Exploiting Spatio-Temporal Dependencies for RNN-based Wind Power Forecasts

Henning Wilms*, Marco Cupelli†, Antonello Monti‡
Automation of Complex Power Systems, RWTH Aachen University, Germany
Email: *hwilms@eonerc.rwth-aachen.de, †mcupelli@eonerc.rwth-aachen.de, ‡amonti@eonerc.rwth-aachen.de

Thorsten Gross
Avacon Netz GmbH, Salzgitter, Germany
Email: thorsten.gross@avacon.de

***Abstract*—Forecasting wind power production using machine learning techniques proves difficult i n p ractice due to the high stochasticity of the wind power time-series. To improve these forecasts, we include wind speed and wind direction from neighboring points in the vicinity of the wind turbine or wind farm in question. We assume that this additional information can improve the wind power forecast as the wind speed and direction at the wind turbine is influenced b y t he d istributed w ind s peeds a nd directions scattered over the topology.**

To make best use of these additional inputs, we propose using convolutional long-short term recurrent neural networks (convLSTM). Their advantage is the property of including temporal dependencies arising from the (wind) time-series as well as spatial dependencies obtained from geographically scattered wind forecasts. convLSTM shows promising results to modulate both temporal as well as spatial dependencies on wind power output time-series.

***Index Terms*—convolutional neural networks, recurrent neural networks, wind power forecasting**

I. Introduction

The increase of power generation from renewables places more and more stress on the electricity grid due to their volatility and uncertainty. This challenges grid hosting capacity and stability, possibly resulting in curtailment of larger wind farms. Forecasts of wind power outputs can help to solve this issue by enabling grid operators to better plan their operation thus allowing for further integration and hence expanding and utilizing of renewable power from wind farms [1], [2].

Forecasting wind power production however proves difficult d ue t o t he h igh s tochasticity o f t he time-series. Wind time-series tend to have a short auto-correlation and seasonality that could be exploited using classical time-series analysis approaches. This calls for multivariate regression techniques that incorporate exogenous, causal information into the forecasting algorithm [3]. The most important causal inputs are the wind speed and wind direction directly at the location of a wind turbine or wind park.

Recurrent neural networks (RNNs) using long-short term memory cells (LSTM) show promising results for time-series forecasting related to the power systems domain, e.g. [1], [4], [5]. They can effectively capture time-series dynamics through their self-feeding connection and map the influence of exogenous variables on a specific time step (multivariate regressions) as well as the influence of previous time steps on the current time step (univariate regression).

The standard RNN approaches include so-called cells or memory units to tackle the issue of exploding or vanishing gradients for deep architectures [6], [7]. These cells use fully-connected, feed-forward operations as in standard artificial feed-forward neural networks (FFNN) to control the state of internal memory vectors as well as the state-to-state transitions [8], [9]. In this paper, we replace these feed-forward operations through convolutional operations [9]. Convolutional neural networks (CNN) are widely applied for image recognition. In the image recognition domain, their strength lies in capturing the structures and relationships within the combination of different pixels or sets of pixels, e.g. [8], [10]. Thus, they modulate spatial dependencies between these pixels.

We aim at combining the strength of CNN's spatial with RNN's temporal mapping capabilities for improved wind power output forecasts using convolutional LSTM (convLSTM) cells. In analogy to pixels in image recognition, the different wind forecasts from different weather stations or a set of scattered wind farms will provide the spatial dimension, whereas the power output time-series will be the temporal dimension. In this paper we use a open source wind data set to assess and compare convLSTM capabilities.

II. Spatio-Temporal Modeling

A. Recurrent neural networks for sequence learning

A standard feed-forward neuron receives external inputs called *features*, denoted by x and infers and output y. Parameters of the neuron, i.e. *weights* $\mathbf{w}$ and *bias* $\mathbf{b}$ are adapted so that the output y is close to the target value called *label* $\hat{y}$. The output of such a neuron is calculated by

$$y = \sigma(\mathbf{w}\ x + \mathbf{b}) \tag{1}$$

DOI: 10.1109/GTDAsia.2019.8715887
Publisher version: https://ieeexplore.ieee.org/document/8715887

where σ is a nonlinear activation function, often of a sigmoid, tanh or exponential shape. The inputs x include any of the exogenous variables used to infer the output y. A neural network built up using these feed-forward neurons hence maps the influence of all the different input variables, also called *features*, onto the target value, i.e. the output.

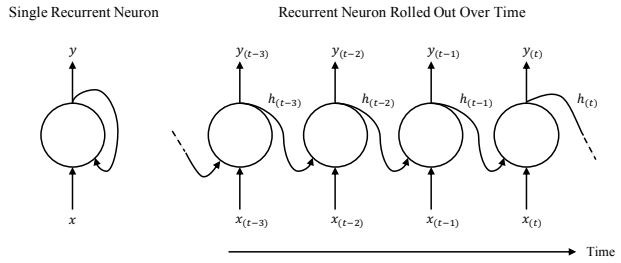

Fig. 1. Recurrent neuron over time

Recurrent neurons possess a self-feeding connection. This connection, called *hidden state* h (as depicted in Fig. 1), captures the dynamics of a time-series and can thus be exploited for time-series modeling when rolled out over time. A recurrent neuron is similar to the feed-forward neuron but includes the hidden state as additional input. It thus produces an output using the following computation:

$$y_t = \sigma(\mathbf{w}\ x_t + \mathbf{u}\ h_{t-1} + \mathbf{b}) \tag{2}$$

Here, $\mathbf{u}$ is another weight factor different from $\mathbf{w}$ to control the influence of h_{t-1}. The hidden state is calculated using the following equation and thus incorporates the recursiveness of RNNs

$$h_{t-1} = \sigma(\mathbf{w}\ x_{t-1} + \mathbf{u}\ h_{t-2} + \mathbf{b}) \tag{3}$$

All the previous computations are for a single neuron. In application these computation all are matrix computations when using batch computation and deep architectures, denoted through capital letters. Further, it is important to note that the parameters $\mathbf{w}$, $\mathbf{u}$ and $\mathbf{b}$ stay the same for the neuron for each time step. They are trained and adapted once to fit all time steps.

As mentioned in the introduction, using these neurons poses various difficulties when using deep architectures, i.e. more than one layer of neurons and / or many roll-out steps. This is due to the fact that the gradients in these deep architectures tend to vanish or explode, considerably impeding back-propagation and gradient descent algorithms (see III-B) for training purposes [7]. Cells have been introduced instead of neurons to tackle this issue [6], [11]. Their self-feeding connection consists of the identity function (derivative is always 1) effectively solving the problem of exploding or vanishing gradients by containing the gradient within the cell. The recursiveness is maintained by retaining, updating and accessing internal memory vectors, i.e. the *hidden states* of these cells from time step to time step. Two memory vectors are passed from time step to time step: the short-term state h and the long-term state c. These *hidden states* are updated via gate-controllers that in itself are neural networks. In standard LSTM cells these are fully connected feed-forward neural networks.

The memory vectors are updated by the gate controllers at each time step and then passed on to the next time step as hidden state. The operations for updating the hidden state using the *input gate* i, *forget gate* f and *output gate* o (see Fig. 2) are as follows [9], [11]:

$$i_t = \sigma(\mathbf{W}_{xi} \cdot x_t + \mathbf{W}_{hi} \cdot h_{t-1} + \mathbf{W}_{ci} \otimes c_{t-1} + \mathbf{b}_i) \tag{4}$$

$$f_t = \sigma(\mathbf{W}_{xf} \cdot x_t + \mathbf{W}_{hf} \cdot h_{t-1} + \mathbf{W}_{cf} \otimes c_{t-1} + \mathbf{b}_f) \tag{5}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \phi(\mathbf{W}_{xc} \cdot x_t + \mathbf{W}_{hc} \cdot h_{t-1} + \mathbf{b}_c) \tag{6}$$

$$o_t = \sigma(\mathbf{W}_{xo} \cdot x_t + \mathbf{W}_{ho} \cdot h_{t-1} + \mathbf{W}_{co} \otimes c_t + \mathbf{b}_o) \tag{7}$$

$$h_t = o_t \otimes \phi(c_t) \tag{8}$$

$\mathbf{W}$ denotes the different weight matrices. The indices indicate the different connections of each of the weight matrices. $\mathbf{b}$ denotes the bias terms for each of the fully connected layers. ϕ signifies a *tanh* as activation function, whereas σ indicates a sigmoid activation function. The $\otimes$-Symbol indicates a Hadamard product multiplication and $\cdot$ is a fully-connected, feed-forward computation.

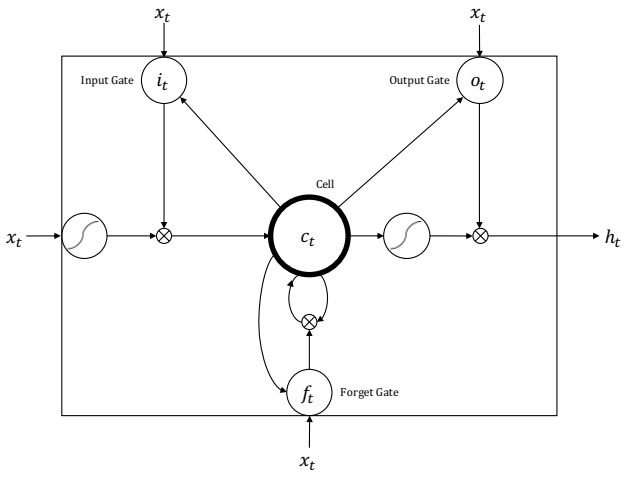

Fig. 2. LSTM cell with fully connected gate controllers

B. Convolutional neural networks for spatial modeling

CNNs have convolutional layers as main building block. In contrast to fully connected layers, the neurons of a convolutional layer are not connected to all the neurons of the previous layer (or inputs for the very first layer) but only to those neurons within their *receptive field* [8]. In Fig. 3 each neuron is depicted as a square that "looks" onto the previous layer of neurons, focusing only on those neurons in its receptive field. The receptive field is indicated by the square in the previous layer, in this case a 3x3 field. For convolutional layers the size of the receptive field is called either *filter* or *kernel size*, defined by its *height* h and *width* w.

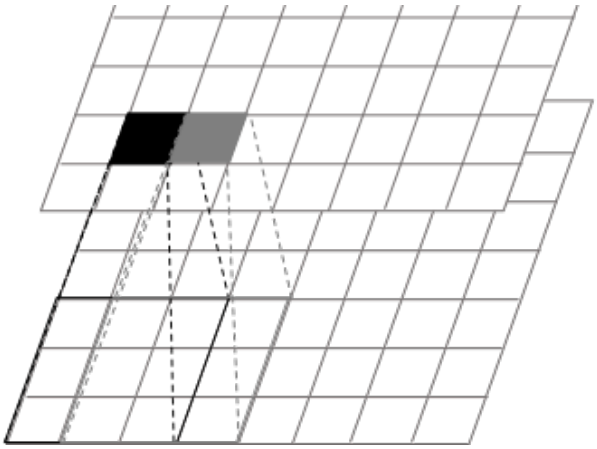

Fig. 3. Convolutional Layer and Receptive Field

When stacking various convolutional layers on top of each other, the size of each layer reduces. This is prevented by padding the margin such that each convolutional layer stays the same size. Usually a single convolutional layer can have several *feature maps* (or *channels* for the input and output layer). In this case, the following layer is connected to all the channels of the previous layer and are stacked on top of each other, i.e. each of the channels of the following layer "looks through" all of the channels of the previous layer [8].
The output z of a convolutional neuron is calculated using eq. 9 with the following notation [12]:

- $z_{i,j,k}$ is the output of the z neuron in the i-th row and j-th column in future map k of the layer l
- s_h and s_w are vertical and horizontal strides, i.e. movement of the the receptive field over the layer
- f_h and f_w are the the size of the receptive field.
- $f_{n'}$ denotes the number of feature maps in the layer $l-1$
- the output of the neuron of layer $l-1$ is identified by $x_{i',j',k'}$ at row i', column j' and feature map k'
- the bias $\mathbf{b}_k$ is the bias of layer l for feature map k of layer l
- all the neurons of a feature map k in layer l are connected with the previous layer $l-1$ neurons in row u and column v (u and v as rows and columns of the receptive field) and $l-1$ feature map k' through $\mathbf{w}_{u,v,k'k}$

$$z_{i,j,k} = \mathbf{b}_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \ \mathbf{w}_{u,v,k'k} \qquad (9)$$

$$with \begin{cases} i' = u \ s_h + f_h - 1 \\ j' = v \ s_w + f_w - 1 \end{cases}$$

By moving the filter over the feature maps or input channels of a convolutional layer, CNNs are capable of capturing spatial dependencies in the input space. Therefore, the input space must be shaped accordingly so that the spatial dependencies are represented in the input space, i.e. a 2D grid with a 3rd dimension as number of input channels must be chosen.

C. Convolutional long-short term memory cells

Convolutional LSTM (convLSTM) cells aim at marrying the two previously presented concepts to modulate spatial as well as temporal dependencies. The major drawback of standard LSTM cells for spatial modeling is their fully-connected calculations with only 2D input tensors, with one dimension representing the time dimension. ConvLSTM cells take 3D input tensors to modulate spatial dependencies at each time step and use the introduced convolutional operation to move a filter over the input grid and channels at each time step [9]. Thus convLSTM cells take 4D inputs: One dimension representing the time dimension, two dimensions for the spatial dimension and allocation within a spatial grid and the last dimension for accommodating more than one input variable per time step and spatial location. Both input-to-state as well as state-to-state (when connecting various layers into deep architectures) computations use this convolutional operation. The standard LSTM gate controllers introduced in eq. 4 to 8 are hence replaced by the following computations:

$$i_t = \sigma(\mathbf{W}_{xi} * x_t + \mathbf{W}_{hi} * h_{t-1} + \mathbf{W}_{ci} \otimes c_{t-1} + \mathbf{b}_i) \quad (10)$$

$$f_t = \sigma(\mathbf{W}_{xf} * x_t + \mathbf{W}_{hf} * h_{t-1} + \mathbf{W}_{cf} \otimes c_{t-1} + \mathbf{b}_f) \quad (11)$$

$$c_t = f_t \circ c_{t-1} + i_t \otimes \phi(\mathbf{W}_{xc} * x_t + \mathbf{W}_{hc} * h_{t-1} + \mathbf{b}_c) \quad (12)$$

$$o_t = \sigma(\mathbf{W}_{xo} * x_t + \mathbf{W}_{ho} * h_{t-1} + \mathbf{W}_{co} \otimes c_t + \mathbf{b}_o) \quad (13)$$

$$h_t = o_t \otimes \phi(c_t) \quad (14)$$

In eq. 10 to 14 $*$ indicates the convolutional operation that replaces the feed-forward computations (denoted as $\cdot$ in eq. 4 to 8) in standard LSTM cells.

III. Experimental Set-up

For evaluating the performance of the convLSTM approach for forecasting wind power output we use a publicly available wind data set from the Global Energy Forecasting Competition 2014 (GEFCom14) [13]. We first train a standard LSTM model as benchmark and then evaluate the convLSTM approach against that benchmark. The standard LSTM model follows the best practice implementation as described e.g. in [14]. For further comparison, we also present the performance of a random forest regressor (RF) as an alternative approach from the machine learning domain.

A. Data Set

In total, the GEFCom data set provides hourly readings of wind speed in $\vec{u}$ and $\vec{v}$ components and the power output for each of 10 different wind farms in Australia for a time span of 2 years. The documentation on the GEFCom data set does not provide the necessary information to conduct a spatial allocation within an input grid emulating the real world locations and dependencies. Therefore, the features are simply reshaped in their given order to form

an input grid of the shape $height = 4$, $width = 5$ and $channels = 2$ (see Fig. 4). Height, width and channels here use the same notation as used in chapter II-B for defining the convolutional operation. Each hourly input instance consists of four values for each of the 10 wind farms. Reedings at 10m and 100m height are separated into two different input channels, $\vec{u}$ and $\vec{v}$ at each height form a separate field, yielding the input shape of [4, 5, 2] (in height, width, channel notation). Spatial dependencies hence are provided when moving along the width axis and also when changing between the upper or lower half of the height axis. The resulting model is denoted as *convLSTM452*.

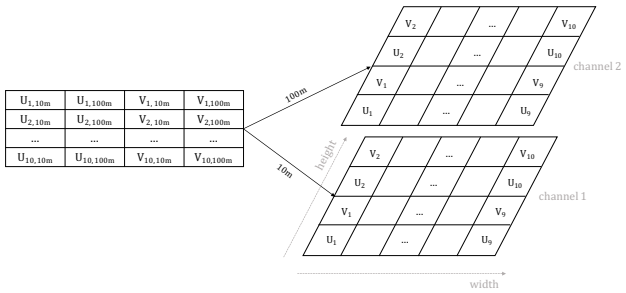

Fig. 4. Feature reshaping of the GEFCom data set into [4, 5, 2] shape

Next to the [4, 5, 2] reshaping, we further trained the convLSTM cells using input spaces reshaped into [5, 2, 4] and [2, 5, 4] shape, yielding the two additional models *convLSTM524* and *convLSTM254*. Here the $\vec{u}$ and $\vec{v}$ components at 10m and 100m each form one of four separate input channels and the spatial grid is laid out as a 5x2 or a 2x5 grid.

After the reshaping of each input instance, the input instances as well as the corresponding target values are sequentialized to form input sequences of 12 time steps for the temporal modulation. As the data set time resolution is hourly, the forecasting algorithms are trained to forecast sequences of 12 hourly time steps.

B. Training of the convLSTM and LSTM model

The convLSTM model and the LSTM model are both trained using the *Adam* optimizer utilizing *mini-batch stochastic gradient descent* and *back-propagation*. To ensure a fair comparability of the two approaches, we perform a *random search* over a reasonable search space adjusting the different *hyperparameters* of the models. Thus, we can assume that the finally evaluated models are very close to the best possible parametrization found within the search space for each of the models. This way, we compare the general capability of these different RNN types for the wind forecasting use case on a representative data sets. For a more elaborate description of neural network training we refer to [8] and [12].

For the convLSTM model we iterate over the parameter search space shown in Table I, the LSTM model uses the search space of Table II. The different hyperparameter meanings are as follows:

Batch size, *learning rate* and *gradient clip* are relevant parameters for the training set up. *l2 regularization* and *dropout keep probability* are regularization techniques preventing overfitting of the models. For further reading we refer to [8] and [12].

The *number of layer wise feature maps* provides separate lists containing the sequence of the number of feature maps over the depth, i.e. hidden layers of the model. For example, [2(4), 8] is a model with two layers, where the first layer (the input layer) possesses 2 stacked channels and the second layer (the hidden layer) has a feature map depth of 8. The last layer always reduces to one single output channel as final output. The 4 in brackets symbolizes the second option for the input shapes [2, 5, 4] and [5, 2, 4] that both have 4 initial input channels rather than 2. These hyperparameters hence describe the complexity and degrees of freedom of the convLSTM model.

Number of units, *hidden layers* and *bidirectionality* are the hyperparameters describing the complexity of the LSTM model, i.e. how many hidden layers contain how many units (LSTM cells per layer) and whether the model is set up bidirectionally (if set to true the model is doubled in complexity and the RNN is rolled out from both directions of the time-series sequence).

TABLE I
HYPERPARAMETERS AND SEARCH SPACE FOR CONVLSTM MODEL

Hyperparameter	**Search Space**
Number of layer wise feature maps	[2(4), 8], [2(4), 8, 16, 8], [2(4), 8, 16, 32, 16, 8], [2(4), 8, 16, 32, 64, 32, 16, 8], [2(4), 8, 16, 32, 64, 128, 64, 32, 16, 8]
Batch size	[40, 50, 70]
Learning rate	[0.001, 0.01]
L2 regularization	[0.01, 0.001, 0.0001, 0]
Gradient clip	[7, 10, 15, 20]
Kernel dimension 1	[2, 3, 4]
Kernel dimension 2	[2, 3, 4]

TABLE II
HYPERPARAMETERS AND SEARCH SPACE FOR LSTM MODEL

Hyperparameter	**Search Space**
Number of units	[10, 20, 30, 40]
Number of hidden layers	[1, 2, 3, 4]
Bidirectional	[True, False]
Batch size	[20, 30, 40]
Dropout keep probability	[0.6, 0.7, 0.8, 0.9, 1.0]
Learning rate	[0.001, 0.01, 0.1]
Gradient clip	[1, 3, 5, 10]

Besides the standard LSTM RNN, we also use a *random forest regressor* (RF) for comparison, following the best practice implementation as described in e.g. [12]. The random forest regressor is equally trained by iterating over a reasonable hyperparameter space (see Table III). The iterated hyperparameters in the search space for the random forest regressor all parametrize the degrees of freedom

and the complexity of the forecasting model. For further reading we refer to [12].

TABLE III
HYPERPARAMETERS AND SEARCH SPACE FOR RANDOM FOREST REGRESSOR

Hyperparameter	Search Space
Max depth	[3, 5, 7, 10]
Max features	[1, ... , 15]
Min samples leaf	[1, ... , 15]
Bootstrap	[True, False]
Number of estimators	[10, 50, 100]

Before initiating training, the data set is split 80:20 into separate training and evaluation data sets. The fitting of the model parameters and selection of hyperparameters are performed on the training data set, the performance of the forecasting models is evaluated and compared on the evaluation data set. This ensures the comparison of the models' generalization capabilities.

All the features within the data set are rescaled to the standard normal, i.e. for each feature j the mean is calculated and then subtracted from each of the entries before dividing the entry by it's variance (see eq. 15). This way the distribution of the feature is maintained while rescaling the features to the unit variance. We do this to make the best use of the sensitive interval of the used activation functions (i.e. derivative of activation function significantly $\neq 0$ within the interval).

$$x_{i,j,\ rescaled} = \frac{x_{i,j} - \bar{x_j}}{\sigma(x_j)} \tag{15}$$

With $x_{i,j}$ a single entry of an individual feature j, $\bar{x_j}$ the mean of all the features j and $\sigma(x_j)$ the variance over all the features j.

C. Evaluation

For evaluating all of the different forecasting approaches, we use the following set of accuracy measures. All the accuracy measures are calculated using the entire evaluation data set after training of the different models has been completed using the training data set.

The R^2*-accuracy measure*, also called *coefficient of determination* offers an unskewed, scale independent interpretation of the goodness of fit for each of the models. This goodness of fit describes the amount of variance in the target variable, that is described by the parametrized and trained model and the input data. A perfect fit yields a $R^2 = 1$ [15], [16].

The *root mean square error* (RMSE, eq. 16) is also calculated to provide an easier interpretation of the forecasting accuracy.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{16}$$

with *i* denoting each output y_i as well as label $\hat{y}_i$ for all the sequences of the evaluation data set comprising n instances.

For further interpretability, the *normalized RMSE* (nRMSE$_{\%}$, eq. 17) is calculated by dividing the RMSE by the overall range of all the labels (targets) in the data set $\hat{y}_{max} - \hat{y}_{min}$:

$$nRMSE_{\%} = \frac{RMSE}{\hat{y}_{max} - \hat{y}_{min}} \times 100 \tag{17}$$

IV. RESULTS

Table IV shows the results of the three different convLSTM models as well as the benchmarks. RMSE and nRMSE$_{\%}$ are both calculated using the same scale of the labels from $\hat{y}_{max} = 5.10$ to $\hat{y}_{min} = -4.1$ for better comparability.

TABLE IV
RESULTS OVERVIEW AND COMPARISON

Model	R^2	RMSE	nRMSE$_{\%}$
convLSTM452	0.6696	0.1986	2.16
convLSTM524	0.7588	0.1697	1.84
convLSTM254	0.7688	0.1661	1.81
Standard LSTM	0.6115	0.2156	2.34
Random Forests	0.7216	0.1823	1.98

V. CONCLUSION AND FURTHER RESEARCH

Table IV shows that the convLSTM524 and convLSTM254 clearly outperform the standard LSTM approach as well as the RF forecaster. All the convLSTM models outperform the standard LSTM model. We thus conclude, that the convLSTM cell is better suited to capture the particularities of spatio-temporal dependencies within wind power forecasts than the standard LSTM approach.

Input spaces convLSTM254 and convLSTM524 resemble the actual geographical formation in their 2x5 or 5x2 input grids and stack the different features in different channels. The two models are quite similar in accuracy and bot clearly outperform convLSTM452, whose input space does not resemble the spatial relationships between the inputs as well as the other two (see reshaping in Fig. 4). In a first indication, we suspect that the convLSTM cell must therefore be capable of finding patterns in spatial (as well as temporal) dependencies. We suspect that the performance of the convLSTM cell's forecasting accuracy can further be improved, the closer the geographical locations are resembled within the input space. As the GEFCom documentation does not provide the (relative) locations of the different wind farms, we assume that a better allocation of the different wind farms in the 5x2 or 2x5 input grid could further increase the performance of the forecasting algorithm.

Thus, when designing a convLSTM based forecasting algorithm for wind power forecasts, the input space should resemble the geographical locations of the different measurements. Comparing the convLSTM524 and convLSTM254 models with the convLSTM452, we deduce that different input features at each location should

rather be stacked in channels, than allocating them within the grid.

Training times for the 15 iterations including one cross-validations (i.e. a total of 30 different models) took, on average, less than 45 minutes per model on a standard 2.7 GHz i5 CPU. The best hyperparameters tended to have kernel sizes of 3 or 4 and used the 8 or 9 layer set up, expanding the number of feature maps to up to 128.

The convLSTM models may prove very useful in cases of no wind forecasts available at the direct location of a wind farm or wind turbine, but there are forecasts available in their near proximity. In these cases, convLSTM cells might be capable of learning the spatio-temporal dependencies of the forecasts in the vicinity and map these onto the location of wind turbine to produce reasonably accurate forecasts. This needs further research and suitable data sets.

ACKNOWLEDGMENT

We thank the European Commission for their funding of the InterFlex H2020 project under the grant agreement no 731289.

REFERENCES

[1] Z. Shi, H. Liang, and V. Dinavahi, "Direct interval forecast of uncertain wind power based on recurrent neural networks," *IEEE Transactions on Sustainable Energy*, vol. 9, no. 3, pp. 1177–1187, 2018.

[2] T. Gross, S. Reese, B. Petters, M. Cupelli, D. Mildt, and A. Monti, "A novel approach to dg curtailment in rural distribution networks – a case study of the avacon grid as part of the interflex field trial," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 2018, Conference Proceedings, pp. 667–672.

[3] H. Wilms, M. Cupelli, and A. Monti, "On the necessity of exogenous variables for load, pv and wind day-ahead forecasts using recurrent neural networks," in *2018 IEEE Electrical Power and Energy Conference (EPEC)*, 2018, Conference Proceedings.

[4] H. Wilms, M. Cupelli, and Monti, "Combining auto-regression with exogenous variables in sequence-to-sequence recurrent neural networks for short-term load forecasting," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, 2018, Conference Proceedings, pp. 673–679.

[5] I. Okumus and A. Dinler, "Current status of wind energy forecasting and a hybrid method for hourly predictions," *Energy Conversion and Management*, vol. 123, pp. 362–371, 2016.

[6] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998. [Online]. Available: https://doi.org/10.1142/S0218488598000094

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, Conference Proceedings, pp. 1310–1318.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[9] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *ArXiv e-prints*, vol. 1506, 2015. [Online]. Available: http://adsabs.harvard.edu/abs/2015arXiv150604214S

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *ArXiv e-prints*, vol. 1409, p. arXiv:1409.4842, 2014. [Online]. Available: http://adsabs.harvard.edu/abs/2014arXiv1409.4842S

[11] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, Conference Proceedings, pp. 850–855 vol.2.

[12] A. Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[13] T. Hong, P. Pinson, S. Fan, H. Zareipour, A. Troccoli, and R. J. Hyndman, "Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond," *International Journal of Forecasting*, vol. 32, no. 3, pp. 896–913, 2016.

[14] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE Transactions on Smart Grid*, vol. PP, no. 99, pp. 1–1, 2017.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[16] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.