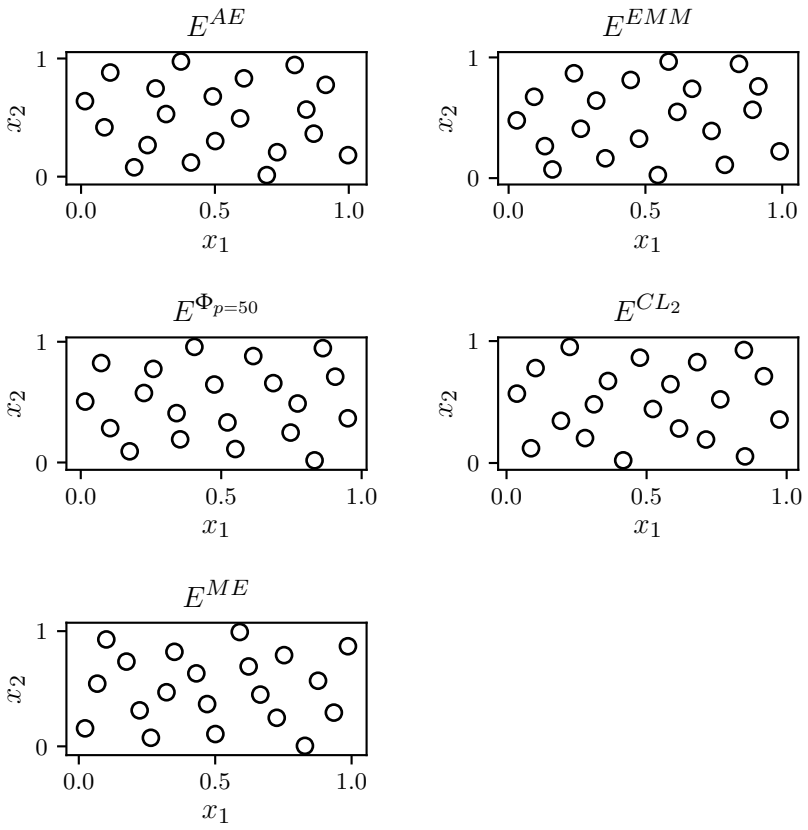


Figure 3.10: Comparison of distance-based objective for $N = 20$ and $n = 2$.

thus the condition number should be minimized. It should be noted that the columns of the design matrix X:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nn} \end{bmatrix}$$

must be scaled to the range $[-1, 1]$.

Pearson correlation coefficient (PCC) (Pearson [1895]) is the standard measure of linear monotonic dependence between two variables. So, for two variables x_i and x_j, the PCC is defined as

$$c_{ij}(x_i, x_j) = \frac{Cov(x_i, x_j)}{\sigma_{x_i}\sigma_{x_j}} = \frac{\sum_{k=1}^{N}(x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)}{\sqrt{\sum_{k=1}^{N}(x_{ki} - \bar{x}_i)^2 \sum_{k=1}^{N}(x_{kj} - \bar{x}_j)^2}} \tag{3.8}$$

with

$$\bar{x}_i = \frac{1}{N}\sum_{k=1}^{N} x_{ki} \text{ and } \bar{x}_j = \frac{1}{N}\sum_{k=1}^{N} x_{kj} \tag{3.9}$$

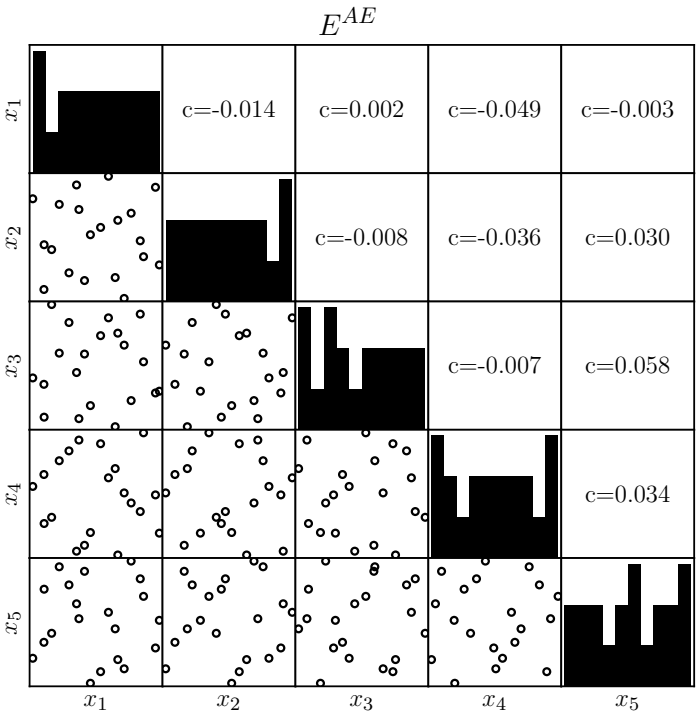


Figure 3.11: Scatter matrix plot and pairwise linear correlation coefficients for the E^{AE} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given.

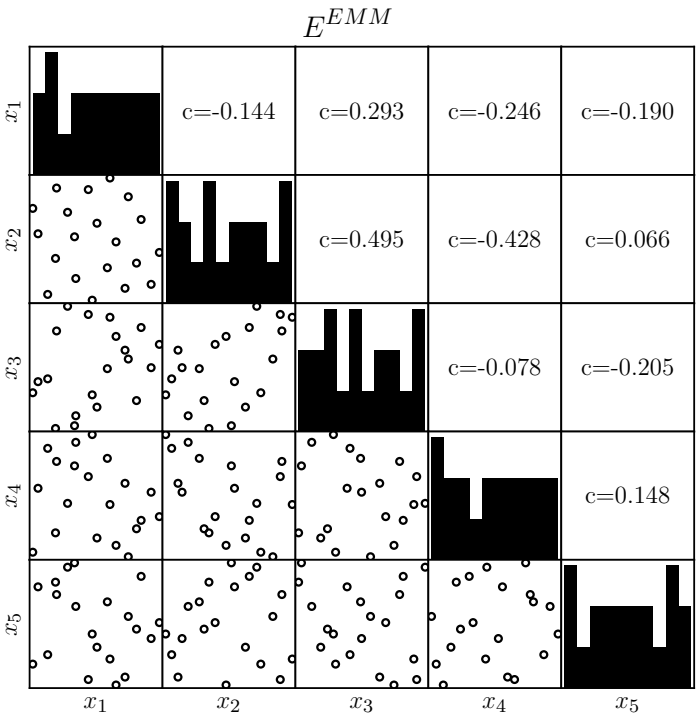


Figure 3.12: Scatter matrix plot and pairwise linear correlation coefficients for the E^{EMM} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given.

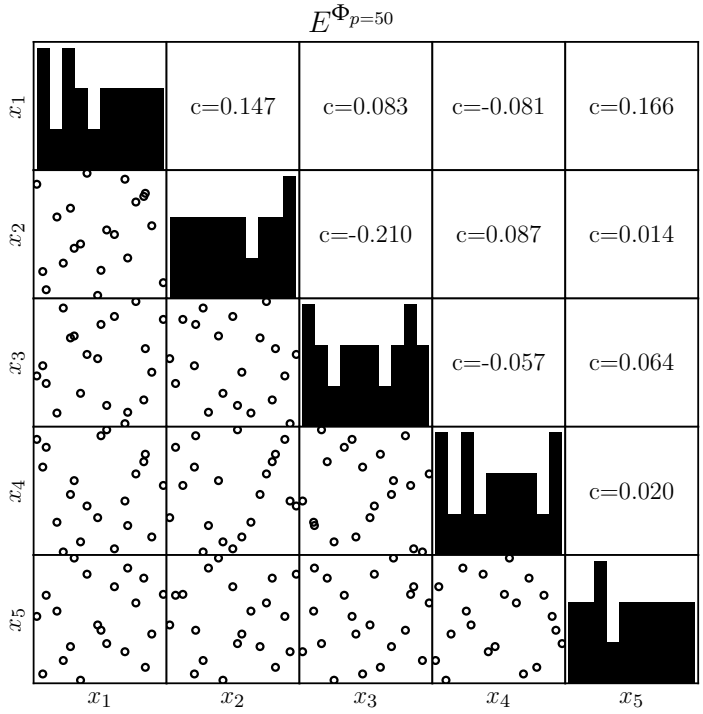


Figure 3.13: Scatter matrix plot and pairwise linear correlation coefficients for the E^{Φ_p} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given.

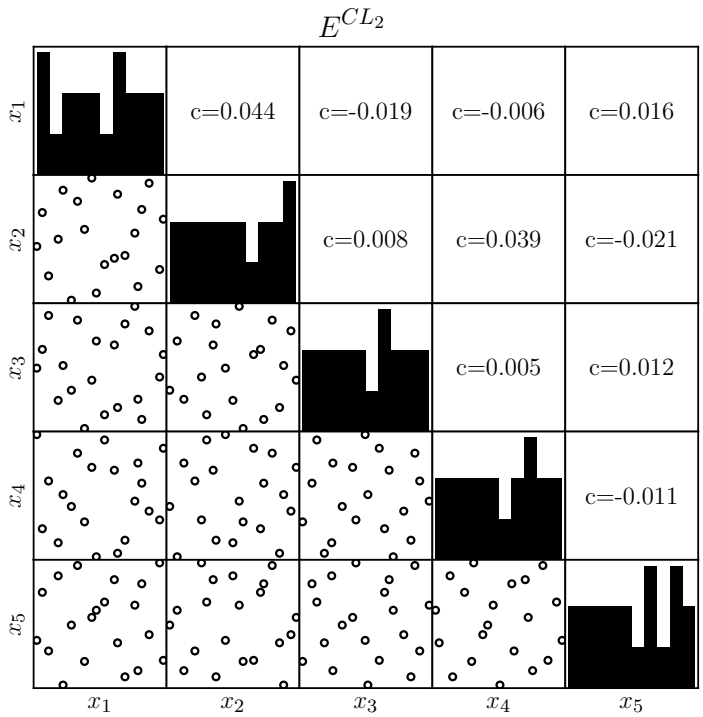


Figure 3.14: Scatter matrix plot and pairwise linear correlation coefficients for the E^{CL_2} criterion $N = 20$ and $n = 5$. On the diagonal the histogram is given.

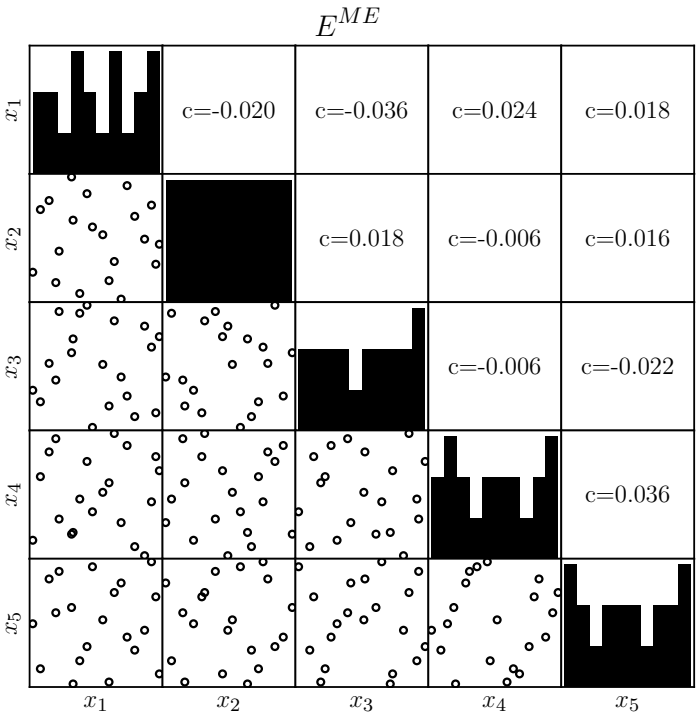


Figure 3.15: Scatter matrix plot and pairwise linear correlation coefficients for the E^{ME} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

The PCC takes a value between -1 and 1 and positive values indicate that the value of x_i tends to increase together with increasing value of x_j, while negative values indicate decreasing value of x_i with increasing value of x_j. Zero value stands for no linear relationship between x_i and x_j. In order to obtain an orthogonal DoE in a multi-dimensional design space, the PCC needs to be evaluated for each pair of columns (all input parameter pairs) in the design matrix X. As a result, a $n \times n$ symmetric correlation matrix with the entries of eq. 3.8 is obtained via

$$C = \begin{bmatrix} 1 & c_{12} & \dots & c_{1n} \\ c_{21} & 1 & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n1} & c_{n2} & \dots & 1 \end{bmatrix}$$

For a perfectly orthogonal DoE, the correlation matrix C of the DoE is equal to the identity matrix (all off-diagonal elements are zero). To achieve an orthogonal DoE, it is possible, for instance, to minimize the sum of squares of the elements of the upper or lower triangular matrix of C:

$$E^{PCC} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}^2 \tag{3.10}$$

or in the normalized version to be in the interval $[0, 1]$ proposed by Conover and Iman [1981], Tang [1993] and Owen [1994] (also used in this work):

$$E^{PCC} = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ij}^2}{n(n-1)/2} \tag{3.11}$$

Maximal absolute Pearson correlation coefficient ($MAPCC$) can be used similar to the E^{EMM} by minimizing the maximal absolute Pearson correlation coefficient $|c_{ij}|$ of the lower or upper triangular matrix of C as demonstrated in Cioppa and Lucas [2007], instead of the sum of all correlations coefficients:

$$E^{MAPCC} = max(..., c_{ij}, ...),\ i = 1, ..., n-1,\ j = i+1, ..., n \tag{3.12}$$

Similarly, other correlation coefficients can also be used by replacing the Pearson correlation coefficient with, for example, the Spearman's rank correlation coefficient (Spearman [1904]) or the Kendall tau rank correlation coefficient (Kendall [1938]) for non-linear and monotonic relationships. For a more detailed overview of correlation coefficients see chapter 4.

Distance correlation coefficient (DCC) is another correlation coefficient (Székely et al. [2007]) which is able to capture non-linear and non-monotonic correlations between two variables x_i and x_j. In literature, it is rarely mentioned. The distance correlation is never zero if there is statistical dependency between two variables x_i and x_j. This might be advantageous if used for optimizing a LHS. To calculate this coefficient for two variables x_i and x_j, first the $N \times N$ distance matrices of all Euclidean pairwise distances D^{x_i}, D^{x_j} are needed for the sample data vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Each of the distance matrices must be centered such that the row and column means are all zero. Then the sample distance covariance (scalar) can be estimated via

$$dCov^2(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{N^2}\sum_{z=1}^{N}\sum_{k=1}^{N} D_{zk}^{x_i} D_{zk}^{x_j} \tag{3.13}$$

with the sample distance variance

$$dVar(\boldsymbol{x}_i) = dCov(\boldsymbol{x}_i, \boldsymbol{x}_i) = \sqrt{\frac{1}{N^2}\sum_{z=1}^{N}\sum_{k=1}^{N} \left(D_{zk}^{x_i}\right)^2} \tag{3.14}$$

and finally the sample distance correlation

$$dc_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{dCov(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{dVar(\boldsymbol{x}_i) dVar(\boldsymbol{x}_j)}} \tag{3.15}$$

The distance correlation is always positive because distances are always positive and ranges between $[0, 1]$. One drawback of this correlation coefficient is its computational complexity of $\mathcal{O}(N^2)$. Also, the pairwise distance matrices can become very large (similar to Gaussian processes) for a large value of N. The corresponding optimization criterion is

$$E^{DCC} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} dc_{ij} \tag{3.16}$$

or also only considering the maximum absolute distance correlation value:

$$E^{MADCC} = max(..., dc_{ij}, ...),\ i = 1, ..., n-1,\ j = i+1, ..., n \tag{3.17}$$

Correlation difference (CD). Instead of assuring that the design plan is free of correlations, also a specific target correlation matrix (e.g. based on the Pearson correlation coefficient) could be set. For this case, the goal is to minimize the difference between the target correlation matrix and the current design correlation matrix with

$$E^{CD} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (c_{ij}^{Tar} - c_{ij}^{X}) \tag{3.18}$$

where c_{ij}^{Tar} denotes the target correlation coefficient and c_{ij}^{X} the current one of the design matrix. This method can be used with all kinds of correlation coefficients.

Another approach to introduce a given correlation to the design plan was presented by Iman and Conover [1982]. This method reorders the design plan matrix X to have the same rank correlation as a reference distribution with desired linear correlation. This method does not need any kind of optimization and can analytically be solved but it cannot be combined with a space-filling objective which makes it quite unattractive for surrogate model creation. A step by step explanation of this method is given in the appendix A and another very good overview can be found in Mildenhall [2005].

Similar to the distance-based criteria, fig. 3.16 to fig. 3.19 show pairwise scatter plots along with the linear correlation coefficients and the corresponding histograms of each variable for a design plan with $n = 5$ and $N = 20$ for most of the presented correlation-based criteria. Furthermore, fig. 3.20 illustrates an example of introducing the following linear correlation matrix to a LHS:

$$C = \begin{bmatrix} 1 & 0.2 & 0 & 0 & 0 \\ 0.2 & 1 & 0 & 0.5 & 0.7 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0.5 & 0 & 1 & 0 \\ 0 & 0.7 & 0 & 0 & 1 \end{bmatrix}$$

All the correlation-based criteria seem to be able to reduce the linear correlation by $1E-3$ to $1E-5$. Quite interesting is the E^{MADCC} criterion which does not seem to minimize the linear correlation that low compared to the other metrics, but the lower two-dimensional projections look very good regarding the space filling property. It looks even better than some of the distance-based criteria. Maybe is this criterion usable for both kinds of optimizations. This should be investigated further in subsection 3.3.4. The introduction of a given correlation matrix via the E^{CD} criterion also worked perfectly for this example.

3.3.3 *Optimization Algorithm*

In most cases permutation-based algorithms are used for the optimization procedure of design plans. For example, a design plan X_{best} and two design points $\boldsymbol{x}_1 = [x_1, x_2, ...x_n]$, $\boldsymbol{x}_4 = [x_1, x_2, ...x_n]$ are picked. Within these design points a specific parameter value is chosen (e.g. x_1) and swapped so $\boldsymbol{x}_1$ contains now the x_1 value of $\boldsymbol{x}_4$ and vice versa (see fig. 3.21). Thus, a new candidate design plan X_{cand} is created which can be compared to X_{best} regarding the objectives. If X_{cand} is better then X_{best}, X_{best} becomes X_{cand} else it will be discarded. This procedure is repeated for several iterations. This is a very crude approach and strictly only accepting better candidates which may lead to the optimization

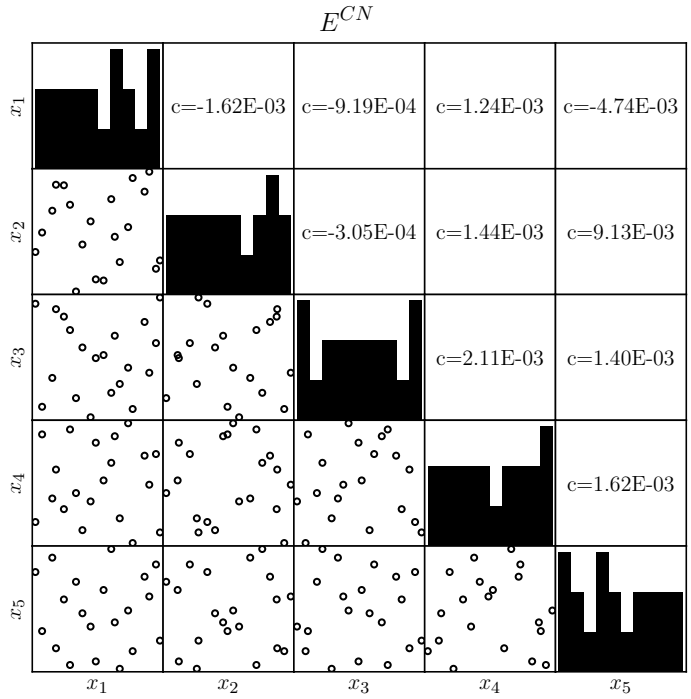


Figure 3.16: Scatter matrix plot and pairwise linear correlation coefficients for the E^{CN} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

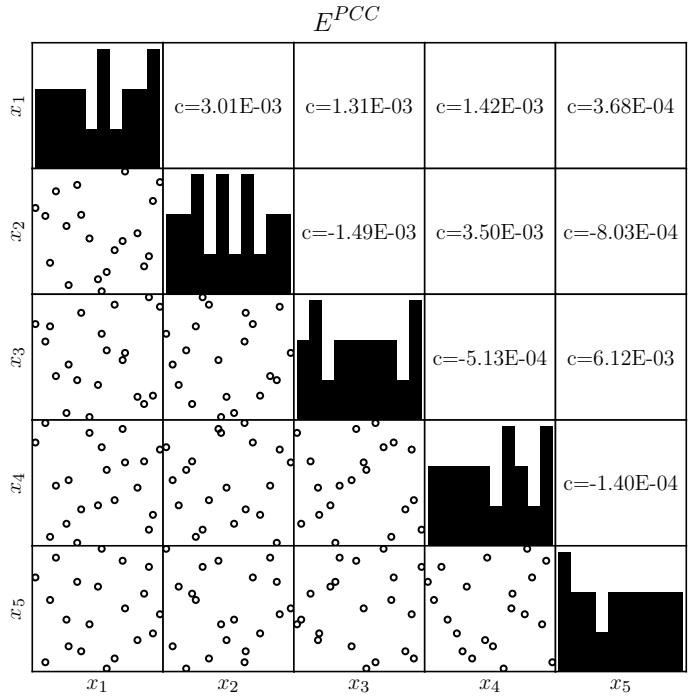


Figure 3.17: Scatter matrix plot and pairwise linear correlation coefficients for the E^{PCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

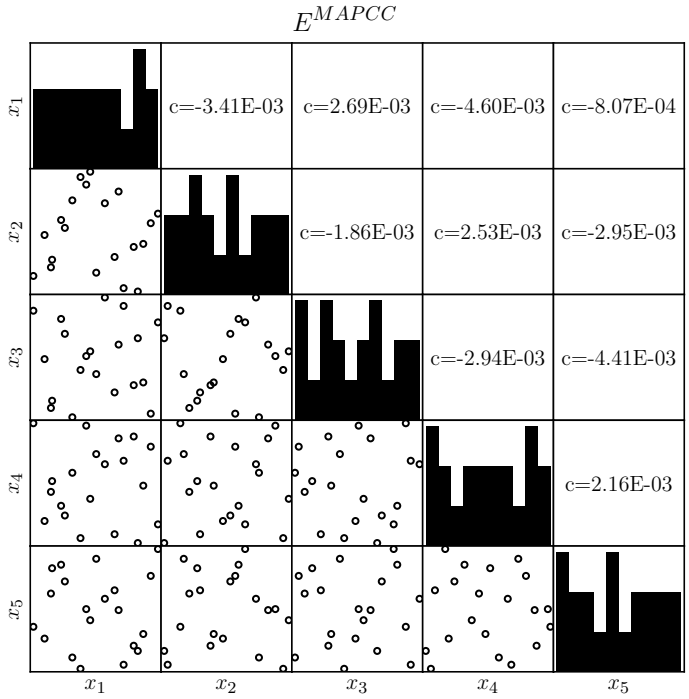


Figure 3.18: Scatter matrix plot and pairwise linear correlation coefficients for the E^{MAPCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

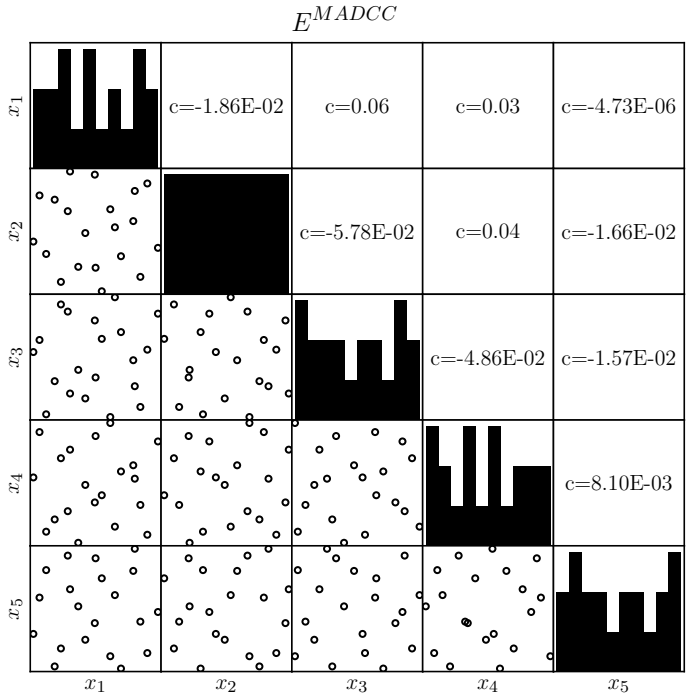


Figure 3.19: Scatter matrix plot and pairwise linear correlation coefficients for the E^{MADCC} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

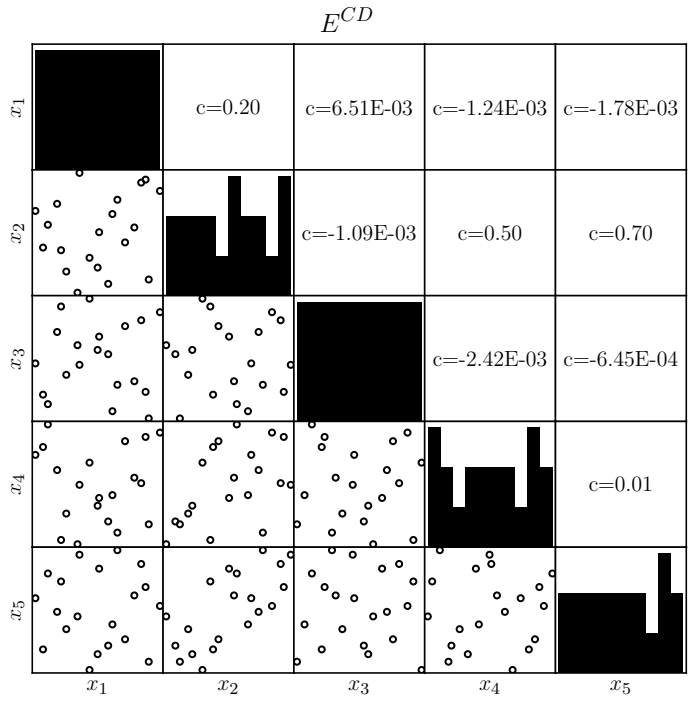


Figure 3.20: Scatter matrix plot and pairwise linear correlation coefficients for the E^{CD} criterion with $N = 20$ and $n = 5$. On the diagonal the histogram is given.

algorithm being trapped in local minima and never reaching the desired global minima. More sophisticated algorithms such as simulated annealing (Kirkpatrick et al. [1983]), which is a version of the Metropolis algorithm (Metropolis et al. [1953]), belonging to the group of Markov chain Monte Carlo methods ($MCMC$) (Gilks et al. [1995]), allow candidates that are less optimal than the currently best design with a certain probability to avoid the effect of being trapped in local minima. The probability can be calculated via

$$p(X_{best}, X_{cand}, T) = \exp\left(-\frac{(\Psi(X_{cand}) - \Psi(X_{best}))}{T}\right) \tag{3.19}$$

with $\Psi()$ being the objective function and T the "temperature", a term which comes from the original physical motivation for annealing as an optimization process. At the start of the iteration process, a high value for the temperature T is chosen leading to a higher probability that worse candidates X_{cand} are accepted in favor of X_{best}. This allows the search algorithm to break away from local minima. The temperature T is reduced with increasing number of iterations effectively reducing that probability assuming that the best design is now close to the global minimum and allowing for the algorithm to find that minimum. Another brute force approach would be to create many different random designs plans and chose the best one regarding the optimization objectives. This works quite well, since the numerical costs to create such a plan are very low. In this work, a combination of both will be used so that a higher number of sampling plans are generated and the best performing one will be used as a start design plan for a simulated annealing optimization. A good overview of further optimization techniques can be found in Jin et al. [2003] and Liefvendahl and Stocki [2006]. In this work, the version of the simulated

annealing algorithm illustrated in alg. 1 is used with I_{opt} being the number of optimization iterations. In this work, it was observed that this number typically is between 1000 and 5000. I_{max} is the number of design perturbations the algorithm will try before going on to the next temperature, provided no new best design is found. I_{bf} is the number of brute force iterations to find a good starting design plan. T is the temperature parameter. $\alpha, \in [0, 1]$ is the factor by which the temperature is modified, i.e. a value of 0.95 results in a 5% reduction in temperature. In alg. 1 $lhs_{bf}()$ denotes the function which creates I_{bf} design plans and returns, regarding $\Psi()$, the best design plan. $perturb()$ denotes the function which perturbs two column elements in X_{cur}. $\mathcal{U}(0, 1)$ is a uniformly distributed random variable within 0 and 1. The values for I_{max}, α and T are chosen based on the suggestions of Morris and Mitchell [1995]. They suggest for I_{max} that it should be larger for larger problems, i.e. those for which the number of possible perturbations, $\binom{N}{2} \times n$, is relatively large. They used 10 times this number for the value of I_{max}. The choice of T should be dependent on possible values of objective Ψ so that at the beginning the temperature is high enough that the algorithm performs a nearly unbiased random walk among candidate designs, but not so high that it wastes too much time doing this.

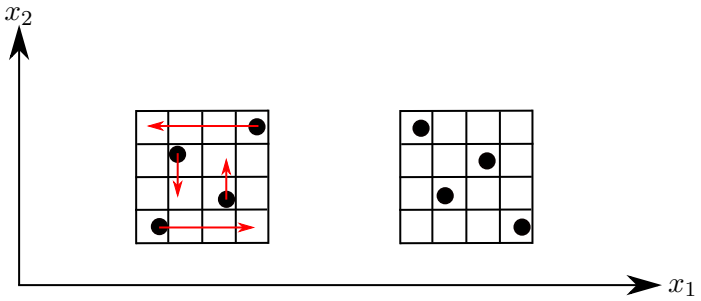


Figure 3.21: Illustration of the design points permutation.

Typically, not only one optimization criterion is used as objective but two or more to cover the space-filling and correlation properties of the design plan. A common approach for this kind of multi-objective optimization is to optimize a weighted average of all the objective functions, as proposed in Joseph and Hung [2008]. Following the suggestions of Joseph and Hung [2008], the objective function was chosen in this work as

$$\Psi = wE^{PCC} + (1 - w)E^{\Phi_p},\ w \in [0, 1] \tag{3.20}$$

which especially makes sense with distance-based objectives like E^{Φ_p} or E^{EMM}, as it was demonstrated that they can introduce high unwanted correlations. The combination with a correlation-based criterion like E^{PCC} allows to reduce these unwanted correlations. Tab. 3.1 shows an overview of selected publications and their optimization algorithms / criteria used to optimize a LHS taken from (Viana et al. [2010]).

3.3.4 *Influence on Surrogate Model Training*

In the following, an investigation of how the presented objectives can influence the prediction quality of a surrogate model will be outlined. For this test, a $\mathcal{DGCN}$ model will be trained on different sample sizes $N = 40, 80, 120$ and the $\hat{R}^2$ together with the $RMSE$ for a test set of $N = 1,000$ uniform distributed samples will be estimated. The $\hat{R}^2$ is esti-

Algorithm 1: Simulated annealing algorithm.

Data: $N, I_{opt}, I_{max}, I_{bf}, T, \alpha$.
Result: X_{best} the optimized design plan.

begin
  $X_{cur} \leftarrow lhs_{bf}(N, I_{bf})$
  $X_{best} \leftarrow X_{cur}$
  $\Psi_{cur} \leftarrow \Psi(X_{cur})$
  $\Psi_{best} \leftarrow \Psi(X_{best})$
  $I \leftarrow 0$
  for $i \leftarrow 0$ **to** I_{opt} **do**
    $X_{cand} \leftarrow perturb(X_{cur})$
    $\Psi_{cand} \leftarrow \Psi(X_{cand})$
    if $\Psi_{cand} < \Psi_{cur}$ ***or*** $\mathcal{U}(0,1) < \exp\left(-\frac{(\Psi_{cand}-\Psi_{best})}{T}\right)$ **then**
      $X_{cur} \leftarrow X_{cand}$
      $\Psi_{cur} \leftarrow \Psi_{cand}$
      $I \leftarrow 0$
      $T \leftarrow T\alpha$
      if $\Psi_{cur} < \Psi_{best}$ **then**
        $X_{best} \leftarrow X_{cur}$
        $\Psi_{best} \leftarrow \Psi_{cur}$
    $I \leftarrow I + 1$
    if $I = I_{max}$ **then**
      $T \leftarrow T\alpha$
    $i \leftarrow i + 1$

mated via 10-fold stratified cross-validation. Additionally, the maximum linear correlation E^{MAPCC} will be measured. The test function is, once again, the Himmelblau test function (eq. 2.8). For the first test, the focus will be on the distance-based criteria and the E^{MADCC} criterion which also seems to be useful for this kind of optimization. Furthermore, a not optimized random LHS is also analyzed serving as the baseline. All design plans are sampled in one shot. The procedure is repeated 25 times and the mean and standard deviation are calculated for the metrics mentioned. Tab. 3.2 summarizes the results. The bold numbers indicate the best results considering the mean and the standard deviation. Regarding the $\hat{R}^2$ and the $RMSE$ criterion, the E^{ME} together with the E^{CL_2} seems to be superior to the other metrics. For the E^{MAPCC} metric the E^{CL_2} is the best criterion. All other algorithms also have fairly low correlation values. The E^{MADCC} seem to work quite well, even though it is a correlation criterion. Overall, no clear best working criteria can be identified for the metrics measured. The difference regarding the $RMSE$ metric to the not optimized LHS reduces the more training samples are available which makes sense, since at a specific number of samples, dependent on the number of variables, the design space is space-filling covered. At $N = 40$ the not optimized LHS overestimates the $\hat{R}^2$ metric drastically which might also be an argument to use an optimized LHS if the sample size is low compared to the amount of input variables n.

Publication	Algorithm	Criteria
Audze [1977]	Coordinates Exchange Algorithm	E^{AE}
Park [1994]	A 2-stage (exchange- and Newton-type) algorithm	Integrated mean squared error and E^{ME}
Morris and Mitchell [1995]	Simulated annealing	E^{Φ_p}
Kenny et al. [2000]	Column-wise-pairwise	E^{Φ_p} and E^{ME}
Fang et al. [2002]	Threshold accepting algorithm	E^{CL_2}
Bates et al. [2004]	Genetic algorithm	E^{AE}
Jin et al. [2003]	Enhanced stochastic evolutionary algorithm	E^{Φ_p}, E^{ME} and E^{CL_2}
Liefvendahl and Stocki [2006]	Column-wise-pairwise and genetic algorithms	E^{EMM} and E^{AE}
Van Dam et al. [2007]	Branch-and-bound algorithm	1-norm and infinite norm distances
Grosso et al. [2009]	Iterated local search and simulated annealing algorithms	E^{Φ_p}

Table 3.1: Overview of published works about LHS optimization and their chosen optimization algorithm / criteria.

For the next test, combinations of distance and correlation-based criteria will be tested. Therefore, the same test is repeated with the following combinations:

$$E^{MAPCC+EMM} = E^{MAPCC} + E^{EMM} \tag{3.21}$$

$$E^{CL2+ME} = E^{CL_2} + E^{ME} \tag{3.22}$$

$$E^{MAPCC+ME} = E^{MAPCC} + E^{ME} \tag{3.23}$$

$$E^{MADCC+ME} = E^{MADCC} + E^{ME} \tag{3.24}$$

$$E^{MADCC+EMM} = E^{MADCC} + E^{EMM} \tag{3.25}$$

N	40	80	120
		$\hat{R}^2$	
Not optimized	**0.83±1.94E-01**	0.99±1.20E-02	1.00±4.45E-03
E^{AE}	0.67±3.18E-01	0.99±1.09E-02	1.00±1.87E-03
E^{EMM}	0.71±2.54E-01	0.99±2.54E-01	1.00±2.54E-03
E^{Φ_p}	0.61±3.21E-01	0.92±2.70E-01	0.92±2.70E-01
E^{CL_2}	0.70±2.95E-01	**0.99±6.63E-03**	1.00±9.75E-03
E^{ME}	0.58±3.28E-01	0.99±1.21E-02	**1.00±5.91E-04**
E^{MADCC}	0.67±3.22E-01	0.99±2.11E-02	1.00±1.61E-03
		$RMSE$	
Not optimized	33.87±16.67	6.39±3.17	2.87±3.99
E^{AE}	31.03±14.93	4.97±3.45	1.72±1.26
E^{EMM}	31.53±15.46	5.11±2.09	2.46±2.85
E^{Φ_p}	29.23±13.02	4.04±1.81	2.61±4.23
E^{CL_2}	**27.17±13.75**	**3.91±1.79**	1.64±2.72
E^{ME}	30.11±12.01	3.92±2.70	**1.14±0.51**
E^{MADCC}	32.93±14.28	5.69±4.75	2.37±1.49
		E^{MAPCC}	
Not optimized	7.01E-02±5.96E-02	4.05E-02±3.63E-02	3.87E-02±3.07E-02
E^{AE}	4.26E-02±3.26E-02	3.06E-02±2.25E-02	2.39E-02±2.05E-02
E^{EMM}	8.58E-02±6.96$E-02$	7.29E-02±6.22E-02	6.51E-02±4.77E-02
E^{Φ_p}	6.95E-02±5.00E-02	4.93E-02±3.31E-02	3.79E-02±3.46E-02
E^{CL_2}	**4.68E-03±4.12E-03**	**2.40E-03±1.77E-03**	**2.58E-03±1.91E-03**
E^{ME}	3.96E-02±3.36E-02	3.29E-02±2.22E-02	1.87E-02±1.76E-02
E^{MADCC}	6.74E-03±5.68E-03	3.11E-03±2.37E-03	2.63E-03±2.65E-03

Table 3.2: Comparison of distance-based LHS objectives and the E^{MADCC} criterion for surrogate model training based on different sample sizes N. The bold numbers indicate the best results considering the mean and the standard deviation.

which seems to be the most promising combination based on the previous test. The E^{Φ_p} is left out, since it has a large calculation time. For most of these combinations, a correct scaling between both objectives is difficult. The E^{Φ_p} has very high values as well as the E^{ME} for large values of N. On the other hand, the E^{MAPCC} and E^{MADCC} have values within $[0, 1]$. The E^{EMM} can also be scaled to the range $[0, 1]$ by dividing it with the maximum possible pairwise distance d_{max} which is based on the lower and upper bounds x_i^l, x_i^u for the n parameters:

$$d_{max} = \sqrt{\sum_i^n \left(x_i^u - x_i^l\right)^2} \tag{3.26}$$

This might lead to no real improvement for the correlation-based criteria in combination with non-scaled distance-based criteria.

The results for the combined objectives are given in tab. 3.3. Once again, no clear winner can be identified but the combination of the best working criteria from the previous test, the E^{ME} criterion together with the E^{CL_2}, also bring the best results here for the $RMSE$ metric. It is nearly equal to the result the E^{ME} got as a single criterion. The E^{MAPCC} has significantly improved for the E^{EMM} criterion in combination with the E^{MAPCC} as an objective. The combination E^{L2+ME} has slightly increased the E^{MAPCC} compared to the values of the single E^{CL_2} criterion.

N	40	80	120
		$\hat{R}^2$	
$E^{MAPCC+EMM}$	**0.77±1.93E-01**	0.99±1.15E-02	**1.00±1.42E-03**
E^{L2+ME}	0.66±2.95E-01	0.99±7.13E-03	1.00±1.46E-03
$E^{MAPCC+ME}$	0.67±2.28E-01	**0.99±4.17E-03**	1.00±2.60E-03
$E^{MADCC+ME}$	0.73±2.88E-01	0.99±6.10E-03	1.00±1.74E-03
$E^{MADCC+EMM}$	0.61±2.75E-01	0.99±3.20E-02	1.00±1.04E-02
		$RMSE$	
$E^{MAPCC+EMM}$	33.18±14.36	6.15±4.01	2.26±1.69
E^{L2+ME}	27.18±10.72	**3.64±0.92**	**1.13±0.69**
$E^{MAPCC+ME}$	30.88±9.93	3.92±1.25	2.08±2.16
$E^{MADCC+ME}$	**21.78±10.06**	3.83±2.05	1.53±0.92
$E^{MADCC+EMM}$	32.59±13.26	4.72±4.34	3.12±4.44
		E^{MAPCC}	
$E^{MAPCC+EMM}$	**8.58E-04±2.26E-03**	**2.86E-04±5.44E-04**	**3.77E-04±7.49E-04**
E^{L2+ME}	3.41E-02±2.59E-02	3.62E-02±3.13E-02	2.53E-02±1.75E-02
$E^{MAPCC+ME}$	1.48E-02±1.55E-02	2.06E-02±1.83E-02	1.64E-02±1.41E-02
$E^{MADCC+ME}$	2.83E-02±2.22E-02	3.31E-02±1.63E-02	2.31E-02±1.70E-02
$E^{MADCC+EMM}$	8.97E-03±7.72E-03	4.50E-03±3.41E-03	3.15E-03±2.31E-03

Table 3.3: Comparison of combined LHS objectives for surrogate model training based on different sample sizes N. The bold numbers indicate the best results considering the mean and the standard deviation.

Important notes:

- There are mainly two types of objectives to optimize a LHS. Firstly, the distance-based criteria which measure the space-filling properties of the design plan. Secondly, the correlation-based criteria, sometimes also called orthogonality criteria, which try to avoid unwanted correlations.
- Several different criteria for each of these objectives types have been investigated. Regarding the distance-based criteria, all of them are capable of being used for a space filling DoE. The same holds for the correlation-based criteria, where all of them can be used to reduce unwanted correlations.

- The main difference of the different objectives are their computational complexity which might be a limitation for cases with a high number of samples to be optimized.
- In most cases it is useful to combine one distance-based and one correlation criterion to a weighted objective function.
- The optimization of a LHS is mostly performed by using a permutation-based algorithm, like the simulated annealing one presented.
- Regarding the influence on the surrogate model prognosis quality, all tested criteria give quite similar results. The E^{L2} criterion gave slightly better results overall. It might also be more beneficial because of its property to also consider the optimization of the pairwise subspaces.
- An optimized LHS is especially beneficial over a non-optimized one if the number of samples is low compared to the number of variables. If it is possible to create a high number of samples, the advantage of the optimization vanishes.

3.4 ADAPTIVE SAMPLING & BAYESIAN OPTIMIZATION

Since the number of necessary sampling points is typically unknown in advance and limited by a specific computational budget, it might be better to start with a lower number of points and extend the design plan sequentially or adaptively. Also because sometimes a one-shot design plan is generated with more designs than really needed. This allows a convergence criterion e.g. the $\hat{R}^2$ value, to be used to guide the iteration process. If the convergence criterion does not change with additional sampling points, this might be a signal to stop the iteration process. The sequential extension of a sampling plan also allows, for example, additional points to be added only in a specific region of interest. This topic is also known as adaptive sampling (Lehmensiek et al. [2002]) or active learning (Sugiyama [2006]), if used in combination with machine learning.

An essential consideration in sequential or adaptive design plans is the trade-off between exploration and exploitation. Exploration is the act of exploring the design space in order to find key regions, such as areas with discontinuities, steep slopes, optima, stable regions that have not been identified before. The goal is similar to that of a one-shot experimental design, where exploration means filling up the design space as homogeneously as possible (Crombecq [2011]). Exploitation means that data points are located in regions which have been identified as potentially interesting; for example, in the area of an optimum. Exploitation involves using the previous results to guide the sampling process (Crombecq et al. [2011]). It is always necessary to start with a design exploration in order to find possible interesting regions for the design exploitation.

3.4.1 *Sequential Latin Hypercube Sampling*

One-shot LHS design plans, consisting of a large number of samples, have the advantage that the samples can be optimized regarding space fillingness and correlations but some-

times the number of samples is much more than really needed, for example to train an accurate surrogate model. Since simulations are expensive to evaluate (CPU / GPU effort), it makes more sense to start with a smaller of number samples and expand them so that instead of discarding the already evaluated samples, it is possible to reuse their results. In literature, the extension of a LHS design plan typically does not satisfy the restriction of still being a LHS (Van Dam et al. [2007], Xiong et al. [2009]). Nevertheless, one way to sequentially generate Latin hypercubes is the idea of nested Latin hypercubes. Rennen et al. [2010] used nested LHS where one set with a larger number of sampling points was generated such that a smaller set is an optimal subset and both are still a LHS. They stated that both sets can only be a LHS if the bigger set with the larger number of sample points is a multiple of N of the first set. Similar works can be found in Husslage et al. [2006] and Qian [2009]. A second option to adapt a Latin hypercube design plan sequentially is to give up the requirement that each sample is placed in exact intervals, resulting in so-called quasi-Latin hypercubes, as proposed in Van Dam et al. [2007] or Xiong et al. [2009]. Optimization is used in order to get an increased minimum distance of samples points if projected onto one of the parameter axes to still provide non-collapsing designs.

A novel approach is the reverse nested LHS, which starts with a design plan of size N_1 and the following design plans are always based on the double number of samples, so $N_2 = 2N_1$. Successively doubling the number of samples, i.e. dividing the LHS bins in half, satisfies the condition that the design plans are always Latin hypercube type samples. Dividing the bins into halves, allows a clear assignment of the existing points to a specific bin. The new set of points in the doubling approach are set in an optimized way in consideration of the existing points. The procedure is illustrated in fig. 3.22. As the figure suggests, there are multiple possible bins for a sampling point in each extension step. The optimization criteria presented in section 3.3 can be used to find the optimal bin for each point. In contrast to the nested LHS, it is not necessary to know the maximal number of samples in advance.

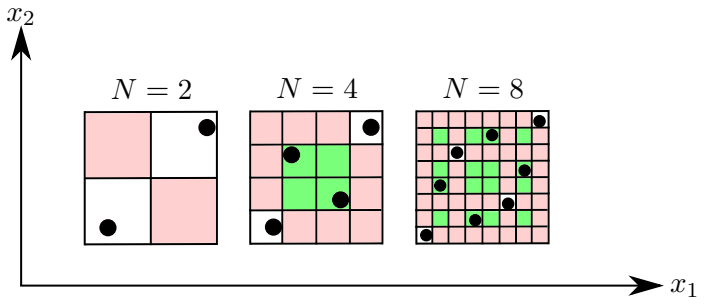


Figure 3.22: Illustration of the sequential LHS approach for $N_1 = 2$, $N_2 = 4$ and $N_3 = 8$. The green fields show the possible free bins for the sequential extension.

The goal of this strategy is to provide a globally accurate surrogate model so that it addresses the exploration part. Fig. 3.23 shows an example of a sequential LHS ($SLHS$) optimized via the E^{ME} criterion, starting with $N = 2$ up to $N = 16$ compared to a one-shot LHS with $N = 16$. In most cases, it is not possible to get the $SLHS$ as good as a one-shot LHS as far as space-filling and correlations are concerned. Nevertheless, the figure depicts that in each sampling step the points are set to optimize the space filling property as good as possible.

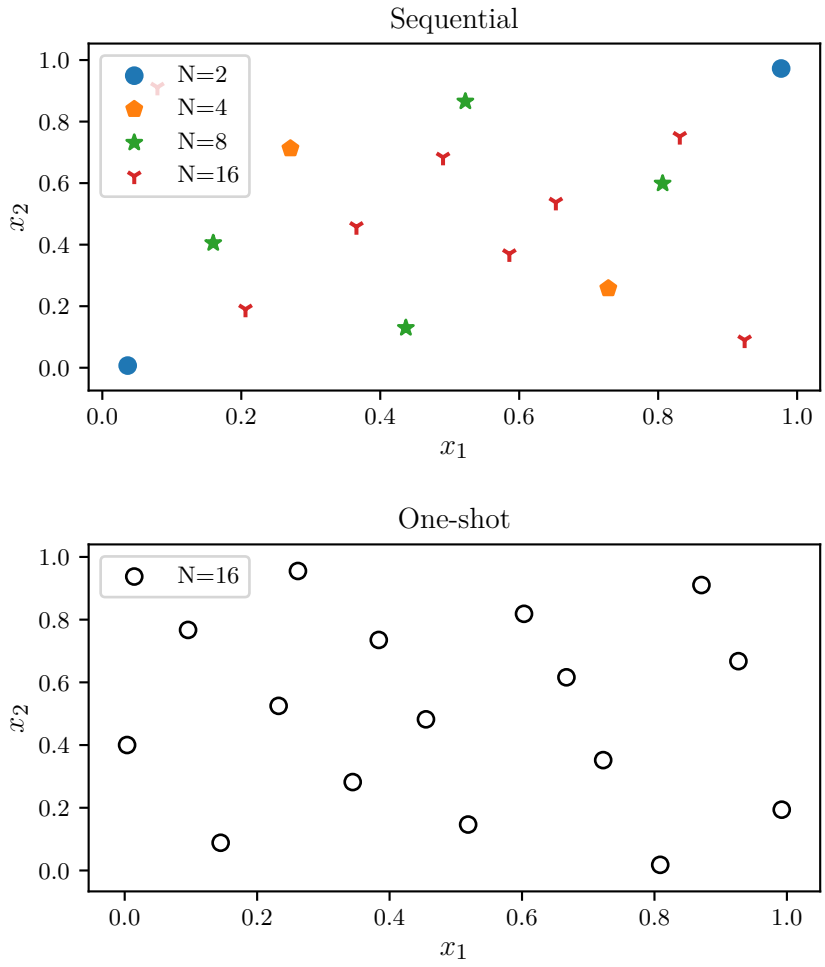


Figure 3.23: Comparison of a sequential LHS vs. a one-shot LHS up to $N = 16$ samples, optimized via the E^{ME} criterion.

As a example to illustrate the sequential LHS, the modified Branin-Hoo (eq. 2.7) function will be approximated by a $\mathcal{DGCN}$ model. For this example, the uncertainty of the model over the design domain will be compared with the position of the extension points for the different distance-based criteria introduced in section 3.3. The sample sizes are $N = 10, 20, 40$, so there are two adaptations. Furthermore, the difference for the E^{ME} criterion will be investigated, to see whether after the first DoE, the use of the trained vs. untrained covariance matrix (optimized correlation length-scales) influences the further adaptations. The trained or conditioned metric is named as $E^{ME_{cond}}$. Additionally, after each training of the $\mathcal{DGCN}$ model, once again a test set with $1,000$ uniform samples is used to evaluate the $\hat{R}^2$ together with the $RMSE$ as in the non-sequential test. This test is also repeated 25 times to measure the mean and standard deviation of the results. Also, the E^{MAPCC} is measured again.

The results for the model uncertainty of one repetition are shown in fig. 3.24. The subplots illustrate the variance of the model over the design space. The magenta colored points are the points the model is trained with, therefore the variance is zero at these positions.

The white points show the adaptation of the $SLHS$. In tab. 3.4 the corresponding mean and standard deviation of the $RMSE$, $\hat{R}^2$ and E^{MAPCC} are given for the 25 repetitions. The figure demonstrates that all algorithms add additional points in positions with high model uncertainty. Since the uncertainty relates to the distance of neighboring points, distance-based criteria work well to create globally accurate models. Especially the E^{ME} and $E^{ME_{cond}}$ criterion seem to set the point very well based on the model uncertainty. The results regarding the $\hat{R}^2$ and the $RMSE$ show that the E^{CL_2} as well as the E^{ME} / $E^{ME_{cond}}$ criterion have the lowest $RMSE$ and highest $\hat{R}^2$ values. A large difference regarding the estimated metrics between the conditioned and the unconditioned covariance matrices cannot be identified in this example. Compared to the non-sequential test, the maximal absolute correlation is increased for all criteria. Especially for the E^{EMM} criterion, it is quite high. The E^{CL_2} together with the E^{MADCC} still have acceptable values.

Typically, a design plan contains not only important parameters that have a decisive influence on the output variables to be investigated, but also parameters that have no influence. However, even the unimportant parameters have an influence on how many samples are necessary to cover the design space sufficiently and are considered in the distance-based optimization criteria. This could be avoided if it was known during the sequential expansion of the samples which parameters have an influence and which do not. In this case, the optimization could only be performed in the important subspaces of the design space so that these are well covered. The surrogate model to be trained (if it recognizes the correct important parameters) would then have enough information, even with fewer samples to cover the design space accurately. Therefore, another example shall illustrate whether the $E^{ME_{cond}}$ criterion has advantageous properties, even if unimportant parameters are included in the design plan. The idea is that the conditioned covariance matrix includes the information of unimportant parameters indirectly because an optimization of unimportant parameters might not influence the $E^{ME_{cond}}$ as strongly as for important parameters. Therefore, the previous test will be repeated but with 5 instead of 2 parameters, by adding 3 irrelevant parameters. The bounds for the 3 artificial parameters are $x_3 \in \mathbb{R}^{[-3,2]}$, $x_4 \in \mathbb{R}^{[2,4]}$, $x_5 \in \mathbb{R}^{[4,8]}$. The correlation length-scales should reduce to zero for these parameters and influence the covariance matrix for the $E^{ME_{cond}}$ metric. In the optimal case, the sequential adaptation focus on the optimization in the lower x_1, x_2 projection. This example should also show how the other distance-based metrics influence the surrogate model approximation in combination with unimportant parameters. The sampling steps are once again $N = 10, 20, 40$.

The results for the corresponding $RMSE$, $\hat{R}^2$ and E^{MADCC} are listed in tab. 3.5. The first noticeable point is that the $RMSE$ values have nearly doubled. It appears that due to the unimportant parameters, it is more difficult for the model to get the same low errors as without the unimportant parameters with the same number of training samples. This can also be seen on the correlation length-scales illustrated in fig. 3.25 which show the length-scales for the previous test without any unimportant parameters and fig. 3.26 for the case with unimportant parameters. Unimportant parameters should have a correlation length-scale close to zero and important ones should have a larger value. For the case with $n = 5$ it can be seen that the correlation length-scales of the unimportant parameters interfere with the prediction because even the unimportant parameters consume information content from the samples - e.g. for determining the correlation lengths in the model. This information content is then no longer available for the important parameters. The explanation of the

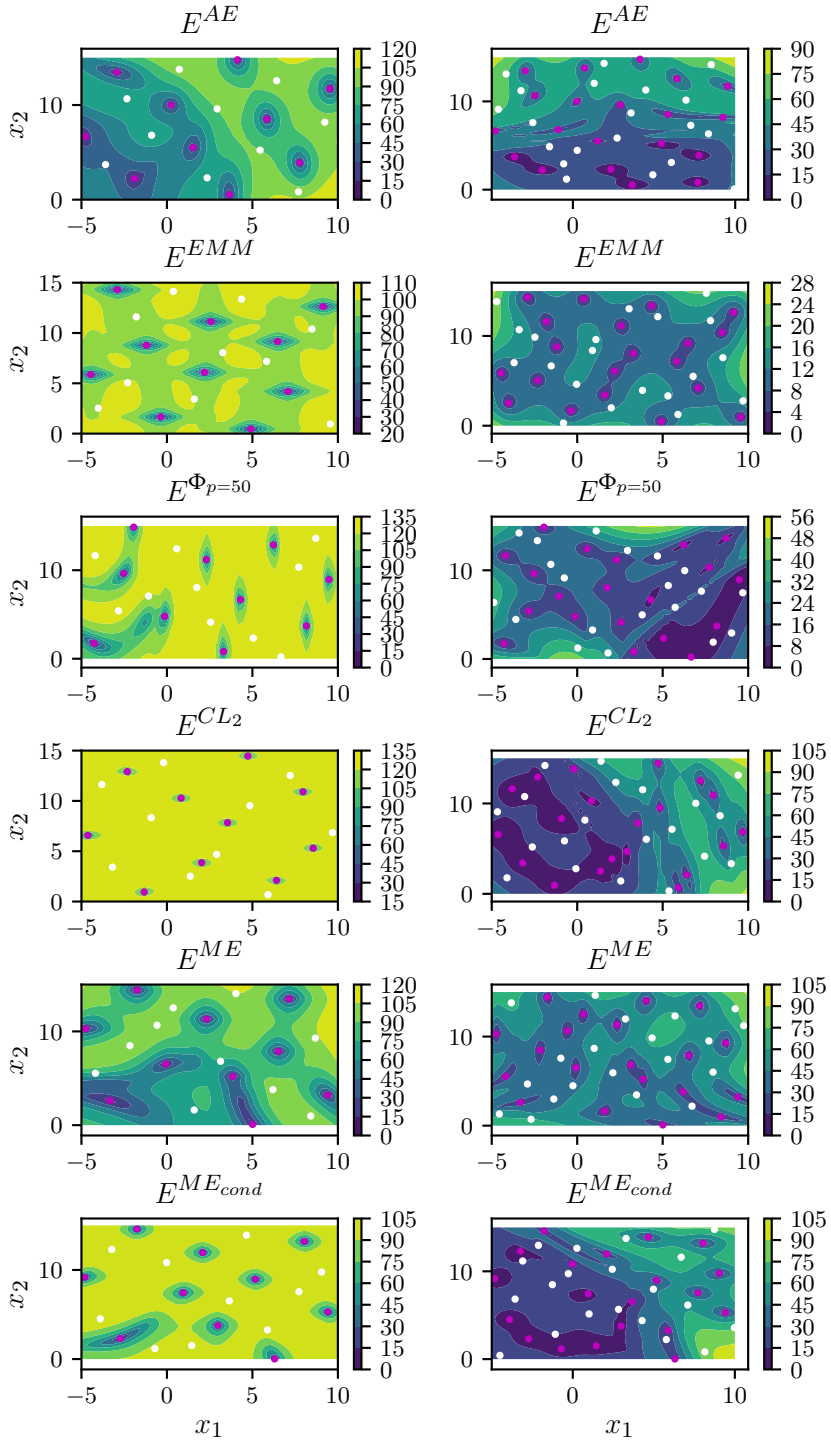


Figure 3.24: Comparison of different distance-based criteria for the sequential LHS. The magenta colored points show the previous DoE and the white points the next adaptation. The contour plots illustrate the variance (uncertainty) of the $\mathcal{DGCN}$ model based on the magenta colored points. The DoE sizes are $N = 10, 20, 40$. The left column shows the first adaptation the right column the second adaptation.

variance in the training data on the other side seems to remain the same and results in nearly equal $\hat{R}^2$ values for both cases. Also, the E^{MAPCC} are very similar. Another notable point is that for the case with unimportant parameters the $RMSE$ results are the lowest for the $E^{ME_{cond}}$ criterion now with a significant difference to the E^{ME} criterion after the second adaptation. For comparison, the length-scales trained on the E^{ME} optimized sampling are also shown in fig. 3.27. It seems that for samplings with unimportant parameters the $E^{ME_{cond}}$ can use the modification of the covariance matrix by the length-scales going to zero which leads to a better performing surrogate model. Incidentally, this example shows the advantage of a possible pre-variable selection before a model is trained. Possible methods with this purpose will be presented in chapter 4.

N	10	20	40
		$\hat{R}^2$	
E^{AE}	0.16±0.27	**0.84±0.09**	0.98±0.03
E^{EMM}	**0.19±0.25**	0.81±0.16	0.99±0.03
E^{CL_2}	0.17±0.25	0.82±0.19	0.99±0.004
E^{ME}	0.11±0.22	0.77±0.22	0.99±0.01
$E^{ME_{cond}}$	0.09±0.17	0.79±0.17	**0.99±0.003**
E^{MADCC}	0.22±0.29	0.83±0.09	0.99±0.01
		$RMSE$	
E^{AE}	28.77±7.75	14.09±5.23	2.61±1.87
E^{EMM}	28.41±7.77	**12.71±5.13**	2.52±1.63
E^{CL_2}	27.72±5.02	15.19±7.17	**1.90±0.86**
E^{ME}	27.80±6.66	15.23±6.23	2.11±1.20
$E^{ME_{cond}}$	26.92±6.65	14.03±4.34	2.35±1.51
E^{MADCC}	**26.18±3.35**	13.44±4.64	2.46±1.57
		E^{MAPCC}	
E^{AE}	0.11±0.08	0.09±0.07	0.07±0.06
E^{EMM}	0.15±0.10	0.15±0.09	0.12±0.09
E^{CL_2}	0.04±0.03	0.03±0.02	0.02±0.01
E^{ME}	0.08±0.04	0.10±0.06	0.06±0.04
$E^{ME_{cond}}$	0.08±0.04	0.10±0.08	0.08±0.06
E^{MADCC}	**0.03±0.02**	**0.03±0.02**	**0.01±0.01**

Table 3.4: Comparison of distance-based sequential LHS objectives and the E^{MADCC} criterion for surrogate model training for the modified Branin-Hoo test function with 2 input parameters. The bold numbers indicate the best results considering the mean and the standard deviation.

N	10	20	40
		$\hat{R}^2$	
E^{AE}	0.34±0.35	0.84±0.17	0.99±0.01
E^{EMM}	**0.36±0.34**	0.74±0.24	0.99±0.01
E^{CL_2}	0.38±0.38	**0.86±0.16**	0.99±0.01
E^{ME}	0.22±0.33	0.83±0.23	0.98±0.07
$E^{ME_{cond}}$	0.26±0.36	0.82±0.17	0.99±0.01
E^{MADCC}	0.28±0.34	0.79±0.21	**1.00±0.004**
		$RMSE$	
E^{AE}	50.78±7.57	**29.63±6.91**	7.07±3.26
E^{EMM}	55.75±12.62	31.77±9.52	5.11±2.75
E^{CL_2}	48.90±6.90	30.66±6.86	5.79±2.80
E^{ME}	**50.54±4.76**	31.77±9.25	7.16±3.77
$E^{ME_{cond}}$	48.57±8.33	31.01±8.56	**4.88±2.31**
E^{MADCC}	49.85±9.52	29.89±8.22	5.34±2.07
		E^{MAPCC}	
E^{AE}	**0.10±0.03**	0.19±0.05	0.13±0.03
E^{EMM}	0.62±0.11	0.45±0.11	0.31±0.06
E^{CL_2}	0.17±0.04	0.16±0.04	**0.10±0.03**
E^{ME}	0.12±0.03	0.24±0.05	0.17±0.04
$E^{ME_{cond}}$	0.11±0.03	0.28±0.07	0.25±0.08
E^{MADCC}	0.17±0.05	**0.17±0.03**	0.11±0.02

Table 3.5: Comparison of distance-based sequential LHS objectives and the E^{MADCC} criterion for surrogate model training for the modified Branin-Hoo test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation.

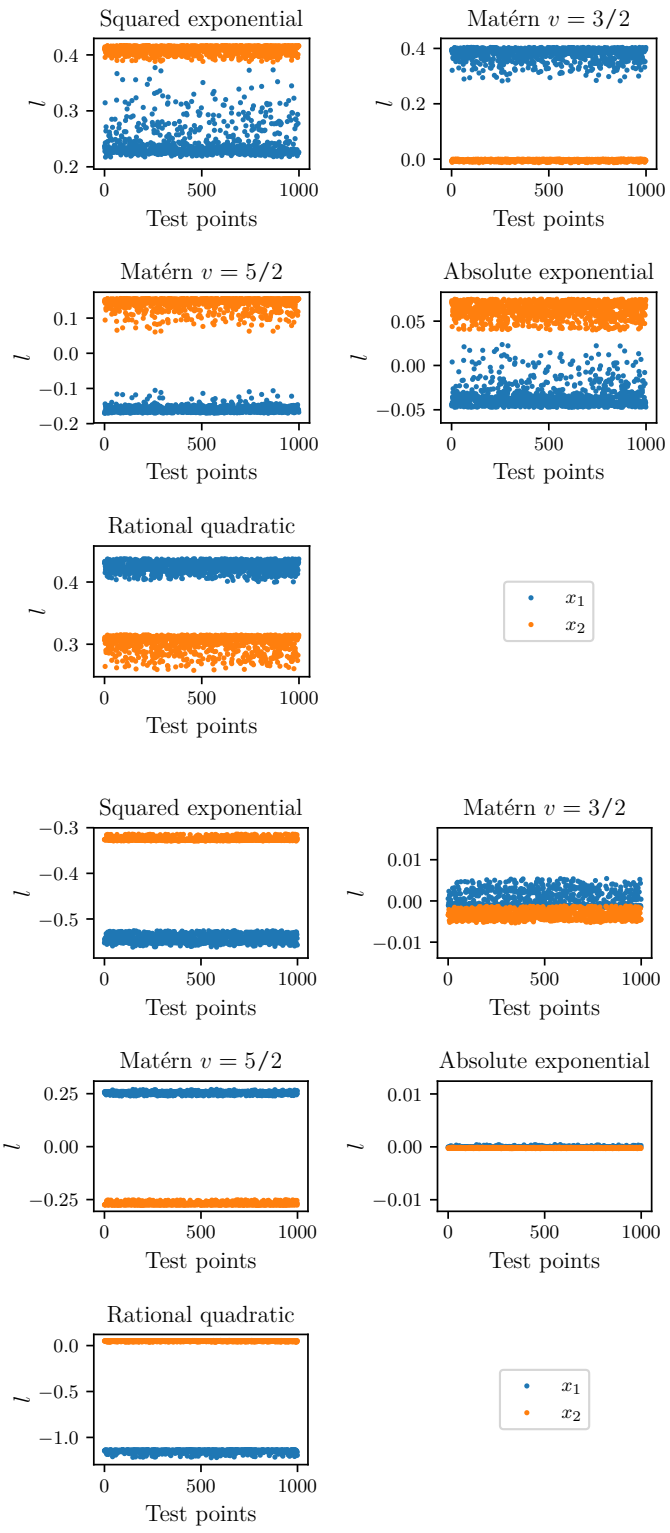


Figure 3.25: Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20$, 40 optimized via the $E^{ME_{cond}}$ objective with $n = 2$. 1,000 test points are shown.

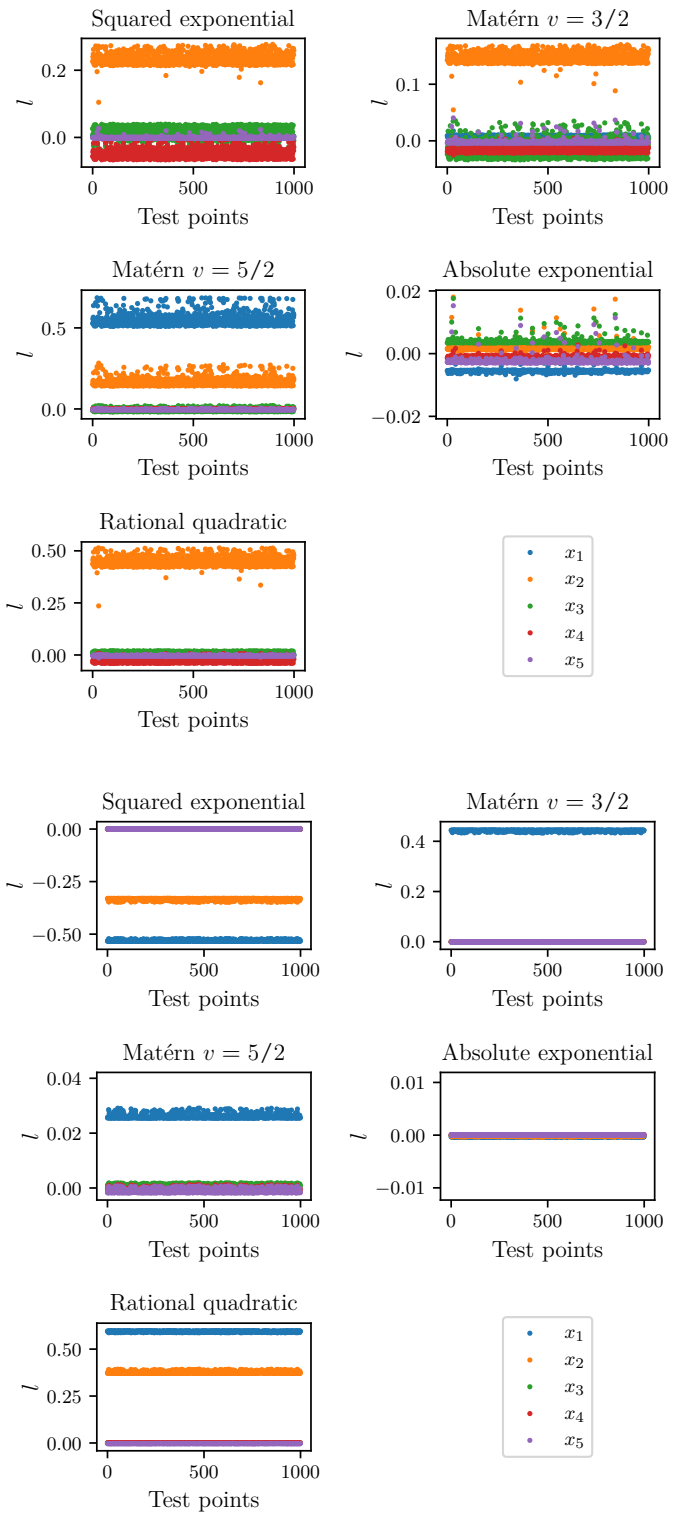


Figure 3.26: Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20$, 40 optimized via the $E^{ME_{cond}}$ objective with $n = 5$. 1,000 test points are shown.

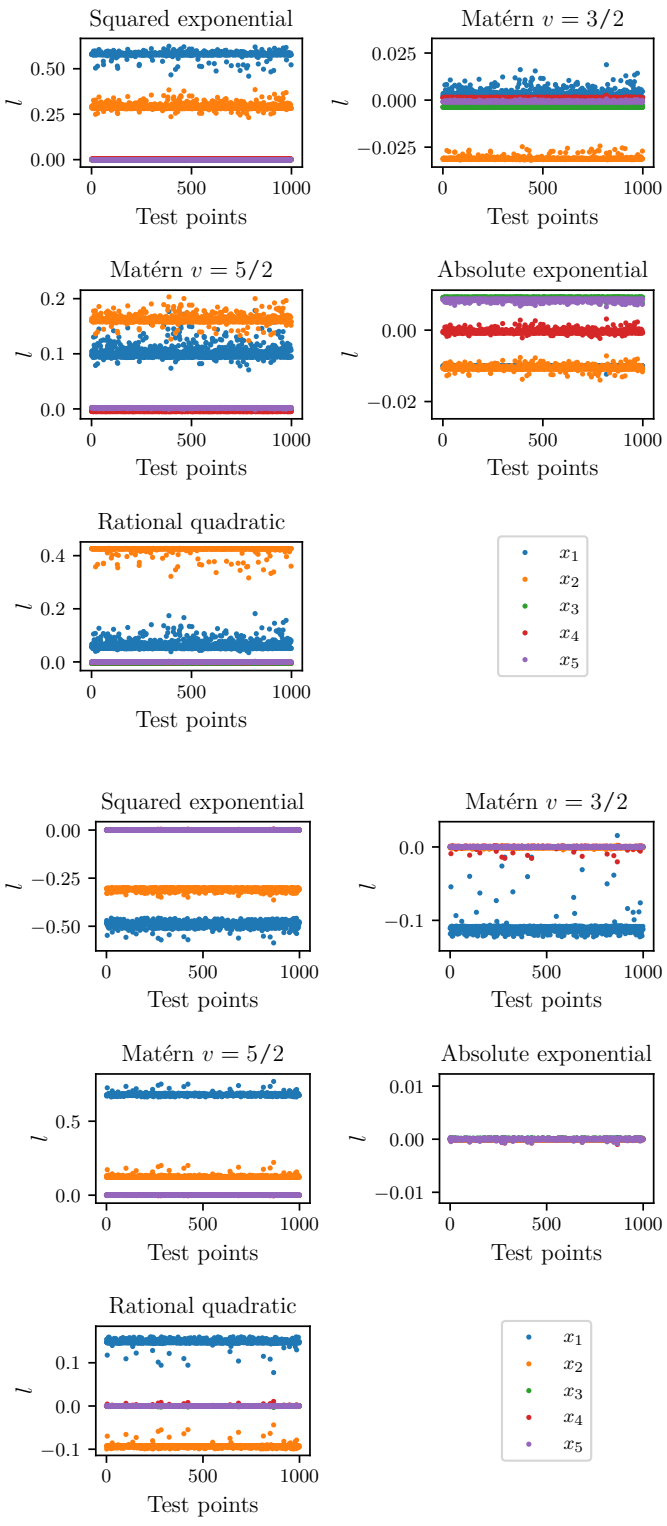


Figure 3.27: Correlation length-scales predictions of the DNN_l part of a $\mathcal{DGCN}$ model after the first adaptation (top) and second adaptation (bottom) with $N = 20$, 40 optimized via the E^{ME} objective with $n = 5$. 1,000 test points are shown.

To further confirm this assumption, the other two already used test functions of section 2.2 will be used, namely the Himmelblau (eq. 2.8) and Eggholder (eq. 2.9) test functions. The last test case will be repeated for these two test function only for the E^{ME} and $E^{ME_{cond}}$ sampling criteria. The parameter limits of x_1 and x_2 are modified according to these test functions. The sampling sizes are increased to $N = 20, 40, 80$ for the Himmelblau test function and $N = 100, 200, 400$ for the Eggholder test function due to their higher complexity.

The results in tab. 3.6 and tab. 3.7 show that the $E^{ME_{cond}}$ criterion works better overall than the E^{ME} regarding the $RMSE$ metric. However, this improvement is sometimes only by a small margin so that an absolute recommendation for the $E^{ME_{cond}}$ over the E^{ME} cannot be given based on these experiments and could be a topic for future research.

N	20	40	80
		$\hat{R}^2$	
E^{ME}	0.49±0.35	**0.77±0.20**	**1.00±0.01**
$E^{ME_{cond}}$	**0.57±0.35**	0.68±0.28	**1.00±0.01**
		$RMSE$	
E^{ME}	127.78±11.19	**68.49±26.08**	9.22±4.81
$E^{ME_{cond}}$	**124.64±11.85**	73.54±27.83	**8.14±3.81**
		E^{MAPCC}	
E^{ME}	**0.07±0.02**	**0.17±0.05**	**0.13±0.03**
$E^{ME_{cond}}$	**0.07±0.02**	0.23±0.06	0.21±0.05

Table 3.6: Comparison of E^{ME} and the $E^{ME_{cond}}$ criterion for surrogate model training for the Himmelblau test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation.

3.4.2 *One-by-One Adaptation*

It is not always possible to double the number of existing designs due to a limitation of time and computational resources. Therefore, another option is to add additional samples individually one-by-one. One very famous approach in combination with solving an optimization problem is called efficient global optimization (EGO) presented in Jones et al. [1998] based on the works of Kushner [1964] and Močkus [1975]. This method relies on the expected improvement, a Bayesian estimator gives for a given test point $\boldsymbol{x}_*$. In literature, this Bayesian estimator is in most cases a Gaussian process (Snoek et al. [2012], Frazier and Wang [2016]). Therefore, this method is especially suitable for the $\mathcal{DGCN}$ model. Assuming N points are evaluated with the corresponding observations $\boldsymbol{y}$ and it should be assumed that a minimum for y is required. Then y_{best} will denote the best observed result so far. If a new point $\boldsymbol{x}_*$ is evaluated, the mathematical maximum improvement will be

N	100	200	400
		$\hat{R}^2$	
E^{ME}	0.15±0.19	0.25±0.30	0.80±0.23
$E^{ME_{cond}}$	**0.14±0.17**	**0.36±0.30**	**0.80±0.04**
		$RMSE$	
E^{ME}	**339.18±16.24**	263.54±71.08	128.89±51.53
$E^{ME_{cond}}$	339.60±17.83	**236.67±69.69**	**122.09±9.59**
		E^{MAPCC}	
E^{ME}	0.04±0.01	**0.06±0.02**	**0.06±0.01**
$E^{ME_{cond}}$	**0.03±0.01**	0.07±0.02	0.06±0.02

Table 3.7: Comparison of E^{ME} and the $E^{ME_{cond}}$ criterion for surrogate model training for the Eggholder test function with 5 input variables (additional 3 unimportant parameters). The bold numbers indicate the best results considering the mean and the standard deviation.

$$I(\boldsymbol{x}_*) = max\left(y_{best} - y_*, 0\right) \tag{3.27}$$

with y_* being a random variable with $\mathcal{N}(\hat{y}_*, \hat{s}_*)$ that corresponds to the function value at $\boldsymbol{x}_*$. The statistical moments $\hat{y}_*$ and $\hat{s}_* = \sqrt{\mathbb{V}\left[\hat{y}_*\right]}$ (see eq. 2.130 or eq. 2.68) are based on the $\mathcal{GP}$ or $\mathcal{DGCN}$ posterior mean prediction and standard error. Therefore, the probability density function (PDF) $\phi\left(\frac{y_*-\hat{y}_*}{\hat{s}_*}\right)$ and the cumulative density function (CDF) $\Phi\left(\frac{y_*-\hat{y}_*}{\hat{s}_*}\right)$ of the random variable y_* can be calculated. Since the improvement $I(\boldsymbol{x}_*)$ is also a random variable, the average expected improvement $E[I]$ (EI) of $\boldsymbol{x}_*$ is simply the improvement $I(\boldsymbol{x}_*)$ times its probability of improvement $PoI(\boldsymbol{x}_*)$ (to obtain a lower minima than observed so far) which is defined as

$$PoI(\boldsymbol{x}_*) = \int_{-\infty}^{y_{best}} \phi\left(\frac{y_* - \hat{y}_*}{\hat{s}_*}\right) dy_* = \Phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right) \tag{3.28}$$

and $EI(\boldsymbol{x}_*)$ can be calculated by (see Hupkens et al. [2014])

$$
\begin{aligned}
EI(\boldsymbol{x}_*) &= \mathbb{E}\left[I(\boldsymbol{x}_*)\right] = \mathbb{E}\left[max\left(y_{best} - y_*, 0\right)\right] = \int_{-\infty}^{y_{best}} I(\boldsymbol{x}_*) PoI(\boldsymbol{x}_*) dy_* \\
&= \int_{-\infty}^{y_{best}} \left(y_{best} - y_*\right) \phi\left(\frac{y_* - \hat{y}_*}{\hat{s}_*}\right) dy_* \\
&= \int_{-\infty}^{y_{best}} \left(y_{best} - y_*\right) \frac{1}{\hat{s}_*\sqrt{2\pi}} \exp\left(-\frac{\left(y_* - \hat{y}_*\right)^2}{2\hat{s}_*^2}\right) dy_* \\
&\text{subst. } u = \frac{y_* - \hat{y}_*}{2\hat{s}_*^2} \rightarrow dy_* = \hat{s}_* du \\
&= \frac{\hat{s}_*}{\hat{s}_*\sqrt{2\pi}} \int_{-\infty}^{\frac{y_{best} - \hat{y}_*}{\hat{s}_*}} \left(y_{best} - \left(u\hat{s}_* + \hat{y}_*\right)\right) \exp\left(-\frac{u^2}{2}\right) du \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{y_{best} - \hat{y}_*}{\hat{s}_*}} \left(y_{best} - \hat{y}_*\right) \exp\left(-\frac{u^2}{2}\right) du + \frac{\hat{s}_*}{\sqrt{2\pi}} \int_{-\infty}^{\frac{y_{best} - \hat{y}_*}{\hat{s}_*}} u \exp\left(-\frac{u^2}{2}\right) du \\
&= \left(y_{best} - \hat{y}_*\right) \Phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right) + \hat{s}_* \phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right)
\end{aligned}
\tag{3.29}
$$

In some works, only the $PoI(\boldsymbol{x}_*)$ is used, which is not recommended, since it does not measure how large the improvement is and favors a small but more probable improvement over a possible much larger but somewhat less likely improvement. To illustrate the recently presented equations, fig. 3.28 shows some of the relationships. The improvement can also be visualized by expanding $EI(\boldsymbol{x}_*) = \int_{-\infty}^{y_{best}} \left(y_{best} - y_*\right) \phi\left(\frac{y_* - \hat{y}_*}{\hat{s}_*}\right) dy_*$ and integrating the first factor, then EI can be written as

$$
\begin{aligned}
EI(\boldsymbol{x}_*) &= \left(y_{best} - \frac{\int_{-\infty}^{y_{best}} y_* \phi\left(\frac{y_* - \hat{y}_*}{\hat{s}_*}\right) dy}{\Phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right)}\right) \Phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right) \\
&= \left(y_{best} - \underbrace{\frac{\int_{-\infty}^{y_{best}} y_* \phi\left(\frac{y_* - \hat{y}_*}{\hat{s}_*}\right) dy}{PoI(\boldsymbol{x}_*)}}_{\bar{y}_*}\right) PoI(\boldsymbol{x}_*)
\end{aligned}
\tag{3.30}
$$

$\bar{y}_*$ denotes the center of mass (centroid) of the integral $\int_{-\infty}^{y_{best}}$.

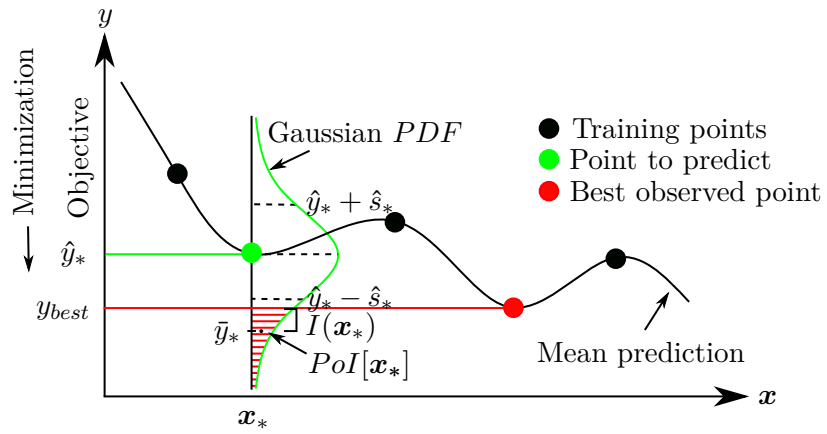


Figure 3.28: Illustration of the expected improvement based on a Gaussian process model prediction.

From eq. 3.29 can be derived that:

- $EI(\boldsymbol{x}_*)$ is high if the prediction $\hat{y}_*$ is lower than the current best value y_{best} or
- $EI(\boldsymbol{x}_*)$ is high if the uncertainty of $\hat{s}_*$ around the point $\boldsymbol{x}_*$ is high.

Intuitively this makes sense, since if the expected improvement is maximized, it will either sample points for which a low value of $\hat{y}_*$ is expected, or points in a region which have not yet been explored ($\hat{s}_*$ is high). In other words, there is a trade-off between exploitation and exploration. Jones et al. [1998] proposed to start with an initial optimized LHS and train a first $\mathcal{GP}$ on it. Then through expected improvement, new samples should be calculated until a required maximum expected improvement threshold is reached, e.g. 1%. This kind of optimization is also known as Bayesian optimization.

Typically, there are not always only objectives in an optimization problem but also constraints to be considered. If these constraint functions are independent of the objective function and also from each other, the expected improvement equation can be extended to (see. Schonlau et al. [1998])

$$EI(\boldsymbol{x}_*) = \left[(y_{best} - \hat{y}_*)\,\Phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right) + \hat{s}_*\phi\left(\frac{y_{best} - \hat{y}_*}{\hat{s}_*}\right)\right] \prod_{i=1}^{n_{con}} \Phi\left(\frac{\hat{y}_{i*}}{\hat{s}_{i*}}\right) \tag{3.31}$$

Note that the $\hat{y}_{i*}$ are the predictions of the constraint outputs and the formulation holds for the case that all constraint must be greater equal zero. For the case that the values of constraint variables are smaller or equal zero, the product term must be changed to $\Phi\left(-\frac{\hat{y}_{i*}}{\hat{s}_{i*}}\right)$.

Furthermore, there are often multiple outputs to be optimized. In literature, there are several works on this topic for the expected improvement algorithm, such as the centroid method (Forrester et al. [2008]), or the hypervolume-based methods (Emmerich et al. [2006], Hupkens et al. [2014], Couckuyt et al. [2014]). All of them are numerically very expensive due to the calculation of the probability integral and the calculation of the hypervolume. A very good overview over the existing methods is given in Kristensen et al. [2016] and Feliot et al. [2017]. Furthermore, in most cases the existing literature only uses these methods for two to three objective functions. The analytical calculation of the probability integrals becomes more complex with increasing dimensionality. A detailed explanation of these methods is beyond the scope of this work. Nevertheless, for a multi-objective optimization problem, the expected improvement is very often replaced by the expected hypervolume improvement criterion mentioned which will be briefly explained. Fig. 3.29 illustrates the definition of the hypervolume which is enclosed by a reference point (in the figure marked as $\boldsymbol{y}_{max}$) and the current Pareto-front Y_P of all objectives. The Pareto-front is defined by a set of points which are non-dominated by any other points and each point represents a solution, where it is not possible to improve one objective without decreasing one of the others. In other words, each point of the Pareto-frontier represents the optimal compromise solution for all objectives. For a minimization problem the dominated hypervolume HV should be maximized in order to minimize the objectives. The hypervolume improvement is defined as

$$I_{hyp}(\boldsymbol{x}_*) = max\left(HV\left(Y_P \cup \boldsymbol{x}_*\right) - HV(Y_P), 0\right) \tag{3.32}$$

together with the probability of the hypervolume improvement:

$$PoI_{hyp}(\boldsymbol{x}_*) = \int_{\boldsymbol{y}\in\overline{HV}} PDF_{\boldsymbol{x}}(\boldsymbol{y})d\boldsymbol{y} \tag{3.33}$$

and the corresponding expected hypervolume improvement EI_{hyp} as

$$EI_{hyp}(\boldsymbol{x}_*) = \int_{\boldsymbol{y}\in\overline{HV}} I_{hyp}(\boldsymbol{x}_*)PDF_{\boldsymbol{x}}(\boldsymbol{y})d\boldsymbol{y} \tag{3.34}$$

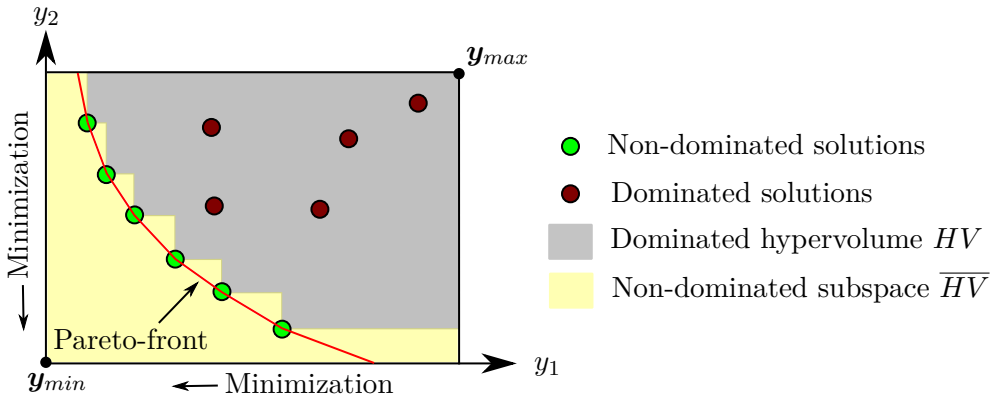


Figure 3.29: Exemplary Pareto-front for two objectives which should be minimized.

The integration of this equation is difficult, even if, as already mentioned, statistical independence of the objectives is assumed. Typically, even in this case, piecewise numerical integration over n_{obj} (number of objectives) dimensional hyper-rectangles seems to be the most appropriate way to deal with this integration. As a simple alternative, one may use Monte-Carlo integration by producing N random samples (Emmerich et al. [2006])

$$\boldsymbol{y}_{i*} = \left[y_{1*} \sim \mathcal{N}(\hat{y}_{1*}, \hat{s}_1*), ..., y_{j*} \sim \mathcal{N}(\hat{y}_{n_{obj}*}, \hat{s}_{n_{obj}*})\right],\ i = 1, ..., N,\ j = 1, ..., n_{obj} \tag{3.35}$$

of the Gaussian distribution with mean $\hat{y}_{j*}$, $j = 1, ..., n_{obj}$ and standard error $\hat{s}_j*$, $j = 1, ..., n_{obj}$ for each of the objectives separately (if they have an independent Gaussian distribution) and by measuring the increase in hypervolume $I_{hyp}(\boldsymbol{y}_{i*})$. Then, the approximate integral reads

$$EI_{hyp}(\boldsymbol{x}_*) \approx \frac{1}{N}\sum_{i=1}^{N} I_{hyp}(\boldsymbol{y}_{i*}) \tag{3.36}$$

This equation can also be generalized for the constrained multi-objective case, assuming independent constraint functions to

$$PoI_{hyp}(\boldsymbol{x}_*) = \int_{\boldsymbol{y}\in\overline{HV}} PDF_{\boldsymbol{x}}(\boldsymbol{y})d\boldsymbol{y} \prod_{i=1}^{n_{con}} \Phi\left(\frac{\hat{y}_{i*}}{\hat{s}_{i*}}\right) \tag{3.37}$$

and

$$EI_{hyp}(\boldsymbol{x}_*) = \int_{\boldsymbol{y}\in\overline{HV}} I_{hyp}(\boldsymbol{x}_*)PDF_{\boldsymbol{x}}(\boldsymbol{y})d\boldsymbol{y} \prod_{i=1}^{n_{con}} \Phi\left(\frac{\hat{y}_{i*}}{\hat{s}_{i*}}\right) \tag{3.38}$$

In this work, the Monte Carlo integration method is used for multiple objectives, since it is a fast method for evaluating many objectives that is capable of handling the complex

probability integral. It is important to note that the objectives must be normalized or scaled for the hypervolume improvement measure in order to avoid the optimization process being dominated by one or a few objectives. Additionally, it is important to choose a sufficient reference point. It is suggested to use the current maximum values of the objectives times a factor for a minimization problem. This ensures that the hypervolume is always determined using all currently available points. A reference point smaller than the current maximum would ensure that some points would be excluded from the optimization if necessary and then not be considered in the further search for the optimal Pareto front. This may result in potential losses.

As an example, the modified Branin-Hoo and Eggholder test function will be optimized via the expected improvement approach and a $\mathcal{DGCN}$ model. Both test functions have one global optimum which is at $x_1 \approx -3.69$, $x_2 \approx 13.63$ with $y \approx -16.64$ for the Branin-Hoo function and at $x_1 = 512$, $x_2 \approx 404.23$ with $y \approx -959.64$ for the Eggholder function. The Eggholder function is a difficult optimization function to deal with due to its many local minima. A start DoE is used for both functions with $N = 10$ for the Branin-Hoo function and $N = 50$ for the Eggholder function optimized via the E^{CL_2} criterion. The abort criterion for the optimization was a budget of 10 additional samples or an $EI(\boldsymbol{x}_*) < 0.05$ for the Branin-Hoo function, and for the Eggholder function, the budget of samples was 150. During the optimization in each iteration the $\mathcal{DGCN}$ model is updated with the new proposed sample and a short retrain based on the actual sampling is carried out. As already mentioned, $\mathcal{DGCN}$ has the advantageous ability to perform so-called online learning (see section 2.9). So a new complete training does not need to be done, only a quick update with a few epochs for the neural network part is needed. A standard $\mathcal{GP}$ would need a full retrain with every new sample to adjust the length-scales, this might be very time consuming dependent on the required number of samples during the optimization. To find the next point $\boldsymbol{x}_*$ which maximizes the expected improvement, a differential evolution algorithm (Storn and Price [1997]) is used as an optimizer of the Python open source library Scipy (Jones et al. [2001]). The evaluations for this search are performed on the currently updated $\mathcal{DGCN}$ model.

Fig. 3.30 and 3.31 illustrate the history of the optimizations for both test functions as well as the position of the global optimum and the start DoE. For the Branin-Hoo function, the optimization found the global optimum very quickly, only one point was outside the global optimum area. For the Eggholder function, the higher budget of samples was needed because the expected improvement algorithm found several local minima next to the global one. It does not fit the global optimum perfectly but is very close with only $N = 200$ samples overall. If this sample count is compared to the accuracy the surrogate model has in the previous tests, it shows that even with $N = 400$ samples, no sufficient approximation quality was obtainable. Thus, an optimization on the globally trained model might not give such a good result.

Another example will be given for the constrained multi-objective case in comparison to Feliot et al. [2017], where multiple multi-objective test functions were taken from Deb et al. [2002] and Chafekar et al. [2003]. In this work, a standard Gaussian process model was used in connection with a different newly developed extension to the EI_{hyp} criterion and advanced optimization methods. In the new extension the constraints are handled using an extended domination rule and a new expected improvement formulation is proposed. In particular, the new formulation makes it possible to deal with problems where no fea-

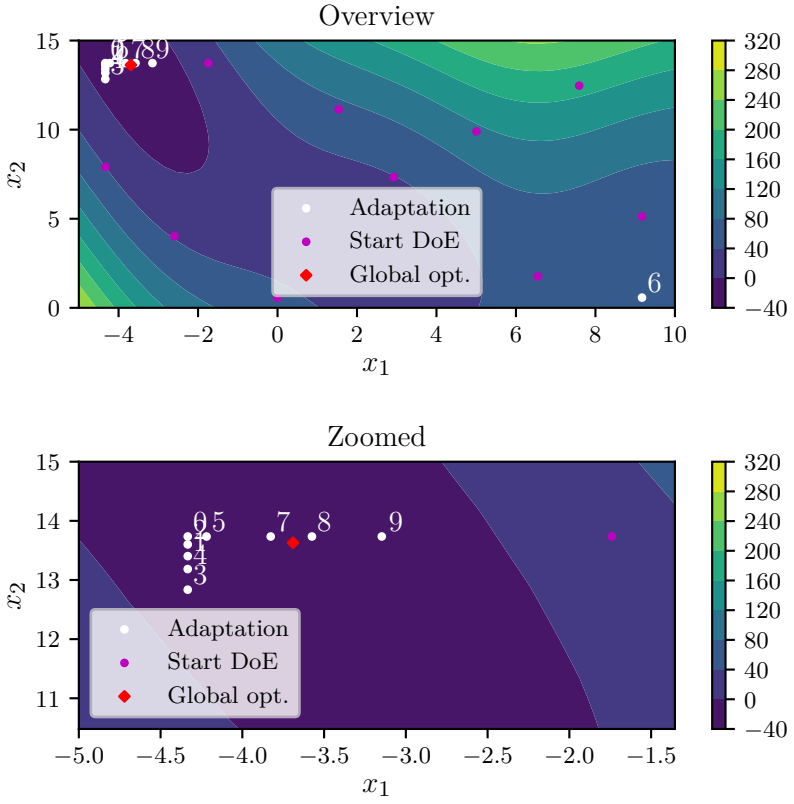


Figure 3.30: Expected-improvement-based optimization of the modified Branin-Hoo function. The global optimum is at $x_1 \approx -3.69$, $x_2 \approx 13.63$ with $y \approx -16.64$. A start *DoE* optimized via the E^{CL_2} criterion of $N = 10$ samples was used. The numbers indicate the order of the samples during the optimization.

sible solution is available from the start (see Feliot et al. [2017] for more details). For the benchmark, the best results of their work are always used, since they compared different settings for their new optimization approach. As a performance measure, they estimated the number of runs (adaptation steps) needed to dominate 90%, 95% and 99% of the possible hypervolume HV in the objective space[1]. Therefore, they repeated the optimization 30 times in order to estimate the mean and standard deviations of these values. If the 99% target was not reached by 250 function evaluations, the process was aborted. For initialization of the $\mathcal{GP}$ model $N = 3n$ samples were used. Feliot et al. [2017] used a LHS optimized via the E^{EMM} criterion for the initialization. Therefore, the constraints might be violated by the initial samples. Since in real world applications, it might not be possible to initiate designs violating the constraints, another method will be used here. For the initialization, a one-by-one sampling approach is used, where the first point is randomly chosen in order to be in the feasible region of the design space. Then points are added one-by-one via an optimization of the E^{EMM} criterion between the new point and the existing points. The optimization is performed in consideration of the constraints. Note, this is only possible for pure input parameter-based constraints, like in this example. This method has

1 Note the number of runs are given without the initial designs.

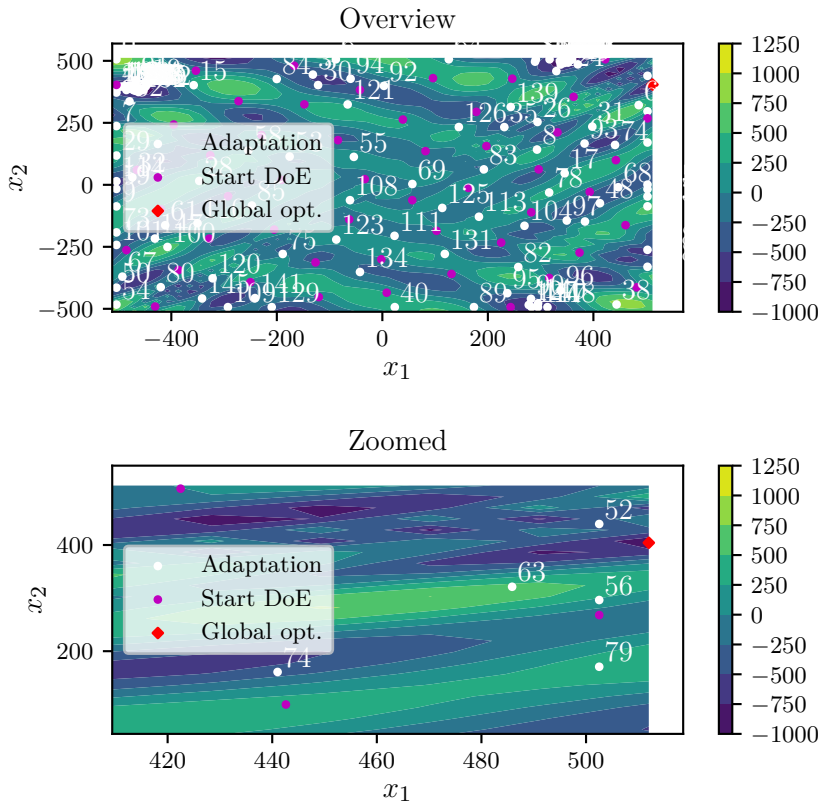


Figure 3.31: Expected-improvement-based optimization of the Eggholder function. The global optimum is at $x_1 = 512$, $x_2 \approx 404.23$ with $y \approx -959.64$. A start *DoE* optimized via the E^{CL_2} criterion of $N = 50$ samples was used. The numbers indicate the order of the samples during the optimization.

the advantage that all points of the initialization contribute to the HV estimation at the beginning. Tab. 3.8 gives an overview of the test functions. Γ gives an estimate of the ratio (in percents) of the volume of the feasible region to that of the search space. Column V gives the volume of the region dominated by the Pareto-front, measured using a reference point $\boldsymbol{y}_0^{ref}$. This information is taken from Feliot et al. [2017]. Fig. 3.32 and fig. 3.33 show the Pareto-frontiers obtained by using the NSGA III (Yuan et al. [2014]) algorithm on the analytical function. Furthermore, they compared their results to the results obtained with the method proposed in Couckuyt et al. [2014] which also focused on the EI-based optimization algorithm improvement so that a second comparison is possible.

The results of the aforementioned works and $\mathcal{DGCN}$ are presented in tab. 3.9. As an optimizer, the same differential evolution algorithm of the Python open source library Scipy (Jones et al. [2001]) was used. Additionally, the results listed are obtained by optimizing the true numerical function (named as reference) to allow for comparing the differences of the results due to the optimizer and due to $\mathcal{DGCN}$. The results show that for all test functions, except the 99% volume of the BNH test function, $\mathcal{DGCN}$ outperforms the other two mentioned works (Feliot et al. [2017], Couckuyt et al. [2014]). For this case, even on the true function, the optimizer was not able to reach the results of Feliot et al. [2017]. Interestingly, these two works focused to find an improved optimization algorithm to get

Problem	Bounds	Objectives	Constraints	Γ(%)	HV	$\mathbf{y}_0^{ref}$
BNH	$x_1 \in [0, 5]$ $x_2 \in [0, 3]$	$f_1 = 4x_1^2 + 4x_2^2$ $f_2 = (x_1 - 5)^2 + (x_2 - 5)^2$	$c_1 = (x_1 - 5)^2 + x_2^2 \leq 25$ $c_2 = (x_1 - 8) + (x_2 + 3)^2 \geq 7.7$	93.6	5,249	$[140, 50]$
SRN	$x_1 \in [-20, 20]$ $x_2 \in [-20, 20]$	$f_1 = (x_1 - 2)^2 + (x_2 - 1)^2 + 2$ $f_2 = 9x_1 - (x_2 - 1)^2$	$c_1 = x_1^2 + x_2^2 \leq 225$ $c_2 = x_1 - 3x_2 \leq -10$	16.1	29,234.95	$[200, 50]$
TNK	$x_1 \in [0, \pi]$ $x_2 \in [0, \pi]$	$f_1 = x_1$ $f_2 = x_2$	$c_1 = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(x_1/x_2)) \leq 0$ $c_2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$	5.1	0.6466	$[1.2, 1.2]$
OSY	$x_1 \in [0, 10]$ $x_2 \in [0, 10]$ $x_3 \in [1, 5]$ $x_4 \in [0, 6]$ $x_5 \in [1, 5]$ $x_6 \in [0, 10]$	$f1 = -\left[25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^4 + (x_5 - 1)^2\right]$ $f_2 = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$c_1 = x_1 + x_2 - 2 \geq 0$ $c_2 = 6 - x_1 - x_2 \geq 0$ $c_3 = 2 - x_2 + x_1 \geq 0$ $c_4 = 2 - x_1 + 3x_2 \geq 0$ $c_5 = 4 - (x_3 - 3)^2 - x_4 \geq 0$ $c_6 = (x_5 - 3)^2 + x_6 - 4 \geq 0$	3.2	16,169	$[0, 80]$
TwoBarTruss	$x_1 \in [0, 0.01]$ $x_2 \in [0, 0.01]$ $x_3 \in [1, 3]$	$f_1 = x_1\sqrt{16 + x_3^2} + x_2\sqrt{1 + x_3^2}$ $f_2 = max(\sigma_1, \sigma_2)$ $\sigma_1 = 20\sqrt{16 + x_3^2}/x_1x_3$ $\sigma_2 = 80\sqrt{1 + x_3^2}/x_2x_3$	$c_1 = max(\sigma_1, \sigma_2) \leq 10^5$	86.3	4,495	$[0.06,\ 10^5]$
WeldedBeam	$x_1 \in [0.125, 5]$ $x_2 \in [0.125, 5]$ $x_3 \in [0.1, 10]$ $x_4 \in [0.1, 10]$	$f_1 = 1.10471x_1^2x_3 + 0.04811x_4x_2(14 + x_3)$ $f_2 = 2.1952/x_4^3x_2$	$c_1 = 13,600 - \tau \geq 0$ $c_2 = 30,000 - \sigma \geq 0$ $c_3 = x_2 - x_1 \geq 0$ $c_4 = P_c - 6,000 \geq 0$ $\tau = \sqrt{(\tau')^2 + (\tau'')^2 + \tau'\tau''/\sqrt{0.25\,(x_3^2 + (x_1 + x_4)^2)}}$ $\tau' = 6,000/\sqrt{2}x_1x_3$ $\tau'' = \frac{6,000(14 + 0.5x_3)\sqrt{0.25\,(x_3^2 + (x_1 + x_4)^2)}}{2\sqrt{2}x_1x_3\,(x_3^2/12 + 0.25(x_1 + x_4)^2)}$ $\sigma = 504,000/x_4^2x_2$ $P_c = 64,746.022(1 - 0.0282346x_4)x_4x_2^3$	45.5	0.4228	$[50, 0.01]$
CONSTR	$x_1 \in [0.1, 1]$ $x_2 \in [0, 5]$	$f_1 = x_1$ $f_2 = (1 + x_2)/x_1$	$c_1 = x_2 + 9x_1 \geq 6$ $c_2 = -x_2 + 9x_1 \geq 1$	52.5	3.8152	$[1, 9]$
WATER	$x_1 \in [0.01, 0.45]$ $x_2 \in [0.01, 0.1]$ $x_3 \in [0.01, 0.1]$	$f_1 = (106,780.37(x_2 + x_3) + 61,704.67)/(8(10^4))$ $f_2 = (3,000x_1)/1,500$ $f_3 = ((305,700)2,289x_2/137.34^{0.65})/(3(10^6))$ $f_4 = ((250)2,289\exp(-39.75x_2 + 9.9x_3 + 2.74))/(6(10^6))$ $f_5 = (25(1.39/(x_1x_2) + 4,940x_3 - 80))/8,000$	$c_1 = 0.00139/(x_1x_2) + 4.94x_3 - 0.08 \leq 1$ $c_2 = 0.000306/(x_1x_2) + 1.082x_3 - 0.0986 \leq 1$ $c_3 = 12.307/(x_1x_2) + 49,408.24x_3 + 4,051.2 \leq 50,000$ $c_4 = 2.098/(x_1x_2) + 8,046.33x_3 - 696.71 \leq 16,000$ $c_5 = 2.138/(x_1x_2) - 7,883.39x_3 - 705.04 \leq 10,000$ $c_6 = 0.417(x_1x_2) + 1,721.26x_3 - 136.54 \leq 2,000$ $c_7 = 0.164/(x_1x_2) + 631.13x_3 - 54.48 \leq 550$	92	0.5138	$[1, 1, 1, 1.6, 3.2]$

Table 3.8: Overview of the constrained multi-objective test functions.

better performance in combination with a standard $\mathcal{GP}$ but it might be more beneficial to use a better surrogate model like demonstrated with the $\mathcal{DGCN}$ results which are made with a standard optimizer. It seems that it is much more beneficial to have a better surrogate model instead of an advanced optimizer. This makes sense, since the optimizer must rely on the surrogate model predictions. The optimization on the $\mathcal{DGCN}$ model was able in many cases to come close to the one on the real numerical function. Feliot et al. [2017] mentioned also in their work that the lag of predictive power of the used $\mathcal{GP}$ algorithm can lead to less good results. This example clearly shows the advantage of having non-stationary correlation length-scales in order to adapt to the non-uniform adaptive sampling.

As it is not always desirable to solve an optimization problem there must be also a good one by one sampling method which allows to create a global accurate model. As mentioned in subsection 3.4.1, sequential or adaptive sampling strategies have the advantage that only as many samples as necessary need to be evaluated. With a one by one strategy, in contrast to the sequential LHS which needs to double the amount of samples in each step, it might be possible to achieve the same result with even fewer samples and to monitor the convergence of the model training very closely. Derived from the idea of expected improvement, it might be useful for a global accurate model to concentrate only on the uncertainty of the model explained by $\hat{s}_*$ (see also Burnaev and Panov [2015]). Therefore, as an objective for the optimization to find the next sample, the following function is proposed:

$$E^s = \hat{s}_* \tag{3.39}$$

Problem	Method	Target 90%	Target 95%	Target 99%
BNH	Feliot et al. [2017]	8.5±0.6	12.7±0.7	**34.6±1.3**
	Couckuyt et al. [2014]	8.4±0.6	12.8±0.7	38.9±2.2
	$\mathcal{DGCN}$	**8.17±0.69**	**12.23±0.67**	38.07±0.63
	Reference	*7.9±0.7*	*12.13±0.76*	*36.93±0.77*
SRN	Feliot et al. [2017]	16.7±0.9	22.4±1	52.6±4.1
	Couckuyt et al. [2014]	20.5±2.4	35.6±5.9	>250
	$\mathcal{DGCN}$	**11.5±0.99**	**16±1.29**	**20.37±1.05**
	Reference	*11.13±0.66*	*12.9±0.59*	*19.03±0.7*
TNK	Feliot et al. [2017]	35.5±2.6	44.1±2.5	71.1±4
	Couckuyt et al. [2014]	43.5±4.6	71.6±11.3	>250
	$\mathcal{DGCN}$	**11.57±2.17**	**10.33±0.98**	**24±1.83**
	Reference	*9.06±0.72*	*9.86±0.76*	*21.13±0.66*
OSY	Feliot et al. [2017]	29±1.7	38.2±3.4	119.8±53
	Couckuyt et al. [2014]	>250	>250	>250
	$\mathcal{DGCN}$	**22.37±2.01**	**22.57±1.93**	**37.8±1.01**
	Reference	*20±0.68*	*20.06±0.72*	*32.93±0.72*
TwoBarTruss	Feliot et al. [2017]	90.9±62	234±-	>250
	Couckuyt et al. [2014]	88.7±68.4	162±29.7	>250
	$\mathcal{DGCN}$	**14.33±0.91**	**16.73±1.57**	**23.07±1.39**
	Reference	*10.83±0.77*	*11.13±0.66*	*11.8±0.65*
WeldedBeam	Feliot et al. [2017]	146.5±41.1	212±33.9	>250
	Couckuyt et al. [2014]	171.3±49.6	229±-	>250
	$\mathcal{DGCN}$	**13.43±0.5**	**13.63±0.48**	**13.53±0.5**
	Reference	*13.03±0.79*	*13.03±0.65*	*13.06±0.77*
CONSTR	Feliot et al. [2017]	12.4±1	19.2±1.4	83.5±5.9
	Couckuyt et al. [2014]	13.8±1.4	26.3±3.3	>250
	$\mathcal{DGCN}$	**10.53±1.06**	**16.03±1.05**	**59.7±1.42**
	Reference	*8.83±0.73*	*13.96±0.75*	*50.96±0.65*
WATER	Feliot et al. [2017]	48.3±3.6	80.7±5.6	139.1±8
	Couckuyt et al. [2014]	53.5±4.8	88.7±7.5	164.3±9.6
	$\mathcal{DGCN}$	**13.93±1.53**	**15.97±1.28**	**17.73±1.81**
	Reference	*12.86±0.66*	*13.06±0.77*	*14.03±0.65*

Table 3.9: Overview of the constrained multi-objective test functions results of the different approaches and the mean and standard deviation of needed samples over 30 iterations.

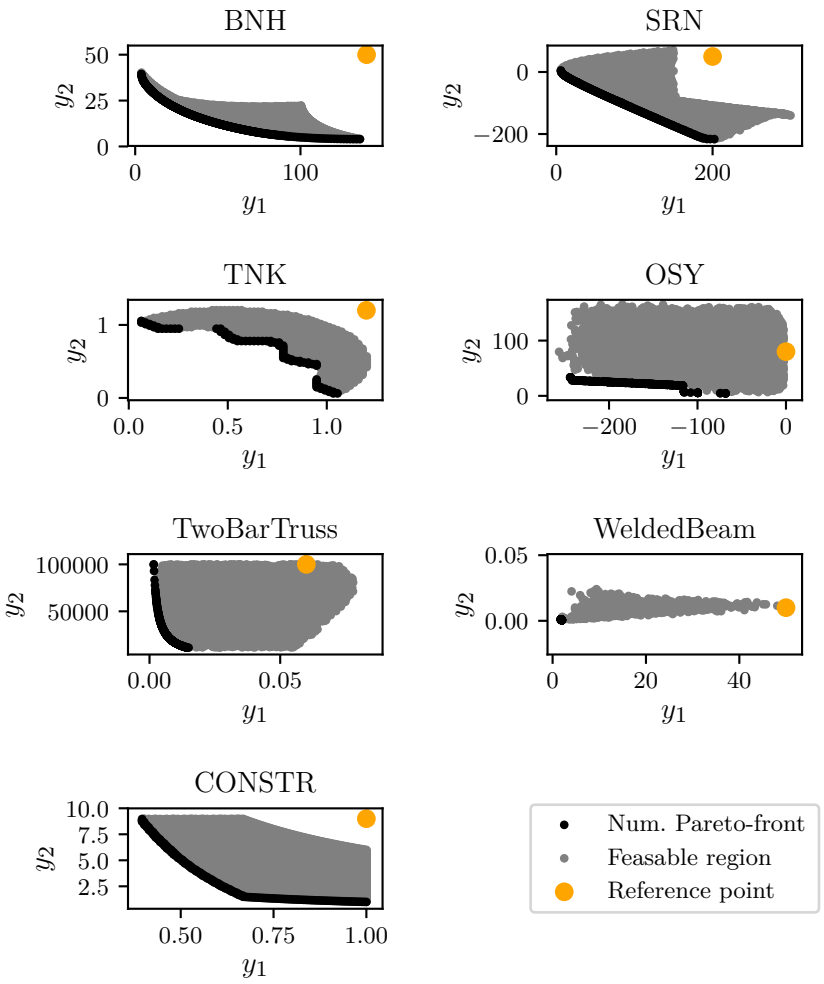


Figure 3.32: Overview of test function and the Pareto-frontiers estimated on the numerical model via optimization. The gray area illustrates the feasible domain. For the WeldedBeam example, the Pareto-front makes up only a very small part of the feasible domain.

The optimizer should find a $\boldsymbol{x}_*$ which maximizes $\hat{s}_*$ over the possible design space. Doing this step by step will lead to an overall reduction of the maximum possible $\hat{s}_*$, thus a reduction of the model uncertainty. To test this approach a $\mathcal{DGCN}$ model should be built for the three test functions namely the Eggholder function (eq. 2.9), Himmelblau function (eq. 2.8) and the modified Branin-Hoo function (eq. 2.7). For the Eggholder function, a budget of 190 additional samples is provided. For the Himmmelblau and Branin-Hoo function only 30 additional samples are allowed. The budgets were chosen so that a good representation of the design space is possible. The higher budget for the Eggholder function is because of its high variance. For all three test functions, a start LHS was created with $N = 10$ samples, optimized via the E^{CL_2} criterion. After the sampling, $1,000$ test points are used to calculate the $RMSE$ and $\hat{R}^2$ metrics. These can be used for comparison with the sequential LHS tests. Hence, the final sample sizes are also chosen so that they correspond to one of the sequential sampling steps. Fig. 3.34 to 3.36 show the results of the samplings. They show that the sampling evenly covers the whole design space and, even more interestingly,

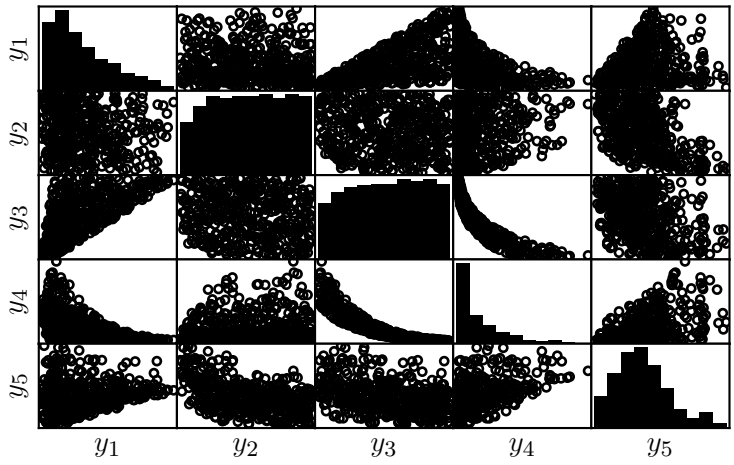


Figure 3.33: Overview of the 6 objectives WATER test function.

it seems to focus also on the parameter range limits which is especially beneficial if the model should also be used for optimization and the optimum is next to these limits as for the Eggholder function. In tab. 3.10 the results for the $RMSE$ and $\hat{R}^2$ metric are listed. Comparing tab. 3.5, 3.6 and 3.7 with tab. 3.10, it is obvious that the result of the $RMSE$ and R^2 have lower values than the best results obtained via the sequential LHS. Especially, the $RMSE$ for the Himmelblau function is about half the value. The reason for this is the sampling at the design space edges which is very important for the Himmelblau function (see fig. 2.13).

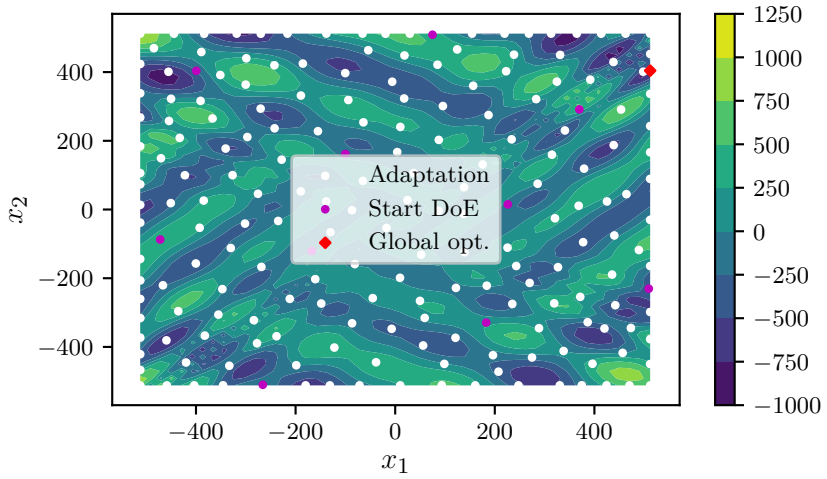


Figure 3.34: Surrogate model standard-error-based one-by-one sampling for the Eggholder test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 190 points.

After demonstrating the adaptive sampling approach via the optimization of the standard error for one dimensional problems, the question now is how to extend this for multiple

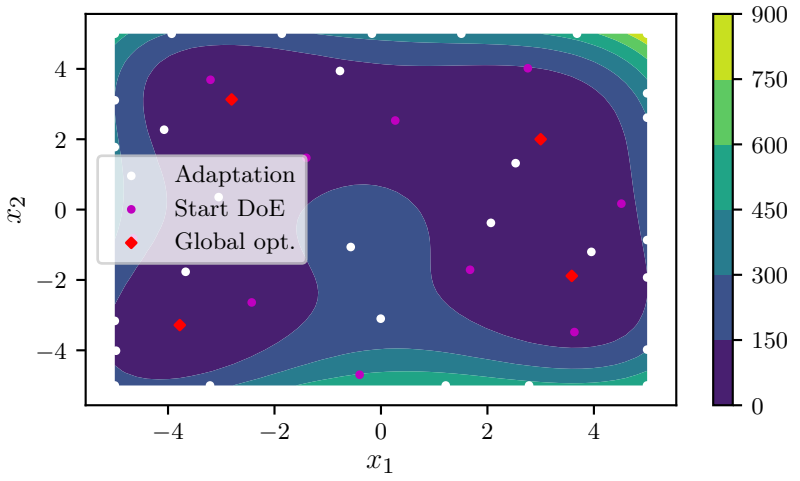


Figure 3.35: Surrogate model standard-error-based one-by-one sampling for the Himmelblau test function with a start *DoE* of $N = 10$ optimized via the E^{CL_2} criterion and additional 30 points.

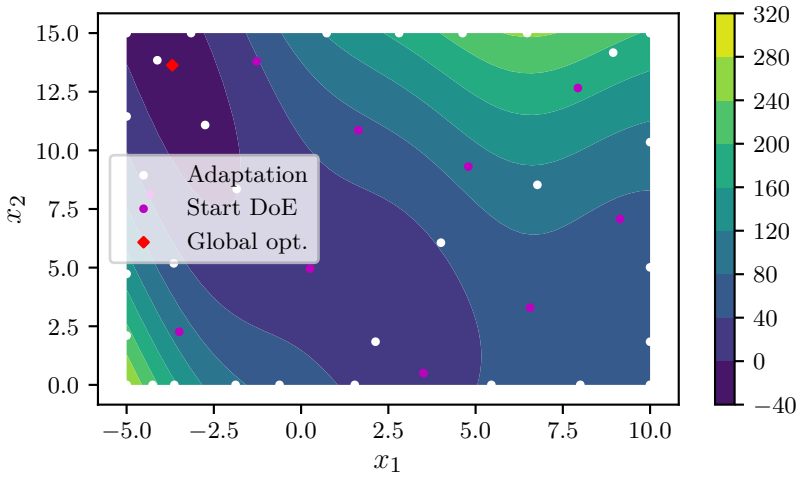


Figure 3.36: Surrogate model standard-error-based one-by-one sampling for the modified Branin-Hoo test function with a start *DoE* of $N = 10$ optimized via the E^{CL_2} criterion and additional 30 points.

Function	N	$RMSE$	$\hat{R}^2$
Eggholder	200	174	0.2
Himmelblau	40	39.11	0.83
mod. Branin-Hoo	40	1.75	1.00

Table 3.10: Overview of the $RSME$ and $\hat{R}^2$ metric for 1,000 test points for three different test functions.

outputs, since the number of outputs corresponding to a single design plan is typically much larger than one. For the sampling approach using the optimization of the standard error, a possible solution for multiple outputs would be to create an overall objective function such as the sum of the variances of all n_o outputs. In this case, it is important that the $\hat{s}_{i*}$ are scaled such that large output values are not dominated by others. It is suggested that the mean / variance normalized outputs are used, since they are needed anyway during the training of the $\mathcal{DGCN}$ or $\mathcal{GP}$ model. The new objective for optimization would be

$$E^s = \sum_{i=1}^{n_o} \hat{s}_{i*} \tag{3.40}$$

To test this approach, the three test functions used in the last experiment should be used in combination so that a single design plan is created for all three test functions. The input parameters $x_1 \in \mathbb{R}^{[-515,512]}$, $x_2 \in \mathbb{R}^{[-515,512]}$ correspond to the Eggholder function, $x_3 \in \mathbb{R}^{[-5,5]}$, $x_4 \in \mathbb{R}^{[-5,5]}$ to the Himmelblau function and $x_5 \in \mathbb{R}^{[-5,10]}$, $x_6 \in \mathbb{R}^{[0,15]}$ to the modified Branin-Hoo function. A start LHS with $N = 10$ samples optimized via the E^{CL_2} criterion is used. A budget of 50 additional samples is allowed.

The results of this test are shown in fig. 3.37. The red marked scatter plots show the important subspaces for the three functions because as mentioned before, x_1, x_2 belong to the Eggholder function, x_3, x_4 belong to the Himmelblau function and x_5, x_6 belong to the modified Branin-Hoo function. All other subspace combinations are unimportant for this example. Interestingly, the sampling approach seems to recognize this and tries to optimize the sampling for these subspaces. Also, the correlation for these is low compared to some of the other subspaces (e.g. x_1, x_3). Furthermore, it tries to cover all design space edges even in the higher dimensions. Therefore, it seems that the summarized prediction standard error is a very interesting sampling criterion for the one-by-one approach. The success of this method is dependent on the fact that the model learns something useful. If the wrong parameters are identified as being important through the correlation length-scale settings, then the sampling will not be optimized according to the correct parameters.

Finally, some research aspects shall be pointed out which are currently state of the art in the field of Bayesian Optimization but will not be discussed in detail here. One topic is the selection of several candidates simultaneously / parallel per adaptation. There are various approaches such as in González et al. [2016], Wang et al. [2016a], Nguyen et al. [2016], Nguyen et al. [2017]. A relatively simple approach would be, to update the model with one selected point, as in the one by one approach presented here, but using the model prediction for the output instead of the true output. This would cause the output to contain the model error, but the uncertainty at this point would be towards zero (dependent on the noise level) and the model would select another candidate at another position if more than one point should be found during one adaptation. This means that any number of points can be selected per adaptation. Of course, this methodology is strongly dependent on the actual model quality of the current adaptation and can not be parallelized.

Another very exciting topic is the optimization in subspaces or lower embeddings so that Bayesian Optimization can be applied efficiently for high-dimensional problems. For example, autoencoders are used as in Moriconi et al. [2019] or random projects are used as in Binois et al. [2015], Wang et al. [2016b]. Other works that would deal with this topic are Kirschner et al. [2019], Moriconi et al. [2020], Letham et al. [2020]. In all of these works, they try to find a way, to identify the correct subspace of important optimization parameters

to handle high-dimensional problems. Methods of sensitivity analysis, as presented in the next chapter 4, could be helpful for this purpose. There are already some works in this direction like Spagnol et al. [2019].

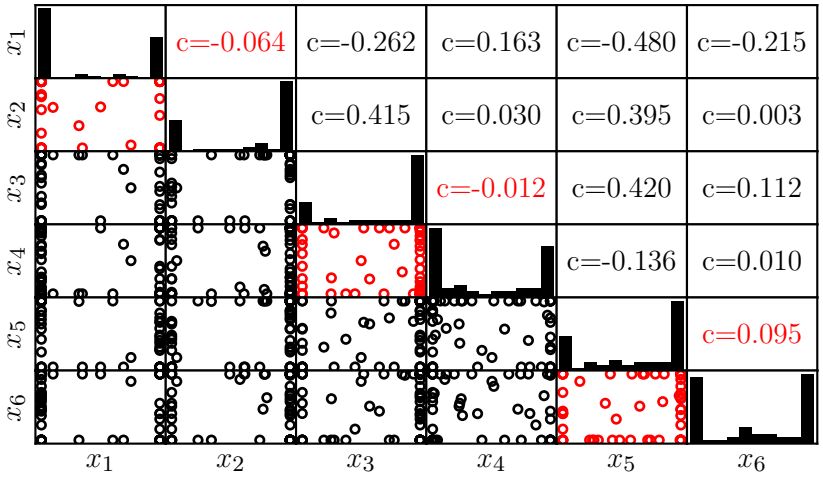


Figure 3.37: Surrogate model standard-error-based one by one sampling for the combined Eggholder, Himmelblau and modified Branin-Hoo test function with a start DoE of $N = 10$ optimized via the E^{CL_2} criterion and additional 50 points.

Important notes:

- There are several ways to perform a sequential / adaptive sampling. All of them try to explore and / or exploit the design space step by step.
- A sequential LHS ($SLHS$) method was presented which can be used as a pure exploration algorithm in order to sequentially obtain a globally accurate surrogate model.
- The $E^{ME_{cond}}$ criterion might be especially beneficial as an optimization objective for this method in combination with a $\mathcal{GP}$-based model, since it incorporates the learning of possible unimportant parameters into the design plan optimization. This is a topic for further investigations.
- Since the $SLHS$ approach adds points with the highest distance to existing points, it also reduces the global uncertainty of $\mathcal{GP}$-based models.
- One drawback of this method is the need for doubling the existing number of samples.
- Besides the sequential LHS approach, one-by-one adaptation methods were also presented such as the expected-(hypervolume)-improvement-based (EI / EI_{hyp}) algorithms or the novel variance-based one.
- This method provides one of the most efficient ways of sampling, either for optimization purposes or globally accurate models.

- The advantage of $\mathcal{DGCN}$ having non-stationary covariance functions showed partially extreme performance gains for EI_{hyp}-based optimization benchmark. Since the correlation length-scale parameter can adapt to locally refined sampling areas.
- The optimization based on the E^s criterion showed that it is also possible to obtain a good space-filling design plan with a one-by-one adaptation. It can even also incorporate the influence of the most important parameters into the positioning of future adaptations so that important subspaces are preferably covered (see fig. 3.37).

3.5 SAMPLING RECONSTRUCTION

In probabilistic assessments, for example via Monte Carlo sampling or sensitivity analysis (the global variance-based method Saltelli et al. [2008], see section 4.4), the data sampling must follow specific uni- or multivariate distribution (the joint density function) and correlation types. These can be specified by the user or might be derived from measurements. In the first case, it is often easy to create the data appropriately, since the correlation can be specified for example by correlation coefficients like the Pearson coefficient (Pearson [1895], see section 3.3) or by analytical functions. Also, the distribution can be specified via one of the well-known function types like the exponential or normal distribution. In the second case, where the data structure is set by measurements, it is much more difficult to reproduce the multivariate distribution and correlation because they might not fit to any known analytical distribution type or correlation coefficient. Therefore, this section will demonstrate how to transform a uniform distribution to any other known or even unknown distribution type via inverse transform sampling. Additionally, it will also present how to reproduce any kind of uni- or multivariate correlation between input parameters with the help of a specific category of unsupervised machine learning algorithm, also known as novelty detection.

3.5.1 *Inverse Transform Sampling*

The inverse transform sampling method is an analytical way of transforming a uniformly distributed parameter $y \sim \mathcal{U}(a = 0, b = 1)$[2] to any other distribution type for which the inverse of the cumulative density function (CDF) $F(x)$, the quantile function $F^{-}1(x)$ (also known as the percent point function, PPF), is known. To illustrate how this method works, $x \sim \mathcal{U}(0, 1)$ will be transformed to an exponential distribution type with its probability density function $f_{pdf}(x) = x \sim \mathcal{E}(\lambda) = \lambda \exp(-\lambda x)$, its CDF $y = F(x) = 1 - \exp(-\lambda x)$ and its inverse function $x = F^{-1}(y) = \frac{-\ln(1-y)}{\lambda}$. Formally, the transformation $x \sim \mathcal{U} \rightarrow x \sim \mathcal{E}$ is performed by

$$x \sim \mathcal{E} = F^{-1}\left(x \sim \mathcal{U}\right) = \frac{-\ln\left(1 - x \sim \mathcal{U}\right)}{\lambda} \tag{3.41}$$

2 a, b describe the lower and upper bound of the uniform distribution.

which is also illustrated in fig. 3.38 and 3.39. This method can also be used in combination with a uniformly optimized LHS as a start solution which is recommended for example if a Monte Carlo sampling will be used for probabilistic analysis as shown in fig. 3.40. This method captures the multivariate distribution of multiple parameters by simply applying this method for each parameter separately.

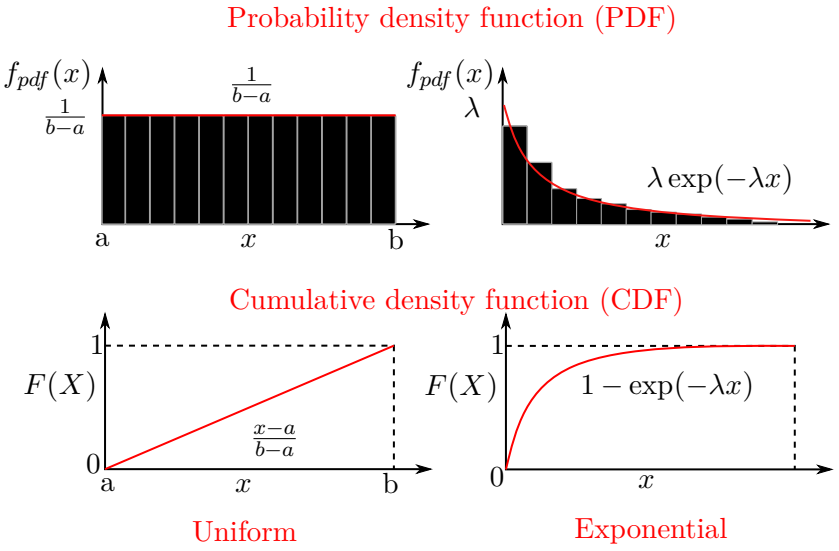


Figure 3.38: Schematic overview of the uniform and exponential distribution type.

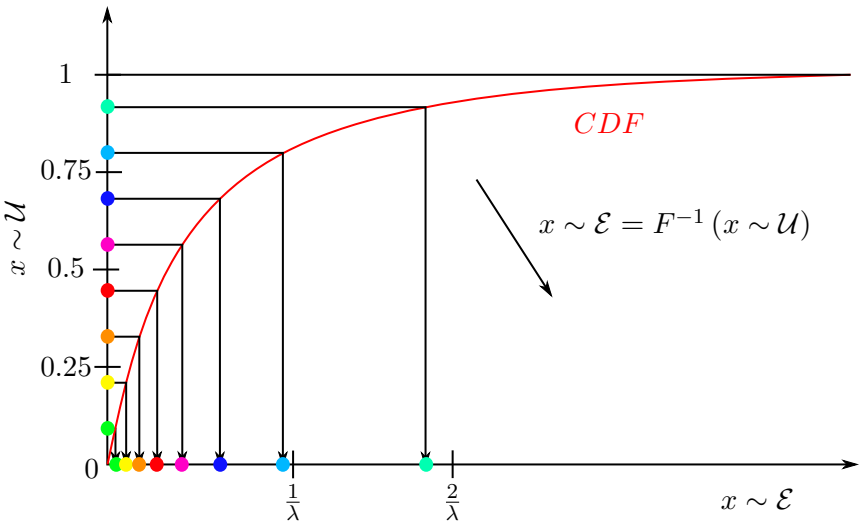


Figure 3.39: Illustration of the inverse transform sampling method.

The next step is to also use this method for distribution types with an unknown CDF and PPF. To do this, the histogram of the measured data will be utilized to build an approximated PPF. Therefore, it is first necessary to calculate the cumulative histogram based on the measured data. To obtain the CDF in the first place, it is possible to simply use the bin edges information along with the height of each bar which represents the cumulative probability for a specific x value, to fit for example a spline interpolation model. The right choice of classes (or bins) n_{bin} for the histogram is important to obtain a good representation of the data. Since the PPF is needed instead of the CDF, the approximation

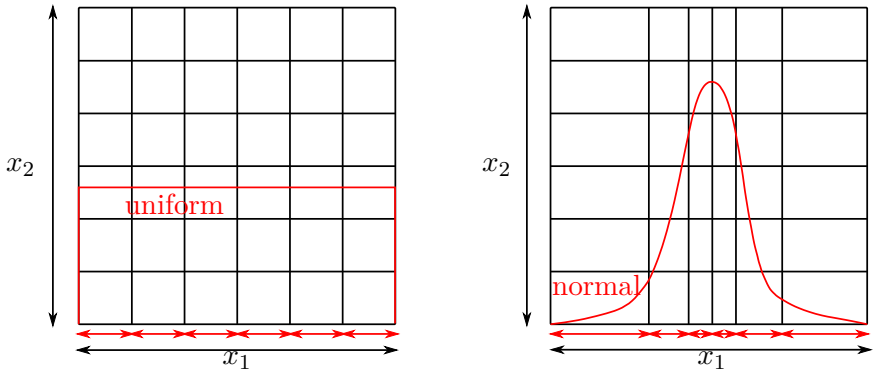


Figure 3.40: Illustration of the inverse transform sampling method for a LHS from uniform to normal distribution.

function (e.g. the spline) can simply be trained with switched values for the cumulative probabilities and the bin positions. So, the bin positions (x values) will be the output and the cumulative probabilities the input of the approximated function. This procedure is illustrated in fig. 3.41.

To show this method working on real data coming from steam turbine measurements with $n = 5$ parameters, fig. 3.42 demonstrates the data which will be resampled (top image) and 500 points resampled (bottom image). It can be seen that the distributions of the resampled points is very similar to the original samples for all input parameters. It can also be seen that the correlations, and therefore the position of the resampled points, do not match the measured data. Obviously the correlations are not adequately described by a linear correlation coefficient. Hence, this is a challenging example for the resampling process. If the correlations would be linear, the correlations in the resampling could be set accordingly with the methods presented in section 3.3. The measured data correlations are clearly non-linear and very strong. Nevertheless, the problem of setting the correlations correctly by even keeping the multivariate distribution, can be solved with a method of resampling the multivariate joint distribution function including the correlations between the input variables which is illustrated in the following.

3.5.2 *Novelty-Detection-based Rejection Sampling*

The problem of the previous example is to reproduce a multivariate correlation between several parameters which cannot be characterized in a simple fashion by a correlation coefficient or an analytical correlation function. To solve this problem, a specific category of unsupervised machine learning algorithm, also known as novelty detection, (see Markou and Singh [2003a]) will be utilized. Novelty detection algorithms are very similar to outlier detection methods, with the difference that they are trained with samples which are not polluted by outliers. After the training, the novelty detection model should be utilized to decide if new points are inliers or outliers. The differences are:

- Novelty detection: The training data is not polluted by outliers, and the user is interested in detecting anomalies in new observations.
- Outlier detection: The training data contains outliers, and the user wants to remove them.

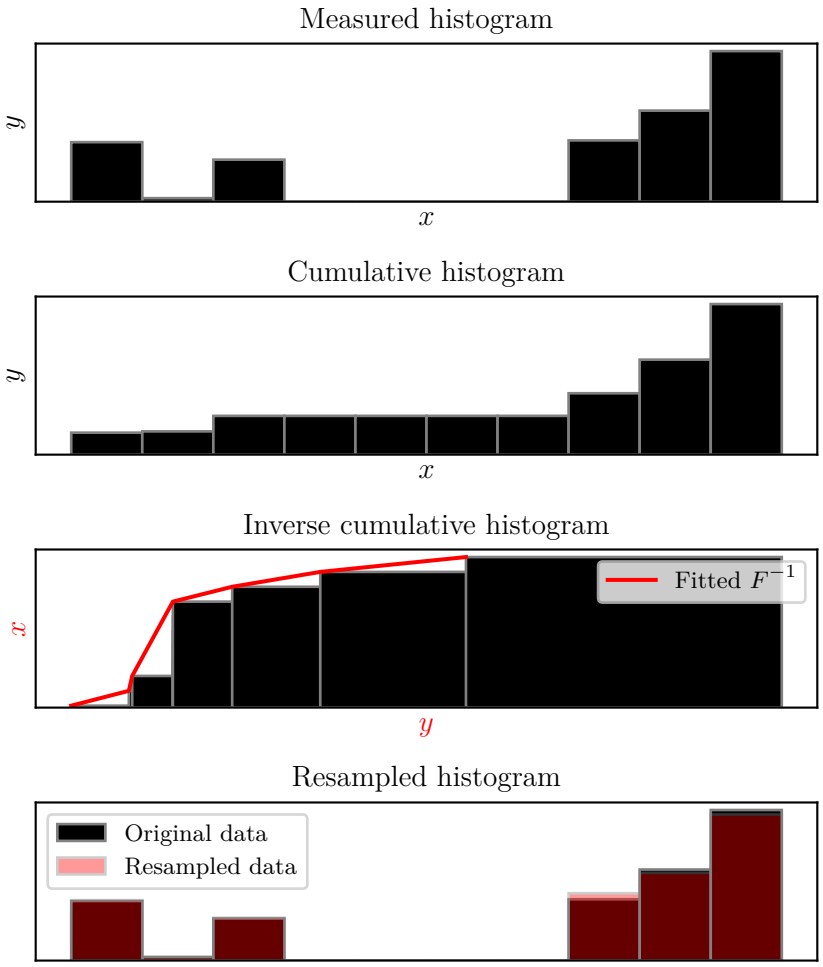


Figure 3.41: Stepwise inverse transform sampling for unknown CDF and PPF. 233 measured samples and $1,000$ reconstructed samples.

There are many applications for outlier / novelty detection:

- Improve surrogate model accuracy by cleaning data sets (outlier detection).
- Improve sensitivity analysis by removing faulty designs (outlier detection).
- Detection of anomalies in a running system e.g. a production line (novelty detection).
- Resample from highly correlated measured data (novelty detection).

For both of these algorithm categories there are several algorithms, for example:

Outlier detection methods:

- Elliptic envelope (assumes Gaussian distributed data, covariance based), see Rousseeuw and Driessen [1999].
- Isolation forest (based on random forest, good for high dimensional data), see Liu et al. [2008].

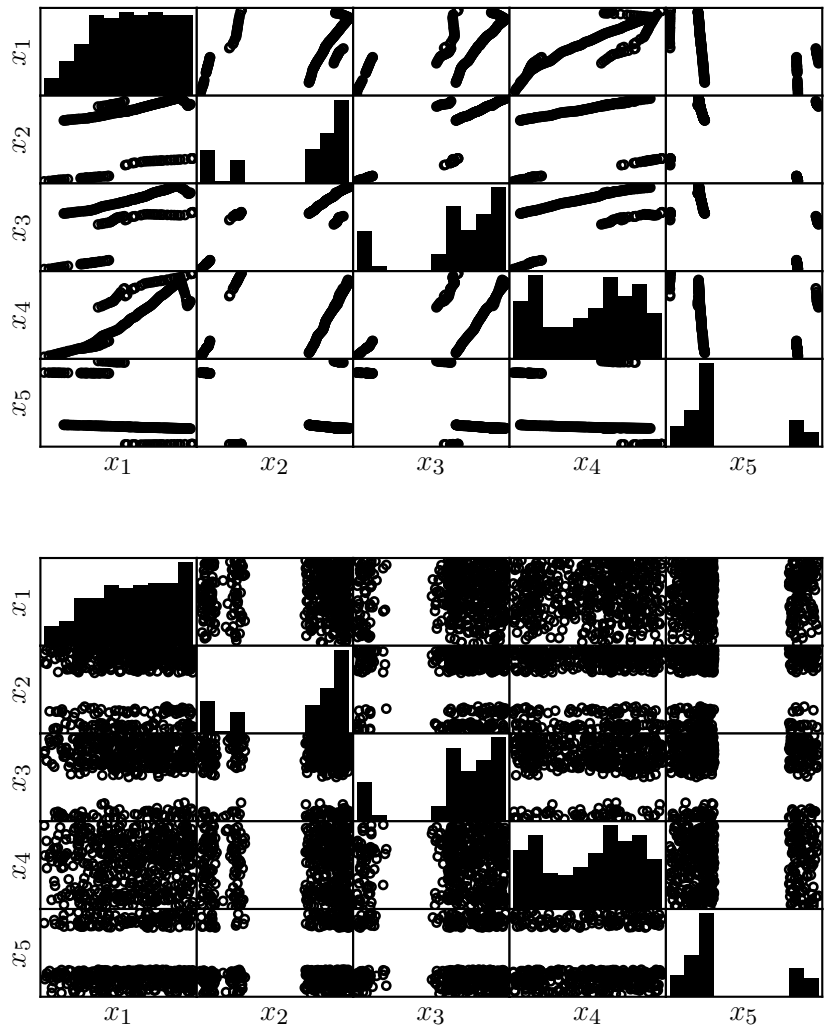


Figure 3.42: Measured operational data from a steam turbine (top) and 500 new samples with the same multivariate distribution (bottom).

- Local outlier factor (based on local density), see Breunig et al. [2000].

Novelty detection methods

- Support vector machine (works well if data is well-separated), see Li et al. [2003].
- Neural networks, see Markou and Singh [2003b].
- Nearest neighbors, see Aghazadeh et al. [2011].
- Basically any classification algorithm.

A trained novelty detection algorithm is typically capable of predicting the probability that a point is a outlier or not. This is not only useful to check if newly created data points are in- or outliers but can also be used for optimization tasks based on surrogate models as a constraint to avoid extrapolation. It is even possible to visualize the decision

boundaries of the trained novelty detection algorithm. Fig. 3.43 shows an example for two input parameters. The novelty detection algorithm used is a one class support vector machine (SVM) (Schölkopf et al. [2001]).

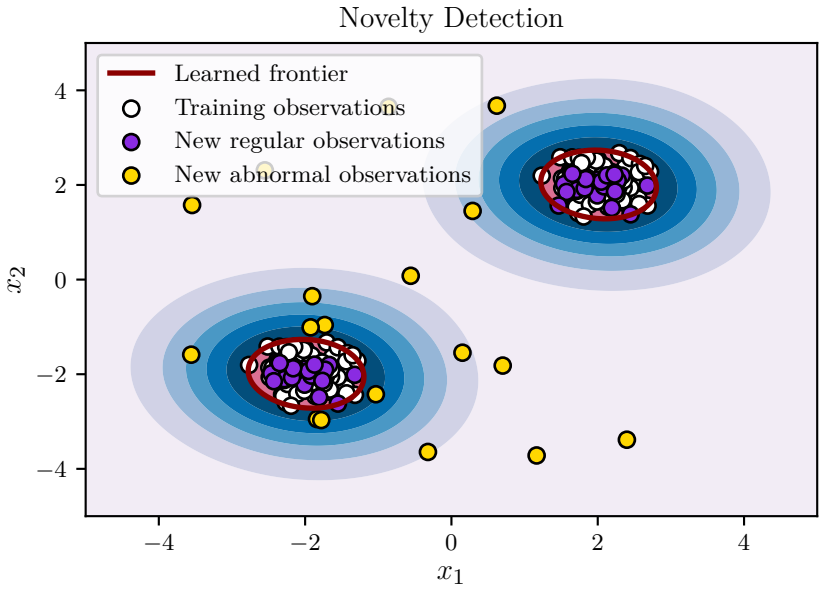


Figure 3.43: Demonstration of novelty detection on new observations and the learned decision boundaries for two parameters x_1, x_2. The used algorithm is a one class support vector machine.

Instead of the one class SVM, the novelty algorithm presented in Quinn and Sugiyama [2014], called Least Squares Anomaly Detection ($LSAD$) will be used for the further investigations. It is a very flexible and fast evaluating novelty detection algorithm. The problem with most novelty detection algorithms is in setting the hyperparameters correctly which control the behavior of the decision boundaries. The $LSAD$ algorithm is controlled mainly by two parameters: θ a kernel length-scale similar to the Gaussian process length-scale, controlling how smooth the result will be and ρ a regularization parameter which controls the sensitivity to outliers. The influence of these parameters is illustrated in fig. 3.44. The right choice of these parameters is crucial to obtaining the right multivariate correlations. As can be seen in the figure, a high value of θ as well as a high value of ρ lead to a wider decision boundary. Therefore, an optimization procedure was developed, where the $LSAD$ algorithm was trained in such a way that it detects all training data as inliers and should minimize these two parameters. The resulting parameters of this optimization automatically provides the decision boundaries which are very close to the training data. It is helpful for the algorithm to normalize the data to have zero mean and variance of one.

To solve the problem of the multivariate correlation visualized in fig. 3.42, the idea is to train the $LSAD$ algorithm on the measured data (training data) with the proposed optimization procedure and then use it to predict if the points created by the inverse transform sampling method are in- or outliers. The result of this procedure is shown in fig. 3.45 for the already presented steam turbine data example.

As can be seen, the multivariate correlations are reconstructed very well, even for such a difficult example but the multivariate distribution is no longer similar to the original one

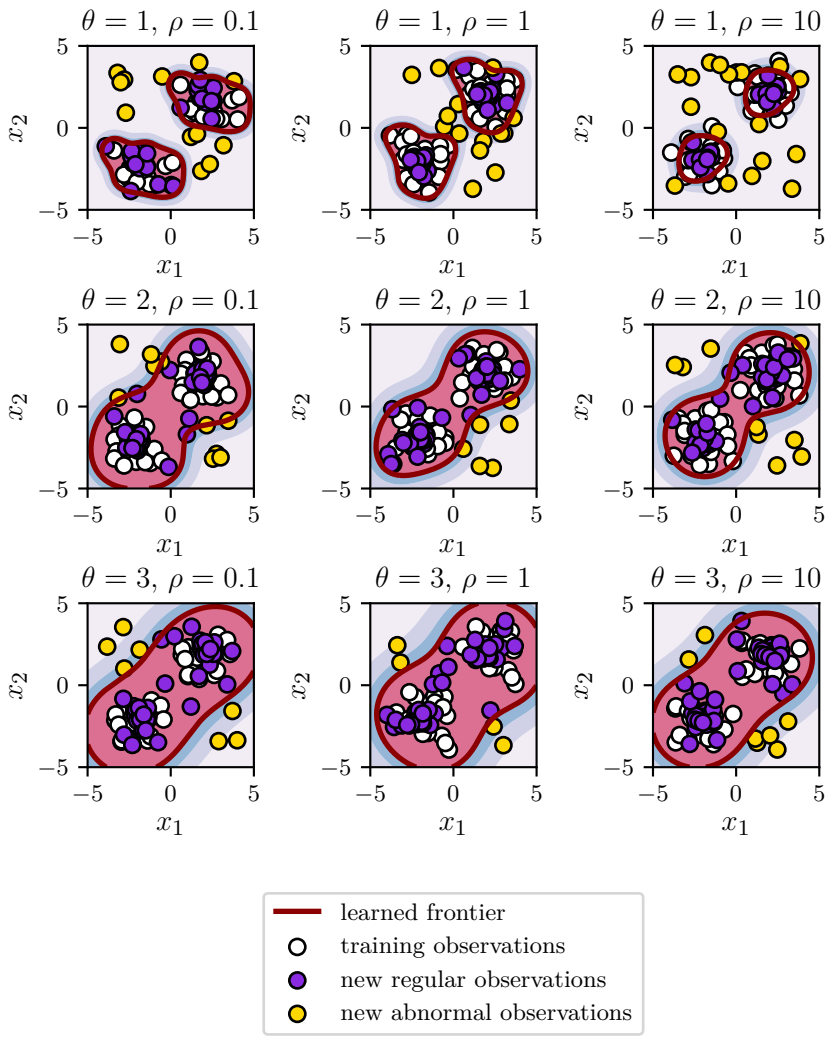


Figure 3.44: Influence of the parameter changes θ and ρ on the *LSAD* algorithm.

due to the reduction of the number of sample points. Therefore, a further algorithm was developed which is a combination of all methods for data reconstruction presented so far. The algorithm is presented in alg. 2. The function $getBins()$ is a function which gives the number of classes for the histogram dependent on the number of samples N_{train} in the training data. The function $hist()$ returns the number of entries per bin, input parameter-wise, based on the histogram with n_{bin} classes. The function $create_sampling()$ creates a sampling via inverse transform sampling based on the training data and a consecutive usage of a trained novelty detection algorithm cl to discard points which do not follow the multivariate correlations. The basic idea of this algorithm is to compare the distribution of points in the histogram bins between the reconstructed sampling and the training data. If both samplings have an equal number of points in each bin (variable-wise), then both also share the same multivariate distribution. This approach can also be seen as a rejection-sampling-based algorithm (Gilks and Wild [1992]). Since often many more points will be

created than training points are available, the bin counts are multiplied to be a multiple of the initial bin count. The concrete steps for this algorithm then are:

1. Find a sampling number c which is higher than a specified minimum number of samples N_{min}, which is by default 20% of the target sample count N_{tar} (the number of samples which will be created with the algorithm). The objective is to find a sampling count which does not fill all bins of the target histogram but also provides more than one sample to be numerically more efficient in the following steps. The reason for this step is that the *create_sampling*() function cannot guarantee to create c or N_{min} samples because of the discarded samples through the novelty detection algorithm cl. In some applications, c is much higher then N_{tar} because many points are discarded in each *create_sampling*() call.

2. The next step is to add one point at a time to the initial X_{recon} which is the reconstructed sampling to be returned at the end, until it has a nearly equal count of points in each bin compared to the training data or a multiple of it. The multiple is calculated in line 8 of alg. 2. It might also be the case that the maximum number of iterations is exceeded, or no further improvement is possible after 10 steps per default or the target bin count difference n_{tbcd} is reached. The objective of this algorithm is the absolute difference of points per bin between the training and reconstructed sampling summed over all bins over all input variables. Therefore, the histograms should be as close to the histogram of the training samples as possible.

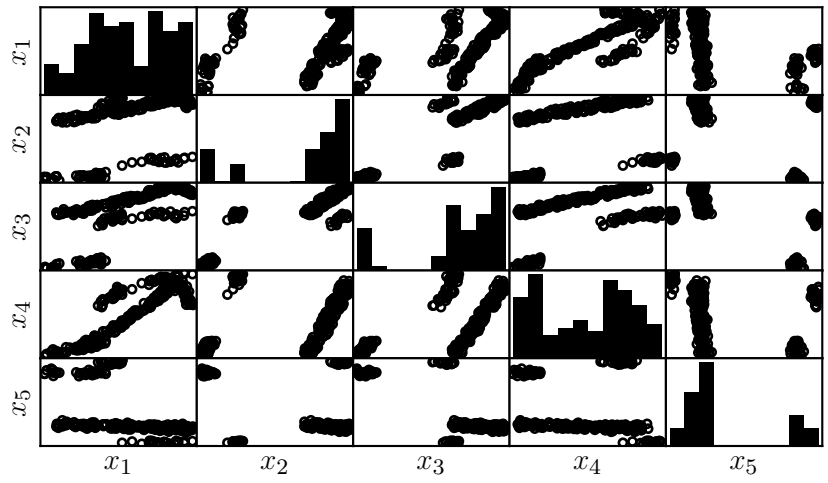


Figure 3.45: Results on using the *LSAD* novelty detection algorithm to reconstruct the multivariate correlations between all parameters. $N = 260$ samples are shown.

The result of this algorithm applied on the test example of steam turbine data is illustrated in fig. 3.46. Compared to fig. 3.42 it can be seen that the usage of the bin optimization algorithm helps a lot to correct the multivariate distribution by keeping also the multivariate correlations. Fig. 3.47 presents another example of a measured industrial data set from the field of electric motors. It shows that this algorithm also works if the data is clearly separated into clusters.

Algorithm 2: Rejection sampling algorithm.

Data: cl, X_{train}, N_{train}, N_{tar}.
Result: X_{recon} the reconstructed sampling.

begin
  $n_{bin} \leftarrow getBins(N_{train})$
  $n_{tbcd} \leftarrow 0$
  $I_{max} \leftarrow 1,000$
  $N_{min} \leftarrow round(N_{tar} \cdot 0.2)$
  $n_{bbcd} \leftarrow 1e66$
  $n_{bcTrain} \leftarrow hist(X_{train}, n_{bin})$
  $n_{bcTrain} \leftarrow n_{bct} \cdot round(N_{tar}/N_{train})$
  $I \leftarrow 0$
  $c \leftarrow N_{min}$
  $N_{tmp} \leftarrow 0$
  $no_improve_c \leftarrow 0$
  while $N_{tmp} < N_{min}$ **do**
    $X_{tmp}, N_{tmp} \leftarrow create_sampling(X_{train}, c, cl)$
    $c \leftarrow 2c$
  $X_{recon} \leftarrow X_{tmp}$
  while $n_{bbcd} > n_{tbcd}$ ***and*** $I_{max} > I$ ***and*** $no_improv_c < 10$ **do**
    $no_improve \leftarrow True$
    $X_{tmp} \leftarrow create_sampling(X_{train}, c, cl)$
    for $j \leftarrow 0$ **to** $len(X_{tmp})$ **do**
      $X_{test} \leftarrow concatenate(X_{recon}, X_{tmp}[j, :])$
      $n_{bcTest} \leftarrow hist(X_{test}, n_{bin})$
      $n_{tmpbcd} \leftarrow sum(abs(n_{bcTrain} - n_{bcTest}))$
      if $n_{tmpbcd} < n_{bbcd}$ **then**
        $X_{recon} \leftarrow X_{test}$
        $n_{bbcd} \leftarrow n_{tmpbcd}$
        $no_improve \leftarrow False$
    if $no_improve = False$ **then**
      $no_improve_c \leftarrow 0$
    else
      $no_improve_c \leftarrow no_improve_c + 1$
    $I \leftarrow I + 1$

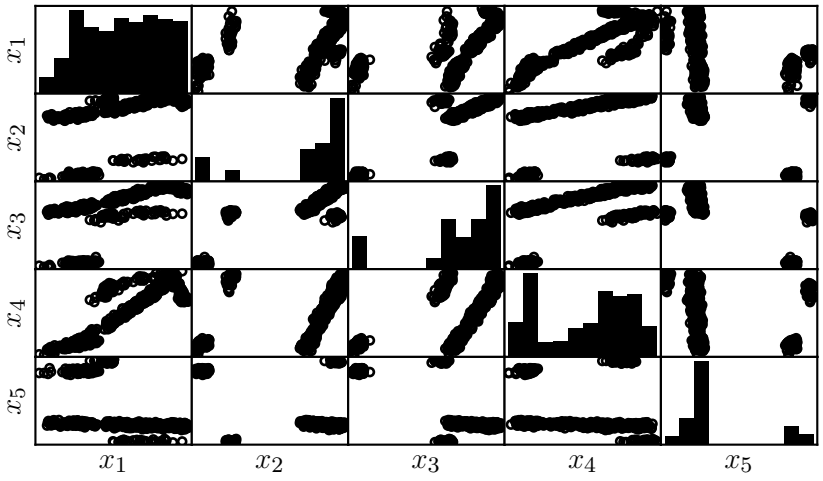


Figure 3.46: Results on using the *LSAD* novelty detection algorithm to reconstruct the multivariate correlations between all parameters with the usage of alg. 2. $N = 534$ samples are shown.

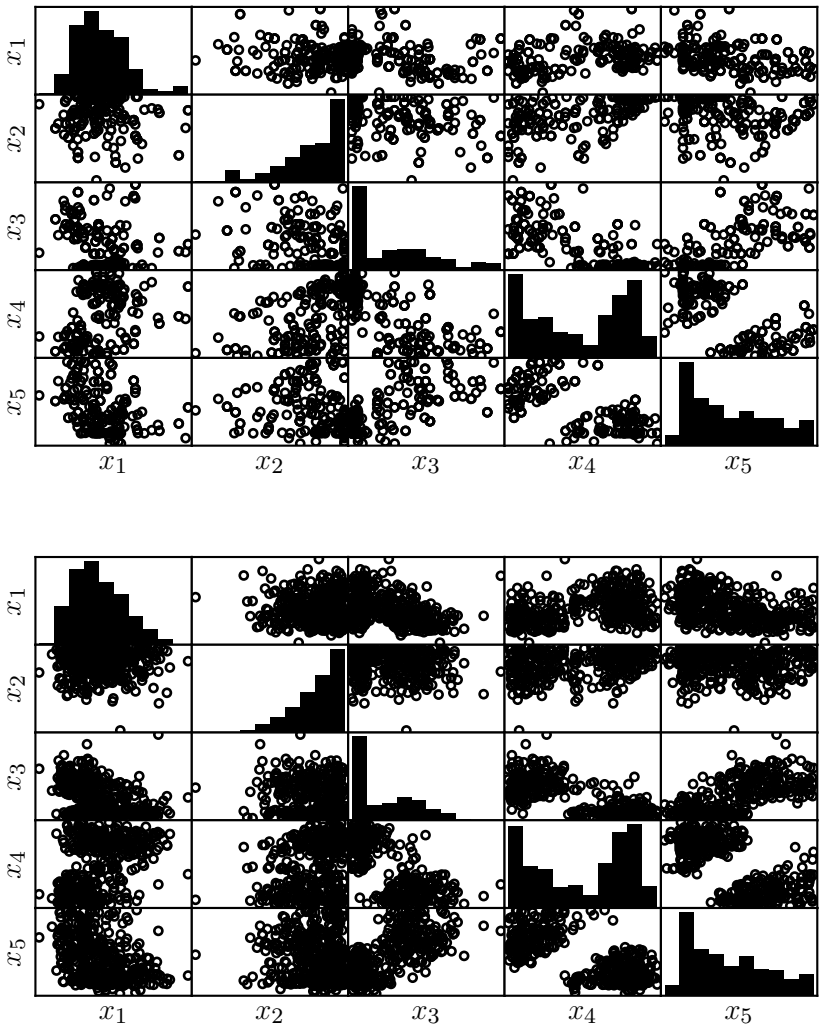


Figure 3.47: Further example of the reconstruction algorithm on an industrial example of measured data. Top shows the measured data with $N = 200$ and bottom the reconstructed data with $N = 425$.

Not mentioned in this work yet, is another quite promising approach to replicate the measured distribution and correlations of a set of input parameters. This approach is called kernel density estimation (KDE) (Scott [2015]). KDE is a non-parametric method to estimate the probability density function of a random variable. Let (x_1, x_2, ..., x_n) be a univariate independent and identically distributed sample drawn from some distribution with an unknown probability density function f_{pdf}. In this case, its kernel density estimator is:

$$\hat{f}_{pdf}(x) = \frac{1}{nh} \sum_{i=1}^{n} k\left(\frac{x - x_i}{h}\right) \tag{3.42}$$

where k is the kernel or covariance function, a non-negative function similar to the Gaussian process covariance function, and $h > 0$ is a smoothing parameter called the bandwidth, also similar to the correlation length-scale l of the $\mathcal{GP}$. It controls how close the resampled samples will be to the training data. Intuitively, one wants to choose h as small as possible; however, there is always a trade-off between the bias of the estimator and its variance. If the amount of training samples is low, a too small bandwidth parameter can lead to a resampling missing some of the possible design space. Several works are available on how to choose these bandwidth parameter (Sheather and Jones [1991], Terrell and Scott [1992], Turlach [1993]). Fig. 3.48 shows the same examples already presented resampled with the KDE method.

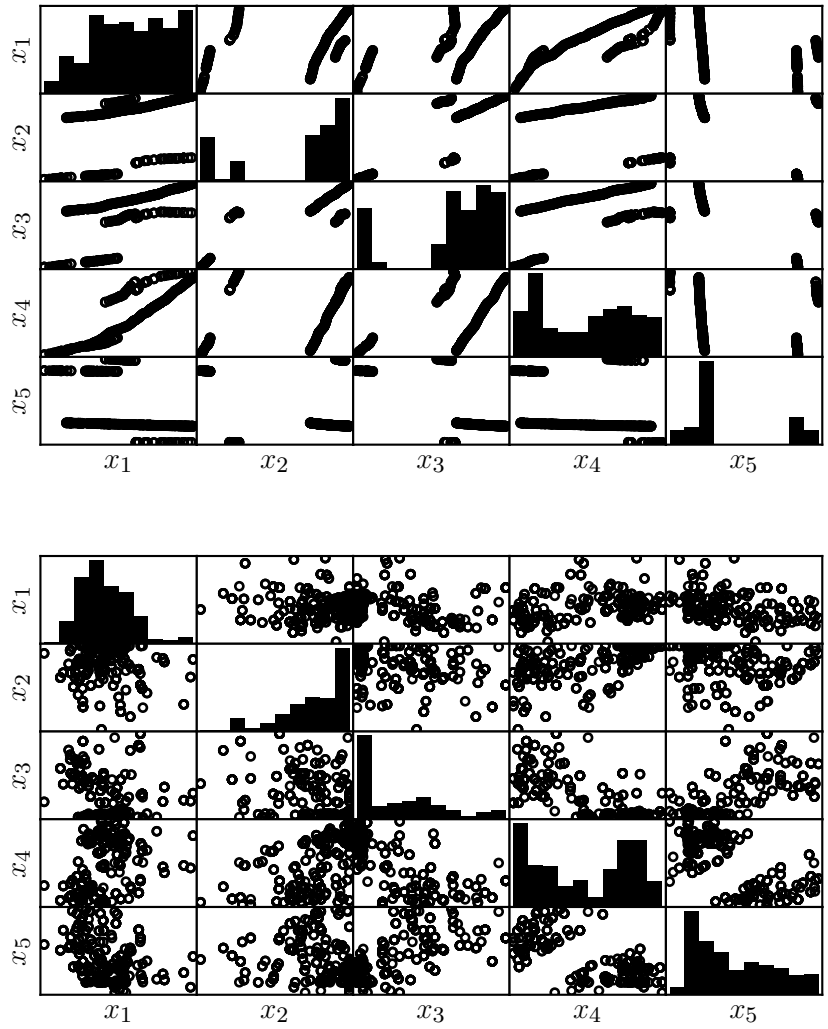


Figure 3.48: KDE used to resample the two examples of industrial data presented. The reconstructed data has $N = 500$ samples.

It turns out for these examples that the bandwidth parameters have to be set very small in order to capture the correlations correctly which leads to the fact that the resampled points are nearly identical to the training data points with a very small variation. This can also be seen in the comparison of the figs 3.47 and fig 3.48. The first one shows a more distributed setting of new points compared to the KDE solution. A further advantage of the self-developed rejection-sampling-based method over the KDE method will be shown in section 4.4, namely the opportunity to resample data points if one or more values of the variables are already given and the remaining variables must be sampled in order to represent the correct multivariate correlations and distribution. For example two parameters x_1 and x_2 have a joint distribution and are correlated and the values for x_2 are already given for the resampling (like a constraint). Then the values for x_1 must be sampled in the resampling process according to the joint distribution and correlation for the given values of x_2.

Important notes:

- Inverse transform sampling is a method to reconstruct any kind of distribution type if the inverse of the CDF, the PPF, is known.
- One way to approximate the PPF is to utilize the cumulative histogram of the measured data. A spline function can be fitted to the cumulative probabilities and the bin positions. So, the bin positions (x values) will be the output and the cumulative probabilities the input of the spline approximation.
- If every variable in X is sampled according to its correct distribution then also the multivariate distribution is correct.
- The inverse transform sampling method does not capture the multivariate correlations between the input parameters.
- Novelty detection algorithms can be used to discard points which do not follow the multivariate correlations. This allows the points from the inverse transform sampling to be filtered so that they also follow the multivariate correlations of the input variables. This method can also be used e.g. as a constraint within an optimization.
- The discarding of points through the novelty detection algorithm destroys the reconstructed multivariate distribution of the inverse transform sampling.
- Therefore, a new rejection sampling algorithm was proposed which includes the approximated inverse transform sampling, the novelty detection and a bin post-processing (to correct the destroyed multivariate distribution) to reconstruct any kind of multivariate distribution and correlations.
- As an alternative approach, the KDE method was briefly presented which can also be used for data reconstruction.

3.6 CONCLUSION

This chapter presented in its first four sections an overview of different sampling strategies, starting from so called one-shot algorithm via a sequential LHS approach to finally the very efficient one-by-one adaptation algorithms. All of them have their application areas dependent on the problem to be solved and the available resources. The expected-improvement-based one-by-one adaptation can also be used for Bayesian optimization as demonstrated.

The MCS / LHS algorithm, can be used to create pre-defined samples of fixed size, provided that a globally accurate model needs to be created and efficiency does not play a big role. The LHS should always be favored over the MCS, since the restriction of one parameter value per bin typically allows a higher information gain (better coverage of the design space) and avoids clustering of points. As presented, there are several different objective functions to optimize a LHS in order to increase its space filling properties or to reduce unwanted input parameter correlations. It is even possible to optimize the LHS so that it represents a user-specified correlation matrix. The investigations made in these

sections showed that most of the objective functions presented deliver similar results if used to create a sampling for surrogate model training. As already mentioned, the E^{CL_2} and the $E^{ME_{cond}}$ criterion might be preferable due to their ability to also optimize the (important) parameter subspaces. The systematic design plans, also mentioned, should be avoided generally, since the dependency on the number of input parameters n and levels l as well as their collapsing behavior if parameters are removed after design plan creation, make these very disadvantageous to the stochastic design plans.

One drawback of the stochastic design plan methods is their limitation to be extended. Therefore, the sequential LHS approach was presented which allows the sample count to be extended by doubling in such a way that it still keeps the properties of a LHS. This method is only efficient if the starting sample size is very small. Nevertheless, it can also be used to check the convergence behavior of the surrogate model e.g. via the $\hat{R}^2$ metric which cannot be done with the one-shot approaches.

Maximum efficiency can be achieved by using one of the one-by-one adaptation methods presented. If the task is to solve an optimization problem directly, the expected improvement (single objective) or the expected hypervolume improvement (multi-objective) methods are recommended methods for the sampling, since the sampling points are placed towards the optimum of interest. If the task is to create a globally accurate model the proposed method to sample point creation in the area of highest uncertainty is one efficient way to do this. For the one-by-one approaches, the LHS one-shot methods presented are always useful for the first start DoE.

In section 3.5 a different topic was presented, namely, how to sample data points in order to have the same multivariate distribution and correlations as a set of measured data. This is especially useful for sensitivity analysis as presented in the next chapter and robustness or reliability methods. The inverse transform sampling method was presented as a tool which can be used to transform a uniformly distributed variable in the unit space $x \sim \mathcal{U}(0,1)$ to any other distribution type if the inverse of the CDF, the PPF, is known. Since very often for measured data, the distribution type does not fit to any of the known distribution types, a simple way to approximate the PPF based on the data histogram was presented. Unfortunately, this method only allows resampling from the same (multi-)variate distribution but does not also copy the multivariate correlations between parameters. Thus, novelty detection algorithms were introduced, a type of unsupervised machine learning algorithm. With these algorithms, it is possible to learn the multi-dimensional parameter boundaries of the training data points so that the algorithm can predict if points lie inside or outside of these. In combination with the created points of the inverse transform sampling method, points lying outside the learned boundaries are discarded. This allows the reconstruction of the multivariate correlations but also impairs the multivariate distribution. Hence, as a final step, a rejection sampling algorithm was proposed which includes all the methods mentioned into one framework and allows data to be sampled similarly to arbitrarily measured data points regarding their multivariate distribution and correlations.

4

SENSITIVITY ANALYSIS

4.1 INTRODUCTION

This chapter provides an overview of sensitivity analysis (SA) methods to estimate the importance ranking of input parameters. These methods can give a deeper insight into the relationships between the input parameters and the target output which might be very important for many applications. These methods can also be used to estimate the main drivers influencing the robustness and reliability of a product if used together with appropriate methods. Also, for surrogate modeling, the reduction of unimportant features can give a boost to the predictive power of the model, since a design space with a reduce dimensionality is easier to model. For example, 100 parameters might constitute too many features for a set with only 100 samples to train a good surrogate model, but if only 10 of the parameters are really important, then 100 samples might be sufficient. Some machine learning algorithms can reduce the influence of unimportant parameters by themselves, like the automatic relevance detection of Gaussian processes (subsection 2.8.7) or the anisotropic moving least squares algorithm (subsection 2.7.1) presented. Also, neural networks are able to reduce the influence of unimportant parameters by setting the corresponding weights and bias terms to zero. Nevertheless, a pre-processing step to reduce the unimportant features can also improve the training of these kinds of algorithms and speed up the convergence of training.

In this work, the relationship between one or multiple input parameters and one output parameter will be categorized as (see also illustrations in fig. 4.1 and 4.2):

- Linear or non-linear relationships.
- Monotonic or non-monotonic relationships. If x, the input parameter, varies then y, the output parameter, varies only in a positive or negative direction which would be the monotonic case. If the variation of y can be positive and negative over the design space of x then it is called a non-monotonic relationship.
- Univariate or multivariate relationships. Multivariate means that one parameter alone influences the output only in combination with one or more additional parameters. An example would be $y = x_1 x_2$, so that y depends on the product of x_1 and x_2.

This categorization will help to differentiate the methods and their capabilities. All mixtures of these categories are theoretically possible.

In section 4.2, the chapter will start with some frequently used types of correlation coefficients which were also presented partially in chapter 3. Correlation coefficients are typically able to capture linear and monotonic relationships and thus are only useful to a limited extent (there are some exceptions which will be presented in section 4.2). They can be applied directly on the data without the help of a surrogate model or expensive further design evaluations and therefore can be used as a pre-selection of important parameters. In contrast, section 4.3 and 4.4 present methods which are able to capture all kinds of

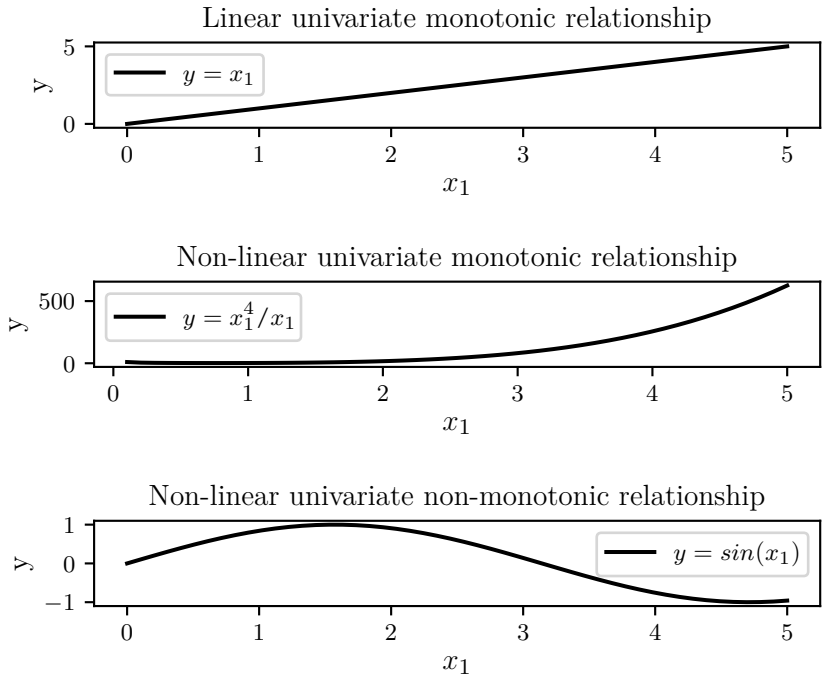


Figure 4.1: Illustration of the different univariate relationship types between a single input and a single output.

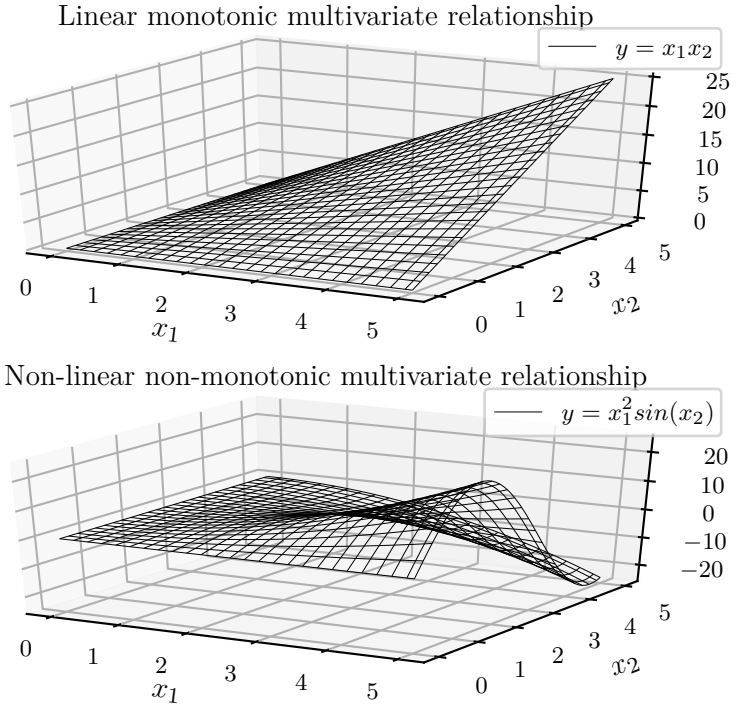


Figure 4.2: Illustration of the different multivariate relationship types between two input parameters and a single output.

relationships but need the help of a surrogate model to be calculated or at least to be calculated efficiently. Hence, these kinds of methods only work if the surrogate model used is reliable. Also, these methods are only useful to a limited extent to remove the feature before the surrogate model training so that it helps to improve the predictive power of the model. Section 4.5 shows a method of sensitivity analysis which is based on local gradients and can be applied on field outputs, for example like FEM node results. They are very useful to visualize sensitivities on geometries. Finally, a conclusion is made in section 4.6. This chapter does not claim to be complete. There are many more methods to analyze the importance of input parameters (Saltelli et al. [2008]).

4.2 TYPES OF CORRELATION COEFFICIENTS

4.2.1 *Overview*

There are several different correlation coefficient types and some of them will be briefly presented in the following. Note that correlation coefficients are, in most cases, only able to analyze univariate correlations.

Pearson correlation coefficient ($PCC \in \mathbb{R}^{[-1,1]}$) (Pearson [1895]) was already explained in subsection 3.3.2 of chapter 3 and is one of the most frequently used coefficients to explain linear monotonic relationships. Its calculation formula was given in eq. 3.8.

Spearman's rank correlation ($SRCC \in \mathbb{R}^{[-1,1]}$) (Hauke and Kossowski [2011]) is a special case of the PCC, where the data is transformed into ranks before calculating the correlation coefficient. For two variables vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ it can be calculated through

$$\begin{aligned} SRCC(\boldsymbol{x},\boldsymbol{y}) &= \frac{\sum_i^N (rk(x_i)-\overline{rk}_x)(rk(y_i)-\overline{rk}_y)}{\sqrt{\sum_i^N (rk(x_i)-\overline{rk}_x)^2}\sqrt{\sum_i^N (rk(y_i)-\overline{rk}_y)^2}} \\ &= \frac{\frac{1}{N}\sum_i^N (rk(x_i)rk(y_i)) - \overline{rk_x rk_y}}{\sigma_{rk_x}\sigma_{rk_y}} \\ &= \frac{cov(rk_x, rk_y)}{\sigma_{rk_x}\sigma_{rk_y}} \end{aligned} \tag{4.1}$$

with $rk(x_i)$ being the rank of the input parameter value x_i, $\overline{rk}_x$ the mean value of the ranks of $\boldsymbol{x}$ (which is equal to $N/2$), σ_{rk_x} is the standard deviation of the ranks of $\boldsymbol{x}$ and $cov(rk(\boldsymbol{x}), rk(\boldsymbol{y}))$ describes the covariance between $rk(\boldsymbol{x})$ and $rk(\boldsymbol{y})$. In contrast to the PCC, the $SRCC$ is also able to detect non-liner monotonic relationships. Another alternative, similar to the $SRCC$, is the Kendall's τ correlation coefficient. Due to its similarity to $SRCC$, it will not be discussed here in detail. Hence, the reader is referred to the literature like McLeod [2005] for further information.

Mutual information ($MI \in \mathbb{R}^{[\geq 0]}$) (Peng et al. [2005], Cover and Thomas [2012]) is a more uncommon type of correlation coefficient. In probability and information theory, the mutual information of two random variables is a measure of the mutual dependence between them. It quantifies the amount of information (based on entropy) obtained about one random variable through observing the other random variable. It is defined as

$$MI(\boldsymbol{x},\boldsymbol{y}) = \sum_i^N p(x_i,y_i)\log\left(\frac{p(x_i,y_i)}{p(x_i)\,p(y_i)}\right) \tag{4.2}$$

with $p(x_i, y_i)$ is the joint probability distribution of x_i, y_i and $p(x_i)$, $p(y_i)$ are the marginal distributions of x_i and y_i respectively. Intuitively, mutual information (MI) measures the information that $\boldsymbol{x}$ and $\boldsymbol{y}$ share. It measures to what extent knowing one of these variables reduces uncertainty about the other (Yang et al. [2012]). Compared to the PCC and $SRCC$, it is able to measure univariate, non-linear and non-monotonic relationships. Due to its range $MI \in \mathbb{R}^{[\geq 0]}$, it is not able to show the direction of correlation like PCC or $SRCC$ and is mostly useful for variable ranking. There are several works about how to calculate a normalized version of the $MI \in \mathbb{R}^{[0,1]}$. For the following tests the version presented in Wijaya et al. [2017] will be used:

$$MI_{norm}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_i^N p(x_i, y_i) \log\left(p(x_i)p(y_i)\right)}{\sum_i^N p(x_i, y_i) \log\left(p(x_i, y_i)\right)} - 1 \tag{4.3}$$

This normalized version is also known as information quality ratio which quantifies the amount of information of a variable based on another variable against total uncertainty.

Distance correlation coefficient ($DCC \in \mathbb{R}^{[0,1]}$) (Székely et al. [2007]) is the last correlation coefficient to be considered in this section and was also already presented in subsection 3.3.2 of chapter 3. Its calculation formula was given in eq. 3.15. Similar to the MI, it is also able to measure univariate, non-linear and non-monotonic relationships and the correlation measure is never zero if there is statistical dependency between two variables x, y.

4.2.2 *Comparison*

To give an better feeling for how these correlation coefficients analyze relationships, fig. 4.3 shows sets of data points with $N = 1,000$ for two variables x and y for which the PCC, $SRCC$, MI and DCC will be calculated (the used data set is inspired by Wikipedia contributors [2020]). The results are given in tab. 4.1. As shown, the PCC and $SRCC$ are able to capture the linear monotonic correlations but fail at the third row of data sets with the non-linear and non-monotonic correlations. MI_{norm} and DCC are also able to capture these. The MI_{norm} underestimates the correlations of set 2 and 6 but on all other sets it is comparable to the DCC. Overall the DCC seems to be favorable over the other tested metrics.

In a second test, uniform distributed data sets with $N = 40, 80, 120, 250, 500, 1,000$ samples are created for the Himmelblau test function (eq. 2.8) already used. This function depends on two parameters x_1, x_2 nearly identically, as illustrated in fig. 4.4. In this test, an additional 8 parameters (x_3, ..., x_{10}) are created but not used within the test function evaluation. This test will show if one or more of the PCC, $SRCC$, MI_{norm} and DCC coefficients are useful for a pre-variable selection and identify correctly that x_1, x_2 are the only important parameters. This test will also analyze if the sample size influences the correlation values or not. Furthermore, the samples are created once via random uniform sampling and once via an optimized LHS to see if this also influences the results. The uniform random sampling is repeated 30 times to measure the mean and standard deviation of the correlation values.

Fig. 4.5 and fig. 4.6 show the results of the correlation values for each of the 10 input parameters for the different sample sizes. For better visualization, the absolute correlation values for the PCC and $SRCC$ coefficients are given. A dashed red line marks the 0.1

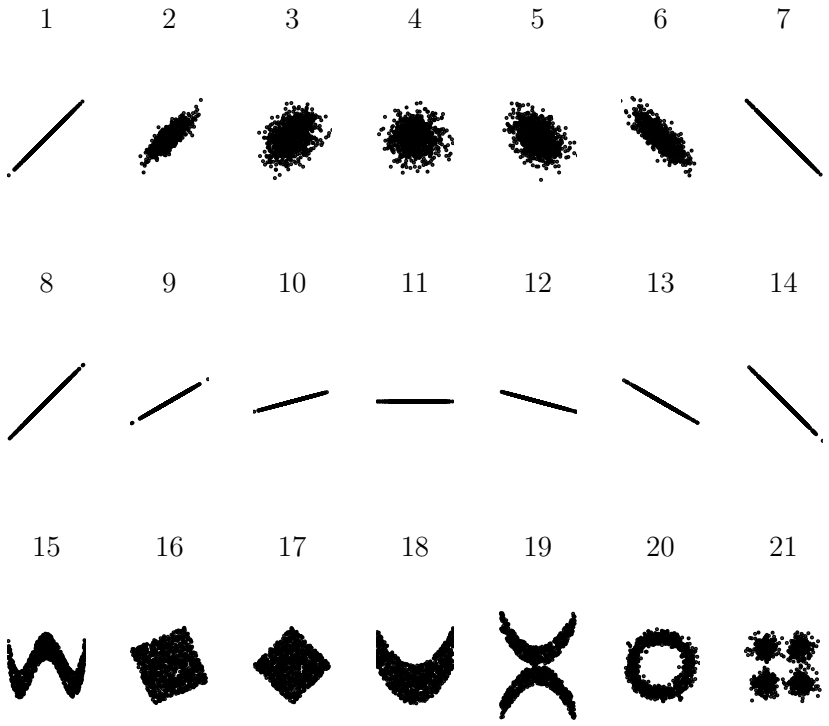


Figure 4.3: Test examples of data points for two variables x, y (the used data set is inspired by Wikipedia contributors [2020]).

Set	PCC	$SRCC$	MI_{norm}	DCC
1	1	1	1	1
2	0.8	0.78	0.13	0.75
3	0.38	0.38	0.07	0.36
4	-0.01	-0.02	0.06	0.05
5	-0.39	-0.38	0.07	0.36
6	-0.81	-0.8	0.14	0.77
7	-1	-1	1	1
8	1	1	1	1
9	1	1	1	1
10	1	1	1	1
11	0.04	0.04	0.06	0.06

Set	PCC	$SRCC$	MI_{norm}	DCC
12	-1	-1	1	1
13	-1	-1	1	1
14	-1	-1	1	1
15	0.01	0.01	0.17	0.34
16	-0.06	-0.07	0.09	0.16
17	-0.01	-0.02	0.09	0.15
18	0.04	0.02	0.12	0.34
19	-0.01	-0.01	0.18	0.21
20	0.01	0.01	0.15	0.17
21	-0.02	-0.03	0.06	0.03

Table 4.1: Results for the different correlation coefficients on 21 different test sets shown in fig. 4.3 with $N = 1,000$ samples.

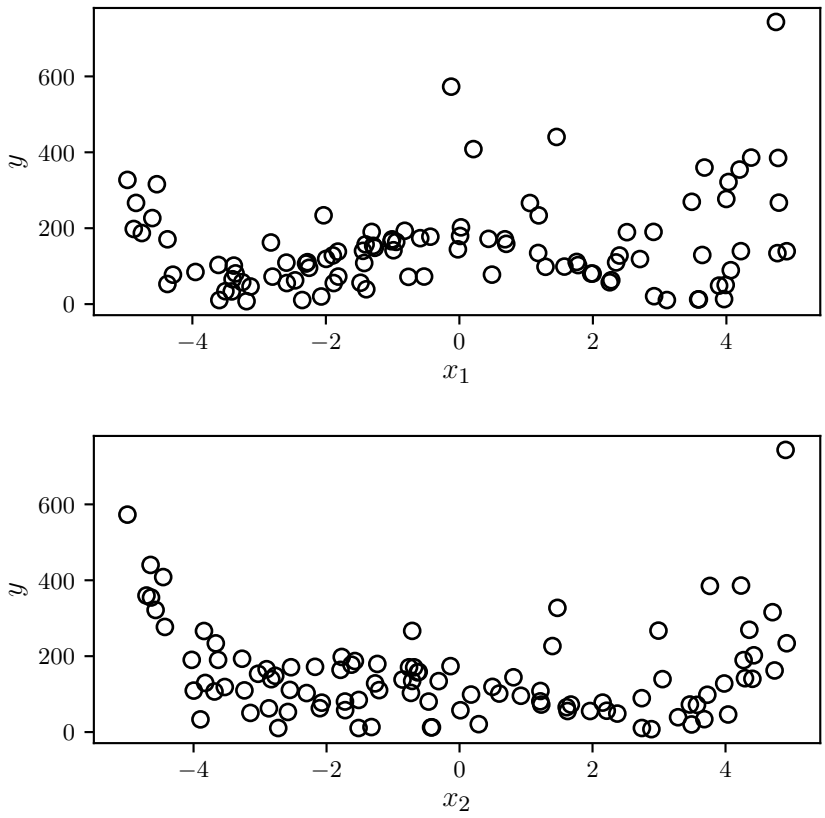


Figure 4.4: Visualization of the one dimensional correlations between y, x_1 and x_2 of the Himmelblau function.

value which should be used as a threshold to decide if a variable is important or not. As can be seen, all coefficients partially have problems to identify that only x_1 and x_2 are the important parameters, especially if the sample size is low. The PCC and $SRCC$ show a worse convergence behavior of the correlation values compared to the MI_{norm} and the DCC. Also, the standard deviation of the PCC and $SRCC$ metric is much higher which makes them somewhat unreliable or dependent on the sampling if no optimized design plan, like the LHS, is used. Overall, the optimized LHS seems to give better results compared to the random uniform sampling. x_1 is not identified as an important parameter for most sample sizes via the PCC and $SRCC$ metric. For sample sizes $N >= 120$ via the LHS, they correctly identify all unimportant parameters. The DCC even needs about $N >= 250$ samples to identify the important and unimportant parameters correctly. The MI_{norm} also only needs $N = 120$ designs for this, but the correlation values for the important parameters x_1 and x_2 are only very close over the threshold value of 0.1. Even though, the DCC coefficient does not identify the unimportant parameters correctly as fast as the others, the values for the two important parameters are clearly over the important threshold for every sample size.

Furthermore, to check the significance of the correlation coefficients, since the selected threshold value was chosen by experience and might not always be the best choice, the mean and standard deviation of $p-$values (Wasserstein and Lazar [2016]) are estimated for

all parameters for all sample sizes over all 30 repetitions. The $p-$value is the probability that the current result of the correlation coefficient have been estimated even though the correlation coefficient were in fact zero (null hypothesis). The smaller the $p-$value is, the stronger the evidence that the null hypothesis can be rejected. The results are given in tab. 4.2. Bold values indicate significant correlation estimates with $p-$values < 0.05 based on the mean value. Starting with $N = 80$ samples DCC and MI_{norm} have significant correlation coefficient estimates for x_1 and x_2. PCC and $SRCC$ only detect correctly x_2 stating with $N = 120$ samples. This underlines the already discussed results given in fig. 4.5 and fig. 4.6.

	$N = 40$				$N = 80$				$N = 120$			
Parameter	PCC	$SRCC$	MI_{norm}	DCC	PCC	$SRCC$	MI_{norm}	DCC	PCC	$SRCC$	MI_{norm}	DCC
x_1	0.38±0.35	0.44±0.28	0.11±0.13	0.18±0.20	0.39±0.25	0.48±0.38	**0.00±0.01**	0.07±0.08	0.54±0.38	0.29±0.19	**0.00±0.00**	**0.03±0.02**
x_2	0.20±0.21	0.22±0.27	0.10±0.11	0.07±0.08	0.35±0.29	0.27±0.24	**0.04±0.10**	**0.01±0.02**	**0.02±0.02**	**0.01±0.01**	**0.00±0.00**	**0.00±0.00**
x_3	0.48±0.29	0.56±0.18	0.39±0.11	0.47±0.27	0.59±0.25	0.64±0.25	0.38±0.15	0.65±0.14	0.53±0.25	0.46±0.23	0.38±0.14	0.59±0.32
x_4	0.55±0.21	0.53±0.25	0.43±0.08	0.66±0.24	0.48±0.31	0.54±0.31	0.49±0.05	0.49±0.34	0.48±0.28	0.53±0.31	0.31±0.19	0.47±0.31
x_5	0.53±0.22	0.58±0.30	0.31±0.20	0.52±0.27	0.47±0.28	0.51±0.24	0.38±0.15	0.54±0.28	0.42±0.30	0.55±0.31	0.41±0.15	0.47±0.28
x_6	0.36±0.24	0.38±0.29	0.36±0.20	0.39±0.26	0.41±0.23	0.47±0.28	0.34±0.13	0.49±0.31	0.59±0.24	0.61±0.26	0.38±0.16	0.55±0.27
x_7	0.39±0.28	0.46±0.33	0.31±0.18	0.43±0.32	0.43±0.24	0.41±0.20	0.40±0.10	0.56±0.27	0.53±0.26	0.45±0.32	0.34±0.17	0.57±0.29
x_8	0.51±0.22	0.50±0.23	0.39±0.12	0.50±0.26	0.54±0.27	0.49±0.24	0.39±0.13	0.59±0.34	0.48±0.25	0.51±0.24	0.28±0.16	0.46±0.26
x_9	0.46±0.36	0.32±0.28	0.16±0.11	0.29±0.22	0.58±0.19	0.53±0.25	0.34±0.16	0.61±0.25	0.53±0.24	0.49±0.27	0.44±0.10	0.50±0.25
x_{10}	0.53±0.24	0.50±0.27	0.35±0.13	0.52±0.29	0.62±0.22	0.64±0.16	0.42±0.10	0.69±0.21	0.62±0.23	0.64±0.28	0.40±0.13	0.65±0.25

	$N = 250$				$N = 500$				$N = 1000$			
Parameter	PCC	$SRCC$	MI_{norm}	DCC	PCC	$SRCC$	MI_{norm}	DCC	PCC	$SRCC$	MI_{norm}	DCC
x_1	0.25±0.28	0.46±0.32	**0.00±0.00**	**0.00±0.00**	0.43±0.42	0.25±0.28	**0.00±0.00**	**0.00±0.00**	0.14±0.19	0.43±0.32	**0.00±0.00**	**0.00±0.00**
x_2	**0.04±0.09**	**0.04±0.06**	**0.00±0.00**	**0.00±0.00**	0.09±0.18	0.06±0.08	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**	**0.00±0.00**
x_3	0.55±0.22	0.57±0.31	0.30±0.19	0.47±0.24	0.50±0.22	0.42±0.24	0.36±0.17	0.41±0.26	0.46±0.29	0.46±0.25	0.40±0.14	0.45±0.28
x_4	0.50±0.20	0.61±0.18	0.40±0.14	0.62±0.26	0.57±0.34	0.52±0.28	0.33±0.20	0.46±0.27	0.61±0.22	0.68±0.25	0.42±0.13	0.65±0.26
x_5	0.53±0.27	0.53±0.29	0.44±0.15	0.59±0.28	0.45±0.21	0.55±0.28	0.40±0.15	0.46±0.26	0.44±0.29	0.49±0.26	0.31±0.23	0.43±0.23
x_6	0.47±0.32	0.50±0.31	0.35±0.15	0.46±0.27	0.55±0.28	0.53±0.33	0.39±0.13	0.64±0.28	0.46±0.22	0.51±0.29	0.37±0.18	0.51±0.34
x_7	0.54±0.26	0.59±0.23	0.27±0.17	0.56±0.25	0.41±0.30	0.41±0.34	0.40±0.16	0.36±0.31	0.56±0.28	0.52±0.26	0.29±0.18	0.62±0.27
x_8	0.48±0.36	0.44±0.28	0.35±0.17	0.44±0.36	0.50±0.33	0.48±0.23	0.43±0.14	0.59±0.26	0.54±0.27	0.58±0.29	0.37±0.17	0.69±0.33
x_9	0.53±0.27	0.65±0.28	0.38±0.13	0.60±0.20	0.36±0.27	0.34±0.30	0.31±0.17	0.34±0.30	0.52±0.28	0.54±0.23	0.39±0.15	0.52±0.23
x_{10}	0.55±0.32	0.54±0.36	0.40±0.13	0.47±0.27	0.64±0.22	0.66±0.21	0.43±0.16	0.55±0.21	0.31±0.24	0.37±0.30	0.34±0.17	0.35±0.32

Table 4.2: Overview of the mean ± standard deviation of the $p-$values for all correlation coefficients and for all sample sizes, estimated over 30 repetitions. The null hypothesis is that there is no correlation. Bold values indicate significant correlation estimates with $p-$values < 0.05 based on the mean value.

If these correlation coefficients would be used for pre-variable selection before training a machine learning algorithm, it turns out (for this example), that the MI_{norm} and DCC always gave the correct important parameters but take higher sample sizes to correctly remove all unimportant parameters. As already mentioned, these types of correlation coefficients are never zero (or very low) if there is a statistical dependence between the variables. Even if unimportant parameters are still left in the data set, this might still work in combination with a machine learning algorithm which can also automatically reduce the influence of unimportant parameters, like for example $\mathcal{DGCN}$. The PCC and $SRCC$ also removed for some sample sizes the important parameters which would result in a problem for the surrogate model training. This makes them very unattractive to use as a pre-variable selection metric, besides their limited capabilities to detect non-linear and non-monotonic relationships.

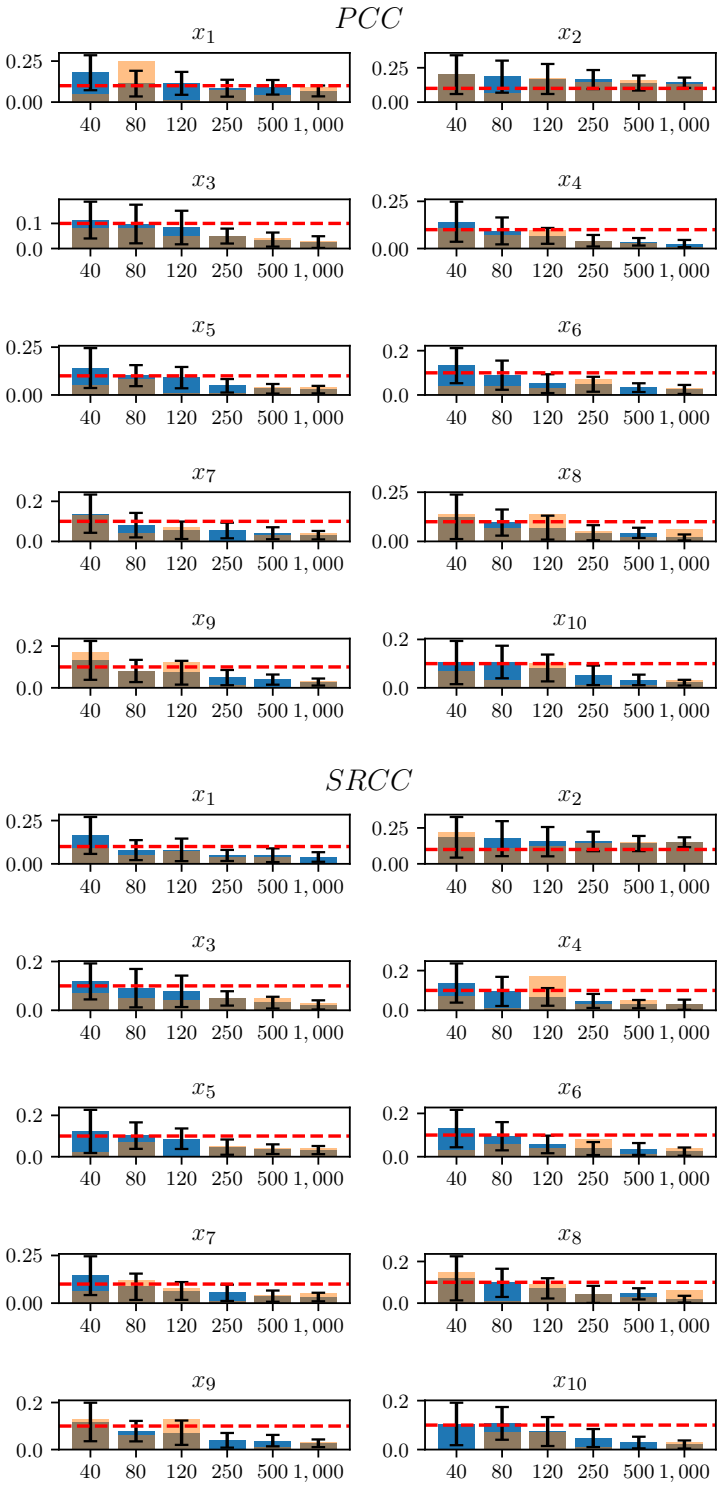


Figure 4.5: Overview of *PCC* and *SRCC* correlation values for all 10 parameters for different sample sizes. x-axis shows sample sizes and y-axis the correlation value. The red dashed line marks the 0.1 correlation value. Blue bars are based on the uniform random sampling and orange transparent bars are based on an optimize *LHS*. Error bars indicates the standard deviation of 30 repetitions for the uniform random sampling. Blue bars are the mean value.

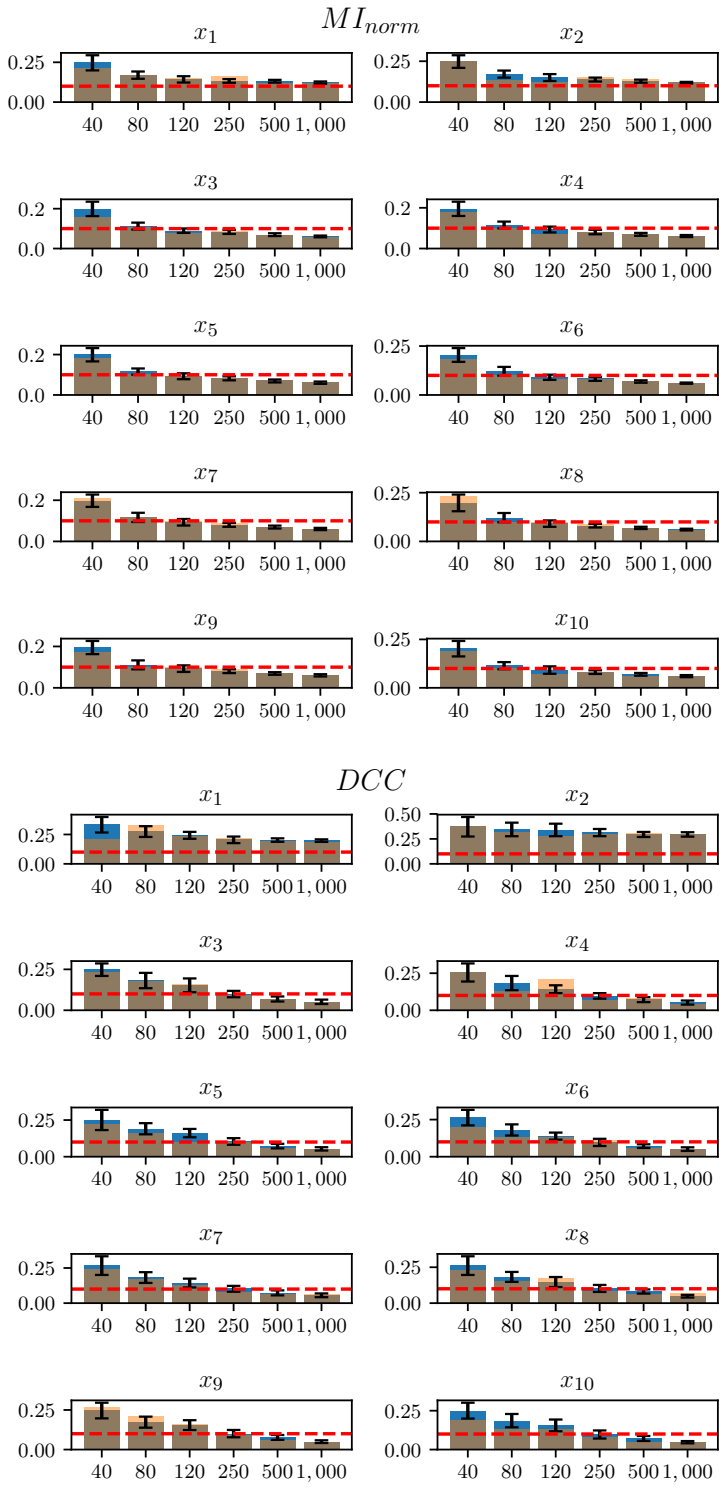


Figure 4.6: Overview of MI_{norm} and DCC correlation values for all 10 parameters for different sample sizes. x-axis shows sample sizes and y-axis the correlation value. The red dashed line marks the 0.1 correlation value. Blue bars are based on the uniform random sampling and orange transparent bars are based on an optimize LHS. Error bars indicates the standard deviation of 30 repetitions for the uniform random sampling. Blue bars are the mean value.

Important notes:

- There exists different types of correlations, namely: linear / non-linear, monotonic / non-monotonic, uni- / multivariate.
- Not all correlation coefficients are able to recognize all of these correlation types. The detection of multivariate correlations by correlation coefficients is especially uncommon.
- The PCC and the $SRCC$ are two typically used correlation coefficients which are very limited and as demonstrated, dependent on the sampling.
- The MI_{norm} and the DCC are two modern correlation coefficients which are able to detect all correlation types except the multivariate. They are also less dependent on the sampling.
- All correlation coefficients presented here need a sufficient number of samples to give correct correlation values.
- The property of the MI_{norm} and the DCC to be never zero if there is a statistical dependence between the two variables to be analyzed, makes them usable as a pre-variable selection method. In conjunction with a surrogate model which is able to reduce the influence of possible remaining unimportant parameters, these coefficients can help to increase the predictive power of the model.
- The PCC and the $SRCC$ are not recommended to be used for a pre-variable selection, since both cannot capture non-linear correlations and the PCC is also not able to detect non-monotonic correlations. Furthermore, in the second example, it was shown that they even remove important parameters.
- Correlation coefficients are the most efficient way to remove unimportant parameters compared to the methods of the following sections, since they do not need a surrogate / numerical model to be involved for further sample evaluations to analyze the correlations.

4.3 COEFFICIENT OF IMPORTANCE

4.3.1 *Overview*

This section will present a straightforward solution to analyze the importance of an input parameter based on surrogate prognosis coefficients which will be called CoI - coefficient of importance (Bucher [2009]). The idea is to train a surrogate model, e.g. a RSM model, on the full set of input parameters in X and to estimate for example the $R^2_{adj}(X)$. Then the CoI for the parameter x_i is calculated through the difference of the $R^2_{adj}(X)$ of the full model and the $R^2_{adj}(X_{\sim i})$ of the reduced model as

$$CoI_i = R^2_{adj}(X) - R^2_{adj}(X_{\sim i}),\ CoI_i \in \mathbb{R}^{[0,1]} \tag{4.4}$$

The relationships which can be identified depend on the predictive power of the surrogate model used. For example, a linear RSM model can only detect monotonic linear relationships (similar to the PCC). If, for example, a powerful model like a DNN or a $\mathcal{GP}$ model is used, all kinds of correlations can be detected. This method would also work with any other prognosis coefficient. Coefficients in $\in [0,1]$ have the advantage that the CoI is also $\in [0,1]$. If cross-terms are left out (e.g. $R^2_{adj}(X_{\sim i,j})$) then even multivariate sensitivities can be estimated. One big drawback of this method is that it involves a retraining of the surrogate model for every CoI_i to be estimated. Therefore, this method is usually applied on fast training models like the RSM. Models like $\mathcal{DGCN}$ would be very ineffective to be used with this kind of sensitivity analysis.

In Xu and Gertner [2008], Da Veiga et al. [2009] and Most [2012] extensions to this approach are presented for correlated inputs. These extensions decompose the variance explained by a variable into its correlated and uncorrelated fractions. These works assume linear correlations which can be described by a correlation or covariance matrix. This is a very limited usage in practice, since correlations are often non-linear and of functional nature. Thus, this topic will be further investigated in subsection 4.4, where a method will be presented which is also able to decompose the correlated and independent parts without any restrictions.

4.3.2 *Numerical Example*

A short numerical example for the CoI estimation will be given for the following test function:

$$y = x_1^2 + x_2 5 + x_3 0.001 + x_2 x_3,\ x_1,\ x_2,\ x_3 \in \mathbb{R}^{[-5,5]} \tag{4.5}$$

The CoI_i will be estimated for all three input parameters. x_1, x_2 have a major influence and x_3 only a minor one. The test function is illustrated in fig. 4.7 for x_1, x_2 as well as in fig. 4.8 where the single dependencies are also visualized. As the surrogate model, a second order RSM model with cross-terms is used. The $R^2_{adj}(X)$ for all input parameters is 1. So calculating the CoI_i holds:

$$\begin{aligned}
CoI_1 &= R^2_{adj}(X) - R^2_{adj}(X_{\sim 1}) = 1.0 - 0.87 = 0.13 \\
CoI_2 &= R^2_{adj}(X) - R^2_{adj}(X_{\sim 2}) = 1.0 - 0.22 = 0.78 \\
CoI_3 &= R^2_{adj}(X) - R^2_{adj}(X_{\sim 3}) = 1.0 - 0.80 = 0.20 \\
CoI_{23} &= R^2_{adj}(X) - R^2_{adj}(X_{\sim 23}) = 1.0 - 0.16 = 0.84
\end{aligned} \tag{4.6}$$

The multivariate CoI_{23} can simply be calculated by keeping x_2, x_3 left out simultaneously. It can be seen that the multivariate CoI is higher than each of the single CoI_i. The results are also visualized in fig. 4.9.

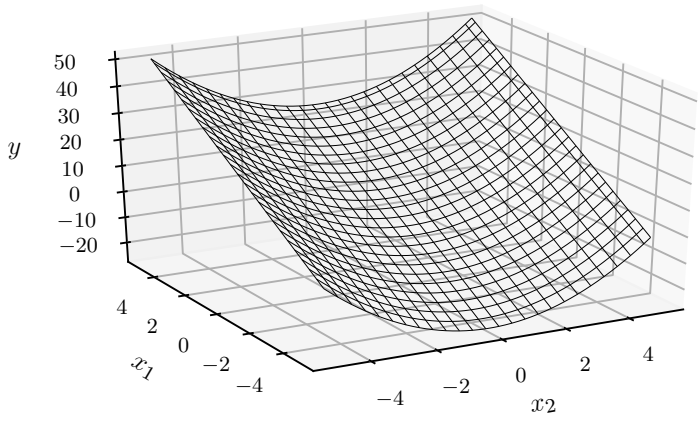


Figure 4.7: Visualization of the test function to be analyzed via the *CoI*.

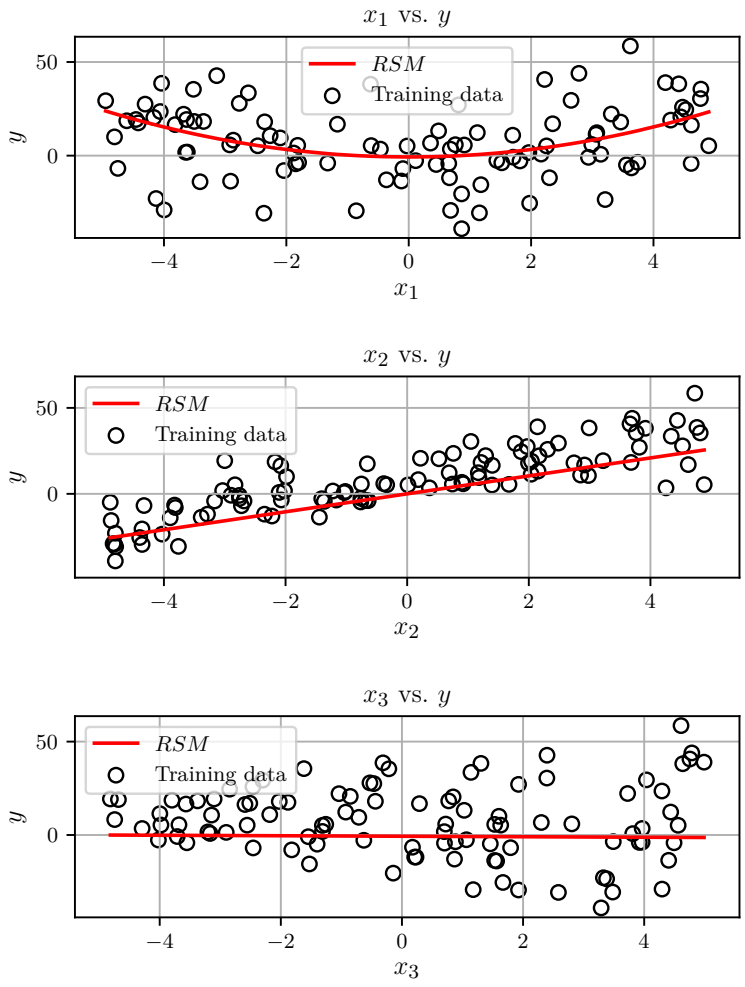


Figure 4.8: Visualization univariate dependencies for x_1, x_2, x_3.

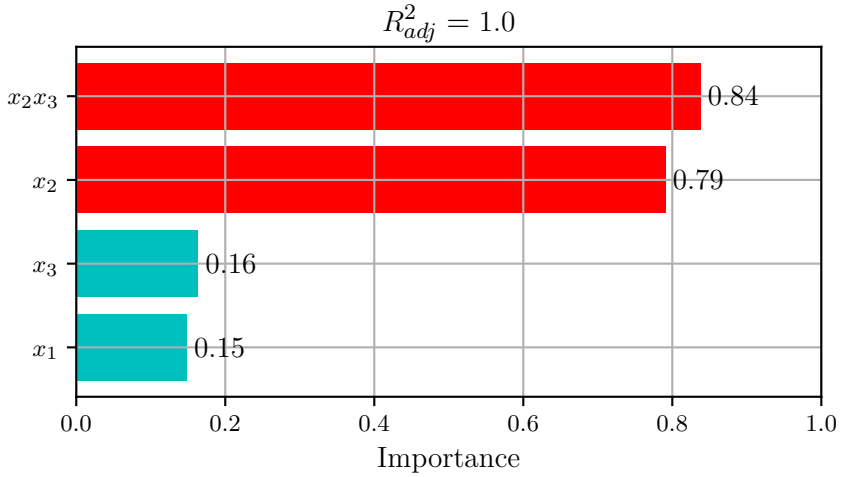


Figure 4.9: Barplot of the CoI results for the parameters x_1, x_2, x_3, x_2x_3.

Important notes:

- The CoI is a simple way to estimate the uni-/ or even multivariate dependencies between the input and output parameters.
- Its power depends on the predictive power of the surrogate model used. A surrogate model with a low prognosis quality might also detect wrong sensitivities.
- Due to the numerically expensive calculation of the CoI, it is only efficiently usable with fast training surrogate models like the RSM.
- For interpolating algorithms like the $\mathcal{DGCN}$, it becomes even more expensive, since cross-validation-based metrics like the $\hat{R}^2$ must be used.

4.4 GLOBAL-VARIANCE-BASED ANALYSIS

4.4.1 *Overview*

Another method to estimate the importance of parameters is the global-variance-based analysis (Saltelli et al. [2008]). This kind of sensitivity analysis is also often referred to as the Sobol method or Sobol indices, after Ilya M. Sobol (Sobol [1993]) or as analysis of variance ANOVA (Lamboni et al. [2011]). Working within a probabilistic framework, it decomposes the variance of the output of the model into fractions which can be attributed to inputs or sets of inputs. Variance-based measures of sensitivities are attractive because they measure them across the whole input space, they can deal with non-linear, non-monotonic responses, and they can measure the effect of interactions in non-additive systems (multivariate sensitivities). In literature, two types of ANOVA can be distinguished: the functional ANOVA-based on Sobol indicies and the ANOVA by design of experiment. The functional ANOVA method does not require any hypothesis on the form of the model except that

the input factors are independent (Brevault et al. [2013]). Under this assumption, Sobol showed the following decomposition for a function f:

$$f(\boldsymbol{x}) = f_0 + \sum_{i=1}^{n} f_i(x_i) + \sum_{i}^{n}\sum_{j>i}^{n} f_{ij}(x_i x_j) + ... + f_{123...n}(x_1, ..., x_n) \tag{4.7}$$

where $f_0 = \mathbb{E}(f(\boldsymbol{x})) = \int_{\Omega^n} f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$. Ω^n is the n-dimensional cube $[0,1]^n$ where the input parameters are defined and assumed to have a uniform distribution $\mathcal{U}(0,1)$ represented by the probability distribution $p(\boldsymbol{x})$. Furthermore, $f_i(x_i) = \mathbb{E}(f(\boldsymbol{x})|x_i) - f_0$, $f_{ij}(x_i, x_j) - \mathbb{E}(f(\boldsymbol{x}|x_i) - \mathbb{E}(f(\boldsymbol{x}|x_j) - f_0$ are defined as the difference between $f(\boldsymbol{x})$ and the sum of all the increasing dimension functions. $f_{123...n}(x_1, ..., x_n)$ denotes the k-th-order interaction among all of the factors. Based on this functional decomposition, Sobol introduces the Sobol indices to quantify the partition of the output variance as

$$\mathbb{V}(f(\boldsymbol{x})) = \sum_{i=1}^{n} \mathbb{V}_i(f_i(x_i)) + \sum_{i}^{n}\sum_{j>i}^{n} \mathbb{V}_{ij}(f_{ij}(x_i x_j)) + ... + \mathbb{V}_{123...n}(f_{1...n}(x_1, ..., x_n)) \tag{4.8}$$

with $\mathbb{V}$ being the variance and $\mathbb{V}_i(f(x_i))$ is the variance of the function output explained by the parameter x_i. In fact, the variances of the terms in the decomposition above are the measures of importance being sought. In particular, $\mathbb{V}_i(f_i(x_i))$ is $\mathbb{V}(\mathbb{E}(y|x_i))$; when this is divided by the unconditional variance $\mathbb{V}(y)$ the first-order sensitivity index can be obtained via

$$S_i = \frac{\mathbb{V}(\mathbb{E}(y|x_i))}{\mathbb{V}(y)} \tag{4.9}$$

The first-order index represents the main effect contribution of each input factor to the variance of the output. The same quantity has been described by different investigators as an importance measure (see Hora and Iman [1986], Ishigami and Homma [1990], Iman and Hora [1990], Saltelli et al. [1993], Homma and Saltelli [1996]) and as a correlation ratio (see McKay [1996], Hofer et al. [2002]). Besides the single contribution of a parameter, it is also interesting to know the multivariate contribution of two factors. Two factors are said to interact when their effect on y cannot be expressed as a sum of their single effects. Interactions may imply, for instance, that extreme values of the output y are uniquely associated with particular combinations of model inputs in a way that is not described by the first-order effects S_i just mentioned. Interactions represent important features of models, and are more difficult to detect than first-order effects (Saltelli et al. [2008]). In eq. 4.8, $\mathbb{V}_{ij}(f_{ij}(x_i x_j))$ describes these interaction effects which are calculated as

$$\mathbb{V}_i j = \mathbb{V}_{ij}(f_{ij}(x_i x_j)) = \mathbb{V}(\mathbb{E}(y|x_i, x_j)) - \mathbb{V}(\mathbb{E}(y|x_i)) - \mathbb{V}(\mathbb{E}(y|x_j)) \tag{4.10}$$

In this equation, $\mathbb{V}(\mathbb{E}(y|x_i, x_j))$ measures the joint effect of the parameter pair x_i, x_j on y. This term is also known as the second-order effect (Box et al. [1978]). Analogous formulas can be written for higher-order terms, enabling the analyst to quantify the higher-order interactions. As the first-order indice was obtained by dividing $\mathbb{V}(\mathbb{E}(y|x_i))$ by $\mathbb{V}(y)$, it is also possible now to rewrite eq. 4.8 in terms of Sobol indices by dividing the whole equation by $\mathbb{V}(y)$:

$$\sum_{i=1}^{n} S_i + \sum_{i}^{n}\sum_{j>i}^{n} S_{ij} + \sum_{i}^{n}\sum_{j>i}^{n}\sum_{l>j}^{n} S_{ijl} + ... + S_{123...n} = 1 \tag{4.11}$$

The number of terms in this equation increases exponentially with the number of input factors.

Finally, it is also possible to estimate the total effects which are often the most interesting ones in practical applications. Total effects are a direct consequence of Sobol's variance decomposition approach and estimation procedure, although they were explicitly introduced and made computationally affordable by other investigators (see Homma and Saltelli [1996], Saltelli and Tarantola [2002]). The total effect index measures for the total contribution to the output variation due to factor x_i, i.e its first-order effect plus all higher-order effects due to interactions (Saltelli et al. [2008]). For a three-factor model, for example, the total effect of x_1 is the sum of all the terms in eq. 4.11, where the factor x_1 is considered:

$$S_{T_1} = S_1 + S_{12} + S_{13} + S_{123} \tag{4.12}$$

If an output parameter is totally dependent on additive sensitivities (S_i), the sum of $\sum_i^n S_i$ equals to 1. On the other hand, if there are also non-additive dependencies, so a significant difference between S_i and S_{T_i}, then the sum of the total effect indices is $\sum_i^n S_{T_i} > 1$ which allows very easy identification of whether there are only additive or also non-additive relationships in the data present. The S_{T_i} could be calculated by estimating all terms in 4.11, but there are as many as $2^n - 1$ of these. Thus, methods were developed like those presented in Homma and Saltelli [1996] which allow the calculation of S_{T_i} at the same cost of first-order indices. These will also be presented in subsection 4.4.2. The estimation of S_{T_i} can give a hint as to which parameters can be left out or set to a constant value without affecting the output y. Another possibility to calculate the total effect is to decompose the variance $\mathbb{V}(y)$ in terms of main effect and residual conditioning with respect to all the input parameters but one, i.e. x_i:

$$\mathbb{V}(y) = \mathbb{V}(\mathbb{E}(y|\boldsymbol{x}_{\sim i})) + \mathbb{E}(\mathbb{V}(y|\boldsymbol{x}_{\sim i})) \tag{4.13}$$

The measure $\mathbb{V}(y) - \mathbb{V}(\mathbb{E}(y|\boldsymbol{x}_{\sim i}))$ is the remaining variance of y that would be left on average if it is possible to determine the true values of $\boldsymbol{x}_{\sim i}$. The average is calculated over all possible combinations of $\boldsymbol{x}_{\sim i}$, since $\boldsymbol{x}_{\sim i}$ are uncertain factors and their true values are unknown. Dividing by $\mathbb{V}(y)$, the total effect indice for x_i can be obtained as (Saltelli et al. [2008])

$$S_{T_i} = \frac{\mathbb{E}(\mathbb{V}(y|\boldsymbol{x}_{\sim i}))}{\mathbb{V}(y)} = 1 - \frac{\mathbb{V}(\mathbb{E}(y|\boldsymbol{x}_{\sim i}))}{\mathbb{V}(y)} \tag{4.14}$$

The Sobol indices presented are able to detect any kind of sensitivities so linear / non-linear, monotonic / non-monotonic as well as uni-/multivariate ones.

Another approach for *SA*-based on the decomposition of the variance can be used. Instead of considering the previously presented functional ANOVA, the conditional variances are calculated with a *DoE*. ANOVA by *DoE* differs from the functional ANOVA in that it can only be used for qualitative or discrete input factors. A discretization of the continuous input factors is necessary. Eq. 4.10 is discretized according to a chosen *DoE* into several levels for each input parameter. By choosing appropriate levels and *DoE*, the conditional

variances can be approximated (Brevault et al. [2013]). This is the classical way sensitivities are estimated along with systematic design plans presented in chapter 3. Since these kinds of design plans are not applied in this work due to the reasons explained before, the way to estimate the ANOVA by *DoE* will not be described here either (see e.g. Lamboni et al. [2011] for an overview).

4.4.2 *Numerical Computation of Sobol Indices*

The computation of the conditional variances such as $\mathbb{V}(\mathbb{E}(y|x_i))$ and $\mathbb{V}(\mathbb{E}(y|x_i, x_j))$ would be the cumbersome, brute-force computation of the multidimensional integrals in the space of the input factors (Saltelli et al. [2008]). Therefore, in this subsection, an efficient method based on a Monte-Carlo-based numerical procedure for computing the full set of first-order and total-effect indices for a model of n factors as proposed in Saltelli and Tarantola [2002] will be presented.

In a short form, the steps to estimate the first and total order indice for x_i via the Monte Carlo approach are:

- Generate two matrices A and B of size $N \times n$ with N number of samples and n number of input parameters. This will be done with respect to the probability distributions and multivariate correlations of the input parameters.

$$A = \begin{bmatrix} x_{A1}^{(1)} & x_{A2}^{(1)} & \dots & x_{Ai}^{(1)} & \dots & x_{An}^{(1)} \\ x_{A1}^{(2)} & x_{A2}^{(2)} & \dots & x_{Ai}^{(2)} & \dots & x_{An}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{A1}^{(N)} & x_{A2}^{(N)} & \dots & x_{Ai}^{(N)} & \dots & x_{An}^{(N)} \end{bmatrix} \tag{4.15}$$

$$B = \begin{bmatrix} x_{B1}^{(1)} & x_{B2}^{(1)} & \dots & x_{Bi}^{(1)} & \dots & x_{Bn}^{(1)} \\ x_{B1}^{(2)} & x_{B2}^{(2)} & \dots & x_{Bi}^{(2)} & \dots & x_{Bn}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{B1}^{(N)} & x_{B2}^{(N)} & \dots & x_{Bi}^{(N)} & \dots & x_{Bn}^{(N)} \end{bmatrix} \tag{4.16}$$

- Define a matrix C_i formed by all columns of B except the i-th column (marked in red) which is taken from A:

$$C_i = \begin{bmatrix} x_{B1}^{(1)} & x_{B2}^{(1)} & \dots & x_{Ai}^{(1)} & \dots & x_{Bn}^{(1)} \\ x_{B1}^{(2)} & x_{B2}^{(2)} & \dots & x_{Ai}^{(2)} & \dots & x_{Bn}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{B1}^{(N)} & x_{B2}^{(N)} & \dots & x_{Ai}^{(N)} & \dots & x_{Bn}^{(N)} \end{bmatrix} \tag{4.17}$$

- Compute the model output for all the input values in the sample matrices A, B, and C_i, obtaining three vectors of model outputs of dimension $N \times 1$:

$$\boldsymbol{y}_A = f(A),\ \boldsymbol{y}_B = f(B),\ \boldsymbol{y}_{C_i} = f(C_i) \tag{4.18}$$

The S_i can now be calculated as

$$S_i = \frac{\mathbb{V}(\mathbb{E}(y|x_i))}{\mathbb{V}(y)} = \frac{\boldsymbol{y}_A \cdot \boldsymbol{y}_{C_i} - f_0^2}{\boldsymbol{y}_A \cdot \boldsymbol{y}_A - f_0^2} = \frac{(1/N)\sum_{j=1}^{N} y_A^{(j)} y_{C_i}^{(j)} - f_0^2}{(1/N)\sum_{j=1}^{N}\left(y_A^{(j)}\right)^2 - f_0^2} \tag{4.19}$$

where

$$f_0^2 = \left(\frac{1}{N}\sum_{j=1}^{N} \boldsymbol{y}_A^{(j)}\right)^2 \tag{4.20}$$

is the mean, and the symbol $(\cdot)$ denotes the scalar product of the two vectors. Similarly, the method estimates total-effect indices as follows

$$S_{T_i} = 1 - \frac{\mathbb{V}(\mathbb{E}(y|\boldsymbol{x}_{\sim i}))}{\mathbb{V}(y)} = 1 - \frac{\boldsymbol{y}_B \cdot \boldsymbol{y}_{C_i} - f_0^2}{\boldsymbol{y}_A \cdot \boldsymbol{y}_A - f_0^2} = 1 - \frac{(1/N)\sum_{j=1}^{N} y_B^{(j)} y_{C_i}^{(j)} - f_0^2}{(1/N)\sum_{j=1}^{N}(y_A^{(j)})^2 - f_0^2} \tag{4.21}$$

The total cost of this approach is $N(n+2)$ which is fairly low compared to other strategies. Error estimates for eq. 4.19 and eq. 4.21 can be obtained by bootstrapping data points from vectors $\boldsymbol{y}_A$, $\boldsymbol{y}_B$ and $\boldsymbol{y}_{C_i}$ which allows the confidence intervals of the sensitivity estimates to be calculated. In Saltelli et al. [2010] an overview is given of further estimators for the S_i and S_{T_i} which may be somewhat more accurate but does not change the ranking of the parameters.

The estimation of the CoI_i in eq. 4.4 presented in the previous section also represents the total order Sobol indice with

$$S_{T_i} = CoI_i = R^2_{adj}(X) - R^2_{adj}(X_{\sim i}) \tag{4.22}$$

By using only the polynomial basis terms which belong to the variable x_i, it is also possible to estimate via the regression approach the first order indice through

$$S_i = R^2_{adj}(X_i) \tag{4.23}$$

4.4.3 *Sobol Indices for Dependent Parameters*

In practical applications, input variables are often correlated, therefore, several works presented methods to extend the classical methods for correlated inputs, as already mentioned, for the coefficient of importance. In these works, assumptions about the model output or approximations up to a certain degree are necessary (Oakley and O'Hagan [2004], Xu and Gertner [2008], Da Veiga et al. [2009]). Recent works showed that it is also possible for the matrix-based approaches to decompose the Sobol indices into correlated and independent parts which are model independent. Most [2012] proposed an extension of the classical Sobol estimator which is based on his previous work Most et al. [2010] where a variable decomposition similar to Xu and Gertner [2008] was realized. The general idea of Most [2012] has been adapted in Mara and Tarantola [2012] and extended in Mara et al. [2015]

for normally distributed random variables. In the approach of Most [2012], linear correlations between the input variables are assumed, in order to decompose the samples into a correlated and an uncorrelated part. This was realized in Mara and Tarantola [2012] through an orthogonalization scheme which may lead to non-unique solution depending on the variable order. In the approach of Most [2012], the contributions of the input variables are decomposed directly by using the correlation information in the standard normal space. This decomposition results in a unique solution. For non-normal distributions, the Nataf model (Nataf [1962]) is applied to transform the original space and the correlation information to the standard normal space. Mara et al. [2015] also uses the Rosenblatt transformation (Rosenblatt [1952]) for the same purposes. The proposed method finally provides modified sampling matrices following the original joint probability distribution, while keeping the samples of a certain variable unchanged. These methods again provide matrices similar to the approach already presented which can be used to estimate the Sobol indices but with the decomposition of correlated S_i^C, $S_{T_i}^C$ and independent S_i^U, $S_{T_i}^U$ parts. Since for the Rosenblatt and Nataf transformation the CDF of the data is needed as well as information about the covariance matrix (or more precisely its Cholesky decomposition), it only has limited application in practice, since this information is not always given or must be approximated. Furthermore, if surrogate models are to be used to evaluate the required sampling matrices, it might give wrong results for the sensitivities because the model needs to extrapolate due to the need to also evaluate the uncorrelated design matrices. Typically, there are no correlated and uncorrelated training points available in practice to train the surrogate model. Alternatively, to the decomposition approach, in Most et al. [2010] a reordering approach is proposed. In this method the variable investigated has been taken as the first one in the variables set for the Cholesky decomposition in the standard normal space. With this approach, the samples required for the estimation of the sensitivity indices can be determined directly. This method has been adapted in Kucherenko et al. [2012]. The drawback of this method is the higher numerical effort compared to the decomposition approach and therefore will not be discussed further.

After giving an overview of existing methods and their limitations (see also section 4.3), a new method to estimate the correlated and independent part of the first and total order indices will be presented. The approach will also use a matrix approach. Furthermore, no knowledge of either the covariance matrix and none of the transformations mentioned (Nataf or Rosenblatt) are necessary. This makes the method applicable for any kind of correlation and multivariate distribution of input parameters. Moreover, every design matrix will follow the original correlations which makes it possible to estimate the indices on surrogate models.

In the previously presented matrix approach, it was assumed that the input parameters were not correlated but in the presence of correlations, the design matrix C_i must also follow these. This means if x_i is correlated to $X_{\sim i}$, then $X_{\sim i}$ in C_i must be adjusted accordingly and because $B(X_{\sim i})$ must be equal to $C_i(X_{\sim i})$ a modification of B is also necessary. So assuming a start with the creation of A and taking x_i of it for C_i, then C_i as well as B must be recreated for every x_i[1]. Therefore, the notation B_i will be further used. A must be created only once if it is used as the first design matrix. To modify C_i according to the conditional probabilities $p(C_i(X_{\sim i})|A(x_i))$ and B_i to the conditional probability

1 At this point the mentioned works of others use the Nataf / Rosenblatt transformation in order to sample in the uniform uncorrelated space and then transform the samplings back to the original space.

$p(B_i(X_{\sim i})|C_i(X_{\sim i}))$ the rejection-sampling-based algorithm presented in section 3.5 will be utilized. So, first matrix A is generated which can be used for all x_i and then x_i of A will be used for the i-th column of C_i. Now the remaining points of C_i are sampled with the rejection sampling algorithm in such a way that the correlation and distribution are considered regarding x_i from A. A small modification of the algorithm 2 is necessary to do this[2]. Then, if C_i is created a matrix B_i must be generated, this time only the column i must be estimated via the rejection sampling algorithm under the consideration of $C_i(X_{\sim i})$. Overall for n parameters, $2n+1$ matrices must be generated via the rejection sampling algorithm. The necessary steps to estimate the sensitivities for parameter x_i are summarized in the following:

1. Create matrix A with respect of the joint distribution and correlations of all parameters in X.

2. Take column i of A and use it for the i-th column of matrix C_i. All other columns of C_i must be created via the rejection sampling method in order to fulfill the joint distribution and correlations but also to take into consideration the existing values of the i-th column of A.

3. Create matrix B_i by taking all columns of C_i except the i-th column because it must differ from A. The i-th column must be generated with the rejection sampling method taking into account the joint distribution and correlations but also the existing columns of C_i which are reused in B_i.

Furthermore, the equations for the first and total order Sobol indices must be extended for the decomposition of the correlated and independent part. Per definition eq. 4.19 corresponds to the correlated first order indice S_i^C and eq. 4.21 to the independent total order indice $S_{T_i}^U$ (see Most [2012]). To explain this in more detail and to better understand the two new formulations for the S_i^U and $S_{T_i}^C$ the first equation to look at would be the correlated first order indice S_i^C. It should be assumed that all $\boldsymbol{y}_A$, $\boldsymbol{y}_{B_i}$, $\boldsymbol{y}_{C_i}$ are normalized to have zero mean and variance one so that the previously used f_0^2 and the division by $\mathbb{V}(y)$ becomes obsolete if the formulation without the dot product is used:

$$S_i^C = \frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{C_i}^{(j)} \tag{4.24}$$

In this equation the elements of $\boldsymbol{y}_A$ and $\boldsymbol{y}_{C_i}$ are based on the matrices A and B_i which share only the i-th column. The idea of this formulation is that if x_i has a huge influence on the output y, then it is very likely that high / low values in $\boldsymbol{y}_A$ are multiplied by high / low values $\boldsymbol{y}_{C_i}$. If x_i has only a small or even no importance, then the combination of high and low values in both vectors are more or less random. So the mean of the product of $\boldsymbol{y}_A$ and $\boldsymbol{y}_{C_i}$ is much higher if x_i is important. Note that in the presence of correlated inputs variables, the combination of high / high or low / low values might also be due to the correlation x_i has to the remaining parameters. So in the extreme case, where x_i has a linear correlation of one to all other parameters, then A, C_i and so $\boldsymbol{y}_A$, $\boldsymbol{y}_{C_i}$ would be exactly the same which would result in the maximum product of the two vectors. Therefore, the original definition of the first order indice also includes the influence of correlations in its

2 The rejection sampling is restricted to the columns left to adjust C_i and B_i.

estimate of importance. In this work therefore, the definition of the correlated Sobol first order indice includes the first order independent influence as well as possible influences through correlations and therefore $S_i^U \leq S_i^C$. So, if S_i^C is larger than S_i^U, it is due to correlations. This is contrary to the mentioned works, where it will be shown in subsection 4.4.4 that their estimates also allow for $S_i^C < S_i^U$.

Next, the new developed definition of S_i^U is provided as

$$S_i^U = \frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{C_i}^{(j)} - \frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{B_i}^{(j)} = \frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} \left(y_{C_i}^{(j)} - y_{B_i}^{(j)}\right) \tag{4.25}$$

C_i and B_i have all elements equal except for x_i. If x_i would have no independent influence on y (e.g. a correlation of one to all other variables) then it would be the case that $y_{C_i} = y_{B_i}$ and so $S_i^U = 0$. The influence of x_i due to correlations on the result of $\boldsymbol{y}_{C_i} - \boldsymbol{y}_{B_i}$ is not present, since all elements $X_{\sim i}$ of C_i and B_i are equal. Thus, the difference between $\boldsymbol{y}_{C_i}$ and $\boldsymbol{y}_{B_i}$ can only occur due to the uncorrelated influence of x_i. The higher the uncorrelated influence of x_i, the higher is the difference between $\boldsymbol{y}_{C_i}$ and $\boldsymbol{y}_{B_i}$. The part $\frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{B_i}^{(j)}$, which can be seen as the first order indice only including importance due to correlation, would be zero if x_i has no correlation to any other parameter, since A and B_i are then fully independent from each other. If x_i is correlated to any other parameter then because x_i from A is in C_i and therefore all $X_{\sim i}$ of C_i must follow the same correlation as x_i in A to $X_{\sim i}$ from A, then B_i and A are not independent anymore. As explained for S_i^C, there is then a higher likelihood that lower / higher values in $\boldsymbol{y}_A$ meet lower / higher values in $\boldsymbol{y}_{B_i}$ and therefore $\frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{B_i}^{(j)}$ is not zero anymore and thus $S_i^U \leq S_i^C$.

The next equation is for the independent total order indice estimation:

$$S_{T_i}^U = 1 - \frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)} \tag{4.26}$$

As mentioned, this equation corresponds to the original S_{T_i} for independent input parameters. Here the influence of all other parameters except x_i, including its possible correlations to all $X_{\sim i}$, is calculated via $\frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)}$ and therefore is the remaining variance the total independent influence of x_i on y including interaction effects with other variables and without correlations. As mentioned before, if x_i has a correlation of one to all other variables then it would be the case that $\boldsymbol{y}_{C_i} = \boldsymbol{y}_{B_i}$ and so $\frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)} = 1$ and $S_{T_i}^U$ would be zero.

Finally, the also newly developed equation for the correlated total order indice is

$$S_{T_i}^C = 1 - \frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)} - \frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_A^{(j)} = 1 - \frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} \left(y_{C_i}^{(j)} - y_A^{(j)}\right) \tag{4.27}$$

After explaining the previous indices, it can quickly be seen that this formulation removes the correlation part from $\frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)}$, so that the remaining variance still includes the effects due to correlations. From this, it also follows that $S_{T_i}^U \leq S_{T_i}^C$. To summarize the unique parts in every equation presented:

- $\frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{C_i}^{(j)}$ is the pure additive influence of x_i on y with correlations.

- $\frac{1}{N}\sum_{j=1}^{N} y_A^{(j)} y_{B_i}^{(j)}$ is the total influence of x_i on y only due to correlations to the remaining parameters. The connection of $\boldsymbol{y}_A$ to $\boldsymbol{y}_{B_i}$ is given over $\boldsymbol{y}_{C_i}$. If no correlations are present this would be zero.
- $\frac{1}{N}\sum_{j=1}^{N} y_{B_i}^{(j)} y_{C_i}^{(j)}$ is the total influence of all variables except x_i on y with correlations and interactions.

A closer look also reveals that the equations presented for the Sobol indices are very similar to the estimation of the Pearson correlation coefficient (eq. 3.8) for two variables with zero mean and variance one. Therefore, the linear Pearson correlation coefficient between the different vectors $\boldsymbol{y}_A$, $\boldsymbol{y}_{B_i}$, $\boldsymbol{y}_{C_i}$ is calculated. As mentioned, a procedure to estimate the uncertainty of those indices would be possible via bootstrapping or by repeating the creation of the sampling matrices several times.

4.4.4 *Numerical Examples*

In the following some sensitivity analysis examples will be given in order to test the new equations along with the proposed rejection sampling. Since these equations presented for S_i^U and $S_{T_i}^C$ represent another way to calculate those indices, together with the nature of the rejection algorithm not providing exact linear correlation coefficient replications, there is no claim to obtain exactly the same values as in the works used for comparison. These other methods use the exact covariance matrices to estimate their indices. In the new approach, it is assumed that the covariance matrices are not known in advance and therefore only data driven. Most important is that the ranking of the variables is similar and that the method does not declare a variable as unimportant if this is not true.

The first example is from Most [2012] and is a purely additive model with

$$y(\boldsymbol{x}) = x_1 + x_2 + x_3 \tag{4.28}$$

The input variables are normally distributed, have zero mean and the covariance:

$$\Gamma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \rho\sigma \\ 0 & \rho\sigma & \sigma^2 \end{bmatrix}$$

Different test cases with $\rho = 0.0, 0.8, 0.99999, -0.8, -0.5$ and $\sigma = 2$ are investigated. Most [2012] used $10,000$ samples created via LHS, for the analysis via the proposed matrix recombination method. Therefore, $10,000$ samples will also be used for the rejection-sampling-based approach in combination with the equations presented. The data to create the new $10,000$ samples from (so the training data for the rejection sampling algorithm) will be sampled with $1,000$ data points. All design matrices are evaluated on the true function for comparison reasons. In practice, a surrogate model must be utilized if no fast evaluating function is available. The analytical results for S_i^C for this example are derived in Da Veiga et al. [2009] as

$$
\begin{aligned}
S_1^C &= \frac{1}{2+\sigma^2+2\rho\sigma} \\
S_2^C &= \frac{(1+\rho\sigma)^2}{2+\sigma^2+2\rho\sigma} \\
S_3^C &= \frac{(\rho+\sigma)^2}{2+\sigma^2+2\rho\sigma}
\end{aligned}
\tag{4.29}
$$

Tab. 4.3 shows the results for the first example. It can be seen that the results for $\rho = 0.0, 0.8, 0.99999$ show good agreements between the analytical solutions and the other numeric results. For the negative correlations $\rho = -0.8, -0.5$ there is a difference between the rejection-sampling-based values and the others results for the variables x_2, x_3 (the correlated ones). There are significant differences for the indices $S_{T_i}^C$, S_i^U which are based on the new formulations. In the matrix recombination approach, it seems that the negative correlations result in lower values for the correlated indices compared to the independent ones. This can be seen more clearly for $\rho = -0.5$. The self-developed formulas always provide an equal or higher value for the correlated indices. The other difference is between S_i^U and $S_{T_i}^U$. In the matrix recombination approach, they are, for this linear test function, nearly equal for all values of ρ. For the rejection-sampling-based values, it seems that the values of S_i^U move over to $S_{T_i}^C$ for the negative correlated cases which might be due to the formulation of the equations. In Most [2012], it also mentioned that correlation might introduce indirect interaction effects between the parameters. Another possible explanation for this is that the positive correlations, where x_2, x_3 change in the same direction and due to the definition of eq. 4.28 their values go purely additively into the result of y, it makes sense that for those cases all have the same value for the first order indices and the total order indices. With the negative correlations, a huge part of the additive independent influence of x_2, x_3 in y vanishes because they work against each other. Since x_3 has a higher variance than x_2 and therefore a wider parameter range which is less compensated by x_2, it must have a higher importance. This kind of behavior can also be seen as an interaction effect between x_2 and x_3. In fig. 4.10, y is plotted versus each input parameter one by one which represents a visualization of the correlated first order indice. It can be seen that the influence of x_2 is only marginal compared to x_3. Especially for $\rho = -0.5$ it looks like a variable with no influence on y. Also, if x_2 is compared to x_1 for the case $\rho = -0.5$, it does not look like they both should have a nearly equal first order indice as in the result of Most [2012]. Since in the new equations the correlated indices are always higher than the independent ones, it makes sense that the results become like this. In any case, both methods are able to analyze each parameter in such a way that no important parameter can be missed. For example, the influence of a variable seems to vanish when looking only at the correlated or uncorrelated formulation of the total effect indices which can be observed for example for x_2, x_3 in the uncorrelated formulation $S_{T_i}^U$ with $\rho = 0.99999$.

The second example will be a coupled non-linear function also taken from Most [2012] with

$$
y(\boldsymbol{x}) = x_1 + 2x_1^2 + x_2 + x_3 + x_2 x_3 \tag{4.30}
$$

Once again, the input parameters are normally distributed with the same covariance as in the first example. This time only two cases will be analyzed with $\rho = 0.0, 0.8$. Tab. 4.4 shows the results of the matrix recombination approach and the rejection sampling. Since

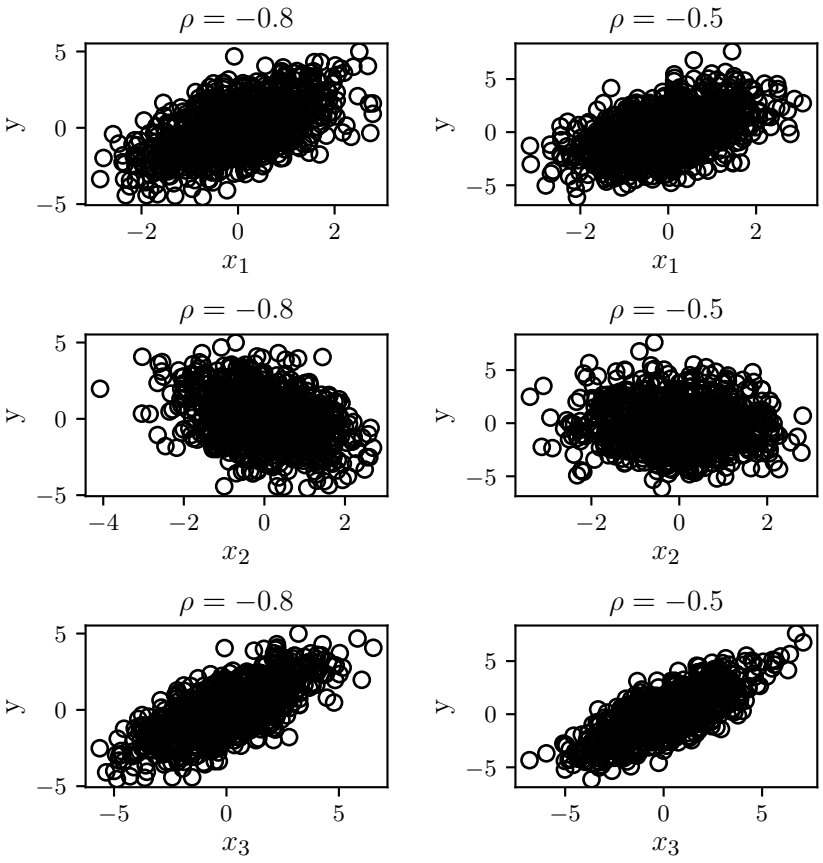


Figure 4.10: Visualization of the first order dependencies between y and x_2, x_3.

here only positive correlations are present, the results match very well between the two methods.

The third example taken from Most [2012] is the well-known Ishigami function (Ishigami and Homma [1990]). This function is defined for independent uniformly distributed variables

$$-\pi \leq x_i \leq \pi,\ i = 1,\ 2,\ 3 \tag{4.31}$$

as follows

$$y(\boldsymbol{x}) = \sin(x_1) + 7\sin(x_2)^2 + 0.1x_3^4\sin(x_1) \tag{4.32}$$

Once again, three different test cases will be tested. The results are given in tab. 4.5. This time for the case $\rho = 0.0$, the analytic results for S_i^C, $S_{T_i}^U$ are also available. The results for $\rho = 0.0,\ 0.5$ match very well between the two methods. The results for the case with $\rho_{12} = 0.3$, $\rho_{13} = 0.5$, $\rho_{23} = 0.8$ does not match again for the two indices $S_{T_i}^C$, S_i^U based on the new equations. x_3 is the least important parameter in the rejection-sampling-based approach regarding $S_{T_i}^C$ instead of the most important parameter as in the matrix recombination approach. S_i^U for x_1, x_2 has nearly no independent first order influence based on the matrix recombination method in contrast to the rejection sampling, where these still

	Analytical	Matrix recombination				Rejection sampling			
	S_i^C	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$
$\rho = 0.0$									
x_1	0.167	0.157	0.168	0.157	0.166	0.171	0.171	0.164	0.171
x_2	0.167	0.154	0.164	0.153	0.163	0.172	0.176	0.155	0.176
x_3	0.667	0.664	0.673	0.669	0.672	0.678	0.679	0.676	0.679
$\rho = 0.8$									
x_1	0.109	0.117	0.108	0.118	0.107	0.117	0.119	0.110	0.112
x_2	0.735	0.738	0.727	0.050	0.039	0.662	0.662	0.034	0.034
x_3	0.852	0.858	0.848	0.169	0.157	0.832	0.835	0.149	0.149
$\rho = 0.99999$									
x_1	0.100	0.088	0.103	0.088	0.101	0.107	0.106	0.095	0.105
x_2	0.900	0.900	0.916	0.000	0.000	0.915	0.915	0.001	0.001
x_3	0.900	0.900	0.916	0.000	0.000	0.894	0.894	0.005	0.005
$\rho = -0.8$									
x_1	0.357	0.361	0.359	0.359	0.359	0.350	0.360	0.360	0.363
x_2	0.129	0.141	0.130	0.136	0.139	0.133	0.302	0.016	0.106
x_3	0.514	0.527	0.513	0.520	0.512	0.552	0.632	0.552	0.549
$\rho = -0.5$									
x_1	0.250	0.240	0.250	0.243	0.249	0.235	0.277	0.239	0.247
x_2	0.000	0.005	0.000	0.179	0.188	0.004	0.205	0.000	0.183
x_3	0.563	0.571	0.570	0.755	0.752	0.556	0.739	0.566	0.738

Table 4.3: Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.28. For this example the analytical solution is also given for S_i^C.

have a high influence. Since the only two parameters which should have an independent total order influence based on the test function equation are x_1, x_3, it is questionable again if this result should remain the same for this correlated case or, as in the results of Most [2012], new independent interaction effects have been introduced due to the correlation and therefore also x_2 should have a much higher $S_{T_i}^U$ value compared to the S_i^U value. The answer to this question is left open for further investigation in the future.

The last example is taken from Mara et al. [2015] and is also based on the linear eq. 4.28. The parameters are normally distributed with zero mean and covariance:

$$\Gamma = \begin{bmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{bmatrix}$$

with two different test cases $(\rho_{12}, \rho_{13}, \rho_{23}) = (0.5, 0.8, 0.0)$ and $(\rho_{12}, \rho_{13}, \rho_{23}) = (-0.5, 0.2, -0.7)$. The proposed method of Mara et al. [2015] is also a matrix recombination algorithm and therefore also named as such for the comparison.

	Matrix recombination				Rejection sampling			
	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$
$\rho = 0.0$								
x_1	0.489	0.518	0.490	0.518	0.440	0.470	0.416	0.445
x_2	0.036	0.256	0.034	0.259	0.054	0.238	0.056	0.241
x_3	0.217	0.452	0.216	0.454	0.295	0.465	0.254	0.423
$\rho = 0.8$								
x_1	0.405	0.362	0.405	0.361	0.401	0.444	0.409	0.451
x_2	0.528	0.540	0.055	0.071	0.407	0.407	0.056	0.056
x_3	0.548	0.592	0.090	0.123	0.507	0.507	0.114	0.114

Table 4.4: Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.30.

	Analytical		Matrix recombination				Rejection sampling			
	S_i^C	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$
$\rho = 0.0$										
x_1	0.314	0.557	0.330	0.570	0.330	0.570	0.320	0.557	0.311	0.549
x_2	0.442	0.442	0.456	0.429	0.456	0.429	0.441	0.444	0.447	0.450
x_3	0.000	0.244	0.008	0.251	0.008	0.251	0.007	0.204	0.017	0.228
$\rho_{13} = 0.5$										
x_1	-	-	0.312	0.453	0.065	0.346	0.300	0.455	0.108	0.348
x_2	-	-	0.484	0.473	0.484	0.473	0.477	0.486	0.461	0.471
x_3	-	-	0.172	0.444	0.071	0.200	0.193	0.417	0.097	0.230
$\rho_{12} = 0.3$, $\rho_{13} = 0.5$, $\rho_{23} = 0.8$										
x_1	-	-	0.306	0.799	0.057	0.354	0.325	0.693	0.304	0.361
x_2	-	-	0.559	0.783	0.022	0.454	0.502	0.629	0.408	0.423
x_3	-	-	0.198	0.927	0.022	0.117	0.184	0.503	0.070	0.156

Table 4.5: Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.32.

Similar to the results of Most [2012], the results of Mara et al. [2015] always provide the same results for the first order and total order indices, as shown in tab. 4.6. The first correlated test case with only positive correlation matches very well between both methods. The second case introducing negative correlations, creates a similar mismatch as the results of Most [2012] for negative correlations. x_2 loses its first order indice due to increased $S_{T_i}^C$ values. It seems that what is missing in S_i^U is exactly added to $S_{T_i}^C$. The two original indices fit very well.

	Matrix recombination				Rejection sampling			
	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$
$\rho_{12} = 0.5,\ \rho_{13} = 0.8,\ \rho_{23} = 0.0$								
x_1	0.94	0.94	0.02	0.02	0.89	0.89	0.02	0.02
x_2	0.40	0.40	0.05	0.05	0.34	0.34	0.05	0.05
x_3	0.58	0.58	0.03	0.03	0.61	0.61	0.04	0.04
$\rho_{12} = -0.5,\ \rho_{13} = 0.2,\ \rho_{23} = -0.7$								
x_1	0.49	0.49	0.72	0.72	0.48	0.60	0.44	0.64
x_2	0.04	0.04	0.37	0.37	0.03	0.39	0	0.37
x_3	0.25	0.25	0.48	0.48	0.24	0.48	0.25	0.49

Table 4.6: Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Mara et al. [2015] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.28.

For all examples, the design matrices were evaluated on the true functions for comparison reasons and to test the new equations without influencing the results by the use of a surrogate model. To demonstrate the estimation of the Sobol indices also on a $\mathcal{DGCN}$ model, the Ishigami test function will be analyzed again. To train the model $N = 200$ samples are used. Tab. 4.7 shows the result of this test. As can be seen, the results obtained based on the $\mathcal{DGCN}$ model are very similar to those obtained on the true analytical function (see tab. 4.5). It seems that even correlated sensitivities can be analyzed correctly on a surrogate model provided its accuracy is sufficient.

	Analytical		Matrix recombination				Rejection sampling			
	S_i^C	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$	S_i^C	$S_{T_i}^C$	S_i^U	$S_{T_i}^U$
$\rho = 0.0$										
x_1	0.314	0.557	0.330	0.570	0.330	0.570	0.320	0.535	0.340	0.555
x_2	0.442	0.442	0.456	0.429	0.456	0.429	0.467	0.447	0.465	0.465
x_3	0.000	0.244	0.008	0.251	0.008	0.251	0.006	0.194	0.004	0.232
$\rho_{13} = 0.5$										
x_1	-	-	0.312	0.453	0.065	0.346	0.328	0.428	0.101	0.347
x_2	-	-	0.484	0.473	0.484	0.473	0.464	0.464	0.484	0.484
x_3	-	-	0.172	0.444	0.071	0.200	0.206	0.406	0.105	0.213
$\rho_{12} = 0.3$, $\rho_{13} = 0.5$, $\rho_{23} = 0.8$										
x_1	-	-	0.306	0.799	0.057	0.354	0.345	0.667	0.294	0.345
x_2	-	-	0.559	0.783	0.022	0.454	0.535	0.635	0.404	0.414
x_3	-	-	0.198	0.927	0.022	0.117	0.173	0.559	0.048	0.143

Table 4.7: Comparison of S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ calculated via the matrix recombination of Most [2012] and the rejection-sampling-based approach together with the presented new indice equations for test function 4.32. This time a $\mathcal{DGCN}$ model is used to evaluate the design matrices.

Important notes:

- The global-variance-based sensitivity analysis decomposes the variance of the output into fractions which can be mapped to its input parameters.
- It is possible to analyze all kind of dependencies as long as the variance can be decomposed correctly.
- These methods typically need many design evaluations, for example, to estimate the Sobol indices if the matrix approach is used. Thus, in practice it makes sense to evaluate the design matrices on surrogate models.
- There are different equations in literature to estimate the first and total order indices.
- Some works even provide the opportunity to decompose the variance in correlated and independent first and total order indices. This is very important in order to never declare a parameter as unimportant if working with correlated data.
- The drawback of all mentioned existing methods is that they only work with linear correlations, for which the covariance matrix is needed. Also, besides the correlated design matrices, they need to evaluate the independent design matrices which might lead to extrapolation if evaluated on surrogate models only trained on the correlated design space. For non-Gaussian distributed input parameters, a transformation into the Gaussian space is necessary. Therefore, the Nataf or Rosenblatt transformation is typically utilized.

- A new approach based on the previously presented rejection sampling approach was presented in order to estimate the S_i^C, $S_{T_i}^C$, S_i^U, $S_{T_i}^U$ for any kind of correlation in the original data space which is necessary to evaluate the design matrices on surrogate models and avoid extrapolation. This method does not need any kind of design space transformation.
- This new approach showed in most test cases very good agreement to the other methods except for negatively correlated examples. It seems that the two new formulations for $S_{T_i}^C$, S_i^U lead to different results for this type of correlation. Some suggestions were made why these differences might occur, but it is still an open topic for future research.
- Also correlated sensitivities can be estimated on a surrogate model if its accuracy is high enough.
- Finally, with all methods presented, no variable would be declared as unimportant if it not the case, and this is the most important aspect.

4.5 DERIVATIVE-BASED GLOBAL SENSITIVITY MEASURES

4.5.1 *Overview*

One drawback of the Sobol indices presented in the previous subsection is that their estimation requires a lot of design evaluations depending on the number of parameters. It turns out to be unacceptable for high-dimensional problems involving a large number of input variables (typically more than 10), if not used in connection with surrogate models. Therefore, Kucherenko and Sobol (Kucherenko et al. [2009b]) have proposed the derivative-based global sensitivity measures ($DGSM$), defined as the integral of the squared derivatives of the model output, showing that it can help to solve the problem of high dimensionality in some cases. This method will be presented in the following as the last method for SA in this work because it is highly efficient together with the multi-output $DGCN$ model to estimate the sensitivities of field outputs (multi-outputs). This method can be used, for example, to analyze the importance of finite element nodes which discretized the stress state of a structure in order to find out how geometry parameters influence those stress states. Furthermore, for problems with many input and output parameters, the Sobol indices are numerically more expensive to calculate compared to the $DGSM$ method.

The derivative-based global sensitivity method uses the second moment of model derivatives as the importance measure. This method is motivated by the fact that a high value of the derivative of the model output with respect to some input variable means that a large variation of model output is expected for a variation of the variable (Lamboni et al. [2013]). This method extends the elementary effect analysis of Morris (Morris [1991]), which was not discussed so far, because it is based on systematic design plans with one parameter varied at a time and therefore it is very inefficient. $DGSM$ allows to capture any small variation of the model output due to input variables. It has been largely studied, for example, in Kucherenko et al. [2009a], Kucherenko et al. [2010] and Patelli et al. [2010]. This method is also able to analyze any kind of dependence similar to the Sobol indices.

4.5.2 *Numerical Computation*

As mentioned, the elementary effect analysis (Morris [1991]) is the starting point for the $DGSM$. A differentiable function $f(\boldsymbol{x})$ should be considered, where $\boldsymbol{x} = (x_1, ..., x_n)$ is the vector of the input parameters x_i. Furthermore, these variables should be defined in the unit hypercube $\Omega^n (0 \leq x_i \leq 1, i = 1, ..., n)$. Local sensitivity measures are based on partial derivatives with

$$E_i(\boldsymbol{x}_*) = \frac{\partial f(\boldsymbol{x}_*)}{\partial x_i} \tag{4.33}$$

E_i is the limit version of a finite differencing scheme where the finite difference for x_i tends to zero. In sensitivity analysis, the partial derivative $\partial f(\boldsymbol{x}_*)/\partial x_i$ is known as a local method, since this measure depends on a nominal point $\boldsymbol{x}_*$ and it changes with $\boldsymbol{x}_*$. This deficiency can be overcome by averaging $E_i(\boldsymbol{x}_*)$ over the design space Ω^n. This averaging is the new extension of the elementary effect analysis which makes it to the $DGSM$ method. In Sobol and Kucherenko [2010] several formulas of the $DGSM$ for uniform distributed or randomly distributed parameters are given. For randomly distributed parameters a differentiable function $f(\boldsymbol{x})$ should be considered again, where $\boldsymbol{x} = (x_1, ..., x_n)$ are independent random variables, defined in the Euclidean space X^n, with cumulative density function (CDF) $F_1(x_1), ..., F_n(x_n)$. Then the following $DGSM$ can be introduced as

$$\nu_i(\boldsymbol{x}_*) = \int_{X^n} \left(\frac{\partial f(\boldsymbol{x}_*)}{\partial x_i} \right)^2 d\boldsymbol{x}_* = \mathbb{E}\left[\left(\frac{\partial f(\boldsymbol{x}_*)}{\partial x_i} \right)^2 \right] \tag{4.34}$$

with F the joint CDF. ν_i is in fact the mean value of $(\partial f(\boldsymbol{x}_*)/\partial x_i)^2$. This Integral can easily be solved for example via Monte Carlo integration. Furthermore, for the partial derivatives the finite difference method can be utilized if the analytical partial derivatives are not available. In Kucherenko et al. [2009b] a link between $DGSM$ and the total order Sobol indice is presented, where it is possible to use the $DGSM$ measure in combination with a weighting function depending on the distribution of the input parameters, to give an upper bound for the total order Sobol indice with $S_{T_i}^U \leq \nu_i$. This relationship could be shown for a large category of continuous distributions.

4.5.3 *Example*

As mentioned, this method is very efficient in calculating sensitivities which makes it preferable to use with a $\mathcal{DGCN}$ model for multi-output problems like the FEM example presented (see subsection 2.10.5). The trained multi-output $DGCN$ model can be used to evaluate a set of samples similar to the Sobol indices (here it is not needed to create different sampling matrices) in combination with the presented rejection sampling algorithm. To estimate the sensitivities a parameter has on a structure, the sensitivity analysis can be seen reversed. The question now is which output parameter (e.g. a FEM node) is affected the most by a single input parameter. To illustrate this, the $DGSM$ is calculated for the example mentioned in subsection 2.10.5. Fig. 4.11 shows the change of node stresses for the minimum / maximum pressure on the rotor structure. Fig. 4.12 visualizes the corresponding $DGSM$ results. As can be seen, the $DGSM$ values for the different nodes give an exact

picture of which nodes are affected the most by the variation of the input parameter.

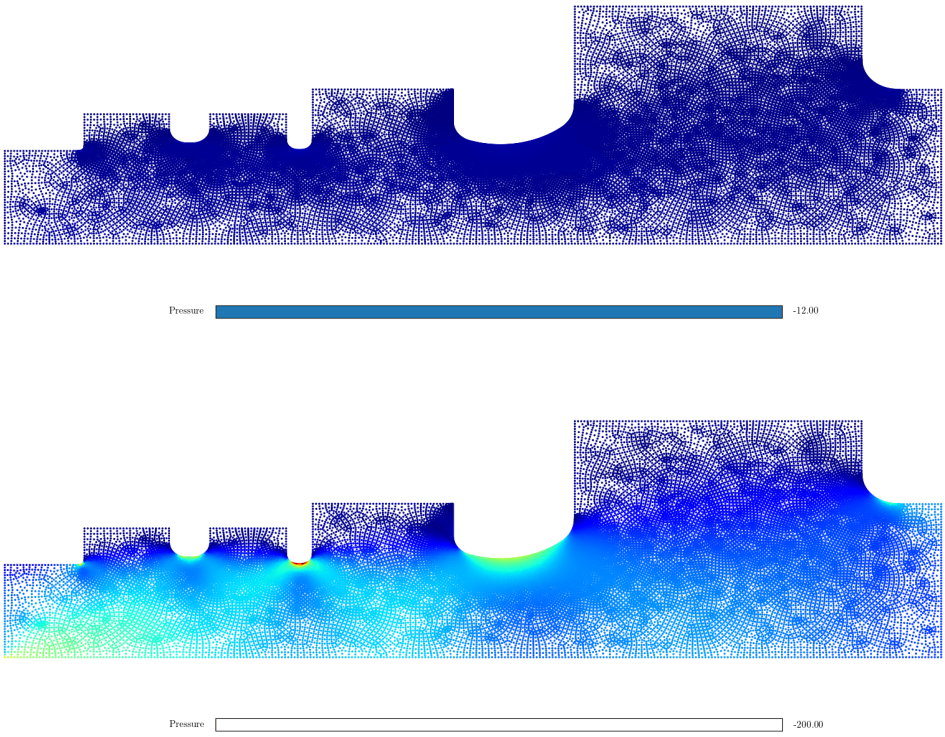


Figure 4.11: Example of the minimum / maximum pressure on the rotor structure and the resulting FEM node stresses.

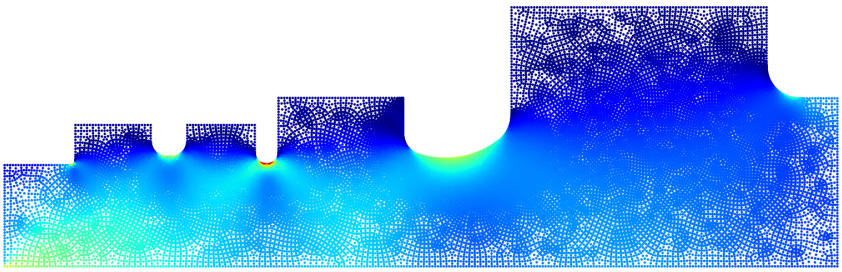


Figure 4.12: $DGSM$ results for the FEM nodes.

Important notes:

- $DGSM$ is a derivative-based method to estimate sensitivities. Its generalization from a local to a global sensitivity measure is made by averaging the partial derivatives over the whole design space.

- *DGSM* can be calculated more efficiently than Sobol indices and can be seen as an upper bound for the total order Sobol indice (Kucherenko et al. [2010]).
- In this work, *DGSM* is utilized to evaluate sensitivities on field outputs, like structures, based on a multi-output $\mathcal{DGCN}$ model. Generally, it is suitable for every multi-output problem to answer the question: which output is affected the most by a single input parameter. At the end it also gives the answer to which parameter is the most important for every output.
- Since the *DGSM* is not scaled between $[0, 1]$ but is based on the mean partial derivatives, it can be used to compare the impact of different inputs on the same field of multiple outputs. This is not possible with the Sobol indices, since their scaling is relative to each output and thus a comparison between multiple outputs would not be possible.

4.6 CONCLUSION

This chapter has given an overview (not complete) of possible methods to analyze the influence of input parameters to an output parameter. It started with a classification of different dependency types which help to differentiate the variety of methods and their strengths and weaknesses. Correlation coefficients were the first type of sensitivity measures presented. The strengths of these measures over the other methods presented is that they can be applied on the raw data without additional sampling and therefore there is no need for further model evaluations on the real numerical simulation / experiment or on a surrogate model. This makes them especially useful to get a first insight into possible sensitivities and thus they are also the recommended method if a pre-variable selection is to be performed before surrogate model training. The weakness of most of these coefficients are their limitations in detecting every type of correlation. There are some exceptions, such as the mutual information metric or the distance correlation coefficient, both presented. In the comparison, it was shown that these coefficients also need a sufficient number of samples to detect the importance of a parameter correctly. In a pre-variable selection it is more important to detect which parameters are unimportant than the correct ranking.

The second category of sensitivity measures presented was the coefficient of importance. This measure utilized the change of prognosis quality in order to analyze the importance of a parameter. Therefore, this method can only be used in conjunction with a surrogate model and typically not as a pre-variable selection. The only way to use it as pre-variable selection efficiently would be to use it with a very fast training algorithm like *RSM* because it needs a retrain of the surrogate model n-times or even more if coupled effects should be analyzed too. The limitation of these sensitivity measures is always based on the predictive power of the surrogate model method used. For example, if a linear *RSM* model is used, then it would not be possible to detect non-linear and non-monotonic sensitivities correctly. For simple problems this method might be suitable.

The third method was the global-variance-based sensitivity analysis which calculates the so-called Sobol indices. This method works by decomposing the variance of an output and maps these fractions on the corresponding input parameters. It is a very powerful method to detect any kind of correlation. As presented, there is even the possibility to distinguish

between correlated and independent sensitivities, but it needs a huge number of samples to be evaluated. Therefore, in practice, this method can typically only be used if the design evaluations are made on a surrogate model. This reveals also its weakness, similar to the CoI, the truth of the estimated sensitivities is dependent on the predictive power of the surrogate model used. If the model does not use a specific input parameter for its prediction, even if it is an important one, this method will also not identify this parameter as important. This makes this method also unsuitable for pre-variable selection. The results of the Sobol indices should always be taken carefully in connection with a prognosis quality measure or some other validation of the model to get a feeling for how reliable these indices are.

The last sensitivity method was the derivative-based global sensitivity measures which uses the design space averaged local partial derivatives of an output to a single input, in order to estimate its importance. Since this method is numerically more efficient compared to the Sobol indice estimation, it is suitable for high dimensional input and output parameter problems. In this work this measure was used for multi-output problems, such as the FEM example presented. It can be used to visualize the importance of a single input parameter on a structure and so on multiple outputs at the same time. This method can, for example, also be used to analyze the importance of an input parameter over time and / or space, since this is also a multi-output problem (as presented in section 2.10).

Not mentioned in this chapter is the use of learned free parameters of surrogate models, for example the correlation length-scale parameters of a $\mathcal{GP}$ model which is also helpful to estimate the importance of a parameter, as suggested in the corresponding chapter. There are a variety of machine learning algorithms where the internal learning of parameters can also be used to make a parameter ranking of the input parameters. Every algorithm with a similar mechanism to the automatic relevance detection can be used for this purpose. For example, the coefficients of a RSM model or the radii parameter of the $AMLS$ are useful for this too. The problem with most of these parameters is the interpretation of the results.

All of the presented methods have their advantages and disadvantages. To give a recommendation on when to use which method, the following points might be helpful:

- For pre-variable selection the mutual information metric or distance correlation coefficient should be used, maybe in connection with an adaptive sampling approach where the change of these metrics is observed. Note that a pre-variable selection is only needed if the surrogate model used has no automatic relevance detection or the amount of input parameters is very large compared to the number of samples. For the $\mathcal{DGCN}$ model typically no pre-variable selection is performed.

- To get a very detailed overview on the sensitivities of the input parameters to one single output, the Sobol indices are a strong choice. Especially, if the decomposition of correlated and independent indices is used. It is important to check the prognosis quality of the surrogate model if one is used. The Sobol indices might even be scaled by this measure.

- To visualize the importance of an input parameter on multiple correlated outputs, the $DGSM$ method is recommended because of its effective calculation and the fact that the values are not scaled between $[0, 1]$. This allows comparisons between the different input parameters for multiple outputs.

Finally, maybe the most important aspect, is that all of these methods need a sufficient number of samples to work. In most cases it is not possible to evaluate this number on the

real simulation / experiment and therefore surrogate models can be used as a replacement. Even if the prognosis quality of those model is perfect, the sensitivity measures might be wrong if the sampling is not created according to the multivariate distribution and multivariate correlations given in the original data. Therefore, methods like the rejection sampling approach presented are key to use these methods in practice in connection with surrogate models efficiently.

5 CONCLUSION

5.1 SUMMARY OF CONTRIBUTIONS

This thesis presented several machine learning related topics in order to perform pattern recognition and design exploration. A new probabilistic model named deep Gaussian covariance network ($\mathcal{DGCN}$) was presented as the main contribution. It was shown how this new model performs in comparison to other state of the art models for several different learning tasks. Other related topics like data generation and sensitivity analysis were also highlighted in combination with this new model. This thesis provided the mathematical background for most of the presented algorithms and tried to give the reader an overview of the state of the art for solving the tasks presented and also which problems still exist. Recommendations are given on how to choose the correct algorithms depending on the problem. This work always claimed to develop algorithms which are useable in industrial scenarios. Therefore, the ease of use (e.g. as few hyperparameters as possible, one algorithm for many different learning tasks) and the compatibility to big data applications were important aspects besides a high efficiency. Since a conclusion was already given at the end of each chapter together with the important notes at the end of each section, in the following the key ideas, contributions and results of this thesis will be summarized in a compact form. Ideas for further research are also discussed in the next section.

Chapter 2

- Section 2.2 presented a critical view of model selection algorithms which are applicable to a wide range of machine learning models. In this section, the main contribution was to highlight the importance of the package structure in k-fold cross-validation. The reader should also get a feeling for the reliability of typically used prognosis measures and their dependencies on the data and machine learning algorithm used.

- Section 2.3 to 2.9 presented the mathematical background for the several methods that were built upon and extended to form the $\mathcal{DGCN}$ model. It should also give the reader the same train of thought as the author had, as to what an algorithm should be capable of, to be usable for several learning tasks. For example, the concept of having one free parameter per input parameter which allows the algorithm automatically to reduce the influence of unimportant parameters. It could be seen that this concept is included in many powerful algorithms and can enhance algorithms which do not have this feature like the moving least squares algorithm and its new developed extension to the anisotropic variant.

- Section 2.10 presents, as already mentioned, the main contribution of this thesis. The $\mathcal{DGCN}$ method combines several different properties a model should have in order to be applicable to a wide range of tasks. This is mainly be done by combining deep neural networks and a Gaussian process. The neural networks are utilized to learn input / test point dependent the free parameters of a Gaussian process in order to

enhance its capabilities. It could be shown that the algorithm performs at state of the art level for several different learning tasks, where typically different algorithms are needed and therefore further expert knowledge is necessary. The main features of this algorithm are:

  - All used kernels within the Gaussian process are automatically non-stationary which is an advantage for non-uniform sampled problems like adaptive sampling approaches or measured data but also to model discontinuities.
  - Automatic choice of the right combination of covariance functions, since every covariance function has its own set of free parameters.
  - Non-stationary noise estimation. Since the noise parameter is also predicted by a DNN, the $\mathcal{DGCN}$ model can learn different noise levels within one data set.
  - Handling infinite data set sizes through mini-batch training which is typically not possible with standard Gaussian processes.
  - Online learning which allows the model to learn directly with the income of new data points very efficiently without needing to retrain the whole model.
  - Extension to sequential dependent problems like, for example, time series.
  - Extension to multi-output problems, where multiple outputs and their correlation are learned efficiently within one model. Examples were given where a $\mathcal{DGCN}$ model learned the relationship over space and time.
  - Image classification at state of the art level, with a much smaller DNN and without extensive research for the optimal net topology.
  - Applicable to regression and classification tasks.
  - Training can be performed on CPU or GPU and even be distributed on multiples of these which makes the algorithm highly scalable.

- Section 2.11 presented some hints for model validation based on the author's experience. This section should be an extension to the model selection section which concluded with the question of what to do if the prognosis coefficients are low and / or not reliable.

Chapter 3

- Section 3.1 to 3.3 gave an overview of existing methods for design plan creation. The main focus here was the usage of stochastic design plans namely the Latin hypercube sampling, since they do not scale exponentially with the number of input parameters. Several metrics known in literature to optimize it were presented. The main contribution here were the comparison of these metrics with the focus on surrogate model training, also for the scenario, where not all parameters are important.

- Section 3.4 presented methods for adaptive sampling. A novel sequential approach for extending a LHS in an optimized way was presented. This extension was also analyzed under the aspect of how it affects the uncertainty of a probabilistic model. Additionally, the concept of expected improvement was introduced which is a kind of Bayesian optimization. It was shown that this optimization technique is especially very efficient in combination with a non-stationary probabilistic model like $\mathcal{DGCN}$.

This has been proven in a benchmark of several different test functions compared to standard Gaussian processes. The idea of the expected improvement has been reused in order to optimize the uncertainty of the $\mathcal{DGCN}$ model, to obtain a global accurate model. This method represents a very efficient way to either perform optimization or to obtain a generally accurate model, since it can use one-by-one adaptation.

- Section 3.5 presented a topic which is not directly related to the classical definition of design of experiments. In this section methods were provided which allow samples to be created according to a given multivariate distribution and even multivariate correlations. This is useful for parameter studies and sensitivity analysis as presented in chapter 4. A novel rejection-sampling-based algorithm was presented which can reproduce every kind of multivariate distribution and correlation. It can even create samples if the sample values for one or more parameters are already given, as needed for the sensitivity analysis with correlated input parameters, presented in section 4.4. The main reason for using such an algorithm is that an extrapolation of the surrogate model in design areas, where no points are given, should be avoided. Typically, this is needed for applications where a mathematical constraint cannot be formulated. The algorithm utilizes an unsupervised machine learning algorithm which belongs to the category of novelty detection. As mentioned in this section, novelty detection can also be used as a constraint within optimization tasks for the already mentioned reason of extrapolation.

Chapter 4

- Section 4.1 to 4.3 defined different kinds of dependencies and gave an overview of two common methods for sensitivity analysis. Correlation coefficients can be used especially as pre-variable selection before surrogate model training. Here, one contribution is the comparison of several different coefficients and their capabilities to detect the defined different types of dependencies. The other method presented is the so-called coefficient of importance and is based on the change of prognosis measure of a surrogate model if an input parameter is left out. This method is only useable effectively with a model like the RSM, since it requires several retrains of the model dependent on the number of input parameters.

- Section 4.4 introduced the global-variance-based sensitivity analysis which estimates the so-called Sobol indices. Here an overview of existing equations was given to estimate those and also how to create the design matrices needed. Since this method relies on many additional design evaluations which follow a specific scheme, it is only usable efficiently in connection with surrogate models to perform the design evaluations. Furthermore, works were presented which showed how to decompose the Sobol indices in independent and correlated contributions. Here a novel way to do this based on the rejection sampling algorithm (with some modifications) was presented. Compared to existing methods, this method has no need for data transformation nor does it need to be only applicable for linear correlations. Thus, it has no limitations in its usage and is therefore much more useful in real world applications. The comparisons showed that there are some differences in the estimated Sobol values between the novel and existing methods which should be analyzed in future research.

- Section 4.5 finally presents another global sensitivity measure based on partial derivatives averaged over the whole design space. It is pointed out that this measure can be used, for example, to visualize sensitivities on field outputs like structures, based on multi-output $\mathcal{DGCN}$ models.

5.2 FUTURE WORK

As usual, there is always room for improvement and further research. Many of the problems tackled in this thesis came from real world applications with which the author was confronted during the time of the thesis. In the future, problems will also arise which cannot be solved with the algorithms developed and therefore reveal further topics of research. In spite of this, some suggestions can already be made. Several of them have been already mentioned throughout this work.

Big data. As presented in section 2.10, the prediction of the $\mathcal{DGCN}$ model utilizes a nearest neighbor search if trained via mini-batch. This means that for every point to be predicted, a set of nearest neighbors equal to the batch size has to be searched for, and then all the vectors and matrices needed for prediction have to be recalculated. This makes the prediction not vectorized and therefore numerically expensive. Parallelized prediction is one solution to speeding things up, but the prediction of millions of points might take longer than the training in certain circumstances. Thus, this is one major topic for future work, namely, how to improve prediction speed if the model is trained via batches. Furthermore, it would be interesting to test the scalability of the algorithm on multiple CPUs / GPUs. The topic of data compression has also been dealt with very briefly, further approaches, possibly based on autoencoders, might deliver significantly better performance compared to the principle component analysis.

Non-Gaussian likelihoods. Throughout this work, the Gaussian likelihood has always been used for the $\mathcal{DGCN}$ model. Extension to non-Gaussian likelihoods could enable the application of the model to new domains.

Covariance functions. Additional covariance functions in the $\mathcal{DGCN}$ model might also be beneficial dependent on the problem, for example a periodic one. Another idea would be to only use the Matérn covariance function class and optimize the smoothness parameter, maybe even dependent on the point to predict. Another topic related to this idea is to predict directly the covariance values of the covariance matrix by a neural network so that no covariance function is needed. Therefore, some regularization must be included so that the matrix will still be invertible. Sigmoid activation function might be useful in the output layer in order to obtain values between zero and one.

Ensembles ensembles of experts (Jacobs et al. [1991]) is a challenge winning approach (Hoch [2015], Zou et al. [2017]) to further increase the predictive power of a model by combining it with several other models. It would make sense to do research in this area with regard to meaningful combination possibilities with $\mathcal{DGCN}$. It would also be conceivable to set up an ensemble with quickly analyzable numerical models. For this purpose, appropriate examples would be necessary.

Net topologies. As mentioned, all experiments presented used the same net topology within the $\mathcal{DGCN}$ model. An efficient algorithm to optimize this topology would also be very interesting and might further improve the accuracy of the model. As briefly tested, a separate DNN_l for each covariance function might also be an aspect to cover more

intensively in future research. A first start could be the Neuroevolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen [2002b]) algorithm.

Sensitivity analysis for correlated inputs. The novel equations to estimate the independent and correlated first and total order Sobol indices presented in section 4.4 showed different results for some test cases than the other works. The reason for this was not fully analyzed and might therefore also be part of future investigations. Next to this, the topic of pre-variable selection is also very interesting. A way to efficiently find the unimportant parameters so that the predictive power of the surrogate model used will not be reduced, is still an open topic. As demonstrated, all coefficients need a certain number of samples in order to detect important and unimportant parameters correctly. The next question is how to determine a threshold in order to decide whether a parameter is unimportant or not. This might even be problem specific. A possible solution to this would be to create parameter subsets based on the correlation coefficient values, create surrogate models for each of them and compare some prognosis metric. This is only applicable if the model training does not need a lot of time and the number of subsets is not too big.

Applications. Several of the examples used in this thesis came from the mechanical engineering area or are more or less academic. It would be interesting to see, how the algorithms developed, work in other domains like medicine or finance. One example from civil engineering, is the usage of $\mathcal{DGCN}$ for structural anomaly detection using seismic measurements (Trapp et al. [2019]). In this work, a multi-output $\mathcal{DGCN}$ provided good results in detecting hard obstacles beforehand, in order to avoid damaging the tunnel boring machine and to adjust the propagation velocity.

Inverse problems. One example showed the calibration of a FEM simulation model. Therefore, the input parameters are searched for a given output signal. In future work, it might be beneficial to investigate the possibilities to directly train the inverse surrogate model and determine how a probabilistic model can be advantageous, since typically there is always more than one solution for such a problem. Maybe a confidence interval can give a bandwidth of possible solutions for the inverse problem. It would speed things up if the model were able to directly predict this bandwidth for each input parameter instead of intensive optimization.

BIBLIOGRAPHY

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. [Online; accessed 20-March-2020].

H. A. Abbass, R. Sarker, and C. Newton. Pde: a Pareto-frontier differential evolution approach for multi-objective optimization problems. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 2, pages 971–978. IEEE, 2001.

M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, 1964.

D. H. Ackley. A connectionist algorithm for genetic search. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 121–135. L. Erlbaum Associates Inc., 1985.

C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

O. Aghazadeh, J. Sullivan, and S. Carlsson. Novelty detection from an ego-centric perspective. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3297–3304. IEEE, 2011.

M. Ahmadi, F. Vahabzadeh, B. Bonakdarpour, E. Mofarrah, and M. Mehranian. Application of the central composite design and response surface methodology to the advanced treatment of olive oil processing wastewater using fenton's peroxidation. *Journal of Hazardous Materials*, 123(1-3):187–195, 2005.

S.-J. Ahn, J. Yoo, B.-G. Lee, and J.-J. Lee. 3d surface reconstruction from scattered data using moving least square method. *Image Analysis and Processing–ICIAP 2005*, pages 719–726, 2005.

D. Allard, J.-P. Chilès, and P. Delfiner. Geostatistics: Modeling spatial uncertainty, 2013.

E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

M. Álvarez, D. Luengo, M. Titsias, and N. Lawrence. Efficient multioutput Gaussian processes through variational inducing kernels. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 25–32, 2010.

S. Arlot, A. Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

D. Atabay, 2017. URL https://pyrenn.readthedocs.io/en/latest/examples.html. [Online; accessed 20-March-2020].

P. Audze. New approach to planning out of experiments. *Problems of dynamics and strengths*, 35:104–107, 1977.

S. Banerjee, A. E. Gelfand, A. O. Finley, and H. Sang. Gaussian predictive process models for large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(4):825–848, 2008.

N. Banupriya, S. Saranya, R. Swaminathan, S. Harikurmar, and S. Palanisamy. Animal detection using deepl learning algorithm. *Journal of Critical Reviews*, 7(1):434–439, 2020.

H. B. Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

P. Barpaga. A gentle introduction to machine learning, June 2019. URL https://towardsdatascience.com/a-gentle-introduction-to-machine-learning-599210ec34ad. [Online; accessed 20-March-2020].

S. Bates, J. Sienz, and V. Toropov. Formulation of the optimal Latin hypercube design of experiments using a permutation genetic algorithm. In *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, page 2011, 2004.

M. Bayes and M. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. Bayes, frs communicated by mr. Price, in a letter to John Canton, amfrs. *Philosophical Transactions (1683-1775)*, pages 370–418, 1763.

R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

J. Bernardo, J. Berger, A. Dawid, A. Smith, et al. Regression and classification using Gaussian process priors. *Bayesian statistics*, 6:475, 1998.

S. Bin and Y. Wenlai. Application of Gaussian process regression to prediction of thermal comfort index. In *Electronic Measurement & Instruments (ICEMI), 2013 IEEE 11th International Conference on*, volume 2, pages 958–961. IEEE, 2013.

K. Binder, D. Heermann, L. Roelofs, A. J. Mallinckrodt, and S. McKay. Monte Carlo simulation in statistical physics. *Computers in Physics*, 7(2):156–157, 1993.

M. Binois, D. Ginsbourger, and O. Roustant. A warped kernel improving robustness in Bayesian optimization via random embeddings. In *International Conference on Learning and Intelligent Optimization*, pages 281–286. Springer, 2015.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Training*, 7 (1), 2008.

C. Bogoclu and D. Roos. A benchmark of contemporary metamodeling algorithms. In *Proceedings of the ECCOMAS Conference*, 2016.

D. M. Borth. A total entropy criterion for the dual problem of model discrimination and parameter estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 77–87, 1975.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

G. E. Box and D. W. Behnken. Some new three level designs for the study of quantitative variables. *Technometrics*, 2(4):455–475, 1960.

G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

G. E. Box and W. J. Hill. Discrimination among mechanistic models. *Technometrics*, 9(1): 57–71, 1967.

G. E. Box and H. Lucas. Design of experiments in non-linear situations. *Biometrika*, 46 (1/2):77–90, 1959.

G. E. Box, W. G. Hunter, J. S. Hunter, et al. *Statistics for experimenters*. John Wiley and sons New York, 1978.

G. E. Box, N. R. Draper, et al. *Empirical model-building and response surfaces*, volume 424. Wiley New York, 1987.

G. E. Box, J. S. Hunter, and W. G. Hunter. *Statistics for experimenters: design, innovation, and discovery*, volume 2. Wiley-Interscience New York, 2005.

R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, 14(4):422–425, 1971.

M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

L. Brevault, M. Balesdent, N. Bérend, and R. Le Riche. Comparison of different global sensitivity analysis methods for aerospace vehicle optimal design. In *10th World Congress on Structural and Multidisciplinary Optimization, WCSMO-10*, 2013.

C. Bucher. *Computational analysis of randomness in structural mechanics: Structures and infrastructures book series, Vol. 3*. Crc Press, 2009.

T. Bui, D. Hernández-Lobato, J. Hernandez-Lobato, Y. Li, and R. Turner. Deep Gaussian processes for regression using approximate expectation propagation. In *International Conference on Machine Learning*, pages 1472–1481, 2016.

T. D. Bui, J. M. Hernández-Lobato, Y. Li, D. Hernández-Lobato, and R. E. Turner. Training deep Gaussian processes using stochastic expectation propagation and probabilistic backpropagation. *arXiv preprint arXiv:1511.03405*, 2015.

E. Burnaev and M. Panov. Adaptive design of experiments based on Gaussian processes. In *International Symposium on Statistical Learning and Data Sciences*, pages 116–125. Springer, 2015.

R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth. Manifold Gaussian processes for regression. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 3338–3345. IEEE, 2016.

R. Caruana, S. Lawrence, and C. L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

D. Chafekar, J. Xuan, and K. Rasheed. Constrained multi-objective optimization using steady state genetic algorithms. In *Genetic and Evolutionary Computation Conference*, pages 813–824. Springer, 2003.

K. Chaloner and R. Brant. A Bayesian approach to outlier detection and residual analysis. *Biometrika*, 75(4):651–659, 1988.

W. Chang. *Surface reconstruction from points*. Department of Computer Science and Engineering, University of California, San Diego, 2008.

O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system, 2020. URL https://github.com/dmlc/xgboost. [Online; accessed 20-March-2020].

X. Cheng, B. Khomtchouk, N. Matloff, and P. Mohanty. Polynomial regression as an alternative to neural nets. *arXiv preprint arXiv:1806.06850*, 2018.

V. Cherkassky and F. M. Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian treed models. *Machine Learning*, 48(1):299–320, 2002.

F. Chollet et al. Keras, 2015. URL https://keras.io. [Online; accessed 20-March-2020].

T. M. Cioppa and T. W. Lucas. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, 49(1):45–55, 2007.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

W. J. Conover and R. L. Iman. Rank transformations as a bridge between parametric and nonparametric statistics. *The American Statistician*, 35(3):124–129, 1981.

R. D. Cook and C. J. Nachtrheim. A comparison of algorithms for constructing exact D-optimal designs. *Technometrics*, 22(3):315–324, 1980.

D. Cornford, I. T. Nabney, and C. K. Williams. Modelling frontal discontinuities in wind fields. *Journal of nonparametric statistics*, 14(1-2):43–58, 2002.

P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

I. Couckuyt, D. Deschrijver, and T. Dhaene. Fast calculation of multi-objective probability of improvement and expected improvement criteria for Pareto optimization. *Journal of Global Optimization*, 60(3):575–594, 2014.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

K. Cremanns and D. Roos. A new optimized anisotropic moving least squares surrogate model with maximized prognosis. In *VII European Congress on Computational Methods in Applied Sciences and Engineering*, Crete Island, Greece, 05-10 June 2016.

K. Cremanns and D. Roos. Deep Gaussian covariance network. *arXiv preprint arXiv:1710.06202*, Oct. 2017.

K. Cremanns, D. Roos, S. Hecker, P. Dumstorff, H. Almstedt, and C. Musch. Efficient multi-objective optimization of labyrinth seal leakage in steam turbines based on hybrid surrogate models. In *Proceedings of ASME Turbo Expo 2016: Turbomachinery Technical Conference and Exposition*, volume 2C, Seoul, South Korea, 13-17 June 2016.

K. Cremanns, S. Hecker, A. Penkner, C. Musch, and D. Roos. Robust design optimization of a steam turbine labyrinth seal based on surrogate models. In *Proceedings of ASME Turbo Expo 2018: Turbomachinery Technical Conference and Exposition*, volume 8, Oslo, Norway, 11-15 June 2018a.

K. Cremanns, A. Penkner, S. Hecker, C. Musch, and D. Roos. Steam turbine exhaust optimization based on Gaussian covariance networks using transient CFD simulations. In *Proceedings of ASME Turbo Expo 2018: Turbomachinery Technical Conference and Exposition*, volume 8, Oslo, Norway, 11-15 June 2018b.

N. Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

K. Crombecq. *Surrogate modeling of computer experiments with sequential experimental design*. PhD thesis, Ghent University, 2011.

K. Crombecq, E. Laermans, and T. Dhaene. Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research*, 214(3):683–696, 2011.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

S. Da Veiga, F. Wahl, and F. Gamboa. Local polynomial estimation for sensitivity analysis on models with correlated inputs. *Technometrics*, 51(4):452–463, 2009.

Z. Dai, A. Damianou, J. González, and N. Lawrence. Variational auto-encoded deep Gaussian processes. *arXiv preprint arXiv:1511.06455*, 2015.

A. Damianou. *Deep Gaussian processes and variational propagation of uncertainty*. PhD thesis, University of Sheffield, 2015.

A. Damianou and N. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

A. C. Damianou, M. K. Titsias, and N. D. Lawrence. Variational inference for latent variables and uncertain inputs in Gaussian processes. *The Journal of Machine Learning Research*, 17(1):1425–1486, 2016.

R. De Maesschalck, D. Jouan-Rimbaud, and D. L. Massart. The Mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

P. B. de Oliveira Filho, M. L. F. Nascimento, and K. V. Pontes. Optimal design of a dividing wall column for the separation of aromatic mixtures using the response surface method. In *Computer Aided Chemical Engineering*, volume 43, pages 361–366. Elsevier, 2018.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

T. Dozat. Incorporating Nesterov momentum into Adam. *openreview*, 2016.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

P. F. Dunn and M. P. Davis. *Measurement and data analysis for engineering and science*. CRC press, 2017.

B. Echard, N. Gayton, and M. Lemaire. Ak-mcs: an active learning reliability method combining Kriging and Monte Carlo simulation. *Structural Safety*, 33(2):145–154, 2011.

B. Efron. Bootstrap methods: another look at the Jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.

B. Efron and G. Gong. A leisurely look at the bootstrap, the Jackknife, and cross-validation. *The American Statistician*, 37(1):36–48, 1983.

M. T. Emmerich, K. C. Giannakoglou, and B. Naujoks. Single-and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006.

K. Fang and A. Shenton. Constrained optimal test signal design for improved prediction error. *IEEE Transactions on Automation Science and Engineering*, 11(4):1191–1202, 2014.

K.-T. Fang, C.-X. Ma, and P. Winker. Centered l_2-discrepancy of random sampling and Latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, 71(237):275–296, 2002.

R. Farwig. Multivariate interpolation of arbitrarily spaced data by moving least squares methods. *Journal of computational and applied mathematics*, 16(1):79–93, 1986.

P. Feliot, J. Bect, and E. Vazquez. A Bayesian approach to constrained single-and multi-objective optimization. *Journal of Global Optimization*, 67(1-2):97–133, 2017.

G. C. Fernandez. Residual analysis and data transformations: important tools in statistical analysis. *HortScience*, 27(4):297–300, 1992.

S. C. Ferreira, R. Bruns, H. Ferreira, G. Matos, J. David, G. Brandao, E. P. da Silva, L. Portugal, P. Dos Reis, A. Souza, et al. Box-Behnken design: an alternative for the optimization of analytical methods. *Analytica chimica acta*, 597(2):179–186, 2007.

S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. In *ACM transactions on graphics (TOG)*, volume 24, pages 544–552. ACM, 2005.

A. Forrester, A. Keane, et al. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

M. Franz and P. Gehler. How to choose the covariance for Gaussian process regression independently of the basis. *In Proc. Gaussian Processes in Practice Workshop*, 2006.

P. I. Frazier and J. Wang. Bayesian optimization for materials design. In *Information Science for Materials Discovery and Design*, pages 45–75. Springer, 2016.

M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

G. M. Fung and O. L. Mangasarian. Multicategory proximal support vector machine classifiers. *Machine learning*, 59(1-2):77–97, 2005.

Y. Gal, M. Van Der Wilk, and C. E. Rasmussen. Distributed variational inference in sparse Gaussian process regression and latent variable models. In *Advances in Neural Information Processing Systems*, pages 3257–3265, 2014.

M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776, 2011.

M. N. Gibbs. *Bayesian Gaussian processes for regression and classification*. PhD thesis, Citeseer, 1998.

W. R. Gilks and P. Wild. Adaptive rejection sampling for Gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 22–30. Springer, 2004.

T. Goel, R. T. Haftka, W. Shyy, and N. V. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, 33(3):199–216, 2007.

P. W. Goldberg, C. K. Williams, and C. M. Bishop. Regression with input-dependent noise: A Gaussian process treatment. In *Advances in neural information processing systems*, pages 493–499, 1998.

J. González, Z. Dai, P. Hennig, and N. Lawrence. Batch Bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pages 648–657, 2016.

Google. Industries targeted by machine learning application developers. In Statista - The Statistics Portal. Retrieved July 1, 2019, from, 2016a. URL https://www.statista.com/statistics/602784/worldwide-machine-learning-industry-focus/. [Online; accessed 20-March-2020].

Google. If your organization is planning to use machine learning, what benefits are you seeking? In Statista - The Statistics Portal. Retrieved July 1, 2019, from, 2016b. URL https://www.statista.com/statistics/695586/worldwide-machine-learning-organizational-goals-planned-adopters/. [Online; accessed 20-March-2020].

D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11(Jul):2051–2055, 2010.

B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

R. B. Gramacy and H. K. Lee. Gaussian processes and limiting linear models. *Computational Statistics & Data Analysis*, 53(1):123–136, 2008.

A. Grosso, A. Jamali, and M. Locatelli. Finding maximin Latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, 197(2):541–547, 2009.

R. F. Gunst and R. L. Mason. Fractional factorial design. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):234–244, 2009.

Z. Han, R. Zimmerman, and S. Görtz. Alternative Co-Kriging method for variable-fidelity surrogate modeling. *AIAA journal*, 50(5):1205–1210, 2012.

Z.-H. Han, S. Görtz, and R. Zimmermann. Improving variable-fidelity surrogate modeling via gradient-enhanced Kriging and a generalized hybrid bridge function. *Aerospace Science and Technology*, 25(1):177–189, 2013.

A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *ISVC*, pages 867–877, 2015.

D. Harrison and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

T. Hastie, S. Rosset, J. Zhu, and H. Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

J. Hauke and T. Kossowski. Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87–93, 2011.

D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

M. Heinonen, H. Mannerström, J. Rousu, S. Kaski, and H. Lähdesmäki. Non-stationary Gaussian process regression with Hamiltonian Monte Carlo. In *Artificial Intelligence and Statistics*, pages 732–740, 2016.

J. Hensman and N. D. Lawrence. Nested variational compression in deep Gaussian processes. *arXiv preprint arXiv:1412.1370*, 2014.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

J. Hensman, A. G. d. G. Matthews, and Z. Ghahramani. Scalable variational Gaussian process classification. *Journal of Machine Learning Research*, 2015.

F. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of Computation of the American Mathematical Society*, 67(221):299–322, 1998.

D. Higdon. Space and space-time modeling using process convolutions. In *Quantitative methods for current environmental issues*, pages 37–56. Springer, 2002.

C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *arXiv preprint arXiv:1801.05894*, 2018.

D. M. Himmelblau. *Fault detection and diagnosis in chemical and petrochemical processes*, volume 8. Elsevier Science Ltd, 1978.

M. Hitsuda. Representation of Gaussian processes equivalent to Wiener process. *Osaka Journal of Mathematics*, 5(2):299–312, 1968.

T. Hoch. An ensemble learning approach for the Kaggle taxi travel time prediction challenge. In *DC@ PKDD/ECML*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

E. Hofer, M. Kloos, B. Krzykacz-Hausmann, J. Peschke, and M. Woltereck. An approximate epistemic uncertainty analysis approach in the presence of epistemic and aleatory uncertainties. *Reliability Engineering & System Safety*, 77(3):229–238, 2002.

T. Homma and A. Saltelli. Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering & System Safety*, 52(1):1–17, 1996.

S. Hora and R. Iman. A comparison of maximum/bounding and Bayesian Monte Carlo for fault tree uncertainty analysis. *Sandia Report SAND85-2839*, 1986.

I. Hupkens, M. Emmerich, and A. Deutz. Faster computation of expected hypervolume improvement. *arXiv preprint arXiv:1408.7114*, 2014.

B. G. M. Husslage et al. Maximin designs for computer experiments. Technical report, Tilburg University, School of Economics and Management, 2006.

R. L. Iman and W.-J. Conover. A distribution-free approach to inducing rank correlation among input variables. *Communications in Statistics-Simulation and Computation*, 11 (3):311–334, 1982.

R. L. Iman and J. C. Helton. An investigation of uncertainty and sensitivity analysis techniques for computer models. *Risk analysis*, 8(1):71–90, 1988.

R. L. Iman and S. C. Hora. A robust measure of uncertainty importance for use in fault tree system analysis. *Risk analysis*, 10(3):401–406, 1990.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

T. Ishigami and T. Homma. An importance quantification technique in uncertainty analysis for computer models. In *Uncertainty Modeling and Analysis, 1990. Proceedings., First International Symposium on*, pages 398–403. IEEE, 1990.

A. I. J. Forrester, A. J. Keane, and N. W. Bressloff. Design and analysis of noisy computer experiments. *AIAA journal*, 44(10):2331–2339, 2006.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

E. Janouchová and A. Kučerová. Competitive comparison of optimal designs of experiments for sampling-based sensitivity analysis. *Computers & Structures*, 124:47–60, 2013.

R. Jin, W. Chen, and A. Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. In *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 545–554. American Society of Mechanical Engineers, 2003.

M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990.

R. A. Johnson, D. W. Wichern, et al. *Applied multivariate statistical analysis*, volume 4. Prentice-Hall New Jersey, 2014.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/. [Online; accessed 20-March-2020].

V. R. Joseph and Y. Hung. Orthogonal-maximin Latin hypercube designs. *Statistica Sinica*, pages 171–186, 2008.

C.-F. Juang. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):997–1006, 2004.

A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

A. Keane and P. Nair. *Computational approaches for aerospace design: the pursuit of excellence*. John Wiley & Sons, 2005.

C. D. Keeling. Atmospheric co_2 records from sites in the SIO air sampling network. *Trends' 93: A compendium of data on global change*, pages 16–26, 1994.

M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Q. Y. Kenny, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000.

E. Keogh and A. Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard. Most likely heteroscedastic Gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400. ACM, 2007.

B.-S. Kim, Y.-B. Lee, and D.-H. Choi. Comparison study on the accuracy of metamodeling technique for non-convex functions. *Journal of Mechanical Science and Technology*, 23 (4):1175–1181, 2009.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

J. Kirschner, M. Mutnỳ, N. Hiller, R. Ischebeck, and A. Krause. Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces. *arXiv preprint arXiv:1902.03229*, 2019.

J. P. Kleijnen, S. M. Sanchez, T. W. Lucas, T. M. Cioppa, et al. *A user's guide to the brave new world of designing simulation experiments*. Tilburg University, 2003.

J. Koehler and A. Owen. 9 computer experiments. *Handbook of statistics*, 13:261–308, 1996.

R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

D. Koller, N. Friedman, S. Džeroski, C. Sutton, A. McCallum, A. Pfeffer, P. Abbeel, M.-F. Wong, D. Heckerman, C. Meek, et al. *Introduction to statistical relational learning*. MIT press, 2007.

J. Konečnỳ, J. Liu, P. Richtárik, and M. Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2015.

S. Konishi and G. Kitagawa. Generalised information criteria in model selection. *Biometrika*, 83(4):875–890, 1996.

P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3232–3237. IEEE, 2010.

S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

K. Krauth, E. V. Bonilla, K. Cutajar, and M. Filippone. AutoGP: Exploring the capabilities and limitations of Gaussian process models. *arXiv preprint arXiv:1610.05392*, 2016.

D. G. Krige. *A statistical approach to some mine valuation and allied problems on the Witwatersrand*. PhD thesis, University of the Witwatersrand, 1951.

A. Krishnamoorthy and D. Menon. Matrix inversion using Cholesky decomposition. In *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2013*, pages 70–72. IEEE, 2013.

T. Krishnamurthy. Comparison of response surface construction methods for derivative estimation using moving least squares, Kriging and radial basis functions. In *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, page 1821, 2005.

J. Kristensen, Y. Ling, I. Asher, and L. Wang. Expected-improvement-based methods for adaptive sampling in multi-objective optimization problems. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2016.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

A. Krizhevsky, V. Nair, and G. Hinton. The CIFAR-10 dataset, 2014. URL http://www.cs. toronto. edu/kriz/cifar. html. [Online; accessed 20-March-2020].

D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev. Why the Monte Carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6): 386–392, 2014.

S. Kucherenko, M. Rodriguez-Fernandez, C. Pantelides, and N. Shah. Monte Carlo evaluation of derivative-based global sensitivity measures. *Reliability Engineering & System Safety*, 94(7):1135–1148, 2009a.

S. Kucherenko, S. Tarantola, and P. Annoni. Estimation of global sensitivity indices for models with dependent variables. *Computer Physics Communications*, 183(4):937–946, 2012.

S. Kucherenko et al. Derivative based global sensitivity measures and their link with global sensitivity indices. *Mathematics and Computers in Simulation*, 79(10):3009–3017, 2009b.

S. Kucherenko et al. A new derivative based importance criterion for groups of variables and its link with the global sensitivity indices. *Computer Physics Communications*, 181 (7):1212–1217, 2010.

S. Kurogi, T. Ueno, and M. Sawa. Batch learning competitive associative net and its application to time series prediction. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 1591–1596. IEEE, 2004.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

K. F. Lam, H. Mui, and H. Yuen. A note on minimizing absolute percentage error in combined forecasts. *Computers & Operations Research*, 28(11):1141–1147, 2001.

M. Lamboni, H. Monod, and D. Makowski. Multivariate sensitivity analysis to measure global contribution of input factors in dynamic models. *Reliability Engineering & System Safety*, 96(4):450–459, 2011.

M. Lamboni, B. Iooss, A.-L. Popelin, and F. Gamboa. Derivative-based global sensitivity measures: general links with sobol indices and numerical tests. *Mathematics and Computers in Simulation*, 87:45–54, 2013.

P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158, 1981.

W. A. Larsen and S. J. McCleary. The use of partial residual plots in regression analysis. *Technometrics*, 14(3):781–790, 1972.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*, pages 329–336, 2004.

N. D. Lawrence. Learning for larger datasets with the Gaussian process latent variable model. In *Artificial Intelligence and Statistics*, pages 243–250, 2007.

Y. LeCun. The MNIST database of handwritten digits, 1998. URL http://yann. lecun. com/exdb/mnist/. [Online; accessed 20-March-2020].

C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics*, pages 464–472, 2016.

P. M. Lee. *Bayesian statistics*. Oxford University Press London:, 1989.

R. Lehmensiek, P. Meyer, and M. Müller. Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits.

International Journal of RF and Microwave Computer-Aided Engineering, 12(4):332–340, 2002.

A. Lendasse, E. Oja, O. Simula, and M. Verleysen. Time series prediction competition: The CATS benchmark. In *IJCNN'2004 proceedings - International Joint Conference on Neural Networks*, pages 1615–1620. IEEE, 25-29 July 2004.

B. Letham, R. Calandra, A. Rai, and E. Bakshy. Re-examining linear embeddings for high-dimensional Bayesian optimization. *arXiv preprint arXiv:2001.11659*, 2020.

D. Levin. The approximation power of moving least-squares. *Mathematics of Computation of the American Mathematical Society*, 67(224):1517–1531, 1998.

K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu. Improving one-class SVM for anomaly detection. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 5, pages 3077–3081. IEEE, 2003.

M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375, 2015.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. [Online; accessed 20-March-2020].

M. Liefvendahl and R. Stocki. A study on algorithms for optimization of Latin hypercubes. *Journal of statistical planning and inference*, 136(9):3231–3247, 2006.

D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, pages 986–1005, 1956.

F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

L. Lueckemeyer, K. Cremanns, and F. Hiss. Probabilistic methods in rotor dynamics of turbine-generator retrofit. In *NAFEMS World Congress*, Stockholm, Sweden, 11-14 June 2017.

D. J. MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

D. J. MacKay. Gaussian processes a replacement for supervised neural networks? *Cambridge University*, 1997.

D. J. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

D. J. MacKay and R. M. Neal. Automatic relevance determination for neural networks. In *Technical Report in preparation*. Cambridge University, 1994.

S. Mair and U. Brefeld. Distributed robust Gaussian process regression. *Knowledge and Information Systems*, pages 1–21, 2015.

G. G. Maisuradze, D. L. Thompson, A. F. Wagner, and M. Minkoff. Interpolating moving least-squares methods for fitting potential energy surfaces: Detailed analysis of one-dimensional applications. *The Journal of chemical physics*, 119(19):10002–10014, 2003.

T. A. Mara and S. Tarantola. Variance-based sensitivity indices for models with dependent inputs. *Reliability Engineering & System Safety*, 107:115–121, 2012.

T. A. Mara, S. Tarantola, and P. Annoni. Non-parametric methods for global sensitivity analysis of model output with dependent inputs. *Environmental modelling & software*, 72:173–183, 2015.

M. J. Marasigan. *LeafSnapPH: A Mobile Application for Identifying Leaf Samples of Philippine Plants*. PhD thesis, University of the Philippines Manila, 2016.

B. D. Marjavaara, T. S. Lundström, T. Goel, Y. Mack, and W. Shyy. Hydraulic turbine diffuser shape optimization by multiple surrogate model approximations of Pareto fronts. *Journal of fluids engineering*, 129(9):1228–1240, 2007.

M. Markou and S. Singh. Novelty detection: a review part 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003a.

M. Markou and S. Singh. Novelty detection: a review part 2: neural network based approaches. *Signal processing*, 83(12):2499–2521, 2003b.

J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

B. Matérn. *Spatial variation*, volume 36. Springer Science & Business Media, 2013.

G. Matheron. Principles of geostatistics. *Economic geology*, 58(8):1246–1266, 1963.

D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

M. McKay. Variance-based methods for assessing uncertainty importance in NUREG-1150 analyses. *Los Alamos National Laboratory LA-UR-96-2695*, 1996.

M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

A. I. McLeod. Kendall rank correlation and Mann-Kendall trend test. *R Package Kendall*, 2005.

B. Merry, J. Gain, and P. Marais. Moving least-squares reconstruction of large models with gpus. *IEEE transactions on visualization and computer graphics*, 20(2):249–261, 2013.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6):1087–1092, 1953.

S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In *Advances in neural information processing systems*, pages 536–542, 1999.

T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

S. J. Mildenhall. Correlation and aggregate loss distributions with an emphasis on the Iman-Conover method. *Report of the research working party on correlations and dependencies among all risk sources, Casualty Actuarial Society*, 2005.

J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

D. C. Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

R. Moriconi, M. P. Deisenroth, and K. Kumar. High-dimensional Bayesian optimization using low-dimensional feature spaces. *arXiv preprint arXiv:1902.10675*, 2019.

R. Moriconi, K. S. Kumar, and M. P. Deisenroth. High-dimensional Bayesian optimization with projections using quantile Gaussian processes. *Optimization Letters*, 14(1):51–64, 2020.

M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995.

T. Most. Variance-based sensitivity analysis in the presence of correlated input variables. In *Proc. 5th Int. Conf. Reliable Engineering Computing (REC), Brno, Czech Republic*, 2012.

T. Most and J. Will. Metamodel of optimal prognosis-an automatic approach for variable reduction and optimal metamodel selection. *Proc. Weimarer Optimierungs-und Stochastiktage*, 5:20–21, 2008.

T. Most, A. Saltelli, and M. Ratto. Estimating total effect sensitivity indices in the presence of correlated input variables. Technical report, Internal report, GRK 1462, Bauhaus-Universität Weimar, 2010.

H.-G. Muller, U. Stadtmuller, et al. Estimation of heteroscedasticity in regression analysis. *The Annals of Statistics*, 15(2):610–625, 1987.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

C. Musch, K. Cremanns, S. Hecker, and A. Penkner. Combined optimisation of the last stage and diffuser in a steam turbine using meta-models. In *Proceedings of 12th European Conference on Turbomachinery Fluid dynamics & Thermodynamics*, Stockholm, Sweden, 03-07, April 2017.

D. E. Myers. Matrix formulation of Co-Kriging. *Journal of the International Association for Mathematical Geology*, 14(3):249–257, 1982.

I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.

K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27, 1990.

A. Nataf. Determination des distribution don t les marges sont donnees. *Comptes Rendus de l Academie des Sciences*, 225:42–43, 1962.

R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. *arXiv preprint physics/9701026*, 1997.

R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

M. Newborn. *Kasparov versus Deep Blue: Computer chess comes of age*. Springer Science & Business Media, 2012.

A. Ng. Machine learning yearning: Technical strategy for ai engineers in the era of deep learning. *Retrieved online at https://www.mlyearning.org*, 2017. URL https://www.deeplearning.ai/machine-learning-yearning/. [Online; accessed 20-March-2020].

V. Nguyen, S. Rana, S. K. Gupta, C. Li, and S. Venkatesh. Budgeted batch Bayesian optimization. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1107–1112. IEEE, 2016.

V. Nguyen, S. Rana, S. Gupta, C. Li, and S. Venkatesh. Budgeted batch Bayesian optimization with unknown batch sizes. *arXiv preprint arXiv:1703.04842*, 2017.

C. Nicholson. A beginner's guide to lstms and recurrent neural networks, 2019. URL https://pathmind.com/wiki/lstm. [Online; accessed 20-March-2020].

J. E. Oakley and A. O'Hagan. Probabilistic sensitivity analysis of complex models: a Bayesian approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(3):751–769, 2004.

G. B. Orr and K.-R. Müller. *Neural networks: tricks of the trade*. Springer, 2003.

A. B. Owen. Controlling correlations in Latin hypercube samples. *Journal of the American Statistical Association*, 89(428):1517–1522, 1994.

B. Ozdemir and L. S. Davis. Scalable Gaussian processes for supervised hashing. *arXiv preprint arXiv:1604.07335*, 2016.

C. J. Paciorek and M. J. Schervish. Non-stationary covariance functions for Gaussian process regression. In *Advances in neural information processing systems*, pages 273–280, 2004.

J.-S. Park. Optimal Latin-hypercube designs for computer experiments. *Journal of statistical planning and inference*, 39(1):95–111, 1994.

E. Patelli, H. J. Pradlwarter, and G. I. Schuëller. Global sensitivity of structural variability by random sampling. *Computer Physics Communications*, 181(12):2072–2081, 2010.

K. Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

F. Perronnin, Z. Akata, Z. Harchaoui, and C. Schmid. Towards good practice in large-scale learning for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3482–3489. IEEE, 2012.

C. Plagemann, K. Kersting, and W. Burgard. Non-stationary Gaussian process regression using point estimates of local smoothness. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 204–219. Springer, 2008.

C. Poultney, S. Chopra, Y. L. Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

M. J. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

M. B. Priestley. *Spectral analysis and time series*, volume 1. Academic press London, 1981.

Y. A. Qi, T. P. Minka, R. W. Picard, and Z. Ghahramani. Predictive automatic relevance determination by expectation propagation. In *Proceedings of the twenty-first international conference on Machine learning*, page 85. ACM, 2004.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

P. Z. Qian. Nested Latin hypercube designs. *Biometrika*, 96(4):957–970, 2009.

J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J. A. Quinn and M. Sugiyama. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters*, 40:36–40, 2014.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

C. E. Rasmussen. *Evaluation of Gaussian processes and other methods for non-linear regression.* University of Toronto, 1999.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

G. Rennen, B. Husslage, E. R. Van Dam, and D. Den Hertog. Nested maximin Latin hypercube designs. *Structural and Multidisciplinary Optimization*, 41(3):371–395, 2010.

J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

P. Rodríguez, M. A. Bautista, J. Gonzalez, and S. Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

M. Rosenblatt. Remarks on a multivariate transformation. *The annals of mathematical statistics*, 23(3):470–472, 1952.

P. J. Ross and P. J. Ross. *Taguchi techniques for quality engineering: loss function, orthogonal experiments, parameter and tolerance design*. McGraw-Hill New York, 1988.

P. J. Rousseeuw and K. V. Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

A. Saltelli and S. Tarantola. On the relative importance of input factors in mathematical models: safety assessment for nuclear waste disposal. *Journal of the American Statistical Association*, 97(459):702–709, 2002.

A. Saltelli, T. Andres, and T. Homma. Sensitivity analysis of model output: an investigation of new techniques. *Computational statistics & data analysis*, 15(2):211–238, 1993.

A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259–270, 2010.

S. Sarkka, A. Vehtari, and J. Lampinen. Time series prediction by Kalman smoother with cross-validated noise density. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 1653–1657. IEEE, 2004.

I. Sato, H. Nishimura, and K. Yokoi. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*, 2015.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

A. M. Schmidt and A. O'Hagan. Bayesian inference for non-stationary spatial covariance structure via spatial deformations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(3):743–758, 2003.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pages 11–25, 1998.

D. W. Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

M. Seeger, C. Williams, and N. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics 9*, 2003.

I. Sekaj. Robust parallel genetic algorithms with re-initialisation. In *International Conference on Parallel Problem Solving from Nature*, pages 411–419. Springer, 2004.

W. W. Shannon, Claude Elwood. A mathematical theory of communication. *Bell System Technical Journal*, 27(4):623–666, 1948.

S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 683–690, 1991.

M. C. Shewry and H. P. Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14(2):165–170, 1987.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

T. W. Simpson, J. Poplinski, P. N. Koch, and J. K. Allen. Metamodels for computer-based engineering design: survey and recommendations. *Engineering with computers*, 17(2): 129–150, 2001.

A. Smola and S. Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32:34, 2008.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

E. Snelson and Z. Ghahramani. Local and global sparse Gaussian process approximations. In *Artificial Intelligence and Statistics*, pages 524–531, 2007.

E. Snelson and Z. Ghahramani. Variable noise and dimensionality reduction for sparse Gaussian processes. *arXiv preprint arXiv:1206.6873*, 2012.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

I. Sobol and S. Kucherenko. Derivative based global sensitivity measures. *Procedia-Social and Behavioral Sciences*, 2(6):7745–7746, 2010.

I. M. Sobol. Sensitivity estimates for nonlinear mathematical models. *Mathematical modelling and computational experiments*, 1(4):407–414, 1993.

A. Spagnol, R. Le Riche, and S. Da Veiga. Bayesian optimization in effective dimensions via kernel-based sensitivity indices. *13th International Conference on Applications of Statistics and Probability in Civil Engineering*, 2019.

C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002a.

K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, page 9, San Francisco, 2002b. Morgan Kaufmann.

J. M. Steele. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

M. Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

M. L. Stein. *Interpolation of spatial data: some theory for Kriging*. Springer Science & Business Media, 2012.

M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. In *Advances in neural information processing systems*, pages 3545–3553, 2014.

R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

M. Sugiyama. Active learning in approximately linear regression based on conditional expectation of generalization error. *Journal of Machine Learning Research*, 7(Jan):141–166, 2006.

S. Sundararajan and S. S. Keerthi. Predictive app roaches for choosing hyperparameters in Gaussian processes. In *Advances in neural information processing systems*, pages 631–637, 2000.

R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston. Random forest: a classification and regression tool for compound classification and QSAR modelling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

B. Szabó and I. Babuška. *Finite element analysis*. John Wiley & Sons, 1991.

G. J. Székely, M. L. Rizzo, N. K. Bakirov, et al. Measuring and testing dependence by correlation of distances. *The annals of statistics*, 35(6):2769–2794, 2007.

B. Tang. Orthogonal array-based Latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

G. R. Terrell and D. W. Scott. Variable kernel density estimation. *The Annals of Statistics*, pages 1236–1265, 1992.

S. Theodoridis and K. Koutroumbas. Pattern recognition. 2003. *Elsevier Inc*, 2009.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

M. Titsias and N. D. Lawrence. Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851, 2010.

M. Trapp, C. Bogoclu, T. Nestorović, and D. Roos. Intelligent optimization and machine learning algorithms for structural anomaly detection using seismic signals. *Mechanical Systems and Signal Processing*, 133:106250, Nov. 2019.

G. V. Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 3:306–307, 1979.

A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49: 560–567, 2012.

P. Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

B. A. Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*. Citeseer, 1993.

E. R. Van Dam, B. Husslage, D. Den Hertog, and H. Melissen. Maximin Latin hypercube designs in two dimensions. *Operations Research*, 55(1):158–169, 2007.

J. Vanhatalo, P. Jylänki, and A. Vehtari. Gaussian process regression with Student-t likelihood. In *Advances in Neural Information Processing Systems*, pages 1910–1918, 2009.

J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

A. Vehtari, A. Gelman, and J. Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and computing*, 27(5):1413–1432, 2017.

J. M. Ver Hoef and N. Cressie. Multivariable spatial prediction. *Mathematical Geology*, 25 (2):219–240, 1993.

F. A. Viana, G. Venter, and V. Balabanov. An algorithm for fast optimal Latin hypercube design of experiments. *International journal for numerical methods in engineering*, 82 (2):135–156, 2010.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11:3371–3408, December 2010.

O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

R. Von Mises. *Mathematical theory of probability and statistics*. Academic Press, 2014.

L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

J. Wang, S. C. Clark, E. Liu, and P. I. Frazier. Parallel Bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*, 2016a.

P. Wang. Residual plots for detecting nonlinearity in generalized linear models. *Technometrics*, 29(4):435–438, 1987.

Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016b.

R. L. Wasserstein and N. A. Lazar. The asa statement on p-values: context, process, and purpose, 2016.

P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

D. R. Wijaya, R. Sarno, and E. Zulaika. Information quality ratio as a novel metric for mother wavelet selection. *Chemometrics and Intelligent Laboratory Systems*, 160:59–71, 2017.

Wikipedia contributors. Correlation and dependence — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Correlation_and_dependence&oldid=935560828. [Online; accessed 20-March-2020].

C. K. Williams. Computing with infinite networks. In *Advances in neural information processing systems*, pages 295–301, 1997.

C. K. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

C. K. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520, 1996.

C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

D. P. Wipf and S. S. Nagarajan. A new view of automatic relevance determination. In *Advances in neural information processing systems*, pages 1625–1632, 2008.

S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Y. Xiao. Sample path properties of anisotropic Gaussian random fields. In *A minicourse on stochastic partial differential equations*, pages 145–212. Springer, 2009.

F. Xiong, Y. Xiong, W. Chen, and S. Yang. Optimizing Latin hypercube design for sequential sampling of computer experiments. *Engineering Optimization*, 41(8):793–810, 2009.

C. Xu and G. Z. Gertner. Uncertainty and sensitivity analysis for models with correlated parameters. *Reliability Engineering & System Safety*, 93(10):1563–1573, 2008.

W. Yang, N. Li, Y. Qi, W. Qardaji, S. McLaughlin, and P. McDaniel. Minimizing private data disclosures in the smart grid. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 415–427, 2012.

I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.

I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

Y. A. Youssef, Y. Beauchamp, and M. Thomas. Comparison of a full factorial experiment to fractional and taguchi designs in a lathe dry turning operation. *Computers & industrial engineering*, 27(1-4):59–62, 1994.

Y. Yuan, H. Xu, and B. Wang. An improved NSGA-III procedure for evolutionary many-objective optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 661–668. ACM, 2014.

M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

X. Zeng and T. R. Martinez. Distribution-balanced stratified cross-validation for accuracy estimation. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1):1–12, 2000.

Z. Zhan, Y. Fu, R.-J. Yang, Z. Xi, and L. Shi. A Bayesian inference based model interpolation and extrapolation. *SAE International journal of materials and manufacturing*, 5 (2):357–364, 2012.

Z. Zhou, Y. S. Ong, M. H. Nguyen, and D. Lim. A study on polynomial regression and Gaussian process global surrogate model in hierarchical surrogate-assisted evolutionary algorithm. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2832–2839. IEEE, 2005.

H. Zou, K. Xu, and J. Li. The YouTube-8M Kaggle competition: Challenges and methods. *arXiv preprint arXiv:1706.09274*, 2017.

Appendices

A

IMAN AND CONOVER RANK CORRELATION METHOD

A.1 MATHEMATICAL FORMULATION

A.1.1 *Declaration of Notation*

- C is the desired rank correlation matrix $(n \times n)$
- P is the lower triangular matrix of C
- I is the pre-existing rank correlation matrix of X
- Q is the lower triangular matrix of I
- n number of random variables
- N number of samples
- X is the sampling matrix composed of n independent random variables and N samples $(N \times n)$ with rank correlation matrix I and their distribution types.
- x_i variable for $i \in [1, ..., n]$
- X^* Sampling matrix $N \times n$ with desired rank correlation matrix C.
- S is the transformation matrix which maps X to X^*.

A.1.2 *Mathematical Formulation of the Rank Correlation Method*

As a remark at the beginning, since the linear correlation and rank correlation are typically close to each other, the re-ordered output will have approximately the desired linear (Pearson) correlation. Furthermore, the desired correlation matrix C must be a correlation matrix for linearly independent random variables, so it must be symmetric and positive definite (in case the Cholesky decomposition will be used).

1. The first step is to generate a matrix X of n random variables and N samples with a possible multivariate distribution.
2. Remove mean value and standard deviation of each variable x_i via:
$$\frac{\boldsymbol{x}_i - \mathbb{E}(x_i)}{\sigma(x_i)} \tag{A.1}$$
so that X is a standardized matrix with mean 0 and standard deviation 1.
3. Calculate I the pre-existing rank correlation matrix of X.
4. Calculate the lower triangular matrix P of the target rank correlation matrix C. For example, with Cholesky factorization $C = PP^T$ or with the Eigenvalue method.

5. Calculate the lower triangular matrix Q of I with $I = QQ^T$.

6. To obtain a sampling matrix X^* that has the desired rank correlation matrix C. A transformation matrix S is needed which is able to map the samplings from X to X^*. A correlation can be introduced to a sampling matrix by multiplying the sampling matrix with the corresponding correlation matrix. So S needs to be the rank correlation matrix which can be multiplied with X to obtain X^*. As a side note, if all variables are normally distributed and so the multivariate distribution is normal, it is possible to simply multiply with the desired correlation matrix which will not change the multivariate distribution. This even works for the Pearson correlation matrix.

7. Next, a transformation of the pre-existing rank correlation matrix I to the desired rank correlation matrix C is needed. This transformation is mathematically given through:
$$SIS^T = C \tag{A.2}$$
with S:
$$S = PQ^{-1} \tag{A.3}$$
which is the simple multiplication of the triangular matrices of the pre-existing rank correlation matrix I and the desired rank correlation matrix C.

8. Now it is possible to calculate the transformation of X to X^* which is the sampling matrix with desired correlation matrix C:
$$X^* = XS^T \tag{A.4}$$
Note that by multiplying with a matrix, here S^T, the values of the original sample points are changed and so the prescribed distribution types are destroyed. So, matrix X^* is just a support matrix to rearrange X so that is has the same rank correlation matrix as X^*.

9. To show that the rank correlation of X^* is equal to C:
$$\begin{aligned} RankCorr(X^*) &= P^T(Q^{-1})^T X^T X Q^{-1} P \\ &= P^T(Q^{-1})^T Q^T Q Q^{-1} P = P^T P = C \end{aligned} \tag{A.5}$$

10. Now it is necessary to ensure that the N sample values of each x_i in X have the same rank order as it has for the same x_i in X^*. That means, for example, that if x_1 the first sample value in X^* has the highest rank (which means the highest numerical value), then in X the first sample value of x_i should also have the highest rank. Therefore, an iteration over each variable x_i in X and X^* for $i \in [1, ...n]$ is needed in order to control that both sample matrices have the same rank order. The result is that X has the same rank correlation matrix as X^*, which is the desired rank correlation matrix C and it does not destroy the multivariate distribution type of each variable, because the values have not been changed as in X^*.

11. For the last step, the mean and standard deviations have to be applied back to X.

The accuracy increases with increasing the number of samples N.