

Machine Learning for Electronic Intelligence

Von der Fakultät für Elektrotechnik und Informationstechnik
der Rheinisch–Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von
Sabine Apfeld, M. Sc.
aus Köln

Berichter: Universitätsprofessor Dr.-Ing. Gerd Ascheid
Universitätsprofessor Dr.-Ing. Dirk Heberling
apl. Professor Dr. rer. nat. Wolfgang Koch

Tag der mündlichen Prüfung: 19.10.2021

Diese Dissertation ist auf den Internetseiten
der Universitätsbibliothek online verfügbar.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Gerd Ascheid for giving me the opportunity to write this thesis at RWTH Aachen University and for his valuable feedback throughout the preparation. I would also like to thank Prof. Dirk Heberling for his interest and for agreeing to review this work. A special thank you to my department head Prof. Wolfgang Koch from Fraunhofer FKIE for his constant enthusiasm and belief in me.

I am very grateful for the support I received from my research group *Sensor and Resources Management* at Fraunhofer FKIE. I would like to thank Dr. Alexander Charlish, Dr. Snezhana Jovanoska, Folker Hoffmann, and Hans Schily for many discussions and their critical questions. A very special thank you to Dr. Isabel Schlangen, who never got tired to provide helpful comments on my work. I really enjoyed our discussions about the formal definition of top k accuracy, passive voice, and hyphens. After more than one and a half years of working from home, I look forward to seeing all of you in person again when this pandemic is over!

Furthermore, I would like to thank the sponsor of the two projects that are the basis of my investigations. Being allowed to use the results helped a lot to reconcile this thesis and the project work conducted at Fraunhofer FKIE. I would also like to express my appreciation to Fraunhofer for giving me the opportunity to carry out this thesis at work and providing a supportive environment.

Last but not least, I would like to thank my husband, my parents, and my sister for being there for me in stressful times and bringing much needed distraction.

Abstract

Electronic intelligence is concerned with gathering information about radar emitters by intercepting and analysing their signals. By collecting this information, characteristic features are found that can be exploited to recognise known emitters. Traditional analysis and identification approaches rely on databases, which contain descriptions of the radars' operational modes. In conventional radar systems, modes are designed to fulfil a certain function and exhibit constant patterns of the waveform parameters. Agile multifunction radars, however, optimise their waveform parameters for the specific situation and therefore, they do not exhibit constant patterns. Consequently, traditional databases cannot effectively represent their emission characteristics. Moreover, they are unable to efficiently capture the relationships between emissions and hence, they cannot model the emitters' behaviour.

This thesis suggests a new emission model, which understands the radar emissions as a language with an inherent hierarchical structure of the five modelling levels *letters, syllables, words, commands, and functions*. Such an emission model allows exploiting machine learning methods, which are designed for natural language processing and in particular for the field of representation learning. Based on this approach, methods are developed for the four tasks of emission prediction, the identification of the emitter type, the learning of behavioural models, and the recognition of unknown emitters. To this end, predictive models are created that capture the behaviour of agile radar emitters. In addition, several architectures are investigated that combine multiple emitter models into an ensemble. Two contrasting approaches are compared for all four tasks throughout this thesis, namely the "memoryless" Markov chain and the Long Short-Term Memory recurrent neural network, which is designed to "remember" the past.

The hierarchical structure of the emission model significantly increases the performance of all considered tasks in comparison to traditional signal analysis methods. These process radar pulses, i.e. letters, based on which emitter identification and recognition of unknown emitters are not successful. Moreover, it is shown that machine learning methods provide large benefits in comparison to conventional approaches in the majority of the settings.

Zusammenfassung

Ziel der elektronischen Aufklärung ist das Zusammentragen von Informationen über Radaremitter durch die Erfassung und Analyse ihrer Signale. Mithilfe der gesammelten Informationen werden charakteristische Merkmale gefunden, anhand derer ein bereits bekannter Emitter identifiziert werden kann. Traditionelle Ansätze zur Analyse und Identifikation benötigen Datenbanken, die Beschreibungen der operationellen Modi der Radarsysteme enthalten. Bei herkömmlichen Radarsystemen sind solche Modi speziell für die Erfüllung einer Funktion ausgelegt und weisen konstante Muster in ihren Wellenform-Parametern auf. Agile Multifunktionsradare optimieren ihre Wellenform-Parameter hingegen für die vorgefundene Situation und zeigen daher keine konstanten Muster. Traditionelle Datenbanken können agile Emissionscharakteristiken aus diesem Grund nicht effektiv repräsentieren. Zusätzlich sind sie nicht in der Lage, die Beziehungen zwischen Emissionen effizient darzustellen und sind somit nicht für die Modellierung des Emitterverhaltens geeignet.

Die vorliegende Arbeit schlägt ein neues Emissionsmodell vor, das die Radaremissionen als eine Sprache mit einer inhärent hierarchischen Struktur begreift, bestehend aus den fünf Modellierungsebenen *Buchstaben*, *Silben*, *Wörter*, *Befehle* und *Funktionen*. Solch ein Emissionsmodell erlaubt es, die Vorteile von Methoden des maschinellen Lernens aus dem Gebiet der natürlichen Sprachverarbeitung im Allgemeinen und des Repräsentationslernens im Speziellen zu nutzen. Basierend auf diesem Ansatz werden in dieser Arbeit Methoden für die vier Aufgaben der Emissionsvorhersage, der Identifikation des Emittertyps, des Lernens von Verhaltensmodellen sowie der Erkennung unbekannter Emmitter entwickelt. Zu diesem Zweck werden Vorhersagemodelle erstellt, die das Verhalten von agilen Radaremittern darstellen können. Zusätzlich werden verschiedene Architekturen zur Kombination von mehreren Emmittermodellen zu einem Ensemble untersucht. Für alle vier Aufgaben werden zwei gegensätzliche Methoden betrachtet, die „gedächtnislose“ Markow-Kette und das Long Short-Term Memory rekurrente neuronale Netz, das speziell dafür entwickelt wurde, sich an die Vergangenheit zu „erinnern“.

Die hierarchische Struktur des Emissionsmodells steigert die Performanz für alle vier angeführten Aufgaben im Vergleich zu klassischen Methoden der Signalanalyse erheblich. Traditionell werden die Pulse der Radare, d.h. die Buchstaben, analysiert, anhand derer die Emmitteridentifikation und die Erkennung unbekannter Emmitter nicht gelingen. Zudem wird gezeigt, dass in der Mehrheit der Szenarien Methoden des maschinellen Lernens große Vorteile gegenüber konventionellen Ansätzen bieten.

Contents

1	Introduction	1
1.1	Scope and Related Work	2
1.2	Main Findings	4
1.3	Structure of the Thesis	5
1.4	Publications	5
1.5	General Notation	6
1.6	Copyright Notice	6
2	Electronic Intelligence	7
2.1	Radar Basics	7
2.1.1	Moving Target Indication & Pulse Doppler	8
2.1.2	Multifunction Radar	9
2.2	Traditional ELINT Processing Chain	9
2.2.1	Interception & Detection	9
2.2.2	Pulse Descriptor Word Extraction	10
2.2.3	Deinterleaving	11
2.2.4	Parameter Extraction	12
2.2.5	Database Lookup	14
2.3	Proposed ELINT Processing Chain	14
2.4	The Five Problems of Electronic Warfare	15
3	Modelling & Learning of Sequential Data	17
3.1	Formal Languages & Grammars	17
3.2	Finite State Machines	20
3.3	Petri Nets	22
3.4	Stochastic Processes	24
3.4.1	Markov Chains	24
3.4.2	Hidden Markov Models	25
3.4.3	Observable Operator Models	30
3.4.4	Predictive State Representations	34
3.5	Neural Networks	36
3.5.1	Activation & Output Functions	37
3.5.2	Error Functions	39

3.5.3	Training	39
3.5.4	Evaluation Metric	41
3.5.5	Recurrent Neural Networks	42
3.5.6	Long Short-Term Memory	43
3.5.7	Gated Recurrent Unit	45
4	Modelling of Agile Radar Emissions	47
4.1	Introduction	47
4.1.1	Hierarchical Emission Model	47
4.1.2	Word Embedding	48
4.1.3	Contributions	50
4.2	Adapted Hierarchical Emission Model	51
4.3	Modelling the Emissions of Example Emitters	52
4.4	Word Embedding for the Radar Language	55
4.5	Summary	56
5	Prediction of Radar Emissions	59
5.1	Introduction	59
5.1.1	Related Work	59
5.1.2	Contributions	60
5.2	Approaches	61
5.2.1	Long Short-Term Memory	62
5.2.2	Markov Chain	64
5.2.3	Comparison Methods	64
5.3	Experimental Results	64
5.3.1	Evaluation Under Ideal Conditions	65
5.3.2	Evaluation with Missing and Additional Symbols	68
5.3.3	Evaluation of the Impact of Input Encoding	71
5.4	Summary	72
6	Identification of the Radar Emitter Type	75
6.1	Introduction	75
6.1.1	Related Work	76
6.1.2	Contributions	77
6.2	Approaches	78
6.2.1	Long Short-Term Memory	79
6.2.2	Markov Chain	81
6.2.3	Comparison Methods	82
6.3	Experimental Results	82
6.3.1	Evaluation Under Ideal Conditions	83
6.3.2	Evaluation with Missing and Additional Symbols	87
6.4	Summary	91
7	Ensembles of Predictive Models	93
7.1	Introduction	93

7.1.1	Contributions	94
7.2	Ensemble Architectures	95
7.2.1	Mixture of Experts	95
7.2.2	Sparsely-Gated Mixture of Experts	95
7.2.3	Stacking (with Input)	96
7.2.4	Online Accuracy-Based Weighting	97
7.2.5	Model Averaging	97
7.3	Ensembles of Long Short-Term Memory Experts	98
7.3.1	Data	98
7.3.2	Implementation	100
7.3.3	Experimental Results	105
7.4	Ensembles of Predictive Radar Models	111
7.4.1	Implementation	111
7.4.2	Experimental Results	112
7.5	Summary	118
8	Recognition of Unknown Radar Emitters	121
8.1	Introduction	121
8.1.1	Related Work	122
8.1.2	Contributions	124
8.2	Training Cases	124
8.2.1	Generation of Known Unknown Emitters	125
8.3	Approaches	126
8.3.1	Long Short-Term Memory with Cross-Entropy Loss	128
8.3.2	Long Short-Term Memory with Entropic Open-Set Loss	128
8.3.3	Long Short-Term Memory with Deep Open Classification Loss	129
8.3.4	Long Short-Term Memory as Unknown Gate	130
8.3.5	Markov Chain	131
8.3.6	Markov Chain as Unknown Gate	131
8.4	Experimental Results	132
8.4.1	Evaluation Under Ideal Conditions	134
8.4.2	Evaluation with Missing and Additional Symbols	147
8.5	Summary	150
9	Conclusions	153
9.1	Future Work	155
A	Appendix: Additional Material	157
A.1	Identification of the Radar Emitter Type	157
A.2	Ensembles of Predictive Models	159
A.3	Recognition of Unknown Radar Emitters	173
	List of Acronyms	181
	List of Symbols	183

List of Figures	187
List of Tables	191
List of Publications	195
Bibliography	197

Chapter 1

Introduction

Electronic intelligence (ELINT), a part of the broader field of electronic warfare (EW), is concerned with collecting information about radar emitters by intercepting and analysing their signals. Therefore, “it is the remote sensing of remote sensors” [1, p. 1]. This is a completely passive process, in which the ELINT receiver does not produce any emissions and hence, it cannot be detected based on any own radiation. The gathered information is used to infer the capabilities of the radars, as well as to determine their type and purpose, which allows to estimate the threat that they pose. In addition, characteristic properties of the radars’ emissions are found such that they can be recognised if they are encountered again. This is especially important for devices like radar warning receivers, which are mounted onto platforms in order to warn about the presence of signals coming from hostile radars, e.g. missile guidance systems. Traditionally, the information about the radars is stored in databases. Each radar is therein described by its operational modes, which are characterised by constant patterns of the waveform parameters. Modes are predefined, hence not adaptive, and specially designed to fulfil a certain function, e.g. long-range search or track update. Moreover, the number of modes a radar can have is limited.

Since its invention, the way a radar operates has undergone major changes (see Figure 1.1). In the first radars, the operational mode was selected by a human operator and therefore, the emission parameters changed slowly. Afterwards, radars became semi-agile in the sense that the mode was selected by a software and hence, fast switching between different emission patterns became possible. Nowadays, agile multifunction radar systems are employed more often. This kind of radar does not have operational modes any more, but chooses the waveform parameters adaptively and optimised for the encountered situation and task [2–6]. Moreover, with the introduction of active electronically scanned array (AESA) antennas [7,8], a mechanical rotation for steering the radar’s beam was replaced by electronic means. Therefore, modern radars are capable of almost instantaneously steering their beams into nearly

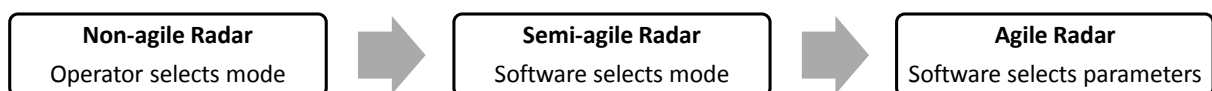


Figure 1.1: The evolution of radars from non-agile to agile. Operational modes constitute a set of predefined parameters, while agile radars can choose more flexibly.

any direction. These capabilities allow agile multifunction radars to perform many different tasks in a time-multiplexed way.

Agile radars pose several challenges to the traditional ELINT approaches. As their emissions do not show constant patterns, they cannot be effectively represented in traditional databases. In addition, databases are not able to efficiently capture the relationship between different waveform parameters and, as a consequence, are unable to model the radars' behaviour. However, many steps in the conventional ELINT processing chain depend on or benefit from the existence of database entries, e.g. emitter identification by a database lookup [1]. Consequently, there is a need for new representations of the signals and the behaviour of agile radars, as well as new methods for processing them.

In recent years, many successes have been achieved through machine learning approaches, which are defined in [9, p. 1] as algorithms that “automatically improve with experience”. They have become very powerful tools, also for tasks that seemed to be impossible for computers to solve. Consequently, machine learning is a promising approach for countering the challenges posed by agile radars. This thesis aims to investigate this direction, whereby mainly exploring two approaches. The first one is a Markov chain (MC) [10], which was described long before the term *machine learning* was introduced. Still, if the transition probabilities of an MC are estimated from data, it fulfils the requirement to improve with experience. The second approach is a Long Short-Term Memory (LSTM) neural network [11]. In contrast to the MC, which does not have a memory, the LSTM is specially designed to keep information about previous inputs in its internal state.

1.1 Scope and Related Work

This thesis considers the four tasks of learning behavioural emitter models, predicting the next emission of a radar, identifying its type based on the received signal, and recognising unknown emitters. Predictions facilitate several functions of an EW device, like sorting the incoming signals by emitter (deinterleaving) [1] or tracking an emitter in signal space, and are essential to other functions like generating tailored signal interferences to degrade the performance of the radar (jamming). As stated above, a reliable identification of the emitter type is crucial for the operation of a radar warning receiver.

The basis for all presented methods is to consider the radar emissions as a language that has an inherent hierarchical structure. The works [12–21] present and further develop this emission model. The radar language described therein contains the four levels *letters*, *words*, *commands*, and *tasks*. To improve the applicability for ELINT, this thesis extends the language by *syllables* and replaces the tasks by the more general category of *functions*. The definition of tasks by the previous work requires knowledge of the internal workings of the radar which cannot be obtained solely from intercepted emissions. An approach from the field of representation learning in natural language processing (NLP), called word embedding [22], is employed to learn

a dense vector representation of the radar language symbols, i.e. the different letters, syllables, words, commands, and functions. Several algorithms for learning these representations exist, e.g. [23–28], while this thesis focuses on the software package *word2vec* [23,24].

Throughout the thesis, the evaluations are performed with simulation data of an airborne multifunction radar that employs three different resource management methods of varying complexity. Since the resource management method highly influences the frequency and agility of the emissions, the radar is regarded as three individual emitters with the same vocabulary but a different grammar. The signals of the emitters are mapped to a stream of discrete symbols. In an actual application, several emitters are usually active at the same time and therefore, the received signals need to be deinterleaved. After this step, separated sequences are obtained, but no information about the emitters' identities is given. Consequently, the overall symbol stream consists of alternating data of several emitters. In the context of machine learning, this is referred to as streaming data with sudden, recurring concept drifts [29–33]. For the application considered in this thesis, a concept corresponds to an emitter.

For predicting the next emission, emitter models are developed using MCs and LSTMs. In the works on the hierarchical radar language [12–21], an emitter model is introduced, which is optimised to decode the internal state of the radar but cannot be used to predict the emissions. In contrast to these works, using machine learning approaches has the advantage that only data and no prior knowledge of the radar's behaviour is required. Other publications on predicting radar emissions are sparse and the emitters used for evaluation are less complex than in this thesis. The authors of [34] employ the hierarchical emission model of [12–21] in combination with predictive state representations (PSRs) [35]. Apart from this model, the work [36] proposes to use a type of recurrent neural network (RNN), a Gated Recurrent Unit (GRU) [37], to predict the next pulse, which corresponds to a letter in the emission model of this thesis. The radars used for evaluation can emit nine different words in [34] and at most 100 words (estimated from the description of the emitters) in [36]. In contrast, the example radars in this thesis use up to 34 440 words. By comparing MCs and LSTMs, the present work investigates whether memory is important for this task.

To identify the type of an emitter, MCs and LSTMs are employed as well. The identification with RNNs like LSTMs and GRUs is also considered in [36,38–41]. However, different representations of the signals are used, which makes the methods only partly comparable to those presented in this thesis. The authors of [42] employ the hierarchical radar language of [12–21] together with a hidden Markov model (HMM) [43]. The evaluation is performed at the word level with two radars, which can make use of the same six words. In contrast, this thesis provides results for all modelling levels and more complex emitters. In addition, it evaluates how the length of the intercepted signal, which maps to the number of consecutive symbols received from the same emitter, influences the identification accuracy.

Moreover, the predictive emitter models are combined into multi-model systems, which are also called ensembles [44]. Ensemble architectures employ two different

concepts in general, which are classifier fusion and classifier selection. In classifier fusion, the individual predictions of all models, which are also called experts in some approaches [45, 46], are combined to yield an overall system output [45, 47]. It was shown that fusing the output of models that fulfil the same task can increase the overall performance [44]. In classifier selection, only a subset of experts is chosen to make a prediction [46, 48, 49]. To increase the informative value, this thesis performs evaluations with a public data set [50] in addition to the combination of the predictive emitter models into an ensemble. No previous publications on ensembles of predictive radar models are available to the best of the author's knowledge. Therefore, this thesis investigates different general concepts [29–32, 45–47].

The present work extends the identification approach to recognise if an input does not belong to any of the known emitters. In the literature, detecting an unknown input is generally referred to as open-set or open-world recognition [51–56]. However, there is not much literature on recognising unknown emitters and it partly lacks details which prohibits a comparison [57–59]. Therefore, this thesis investigates six approaches from the general machine learning literature as a solution to this problem, whereby four of them are based on LSTMs and two on MCs. In addition to the conventional cross-entropy loss for training an LSTM, two loss functions are considered, which are specially designed for open-set recognition [54, 56]. A challenge in open-set recognition is that training data is only available for “known unknown” classes, of which the type and structure are known in advance. However, “unknown unknown” input is encountered only during the deployment of the system. Therefore, the classifiers are trained with the known unknown data, which should help them to also recognise unknown unknown input. In addition to the evaluation of different methods, this thesis presents methods to generate known unknown data.

1.2 Main Findings

By investigating these tasks and approaches, this thesis makes the following main findings:

- Machine learning approaches developed in the field of NLP can be employed for representing the signals of agile multifunction radars.
- Modelling the radar signals at different levels in a hierarchical language significantly facilitates the prediction of the multifunction radars' emissions. Moreover, this representation is what makes the identification of the emitter types and the recognition of unknown emitters possible.
- Machine learning approaches provide a large benefit compared to conventional methods for multiple tasks.
- An approach with an internal memory like the LSTM only has advantages over an approach without memory like the MC for more complicated tasks. In simpler settings, the less complex approach outperforms the method with memory.

- Ensembles of predictive emitter models increase the prediction accuracy if the ensemble architecture is chosen appropriately. Otherwise, the performance decreases by corrupting the internal state of the LSTM emitter models.

1.3 Structure of the Thesis

This thesis contains nine chapters, which describe the application of machine learning approaches in the field of ELINT. Chapter 2 provides an overview of the basics on radar and an introduction to ELINT. This chapter highlights the challenges introduced by agile radars into the traditional processing chain and suggests a suitable alternative to circumvent these challenges. Chapter 3 provides background information about methods for modelling sequential data like radar signals and compares them in terms of modelling power or expressiveness. In addition, the corresponding learning procedures are presented.

The main part of the thesis describes the hierarchical emission model, as well as the machine learning methods employed. Chapter 4 provides details on the emission model, which considers the radar emissions as a language with a hierarchical structure. Furthermore, it presents the applications of the emission model and an algorithm from the word2vec package on the signals of the example emitters. Chapter 5 introduces the predictive emitter models using MCs and LSTMs, which are trained to predict an emitter's next letter, syllable, word, command, or function. It gives the implementation details and information about the learning procedure. Moreover, it provides the evaluation of the methods, both with ideal and corrupted data, and shows the benefits of employing word embeddings. Chapter 6 presents methods to identify an emitter based on the symbols it uses. Again, LSTMs and MCs are compared. The chapter details the training of the LSTMs, which is conducted with different sequence lengths, i.e. numbers of consecutive symbols received from the same emitter. Additionally, this chapter provides the evaluation of the methods with ideal and corrupted symbol sequences. Chapter 7 gives an introduction to ensemble methods for predictive models processing streaming data in general and for predictive radar models in particular. It provides the implementation details and the evaluation of the approaches with the public dataset, as well as with the predictive emitter models. The latter are additionally evaluated with corrupted data. Chapter 8 introduces several approaches for recognising if an input does not belong to any of the known emitters. It gives the details on the generation of the known unknown data, as well as on the implementation of six methods, and presents the evaluation with ideal and corrupted data. Finally, Chapter 9 provides the summary and conclusions of the thesis, as well as possible directions for future work.

1.4 Publications

During the work on this thesis, the following publications were made or are currently under review. Parts of Chapter 4 have been published in [60], as well as parts of

Chapter 5. Chapter 6 has been partly published in [61]. The publication [62] presents content from Chapter 5 and Chapter 6. A paper on the content of Section 7.3 [63] has been accepted for publication and a paper on the content of Section 7.4 [64] is currently under review. The findings from Chapter 8 are published in [65]. In addition, the author of this thesis published the papers [66–68], which are related to the topic of this thesis but out of scope.

1.5 General Notation

In this thesis, the following general conventions are used. Vectors are denoted with bold lower case letters and matrices with bold capitals, e.g. \mathbf{a} or \mathbf{A} , respectively. The cardinality of a set A , i.e. the number of its elements, is written as $|A|$. Estimates are marked with a hat, e.g. \hat{a} . The letters i and j are used to index sequences, while time is indexed with the notation t .

1.6 Copyright Notice

Parts of this work have been published or accepted for publication in IEEE [60–63] © 2019/2020/2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Chapter 2

Electronic Intelligence

Electronic intelligence (ELINT) deals with the interception, detection, analysis, localisation, classification, and identification of radar signals. In contrast to communication intelligence (COMINT), signals used for communication are not of interest. The purpose of ELINT is to establish situational awareness by identifying signal characteristics which help in determining the functionality, capability, and hence the threat level of the emitter, and also to provide information which can be used to recognise an emitter if it is encountered again. ELINT belongs to the broader field of electronic warfare (EW), which additionally contains electronic attack (or countermeasures) and electronic protection (or counter-countermeasures).

This chapter gives a short introduction on radar. Afterwards, it describes details on the traditional ELINT processing chain for emitter identification. Finally, it presents the ELINT processing chain proposed in this thesis.

2.1 Radar Basics

Before describing the methods used by ELINT, the basics of radar operation are given. Only the parts relevant for understanding this thesis are discussed, for a more detailed description see e.g. [7].

Radio detection and ranging (radar) systems work by emitting electromagnetic energy, which is scattered by a target into many directions. If enough energy is reflected back to the radar's receiver, the target is detected. By measuring the time between the emission of the energy t_e and the reception of the echo t_r and knowing that electromagnetic signals travel with speed of light c , the distance or range to the target R is determined by

$$R = \frac{c(t_r - t_e)}{2}. \quad (2.1)$$

This equation is only strictly true in a vacuum. If the signals propagate through a medium like air, the speed reduces slightly [7].

Many radar systems work with pulses. The time between two pulses is called the pulse repetition interval (PRI). It determines the maximum unambiguous range R_u , which is the maximum range at which the echo from the target is received before a new pulse is emitted. It is given by

$$R_u = \frac{c \cdot \text{PRI}}{2}. \quad (2.2)$$

It is also common to characterise a waveform by the reciprocal of the PRI, which is the pulse repetition frequency (PRF),

$$\text{PRF} = \frac{1}{\text{PRI}}. \quad (2.3)$$

Echoes of pulses with the same PRI and radio frequency (RF) can be integrated over time to increase the detection probability. A repetition of pulses with identical parameters is called a burst or batch or, in the case of coherent integration with pulse-to-pulse phase coherence, a coherent processing interval (CPI). The amount of time a radar transmits pulses into the direction of a single target is called the dwell time.

The range resolution ΔR , which is the minimum distance between two targets such that they can be detected individually, depends on the signal bandwidth (BW) B_S ,

$$\Delta R = \frac{c}{2B_S}. \quad (2.4)$$

While the radar is emitting a pulse, its receiver is inactive. Therefore, echoes that arrive at the radar while it is sending cannot be received. This effect is called eclipsing and the chance of its occurrence increases with decreasing PRI.

2.1.1 Moving Target Indication & Pulse Doppler

Moving target indication (MTI) and pulse Doppler systems use the Doppler effect to discriminate between radar echoes returned by moving targets and those by static objects like buildings, mountains, or the ground, if the radar is airborne. Moreover, it is used to measure the radial velocity v_r between the target and the radar. The Doppler effect causes a frequency shift f_d , depending on the RF of the radar and v_r ,

$$f_d = \frac{2 \cdot v_r \cdot \text{RF}}{c}. \quad (2.5)$$

The maximum unambiguous velocity v_u a radar can measure depends on its PRI and RF,

$$v_u = \frac{c}{2 \cdot \text{PRI} \cdot \text{RF}}, \quad (2.6)$$

where it is assumed that the direction of the movement, i.e. receding or approaching, is known. Velocities that are equal to v_u and integer multiples of it are called blind speeds in MTI systems because they lead to a phase shift of zero and therefore, targets moving at these radial velocities are rejected as stationary. Unfortunately, the maximum unambiguous range R_u increases with increasing PRI whereas v_u increases with decreasing PRI. Therefore, the selection of the PRI always comes with a compromise between unambiguous range and unambiguous velocity measurement.

Traditionally, MTI and pulse Doppler systems were characterised by the way they performed the filtering of echoes coming from stationary targets [7]. However, in modern systems, the filtering methods became similar and therefore, a different definition is applied. According to [7], an MTI radar is a system that operates with a PRI high enough to provide unambiguous range measurements. On the other hand, pulse

Doppler radars use a low PRI that causes ambiguous range, but unambiguous velocity measurements. This categorisation can also be expressed in terms of low, medium, or high PRF systems [8]. Low PRF radars are characterised by unambiguous range measurements with PRFs between 0.1 kHz and 4 kHz. High PRF systems employ PRFs from about 30 kHz to 100 kHz and higher, while providing unambiguous velocity measurements. Everything in between belongs to the medium PRF regime. Of course, these values vary between systems as they depend on the furthest range and highest velocity of potential targets and should only serve as a rough reference.

2.1.2 Multifunction Radar

Directional radars concentrate the transmitted power into a specific direction, whereby the (main) beam is defined by the sector between the half-power points. With the employment of active electronically scanned array (AESA) antennas, radars gain the capability to almost instantaneously steer their beam into almost any direction. AESAs consist of many radiating elements, which are individually controllable in phase and amplitude [7]. By doing so, the radar beam can be pointed without mechanically moving the antenna.

In addition to the fast steering of AESAs, the rapid growth of computational power and miniaturisation of electronic devices make software controlled, agile multifunction radars possible. Based on the situation, the software of agile radars adaptively selects the waveform parameters. For example, if less energy is returned by a target, e.g. because it is far away, the number of pulses that are transmitted into the direction of the target is increased to increase the reflected energy. Moreover, blind speeds and eclipsing can be avoided by adaptively selecting the PRI and RF based on the target's range and relative velocity. As the software selects the waveform parameters adaptively, multifunction radars cannot be described in terms of predefined modes.

Sophisticated radar resource management methods assign the radar resources, e.g. time or energy, to different tasks. These can be highly adaptive and based on different parameters, like task priority and utility [6, 69]. For example, efficient radar resource management allows the radar to spend less energy and time on targets that follow a straight and therefore predictable trajectory and more on manoeuvring targets.

2.2 Traditional ELINT Processing Chain

Figure 2.1 shows the conventional processing chain for emitter identification. The individual steps are detailed in the following subsections.

2.2.1 Interception & Detection

The first step in the processing chain is to intercept and detect signals. An intercept occurs if the receiver is tuned to the RF band of the emitter and the emitter is transmitting into the direction of the receiver. If the ELINT receiver's antenna is not omni-directional, it must be directed towards the emitter. If the received signal strength is sufficiently large, the signal is detected.

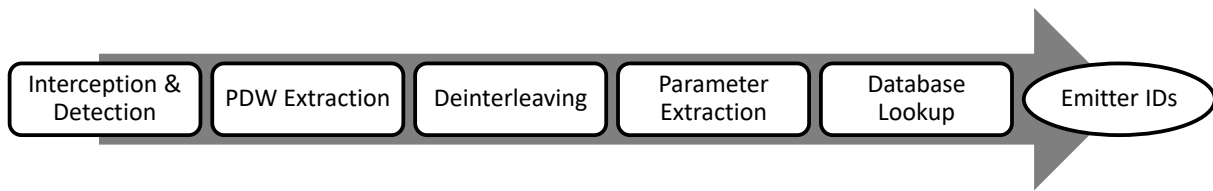


Figure 2.1: The traditional ELINT processing chain for emitter identification.

Achieving both a high probability of intercept and detection requires properties of the receiver that cannot be met at the same time with acceptable costs. Radar systems operate in a wide frequency range and usually at least the frequencies from 2 GHz to 18 GHz are of interest. Covering the complete frequency range at once with a single wideband receiver results in a high probability of intercept but reduced sensitivity and hence, a lower probability of detection. Another approach is using several channels in the receiver to cover the RF range of interest. The resulting system has a high probability of intercept and detection, but is expensive, large, and heavy. A common solution to the problem is to use a relatively narrowband receiver with high sensitivity and to divide the frequency range into bands of the receiver's instantaneous bandwidth. Now, either an operator can manually search for emissions on the different bands or an automatic search can be performed, which visits the RF bands in a predefined order. A search strategy defines this order, along with the duration of the visit, which is called the (receiver) dwell time. A common strategy is the band sweep, which visits the frequency bands periodically from lowest to highest. However, it bears the risk of a synchronisation between emitter and receiver, such that the emitter is always sending in a direction away from the receiver when the receiver is tuned to the correct frequency and therefore, an intercept never occurs. In the literature, several approaches for optimising search strategies with different goals can be found, e.g. [70–73]. Two of them are provided by the author of this thesis [66,67].

2.2.2 Pulse Descriptor Word Extraction

After a signal is intercepted and detected, pulse descriptor words (PDWs) are extracted from it. Each PDW contains information about a single pulse. Usually, at least the time of arrival (TOA), RF, and pulse width (PW) are given (see Figure 2.2). Since

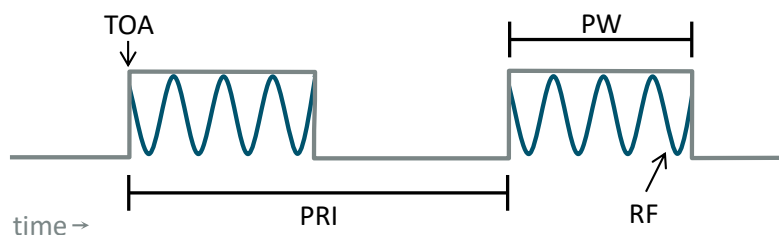


Figure 2.2: Visualisation of the waveform parameters TOA, RF, PW, and PRI.

the PRI is a parameter based on two pulses, it cannot be assigned to the PDWs directly. However, it is an important property and further analysed in the subsequent processing steps. Other common parameters are the angle of arrival (AOA) and amplitude (Amp) of the signal, as well as the modulation on pulse (MOP), which describes a change of phase, Amp, or RF within a single pulse.

2.2.3 Deinterleaving

Usually, there is more than one emitter operating within a frequency band. Hence, the received PDWs must be sorted by emitter in order to analyse them. This process is called deinterleaving and a schematic representation is shown in Figure 2.3. If AOA information is available, PDWs can be clustered according to it in a first step. Since an emitter cannot change the AOA of its signals very fast, the information is valuable. Also RF can be used for clustering. Per cluster, further analysis of the PDWs is performed. By calculating the differences of TOA between different PDWs t_1 and t_2 with $t_1 < t_2$, possible PRIs $\widehat{\text{PRI}}$ are determined,

$$\widehat{\text{PRI}} = \text{TOA}_{t_2} - \text{TOA}_{t_1}. \quad (2.7)$$

The difference in TOA is calculated for every pair of PDWs and a histogram, called the delta-T histogram, is built, which comes with several issues. First of all, many harmonics, i.e. integer multiples of the true PRI, are observed. A reduction of harmonics is achieved by the complex delta-T histogram [74]. Instead of adding a count of one for each occurrence of a possible PRI, a complex phase factor $\varphi(t_2, t_1)$ is determined

$$\varphi(t_2, t_1) = \exp \left(\frac{2\pi i \text{TOA}_{t_2}}{\text{TOA}_{t_2} - \text{TOA}_{t_1}} \right), \quad (2.8)$$

where i is the imaginary unit. This factor is chosen such that the counts of the integer multiples of the true PRIs distribute over the unit circle and almost sum up to zero.

However, another issue that is shared by both histogram methods is the flattening of the peaks when a modulation on the PRI is used (more details on modulations are given in the following Section 2.2.4). A comparison of both methods is shown in Figure 2.4. The input data consists of PDWs from three emitters, two with a stable PRI of 750 μs and 900 μs , respectively, and one that uses a jittered PRI with values that are randomly chosen between 330 μs and 390 μs . In the figure, the true PRIs are marked in red. The two emitters with stable PRI induce clear peaks in both histograms. In the

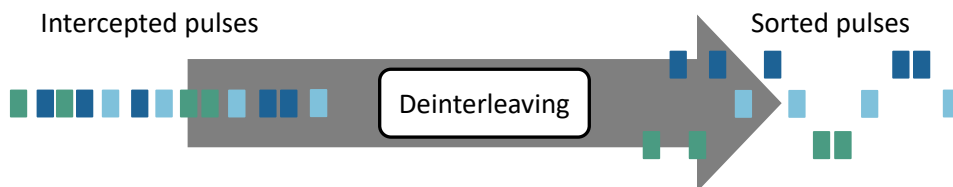


Figure 2.3: Schematic representation of the deinterleaving process.

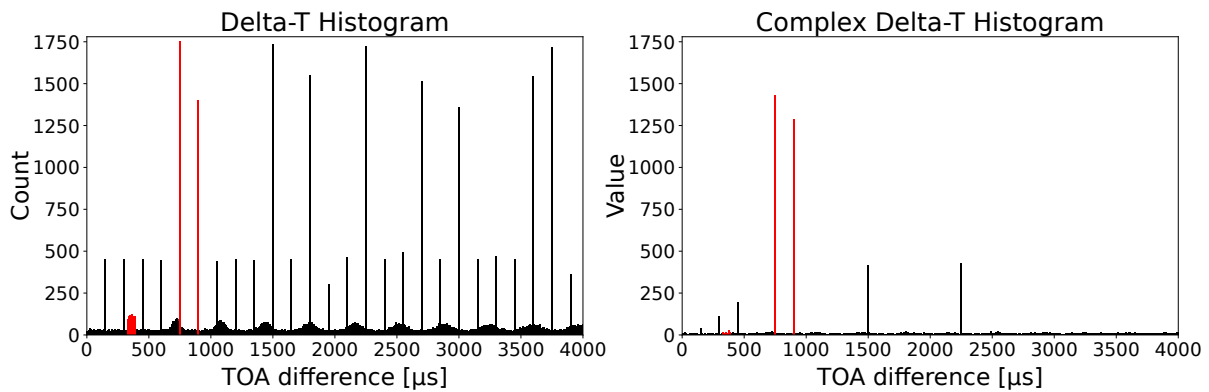


Figure 2.4: Comparison of the delta-T and the complex delta-T histogram for finding possible PRIs in a stream of PDWs. The true PRIs are marked in red.

delta-T histogram, however, there are many harmonics as well. These are suppressed by using the complex delta-T histogram. The jittered PRI is spread over several bins with both methods and cannot be clearly detected.

After potential PRIs have been identified by peaks in the histogram, the sequences of PDWs belonging to the PRIs must be found. This is accomplished by a sequence search [75]. For each of the candidate PRIs it is checked whether a sequence of PDWs with this PRI can be found, starting from the PDW with the lowest TOA. In each step, the candidate PRI is added to the TOA of the current PDW and it is checked if a PDW with the resulting TOA exists. If some minimum number of matching PDWs is found, a sequence is declared and the PDWs are removed from the stream. This procedure is repeated with all candidate PRIs or until less than the required minimum number of PDWs are left. Information about the possible emitters improves the deinterleaving results a lot since a sequence search can be performed with the PRIs given in the database or the bin size of the histogram can be increased when looking for a jittered PRI.

2.2.4 Parameter Extraction

After the PDWs are deinterleaved into separated sequences, the waveform characteristics are further analysed in order to match them to a mode or to infer the capabilities of the radar. Modes are characterised by constant waveform parameters, which are not adaptive and specially designed to fulfil a certain function. The main parameters used to describe a mode are the PRI or PRF, RF, and PW, as well as the modulation of each feature. Examples for common PRI modulations are shown in Figure 2.5. Similar inter-pulse modulation types are applied to RF and PW.

To determine which modulations are present in the deinterleaved PDW sequence, it is checked if the characteristics match the description of the common modulations. For example, to declare a stable PRI, RF, or PW, the standard deviation of the corresponding values from the mean is compared to a threshold. For a stagger PRI, there must be a constant frame rate in the sequence, which corresponds to the sum of the

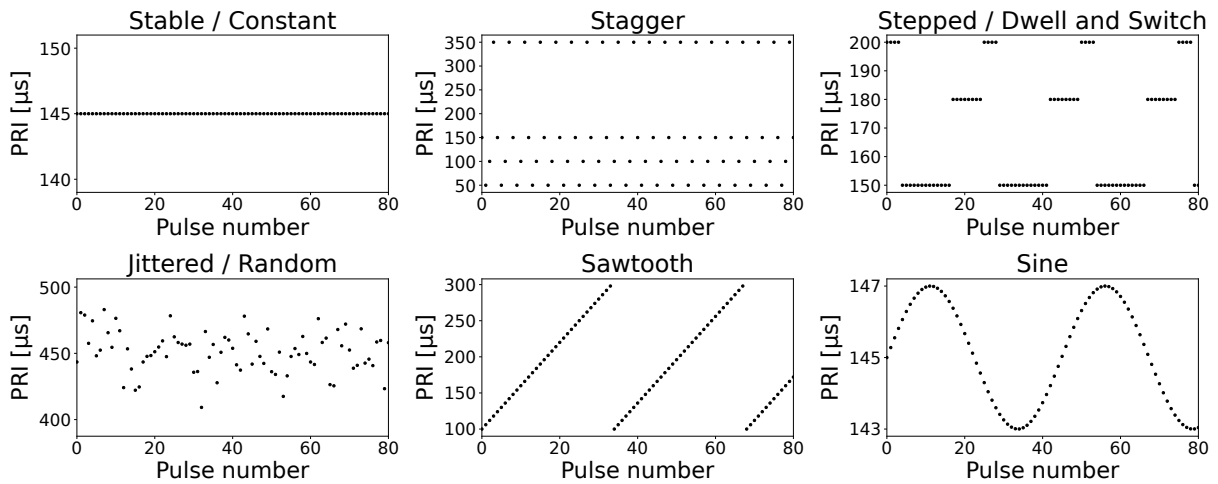


Figure 2.5: Examples of different PRI modulation types.

individual PRI levels in each repetition. Similar characteristics can be exploited for recognising other modulation types, see e.g. [68,76–79]. The type of radar and its capabilities can be inferred based on the estimated parameters and modulations. Table 2.1 lists the typical purposes of different PRI modulation types. If the PRI modulation of the intercepted signal is known, the most likely type of the radar is obtained. Further analysis of the waveform parameters give insight to the capabilities of the radar [1,7]. As described in Section 2.1, the maximum unambiguous range R_u and velocity v_u can be inferred from the measured PRI and RF. Moreover, the range resolution ΔR can be estimated from the measured bandwidth. From the choice of the PRI, it can be inferred whether the radar is designed for accurate range or velocity measurements. If a stepped PRI modulation is observed, it can be concluded that the radar's goal is to measure both range and velocity with a good accuracy. The usage of a high PRI and a stagger modulation, however, indicates that it is an MTI radar.

Table 2.1: Purpose of different PRI modulation types [1].

Modulation	Purpose
Stable	Search or track, MTI, pulse Doppler
Stagger	Elimination of blind speeds (MTI)
Stepped	Resolution of ambiguities in range or velocity (pulse Doppler)
Jittered	Reduction of jamming effects, complication of PRI analysis
Sliding (e.g. Sawtooth)	Assurance of a constant altitude coverage while scanning in elevation, avoidance of eclipsing
Wobulated (e.g. Sine)	Missile guidance, avoidance of eclipsing, ranging

Table 2.2: Simplified example of an emitter mode database.

Emitter	Mode	PRI [μ s]			RF [GHz]			PW [μ s]		
		min	max	mod	min	max	mod	min	max	mod
Em01	M01	740	760	stable	10.0	11.0	stepped	7	8	stable
	M02	400	460	jittered	9.5	9.8	stable	4	5	jittered
Em02	M01	855	905	stable	4.5	5.0	stable	8	9	stable
	M02	200	700	stagger	4.5	5.5	stable	2	7	stagger
	M03	300	900	sine	4.5	5.0	stable	3	9	sine
Em03	M01	880	920	stable	4.7	5.3	stable	8	9	stable
	M02	900	1800	sawtooth	4.0	5.0	stable	9	18	sawtooth

2.2.5 Database Lookup

To finally identify an emitter, the parameters of the deinterleaved PDW sequences are matched against an emitter mode database [1]. A simplified example of such a database is given in Table 2.2. Real databases contain more information like the number of pulses per level of a stepped PRI or the individual levels of a stagger modulation. A range of values is given for each parameter because of measurement uncertainties and the emitters' ability to change their parameters within certain limits. If the parameters of a PDW sequence match a certain mode, the corresponding emitter ID and possibly mode ID are provided at the output of the system. However, there might be ambiguities in the data. For example, when receiving a PDW sequence with a stable PRI of 900 μ s, a stable RF of 5 GHz, and a stable PW of 8 μ s, it is not clear whether this sequence belongs to mode M01 of emitter Em02 or mode M01 of emitter Em03.

2.3 Proposed ELINT Processing Chain

Figure 2.6 shows a schematic visualisation of the processing chain as proposed in this thesis. The first three steps are identical to the traditional processing chain (see Figure 2.1). The parameter extraction step is replaced by a symbol extraction step. As the behaviour of agile multifunction radars cannot be effectively represented in a traditional database, the emission model presented in this thesis describes the radar emissions as a language with a hierarchical structure. The symbol extraction step maps the deinterleaved PDW sequence to the symbols of the radar language. Afterwards, the symbol sequence is processed by an emitter identification approach, which in this thesis is either a Long Short-Term Memory (LSTM) or a Markov chain (MC). One of the outputs of the overall processing chain is then the determined emitter ID or several IDs with the corresponding probabilities. If the input data belongs to a

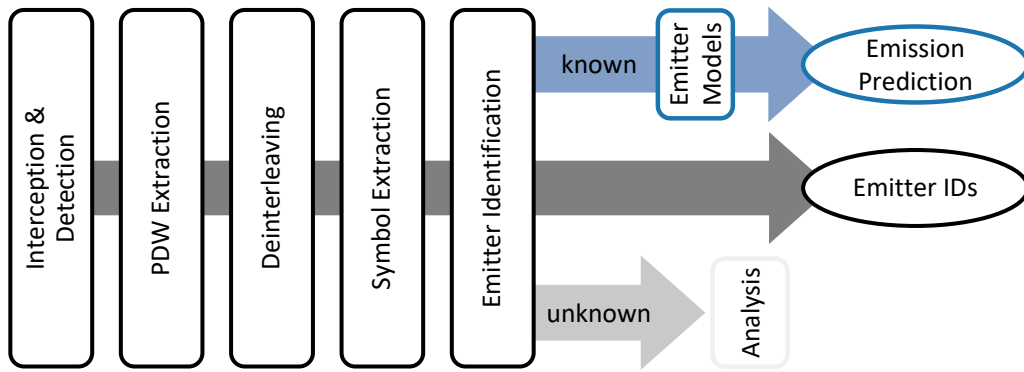


Figure 2.6: The processing chain for emitter identification and emission prediction proposed in this thesis.

known radar, a single emitter model or an ensemble of those is activated to make a prediction on the next emission of the radar. Otherwise, the signal is recorded for further analysis. The steps covered in this thesis are the emission prediction using the emitter models, the identification of emitters based on their symbols, and the recognition of unknown emitters. In addition, strategies for combining the emitter models into an overall system are presented.

2.4 The Five Problems of Electronic Warfare

The four problems of EW are defined in [80] as:

1. **Classification/Identification:** Given a sequence of observations and several radar models: How to choose the radar model which best describes the observations?
2. **Decoding:** Given a sequence of observations and a radar model: How to choose a sequence of internal radar states that best explains the observations?
3. **Prediction:** Given a sequence of observations and a radar model: How to compute the next expected observation and its probability?
4. **Learning/Training:** Given only a sequence of observations: How to adapt the parameters of the radar model (or train the radar model) to fit the observations?

This thesis introduces a fifth problem:

5. **Unknown Emitter Recognition:** Given a sequence of observations and several radar models: How to recognise if none of the radar models is suitable to describe the observations?

The present work provides new solutions to the first, third, fourth, and fifth problem.

Chapter 3

Modelling & Learning of Sequential Data

This chapter presents different general possibilities for modelling systems that produce sequential data like radar signals. If training procedures are available, which adapt the model parameters based on data, this chapter also highlights the corresponding methods. One of the most commonly used system category is the discrete event system (DES). DESs are defined in [10] as systems with a discrete state space for which the evolution of states only depends on the occurrence of asynchronous or clocked events. In addition, they are dynamic (the output depends on previous inputs), time-invariant (the behaviour does not change over time) and nonlinear. Modelling these kinds of systems is part of system theory, which is an area of engineering. The following section introduces formal languages and grammars as they are strongly linked to modelling DESs with e.g. finite state machines (FSMs) and hidden Markov models (HMMs).

3.1 Formal Languages & Grammars

A formal language is a collection of sentences of finite length, which are constructed based on symbols from a finite alphabet [81]. A grammar can be regarded as a procedure that lists the sentences of a language. Formally, a grammar G is defined as [82]

$$G = (\mathcal{N}, \Sigma, \mathbb{P}, S), \quad (3.1)$$

with

- \mathcal{N} finite set of nonterminal symbols, i.e. the symbols that are replaced according to the production rules and are not part of the final words,
- Σ finite set of terminal symbols, i.e. the grammar's alphabet,
- \mathbb{P} set of production rules,
- S $S \in \mathcal{N}$, start symbol.

From the grammar's production rules, one can infer the words that can be generated. A production rule may be of the form $\Lambda \rightarrow \gamma$ with $\Lambda \in \mathcal{N}$ and $\gamma \in (\mathcal{N} \cup \Sigma)^*$. This rule means that a nonterminal symbol Λ can be replaced by a sequence of other symbols. On the left hand side of a production rule, there must be at least one nonterminal symbol in addition to some optional terminal symbols. The star operator $*$ is called the Kleene closure or Kleene star. It describes the set of all words which can be

formed by arbitrary combinations of the language's symbols and words. This set also contains the empty word χ . In this case, the expression $\gamma \in (\mathcal{N} \cup \Sigma)^*$ means that γ is an arbitrary sequence of nonterminal and terminal symbols.

The production rules are best explained using an example. Let a grammar $G = (\mathcal{N}, \Sigma, \mathbb{P}, S_0)$ be defined as

$$\mathcal{N} = \{S_0, S_1\}, \quad (3.2)$$

$$\Sigma = \{a, b, c\}, \quad (3.3)$$

$$\mathbb{P} = \{S_0 \rightarrow aS_1 | b, \quad (3.4)$$

$$S_1 \rightarrow cS_0 | a\}. \quad (3.5)$$

The first production rule (3.4) defines that the start symbol S_0 can either be replaced by the sequence consisting of the terminal symbol a and the nonterminal symbol S_1 or the terminal symbol b . Analogously, the second rule (3.5) means that the nonterminal symbol S_1 can either be replaced by the sequence consisting of the terminal symbol c and the nonterminal symbol S_0 or the terminal symbol a . By applying these rules, one can derive, amongst others, these sequences of the language

$$S_0 \Rightarrow b,$$

$$S_0 \Rightarrow aS_1 \Rightarrow aa,$$

$$S_0 \Rightarrow aS_1 \Rightarrow acS_0 \Rightarrow acb,$$

$$S_0 \Rightarrow aS_1 \Rightarrow acS_0 \Rightarrow acaS_1 \Rightarrow acaa,$$

$$S_0 \Rightarrow aS_1 \Rightarrow acS_0 \Rightarrow acaS_1 \Rightarrow acacS_0 \Rightarrow acacb.$$

This grammar defines a language with an infinite number of sequences or words of finite length. Words of the language either start with a or b while the word starting with b only consists of this one symbol. In addition, all words with two or more symbols end with aa or cb .

In 1959, Noam Chomsky divided formal grammars into different categories based on the properties of their production rules [81]. This classification is known as the Chomsky hierarchy and is shown in Figure 3.1. The regular grammars are the least expressive ones, in terms of the restrictions posed on the production rules described

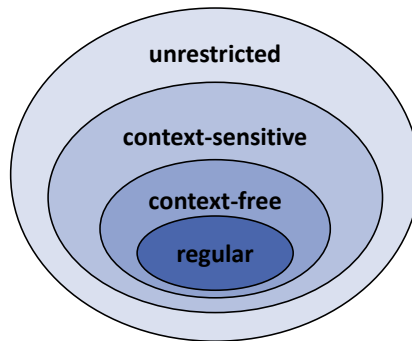


Figure 3.1: Visualisation of the Chomsky hierarchy for classes of formal grammars.

below, whereas the unrestricted grammars are equivalent to Turing machines. The less restricted grammars contain the more restricted ones as proper subsets. A Turing machine is a mathematical model of computation which is especially important for computability theory in the area of theoretical computer science. It is a purely theoretical model used for analysing what a computer can actually compute and is therefore not further considered here. In the following sections it is only used for comparison of expressiveness. Details on Turing machines can be found in [83].

An overview of the classes of formal grammars and examples of models, that are able to represent them, is shown in Table 3.1. The Chomsky hierarchy defines the classes as follows [15,81]:

- **Regular grammar.** Only production rules of the form $\Lambda \rightarrow a\Pi$ or $\Lambda \rightarrow a$ are allowed, with $\Lambda, \Pi \in \mathcal{N}$ and $a \in \Sigma$. This means that the left hand side of the rule can only consist of one nonterminal symbol and no others symbols. The right hand side can consist of only one terminal symbol or one nonterminal symbol followed by a terminal symbol. This is the only type of grammar which can be represented by FSMs, MCs, and HMMs (see Sections 3.2, 3.4.1, and 3.4.2).
- **Context-free grammar.** Only production rules of the form $\Lambda \rightarrow \beta$ with $\Lambda \in \mathcal{N}$ and $\beta \in (\mathcal{N} \cup \Sigma)^*$, $\beta \neq \chi$ are allowed. On the left hand side, there must be one nonterminal symbol and on the right hand side, any sequence of symbols is allowed except for the empty word. This type of production rules includes the regular grammars. This can easily be seen as β is only restricted by exclusion of the empty word. Hence, it can be replaced by $a\Pi$ or a , which converts the production rules to those of regular grammars. Context-free grammars can be represented by pushdown automata, which are purely theoretic and therefore not considered here. An introduction can be found in [83].
- **Context-sensitive grammar.** Production rules of the form $\alpha_1\Lambda\alpha_2 \rightarrow \alpha_1\beta\alpha_2$ are allowed, with $\Lambda \in \mathcal{N}$, $\alpha_1, \alpha_2, \beta \in (\mathcal{N} \cup \Sigma)^*$, and $\beta \neq \chi$. This means that the replacement of Λ can be dependent on the context given by α_1 and α_2 . By excluding the empty word for β , the deletion of symbols is prohibited. For the context α_1, α_2 the empty word is not excluded, such that this type of grammar includes the regular and context-free grammars. Context-sensitive grammars can be represented by linear-bounded Turing machines.
- **Unrestricted grammar.** All production rules of the form $\alpha_1\Lambda\alpha_2 \rightarrow \gamma$ are allowed. It holds that $\alpha_1, \alpha_2, \gamma \in (\mathcal{N} \cup \Sigma)^*$ without any restrictions. These grammars can be represented by Turing machines and recurrent neural networks (RNNs) (see Section 3.5.5).

In addition to this categorisation, the self-embedding property is important. A grammar is self-embedding if a derivation of the form $\Lambda \Rightarrow \dots \Rightarrow \alpha_1\Lambda\alpha_2$ can be achieved by the application of potentially several production rules, with $\Lambda \in \mathcal{N}$, $\alpha_1, \alpha_2 \in (\mathcal{N} \cup \Sigma)^*$, and $\alpha_1, \alpha_2 \neq \chi$. The work [84] shows that a non self-embedding context-free grammar defines a regular language.

Table 3.1: Chomsky hierarchy of formal grammars with suitable models.

Grammar	Model
Regular	Finite state machine, Markov chain, hidden Markov model
Context-free	Pushdown automaton
Context-sensitive	Linear-bounded Turing machine
Unrestricted	Turing machine, recurrent neural network

As an extension to (deterministic) grammars, a stochastic variant can be defined. Therein, every production rule is assigned a probability, while the probabilities of the rules with identical left hand side must sum to 1. The probability of a derivation, i.e. one specific way to reach the final output from the start symbol, is equal to the product of the probabilities of the applied production rules. For example, the deterministic grammar in (3.2) to (3.5) can be converted to a stochastic grammar defined by

$$\mathcal{N} = \{S_0, S_1\}, \quad (3.6)$$

$$\Sigma = \{a, b, c\}, \quad (3.7)$$

$$\mathbb{P} = \{S_0 \xrightarrow{0.8} aS_1, \quad (3.8)$$

$$S_0 \xrightarrow{0.2} b, \quad (3.9)$$

$$S_1 \xrightarrow{0.7} cS_0, \quad (3.10)$$

$$S_1 \xrightarrow{0.3} a\}. \quad (3.11)$$

3.2 Finite State Machines

A possible model for DESs is the finite state machine (FSM). A deterministic FSM H is defined by [10]

$$H = (\mathcal{X}, \mathcal{F}, f_{\mathcal{X}}, \Gamma, \check{x}_0, \mathcal{X}_m), \quad (3.12)$$

with

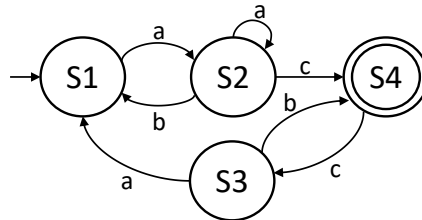


Figure 3.2: A state diagram of an FSM. The set of states is $\mathcal{X} = \{S1, S2, S3, S4\}$, the set of events is $\mathcal{F} = \{a, b, c\}$, the initial state is $\check{x}_0 = S1$, the set of accepting states is $\mathcal{X}_m = \{S4\}$.

\mathcal{X}	set of states (for FSMs, this is a finite set),
\mathcal{F}	finite set of events in H ,
$f_{\mathcal{X}}$	$f_{\mathcal{X}} : \mathcal{X} \times \mathcal{F} \rightarrow \mathcal{X}$, state transition function, $f_{\mathcal{X}}(\check{x}, \check{e}) = \check{x}'$ means that event \check{e} in state \check{x} causes a transition to \check{x}' ,
Γ	$\Gamma : \mathcal{X} \rightarrow \mathcal{P}(\mathcal{F})$, function of active or feasible events, $\Gamma(\check{x})$ is the set of events \check{e} for which $f_{\mathcal{X}}(\check{x}, \check{e})$ is defined,
$\mathcal{P}(\mathcal{F})$	power set of \mathcal{F} ,
\check{x}_0	initial state,
\mathcal{X}_m	set of marked or accepting states, $\mathcal{X}_m \subseteq \mathcal{X}$.

Figure 3.2 shows the state diagram of an FSM. The set of states \mathcal{X} consists of $S1$, $S2$, $S3$, and $S4$. $S1$ is the initial state and marked by an arrow. The only accepting state in \mathcal{X}_m is $S4$, which is marked by a circle with two lines. $\mathcal{F} = \{a, b, c\}$ defines the set of events. As e.g. the event c is not feasible in state $S3$ and therefore $\Gamma(S3) = \{a, b\}$, the state transition function $f_{\mathcal{X}}$ is only partially defined.

DESS like FSMs are commonly represented by formal languages defined on the set of events \mathcal{F} . This language consists of a sequence of symbols, which correspond to words, of finite length which are generated from the set of events [10]. Based on the set $\mathcal{F} = \{a, b, c\}$ from Figure 3.2, different languages can be defined, e.g.

$$L_1 = \{\chi, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}. \quad (3.13)$$

This language contains all words up to a length of two, including the empty word χ . However, an infinite number of words of finite length can be defined on an alphabet, which is in this case the set of events, e.g.

$$L_2 = \{\chi, a, b, c, d, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, \dots\}. \quad (3.14)$$

Using the representation as a formal language, the automaton in Figure 3.2 only accepts those words starting with an a as only those can leave state $S1$. In addition, the words need to end with b or c to reach the only accepting state $S4$. For example, this FSM accepts the word $aacbb$ but not acc as this word ends in $S3$.

Up to now, every event causes a clearly defined state transition. When modelling a system of which the dynamics are not entirely known, stochastic FSMs can be used. In general, these are defined in the same way as deterministic automata, except for the state transition function $f_{\mathcal{X}}$, which returns a set of possible next states instead of a single one. Hence, it is defined as $f_{\mathcal{X}} : \mathcal{X} \times \mathcal{F} \rightarrow \mathcal{P}(\mathcal{F})$. In addition, the initial state can be composed of a set of possible initial states.

FSMs have restricted expressiveness. For example, it is not possible to construct an FSM which only accepts the language of palindromes, i.e. words which are the same forwards and backwards. The reason is that an FSM does not have a memory. At a certain point in time, it only knows its current state but not the sequence of states which led to it. Therefore, an FSM which accepts the palindrome $aabaa$ would

also accept the sequence *aaba*. Of course, one could design an FSM in which the number of states corresponds to the number of symbols in the palindrome *aabaa*. However, this FSM could not accept palindromes of arbitrary length, e.g. it would not accept the palindrome *aaabbaaa*. In general, FSMs can only accept the class of regular grammars. The grammar of the palindrome language, however, is context-free and self-embedding.

3.3 Petri Nets

DEs can as well be modelled by Petri nets, which were developed in the 1960s by Carl Adam Petri. In general, a Petri net is represented by a graph that consists of two different types of nodes, the places and the transitions. Only nodes of different types can be connected to each other. Formally, a Petri net graph G_P is defined by [10]

$$G_P = (P, T, A, f_w), \quad (3.15)$$

with

- P finite set of places,
- T finite set of transitions,
- A $A \subseteq (P \times T) \cup (T \times P)$, set of arcs connecting places and transitions in G_P ,
- f_w $f_w : A \rightarrow \{1, 2, 3, \dots\}$, weight function for the weights of the arcs.

Places can be interpreted as conditions that need to be fulfilled for a transition, which resembles an event. To mark the fulfilment of conditions, tokens are used. A marked Petri net graph G_P^m is therefore defined as [10]

$$G_P^m = (P, T, A, f_w, \mathbf{x}), \quad (3.16)$$

with definitions as above and

- \mathbf{x} marking of the places $P = \{p_1, p_2, \dots, p_{|P|}\}$,
- $\mathbf{x} = [x(p_1), x(p_2), \dots, x(p_{|P|})] \in \mathbb{N}^{|P|}$.

This means that \mathbf{x} is a vector that stores the number of tokens for every place in P . Figure 3.3a shows an example of a Petri net graph with three tokens. Places are visualised by circles and transitions by vertical lines. The set of places is $P = \{p_1, p_2, p_3, p_4\}$ and the set of transitions is $T = \{t_1, t_2, t_3\}$. The place p_1 has two tokens, represented by triangles, and p_4 has one. As the weights of the arcs are all 1, they are not shown.

The distribution of the tokens, i.e. the vector \mathbf{x} , constitutes the current state of the Petri net. State transitions are represented by moving the tokens through the net, which happens when a transition “fires”. A transition can only fire if all places that are input to the transition have at least as many tokens as connecting arcs multiplied by the arc weights. For example, the transition t_2 in Figure 3.3a needs one token from

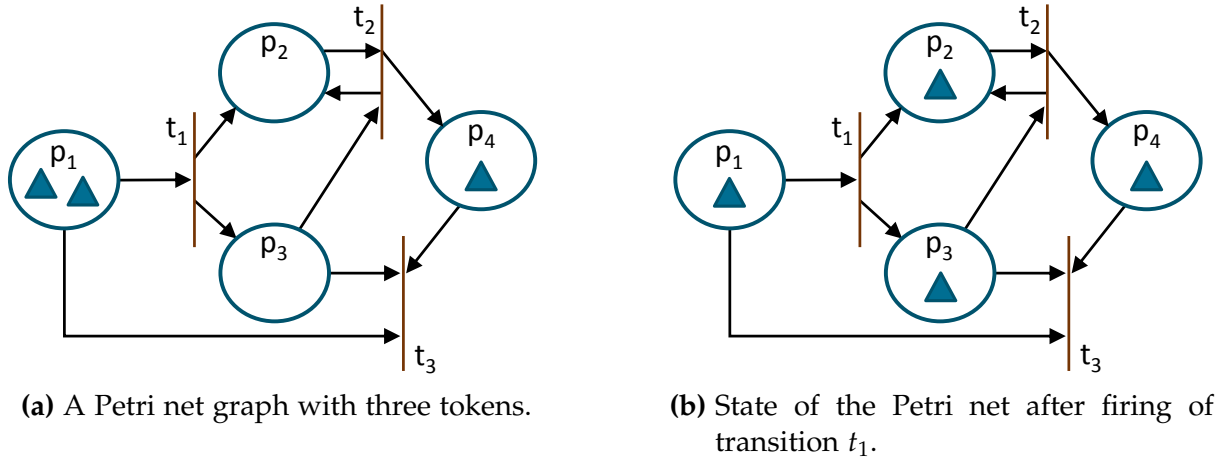


Figure 3.3: Example of a Petri net graph in different states. The places are $P = \{p_1, p_2, p_3, p_4\}$, the transitions are $T = \{t_1, t_2, t_3\}$. All arc weights are one.

the place p_2 and another one from p_3 to fire. The state in Figure 3.3a allows firing of transitions t_1 or t_3 . The order of firing is not defined for Petri nets. Considering the scenario of t_1 firing, the next state is shown in Figure 3.3b. Since transition t_1 is connected to two places p_2 and p_3 , it provides one token to each of these places, i.e. the number of tokens increases. Conversely, the number of tokens can also decrease. As a consequence, Petri nets have an infinite state space.

Formally, the transition function $f_T : \mathbb{N}^{|P|} \times T \rightarrow \mathbb{N}^{|P|}$ of a Petri net $(P, T, A, f_w, \mathbf{x})$ is defined for a transition $t_j \in T$ if and only if [10]

$$x(p_i) \geq f_w(p_i, t_j) \text{ for all } p_i \in I(t_j). \quad (3.17)$$

Here $I(t_j)$ is the set of places that are input to transition t_j . If $f_T(\mathbf{x}, t_j)$ is defined, the new state $\mathbf{x}' = f_T(\mathbf{x}, t_j)$ is given by

$$x'(p_i) = x(p_i) - f_w(p_i, t_j) + f_w(t_j, p_i), \quad i = 1, \dots, |P|. \quad (3.18)$$

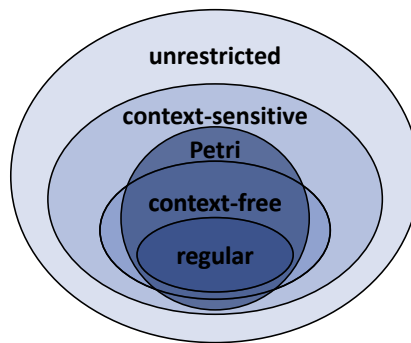


Figure 3.4: The relationship between Petri net languages and the Chomsky hierarchy.

Due to their infinite state space, Petri nets are more expressive than FSMs, i.e. for every FSM an equivalent Petri net can be designed but not vice versa [10]. To determine an accepted language analogous to FSMs, states of the Petri nets are specified as accepting. However, Petri nets are not able to accept the language of palindromes either [85]. The Petri net language cannot be exactly mapped to the Chomsky hierarchy. The author of [86] shows that all Petri net languages are subsets of the context-sensitive languages but there exists context-free languages which are not Petri net languages, e.g. the palindrome language. Figure 3.4 visualises this relationship.

3.4 Stochastic Processes

If not all circumstances that lead to a certain system state are known, the process seems to be random to an observer. Such sequences of states can be described by the mathematical concept of stochastic or random processes. A family of random variables $\{X_t\}_{t \in \mathcal{T}}$ indexed by $\mathcal{T} \subseteq [0, \infty)$ is called stochastic or random process. Usually, the index is interpreted as time. If $\mathcal{T} = \mathbb{N}$ (or $\mathcal{T} = \mathbb{N}_0$), then $\{X_t\}_{t \in \mathcal{T}}$ is a discrete-time and if $\mathcal{T} = [0, \infty)$ a continuous-time process [87].

3.4.1 Markov Chains

A special variant of a stochastic process is the Markov process, which is in the case of a discrete state space also called Markov chain (MC). A Markov process describes the evolution of systems that can take on different states. It has the “memoryless” property, i.e. the probability of the next state only depends on the current state but not on the previous states. For modelling of DESs, the definition of a discrete-time MC is of importance [87]. A stochastic process $\{X_t\}_{t \in \mathbb{N}_0}$, which can take on values in an at most countable state space \mathcal{X} is called MC or fulfils the Markov property if

$$\begin{aligned} &P(X_{t+1} = \check{x}_{t+1} | X_t = \check{x}_t, X_{t-1} = \check{x}_{t-1}, \dots, X_1 = \check{x}_1, X_0 = \check{x}_0) \\ &= P(X_{t+1} = \check{x}_{t+1} | X_t = \check{x}_t) \end{aligned} \quad (3.19)$$

holds for all $t \in \mathbb{N}_0$, all $\check{x}_0, \check{x}_1, \dots, \check{x}_t, \check{x}_{t+1} \in \mathcal{X}$, if the conditional probabilities are well defined, i.e. if $P(X_t = \check{x}_t, \dots, X_1 = \check{x}_1, X_0 = \check{x}_0) > 0$. Otherwise, a division by 0 would occur according to Bayes’ rule. In the following, the random variable is omitted and the above formula abbreviated by

$$P(\check{x}_{t+1} | \check{x}_t, \check{x}_{t-1}, \dots, \check{x}_1, \check{x}_0) = P(\check{x}_{t+1} | \check{x}_t). \quad (3.20)$$

The transition probabilities from state \check{x}_t to state \check{x}_{t+1} are defined in a transition matrix $A \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{X}|}$. To determine the state transition probabilities for every state $\check{x}_i, \check{x}_j \in \mathcal{X}$ from data, the number of transitions between \check{x}_i and \check{x}_j are counted and normalised by the total number of transitions from state \check{x}_i to every other state in \mathcal{X} ,

$$P(\check{x}_j | \check{x}_i) = a_{ij} = \frac{\text{count}(\check{x}_i, \check{x}_j)}{\sum_{\check{x} \in \mathcal{X}} \text{count}(\check{x}_i, \check{x})}, \quad (3.21)$$

where $\text{count}(\check{x}_i, \check{x}_j)$ is the number of state transitions from \check{x}_i to \check{x}_j .

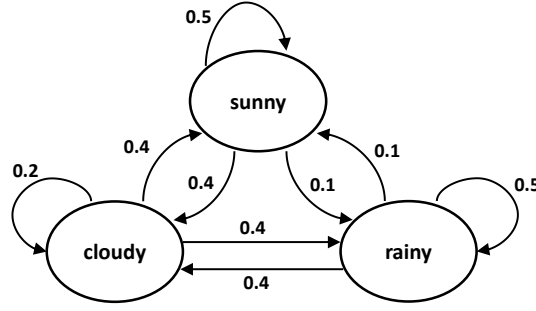


Figure 3.5: Example of an MC that describes the transition probabilities of the weather between “sunny”, “cloudy”, and “rainy”.

The Markov property states that only the immediate predecessor state influences the current state. Moreover, the duration of the state is irrelevant as well. An example is given in [10]: Assume the process has made a transition to state \check{x}_i at time t_1 . At time $t_2 > t_1$ the process is still in state \check{x}_i , hence no transition took place. Based on the Markov property, trying to predict the next state based on the information “the state at time t_2 is \check{x}_i ” must yield the same result as a prediction based on the information “the state at time t_2 is \check{x}_i and it was \check{x}_i in the interval $[t_1, t_2]$ ”. It follows that the retention times are described by an exponential distribution. The proof can be found in [10]. Semi-Markov processes do not have this restriction, i.e. the retention time in each state can be described by an arbitrary distribution. However, if an event occurs, these processes behave like conventional Markov processes such that the transition probability to another state only depends on the current state.

MCs can be represented by state diagrams, as shown in Figure 3.5. The three states of this chain are “sunny”, “cloudy”, and “rainy”. The values attached to the arrows between the states show the transition probabilities.

To equip the MC with more “memory”, higher order variants are defined, for which the probability of the next state does not only depend on the current state but on the last n states,

$$P(\check{x}_{t+1} | \check{x}_t, \check{x}_{t-1}, \dots, \check{x}_1, \check{x}_0) = P(\check{x}_{t+1} | \check{x}_t, \check{x}_{t-1}, \dots, \check{x}_{t-n+1}). \quad (3.22)$$

However, the size of the transition matrix A grows from dimension $|\mathcal{X}| \times |\mathcal{X}|$ for a first order MC to $|\mathcal{X}|^n \times |\mathcal{X}|$ with order n .

3.4.2 Hidden Markov Models

A hidden Markov model (HMM) introduces another layer to an MC. In an HMM, the states $\{X_t\}$ of the process are not directly observable but only its “emissions” $\{Y_t\}$. The sequence of hidden states is described by an MC. In every state a symbol from the alphabet \mathcal{Y} is “emitted” with a certain probability. These symbols are observable. Formally, a first order HMM is defined by [43]

$$\lambda = (\mathcal{X}, \mathcal{Y}, A, B, \pi), \quad (3.23)$$

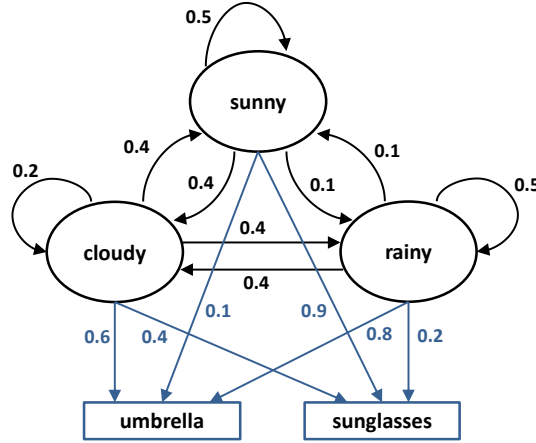


Figure 3.6: Example of an HMM based on the MC from Figure 3.5. It models the observations “sunglasses” and “umbrella” depending on the hidden states of the weather.

with

\mathcal{X}	set of states, i.e. the possible values of $\{X_t\}$,
\mathcal{Y}	alphabet of the observations, i.e. the possible values of the emissions $\{Y_t\}$,
$A \in \mathbb{R}^{ \mathcal{X} \times \mathcal{X} }$	matrix of the transition probabilities between states of the underlying Markov chain,
$B \in \mathbb{R}^{ \mathcal{X} \times \mathcal{Y} }$	matrix of emission probabilities,
$\pi \in \mathbb{R}^{ \mathcal{X} }$	probability distribution of the initial state.

As the entries of both matrices A and B represent probabilities, they must be non-negative and the rows must sum up to 1.

As an example the MC from Figure 3.5 is extended. Assume a person spends a few days in a room without a window and cannot observe the weather directly. Based on the observations that a visitor carries sunglasses or an umbrella, the person can calculate the probabilities of the hidden states of the weather. Figure 3.6 shows the HMM with the observations and the corresponding emission probabilities in blue. Here, $\mathcal{X} = \{\check{x}_s, \check{x}_c, \check{x}_r\}$ with $\check{x}_s = \text{sunny}$, $\check{x}_c = \text{cloudy}$, and $\check{x}_r = \text{rainy}$. The set of observations corresponds to $\mathcal{Y} = \{\check{y}_s, \check{y}_u\}$ with $\check{y}_s = \text{sunglasses}$ and $\check{y}_u = \text{umbrella}$.

Three problems have been defined [43], which got to be known as the classical problems of HMMs:

1. **Evaluation:** Given a sequence of observations O and an HMM λ . How can the probability that the model generated the sequence $P(O|\lambda)$ be computed? This problem is solved by the forward algorithm.

2. **Decoding:** Given a sequence of observations O and an HMM λ . How can a sequence of internal states $\{X_t\}$ that describes the observations best, i.e.

$$\operatorname{argmax}_{\{X_t\}} P(O|\{X_t\}, \lambda), \quad (3.24)$$

be selected? A solution is found by the Viterbi algorithm.

3. **Learning/Training:** Given a set of sequences of observations and an HMM. How can the parameters of the HMM be adapted such that the probability that it generates the sequences is maximised? A solution can be found using the Baum-Welch algorithm.

For the solution of the first problem, the evaluation, an example is given. Assume that for the HMM in Figure 3.6 the observations $O = \check{y}_u \check{y}_u \check{y}_s$ are made for three consecutive days. To determine the probability of this sequence $P(O|\lambda)$, the probabilities of all possible sequences of internal states that can lead to O need to be summed. The probability of the state sequence is calculated by the product of the transition probabilities between the states given by A and the emission probabilities in each state given by B . Let the distribution for the initial state be $1/3$ for each of the states, i.e. $\pi = (\pi_{\check{x}_s}, \pi_{\check{x}_c}, \pi_{\check{x}_r}) = (1/3, 1/3, 1/3)$. For easier readability, the dependency on the model λ is implicitly assumed in the following such that e.g. $P(\check{x}_s|\check{x}_c)$ is written instead of $P(\check{x}_s|\check{x}_c, \lambda)$. The probability of the sequence O for the sequence of internal states $\check{x}_s \check{x}_c \check{x}_s$ is then calculated by

$$\begin{aligned} P(O|\check{x}_s, \check{x}_c, \check{x}_s) &= \pi_{\check{x}_s} \cdot P(\check{y}_u|\check{x}_s) \cdot P(\check{x}_c|\check{x}_s) \cdot P(\check{y}_u|\check{x}_c) \cdot P(\check{x}_s|\check{x}_c) \cdot P(\check{y}_s|\check{x}_s) \\ &= 1/3 \cdot 0.1 \cdot 0.4 \cdot 0.6 \cdot 0.4 \cdot 0.9 = 0.00288. \end{aligned} \quad (3.25)$$

To determine the probability for the given sequence, the probabilities of all state sequences need to be summed,

$$\begin{aligned} P(O|\lambda) &= P(O|\check{x}_s, \check{x}_s, \check{x}_s) + P(O|\check{x}_s, \check{x}_s, \check{x}_c) + P(O|\check{x}_s, \check{x}_s, \check{x}_r) + P(O|\check{x}_s, \check{x}_c, \check{x}_s) + \\ &P(O|\check{x}_s, \check{x}_c, \check{x}_c) + P(O|\check{x}_s, \check{x}_c, \check{x}_r) + P(O|\check{x}_c, \check{x}_s, \check{x}_s) + \dots \end{aligned} \quad (3.26)$$

Since this naive approach is very inefficient, a more sophisticated method called the forward algorithm was developed [43]. It uses techniques of dynamic programming to store intermediate results and hence prevents calculating the same probabilities multiple times. The steps are as follows:

1. Calculate the initial probability for every state $\check{x} \in \mathcal{X}$ based on the first observation, $P(\check{x}|\check{y}_1)$. For example for the state “cloudy” and the observation “umbrella”,

$$P(\check{x}_c|\check{y}_u) = \pi_{\check{x}_c} \cdot P(\check{y}_u|\check{x}_c) = 1/3 \cdot 0.6 = 0.2. \quad (3.27)$$

2. Calculate the probability for every possible state transition, depending on the second observation. For example for the state “cloudy” with the probability 0.2 obtained in the first step and the second observation “umbrella”, the probability for a state transition to the state “sunny” is given by,

$$P(\check{x}_s|\check{x}_c) = P(\check{x}_c|\check{y}_u) \cdot P(\check{x}_s|\check{x}_c) \cdot P(\check{y}_u|\check{x}_s) = 0.2 \cdot 0.4 \cdot 0.1 = 0.008. \quad (3.28)$$

3. For the third observation, the probability of the state is computed by the sum P_{sum} of the possible state transitions obtained in the second step. This sum corresponds to the probability of a certain state given the observations seen so far. Apart from that, the procedure is the same as in the step before. The sum P_{sum} is called *forward* probability.
4. Repeat the calculation of the probabilities of the state transitions and observations until the end of the observation sequence.
5. The final result is the sum of the sum [sic] of the state probabilities in the last step.

A solution to the calculation of the most probable sequence of internal states given the observations, i.e. the decoding problem, is provided by the Viterbi algorithm, which uses dynamic programming [43]. The algorithm works as follows:

1. Calculate the initial probability for every state based on the first observation. For example for the state “cloudy” and the observation “umbrella”,

$$P(\check{x}_c|\check{y}_u) = \pi_{\check{x}_c} \cdot P(\check{y}_u|\check{x}_c) = 1/3 \cdot 0.6 = 0.2. \quad (3.29)$$

2. Calculate the probability for every possible state transition, depending on the second observation. For example for the state “cloudy” with the probability 0.2 obtained in the first step and the second observation “umbrella”, the probability for a state transition to the state “sunny” is given by

$$P(\check{x}_s|\check{x}_c) = P(\check{x}_c|\check{y}_u) \cdot P(\check{x}_s|\check{x}_c) \cdot P(\check{y}_u|\check{x}_s) = 0.2 \cdot 0.4 \cdot 0.1 = 0.008. \quad (3.30)$$

3. For the third observation the probability of the state is assumed to be the maximum P_{max} of the possible state transitions obtained in the second step. For the state “sunny” this is the state transition from “cloudy”. Apart from that, the procedure is the same as in the step before.
4. Repeat the calculation of the probabilities of the state transitions and observations until the end of the observation sequence.
5. Determine the final state with the largest probability and trace back the path that led to this state. This path is the most probable sequence of hidden states. In this example, it is $\check{x}_r\check{x}_c\check{x}_s$.

As can easily be seen, this algorithm is very similar to the forward algorithm. The first two steps are even identical. However, in the third step the maximum of probabilities and not the sum is used. Of course, the procedures differ in the last step as the goals of the calculations are different.

The third problem, i.e. training, is solved by the Baum-Welch algorithm [43]. It is an iterative procedure and a variant of the expectation-maximisation (EM) algorithm, which is used to estimate the matrix of state transition probabilities A and the matrix

of emission probabilities \mathbf{B} . Since the states of an HMM cannot be directly observed, the state transition probabilities cannot be determined by counting according to (3.21) but need to be estimated. For a given observation sequence $O = \check{y}_1\check{y}_2\ldots\check{y}_{|O|}$ the probability that the HMM is in state \check{x}_i at time t and performs a transition to state \check{x}_j at the next time step $t + 1$ is calculated for every time step t . This probability is normalised by the summed probabilities for a transition to all states $\check{x} \in \mathcal{X}$,

$$\hat{P}(\check{x}_j|\check{x}_i) = \hat{a}_{ij} = \frac{\sum_{t=1}^{|O|-1} P(X_t = \check{x}_i, X_{t+1} = \check{x}_j|O, \lambda)}{\sum_{t=1}^{|O|-1} \sum_{\check{x} \in \mathcal{X}} P(X_t = \check{x}_i, X_{t+1} = \check{x}|O, \lambda)}. \quad (3.31)$$

The term $P(X_t = \check{x}_i, X_{t+1} = \check{x}_j|O, \lambda)$ is commonly abbreviated by $\xi_t(\check{x}_i, \check{x}_j)$ in the literature. It is given by

$$\xi_t(\check{x}_i, \check{x}_j) = \frac{P(\check{x}_i, \check{y}_1, \dots, \check{y}_t|\lambda) \cdot P(\check{x}_j|\check{x}_i) \cdot P(\check{y}_{t+1}|\check{x}_j, \lambda) \cdot P(\check{y}_{t+1}, \dots, \check{y}_{|O|}|\check{x}_j, \lambda)}{P(O|\lambda)}. \quad (3.32)$$

Here, the term $P(\check{y}_{t+1}, \check{y}_{t+2}, \dots, \check{y}_{|O|}|\check{x}_j, \lambda)$ is called *backward* probability in analogy to the forward probability defined by $P_{sum_i}(\check{x}_i) = P(\check{x}_i, \check{y}_1, \check{y}_2, \dots, \check{y}_t|\lambda)$ as described above. It is calculated analogously to the forward probability. Determining the denominator $P(O|\lambda)$ is the problem of evaluation, which is solved by the forward algorithm. The estimation of the transition probability contains the transition probability as well, which is the value from the previous step. If no a-priori information is available, the parameters are initialised randomly. To estimate the emission probability of every state, the probability that the HMM is in state \check{x}_i at time step t is needed,

$$\gamma_t(\check{x}_i) = P(X_t = \check{x}_i|O, \lambda) = \frac{P(\check{x}_i, O|\lambda)}{P(O|\lambda)} = \frac{P(\check{x}_i, \check{y}_1, \dots, \check{y}_t|\lambda) \cdot P(\check{y}_{t+1}, \dots, \check{y}_{|O|}|\check{x}_i, \lambda)}{P(O|\lambda)}. \quad (3.33)$$

Now, the above probabilities for every emission $\check{y} \in \mathcal{Y}$ need to be summed, whereby the observation \check{y}_t needs to be equal to the emission \check{y} . An auxiliary variable is defined,

$$\gamma_t(\check{x}_i, \check{y}) = \begin{cases} \gamma_t(\check{x}_i) & \text{if } \check{y}_t = \check{y}, \\ 0 & \text{else.} \end{cases} \quad (3.34)$$

Afterwards, normalisation by the probabilities of every emission is performed,

$$\hat{P}(\check{y}|\check{x}_i, \lambda) = \hat{b}_{\check{x}_i}(\check{y}) = \frac{\sum_{t=1}^{|O|} \gamma_t(\check{x}_i, \check{y})}{\sum_{t=1}^{|O|} \gamma_t(\check{x}_i)}. \quad (3.35)$$

The iterative procedure for estimating the transition and emission probabilities works as follows:

1. Initialise \mathbf{A} , \mathbf{B} .
2. Calculate $\xi_t(\check{x}_i, \check{x}_j)$ for every time step t and every state \check{x}_i, \check{x}_j according to (3.32).

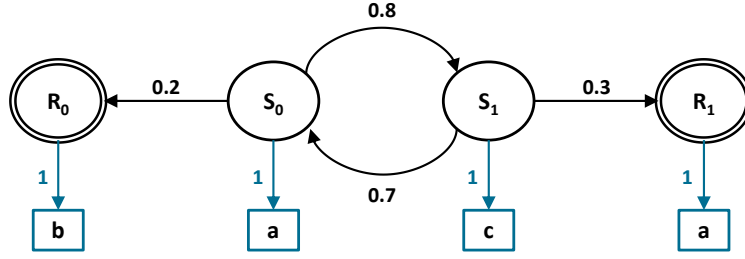


Figure 3.7: The HMM corresponding to the stochastic grammar defined in (3.6) to (3.11). The production rules are $\mathbb{P} = \{S_0 \xrightarrow{0.8} aS_1, S_0 \xrightarrow{0.2} b, S_1 \xrightarrow{0.7} cS_0, S_1 \xrightarrow{0.3} a\}$. The states S_0 and S_1 represent the nonterminal symbols while states R_0 and R_1 are terminating.

3. Calculate $\gamma_t(\check{x}_i)$ for every time step t and every state \check{x}_i according to (3.33).
4. Update the estimate \hat{a}_{ij} for every state \check{x}_i, \check{x}_j according to (3.31).
5. Update the estimate $\hat{b}_{\check{x}_i}(\check{y})$ for every state \check{x}_i and every emission \check{y} according to (3.35).
6. Repeat steps 2-5 until convergence of the estimates.

In the EM algorithm, steps 2 and 3 are called E-step and steps 4 and 5 M-step.

HMMs and MCs can as well be used to model grammars and are especially well suited for stochastic variants. In an HMM, the observed output corresponds to the terminal symbols, whereas the states of the underlying MC and the transitions between them describe the stochastic production rules. An example for the grammar from (3.6) to (3.11) is shown in Figure 3.7. As all emission probabilities are equal to 1, also an MC is suitable to model the grammar. However, HMMs and MCs can only represent the class of (stochastic) regular grammars.

3.4.3 Observable Operator Models

Observable operator models (OOMs) are introduced in [88] as a generalisation of HMMs. In contrast to HMMs, the model trajectory is not understood as a sequence of internal states, but as a sequence of linear operators. As a consequence, only methods from linear algebra and stochastics are needed to represent them. This leads to an efficient learning algorithm, which allows for adapting the model parameters based on training data.

A comparison of the different views of HMMs and OOMs is shown in Figure 3.8. On the left hand side the HMM view is depicted. The trajectory of the model is understood as a sequence of states $\check{x}_a\check{x}_b\check{x}_a\check{x}_b$ which is generated by the application of the same operator T . On the right hand side the OOM view is shown, which interprets the sequence $\check{x}_a\check{x}_b\check{x}_a\check{x}_b$ as being generated by the application of different operators T_a and T_b to an initial vector w_0 , i.e. $T_b(T_a(T_b(T_a(w_0))))$. In this case the operators are the observable part of the model, hence the name.

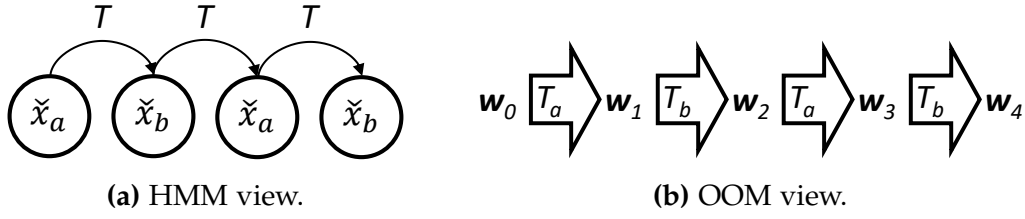


Figure 3.8: The HMM view of the model trajectory as a sequence of states and the OOM view as a sequence of linear operators [88].

The definition of HMMs can be reformulated to get to the definition of OOMs, which is shown in [88]. For HMMs, the probability of an observation can be calculated using matrices A and B from (3.23), which describe the state transition and the emission probabilities in each state, respectively. Based on B , the matrix of emission probabilities $B_{\check{y}_i}$ for some possible emission $\check{y}_i \in \mathcal{Y}$ is defined as a diagonal matrix with the emission probabilities for \check{y}_i in every state of the HMM on the diagonal. For example, the emission matrix B of the HMM in Figure 3.6 with the states sunny, cloudy, and rainy and the emissions umbrella and sunglasses is given by

$$B = \begin{pmatrix} 0.1 & 0.9 \\ 0.6 & 0.4 \\ 0.8 & 0.2 \end{pmatrix}. \quad (3.36)$$

Then, the emission matrix $B_{\check{y}_s}$ with $\check{y}_s = \text{sunglasses}$ is constructed as

$$B_{\check{y}_s} = \begin{pmatrix} 0.9 & 0 & 0 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0.2 \end{pmatrix}. \quad (3.37)$$

Using this matrix, an operator for an HMM can be defined as $T_{\check{y}_i} := A^T \cdot B_{\check{y}_i}$, with T being transposition. By adding the matrices of emission probabilities for every possible emission, the unit matrix results,

$$\sum_{\check{y} \in \mathcal{Y}} B_{\check{y}} = \mathbf{id} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & 1 \end{pmatrix}. \quad (3.38)$$

This arises from the condition that the probabilities of possible emissions in every state must sum up to 1. It follows that the matrix of state transition probabilities A can be reconstructed from the defined operators $T_{\check{y}_i}$,

$$A^T = A^T \cdot \mathbf{id} = A^T \cdot (B_{\check{y}_1} + \dots + B_{\check{y}_{|\mathcal{Y}|}}) = T_{\check{y}_1} + \dots + T_{\check{y}_{|\mathcal{Y}|}}. \quad (3.39)$$

Consequently, HMMs can be defined based on the operators by $\lambda = (\mathbb{R}^n, (T_{\check{y}})_{\check{y} \in \mathcal{Y}}, \pi)$. By relaxing some constraints of this formulation and replacing the $T_{\check{y}}$ of HMMs by $\tau_{\check{y}}$, the definition of OOMs results. An n -dimensional OOM is defined by

$$\mathcal{A} = (\mathbb{R}^n, (\tau_{\check{y}})_{\check{y} \in \mathcal{Y}}, w_0), \quad (3.40)$$

with

$$\begin{aligned} w_0 &\in \mathbb{R}^n && \text{initial state vector,} \\ \tau_{\check{y}} : \mathbb{R}^n &\mapsto \mathbb{R}^n && \text{linear operators,} \\ \mathcal{Y} &&& \text{alphabet of the observations with } |\mathcal{Y}| = n, \end{aligned}$$

and the following constraints

1. $\mathbf{1}w_0 = 1$, i.e. the values in w_0 sum up to 1,
2. The columns of the matrix $\mu = \sum_{\check{y} \in \mathcal{Y}} \tau_{\check{y}}$ sum up to 1,
3. For all sequences $\check{y}_{i_0} \dots \check{y}_{i_j}$ it holds: $\mathbf{1}\tau_{\check{y}_{i_j}} \circ \dots \circ \tau_{\check{y}_{i_0}} w_0 \geq 0$.

Here $\mathbf{1} = (1, \dots, 1)$ is the n -dimensional row vector consisting of 1 and \circ the concatenation of the operators. For HMMs, w_0 corresponds to the probability distribution of the initial state π . Since it is a probability distribution, no negative values are allowed. For the w_0 in OOMs, only the first condition that the values sum up to 1 must hold. However, negative values are possible. Similarly, the matrix μ corresponds to the matrix A of state transition probabilities in HMMs, which is a stochastic matrix. For OOMs, the matrix μ might as well contain negative values. According to the second condition above, the values must only sum up to 1. The third condition makes sure that positive values result when the OOM is used for calculating probabilities.

From this derivation of OOMs from HMMs by relaxing some of the limitations, it is clear that OOMs are a generalisation of HMMs. Moreover, in [88] it is shown that every two-dimensional OOM and every OOM with only positive entries in w_0 and μ is an HMM.

To learn an OOM from data, the dimension n needs to be set. If it is selected too low, the model is not able to represent the data. If it is selected too high, overfitting on the training data might occur. In the case of overfitting, memorisation of the training data leads to a loss of the ability to generalise and hence worse performance for unseen data. The same problem also exists for HMMs and neural networks (see Section 3.5).

For the estimation of the linear operators $\tau_{\check{y}}$ and the initial vector w_0 from data, so-called characteristic events A_i are required. These are defined by a partition of the set of output sequences of length k_o , \mathcal{O}^{k_o} , with k_o suitably large, into n subsets with n being the dimension of the underlying stochastic process,

$$\mathcal{O}^{k_o} = A_1 \cup \dots \cup A_n. \quad (3.41)$$

For the subsets A_i to be characteristic events, there must exist sequences $\bar{y}_1, \dots, \bar{y}_n$, where $\bar{y}_i = \check{y}_1 \dots \check{y}_{|\bar{y}_i|}$ denotes a sequence of symbols, such that the matrix of dimension $n \times n$

$$\left(P \left[A_i | \bar{y}_j \right] \right)_{i,j} \quad (3.42)$$

with

$$P \left[A_i | \bar{y}_j \right] = \sum_{\bar{a} \in A_i} P \left[\bar{a} | \bar{y}_j \right] \quad (3.43)$$

is nonsingular, i.e. possesses an inverse.

Based on the characteristic events A_1, \dots, A_n an OOM $\mathcal{A}(A_1, \dots, A_n)$ can be constructed, which is equivalent to an arbitrary OOM of the same stochastic process, but has the property of being interpretable. Interpretability is defined as the property that the n state vectors describe the probability of the occurrence of the n characteristic events, i.e. if the OOM is in state $\mathbf{w}_t = (w_t^1, \dots, w_t^n)$ at time step t , the probability of the characteristic event A_i should be equal to

$$P \left[A_i | \mathbf{w}_t \right] = w_t^i. \quad (3.44)$$

Interpretable OOMs $\mathcal{A}(A_1, \dots, A_n)$ possess properties that are required for the learning procedure, which are

1. $\mathbf{w}_0 = (P[A_1], \dots, P[A_n])$,
2. $\tau_{\bar{y}} \mathbf{w}_0 = (P[\bar{y}A_1], \dots, P[\bar{y}A_n])$,

where $\tau_{\bar{y}} = \tau_{\check{y}_n} \circ \dots \circ \tau_{\check{y}_0}$. Besides choosing the dimension n , the characteristic events A_i need to be selected in order to design the interpretable OOM. After this is done, the initial state vector \mathbf{w}_0 and the operators $\tau_{\check{y}}$ need to be estimated from the data. Based on a sequence $O = \check{y}_0 \check{y}_1 \dots \check{y}_{|O|}$, the initial state vector \mathbf{w}_0 is calculated. It follows from the properties of interpretable OOMs that $\hat{\mathbf{w}}_0 = (\hat{P}[A_1], \dots, \hat{P}[A_n])$ is a good first estimate. The probabilities of the characteristic events $\hat{P}[A_i]$ are estimated by counting the occurrences in the sequence O ,

$$\hat{P}[A_i] = \frac{\text{count}(\bar{a}_i \in O)}{\text{count}(\bar{a} \in O)}, \quad \bar{a}_i \in A_i. \quad (3.45)$$

The operators are estimated next. To do so, the second property of interpretable OOMs is needed. It follows from this property that for every sequence \bar{y}_j it holds

$$\tau_a(\tau_{\bar{y}_j} \mathbf{w}_0) = (P[\bar{y}_j a A_1], \dots, P[\bar{y}_j a A_n]). \quad (3.46)$$

From linear algebra it follows that an n -dimensional linear operator is uniquely determined by the values it takes on for n linearly independent vectors. From this, an estimate of the operator τ_a for some $a \in \mathcal{Y}$ is derived. First, n linearly independent vectors $\mathbf{v}_j := \tau_{\bar{y}_j} \mathbf{w}_0$ are estimated,

$$\hat{\mathbf{v}}_j = (\hat{P}[\bar{y}_j A_1], \dots, \hat{P}[\bar{y}_j A_n]), \quad j = 1, \dots, n. \quad (3.47)$$

Similar to the procedure above, the $\hat{P}[\bar{y}_j A_i]$ are estimated by counting,

$$\hat{P}[\bar{y}_j A_i] = \frac{\text{count}(\bar{y} \bar{a}_i \in O)}{|O| - |\bar{y}_j| - k_o + 1}, \quad \bar{a}_i \in A_i. \quad (3.48)$$

Here, $|O|$ is the length of sequence O , $|\bar{y}_j|$ is the length of sequence \bar{y}_j and k_o the length of the characteristic events A_i . The results of applying the linear operator τ_a are determined as

$$\hat{P}[\bar{y}_j a A_i] = \frac{\text{count}(\bar{y} a \bar{a}_i \in O)}{|O| - |\bar{y}_j| - k_o}, \quad \bar{a}_i \in A_i. \quad (3.49)$$

From this, two matrices are estimated,

$$\hat{V} = \begin{pmatrix} \hat{v}_1 & \dots & \hat{v}_n \end{pmatrix}, \quad (3.50)$$

$$\hat{W}_a = \begin{pmatrix} \hat{v}'_1 & \dots & \hat{v}'_n \end{pmatrix}, \quad (3.51)$$

with $\hat{v}'_j = \left(\hat{P}[\bar{y}_j a A_1], \dots, \hat{P}[\bar{y}_j a A_n] \right)$. Finally the estimation of the linear operator τ_a is given by

$$\hat{\tau}_a = \hat{W}_a \hat{V}^{-1}. \quad (3.52)$$

In [88] extensions of this basic idea of the learning procedure that make it more efficient are described. A problem of this learning procedure is that it cannot guarantee the third condition of the above definition of OOMs (3.40). Therefore, models that output negative “probabilities” are possible. The author argues that such “almost-OOMs” are nevertheless useful in practice.

3.4.4 Predictive State Representations

Predictive state representations (PSRs) are developed and refined in [35, 89, 90] and are closely related to OOMs. The key idea of PSRs is to represent the state of the system as a prediction of the future observations. Let \bar{y}_h be a sequence of observations $\bar{y}_h = \check{y}_0 \check{y}_1 \dots \check{y}_t$ called the history up to time step t . A test is a sequence $\bar{y}_t = \check{y}'_{t+1} \check{y}'_{t+2} \dots \check{y}'_{t+n}$ of possible future observations. A prediction for a test \bar{y}_t corresponds to the probability that the test will “succeed”, i.e. that the actual observations \check{y}_i will be equal to the predicted observations \check{y}'_i and is denoted by $P(\bar{y}_t | \bar{y}_h)$.

Both an uncontrolled and a controlled version, in which the state of the system can be influenced by taking actions, are defined. For the application considered in this thesis, only the uncontrolled variant is of interest. The uncontrolled version can be regarded as a controlled system with only a single action, which can therefore be omitted from the system equations.

In [89], the system-dynamics matrix D is introduced, which is a theoretical matrix that contains an infinite number of rows and columns. The rows are indexed by all possible histories \bar{y}_h including the empty history ϕ_h , while the columns correspond to all possible tests \bar{y}_t . The tests and histories are sorted by increasing length

and in lexicographical order within the same length. Each entry D_{ij} corresponds to $P(\bar{y}_t(i)|\bar{y}_h(j))$, where $\bar{y}_t(i)$ denotes the i th test in the set of all possible tests and $\bar{y}_h(j)$ the j th history from all possible histories,

$$\mathbf{D} = \begin{array}{c} \bar{y}_h(1) = \phi_h \\ \bar{y}_h(2) \\ \vdots \\ \bar{y}_h(j) \\ \vdots \end{array} \begin{array}{c} \bar{y}_t(1) \quad \dots \quad \bar{y}_t(i) \quad \dots \\ \hline P(\bar{y}_t(1)|\bar{y}_h(1)) \quad \dots \quad P(\bar{y}_t(i)|\bar{y}_h(1)) \quad \dots \\ P(\bar{y}_t(1)|\bar{y}_h(2)) \quad \dots \quad P(\bar{y}_t(i)|\bar{y}_h(2)) \quad \dots \\ \dots \quad \dots \quad \dots \quad \dots \\ P(\bar{y}_t(1)|\bar{y}_h(j)) \quad \dots \quad P(\bar{y}_t(i)|\bar{y}_h(j)) \quad \dots \\ \dots \quad \dots \quad \dots \quad \dots \end{array} . \quad (3.53)$$

The (non-unique) set of linear core tests $\mathcal{Q} = \{\bar{y}_t(i_1), \bar{y}_t(i_2), \dots\}$ is defined as the tests corresponding to the columns in \mathbf{D} such that all columns of \mathbf{D} are linearly dependent on the columns corresponding to the tests in \mathcal{Q} . Analogously, the set of core histories $\mathcal{H} = \{\bar{y}_h(j_1), \bar{y}_h(j_2), \dots\}$ is defined as a set of histories corresponding to the rows of \mathbf{D} such that all rows in \mathbf{D} are linearly dependent on the rows corresponding to the histories in \mathcal{H} . These sets are the maximal set of linearly independent columns or rows in \mathbf{D} . A PSR model is then defined by [90]

$$\mathcal{A}_{PSR} = (\mathcal{Q}, \mathcal{Y}, \mathbf{P}(\mathcal{Q}|\phi_h), \{\forall \check{y} \in \mathcal{Y} : \mathbf{M}_{\check{y}}\}, \{\forall \check{y} \in \mathcal{Y} : \mathbf{m}_{\check{y}}\}) \quad (3.54)$$

with

\mathcal{Q}	set of core tests,
\mathcal{Y}	alphabet of the observations,
$\mathbf{P}(\mathcal{Q} \phi_h)$	initial prediction vector/state,
ϕ_h	empty or null history,
$\{\forall \check{y} \in \mathcal{Y} : \mathbf{M}_{\check{y}}\}, \{\forall \check{y} \in \mathcal{Y} : \mathbf{m}_{\check{y}}\}$	model update parameters.

The i th entry of the initial prediction vector $\mathbf{P}(\mathcal{Q}|\phi_h)$ is defined to be $P(q_i|\phi_h)$ for all core tests $q_i \in \mathcal{Q}$. Moreover, $\mathbf{m}_{\check{y}}$ is a column vector such that $\mathbf{P}(\mathcal{Q}|\bar{y}_h)\mathbf{m}_{\check{y}} = P(\check{y}|\bar{y}_h)$ for all histories \bar{y}_h . The matrix $\mathbf{M}_{\check{y}}$ is defined to be a $|\mathcal{Q}| \times |\mathcal{Q}|$ matrix with the i th column being $\mathbf{m}_{\check{y}q_i}$ such that $P(\check{y}q_i|\bar{y}_h) = \mathbf{P}(\mathcal{Q}|\bar{y}_h)\mathbf{m}_{\check{y}q_i}$ for all histories \bar{y}_h . By this definition, the state of the system after history \bar{y}_h is $\mathbf{P}(\mathcal{Q}|\bar{y}_h)$ for all \bar{y}_h . The update equation for the prediction vector upon seeing a new observation after history \bar{y}_h is given by

$$\mathbf{P}(\mathcal{Q}|\bar{y}_h\check{y}) = \frac{[P(\check{y}q_i|\bar{y}_h) \dots P(\check{y}q_{|\mathcal{Q}|}|\bar{y}_h)]}{P(\check{y}|\bar{y}_h)} = \frac{\mathbf{P}(\mathcal{Q}|\bar{y}_h)\mathbf{M}_{\check{y}}}{\mathbf{P}(\mathcal{Q}|\bar{y}_h)\mathbf{m}_{\check{y}}}. \quad (3.55)$$

For learning a PSR model, the Suffix-History algorithm is proposed in [90]. At first, an estimate of the system dynamics matrix $\hat{\mathbf{D}}$ is required, from which the set of core tests $\hat{\mathcal{Q}}$ and core histories $\hat{\mathcal{H}}$ can be determined by finding linearly independent components. Based on these estimates, the model update parameters are found by

$$\mathbf{m}_z = \hat{\mathbf{P}}^{-1}(\hat{\mathcal{Q}}|\hat{\mathcal{H}})\hat{\mathbf{P}}(z|\hat{\mathcal{H}}), \quad z \in \{\check{y}, \check{y}q_i\}. \quad (3.56)$$

To estimate \hat{D} , the training sequence of observations $\bar{y} = \check{y}_0\check{y}_1 \dots \check{y}_{|\bar{y}|}$ is divided into suffixes that are treated as independent training sequences. Then, the entries of \hat{D} are determined by

$$\hat{P}(\bar{y}_t|\bar{y}_h) = \frac{\text{count}(\bar{y}_h, \bar{y}_t)}{\text{count}(\bar{y}_t)}, \quad (3.57)$$

where $\text{count}(\bar{y}_h, \bar{y}_t)$ is the number of times that the history \bar{y}_h is followed by the test \bar{y}_t and $\text{count}(\bar{y}_t)$ the number of times that \bar{y}_t appears in the training sequence. However, as the entries of \hat{D} are only estimates of the true values in D , the test for linear independence of the rows and columns cannot be performed with strict criteria. Therefore, the singular values of \hat{D} with a threshold on the uncertainty are used as the linear independence test instead and finally, \hat{Q} and \hat{H} are obtained to yield the model update parameters as defined in (3.56).

It is shown in [89] that uncontrolled PSRs are equivalent to interpretable OOMs, whose characteristic events resemble the PSRs' tests. Hence, PSRs possess more expressive power than HMMs.

3.5 Neural Networks

The initial idea of artificial neural networks was to reproduce the functionality of the brain, in which neurons receive stimuli from other neurons. If a certain threshold is crossed, a neuron “fires” and passes the stimulus on to the next neuron. In doing so, a stimulus can be reinforced or inhibited. Artificial neurons have a similar functionality. They receive “stimuli” in the form of numerical values at their inputs which are transmitted to other neurons using connections. The inputs are summed and passed through an activation function, which determines how strongly the inputs are passed on to the output, i.e. to what degree the neuron is activated. In addition, the connections between neurons have weights by which values that are transmitted using the connections are multiplied. If a weight is greater than 1, it has a reinforcing effect, otherwise it is inhibiting.

Neural networks are classified based on the structure of the connections between the neurons, which is part of the architecture of the network. Feedforward networks only have connections between neurons in consecutive layers and only into the direction of the output layer. An example of such a feedforward neural network is shown in Figure 3.9. The network depicted consists of one input layer, one hidden layer, and one output layer. The input layer has three neurons, such that the network can take three values $x = (x_1, x_2, x_3)$ as input. The input neurons only pass on the inputs without altering them. The output layer of the example network consists of two neurons, therefore the output of the network comprises two values $\hat{y} = (\hat{y}_1, \hat{y}_2)$. The connections between the neurons have weights u_i . For improved readability, these are only partly shown. In addition to the network's input x , the hidden layer has an additional input, called the bias, that is constantly set to 1 with weighted connections b . It enables the network to output a value different from 0 when the input is all 0. In this example, each neuron is connected to every neuron in the next layer. This

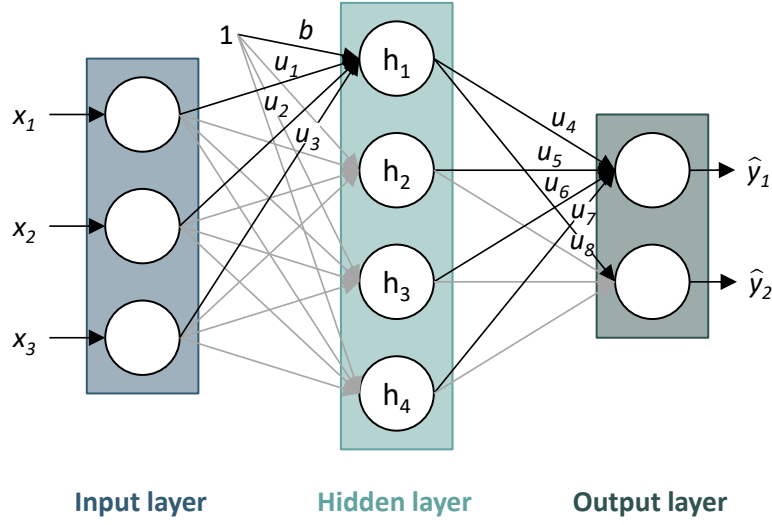


Figure 3.9: A simple feedforward neural network with one hidden layer.

is the typical architecture, which is called fully-connected or dense. However, other architectures with different kinds of connections exist.

The first neuron in the hidden layer calculates its activation h_1 , as follows,

$$h_1 = f(in_{h_1}) = f(u_1 \cdot x_1 + u_2 \cdot x_2 + u_3 \cdot x_3 + b). \quad (3.58)$$

Here, the activation function f can be one of the functions described in the next section and $in_{h_1} = u_1 \cdot x_1 + u_2 \cdot x_2 + u_3 \cdot x_3 + b$ represents the input that the neuron receives over its connections to the input neurons. Based on the results of the neurons in the hidden layer, the first output neuron calculates the following function to obtain \hat{y}_1 ,

$$\hat{y}_1 = f(in_{\hat{y}_1}) = f(u_4 \cdot h_1 + u_5 \cdot h_2 + u_6 \cdot h_3 + u_7 \cdot h_4). \quad (3.59)$$

Depending on the task, the number of hidden layers as well as the number of neurons per layer must be chosen appropriately. If the task is to classify images, one neuron per pixel is required in the input layer of a standard neural network architecture and one neuron per class that the images are supposed to be divided into in the output layer.

3.5.1 Activation & Output Functions

One of the common activation functions is the logistic function $\sigma(x)$ which belongs to the class of sigmoid functions,

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \quad (3.60)$$

It is common because of its simple derivative which is needed for the training procedure (see below),

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)). \quad (3.61)$$

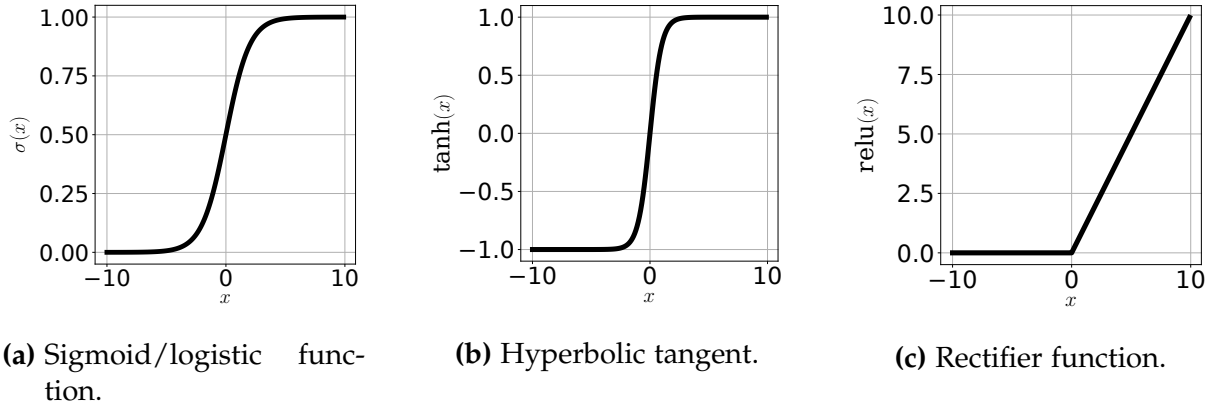


Figure 3.10: Examples for activation functions. Note the different scaling of the y-axis. The sigmoid function restricts the output to the interval $[0, 1]$, the hyperbolic tangent to $[-1, 1]$. The rectifier function is unbounded in positive direction.

Just like the logistic function, the hyperbolic tangent $\tanh(x)$ belongs to the class of sigmoid functions,

$$\tanh(x) = 1 - \frac{2}{\exp(2x) + 1}. \quad (3.62)$$

The derivative is very simple as well,

$$\tanh'(x) = 1 - \tanh^2(x). \quad (3.63)$$

Recently, the rectifier function is commonly applied. In the context of neural networks, it is also called rectified linear unit (ReLU),

$$\text{relu}(x) = \max(0, x), \quad (3.64)$$

with the derivative

$$\text{relu}'(x) = \begin{cases} 1 & x > 0, \\ 0 & \text{else.} \end{cases} \quad (3.65)$$

The three functions are shown in Figure 3.10 with example input values $x \in [-10, 10]$. The logistic function, which is in the context of neural networks usually called sigmoid function, squashes the output into the interval $[0, 1]$, while the hyperbolic tangent transforms it into the interval $[-1, 1]$. The rectifier function sets negative values to 0 and leaves other values unmodified.

Neural networks employed for classification or prediction should usually output a probability for each of the possible output values or classes $c \in \{1, 2, \dots, C\}$. Therefore, the last layer most often consists of the softmax function to normalise the output of the second to last layer to represent a probability distribution. In the context of neural networks, the unnormalised activations of the second to last layer are commonly called the logits $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_C)$. The softmax function is given by

$$\hat{y}_c = \frac{\exp(\tilde{y}_c)}{\sum_{i=1}^C \exp(\tilde{y}_i)}, \quad c = 1, 2, \dots, C. \quad (3.66)$$

3.5.2 Error Functions

The goal of training a network is to adjust the connection weights u_i between the neurons such that the error between the desired and the actual output is minimised. To do so, an error (or loss) function needs to be defined. One possibility is the mean squared error (MSE),

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \frac{1}{M} \sum_{m=1}^M (y_m - \hat{y}_m)^2. \quad (3.67)$$

Here, $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M)$ is the actual output of the network and $\mathbf{y} = (y_1, y_2, \dots, y_M)$ the desired one. The factor $1/2$ is introduced to simplify the derivative.

Another common error function is the cross-entropy. It describes the difference between two probability distributions P and Q as

$$\mathbb{H}(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x), \quad (3.68)$$

where \mathbb{E} is the expected value. In the context of neural networks, P describes the true probability distribution that is to be learnt and Q the distribution estimated by the network. For details on probability and information theory see [91].

3.5.3 Training

Neural networks are most commonly trained using a procedure called “Backpropagation of Error”, which is based on gradient descent. Consider the function $h(x)$, which resembles the MSE, and its gradient $h'(x)$ shown in Figure 3.11. The goal of gradient descent is to find the minimum of $h(x)$. To do so, the gradient $h'(x)$ for some x is calculated. If it is positive, like in the right half of the figure, x needs to be decreased to move closer to the minimum and if the gradient is negative, x needs to be increased, i.e. $x_{new} \propto x - h'(x)$. Usually, the error “landscape” of neural networks is very complex with many local minima and saddle points.

Training neural networks consists of two steps. In the first step, the input is propagated “forwards” from the input neurons through the neurons in the hidden layers to the output neurons. At the output, the actual result $\hat{\mathbf{y}}$ is compared to the desired output \mathbf{y} . From these, the error function calculates the error E , which is then propagated “backwards” from the output neurons through the neurons in the hidden layers to the input neurons. On the way, the contribution of each connection weight u_i to the error is calculated and the weights are updated accordingly. From this backwards step, the procedure gets its name.

To determine the contribution of a weight u_i to the error E , the derivative of the error function at the weight is calculated. For the network shown in Figure 3.9, the derivative at e.g. weight u_4 is

$$\frac{\partial E}{\partial u_4} = \frac{\partial E}{\partial \text{in}_{\hat{y}_1}} \cdot \frac{\partial \text{in}_{\hat{y}_1}}{\partial u_4}. \quad (3.69)$$

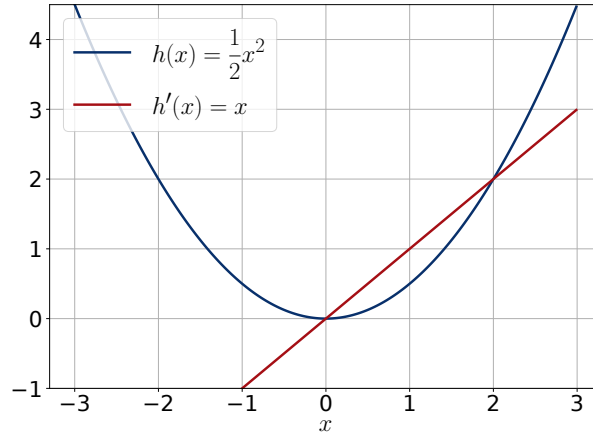


Figure 3.11: A function and its gradient.

With

$$in_{\hat{y}_1} = u_4 \cdot h_1 + u_5 \cdot h_2 + u_6 \cdot h_3 + u_7 \cdot h_4 \quad (3.70)$$

it follows

$$\frac{\partial in_{\hat{y}_1}}{\partial u_4} = h_1. \quad (3.71)$$

From using the chain rule and the MSE error function defined by (3.67) with $M = 1$ it results

$$\begin{aligned} \frac{\partial E}{\partial u_4} &= \frac{\partial E}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial in_{\hat{y}_1}} \cdot \frac{\partial in_{\hat{y}_1}}{\partial u_4} \\ &= (y_1 - \hat{y}_1) \cdot f'(in_{\hat{y}_1}) \cdot h_1. \end{aligned} \quad (3.72)$$

The weight u_4 is then updated by

$$u_{4,new} = u_4 - \eta \frac{\partial E}{\partial u_4}. \quad (3.73)$$

Here, η is called the learning rate. It is set to a value smaller than 1 since a learning rate that is chosen too big can lead to overshooting the desired goal. However, if it is chosen too small, convergence is slow. Selecting the learning rate and possibly adapting it during training is a complex topic in itself. Several optimisers have been proposed, while the Adam optimiser presented in 2015 [92] is a popular choice. It adapts the initial learning rate individually for different parameters based on estimates of the first and second order moments of the gradient. The algorithm got its name from the term “adaptive moment estimation”. An overview of several optimisers is given in e.g. [91].

For weights that are not directly connected to an output neuron, the chain rule needs to be applied multiple times. In addition, the error of each other neuron connected in the forward direction needs to be considered. For example, the first neuron

in the hidden layer with weight u_1 is connected to the output \hat{y}_1 and \hat{y}_2 . Therefore it follows

$$\begin{aligned}\frac{\partial E}{\partial u_1} &= \frac{\partial E}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial \text{in}_{\hat{y}_1}} \cdot \frac{\partial \text{in}_{\hat{y}_1}}{\partial h_1} \cdot \frac{\partial h_1}{\partial \text{in}_{h_1}} \cdot \frac{\partial \text{in}_{h_1}}{\partial u_1} + \frac{\partial E}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \text{in}_{\hat{y}_2}} \cdot \frac{\partial \text{in}_{\hat{y}_2}}{\partial h_1} \cdot \frac{\partial h_1}{\partial \text{in}_{h_1}} \cdot \frac{\partial \text{in}_{h_1}}{\partial u_1} \\ &= (y_1 - \hat{y}_1) \cdot f'(\text{in}_{\hat{y}_1}) \cdot u_4 \cdot f'(\text{in}_{h_1}) \cdot x_1 + (y_2 - \hat{y}_2) \cdot f'(\text{in}_{\hat{y}_2}) \cdot u_8 \cdot f'(\text{in}_{h_1}) \cdot x_1 \\ &= ((y_1 - \hat{y}_1) \cdot f'(\text{in}_{\hat{y}_1}) \cdot u_4 + (y_2 - \hat{y}_2) \cdot f'(\text{in}_{\hat{y}_2}) \cdot u_8) \cdot f'(\text{in}_{h_1}) \cdot x_1\end{aligned}\quad (3.74)$$

For deep neural networks, i.e. networks with several hidden layers, the terms expand accordingly. Backpropagation is a very efficient procedure for error minimisation since the intermediate results of the derivatives can be stored in the neurons and only need to be calculated once on the way back from the output to the input. To determine the exact error, the complete data would need to be processed by the network and the overall error calculated. However, this is computationally intense. On the other hand, the gradient of a single example is very inexact and noisy. As a compromise, the training data is usually divided into batches which comprise e.g. $M = 1024$ examples. When doing so, the training procedure is referred to as stochastic gradient descent.

One big problem of training neural networks is overfitting, in which case the network memorises the training data and is not able to generalise. This leads to very poor performance for data not seen during training. To obtain a good estimate of the network's performance on unseen input, a part of the available data is kept as a test set and not used for training. Usually, the data is further divided to obtain a validation set. This validation set is not used for training either but serves for fine-tuning the network architecture like the number of neurons and layers. When doing so, the network is indirectly adapted to the validation set. Therefore, only the performance on the test set, which did not influence the training in any way, is an objective estimate of the network's performance on new data. Overfitting can be recognised by an increasing accuracy on the training set and a decreasing performance on the validation set. As mentioned above, the test set should not be used to monitor the training progress.

Several methods to avoid overfitting have been proposed. A commonly used procedure is the so-called dropout, which is presented in [93]. Dropout deactivates a certain percentage of randomly chosen neurons during training. In each run, different neurons are selected such that the network needs to learn how to compensate for these failures, which leads to better generalisation capabilities. When the network is employed, the dropout layer is inactive. See e.g. [91] for an overview of further methods to prevent overfitting.

3.5.4 Evaluation Metric

One of the usual metrics to measure the performance of neural networks is the top k accuracy. It provides the percentage of cases that the true class is among the top k outputs of the network sorted by probability. Given a number of examples M , the model outputs a set of vectors $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M\}$. Each vector $\hat{y}_m = (\hat{y}_{m,1}, \dots, \hat{y}_{m,C})$ represents the result of the softmax operation for the number of classes C . Suppose

that $\pi_{\hat{y}_m}$ denotes a (non-unique) permutation of $\{1, \dots, C\}$ which sorts the elements of \hat{y}_m by size, i.e.

$$\pi_{\hat{y}_m} : \{1, \dots, C\} \rightarrow \{1, \dots, C\}, \quad (3.75)$$

$$i \mapsto j = \pi_{\hat{y}_m}(i) \quad (3.76)$$

such that $\hat{y}_{m, \pi_{\hat{y}_m}(i)} \geq \hat{y}_{m, \pi_{\hat{y}_m}(j)} \forall 1 \leq \pi_{\hat{y}_m}(i) < \pi_{\hat{y}_m}(j) \leq C$.

Then, the top k elements T_k of \hat{y}_m are defined as the inverse permutation $\pi_{\hat{y}_m}^{-1}$ of the first k indices, $k \leq C$, of the vector $\mathbf{y}'_m = (y'_{m,1}, \dots, y'_{m,C}) = (\hat{y}_{m, \pi_{\hat{y}_m}^{-1}(1)}, \dots, \hat{y}_{m, \pi_{\hat{y}_m}^{-1}(C)})$, i.e.

$$T_k(\hat{y}_m) = (\pi_{\hat{y}_m}^{-1}(1), \dots, \pi_{\hat{y}_m}^{-1}(k)). \quad (3.77)$$

Given the set $Y = (y_1, y_2, \dots, y_M)$ of true labels $y_m \in \{1, \dots, C\}$ for each of the M examples, the top k accuracy for $k \leq C$ is defined as

$$\text{acc}(\hat{Y}, Y, k) = \frac{1}{M} \sum_{m=1}^M \mathbb{1}_{T_k(\hat{y}_m)}(y_m) \cdot 100\%, \quad (3.78)$$

where $\mathbb{1}_A(x)$ denotes the indicator function

$$\mathbb{1}_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise.} \end{cases} \quad (3.79)$$

3.5.5 Recurrent Neural Networks

Recurrent neural networks (RNNs) differ from feedforward neural networks in having neurons that are connected to themselves, i.e. they use their own output \mathbf{h}_{t-1} , which is sometimes referred to as the hidden state of the network, as an extra input in the next time step t ,

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t), \quad (3.80)$$

see also Figure 3.12. Because of this feedback they are especially well suited to process sequential data. To visualise how the input from the previous time steps is incorporated into processing the current time step, Figure 3.13 shows an RNN unrolled over time. In this figure, each grey box represents the same RNN in different time steps.

For training RNNs, backpropagation is used with some modifications, which is called backpropagation through time [95]. The network needs to be unrolled as shown in Figure 3.13 and the connections must be trained together because these are actually the same connections in different time steps. In the end, there must only be one weight value per connection.

RNNs are very powerful tools in theory. In [96, 97] it is shown that an RNN with finitely many neurons and sigmoid activation resembles a Turing machine. However, it was soon noticed that RNNs are hard to train using backpropagation through time, e.g. [98]. The main problem is that an RNN behaves like a very deep feedforward neural network when unrolled over many time steps. When using backpropagation

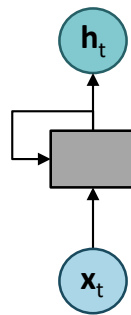


Figure 3.12: RNNs have connections which form a feedback from their own output to their input [94].

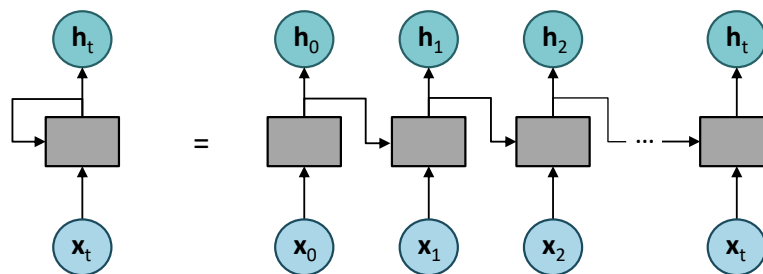


Figure 3.13: An RNN unrolled over time [94].

through time, this can lead to two situations. If parts of the gradient are big, continuous multiplication by applying the chain rule leads to an exploding overall gradient. This problem can easily be solved by clipping the error signal, which sets the gradient to a pre-defined value if a certain threshold is crossed. Much worse is the second case, the vanishing gradient. If small values below 1 are multiplied for many time steps, the gradient vanishes and learning is no longer possible. The update equation for the weights in (3.73) shows that the gradient is multiplied by the learning rate and then subtracted from the current weight. If the gradient - in this example $\frac{\partial E}{\partial u_4}$ - is very close to 0, the weight does not change noticeably and no learning takes place. The problem of many multiplications can already be anticipated from (3.74) for only one hidden layer. To overcome this problem, Long Short-Term Memory networks were developed by Hochreiter and Schmidhuber in 1997 [11]. This variant of an RNN is presented in the following section.

3.5.6 Long Short-Term Memory

Long Short-Term Memory networks (LSTMs) were introduced in the year 1997 [11] as a solution to the vanishing gradient problem. However, they gained importance, especially in the field of speech processing, only recently when big companies like Google, Apple, and Amazon started to incorporate LSTMs into their products in the years 2015/2016 [99–102].

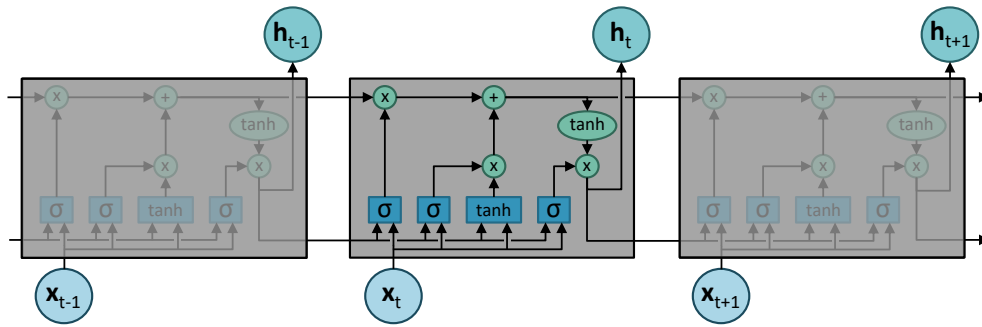


Figure 3.14: Internal structure of an LSTM cell, unrolled over three time steps [94].

LSTMs got their name from an interpretation of short- and long-term memory in neural networks. In [11], the connection weights are considered as the long-term memory since they change rather slowly during training and are fixed afterwards. The short-term memory is defined as the activation of the neurons depending on the current input and, through the recurrent connections, also on the previous input. The long short-term memory of the presented LSTMs refers to their ability to keep more time steps of the current sequence in memory than standard RNNs.

In their work, Hochreiter and Schmidhuber present a new architecture and a new gradient based training procedure. They combine a variant of backpropagation through time and an adapted version of real-time recurrent learning [103] for training. An improved training procedure is later introduced in [104]. LSTMs contain so-called memory cells, which allow a constant error flow without vanishing gradients. Consequently, LSTMs can learn long-term dependencies of about 1000 time steps in the input without losing the ability to cope with shorter time dependencies.

The main parts of an LSTM cell (see Figure 3.14) and the way the vanishing gradient problem is solved, are the gates. These are input, output, and forget gate, whereby the forget gate was introduced two years later in [105]. In Figure 3.14, the gates are represented by the blue rectangles with the σ symbol for the sigmoid function. The left gate is the forget gate, the middle one is the input gate, and the right one the output gate. The forget gate processes both the output from the previous time step h_{t-1} and the current input to the network x_t . These are concatenated before being input to the sigmoid function.

Every LSTM cell has an internal state C_t , which is controlled by the gates. In Figure 3.14, the state is represented by the line at the top which goes straight through the cell from left to right. The state from time step t is input for the time step $t+1$. During training, the gates learn to decide which information is important and should be included in the state. The forget gate learns when the state of the cell should be reset, i.e. forgotten. In the field of speech processing this is necessary when translating texts, for example. If a sentence is completely translated and the content of the next sentence is not related to the previous one, the internal state of the LSTM should be reset and the previous sentence forgotten.

The update equations for the output of the forget gate f , the input gate i , and the output gate o , as well as the cell state C and the output h are defined by

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \quad (3.81)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \quad (3.82)$$

$$C_t = f_t C_{t-1} + i_t \tanh(W_C[h_{t-1}, x_t] + b_C), \quad (3.83)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \quad (3.84)$$

$$h_t = o_t \tanh(C_t), \quad (3.85)$$

where W_j are the weight matrices, b_j the biases with $j \in \{f, i, c, o\}$, x_t is the input, and $[\dots, \dots]$ represents vector concatenation. In [106], “peephole connections” were introduced to allow the gates to see the cell state. The update equations for the gates are then given by

$$f_t = \sigma(W_f[C_{t-1}, h_{t-1}, x_t] + b_f), \quad (3.86)$$

$$i_t = \sigma(W_i[C_{t-1}, h_{t-1}, x_t] + b_i), \quad (3.87)$$

$$o_t = \sigma(W_o[C_t, h_{t-1}, x_t] + b_o). \quad (3.88)$$

By the design of the LSTM cell, it is possible to keep the internal state constant over many time steps. It is therefore protected from the problems which arise from a vanishing gradient. LSTMs are, as RNNs in general, not restricted in their expressiveness and equivalent to universal Turing machines. Therefore, they can learn the language of palindromes, which is shown in [107, 108].

3.5.7 Gated Recurrent Unit

Inspired by the LSTM, the Gated Recurrent Unit (GRU) was proposed in 2014 [37]. It uses gates as well to mitigate the vanishing gradient problem of standard RNNs. However, it is much simpler than the LSTM. Instead of three gates, the GRU only uses two, which are called reset and update gate, i.e. GRUs do not have an output gate. Like the forget gate in LSTMs, the reset gate learns when the current state of the GRU should be reset, and the update gate determines how much of the hidden state h_{t-1} is kept in the current state h_t . Figure 3.15 shows the internal structure. Here, the two boxes labelled σ are the gates with the left one being the reset and the right one being the update gate.

The update equations for the reset gate r , the update gate z , the candidate output \tilde{h} , and the actual output h are given by

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \quad (3.89)$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \quad (3.90)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1})), \quad (3.91)$$

$$h_t = z_t h_{t-1} + (1 - z_t) \tilde{h}_t, \quad (3.92)$$

where W_j , U_j are weight matrices with $j \in \{r, z, h\}$ and \odot denotes element-wise multiplication.

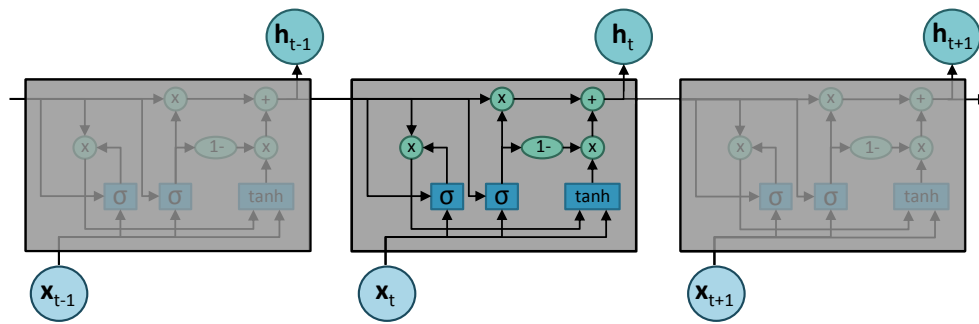


Figure 3.15: Internal structure of a GRU cell, unrolled over three time steps.

Studies that evaluate the differences between gated RNN architectures like LSTM and GRU show that none of them is fundamentally superior to the other [109,110], although the LSTM is assumed to be the more powerful architecture. Since it has more parameters and is therefore harder to train, the difference might not be relevant in practice.

Chapter 4

Modelling of Agile Radar Emissions

This chapter describes the existing emission model, which can be found in the open literature. Afterwards, this chapter introduces the concept adapted for this thesis and demonstrates it using example emitters. Parts have been published in [60].

4.1 Introduction

As described in Chapters 1 and 2, the information gathered about radar systems is traditionally stored in databases. However, modern radars are becoming more agile as they perform multiple tasks simultaneously and choose waveform parameters adaptively. The emissions are not based on operational modes any more and the relationships between different emission patterns cannot be efficiently represented in static databases. Therefore, new techniques for modelling the emissions of multifunction radars are needed. In the following, a possible solution is described.

4.1.1 Hierarchical Emission Model

The works [12–21] present and extend a model that considers the radar emissions as a language with an inherent hierarchical structure. A visualisation of this model is shown in Figure 4.1. On the lowest level there are pulses, represented by rectangles. In analogy to natural language, pulses correspond to letters. On the next level, the letters are used to form words, which in turn can be used to form commands. A sequence of commands constitutes a task on the highest level.

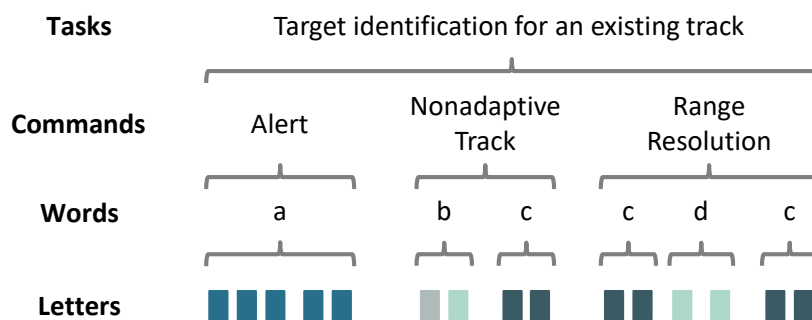


Figure 4.1: Hierarchical model of the emissions of a multifunction radar [20]. The radar emissions are modelled as a hierarchical language.

Command	Words	Command	Words
Four-word search	$[w_1 w_2 w_4 w_5]$	Track maintenance	$[w_1 w_7 w_7 w_7]$
	$[w_2 w_4 w_5 w_1]$		$[w_2 w_7 w_7 w_7]$
	$[w_4 w_5 w_1 w_2]$		$[w_3 w_7 w_7 w_7]$
	$[w_5 w_1 w_2 w_4]$		$[w_4 w_7 w_7 w_7]$
Three-word search	$[w_1 w_3 w_5 w_1]$		$[w_5 w_7 w_7 w_7]$
	$[w_3 w_5 w_1 w_3]$		$[w_6 w_7 w_7 w_7]$
	$[w_5 w_1 w_3 w_5]$		$[w_1 w_8 w_8 w_8]$
Acquisition (ACQ)	$[w_1 w_1 w_1 w_1]$		$[w_2 w_8 w_8 w_8]$
	$[w_2 w_2 w_2 w_2]$		$[w_3 w_8 w_8 w_8]$
	$[w_3 w_3 w_3 w_3]$		$[w_4 w_8 w_8 w_8]$
	$[w_4 w_4 w_4 w_4]$		$[w_5 w_8 w_8 w_8]$
	$[w_5 w_5 w_5 w_5]$		$[w_6 w_8 w_8 w_8]$
Nonadaptive track (NAT) or Track maintenance	$[w_1 w_6 w_6 w_6]$		$[w_1 w_9 w_9 w_9]$
	$[w_2 w_6 w_6 w_6]$		$[w_2 w_9 w_9 w_9]$
	$[w_3 w_6 w_6 w_6]$		$[w_3 w_9 w_9 w_9]$
	$[w_4 w_6 w_6 w_6]$		$[w_4 w_9 w_9 w_9]$
	$[w_5 w_6 w_6 w_6]$		$[w_5 w_9 w_9 w_9]$
Range resolution	$[w_7 w_6 w_6 w_6]$	Fine track maintenance (FTM)	$[w_6 w_9 w_9 w_9]$
	$[w_8 w_6 w_6 w_6]$		$[w_7 w_7 w_7 w_7]$
	$[w_9 w_6 w_6 w_6]$		$[w_8 w_8 w_8 w_8]$
ACQ, NAT or FTM	$[w_6 w_6 w_6 w_6]$		$[w_9 w_9 w_9 w_9]$

Table 4.1: Possible combinations of words of the “Mercury” emitter and the connection to commands [20].

The modelling is shown using an example emitter called “Mercury”, which is based on a real anti-aircraft defence multifunction radar that was modified such that it could be published. This emitter can make use of nine different words, which mainly differ in the values of the PRF. The connection between sequences of words and the commands of the radar is shown in Table 4.1.

4.1.2 Word Embedding

In order to process the radar language by an algorithm, a numerical representation is needed. The same holds true for words of a natural language, for which several approaches have been suggested. The simplest solution to convert words to vectors is the so-called one-hot encoding (see left part of Figure 4.2). In this encoding, the dimension of the word vector equals the number of words in the vocabulary. Each word is assigned a position in the vector such that the vector consists of all zeros except for a single one at the position of the represented word.



One-Hot Encoding		Word Embedding	
Berlin	= [1, 0, 0, 0, 0, 0, 0, 0, ..., 0]	Berlin	= [0.85, 0.52, 0.12, ..., 0.37]
Paris	= [0, 1, 0, 0, 0, 0, 0, 0, ..., 0]	Paris	= [0.83, 0.54, 0.11, ..., 0.92]
Germany	= [0, 0, 1, 0, 0, 0, 0, 0, ..., 0]	Germany	= [0.27, 0.71, 0.81, ..., 0.39]
France	= [0, 0, 0, 1, 0, 0, 0, 0, ..., 0]	France	= [0.28, 0.73, 0.83, ..., 0.94]
 Dimension = vocabulary size = huge		 Dimension = n << vocabulary size	

Figure 4.2: Examples for the numerical representation of words using one-hot encoding and word embedding.

One-hot encoding is simple but very inefficient. Moreover, relations between words are not represented as every word has the same distance in vector space to every other word. One solution for solving these problems is to use word embeddings, which are dense vector representations of words (see right part of Figure 4.2). Several algorithms for learning word embeddings were suggested, e.g. [23–28]. This thesis focuses on a software package patented by Google, called word2vec [23,24]. A visualisation of the architectures contained in word2vec is shown in Figure 4.3. Both architectures represent a neural network. With continuous bag-of-words (CBOW), it is trained to predict a target word from its context words. Here, the order of the context words is not important, which resembles the bag-of-words representation sometimes encountered in natural language processing (NLP). Skip-gram, which is a kind of language model as well and further described below, uses the opposite approach and learns to predict the context words from the target word. After training, the matrix W acts as a lookup table for the vector representations. These algorithms use the idea that similar words should appear in similar contexts and therefore create context-dependent vector representations. By training a neural network to predict a target word from its context words (or vice versa), embeddings are found in which similar words are close to each other.

In this thesis, the Skip-gram model is used with negative sampling to learn vector representations for the radar language. The objective of Skip-gram is to maximise the average log probability of the context words given the target word,

$$\frac{1}{k_s} \sum_{s=1}^{k_s} \sum_{-k_c \leq j \leq k_c, j \neq 0} \log P(w_{s+j} | w_s), \quad (4.1)$$

with $w_1 w_2 \dots w_{k_s}$ a sequence of k_s training words and k_c the size of the context. Since the cost of computing $P(w_{s+j} | w_s)$ using the softmax function as defined in (3.66) is proportional to the vocabulary size, less computationally intense alternatives are suggested. One of them is negative sampling, which defines a new objective in the training. With negative sampling, the goal is to learn to distinguish between true

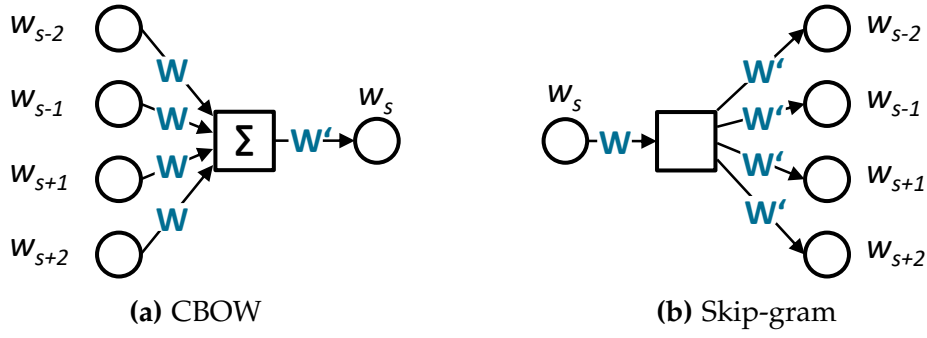


Figure 4.3: The architectures for learning word embeddings contained in word2vec [23].

context words w_{s+j} and “noise” words w_i (negative samples). To do so, the term $\log P(w_{s+j}|w_s)$ in (4.1) is replaced by

$$\log \sigma(\mathbf{v}'_{w_{s+j}} \mathbf{v}_{w_s}) + \sum_{i=1}^{k_n} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-\mathbf{v}'_{w_i} \mathbf{v}_{w_s}) \right]. \quad (4.2)$$

Here, $\sigma(\cdot)$ is the sigmoid function as defined in (3.60) and \mathbf{v}_w corresponds to the word vector of w defined by W , whereas \mathbf{v}'_w corresponds to the vector representation defined by W' (see Figure 4.3). $P_n(w)$ represents the noise distribution for the negative samples and the number of negative samples is given by k_n . The objective defined by (4.2) therefore encourages the maximisation of the probability that the target word and the context words occur together, while minimising the estimated probability that the target word and the noise words occur together.

To determine how similar two words w_i and w_j are, the cosine similarity between the vectors \mathbf{v}_{w_i} and \mathbf{v}_{w_j} is used, which is defined as the cosine of the angle between the corresponding word vectors,

$$\text{cos}_{sim}(w_i, w_j) = \frac{\mathbf{v}_{w_i} \mathbf{v}_{w_j}}{\|\mathbf{v}_{w_i}\|_2 \|\mathbf{v}_{w_j}\|_2}. \quad (4.3)$$

The authors of [25] observe that semantic relationships between words are captured by word embeddings. For example, the vector offset between word pairs that represent singular and plural are almost constant, e.g. $\mathbf{v}_{apple} - \mathbf{v}_{apples} \approx \mathbf{v}_{car} - \mathbf{v}_{cars}$. Moreover, it is noticed that the word closest (in terms of cosine similarity) to the vector resulting from calculating $\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman}$ is \mathbf{v}_{queen} and $\mathbf{v}_{Paris} - \mathbf{v}_{France} + \mathbf{v}_{Italy} \approx \mathbf{v}_{Rome}$.

4.1.3 Contributions

The hierarchical emission model described in Section 4.1.1 is adapted to better suit the needs of ELINT and an additional modelling level is introduced. Furthermore, word embeddings, as described in Section 4.1.2, are employed for the symbols of the

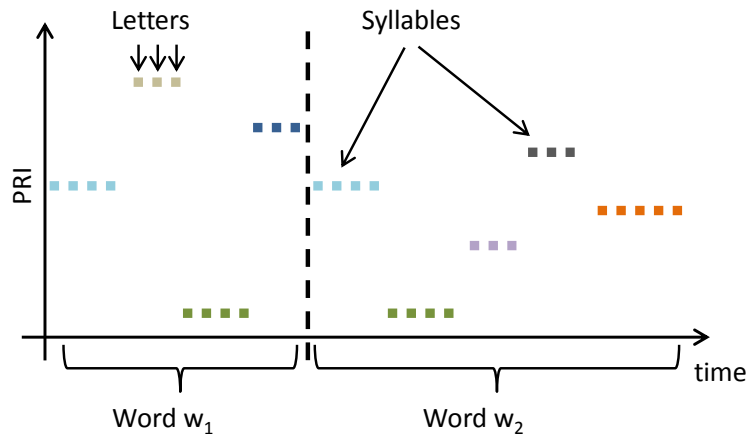


Figure 4.4: Example for modelling the different PRI levels as syllables [60].

radar language. As word embeddings are context-dependent vector representations, they allow for the context-based analysis of radar signals.

Section 4.2 describes the adapted hierarchical emission model in detail and Section 4.3 presents an example. Section 4.4 details the implementation of word embeddings for the language of the example emitters and Section 4.5 provides a summary.

4.2 Adapted Hierarchical Emission Model

For modern multifunction radars, which perform several functions in parallel by using time-multiplexing, the hierarchical emission model described above seems to be well suited in general. However, the work presented in Section 4.1.1 assumes that it is known how the modelled radar functions internally, which is most often not the case in an ELINT application. Without this knowledge, radar tasks and commands cannot be modelled in the way it is suggested in the presented publications. For example, the task and commands given in Figure 4.1 cannot be defined based on knowledge acquired only from the emissions. The hierarchical emission model is therefore adapted to better suit the needs of ELINT.

In the adapted emission model, tasks are replaced by a broader category of functions. Moreover, the definition of a command is changed. Commands are now divided into categories, based on how many syllables they contain and whether the corresponding pulses/letters make use of a high, medium, or low PRF.

For some structures of radar emissions, a level between letters and words might be useful. An example is a radar that uses a stepped PRI modulation (see Figure 4.4). Each word is formed by combining some PRI values from a set of possibilities. Some words can contain the same PRIs, but without a level between pulses and words, this relationship cannot be modelled. Therefore, another level is introduced, which is called syllables in analogy to natural language.

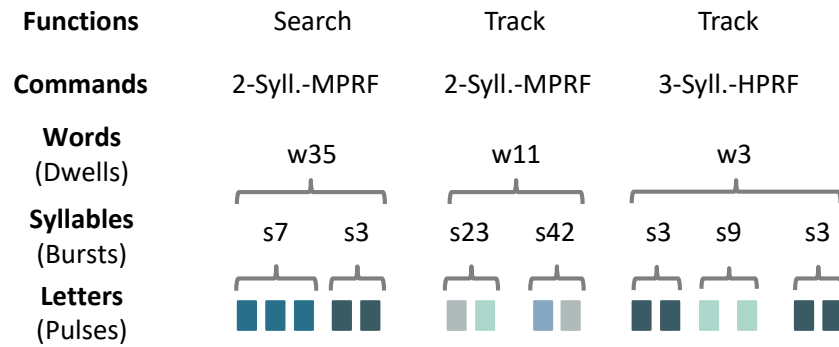


Figure 4.5: The adapted hierarchical emission model.

In summary, the following levels of modelling are used (see also Figure 4.5):

- **Letters:** Letters are characterised by the pulse parameters of the radar, e.g. PRF, RF, and PW, as well as the modulations for each or a MOP.
- **Syllables:** Letters are combined to form syllables, which correspond to radar bursts or CPIs.
- **Words:** Common combinations of syllables form words, e.g. radar dwells.
- **Commands:** Commands can be regarded as word types or classes.
- **Functions:** Functions describe the purpose of the emissions, e.g. searching or tracking targets.

Like in a natural language, words are combinations of syllables that have a meaning, but if two words share a syllable they do not necessarily have the same meaning. In that analogy, commands correspond to parts of speech.

4.3 Modelling the Emissions of Example Emitters

The emission model is shown using simulations of an airborne radar, which were developed during a study on radar resource management at Fraunhofer FKIE. For the simulations, eight different scenarios were defined. These contain, amongst others, scenarios with raids of hostile aircraft, fighters, jamming, and missiles. Three different radar resource management algorithms were implemented. The method used for resource management determines how the radar distributes the available resources, e.g. time, across different tasks and how it chooses the waveform parameters. Consequently, it determines the complexity and agility of the emissions. The first resource management technique implemented uses a Quality of Service (QoS) approach. For every task, a performance model calculates how much utility is obtained by assigning resources to it. A central resource manager allocates the resources to the different tasks such that the utility is maximised. The approach is described in [69]. The other

Table 4.2: Number of symbols used by the different resource management methods.

Symbol	QoS	Rules-v1	Rules-v2
Letter	18	13	18
Syllable	25 380	103	27 786
Word	26 653	21	34 440
Command	10	2	10
Function	3	3	3

resource management methods use one of two different sets of rules to choose the waveform parameters and allocate time to different tasks. The more sophisticated version of the rules is described in [6].

Because the resource management algorithm highly influences the structure of the emissions, the radar can be regarded as three distinct radars with the same vocabulary but a different grammar. This means that letters, syllables, words, commands, and functions are the same. What differs is the number of symbols that are actually used and the order in which they appear, as well as the agility and frequency. In the following, the three radar versions are abbreviated with QoS, Rules-v1, and Rules-v2 for the more sophisticated version of the rule-based approaches.

Based on the pulse parameters saved during the simulations of the radars, the emission model is developed. Table 4.2 depicts the number of symbols that are actually emitted by each radar type, while Tables 4.3 to 4.5 show the letters, syllables, words, and commands of the radars. The QoS and the Rules-v2 radar use a great variety of different emissions, while the Rules-v1 is rather simple.

Table 4.3 shows the letters that can be used, ordered by ascending PRF and numbered from 1 to 18. These letters correspond to pulses and are characterised by PRF, RF, and PW. The definition based on these features is specific to the example radar. If the radar used e.g. a MOP, it would need to be considered in the definition of letters. The example radar emits pulses in the medium pulse repetition frequency (MPRF) and high pulse repetition frequency (HPRF) range as defined in Section 2.1.1. The MPRF range contains the letters l_1 to l_8 , all other letters belong to the HPRF range. The values are taken from the open radar literature (see Chapter 5 in [7]) to avoid dealing with sensitive data.

From the letters, syllables can be formed. The possible combinations are shown in Table 4.4. The radar modelled here chooses a certain number of repetitions of letters for each syllable. For example, s_1 consists of 16 times l_1 , s_2 repeats l_1 17 times, and so on. The letters in the MPRF range can be repeated up to 256 times and those in the HPRF range up to 4096 times. As is typical for an airborne radar, it integrates many repetitions of the same pulse. This behaviour can be seen from the way the syllables are constructed. For ground- or sea-based radars the emission model would be different. In total, the airborne example radar can make use of 40 338 different

Table 4.3: Letters of the example radar.

Name	PRF [Hz]	RF [MHz]	PW [μ s]
l_1	8880	10 000	12.387
l_2	10 850	10 000	10.138
l_3	12 040	10 000	9.136
l_4	12 820	10 000	8.580
l_5	14 110	10 000	7.795
l_6	14 800	10 000	7.432
l_7	15 980	10 000	6.883
l_8	16 770	10 000	6.559
l_9	75 420	10 000	1.201
l_{10}	77 930	10 000	1.201
l_{11}	79 310	10 000	1.201
l_{12}	82 210	10 000	1.201
l_{13}	86 150	10 000	1.201
l_{14}	90 200	10 000	1.201
l_{15}	96 200	10 000	1.201
l_{16}	98 512	10 000	1.201
l_{17}	100 120	10 000	1.201
l_{18}	105 380	10 000	1.201

Table 4.4: Syllables of the example radar.

Name	Combination
$s_1 - s_{241}$	$\{16, 17, \dots, 256\} \cdot l_1$
$s_{242} - s_{482}$	$\{16, 17, \dots, 256\} \cdot l_2$
$s_{483} - s_{723}$	$\{16, 17, \dots, 256\} \cdot l_3$
$s_{724} - s_{964}$	$\{16, 17, \dots, 256\} \cdot l_4$
$s_{965} - s_{1205}$	$\{16, 17, \dots, 256\} \cdot l_5$
$s_{1206} - s_{1446}$	$\{16, 17, \dots, 256\} \cdot l_6$
$s_{1447} - s_{1687}$	$\{16, 17, \dots, 256\} \cdot l_7$
$s_{1688} - s_{1928}$	$\{16, 17, \dots, 256\} \cdot l_8$
$s_{1929} - s_{5769}$	$\{256, 257, \dots, 4096\} \cdot l_9$
$s_{5770} - s_{9610}$	$\{256, 257, \dots, 4096\} \cdot l_{10}$
$s_{9611} - s_{13451}$	$\{256, 257, \dots, 4096\} \cdot l_{11}$
$s_{13452} - s_{17292}$	$\{256, 257, \dots, 4096\} \cdot l_{12}$
$s_{17293} - s_{21133}$	$\{256, 257, \dots, 4096\} \cdot l_{13}$
$s_{21134} - s_{24974}$	$\{256, 257, \dots, 4096\} \cdot l_{14}$
$s_{24975} - s_{28815}$	$\{256, 257, \dots, 4096\} \cdot l_{15}$
$s_{28816} - s_{32656}$	$\{256, 257, \dots, 4096\} \cdot l_{16}$
$s_{32657} - s_{36497}$	$\{256, 257, \dots, 4096\} \cdot l_{17}$
$s_{36498} - s_{40338}$	$\{256, 257, \dots, 4096\} \cdot l_{18}$

syllables, of which 25 380 appear in the simulation data of the QoS radar, 27 786 in the data of the Rules-v2 radar, and only 103 in the data of the simple Rules-v1 radar.

Syllables are combined to form words. The simulated radar is able to combine all MPRF and all HPRF syllables of the same length, respectively. When doing so, it adheres to the following rules. In the MPRF range, no syllables occur more than once in a word. Also, the syllables are only combined in ascending order of the PRF. Words in the HPRF range repeat a syllable only in the beginning of the word or the complete word consists of only one syllable. Moreover, only one syllable can occur more than once in each word. Based on these rules, 26 101 507 different words can be formed. In the simulation data of the QoS radar, only a small fraction consisting of 26 653 words occurs. The Rules-v2 radar uses 34 440 words, and the Rules-v1 only 21. Note that the emission model presented in Section 4.1.1 uses the Mercury emitter with nine words as an example. Although only a small fraction of the words are actually used, all of them are considered in the emission model to keep the flexibility. Table 4.5 shows a summary for words and commands. For example, w_1 contains the syllables s_1

Table 4.5: Words and commands of the example radar.

		Command	Word
MPRF		2-Syllable Word	$w_1 - w_{6748}$
		3-Syllable Word	$w_{6749} - w_{20244}$
		4-Syllable Word	$w_{20245} - w_{37114}$
		5-Syllable Word	$w_{37115} - w_{50610}$
		6-Syllable Word	$w_{50611} - w_{57358}$
		7-Syllable Word	$w_{57359} - w_{59286}$
		8-Syllable Word	$w_{59287} - w_{59527}$
HPRF		2-Syllable Word	$w_{59528} - w_{443627}$
		3-Syllable Word	$w_{443628} - w_{3593247}$
		4-Syllable Word	$w_{3593248} - w_{26101507}$

and s_{242} , which repeat the letters l_1 and l_2 16 times, respectively. Because of the very large number of words, a complete table connecting them to syllables is not shown here. Instead, Table 4.5 displays them organised by commands, which form the next hierarchy level. These are characterised by a separation into MPRF and HPRF as well as the number of syllables they contain.

Functions are on the next and highest level of the hierarchy. The radar performs the functions “search”, “confirm”, and “track”, along with “calibration” and “missile link”. The last two, however, are only executed from time to time and do not create interceptable radar emissions. Therefore, gaps are introduced in the simulation data where those functions would have been performed and consequently, they are not considered in the modelling. The function “track” shows the most diversity when choosing the waveform parameters. For the functions “search” and “confirm” only the words with eight and four syllables of the MPRF and HPRF range are used. Besides the diversity, there are only a few words - in total 30 of the 26 653 words contained in the QoS simulation data - which are used by more than one function.

4.4 Word Embedding for the Radar Language

The simulation data of the three resource management versions is divided into a training, a validation, and a test set. The training set contains six of the eight scenarios with 1440 Monte Carlo runs per emitter type. The validation set consists of two scenarios with 240 runs per emitter type. For testing, all eight scenarios are used with 480 Monte Carlo runs per emitter type. For letters, a very large amount of data is available and hence, only two runs per scenario and emitter type are considered for training and testing. The word vectors are learnt with the TensorFlow [111] frame-

Table 4.6: word2vec parameters used. Values for the QoS and Rules-v2 radars are shown before and those for the Rules-v1 after the slash. Batch size and skip window are the same for all.

Parameter	Letter	Syllable	Word	Command	Function
Embedding size	8/8	64/16	64/8	4/2	2/2
Batch size	128	128	128	128	128
Skip window	30	30	30	30	30
Num sampled	4/4	64/4	64/4	2/1	1/1

work for Python [112] on the training and validation set with the word2vec example implementation¹ by the TensorFlow authors as a basis. The implementation employs Skip-gram with negative sampling. See Table 4.6 for the parameters used.

When using word embeddings, an additional word vector for unknown symbols (represented as UNK) is introduced. In NLP, infrequent words like names are replaced by UNK. In the considered application, the word vector for UNK allows for more flexibility with handling new symbols which did not appear in the training or validation data. As the data from the test set is not used to create the word embeddings, a few syllables and words appear that are mapped to UNK. In the test data of the QoS radar, there are 917 unknown syllables (0.013% of the test data) and 2931 unknown words (0.21%). The test data of the Rules-v2 radar contains 612 unknown syllables (0.007%) and 2400 unknown words (0.139%). For the Rules-v1 radar, there are no unknown symbols and no unknown letters, commands, or functions appear for all radars.

As a result of learning the embeddings, a dictionary of the symbols of each emitter is obtained. Let ω be a symbol in the set of symbols Ω^l at modelling level $l \in \{\text{letters, syllables, words, commands, functions}\}$, including the UNK symbol. For each emitter e in the set of emitters $\mathcal{E} = \{\text{QoS, Rules-v1, Rules-v2}\}$, the set Ω_e^l only contains the symbols that appear in its training or validation set. Here, $\Omega^l = \bigcup_{e \in \mathcal{E}} \Omega_e^l$ corresponds to the global dictionary containing the symbols from all emitters, while Ω_e^l is the individual dictionary of emitter e .

4.5 Summary

Multifunction radars can be modelled as systems that speak a language. Like a natural language, the radar language consists of different levels, which are called letters, words, commands, and tasks in the original approach described in the literature. To better suit the needs of ELINT, the emission model is adapted in this thesis to contain

¹ https://android.googlesource.com/platform/external/tensorflow/+/2db2230841e851e80374b6c5d9e6d9d7f35e0384/tensorflow/examples/tutorials/word2vec/word2vec_basic.py

letters, syllables, words, commands, and functions. The adapted emission model is shown using three variants of a simulated airborne radar with different resource management methods of varying complexity.

In NLP, word embeddings are used to represent words as dense vectors, which are learnt by a neural network. The resulting word embedding contains context-dependent vector representations of the words and also captures semantic relationships, which allows for calculations like $v_{king} - v_{man} + v_{woman} \approx v_{queen}$. This thesis applies word embeddings to the symbols (i.e. letters, syllables, words, commands, and functions) of the radar language to enable context-dependent analysis of radar signals and take advantage of the dense vector representation when further processing the radar symbols with neural networks.

Chapter 5

Prediction of Radar Emissions

This chapter provides an overview of the literature on predicting radar emissions. Afterwards, it presents the approaches proposed in this thesis and provides an evaluation. According to the definitions given in Section 2.4, predicting the emissions constitutes the third problem of EW. This chapter also addresses the fourth problem of learning or training a model. Parts have been published in [60,62].

5.1 Introduction

The goal of ELINT is to collect information about radar emitters that helps to identify their type and purpose and hence the threat that they pose to a platform. Based on the gathered information, models of the radar emitters can be built, which allow for the prediction of future emissions. These forecasts are especially useful for sorting simultaneously received signals by emitter (deinterleaving, see Section 2.2.3) and generating tailored signal interference to degrade the performance of the radar (jamming).

For non-agile or semi-agile radars, the prediction of the next emission is much easier than for modern agile radars since they do not change their mode and hence their waveform parameters very often. Agile radars, however, are capable of adapting their waveform parameters to the current situation such that the emissions change rapidly and repeatedly. Therefore, more sophisticated methods for prediction are needed. In the following sections, solutions for this problem are presented and evaluated.

5.1.1 Related Work

The authors of the hierarchical emission model presented in Section 4.1.1 suggest to define a formal radar grammar, which can be used to generate an FSM and an HMM (see Chapter 3). In [12–21], the focus is set on solving the second problem of EW (see Section 2.4), the decoding of the radar’s internal state sequence, which corresponds to a sequence of commands defined in Table 4.1 of Section 4.1.1. The modelling suggested in the PhD thesis by Visnevski [15] requires the manual definition of the grammar, including the production rules, by an expert. The resulting syntactic model is optimised to explain the radar’s emissions and as a consequence, it can generate emissions that are not part of the actual radar language. Therefore, it cannot be used to predict upcoming radar symbols.

Based on the emission model suggested by Visnevski et al. [12–21], the authors of [34] present a method for predicting radar emissions. Like Visnevski et al., they use the Mercury emitter (see Section 4.1.1) for evaluation. Instead of using HMMs, PSRs (see Section 3.4.4) are employed for state recognition and prediction. The goal is to predict a combination of words that form a command. First, they estimate the current functional state of the radar. To do so, they follow the same approach as [12–21], i.e. for every model, they determine the probability that it has generated the current sequence of words. Afterwards, the PSR for every state makes a prediction. These are fused based on the probabilities assigned to each state to yield the final results.

Apart from the hierarchical emission model, the authors of [36] predict radar emissions based on a discretised PDW representation, consisting of PRI and PW. After discretisation, PRI and PW are encoded into one-hot vectors individually. Afterwards, embeddings are learnt to reduce the dimensionality. This is also done separately for PRI and PW, however, the authors do not provide details about the embedding. The embedded representations are concatenated and input to a GRU (see Section 3.5.7), which learns to predict the next discretised value of PRI and PW. To make the prediction more robust, the GRU is trained with corrupted data. Each pair of PRI and PW is tagged to belong to a radar pulse or noise segment, while the training labels are only generated from pulses. For evaluation, five different “stream classes” are defined. These correspond to waveform parameters with constant statistical features up to some variation of the PRI within certain limits. For each class, a GRU is trained to make predictions. The considered sequences are rather short and consist of 20 to 25 pulses.

5.1.2 Contributions

Based on the adapted hierarchical emission model presented in Chapter 4, methods for predicting the next emission of a radar emitter are proposed. As opposed to the work by Visnevski et al. [12–21], the radar model does not require manual definitions by an expert but it is instead learnt from the available data sequences.

Two methods are of particular interest, the LSTM and the MC. This comparison is especially interesting since the two approaches are in contrast to each other in terms of memory. The LSTM possesses an internal state (its “memory”) and employs a gating mechanism to mitigate the vanishing gradient problem of standard RNNs. Consequently, it is able to learn about long-term dependencies in the input. In contrast to LSTMs, MCs exhibit the “memoryless” property, i.e. the probabilities for the next state of the MC only depend on the current state and not on the previous states. This chapter demonstrates whether memory is important for predicting the signals of a multifunction radar by a thorough comparison of LSTMs and MCs, both with ideal and corrupted data.

In contrast to [34], which also employs the hierarchical emission model, the presented evaluations consider all modelling levels. Moreover, the examined emitter types are much more complex than those considered in the previous work. The emitter in [34] can make use of nine words, while the radar types considered in this thesis

Table 5.1: Training, validation, and test set per emitter type. The numbers in brackets correspond to the reduced number of runs for letters.

Training		Validation		Test	
[runs]	[h]	[runs]	[h]	[runs]	[h]
1440 (12)	64.0	480 (12)	25.3	480 (16)	19.3

use up to 34 440 words. Also the method using GRUs presented in [36] is evaluated with less complex data. Although it does not use the hierarchical emission model, the methods are directly comparable on the lowest level, i.e. pulses or letters. For parts of the other modelling levels, the complexity can at least be estimated. The five classes defined in [36] resemble commands in the adapted hierarchical emission model and the specific realisations of the emissions correspond to words. The authors state that for each class, 5000 training sequences are generated with random PRI. However, for the class with the maximal difference between the upper and lower bound of the PRI, which is $500\text{ }\mu\text{s}$, this results in at most 100 different realisations after discretisation with steps of $5\text{ }\mu\text{s}$. For the other classes, the number of possible realisations (or words) is even less.

As a result of this chapter, behavioural models of the example emitters at different modelling levels are obtained. Moreover, the impact of corrupted data as well as the input encoding is shown. Section 5.2 provides details on the implementation and training of the LSTMs and the MCs. Section 5.3 describes and presents the evaluation, and Section 5.4 summarises the chapter.

5.2 Approaches

Using the simulations of the example emitters presented in Section 4.3, LSTMs and MCs are learnt as behavioural models of the emitters. For training or learning the models, the data is split into training, validation, and test set (see Table 5.1). The same sets were also used for learning the vector representations (see Section 4.4), i.e. six of the eight scenarios are used for training, the other two for validation, and all eight scenarios are used for testing. The validation set is only used for the LSTMs since for the MCs, there is no need to try several parameters. For the letters, a very large amount of data is available which would have taken several month to train. Hence, the LSTMs and MCs for predicting letters are only trained and tested on two runs per scenario.

Tables 5.1 and 5.2 provide an overview over the number of runs, the duration of the simulations, as well as the amount of data available per emitter. Table 5.2 shows that there is a slight data imbalance in the simulations because the syllables of the QoS radar tend to contain more letters, which results in less syllables, words, commands, and functions.

Table 5.2: Amount of data in the sets of each emitter. The values for letters are the reduced amount.

Symbol	Radar	Train. [MB]	Val. [MB]	Test [MB]
Letter	QoS	713.2	811.2	993.3
	Rules-v1	613.8	711.1	850.3
	Rules-v2	661.1	795.3	924.4
Syllable	QoS	149.1	46.9	49.2
	Rules-v1	187.5	66.5	63.6
	Rules-v2	185.1	64.9	62.7
Word	QoS	58.7	18.4	19.3
	Rules-v1	68.2	23.3	22.8
	Rules-v2	71.1	25.2	24.0
Command	QoS	34.4	10.9	11.3
	Rules-v1	40.9	14.2	13.7
	Rules-v2	42.0	14.6	14.1
Function	QoS	16.0	5.1	5.3
	Rules-v1	18.9	6.6	6.4
	Rules-v2	19.4	6.8	6.6

5.2.1 Long Short-Term Memory

Per resource management technique and modelling level, one LSTM is trained, resulting in 15 different and independent networks. The LSTMs are implemented using Python and TensorFlow. Figure 5.1 shows the general architecture of the networks. The input consists of sequences of strings that are read from a text file containing the names of the symbols according to the definitions given in Tables 4.3 to 4.5 of Section 4.3. The word embedding layer maps the strings to vectors, which are arrays of floating point values. Those are then processed by the LSTM layers with peephole connections (see (3.86) to (3.88)), which are followed by a dropout layer with a rate of 50 % to mitigate overfitting. Since this layer is only active during training, it is depicted in grey. The final dense layer before softmax consists of one neuron per symbol of the modelling level (e.g. 103 for the syllables of the Rules-v1 radar, see Table 4.2) and ReLU activation is applied. The activations of the dense layer (often called the “logits” of the network) are normalised by the softmax layer to yield a probability distribution over the possible next symbols. For training, the input consists of symbol pairs (ω_i, ω_j) where $\omega_i, \omega_j \in \Omega^l$ are symbols in the set of symbols Ω^l at modelling level $l \in \{\text{letters, syllables, words, commands, functions}\}$, including the UNK symbol. Given the symbol ω_i , the label corresponds to the next symbol ω_j . It is encoded into a one-hot vector such that it can be interpreted as a probability distribution to match the desired output of the network and allow for training with the cross-entropy loss.

For each network, a batch size of 120 is used except for those trained on the letters, for which the batch size is set to 12. Batch sizes of 120 and 12 were chosen because

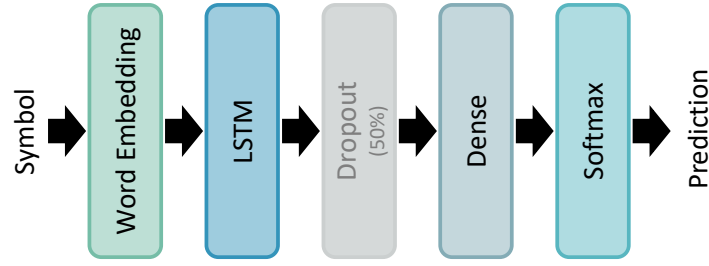


Figure 5.1: General architecture of the networks for prediction. The dropout layer is shown in grey since it is only active during training.

they are common factors of 1440 and 480. With a batch size of 120, 120 simulation runs are fed to the network in parallel with one symbol per run and batch. This means that the input to the network consists of a 120 dimensional symbol vector containing the symbols from each simulation run at a given time, i.e. for step t

$$\mathcal{B}_t = (\omega_t^1, \omega_t^2, \dots, \omega_t^{120}) \quad (5.1)$$

and for step $t + 1$

$$\mathcal{B}_{t+1} = (\omega_{t+1}^1, \omega_{t+1}^2, \dots, \omega_{t+1}^{120}), \quad (5.2)$$

with the superscript denoting the number of the simulation run. For an easier explanation, the runs are displayed in the order $1, 2, \dots, 120$ here, but actually appear shuffled. The batches \mathcal{B} are input to the network in parallel such that there exist 120 internal states and a vector of 120 error values, for which the gradient is calculated. As described in Section 3.5.3, batch training is a compromise between the computationally expensive procedure of calculating the error on the complete data and updating the weights based on a noisy gradient due to a low sample size. The networks are trained using the Adam optimiser with a learning rate of 0.002 and the cross-entropy loss (see (3.68)), and 50 % dropout after the last LSTM layer. The LSTM cell state is kept between batches. Table 5.3 shows the number of layers and the number of cells per layer employed for each modelling level. The values were found by testing several architectures and choosing the one with the smallest loss on the validation set.

Table 5.3: LSTM parameters for the different modelling levels of the example radars.

Radar	Parameter	Letter	Syllable	Word	Command	Function
QoS	# layers	2	2	2	2	1
	# cells/layer	256	128	256	32	16
Rules-v1	# layers	1	1	2	1	1
	# cells/layer	64	64	32	4	4
Rules-v2	# layers	2	2	2	2	1
	# cells/layer	32	32	64	16	16

5.2.2 Markov Chain

As a second approach, an MC is implemented per radar and modelling level, also resulting in 15 different and independent MCs. A first order MC is used, which means that only the current state \check{x}_t influences the probability of the next state \check{x}_{t+1} according to (3.20). Due to the large number of syllables and words emitted by the QoS and Rules-v2 radar, higher order MCs would rapidly cause memory and performance issues.

Based on the data in the training set, which consists of tuples (ω_i, ω_j) as defined above, the probability of the next symbol ω_j given the symbol ω_i is estimated for every symbol pair $\omega_i, \omega_j \in \Omega^l$ and every emitter e in the set of emitters $\mathcal{E} = \{\text{QoS}, \text{Rules-v1}, \text{Rules-v2}\}$ by

$$\hat{P}_e(\omega_j|\omega_i) = \frac{\text{count}(\omega_i, \omega_j)}{\sum_{\omega \in \Omega^l} \text{count}(\omega_i, \omega)}, \quad (5.3)$$

where $\text{count}(\omega_i, \omega_j)$ is the number of times that ω_j follows ω_i in the training set of emitter e .

The symbols ω_i are mapped to integers before they are input to the MC. This is done based on the index of the symbol in the embedding matrix and hence, the transition matrices of the MCs also contain the index of the UNK symbol. However, as it does not appear in the training data $\hat{P}_e(\text{UNK}|\omega_i) = 0$ for all $e \in \mathcal{E}$. If the data is corrupted and therefore contains symbols from a different emitter, those are mapped to UNK.

An MC and not an HMM was chosen for predicting the next symbol because the estimation of the internal state, which would be the functions search, confirm, or track, introduces additional complexity which is not necessary to solve the task.

5.2.3 Comparison Methods

For further performance assessment, the LSTMs and MCs are compared to three simple prediction strategies. The first one is random guessing with uniform probability for the symbols. The second strategy predicts the symbols that most frequently appear in the training data, hence the prediction is always the same. The third one consists of repeating the last symbols seen.

5.3 Experimental Results

In the first part of the evaluation, the data is ideal, i.e. without missing or additional symbols which potentially belong to a different radar. Corrupted data might be caused by missed detections of the emitters' signals and errors made in the deinterleaving or symbol extraction step (see Section 2.3). In the second part, the evaluation is performed with data containing missing and additional symbols.

The results are obtained on the test set, which was not used for training the LSTMs and the MCs. Given are the top 1, top 5, top 10, and top 20 prediction accuracies

$\text{acc}_e(\hat{Y}, Y, k)$, $k = 1, 5, 10, 20$ (see Section 3.5.4). Here, $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_M)$ with the batch size $M \in \{12, 120\}$ is the actual output of the prediction method for the data of emitter e and $Y = (y_1, \dots, y_M)$ the desired output, i.e. the labels. The values for guessing represent the probabilities that the correct symbol is among the randomly chosen ones if 1, 5, 10, or 20 symbols are selected. For letters, commands, and functions there are less than 20 symbols to choose from such that an accuracy of 100 % can be easily achieved. From the way the word embedding is designed, it follows that the LSTMs and the MCs can always predict one extra symbol (UNK). To keep the comparison fair, the UNK symbol is accounted for in the results for the random guessing strategy. The strategy of always predicting the most frequent symbols chooses the most frequent symbol for the top 1 accuracy, the 10 most frequent symbols for top 10 accuracy, and so on. For repeating the last symbols, the top 20 accuracy is obtained by predicting the last 20 symbols in reversed order. These might not necessarily be 20 different symbols such that an accuracy of less than 100 % is achieved for letters, commands, and functions.

5.3.1 Evaluation Under Ideal Conditions

Figure 5.2 shows the prediction accuracies for the different strategies with ideal data. Detailed results for the MCs and the LSTMs are given in Tables 5.4 and 5.5. Table 5.4 additionally shows the results of the repeating strategy for letters, syllables, words, and commands since it is the best simple strategy in terms of top 1 accuracy for these symbols except for the commands of the QoS radar. In Table 5.5, the results of the strategy that predicts the most frequent symbols for commands and functions is shown in comparison to the results of the MCs and LSTMs. This approach is the best simple strategy for the commands of the QoS radar and the functions of all emitter types.

As is typical for an airborne radar, the simulated emitter integrates many pulse repetitions. For example, word $w_{26101507}$ consists of 16 384 repetitions of the same letter l_{18} . If the radar repeats this word, the sequence of that letter becomes even longer. As one can see from the top 1 accuracy of always predicting the most common letter l_{18} , it makes up more than 85 % of the test data (see Figure 5.2) and repeating the last letter results in a top 1 accuracy of more than 99.9 %, as seen in Table 5.4. These sequences seem to be too long for an LSTM to learn. For all simulated radar variants, smaller networks learnt to always predict l_{18} . Larger networks were able to predict l_{18} if it was correct. However, if l_{18} was not the next letter, the larger LSTMs predicted all possible letters with equal probability. Hence, the LSTMs could learn when l_{18} was not the right answer but could not tell which letter was correct. Further increasing the network size did not solve the problem. The reason for this might be the heavily imbalanced training data. Both outcomes from training basically result in the same performance as always predicting the most frequent letters. However, this emitting behaviour is specific to airborne radars and the results will be much different for ground radars. In contrast to the LSTMs, the MCs are able to predict the next letter with a top 1 accuracy of about 99.9 % and achieve the exact same results

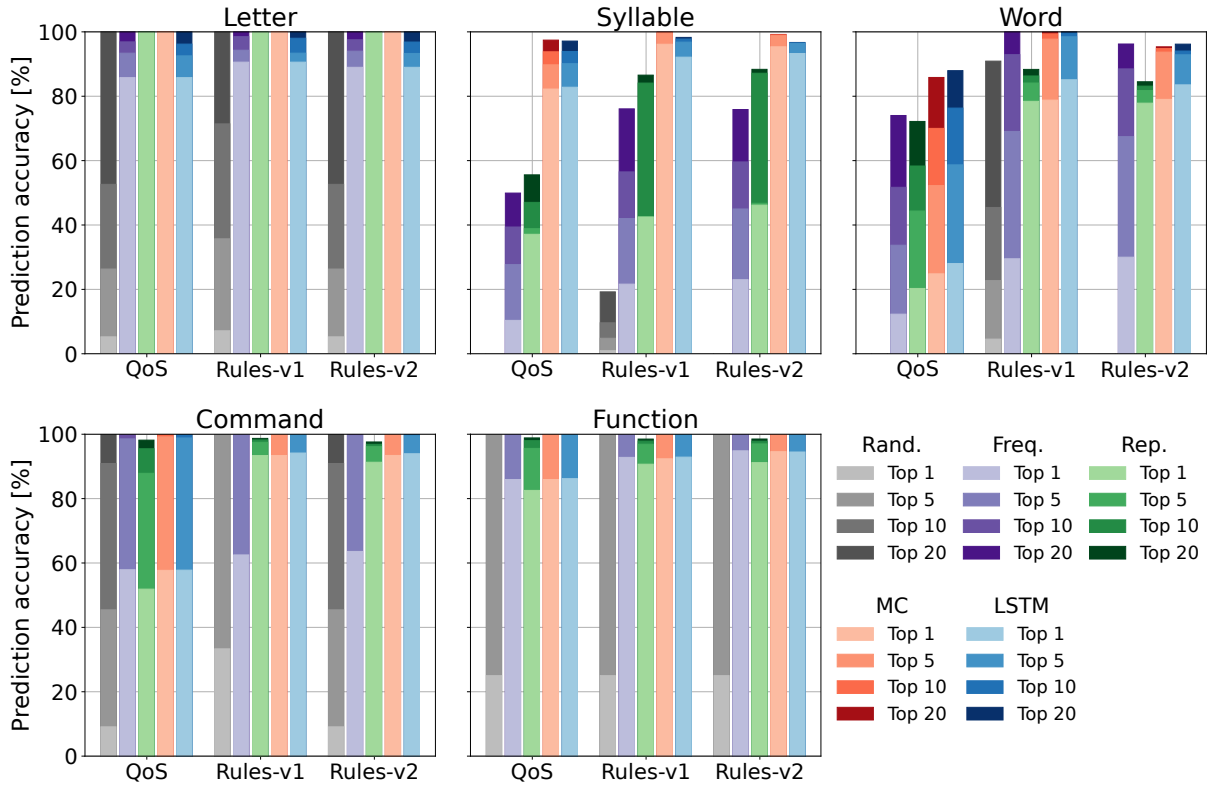


Figure 5.2: Prediction accuracies of the different methods, which are random guessing (Rand.), predicting the most frequent symbols (Freq.), repeating the last symbols (Rep.), as well as MC and LSTM.

as the repeating strategy (see Table 5.4). The entries on the diagonal of the MCs' transition matrices are all very close to 1, which means that the MCs basically repeat the last letter. Considering the data, this seems to be the best strategy.

For the QoS radar, the LSTM and the MC achieve a large improvement for the syllables with a top 1 accuracy of 82.8 % and 82.24 %, respectively (see Table 5.4). The best simple strategy, namely repeating the last syllable, reaches 37.11 % accuracy. As the number of syllables used by the QoS and the Rules-v2 radar is very large, no bar can be seen for random guessing in Figure 5.2 because the top 20 accuracy is below 0.1 %. Also for predicting the syllables of the other two radars, the MCs and LSTMs clearly achieve higher accuracies than the simple strategies. For the syllables of the QoS radar, the LSTM provides slightly better results, but the differences are not notable. The MCs outperform the LSTMs for the syllables of the two rule-based emitters. From the results for the syllables of the Rules-v2 radar, it is seen that although it uses 27 786 different syllables, the 20 most frequent make up 75.88 % of the test data. For the QoS radar, this value is 49.92 %, which is also significant when considering that it uses 25 380 different syllables during the simulation runs.

Words are hardest to predict, but the LSTMs and the MCs still outperform the simple strategies, while the LSTMs achieves a higher accuracy than the MCs for all emitter types. As seen in Table 5.4, the LSTM can improve the result from 20.29 %

Table 5.4: Prediction accuracies [%] of the MCs, the LSTMs, and the repeating strategy. The best results are marked in bold.

		QoS			Rules-v1			Rules-v2		
		MC	LSTM	Rep.	MC	LSTM	Rep.	MC	LSTM	Rep.
Letter	Top 1	99.93	85.78	99.93	99.90	90.59	99.90	99.92	88.99	99.92
	Top 5	100.00	92.55	99.93	100.00	93.38	99.90	100.00	93.29	99.92
	Top 10	100.00	96.20	99.93	100.00	98.02	99.90	100.00	96.84	99.92
	Top 20	100.00	100.00	99.93	100.00	100.00	99.90	100.00	100.00	99.92
Syllable	Top 1	82.24	82.80	37.11	96.15	92.14	42.49	95.36	93.27	46.18
	Top 5	89.77	90.12	38.86	99.59	96.81	42.59	98.81	96.43	46.75
	Top 10	93.81	93.91	47.03	99.87	97.78	84.09	99.08	96.59	87.15
	Top 20	97.44	97.14	55.58	99.92	98.28	86.56	99.20	96.71	88.36
Word	Top 1	24.83	28.02	20.29	78.80	85.12	78.42	79.03	83.54	77.85
	Top 5	52.25	58.69	44.34	97.74	98.48	84.11	93.71	92.88	81.78
	Top 10	70.02	76.32	58.33	99.39	99.60	86.30	94.81	94.04	83.14
	Top 20	85.84	87.95	72.17	99.93	99.99	88.34	95.36	96.19	84.50
Command	Top 1	57.72	57.78	51.88	93.38	94.16	93.37	93.41	93.94	91.30
	Top 5	99.19	98.85	87.86	100.00	100.00	97.50	99.81	99.54	96.14
	Top 10	100.00	99.90	95.49	100.00	100.00	98.23	100.00	99.96	96.87
	Top 20	100.00	100.00	98.17	100.00	100.00	98.71	100.00	100.00	97.60

top 1 accuracy achieved by repeating the last word to 28.02 % for the QoS radar. The MC provides 24.83 % accuracy here. Nevertheless, these accuracies are much lower than for the prediction of syllables and could possibly be improved based on the prediction of syllables. Again, no bar is visible in Figure 5.2 for the random guessing strategy for the QoS and the Rules-v2 radar since the number of emitted words is very large. However, despite the large number of different words used in general, also here the 20 most frequent ones constitute the majority of the test data (74.02 % for the QoS and 96.23 % for the Rules-v2 radar).

For the commands and functions, the results of the best simple strategies, the LSTMs, and the MCs are very similar. Mainly, the radars search for new targets, as shown by the top 1 accuracy of prediction the most frequent function of 85.92 % to 94.85 % (see Table 5.5).

The results suggest that the performance gain when using an LSTM or an MC is largest if there are many different symbols in use and the order in which they are emitted is rather complex. For simple resource management algorithms or few symbols, there are simpler strategies which can achieve high prediction accuracy and the effort for training an LSTM or MC might not be worth it. However, for complex resource management with many symbols, the gain can be very large. In general, it is

Table 5.5: Prediction accuracies [%] of the MCs, the LSTMs, and the strategy that predicts the most frequent symbols. The best results are marked in bold.

		QoS			Rules-v1			Rules-v2		
		MC	LSTM	Freq.	MC	LSTM	Freq.	MC	LSTM	Freq.
Command	Top 1	57.72	57.78	57.91	93.38	94.16	62.51	93.41	93.94	63.59
	Top 5	99.19	98.85	98.56	100.00	100.00	100.00	99.81	99.54	99.70
	Top 10	100.00	99.90	100.00	100.00	100.00	100.00	100.00	99.96	100.00
	Top 20	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Function	Top 1	85.94	86.20	85.92	92.37	92.89	92.81	94.58	94.48	94.85
	Top 5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	Top 10	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
	Top 20	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

observed that the LSTM outperforms the MC if the symbols are hard to predict, like it is the case for words. If the prediction is simpler, the MC has an advantage over the LSTM. By definition, the syllables that form a word often occur together in a specific order. Therefore, it is not surprising that sequences of syllables exhibit a pattern that is easier to predict than that of words.

5.3.2 Evaluation with Missing and Additional Symbols

In reality, the received signal is probably not ideal. Symbols might be missing or additional symbols from the same or another radar might be inserted. Hence, the performance of the LSTMs and the MCs is assessed when faced with corrupted data after training with ideal data. For guessing and predicting the most frequent symbols, missing and additional symbols do not change the predictions. For repeating the last symbol, these cases impact the prediction a lot. Therefore, the evaluation is restricted to a comparison of the LSTMs, the MCs, and the repeating strategy.

Figure 5.3 visualises what happens when symbols are missing or additional symbols are inserted. As it is not expected that any strategy can predict missing symbols, one prediction error would be the best case. For an additional symbol, at least two errors are expected. Two scenarios are considered in the evaluation. In the first one,

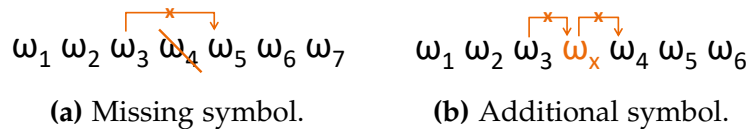


Figure 5.3: With a missing or additional symbol, at least one or two prediction errors, respectively, are expected to be made.

Table 5.6: Mean \pm standard deviation of the relative top 1 prediction accuracy $\overline{\text{acc}}_{\text{rel}}(\mathcal{E})$ [%] with missing or additional syllables.

	Rate	Syllable		
		MC	LSTM	Repeating
Missing	1 %	-0.59 ± 0.04	-1.71 ± 0.07	$-\mathbf{0.10 \pm 0.08}$
	5 %	-2.94 ± 0.22	-7.86 ± 0.34	$-\mathbf{0.56 \pm 0.43}$
	10 %	-5.90 ± 0.45	-14.22 ± 0.36	$-\mathbf{1.19 \pm 0.89}$
	20 %	-11.93 ± 0.99	-24.60 ± 0.31	$-\mathbf{2.68 \pm 2.01}$
Missing in blocks	1 %	-0.14 ± 0.02	-0.39 ± 0.06	$-\mathbf{0.08 \pm 0.05}$
	5 %	-0.72 ± 0.11	-1.82 ± 0.09	$-\mathbf{0.38 \pm 0.26}$
	10 %	-1.42 ± 0.22	-3.59 ± 0.23	$-\mathbf{0.76 \pm 0.46}$
	20 %	-2.86 ± 0.44	-7.00 ± 0.48	$-\mathbf{1.55 \pm 0.96}$
Additional	1 %	$-\mathbf{1.95 \pm 0.02}$	-2.77 ± 0.17	-1.96 ± 0.03
	5 %	$-\mathbf{9.42 \pm 0.10}$	-12.69 ± 0.64	-9.47 ± 0.10
	10 %	$-\mathbf{17.95 \pm 0.22}$	-23.31 ± 1.25	-18.03 ± 0.21
	20 %	$-\mathbf{32.92 \pm 0.39}$	-40.15 ± 2.69	-33.10 ± 0.36
Additional in blocks	1 %	$-\mathbf{1.18 \pm 0.01}$	-1.44 ± 0.06	$-\mathbf{1.18 \pm 0.01}$
	5 %	-5.69 ± 0.04	-6.53 ± 0.32	$-\mathbf{5.65 \pm 0.05}$
	10 %	-10.86 ± 0.04	-12.34 ± 0.58	$-\mathbf{10.84 \pm 0.11}$
	20 %	-19.91 ± 0.07	-22.90 ± 0.95	$-\mathbf{19.85 \pm 0.21}$

single symbols are randomly removed or inserted. In the second scenario, missing or additional blocks of five symbols are considered. The data is randomly created from the test set with 1 %, 5 %, 10 %, and 20 % missing or additional symbols. The additional symbols are randomly chosen from the symbols of all emitters. Here, only the results for syllables and words are presented as these are the most interesting modelling levels considering the results obtained with ideal data.

Tables 5.6 and 5.7 show the mean relative top 1 accuracies $\overline{\text{acc}}_{\text{rel}}(\mathcal{E})$ and the standard deviations for syllables and words of the MCs, the LSTMs, and the repeating strategy with respect to the results for ideal data and averaged over all emitter types $e \in \mathcal{E}$,

$$\overline{\text{acc}}_{\text{rel}}(\mathcal{E}) = \frac{\overline{\text{acc}}_{\text{corrupt}}(\mathcal{E}) - \overline{\text{acc}}_{\text{ideal}}(\mathcal{E})}{\overline{\text{acc}}_{\text{ideal}}(\mathcal{E})} \cdot 100 \%, \quad (5.4)$$

with

$$\overline{\text{acc}}(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{acc}_e(\hat{Y}, Y, k), \quad k = 1. \quad (5.5)$$

In addition, Figure 5.4 depicts a comparison of the absolute top 1 accuracies of the three methods for the syllables and words of the QoS radar. Note that the results

Table 5.7: Mean \pm standard deviation of the relative top 1 prediction accuracy $\overline{\text{acc}}_{\text{rel}}(\mathcal{E})$ [%] with missing or additional words.

	Rate	Word		
		MC	LSTM	Repeating
Missing	1 %	-0.14 ± 0.03	0.76 ± 1.55	-0.15 ± 0.01
	5 %	-0.78 ± 0.09	-0.70 ± 1.50	-0.83 ± 0.03
	10 %	-1.60 ± 0.26	-2.41 ± 2.01	-1.72 ± 0.11
	20 %	-3.57 ± 0.54	-5.76 ± 4.00	-3.85 ± 0.18
Missing in blocks	1 %	-0.08 ± 0.01	-0.01 ± 0.51	-0.08 ± 0.01
	5 %	-0.38 ± 0.16	-0.04 ± 1.58	-0.39 ± 0.15
	10 %	-0.80 ± 0.26	-1.29 ± 2.53	-0.86 ± 0.19
	20 %	-1.60 ± 0.55	-3.86 ± 2.63	-1.66 ± 0.51
Additional	1 %	-1.87 ± 0.07	-0.69 ± 1.25	-1.94 ± 0.04
	5 %	-9.14 ± 0.27	-6.45 ± 2.08	-9.35 ± 0.26
	10 %	-17.47 ± 0.56	-11.64 ± 2.50	-17.85 ± 0.49
	20 %	-32.02 ± 1.05	-21.72 ± 2.85	-32.71 ± 0.84
Additional in blocks	1 %	-1.17 ± 0.01	-0.65 ± 1.47	-1.18 ± 0.03
	5 %	-5.67 ± 0.10	-6.63 ± 2.82	-5.59 ± 0.12
	10 %	-10.82 ± 0.10	-11.51 ± 2.89	-10.62 ± 0.29
	20 %	-19.72 ± 0.18	-20.37 ± 3.22	-19.57 ± 0.54

cannot be directly related to those presented in Tables 5.6 and 5.7 since the tables display the mean over all emitters.

Figure 5.4 and Tables 5.6 and 5.7 show that blocks of missing or additional syllables in general cause less harm than single ones for all strategies. Interestingly, the LSTMs' accuracies for additional syllables in blocks are better than for missing single syllables. Missing words do not have much impact on the performance and, as expected, additional words are worse than missing ones. However, the difference between the single and blockwise case is not as clear for the LSTMs with words as it is for syllables. The MCs are more robust than the LSTMs with respect to missing and additional syllables (see Tables 5.6 and 5.7). For single additional words, the LSTMs' accuracies exhibit less decrease. However, the differences are small if the additional words occur in blocks of five. The results of the LSTMs vary more per emitter, especially for words, as seen by the standard deviation depicted in Table 5.7, which is higher than that of the MCs in most cases. The repeating strategy is most robust with corrupted data regarding syllables. For single additional syllables, the MCs are slightly better, but not notably. However, the absolute accuracies of the MCs and LSTMs are still higher than those of the repeating strategy. For words, the LSTMs are the most robust models in most cases.

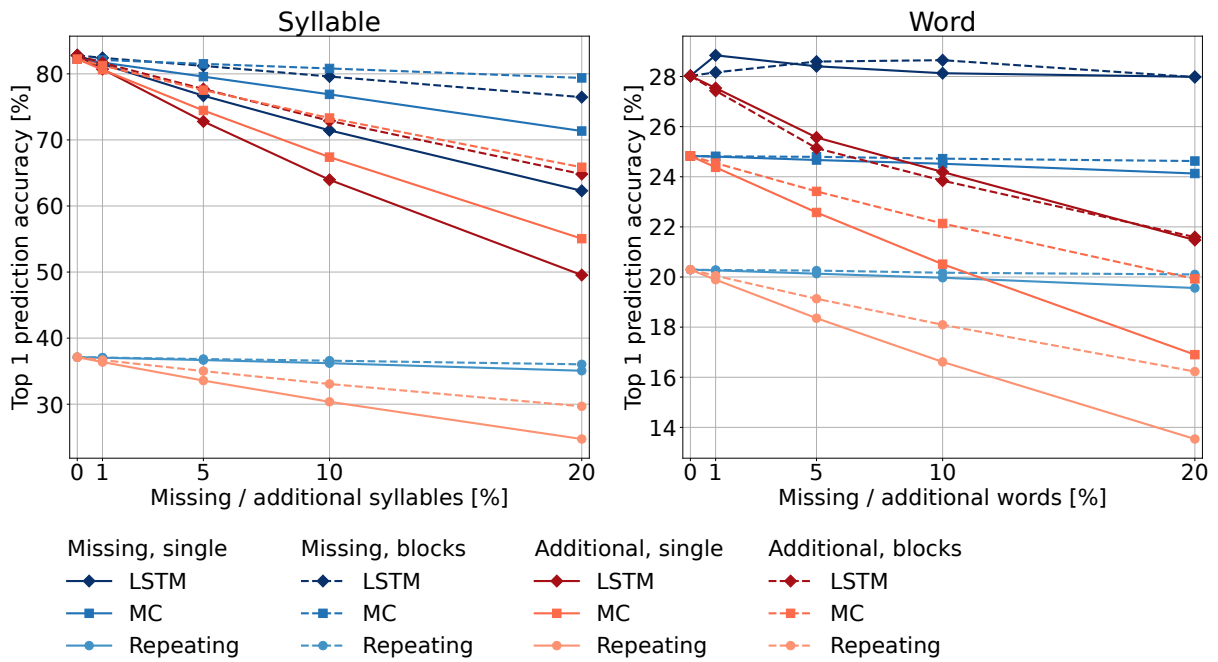


Figure 5.4: Comparison of the impact of missing and additional syllables and words on the top 1 accuracy of the LSTMs, the MCs, and the repeating strategy for the QoS radar.

5.3.3 Evaluation of the Impact of Input Encoding

Figure 5.5 shows a comparison of the course of training and validation loss while training the LSTM with word embedding and one-hot encoding for the syllables of the QoS radar. Several training runs were performed and they all showed similar behaviour. It is observed that the results with word embedding are slightly better. An interesting aspect is the instability of training with one-hot encoding. This could probably be resolved by adapting the training parameters, but here the differences in training under the same conditions are to be demonstrated. Moreover, training is faster with word embedding (per epoch about 130 min vs. 165 min) due to the smaller input sizes. The one-hot encoding results in a 25 380 dimensional vector, while the embedded representation has 64 dimensions. Of course, generating the word embedding also takes some time, but to find the best model, several architectures and parameters need to be tested. Therefore, the training is repeated several times and faster training of the networks is important. Moreover, for a stable training, neural networks prefer vectors with only small differences between the values of the entries. Using a word embedding provides an advantage here since it resembles a normalisation of the input data.

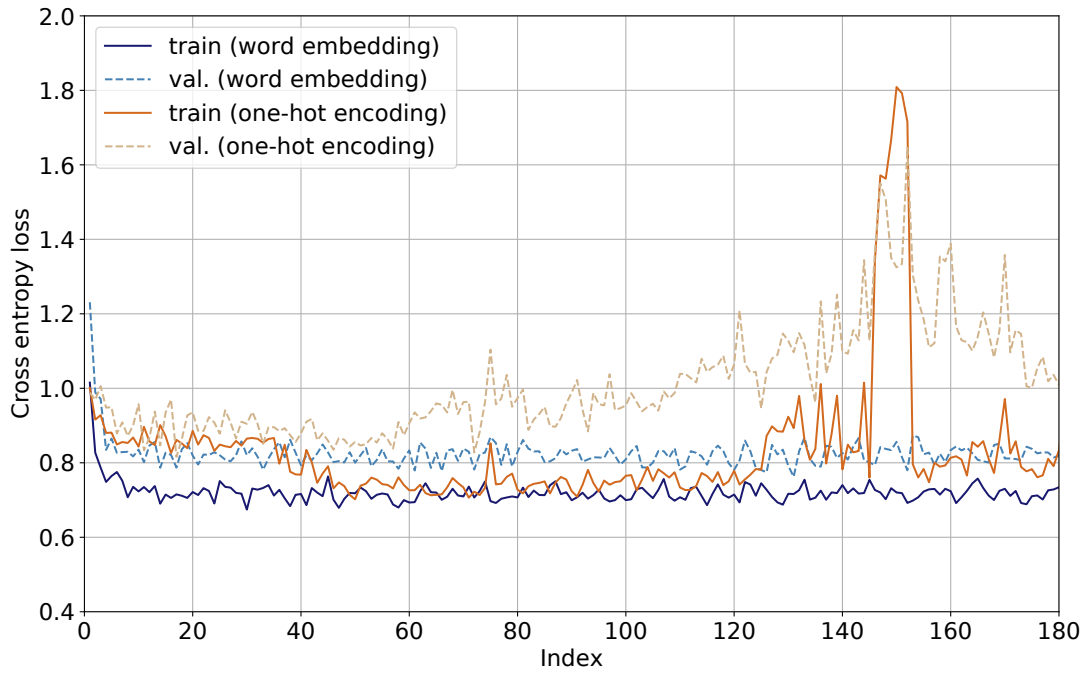


Figure 5.5: Course of training and validation loss for **syllables of the QoS radar** with word embedding and one-hot encoding [60]. The index on the x-axis represents 8534 batches with 120 syllables each. (E.g. index 20 corresponds to $20 \cdot 8534 \cdot 120 = 20\,481\,600$ processed syllables.)

5.4 Summary

This chapter presents methods for predicting a radar’s next emission. To this end, a simulated airborne multifunction radar with three different resource management methods is employed. The basis for the prediction is the adapted hierarchical emission model of understanding the radar emissions as a language, which is described in Chapter 4.

One of the presented methods, an MC, exhibits the memoryless property. In contrast, the other one, an LSTM, is specially designed to remember the past input sequence. The results show that it depends on the complexity of the task whether an approach with or without memory is to be preferred. Both the MCs and the LSTMs are well suited for emission prediction, even with corrupted data. Only for the most complex task of predicting radar words, the LSTMs provide notably better results.

Both approaches are compared to simple prediction strategies, namely random guessing, repeating the last symbols, and predicting the most frequent ones. The LSTMs and MCs achieve very good performance for complex behaviour with many possible emissions. For more regular behaviour with less symbols, the simpler methods achieve almost as good or better results. Nevertheless, for syllables of the example radars, the improvement in top 1 prediction accuracy of the LSTMs and MCs can be more than 120 % compared to the best simple strategy.

Sequences in the order of 10 000 repetitions of the same letter are too long for an LSTM to learn. Also the MC learns to predict the last letter and therefore provides the same accuracy as the repeating strategy. Hence, the GRU based method presented in [36] is not expected to work well either for the data given in this thesis. Here, the hierarchical emission model provides an advantage. As the syllables can be predicted very well, the next letter can be determined by mapping the predicted syllable to its letters. A disadvantage is that many pulses need to be received before a syllable can be detected with high certainty. A possible solution is to use an approach that maintains several hypotheses about the underlying syllables while receiving the letters.

If there are missing and additional symbols in the data, the prediction accuracy decreases, while missing and additional symbols in blocks are less harmful than single ones. Moreover, the results show that the repeating strategy is very robust with respect to corrupted data for syllables, but still provides worse results than the LSTMs and the MCs. In this case, the MCs are also more robust than the LSTMs. For words, however, the LSTMs exhibit less accuracy decrease in most cases.

In addition, this chapter shows that the application of a word embedding instead of one-hot encoding for representing the symbols of the radar language has advantages regarding the training speed and stability, as well as the performance achieved. This is due to the much smaller input size and a normalisation performed by the embedding.

Chapter 6

Identification of the Radar Emitter Type

This chapter presents methods to identify the radar emitter type based on its emissions. First, it gives an introduction and a literature overview. Afterwards, it describes and evaluates the proposed methods, both under ideal conditions and with corrupted data. Identifying the radar type corresponds to the first problem of EW (see Section 2.4), the classification. Parts of the chapter have been published in [61,62].

6.1 Introduction

Identification of the radar emitter type from a stream of pulses is an important aspect in the field of EW as it provides information about the threat an emitter poses to a platform. As described in Section 2.2, emitter identification is traditionally performed by comparing the parameters stored in a database to the ones measured from the received signal. However, modern multifunction radars are agile such that simple pattern matching methods do not provide satisfactory identification accuracy any more. Therefore, new methods for the identification of agile radars are presented and compared in this chapter.

Data from different radars is collected by listening to a specific RF for a certain amount of time, which is called the (receiver) dwell time. Usually, several emitters are active simultaneously and hence, pulses from different radars are received at the same time. The first processing step therefore consists of deinterleaving the pulses into sequences that should contain PDWs from one emitter only (see Section 2.2.3). After this step, separated PDW sequences are obtained, but no information about the emitters' identities is given. Consequently, it is not possible without major effort to combine several short sequences received in different dwells into a longer sequence of PDWs of the same emitter. Longer sequences, however, can improve the identification accuracy. Effectively, data from different emitters is alternating in the input to the identification method. Figure 6.1 provides a visualisation. The input to the identi-

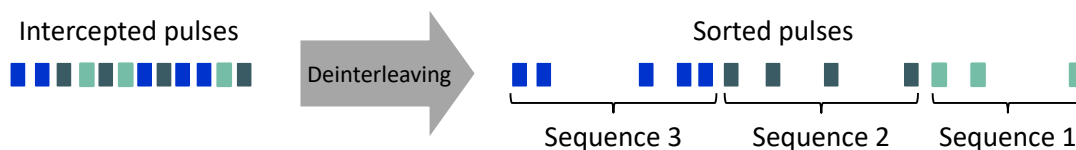


Figure 6.1: The input to the identification method is a stream of sequences that consist of deinterleaved pulses from potentially different emitters [61].

fication method or classifier is shown on the right. It contains three separated PDW sequences that the classifier processes sequentially. However, it is not known whether the sequences belong to one, two, or three emitters and usually, the classifier's input consists of deinterleaved PDW sequences from different emitters.

6.1.1 Related Work

With the rise of machine learning and especially neural networks, several methods for emitter type identification have been proposed. First attempts date back to 1990 [113], but since then radars have become more complex and neural networks more powerful. More recent approaches include the use of deep learning techniques [114–116]. However, the methods presented in these papers need a fixed-length representation of their input and are therefore not efficiently applicable for processing streaming data.

Emitter identification with RNNs, either an LSTM or a GRU, is suggested in [36, 38–41]. RNNs have the advantage that they are able to process sequences of varying length by feeding their own output back as an input (see Section 3.5.5).

In [38], a sequence of n PDWs with several features (PRI, RF, PW, BW, Amp, RF shift) is projected to a higher dimension and averaged by a trainable layer before being processed by a GRU. The evaluation is performed with 14 emitters in four classes. Emitters in the same class are hard to distinguish because of ambiguities in the waveform parameters, while emitters in different classes are easily separable. Some of the emitters exhibit long-term agilities, which should be captured by the GRU.

The authors of [39] use the features PRI, RF, and PW. Two types of input normalisation are performed and the results concatenated, which are then used as input to parallel layers of LSTM cells that process different features separately. The first normalisation approach is called min-max normalisation. It uses the global minimum and maximum of the training data distribution to convert the features into the range $[-1, 1]$. The second approach is called per-sequence normalisation, which only uses the minimum and maximum of the current sequence. The method is evaluated with 17 emitter classes, however, no details on the waveform parameters are provided.

In [36], the same method that is used for prediction (see Section 5.1.1) is also used for identification. The considered features of the pulses are PRI and PW. These are discretised with reference to a global maximum, embedded into a vector, and input to a GRU network. Based on this previous work, the author of [40] uses an iterative clustering approach to sort the PDWs by function (called mode in the paper). Here, the PRI is used as the only parameter. The idea is to give more weight in the identification process to pulse groups that provide more discriminative features, which is measured by the expected loss of the classifier. The author states that the algorithm aggregates information from the stream of pulses to obtain more data for identification. However, no explanation is provided on how it is ensured that the aggregated information actually represents a single emitter.

Attention [117] is introduced in [41] to reduce the effect of missing and additional pulses on the identification accuracy. Missing pulses cause harmonics of the PRI in the stream, e.g. a single missing pulse induces a PRI that is twice the original value.

With additional pulses, the measured PRI value is smaller than the true one. The attention mechanism in the proposed method learns to ignore those corrupted PRI values. The PDW stream consists of PRI and PW, which are input to separate GRUs for identification. The processing path for the PRI contains the attention block, which is inserted before the GRU. The evaluation is performed with nine emitters that exhibit waveform parameters with constant statistical features.

An HMM (see Section 3.4.2) is used for radar type identification in [42]. The presented approach is based on the hierarchical emission model introduced by Visnevski et al. (see Section 4.1.1) and the focus is set on radar words. The hidden states of the HMM correspond to the true words, while the observations represent the detected ones. To improve the robustness, the emission probabilities are set such that the observed word is only identical to the true word with a probability of 65 %. For every emitter type, an HMM is learnt. Then, identification corresponds to determining which model most likely generated the received data. The approach is evaluated with two emitters that use six different words, which are shared by both radars.

6.1.2 Contributions

This chapter develops two methods for identification, which are applicable to the adapted hierarchical emission model as described in Chapter 4. As also done for prediction (see Chapter 5), MCs as introduced in Section 3.4.1 without memory and LSTMs that are specially designed to remember are compared to see whether memory is important for the identification of the emitter type.

The emission parameters of the example emitters used throughout this thesis (see Section 4.3) highly overlap. It is evaluated whether the LSTMs and the MCs are able to distinguish between the emitter types and therefore identify the resource management method. Moreover, it is demonstrated which level of the emission model is best suited for identification. Both methods are compared to two simple strategies and evaluated under several conditions.

As described above, the input to the classifier consists of a stream of alternating sequences from different emitters (see Figure 6.1). None of the previous works thoroughly investigates how the length of a consecutive sequence of data from the same emitter influences the identification accuracy. This chapter shows how the identification accuracy depends on this length, which is called the sequence length. The impact of the sequence length on training the LSTMs and testing all approaches is evaluated.

The method presented in [42] also employs the hierarchical emission model. However, only the word level is considered, while all modelling levels are evaluated in this chapter. In addition, the example emitters used in this thesis are much more complex than those presented in [42], which only make use of six words as opposed to the maximum of 34 440 words considered here.

Section 6.2 provides details about the data and the implementation of the methods, Section 6.3 presents the evaluation, and Section 6.4 gives the summary.

Table 6.1: "Overlap matrix" for syllables [%]. Rv1 = Rules-v1, Rv2 = Rules-v2.

		Target		
		QoS	Rv1	Rv2
Source	QoS	100.00	0.41	96.50
	Rv1	100.00	100.00	100.00
	Rv2	88.14	0.37	100.00

Table 6.2: "Overlap matrix" for words [%]. Rv1 = Rules-v1, Rv2 = Rules-v2.

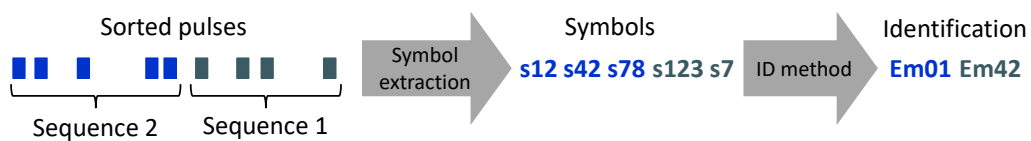
		Target		
		QoS	Rv1	Rv2
Source	QoS	100.00	0.06	80.07
	Rv1	80.95	100.00	100.00
	Rv2	61.97	0.06	100.00

6.2 Approaches

With the data obtained from the simulations of the example radars, LSTMs and MCs are trained to identify the underlying emitter type. The same training, validation, and test sets are used as for learning the word embedding in Chapter 4 and predicting the emissions in Chapter 5. An overview is given in Tables 5.1 and 5.2 of Chapter 5.

To provide an estimate on the similarity of the example emitters and hence the difficulty of separating them, Tables 6.1 and 6.2 show the "overlap matrices" for syllables and words. Here, the rows indicate the "source" and the columns the "target", e.g. the row "Rv2" and the column "QoS" of Table 6.1 say that 88.14 % of the syllables used by the Rules-v2 radar are also emitted by the QoS radar. A large portion of the syllables and words are used by several emitters with the Rules-v1 radar being an exception. For letters, commands, and functions, the overlap is even bigger. The QoS and the Rules-v2 radar employ all of the 18 letters and 10 commands, resulting in 100 % overlap for these two emitters. The Rules-v1 radar uses less letters and commands, but all of them are emitted as well by the other two radars. The functions exhibit a 100 % overlap for all three emitter types.

Figure 6.2 shows a schematic overview of the processing chain after deinterleaving. The sorted PDWs are mapped to the defined symbols by the symbol extraction step. Sequences of symbols are then processed by the identification method and an emitter ID is output per sequence. In reality, the length of the sequences received from the same emitter, i.e. the number of symbols with the same colour in the figure, varies. Therefore, the impact of the sequence length on training and testing is evaluated in the following.

**Figure 6.2:** Symbols are extracted from the deinterleaved PDW sequences, which are then input to the identification method.

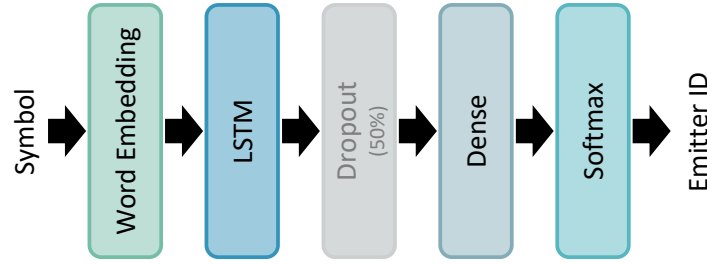


Figure 6.3: General architecture of the networks for identification. The dropout layer is shown in grey since it is only active during training.

6.2.1 Long Short-Term Memory

Figure 6.3 shows the general architecture of the networks, which is the same as for prediction, except for the last dense layer and the labels. As for prediction, the input consists of sequences of strings containing the name of the symbol. These strings are mapped to dense vectors of floating point values by the word embedding layer, which are then processed by the LSTM layer with peephole connections (see (3.86) to (3.88)). After the LSTM layer, a dropout of 50 % is applied (see Section 3.5.3). It is followed by a feedforward layer in which every neuron is connected to every output of the dropout layer, which is called a dense layer. It employs linear activation and maps the output of the LSTM layer to activations for each emitter type. The number of neurons in the dense layer corresponds to the number of emitters, i.e. three. The softmax layer normalises the output of the dense layer to yield a probability distribution over the emitter's class indices (see Section 3.5.1). For training, the input consists of pairs $(\omega, \text{idx}(e))$ where $\omega \in \Omega^l$ is a symbol at modelling level l and $\text{idx}(e)$ the class index of the emitter $e \in \mathcal{E}$ with $\text{idx}(e) \in \{0, 1, 2\}$. The label $\text{idx}(e)$ is represented as a one-hot vector such that it can be interpreted as a probability distribution and allows for training with the cross-entropy loss as defined in Section 3.5.2.

Three network types, that are trained with different sequence lengths, are developed. To simulate the alternating sequences from different emitters (see Figure 6.2), the input is changed to symbols from another emitter after the specified sequence

Table 6.3: LSTM parameters for the different modelling levels and network types.

Type	Parameter	Letter	Syllable	Word	Command	Function
LSTM ₁₀	# layers	1	1	1	1	1
	# cells/layer	16	4	4	4	4
LSTM _{rand}	# layers	1	1	1	1	1
	# cells/layer	16	4	4	4	4
LSTM _{scen}	# layers	1	1	1	1	1
	# cells/layer	4	4	8	16	16

Algorithm 1 Train or test with a random sequence length.

Input: Set of emitters \mathcal{E} , minimum len_min and maximum len_max sequence length

Global variable: seq_len chosen for all chunks

```

1: global initialisation of  $e \in \mathcal{E}$  for all chunks
2:  $seq\_len \leftarrow random \in [len\_min, len\_max]$ 
3:  $s \leftarrow 0$ 
4: while training or testing not finished do
5:    $data \leftarrow$  next symbol from emitter  $e$ 
6:   train or test network with  $data$ 
7:    $s \leftarrow s + 1$ 
8:   if  $s = seq\_len$  then
9:      $e \leftarrow random \in \mathcal{E} \setminus e$ 
10:     $seq\_len \leftarrow random \in [len\_min, len\_max]$ 
11:     $s \leftarrow 0$ 
12:   end if
13: end while

```

length is reached. Per network type, the number of LSTM layers and cells are optimised independently by trying several configurations and keeping the one with the lowest loss on the validation set. The networks are implemented using Python and TensorFlow. One network type is trained with a sequence length of ten symbols, which is called LSTM₁₀. Another one is trained with random sequence lengths in the interval $[1, 1400]$, it is called LSTM_{rand}. The third network type is trained with the complete scenarios and is called LSTM_{scen}. The complete scenarios consist of about 5 to 15 million letters each, those are mapped to about 7000 to 30 000 syllables and about 1400 to 6000 words, commands, and functions. Per symbol level, i.e. letter, syllable, word, command, and function, an LSTM is trained. In total, there are 15 independent networks, one per symbol level and sequence length used during training. The parameters of these networks are shown in Table 6.3.

The networks are trained with the Adam optimiser, a learning rate of 0.0002, and the cross-entropy loss. During training, batches containing the symbols of 120 simulation runs are fed in parallel to the networks. The amount of data needed to be reduced for letters because otherwise the training would have taken too long. Therefore, the batch consists of 12 simulation runs. The state of the LSTM layer is kept between batches. In each batch, several emitters need to be represented because otherwise, the network parameters are adapted to always identify the emitter of the current batch and training never converges. Therefore, each batch \mathcal{B} is split into chunks of 40 symbols from each emitter (4 for letters), i.e. for step t

$$\mathcal{B}_t = (\underbrace{\omega_t^{r_{1,1}}, \dots, \omega_t^{r_{40,1}}}_{e_1}, \underbrace{\omega_t^{r_{1,2}}, \dots, \omega_t^{r_{40,2}}}_{e_2}, \underbrace{\omega_t^{r_{1,3}}, \dots, \omega_t^{r_{40,3}}}_{e_3}), \quad (6.1)$$

and for step $t + 1$

$$\mathcal{B}_{t+1} = (\underbrace{\omega_{t+1}^{r_{1,1}}, \dots, \omega_{t+1}^{r_{40,1}}}_{e_1}, \underbrace{\omega_{t+1}^{r_{1,2}}, \dots, \omega_{t+1}^{r_{40,2}}}_{e_2}, \underbrace{\omega_{t+1}^{r_{1,3}}, \dots, \omega_{t+1}^{r_{40,3}}}_{e_3}). \quad (6.2)$$

Here, $r_{i,j}$ is the simulation run r_i of emitter e_j . Like in the training for prediction, the runs actually appear in a shuffled order and are only shown sorted here for an easier visualisation. At initialisation, the batch is evenly divided to contain 40 runs for each of the emitters. However, after the specified sequence length is reached, the next emitter for each chunk is randomly chosen. Hence, there is no guarantee that every emitter is represented in each batch, but as there are several million batches generated during training, the data from every emitter is equally often used on average. Algorithm 1 depicts the pseudocode for this procedure employed for training or testing with a random sequence length. If $len_min = len_max = 10$, the algorithm corresponds to training with a sequence length of ten symbols. The same procedure is employed independently for validation.

6.2.2 Markov Chain

In the MC approach, the probability for each emitter type is estimated based on the probability that the emitter generated the input sequence. Let $e \in \mathcal{E}$ be an emitter and $\omega \in \Omega^l$ a symbol as defined above. For a sequence of symbols $\bar{\omega} = \omega_1 \dots \omega_{|\bar{\omega}|}$, the probability of emitter e_i is determined by Bayes' rule,

$$\hat{P}(e_i|\bar{\omega}) = \frac{\hat{P}(\bar{\omega}|e_i) \cdot \hat{P}(e_i)}{\hat{P}(\bar{\omega})}. \quad (6.3)$$

The probability of the data $\bar{\omega}$ given the emitter e_i is calculated by

$$\hat{P}(\bar{\omega}|e_i) = \hat{P}_{e_i}(\omega_1) \cdot \prod_{t=1}^{|\bar{\omega}|-1} \hat{P}_{e_i}(\omega_{t+1}|\omega_t), \quad (6.4)$$

where $\hat{P}_{e_i}(\omega_1)$ is estimated from counting the occurrences $\text{count}(\omega_j)$ of each symbol in the data of emitter e_i ,

$$\hat{P}_{e_i}(\omega_j) = \frac{\text{count}(\omega_j)}{\sum_{\omega \in \Omega^l} \text{count}(\omega)}, \quad (6.5)$$

and $\hat{P}_{e_i}(\omega_{t+1}|\omega_t)$ is obtained from (5.3) in Chapter 5. It is assumed that the prior probability of each emitter is the same, i.e. $\hat{P}(e_i) = \frac{1}{|\mathcal{E}|}$ for all emitters. The probability of the sequence $\hat{P}(\bar{\omega})$ is determined by marginalisation,

$$\hat{P}(\bar{\omega}) = \sum_{e \in \mathcal{E}} \hat{P}(\bar{\omega}|e). \quad (6.6)$$

6.2.3 Comparison Methods

Two simple methods are employed for comparison. The first one checks for each emitter $e \in \mathcal{E}$ if the current input is in its individual set of symbols Ω_e^l , i.e. it performs a dictionary lookup to estimate the probability

$$\hat{P}_d(e|\omega_j) = \begin{cases} 1 & \text{if } \omega_j \in \Omega_e^l, \\ 0 & \text{otherwise.} \end{cases} \quad (6.7)$$

If the current symbol can be found in the dictionaries of more than one or none of the emitters, the probability is equally distributed. Which emitter is the top 1 prediction is randomly chosen. This method resembles matching the waveform parameters to a database without considering the temporal structure of the emissions. For a sequence of symbols $\bar{\omega}$, the probability for emitter e is determined as

$$\hat{P}_d(e|\bar{\omega}) = \kappa \prod_{t=1}^{|\bar{\omega}|} \hat{P}_d(e|\omega_t), \quad (6.8)$$

where κ is a normalisation factor. The second simple method is random guessing with a probability of $\frac{1}{|\mathcal{E}|} \approx 33.33\%$ for each emitter.

6.3 Experimental Results

Since the identification accuracy is influenced by the number of consecutive symbols received from the same emitter, both methods are tested with different sequence lengths in the set $\mathcal{S} = \{1, 10, 50, 100, 200, 400, 600, 800, 1000, 1200, 1400\}$. The shortest scenarios contain 1400 symbols, therefore this is chosen to be the maximum. In the adapted hierarchical emission model, 1400 words correspond to about 1.5 min to 3 min, 200 words last about 35 s to 45 s. On average, a word consists of five to six syllables, therefore 1400 syllables correspond to about 25 s to 50 s, 200 to about 6 s to 9 s. These values are only intended for a rough reference since they strongly depend on the scenarios and the emitters. A short reaction time is not important for an ELINT application and therefore, the decision can usually be made based on longer sequences of symbols.

The results are obtained at the end of a sequence of length $s \in \mathcal{S}$. As the data is slightly imbalanced (see Table 5.2), the mean of the identification accuracies per emitter are reported,

$$\text{acc}(s) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \text{acc}(e, s), \quad (6.9)$$

with

$$\text{acc}(e, s) = \text{acc}(\hat{Y}_s, Y_e, k), \quad k = 1. \quad (6.10)$$

Here, \hat{Y}_s corresponds to the output of the identification method after having processed s symbols and Y_e to the set of labels, which only contains the $\text{idx}(e)$ for the

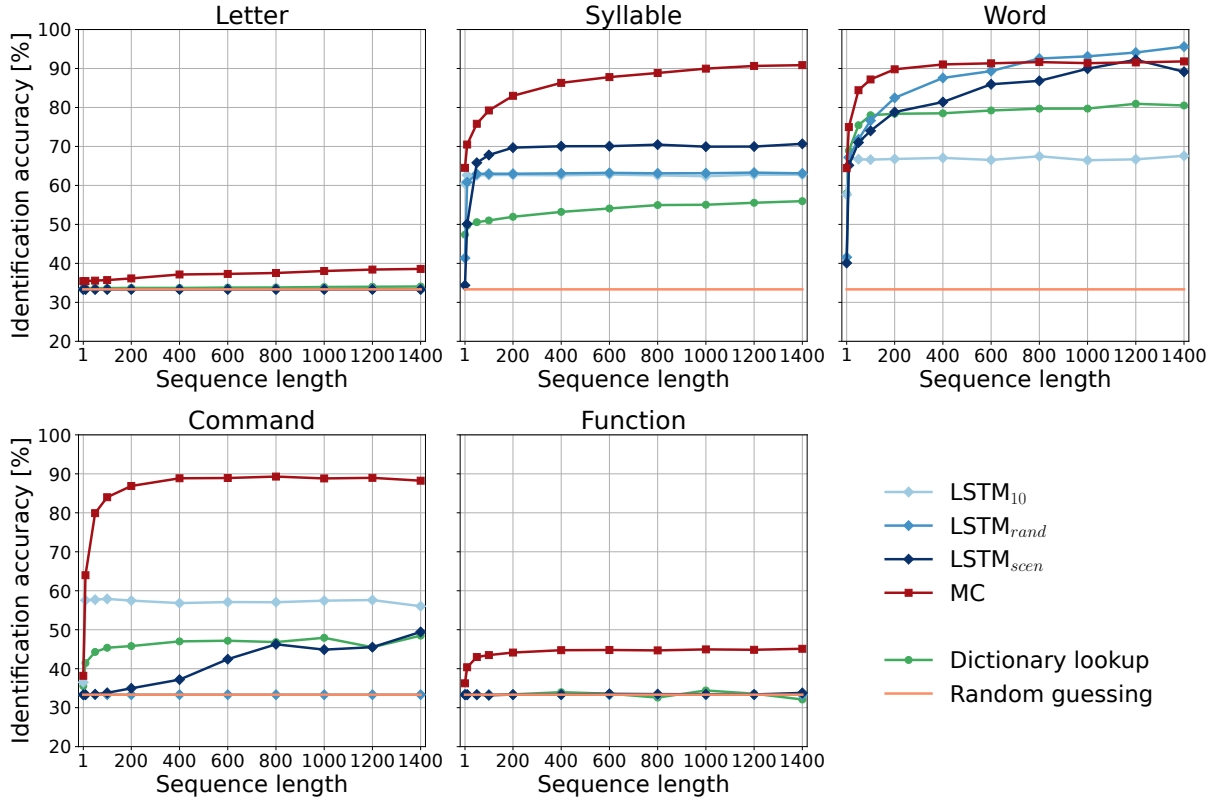


Figure 6.4: Identification accuracies of the different methods on the test data of all emitters.

data of emitter e . The methods are evaluated under ideal conditions and with corrupted data, in which symbols are missing or additional symbols that potentially belong to a different emitter are inserted.

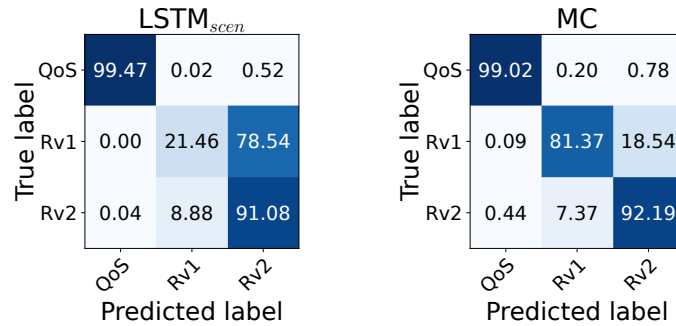
6.3.1 Evaluation Under Ideal Conditions

The identification accuracies are displayed in Figure 6.4, while the exact results of the MCs and the LSTMs for syllables, words, and commands are given in Table 6.4. As can be observed, the LSTMs are not able to identify the emitter types based on their letters and the results are equal to random guessing. Since the simulated radar is airborne, it integrates many pulses and therefore sequences of the same letter are very long (e.g. word $w_{26101507}$ consists of 16 384 repetitions of the same letter). If words are repeated, the sequences become even longer and are too long for an LSTM to learn. The MC does not perform much better either. Since the most common letter l_{18} is shared by all three radars, the results of the dictionary lookup equal random guessing as well.

For syllables, the dictionary lookup achieves much better results than guessing. For a single syllable, however, the accuracy is only 47.37%. The LSTM₁₀ provides a much higher identification accuracy than the other LSTMs when tested with only one syllable (60.5% vs. 34.37% for the LSTM_{scen} and 41.36% for the LSTM_{rand}). However, with increasing sequence length the other two LSTMs come close or are even better.

Table 6.4: Identification accuracies [%] of the MCs and LSTMs at different sequence lengths for selected modelling levels.

		Sequence length										
		1	10	50	100	200	400	600	800	1000	1200	1400
Syllable	MC	64.52	70.49	75.82	79.23	83.00	86.32	87.79	88.85	89.97	90.65	90.86
	LSTM ₁₀	60.50	62.63	62.60	62.67	62.67	62.61	62.78	62.58	62.37	62.75	62.76
	LSTM _{rand}	41.36	60.83	62.96	63.00	63.00	63.10	63.22	63.11	63.13	63.29	63.10
	LSTM _{scen}	34.37	50.10	65.82	67.81	69.70	70.06	70.07	70.46	69.95	69.97	70.67
Word	MC	64.42	74.97	84.42	87.19	89.79	91.04	91.33	91.65	91.37	91.56	91.82
	LSTM ₁₀	57.64	67.10	66.72	66.62	66.78	67.06	66.53	67.48	66.46	66.70	67.59
	LSTM _{rand}	41.58	67.23	71.83	76.63	82.47	87.58	89.33	92.56	93.12	94.11	95.62
	LSTM _{scen}	40.09	65.16	71.01	74.00	78.83	81.40	85.96	86.83	89.94	92.24	89.17
Command	MC	38.18	63.98	79.91	83.99	86.89	88.86	88.92	89.30	88.82	88.97	88.24
	LSTM ₁₀	36.52	57.57	57.72	57.90	57.47	56.83	57.10	57.06	57.46	57.62	56.03
	LSTM _{rand}	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.33	33.33
	LSTM _{scen}	33.31	33.32	33.45	33.82	34.97	37.20	42.43	46.26	44.89	45.53	49.43

**Figure 6.5:** Confusion matrices of the LSTM_{scen} and the MC at a sequence length of 1400 syllables. Rv1 = Rules-v1, Rv2 = Rules-v2.

The confusion matrices of the networks show that the Rules-v1 radar is not recognised with high accuracy, regardless of the sequence length (see Figure 6.5 for the confusion matrix with a sequence length of 1400 syllables). All syllables of the Rules-v1 radar are also contained in the dictionaries of the other radars (see Table 6.1). However, the MC is able to distinguish between the three emitters. Even for a single syllable, the MC outperforms the LSTM₁₀ with an accuracy of 64.52 %. Figure 6.5 shows the confusion matrices of the LSTM_{scen} and the MC at a sequence length of 1400 syllables, while the confusion matrices for all sequence lengths can be found in Figures A.1 and A.2 of the appendix.

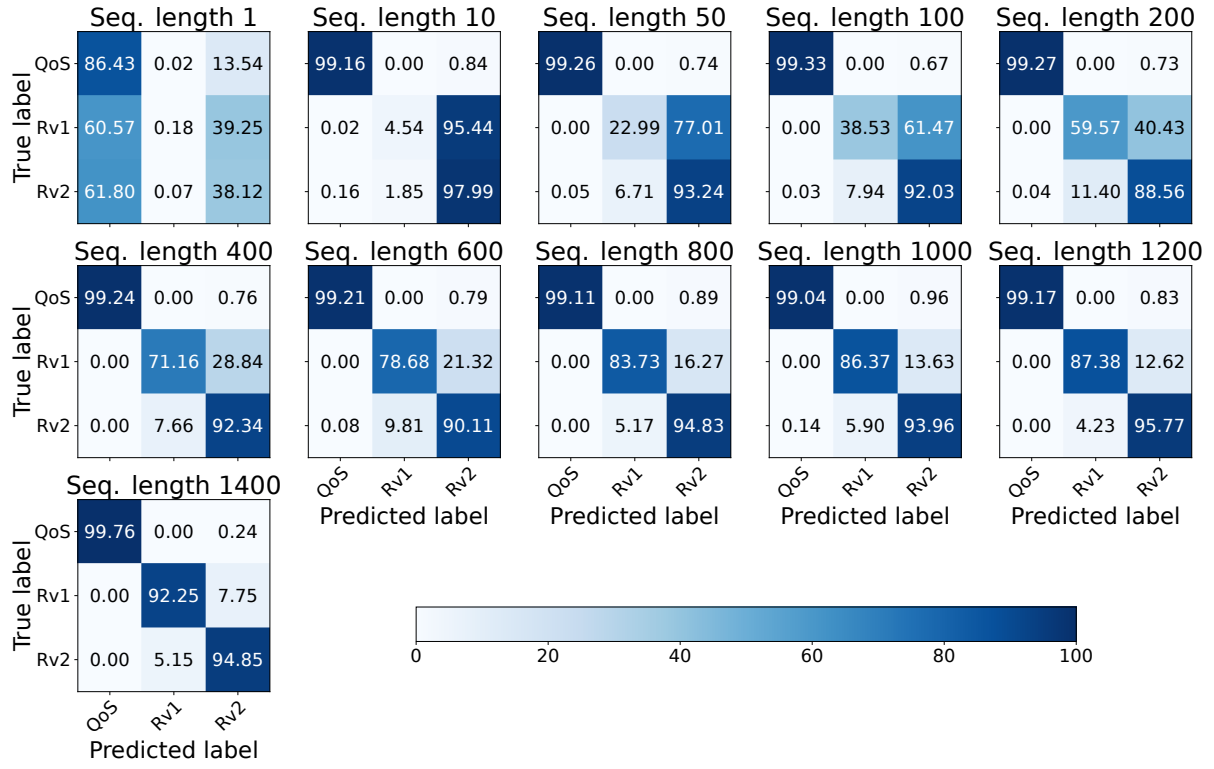


Figure 6.6: Confusion matrices at different sequence lengths of the LSTM_{rand} for words [61]. Rv1 = Rules-v1, Rv2 = Rules-v2.

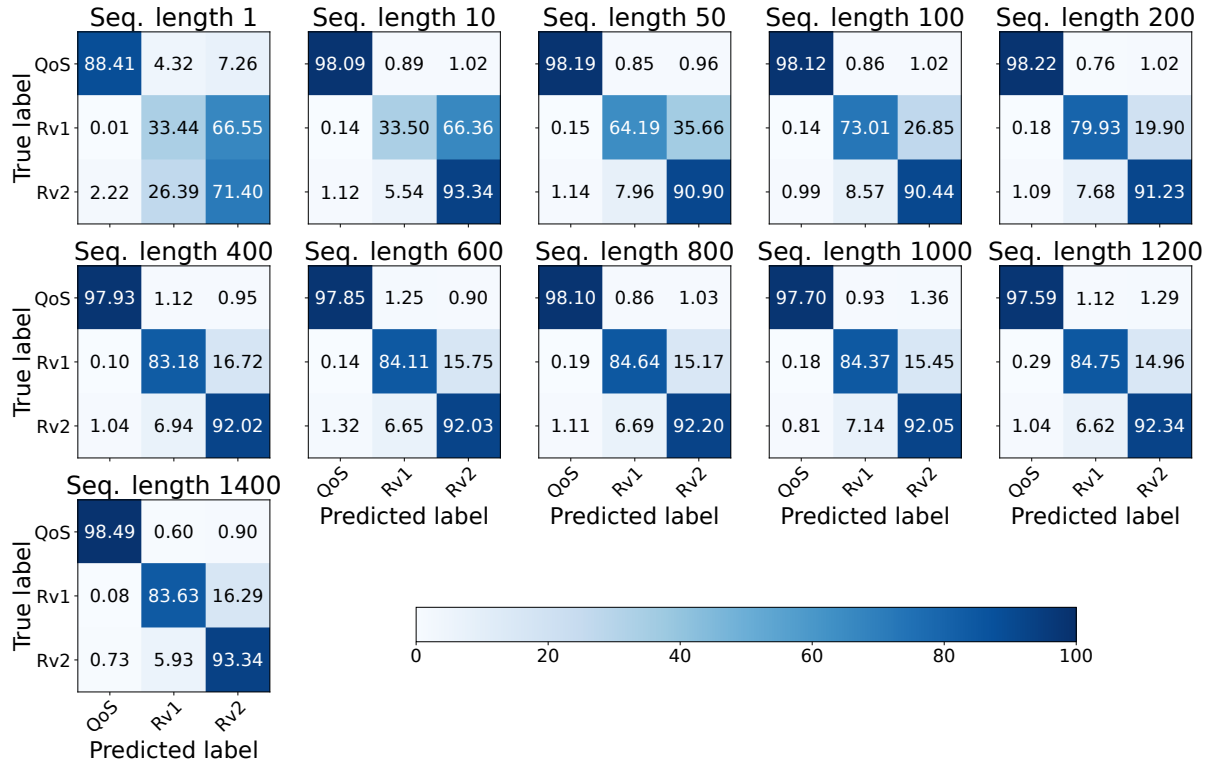


Figure 6.7: Confusion matrices at different sequence lengths of the MC for words. Rv1 = Rules-v1, Rv2 = Rules-v2.

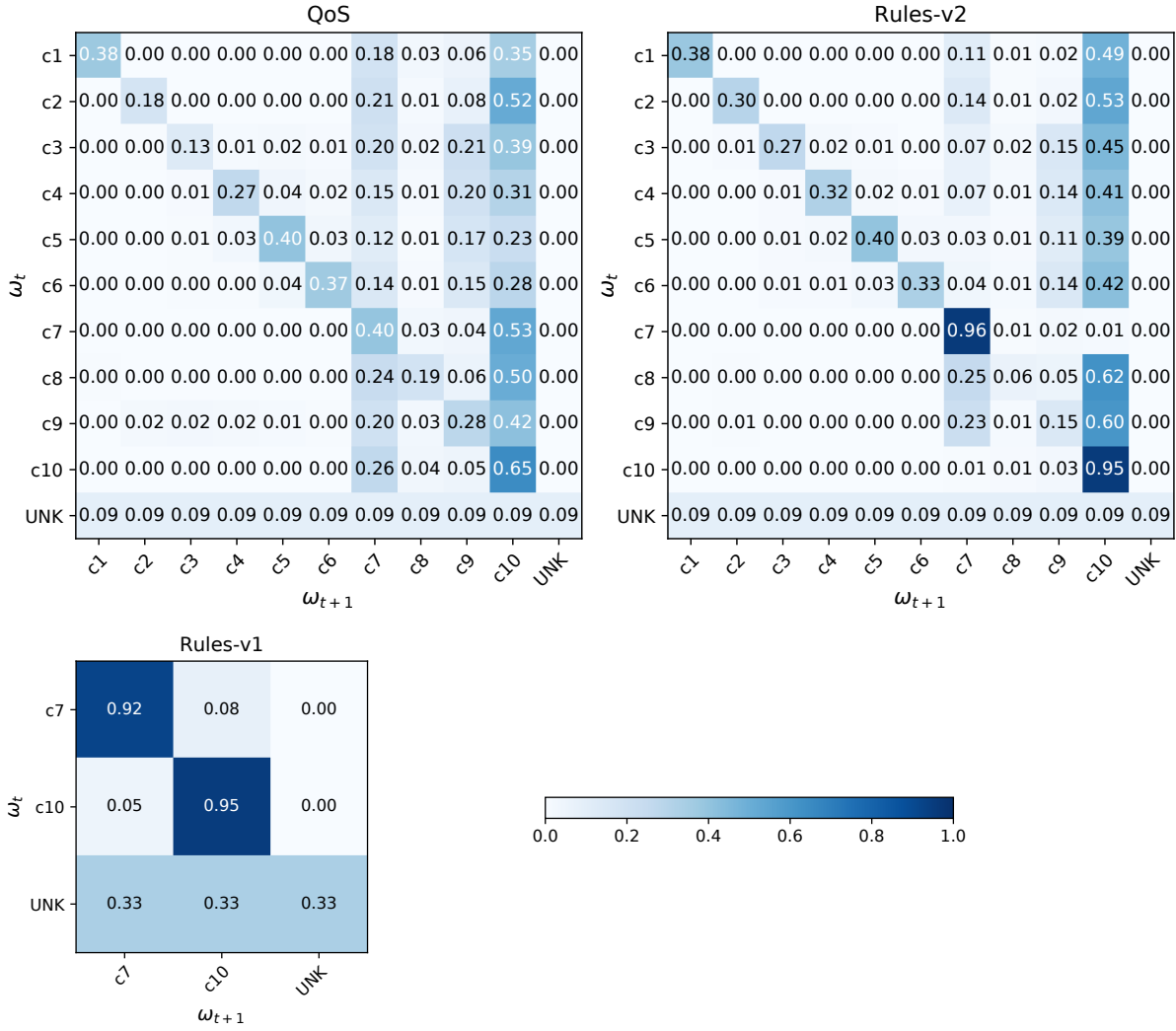


Figure 6.8: Transition matrices for the commands of the different emitter types.

Also for words, the MC's identification accuracy is higher than that of the LSTMs if the sequence length is short. With increasing length, the accuracies converge and eventually, the $LSTM_{rand}$ provides better results. For only one word, the $LSTM_{10}$ provides a higher accuracy than the other LSTMs. However, the accuracy does not improve with increasing sequence length since the network has not learnt to make use of longer sequences. And still, it is outperformed by the MC. Figure 6.6 shows the confusion matrices of the $LSTM_{rand}$ for words at different sequence lengths. The QoS radar is identified with a very high accuracy also with short sequences. As the two rule-based approaches are more similar in their behaviour regarding words, longer sequences are needed to discriminate between them. In Figure 6.7, the confusion matrices of the MC are shown. The higher accuracy in comparison to the $LSTM_{rand}$ is achieved because it is able to distinguish between the two rule-based emitters already with shorter sequence lengths. However, with a sequence length of 600 symbols and more, the results do not increase any more. The dictionary lookup

performs comparably to the LSTMs and the MC at short sequence length of up to 100 symbols. With longer sequences, however, the other methods provide better results.

For syllables and commands, the MCs achieves much higher accuracies than the LSTMs. The reason why the MC outperforms the LSTMs by such a large amount is not directly clear. In general, the LSTMs were hard to train and no improvement of the accuracy could be seen for several choices of the number of layers and number of cells per layer. The LSTM training procedure is very susceptible to the adjustable variables and is possibly too complex for this kind of data. For the commands, for example, one can see clear differences in the transition matrices of the three radar versions (see Figure 6.8). The commands that appear most often (c_7, c_{10}) are also those with the largest differences in the transition probabilities for the QoS and the two rule-based radars, making the emitters distinguishable for an MC. Since the Rules-v1 radar uses much less commands than the Rules-v2 radar, they can be separated as well with longer sequence lengths.

For functions, the identification accuracy is basically identical to random guessing. Since the radars only use three different functions (search, confirm, track) and their behaviour is similar, they cannot be distinguished based on the functions. Only the MC achieves better results than random guessing, but the transition matrices of the radars differ only slightly, which makes them hard to separate.

6.3.2 Evaluation with Missing and Additional Symbols

To simulate the imperfections of an actual application, the methods are tested with corrupted data that contains missing or additional symbols. The additional symbols are randomly selected from the global dictionary Ω^l , i.e. also symbols from different emitters are inserted. Two cases are considered for the evaluation. In the first one, single symbols are randomly removed or inserted into the data of the test set. In the second case, symbols are removed or inserted in blocks of five. Both cases are evaluated with 1 %, 5 %, 10 %, and 20 % corrupted data. Since the LSTMs' results for letters, commands, and functions are already unsatisfactory for ideal data, only the results for syllables and words are presented.

Using the formulation developed in Section 6.2.2, the MCs are not robust with respect to corrupted data. The transition matrices for syllables and words obtained from Chapter 5 are very sparse with only a few transitions of high probability. Therefore, when calculating $\hat{P}_{e_i}(\omega_{t+2}|\omega_t)$ in (6.4) instead of $\hat{P}_{e_i}(\omega_{t+1}|\omega_t)$ because of a missing symbol, this probability has a high chance of being zero. The data shows that words are more often repeated. Therefore, missing words do not cause a difference if $\omega_{t+2} = \omega_{t+1}$. For syllables, however, repetitions are not common and $\omega_{t+2} \neq \omega_{t+1}$ most often. Also additional symbols either from the same or from a different emitter can cause $\hat{P}(\bar{\omega}|e_i)$ in (6.4) to be zero for the complete sequence, resulting in a very large accuracy decrease. Therefore, for comparison, an MC is learnt for which $\hat{P}_{e_i}(\omega_j|\omega_i)$ is set to a small value $\varepsilon = 0.001$ if it is 0. Hence, the rows of the transition matrices do not sum up to 1 any more. If they are normalised, all entries become small due to the sparsity, resulting in a low overall accuracy as shown below. The

Table 6.5: Mean \pm standard deviation of the relative top 1 identification accuracy $\overline{\text{acc}}_{rel}(\mathcal{S})$ [%] with missing or additional syllables.

		Syllable				
	Rate	MC	MC $_{\epsilon}$	LSTM $_{10}$	LSTM $_{rand}$	LSTM $_{scen}$
Missing	1 %	-7.34 ± 4.59	-0.22 ± 0.74	0.19 ± 0.18	0.18 ± 0.14	0.36 ± 0.31
	5 %	-14.90 ± 6.97	-3.55 ± 2.94	0.20 ± 0.17	0.16 ± 0.09	0.43 ± 0.40
	10 %	-19.08 ± 8.03	-6.85 ± 5.35	0.15 ± 0.30	0.29 ± 0.17	0.68 ± 0.37
	20 %	-23.91 ± 9.07	-10.05 ± 6.91	0.28 ± 0.28	0.55 ± 0.26	0.70 ± 0.54
Missing in blocks	1 %	-2.28 ± 1.98	0.08 ± 0.13	0.19 ± 0.18	0.09 ± 0.13	0.34 ± 0.42
	5 %	-6.34 ± 4.26	0.13 ± 0.17	0.00 ± 0.16	0.21 ± 0.18	0.51 ± 0.41
	10 %	-8.56 ± 5.04	0.05 ± 0.28	-0.03 ± 0.14	0.18 ± 0.19	0.36 ± 0.56
	20 %	-11.51 ± 5.96	-0.18 ± 0.72	0.21 ± 0.10	0.19 ± 0.22	0.62 ± 0.54
Additional	1 %	-10.01 ± 5.93	-0.98 ± 0.51	-0.28 ± 0.25	0.01 ± 0.09	0.43 ± 0.51
	5 %	-19.76 ± 8.27	-5.09 ± 2.97	-1.69 ± 0.26	-0.66 ± 0.33	0.22 ± 0.93
	10 %	-25.64 ± 8.99	-8.63 ± 3.86	-2.90 ± 0.54	-3.05 ± 0.93	-0.42 ± 1.16
	20 %	-33.19 ± 9.56	-13.25 ± 4.55	-5.64 ± 0.77	-5.64 ± 1.67	-4.03 ± 5.06
Additional in blocks	1 %	-4.37 ± 3.01	-0.36 ± 0.17	-0.42 ± 0.17	-0.03 ± 0.10	0.33 ± 0.36
	5 %	-12.03 ± 5.93	-2.73 ± 1.63	-1.79 ± 0.25	-0.90 ± 0.31	0.31 ± 0.99
	10 %	-17.00 ± 6.95	-5.18 ± 2.34	-3.70 ± 0.30	-2.06 ± 0.67	-0.19 ± 1.55
	20 %	-23.50 ± 7.54	-8.88 ± 2.88	-7.07 ± 0.41	-6.18 ± 1.81	-1.18 ± 2.29

probability for each emitter is still normalised after calculating it according to (6.3). This version is called MC $_{\epsilon}$.

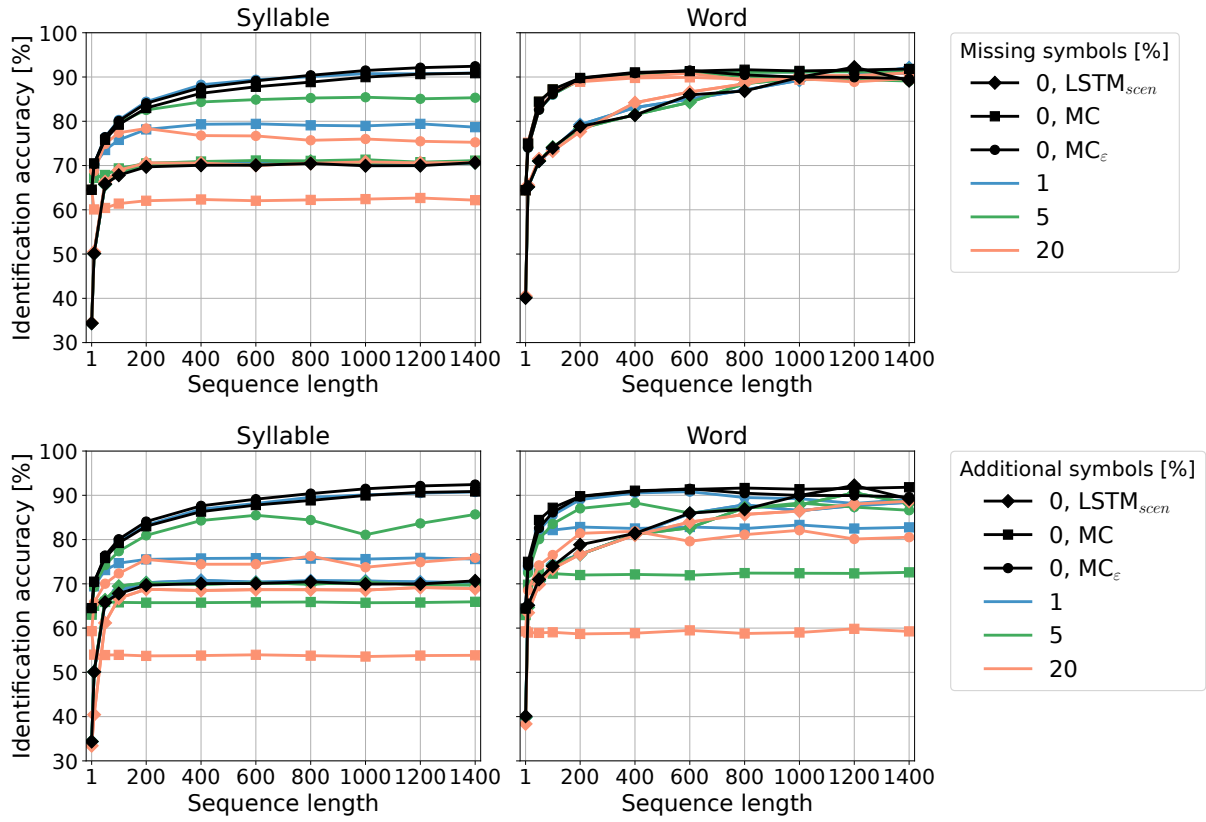
Tables 6.5 and 6.6 show the accuracy $\overline{\text{acc}}_{rel}(\mathcal{S})$ averaged over all sequence lengths in the set \mathcal{S} and relative to the results obtained with ideal data

$$\overline{\text{acc}}_{rel}(\mathcal{S}) = \frac{\overline{\text{acc}}_{corrupt}(\mathcal{S}) - \overline{\text{acc}}_{ideal}(\mathcal{S})}{\overline{\text{acc}}_{ideal}(\mathcal{S})} \cdot 100\%, \quad (6.11)$$

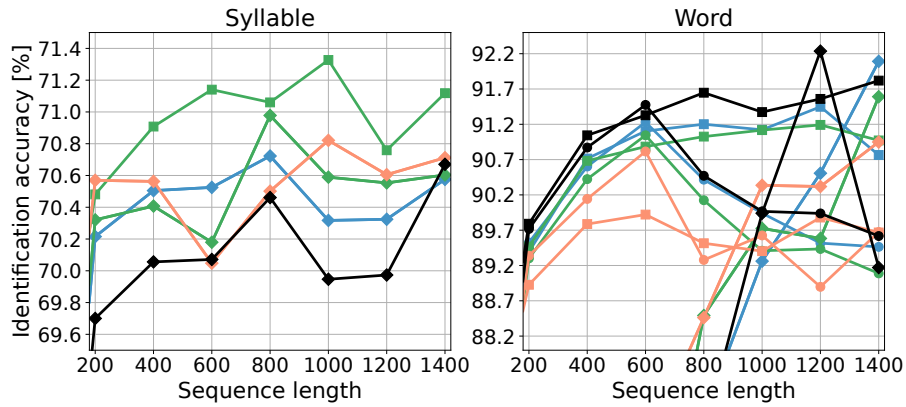
with

$$\overline{\text{acc}}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \text{acc}(s). \quad (6.12)$$

The higher the relative accuracy, the more robust the methods are with respect to corrupted data. However, this metric does not reflect the absolute accuracies with missing and additional symbols. Figure 6.9 presents a comparison of the absolute accuracies of the LSTM $_{scen}$, the MC, and the MC $_{\epsilon}$ with corrupted data, for easier readability only with single missing or additional symbols.



(a) Identification accuracies with missing (top row) or additional (bottom row) symbols [62].



(b) Zoom into the plots for missing syllables or words.

Figure 6.9: Identification accuracies of the LSTM_{scen} , the MC, and the MC_ϵ with missing or additional symbols.

Table 6.6: Mean \pm standard deviation of the relative top 1 identification accuracy $\overline{\text{acc}}_{\text{rel}}(\mathcal{S})$ [%] with missing or additional words.

	Rate	Word				
		MC	MC $_{\epsilon}$	LSTM $_{10}$	LSTM $_{\text{rand}}$	LSTM $_{\text{scen}}$
Missing	1 %	-0.32 ± 0.30	-0.16 ± 0.14	-0.24 ± 0.81	-0.13 ± 0.53	0.21 ± 1.37
	5 %	-0.32 ± 0.29	-0.32 ± 0.26	0.77 ± 1.12	-0.11 ± 1.05	0.01 ± 1.47
	10 %	-0.28 ± 0.49	-0.12 ± 0.24	0.19 ± 0.47	-0.08 ± 0.72	0.15 ± 1.15
	20 %	-1.13 ± 0.96	-0.37 ± 0.54	0.09 ± 0.63	0.46 ± 0.94	0.57 ± 1.53
Missing in blocks	1 %	-0.98 ± 0.69	-0.21 ± 0.22	0.38 ± 0.54	-0.35 ± 0.65	-0.36 ± 1.43
	5 %	-2.42 ± 1.64	-0.37 ± 0.36	0.36 ± 0.46	-0.24 ± 0.54	-0.02 ± 2.00
	10 %	-3.35 ± 2.16	-0.50 ± 0.44	0.18 ± 0.63	0.01 ± 0.75	0.24 ± 1.47
	20 %	-5.02 ± 2.97	-0.78 ± 0.55	0.23 ± 0.85	0.38 ± 0.94	-0.36 ± 2.91
Additional	1 %	-6.99 ± 3.34	-0.80 ± 0.41	-0.19 ± 0.77	-6.98 ± 5.35	-1.12 ± 1.81
	5 %	-16.92 ± 6.23	-3.12 ± 1.04	0.00 ± 1.30	-14.51 ± 9.70	-1.21 ± 1.28
	10 %	-22.82 ± 7.22	-5.52 ± 1.07	-0.84 ± 1.46	-16.61 ± 10.47	-0.44 ± 1.50
	20 %	-30.74 ± 8.24	-9.92 ± 1.47	-2.14 ± 2.55	-18.30 ± 11.03	-2.30 ± 1.41
Additional in blocks	1 %	-2.82 ± 1.36	-0.81 ± 0.38	0.51 ± 1.01	-4.30 ± 3.33	-0.03 ± 2.15
	5 %	-9.06 ± 3.39	-2.03 ± 0.78	-0.52 ± 1.22	-11.43 ± 7.70	-0.84 ± 2.51
	10 %	-14.28 ± 4.56	-3.82 ± 1.04	-1.20 ± 1.50	-14.84 ± 9.17	-1.14 ± 2.46
	20 %	-21.14 ± 5.53	-7.05 ± 1.15	-3.09 ± 2.31	-18.15 ± 10.04	-2.73 ± 1.81

All LSTMs are in general very robust with respect to missing and additional symbols, the LSTM $_{\text{rand}}$ with additional words being an exception. Both for syllables and words, the LSTM $_{\text{scen}}$ is the LSTM-based method that achieves the best trade-off with respect to accuracy and robustness. Additional symbols cause a bigger decrease in accuracy than missing ones, which have little impact. The difference between single additional symbols and additional symbols in blocks of five is small.

Also the accuracy of the MC does not change much with missing words (see Figure 6.9). With missing syllables, however, the accuracy rapidly decreases. The MC $_{\epsilon}$ achieves much better results than the MC. Although the MC $_{\epsilon}$ provides the overall highest accuracies for syllables, it is not robust with respect to corrupted data as it exhibits a large accuracy decrease. In contrast, the LSTM can take into consideration that the rest of the symbols matches a specific emitter because it has a memory.

Figure 6.10 shows a comparison of the MC and different variants, which are the MC $_{\epsilon}$ as defined above, a version MC $_{\epsilon}^{\text{norm}}$, for which the rows of the transition matrix are normalised after adding ϵ , and two versions that set the entries that are 0 to $10\epsilon = 0.01$, both in an unnormalised MC $_{10\epsilon}$ and a normalised MC $_{10\epsilon}^{\text{norm}}$ variant. For syllables, all MC variants perform comparably with ideal data, while the MC $_{\epsilon}$ slightly outperforms the other versions. It is also the most robust version, i.e. increasing the

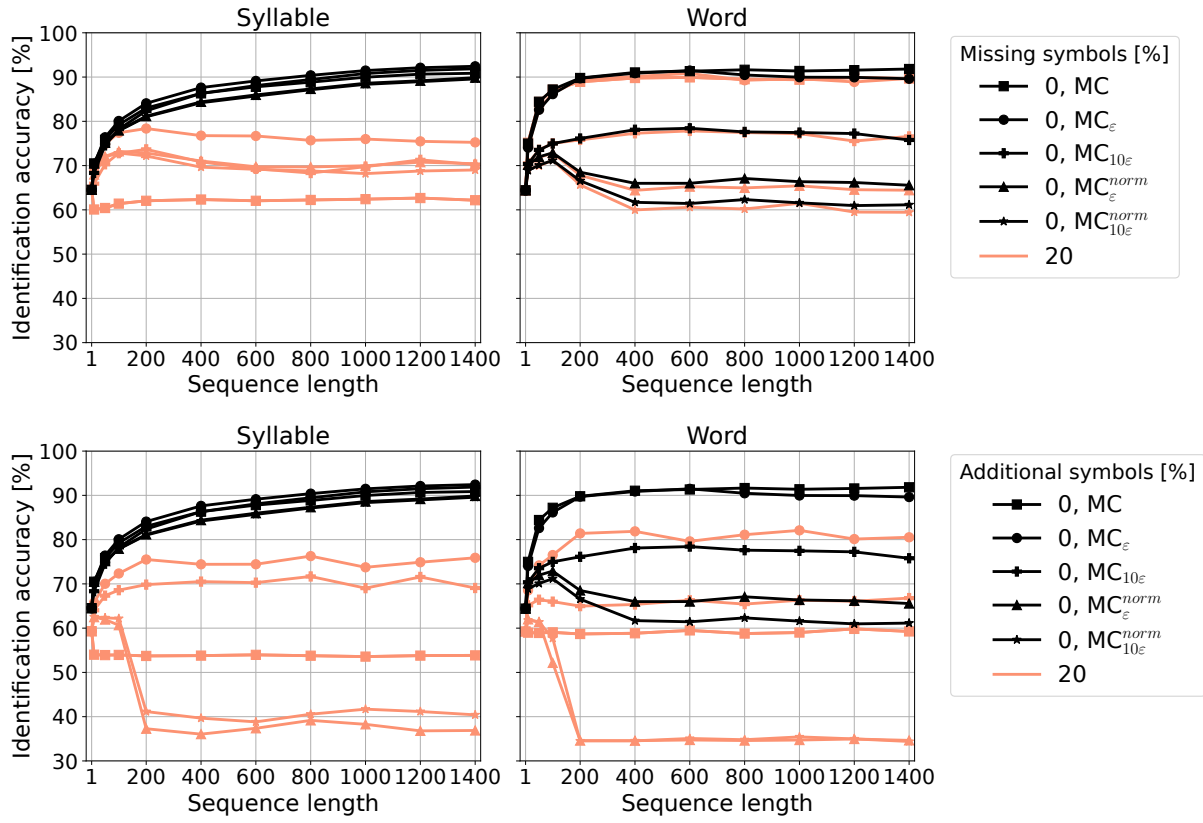


Figure 6.10: Identification accuracies of the different MC variants with missing (top row) or additional (bottom row) symbols.

new value ε for entries in the transition matrix with a probability of 0 does not increase the robustness. The same is observed for words, however, in this case the $MC_{10\varepsilon}$ variant and those using normalisation also provide much worse results for ideal data. Therefore, altering the values in the transition matrices makes the MC more robust up to some point, but at the same time the criterion for an identification of the emitter type is weakened and eventually, all emitters become equally likely. The same effect would be achieved by training the MCs with corrupted data. In this case, the percentage of missing or additional symbols influences the value of ε .

6.4 Summary

This chapter presents methods to distinguish between different radar emitter types. They are based on LSTMs and MCs, which use the adapted hierarchical emission model of a radar language as introduced in Chapter 4. The comparison between LSTMs and MCs is especially interesting as they are in contrast to each other in terms of memory. The evaluation is performed using the data of a simulated multifunction radar with three different resource management methods of varying complexity.

The symbols emitted by the different radar types highly overlap. Nevertheless, both the LSTM and the MC are able to identify the emitter type based on the frequency and agility of its emissions and are therefore capable of recognising the resource management method. Moreover, they clearly outperform simple methods like a dictionary lookup, which resembles matching the waveform parameters to a database as it is traditionally done (see Section 2.2). The highest improvement compared to the dictionary lookup is achieved for syllables, words, and commands and ranges from 19 % for words with the LSTM to 80 % for commands with the MC. Furthermore, it is seen that radar words, which correspond to radar dwells, are the modelling level best suited for identification.

The evaluations show that the identification accuracy depends on the length of consecutive emissions received from the same radar and that longer sequences are needed to discriminate between similar emitter types. The sequence length also influences the training of the LSTM and hence, three different variants are implemented.

For identification with ideal data, the MC outperforms the LSTM. Especially for syllables and commands, the identification accuracy of the MC is much higher. However, if symbols are missing or there are additional symbols in the data, the accuracy of the MC rapidly decreases while the LSTM can take advantage of its memory and is very robust, despite of being trained with ideal data only. By adapting the transition probabilities of the MCs used to model the emitters, the identification accuracy can be made more robust with respect to corrupted data. However, this needs to be done carefully and has its limitations as it influences the accuracy for ideal data. Since in an actual application corrupted data is very likely, LSTMs are to be preferred because of their inherent robustness.

In addition, the advantages of the hierarchical emission model are demonstrated. None of the methods is able to identify the emitters based on their letters. With syllables and words, however, high accuracies are achieved.

Chapter 7

Ensembles of Predictive Models

This chapter suggests and evaluates several approaches to combine multiple predictive models, like the radar models presented in Chapter 5, into an overall system. To provide more general guidelines for the selection of the architecture, the methods are compared with a public dataset first. Based on the gained insights, the evaluation is afterwards performed using the radar data presented in the previous chapters. Parts of this chapter have been accepted for publication in [63] and are under review for publication in [64].

7.1 Introduction

To employ the predictive emitter models presented in Chapter 5, they need to be combined into a multi-model system or an ensemble, for which several options are available. Research on methods for combining multiple classifiers, which are called experts in some approaches, considers two basic principles: classifier fusion and classifier selection. Classifier fusion is usually applied to leverage the effect that the errors made by the individual models tend to “average out”. Therefore, the performance is increased by considering the predictions of an ensemble of models that fulfil the same task, see e.g. [44]. In classifier selection, the model (or subset of models) that has the highest probability of making the best prediction for a given input is selected, see e.g. [46, 48, 49].

In the scenario considered in this thesis, the input to the ensemble of radar models is a stream of sequences that consist of deinterleaved pulses from potentially different emitters (see Figure 6.1). Therefore, the data from different sources is alternating in the input stream. In the context of processing streaming data, a time-dependent change of the underlying data distribution is called a concept drift in the literature [29–33], which can be observed in many real world applications. For example, the occurrence of weather events is influenced by the season, the statistical features of spam on Twitter change because spammers try to avoid detection [118], and the internet traffic caused by video-on-demand services is affected by the time of the day, the day of the week, public holidays, the weather, or a pandemic. Depending on the application, a concept corresponds, e.g. to the season of the year or, as in the application considered in this thesis, an emitter type.

In general terms, the next element in a stream of discrete symbols is to be predicted. Apart from the next emission of a radar, these symbols might as well correspond to the next word in a sentence, the next protein in a sequence, or the next action of a road user (e.g. turn left, switch lane). In the considered scenario, the streaming

data exhibits sudden and recurring concept drifts, which means that the underlying data distribution changes abruptly and repeatedly. For each distribution, which corresponds to a concept, an expert is learnt. In classifier selection approaches, the best expert, i.e. the one trained for the current concept, needs to be chosen to make the prediction when new data arrives. If the current concept can be identified based on the received data, it is clear which expert to choose. However, if the stability period of the concept is short, i.e. the number of consecutive symbols following the same distribution is small, or the concepts are very similar, the identification accuracy might be low and hence, classifier fusion might be the preferred option. Here, stability period is the general term for sequence length used in Chapter 6. This chapter demonstrates the properties of several ensemble architectures with respect to prediction accuracy in different settings.

In order to increase the informative value of the evaluation and because no data from different multifunction radars than the presented example emitter is available, the evaluation of ensemble methods with LSTM experts is performed with a public dataset that exhibits useful properties. Afterwards, the insights are used to present a case study on ensembles of predictive radar models, both with LSTM and MC experts.

7.1.1 Contributions

Based on the evaluation with the public dataset, this chapter answers the following questions:

1. Under which conditions should classifier fusion or selection be preferred?
2. How does the length of the stability period influence the accuracy?
3. Depending on the architecture, the LSTM experts need to process data produced by a different concept. How does this affect their internal state?
4. In which situations is a state reset of the experts beneficial or harmful?

By answering these questions, several contributions are made. Firstly, new implementations of six different ensemble architectures using either classifier fusion or selection are proposed. These architectures are implemented in several configurations, including different options for a state reset of the LSTM experts, a weight reset for online weighting methods, and the number of selected experts for classifier selection approaches. To the best of the author's knowledge, these are the first implementations of the presented ensemble architectures with independent LSTMs that perform different tasks. Secondly, through the evaluation of the architectures and configurations in six scenarios with different properties, this chapter provides guidelines for the selection and configuration of the proposed networks. These guidelines consider the availability of drift detection, the achieved identification accuracy, and the length of the stability period. Additionally, to confirm the guidelines obtained from the evaluation on the public dataset, a case study with ensembles of predictive radar models is presented for the best architectures and configurations.

Section 7.2 presents the architectures that are compared in this chapter. Section 7.3 gives the details on the implementation for the public dataset, as well as the evaluation. Section 7.4 contains details on the ensemble approaches for the predictive radar models and Section 7.5 provides the summary of the chapter.

7.2 Ensemble Architectures

7.2.1 Mixture of Experts

In the Mixture of Experts (MoE) approach [45] (see Figure 7.1), every expert e in the set of experts $E = \{e_1, e_2, \dots, e_{|E|}\}$ is provided the current input s and makes a prediction $\hat{y}^e = (\hat{y}_1^e, \dots, \hat{y}_C^e)$, where C is the number of output classes. An additional classifier, called the gating network, is trained to recognise which expert most probably makes the best prediction for a given input. The gating network outputs a weight g_e for the prediction of each expert and the final output \hat{y} is obtained by a weighted sum of the individual outputs,

$$\hat{y} = \sum_{e \in E} g_e \cdot \hat{y}^e. \quad (7.1)$$

7.2.2 Sparsely-Gated Mixture of Experts

In [46], the sparsely-gated MoE layer is introduced. Here, the MoE is not designed to be the complete architecture but a general purpose layer in a network with many other layers. The output of the gating network $g = (g_1, g_2, \dots, g_{|E|})$ is constrained to be a sparse vector, containing only d non-zero entries. This is ensured by keeping only the d largest values. The motivation is to reduce the computational cost of large models since only d experts need to be activated to do the computation.

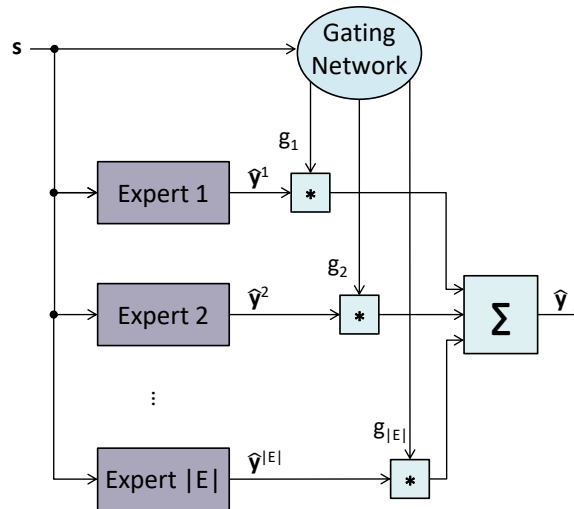


Figure 7.1: Mixture of Experts [63].

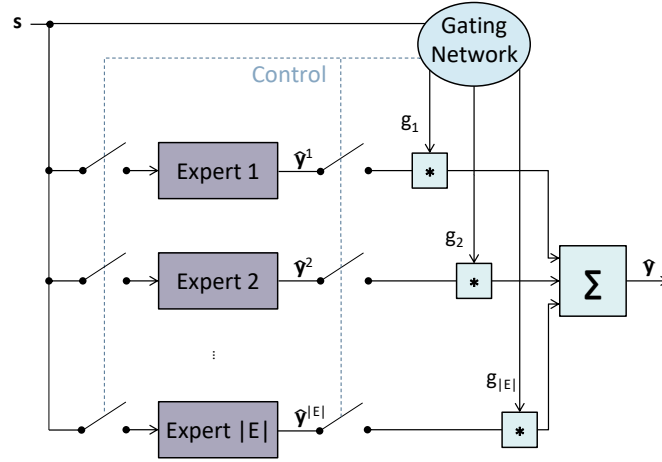


Figure 7.2: Sparsely-Gated Mixture of Experts [63].

For the presented comparison, the sparsely-gated MoE layer is adopted and employed as another multi-model architecture (see Figure 7.2). With $d = 1$, the gating network in the sparsely-gated MoE architecture only routes the input sequence s to the expert e_i with the highest weight. The overall output \hat{y} is then equal to the output of this expert

$$\hat{y} = \hat{y}^{e_i} \text{ with } e_i = \underset{e \in E}{\operatorname{argmax}} g_e, \quad (7.2)$$

where g is considered to be a function assigning weights to the experts e . For $d > 1$, a mixing between the d experts with the highest weights is performed analogously to the weighted sum in (7.1).

7.2.3 Stacking (with Input)

In the stacking architecture [47] (see Figure 7.3a), a combiner (sometimes also called meta-learner or generaliser) is employed to fuse the outputs of the individual experts for the input sequence s . The parameters of the combiner are adapted based on the experts' output. The final output \hat{y} is a (usually learnt) function of the output of each expert \hat{y}^e ,

$$\hat{y} = f_c(\hat{y}^{e_1}, \hat{y}^{e_2}, \dots, \hat{y}^{e_{|E|}}). \quad (7.3)$$

In a variant of the stacking architecture (see Figure 7.3b), the input s is also used to adapt the parameters of the combiner. The final output \hat{y} is a function of the experts' output \hat{y}^e as well as the input s ,

$$\hat{y} = f_s(\hat{y}^{e_1}, \hat{y}^{e_2}, \dots, \hat{y}^{e_{|E|}}, s). \quad (7.4)$$

To avoid overfitting of the combiner on the output of the experts for the training data, it should not be trained on the same data as the experts. There are two ways to achieve this goal. The first is to split the training data into K parts and, analogously to K -fold cross-validation, train the experts on $K - 1$ of them. This is done for every

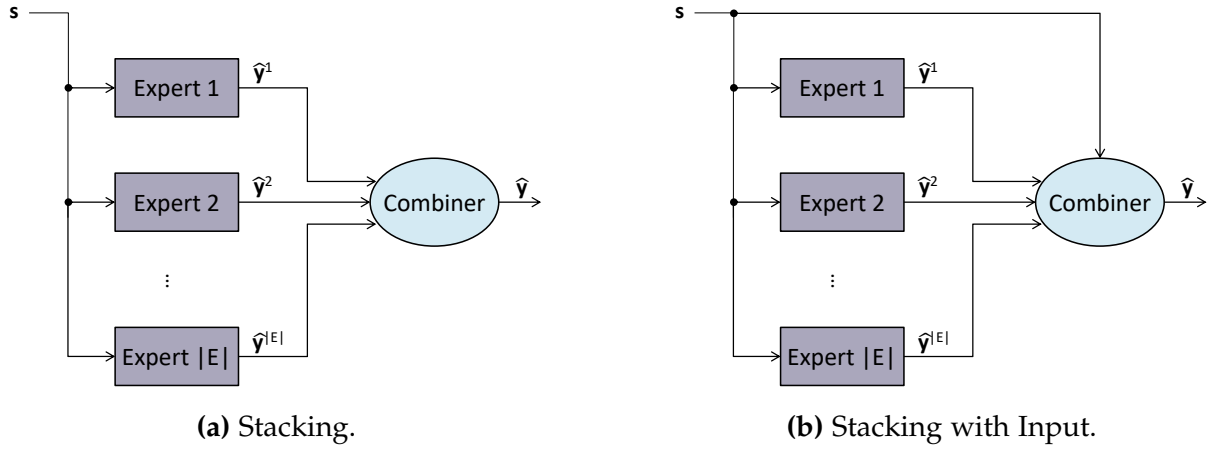


Figure 7.3: The two variants of the stacking architecture [63].

part of the training data, resulting in K experts per concept in the case considered here. All experts then make predictions on the part of the training data that they were not trained on. These predictions are used to train the combiner. Afterwards, each expert is trained with the complete training data to obtain the final experts. In this procedure, $K + 1$ training runs per expert are needed.

The second much less computationally demanding option is to split the training data into two parts and then train the experts on one part and the combiner on the other. The disadvantage of this method is that less training data is available. If the combiner is trained on a hold-out part of the training data and not using the K -fold procedure described above, the method is sometimes referred to as blending.

7.2.4 Online Accuracy-Based Weighting

Online accuracy-based weighting is commonly applied to account for concept drift when processing streaming data, e.g. [29–32]. As the name suggests, the weights for the experts are determined online and based on the expected accuracy (see Figure 7.4). The final result is a weighted sum of the individual outputs, analogous to (7.1).

7.2.5 Model Averaging

In the model averaging architecture (see Figure 7.5), the output of the experts \hat{y}^e for the input sequence s is averaged to yield the final result \hat{y} ,

$$\hat{y} = \frac{1}{|E|} \sum_{e \in E} \hat{y}^e. \quad (7.5)$$

Since all experts are always assigned the same weight, this approach does not adapt to the expected accuracy or the current input.

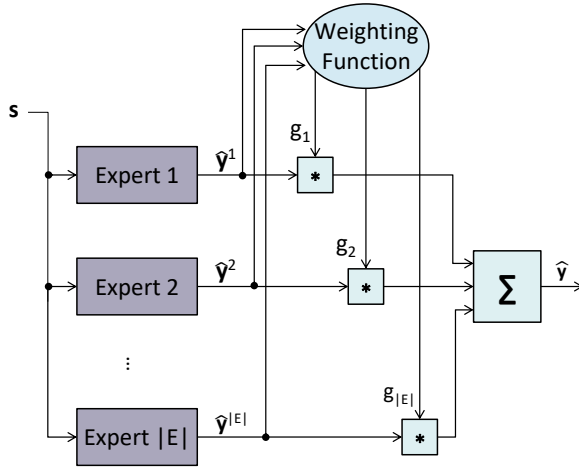


Figure 7.4: Online Accuracy-Based Weighting [63].

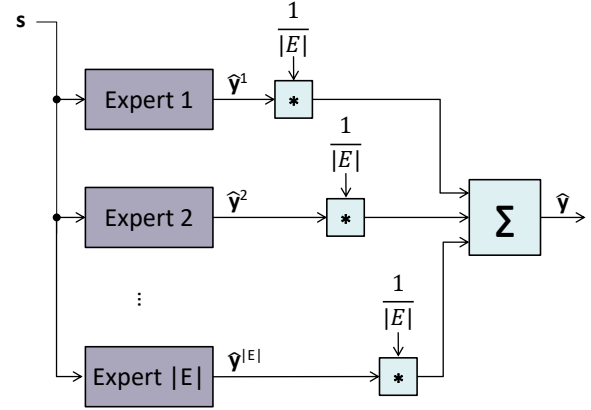


Figure 7.5: Model Averaging [63].

7.3 Ensembles of Long Short-Term Memory Experts

This section presents the implementation and evaluation of the different ensemble architectures with LSTM experts on a public dataset. The data and the scenarios that are analysed are described in Section 7.3.1. In Section 7.3.2, details on the implementation of the experts, the ensemble architectures and configurations are provided and the experimental evaluation is presented in Section 7.3.3.

7.3.1 Data

The different architectures are evaluated using the Sequence Prediction Challenge (SPiCe) data¹ [50]. It consists of 15 different sequential problems from several domains, which include NLP, biology, and software engineering. Each problem contains sequences of discrete symbols, which are numbered starting from 0. The alphabets of the problems differ in the number of symbols within a range of 10 to 6722. The special symbol -1 marks the end of a sequence and is not part of the alphabet since it is artificially inserted and not generated by the underlying process. For each problem, a training, a validation, and a test set are defined. Table 7.1 provides a rough overview of the problems, for more details see [50].

The SPiCe data is adapted to suit the needs for simulating streaming data. It was originally divided into several sequences per problem. However, here it is interpreted as one long stream and a -1 is inserted between the last symbol of a sequence and the first symbol of a new sequence. Moreover, there are several problems on synthetic data, e.g. problems 1 to 3 are generated by non-stationary HMMs. For this data, the test set of SPiCe contains the probability distribution for the next symbol instead of a concrete sample. To perform the evaluation in terms of top k accuracy (see

¹ <https://spice.lis-lab.fr/>

Table 7.1: Overview of the problems defined in SPiCe.

Problem	Alphabet size	Domain
1	20	synthetic (HMM)
2	10	synthetic (HMM)
3	10	synthetic (HMM)
4	33	NLP
5	49	NLP
6	60	partly synthetic, software engineering
7	20	biology
8	48	NLP
9	11	partly synthetic, software engineering
10	20	biology
11	6722	NLP
12	21	synthetic
13	702	NLP
14	27	partly synthetic, NLP
15	32	partly synthetic, NLP

Section 3.5.4), a concrete next symbol in the sequence is obtained by sampling from the given distribution. This is done once and the result is saved as the test set.

To evaluate different aspects of the architectures, subsets of the problems are combined into scenarios. The definitions of the scenarios are given in Table 7.2. They are chosen such that the selection results in different properties of the combined problems. Between the scenarios, the difficulty of identifying the current problem or concept is varied as well as whether the problems represent the same domain. Moreover, for the *bad* scenario, the problems were chosen such that the scenario contains combinations of experts that are especially bad at predicting the symbols of other problems in the scenario.

Table 7.2: Definition of the scenarios as subsets of the problems.

Scenario	Problems	Id. difficulty	Domain
HMM	1, 2, 3	medium	same
NLP	4, 5, 8, 11, 13	easy	same
Hard	1, 2, 3, 7, 9, 12	hard	mixed
Easy	6, 8, 9, 11, 14, 15	easy	mixed
Bad	1, 5, 7, 9, 10, 11, 12	medium	mixed
All	1 to 15	hard	mixed

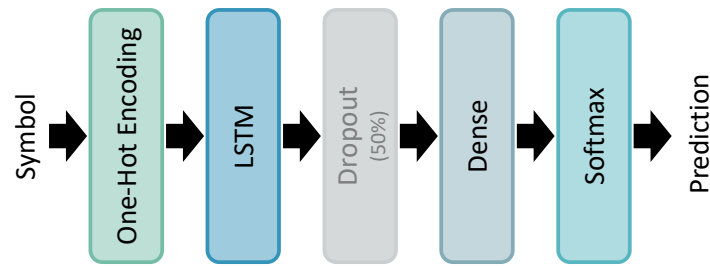


Figure 7.6: The general architecture of the experts [63]. The dropout layer is only active during training and therefore shown in grey. The architecture corresponds to the one used for the radar models in Chapter 5, except for the one-hot encoding.

7.3.2 Implementation

In the following sections, the details on the implementation of the different architectures are given. All architectures are implemented using Python and TensorFlow. For the neural-network-based methods, several architectures are trained, which differ in the number of layers and number of cells or neurons per layer. During training, checkpoints are created when the current validation loss is smaller than all previous validation losses. The checkpoint with the best validation loss is kept and the final architecture is chosen based on the validation loss of its best checkpoint.

7.3.2.1 Long Short-Term Memory Experts

For each problem $p = 1, \dots, 15$, an LSTM expert e_p is trained to predict the next symbol in the sequence. The input to the LSTM is a stream $V = v_1 v_2 \dots$ of one-hot encoded vectors v_i with $\dim(v_i) = C_p$, where C_p is the number of symbols used in problem p , including the end-of-sequence symbol -1 . In this encoding, $-1 \equiv (1, 0, \dots, 0)$, $0 \equiv (0, 1, 0, \dots, 0)$, $1 \equiv (0, 0, 1, \dots, 0)$, and so forth. The experts are trained on 70 % of the training data per problem, the remaining 30 % are used to train the combiner in the two variants of the stacking architecture.

Table 7.3: LSTM parameters of the experts.

Problem	# LSTM layers	# Cells/layer
4	1	16
1, 2	1	32
3, 13, 15	1	128
6	1	256
11	2	32
7, 8, 9, 12	2	64
5, 10, 14	2	128

The general architecture of the experts is shown in Figure 7.6. Each expert contains one or two LSTM layers (see Table 7.3 for details) with peephole connections as defined in (3.86) to (3.88). A dropout layer with 50 % dropout rate is used and a dense layer with ReLU activation (see (3.64)). The logits $\tilde{\mathbf{y}}^{e_p} = (\tilde{y}_1^{e_p}, \dots, \tilde{y}_{C_p}^{e_p})$ obtained from the dense layer are normalised using the softmax function (see (3.66)) to obtain the final output $\hat{\mathbf{y}}^{e_p} = (\hat{y}_1^{e_p}, \dots, \hat{y}_{C_p}^{e_p})$ of expert e_p .

For training, the Adam optimiser is used to minimise the estimated cross-entropy loss $\mathbb{H}(P, Q)$ between the true distribution P and the distribution Q learnt by the network. A learning rate of 0.002 and a batch size of 128 are applied. Each example in the batch consists of one symbol with the corresponding next symbol in the sequence as the label. The state of the LSTM layers is kept between batches.

7.3.2.2 (Sparsely-Gated) Mixture of Experts

In the original publication on the MoE architecture [45], the experts and the gating network are trained together. A special error function leads to a partition of the problem space into subspaces which are handled by the individual experts. However, different approaches were suggested to divide the problem space into subspaces before training (e.g. [119, 120]). Based on the partitioning strategy applied, the approaches are classified into two categories [121]. These are Mixture of Implicitly Localised Experts (MILE) and Mixture of Explicitly Localised Experts (MELE). In the case considered here, the experts are trained individually to perform different tasks, hence these instantiations of the network belong to the MELE category.

In the considered application, the gating network needs to identify the current problem (or concept) in order to decide which expert makes the best prediction. Since the input is sequential, an LSTM is trained for the identification task. One gating network is needed per scenario defined in Table 7.2. The gating networks have the same architecture as the experts, which is shown in Figure 7.6. They are trained with a learning rate of 0.0002, cross-entropy loss using the Adam optimiser, and a batch size of $\left\lfloor \frac{128}{|S|} \right\rfloor \cdot |S|$ with $|S|$ the number of problems (equal to the number of experts $|E|$) in the scenario. The first batch is initialised to contain the same number of symbols per problem. In each training step, one symbol is provided with the corresponding

Table 7.4: LSTM parameters of the gating networks.

Scenario	# LSTM layers	# Cells/layer	# Neurons (dense)
HMM	1	32	3
NLP	1	16	5
Hard	1	16	6
Easy	1	16	6
Bad	2	64	7
All	1	16	15

problem p as the label. Also here, the state of the LSTM layer is kept between batches. The number of LSTM layers, the number of cells per layer and the number of neurons in the dense layer are shown in Table 7.4. The size of the dense layer corresponds to the number of problems $|S|$ in the scenario.

The identification accuracy is influenced by the stability period of a concept, i.e. the number of consecutive symbols originating from the same concept before a change occurs. To study this effect, the networks are trained with different stability periods of randomly chosen lengths, ranging from 1 to 1000 symbols. After the specified number of symbols was processed for one problem, data of a different problem is used for training and the stability period is set to a new random length. This is done analogously to training the LSTMs for identification of the emitter type as described in Chapter 6 (see Algorithm 1). Note that the same problem p in scenario S can be selected more than once, but not twice in a row. Algorithm 1 is executed for each chunk of size $\left\lfloor \frac{128}{|S|} \right\rfloor$ in the batch. Due to the randomness, not every problem is necessarily represented in all batches.

For the MoE and the sparsely-gated MoE, the gating networks are identical, the only difference between the architectures being in the “wiring”. A normalisation of the d weights in the sparsely-gated MoE is performed before mixing such that they sum up to 1. All values for $d = 1, \dots, |S| - 1$ are tested, with $|S|$ the number of experts in the scenario. For $d = |S|$, MoE and sparsely-gated MoE are identical. All ensemble architectures implemented in this section perform the mixing on the top 20 output of the experts. If the vocabulary size of the problem is smaller than 20, the output is reduced correspondingly, i.e. $k = \min(C_p, 20)$ for each problem p . The same holds true for the other investigated architectures presented in the following.

7.3.2.3 Stacking (with Input)

Two versions of the combiner are implemented per variant of the stacking architecture. One consists of an LSTM, the other one is built with a feedforward network. As stated above, the combiner is trained with 30 % of the training data per problem. The experts are not trained on this part.

Both versions of the combiner take as input the top 20 predictions of the experts. These consist of the indices of the predicted symbol and the probability as-

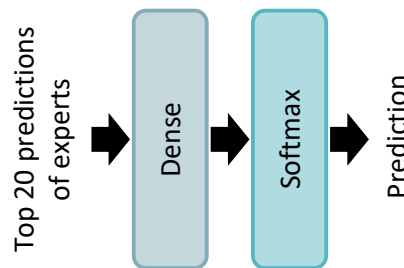


Figure 7.7: The general architecture of the feedforward combiner [63]. The number of dense layers is given in Table 7.6.

Table 7.5: Parameters of the LSTM combiner.

Scenario	With input	# LSTM layers	# Cells/layer	# Neurons (dense)
HMM	no	1	32	21
	yes	1	8	21
NLP	no	2	64	6723
	yes	1	256	6723
Hard	no	2	32	22
	yes	1	16	22
Easy	no	2	64	6723
	yes	2	64	6723
Bad	no	1	32	6723
	yes	2	64	6723
All	no	1	32	6723
	yes	1	128	6723

signed to each symbol. For stacking with input, the current symbol in the stream is used in addition to the top 20 predictions, hence the dimension of the input is $1 + 2 \sum_p \min(C_p, 20)$ in this case and otherwise, it is $2 \sum_p \min(C_p, 20)$, where $p \in S$ are the problems in scenario S . The combiners are also trained with random stability periods (analogous to Algorithm 1), a batch size of $\left\lfloor \frac{128}{|S|} \right\rfloor \cdot |S|$, the Adam optimiser with cross-entropy loss, and a learning rate of 0.002.

The LSTM combiner follows the architecture shown in Figure 7.6, except for the one-hot encoding since the input consists of the top 20 predictions of the experts. The feedforward combiner is built as shown in Figure 7.7, with ReLU used as activation. Tables 7.5 and 7.6 show the number of LSTM or dense layers and the number of cells or neurons of each combiner per scenario. The number of neurons in the last dense layer corresponds to the maximum vocabulary size of the problems contained in the scenario, i.e. $\max_{p \in S}(C_p)$.

Unfortunately, the LSTM combiner is very hard to train. The time needed per 1000 batches, which is measured during training, varies between 1 h to 3 h depending on the scenario and the network size. For the feedforward combiner, the time per 1000 batches lies between 1 min to 2 min. The long training time of the LSTM combiner was the reason to train the combiner on the top 20 output of the experts and not the full output. To keep the comparison fair, the same is done for the feedforward combiner.

7.3.2.4 Online Accuracy-Based Weighting

Online accuracy-based weighting is implemented with two different weighting functions. The first one uses the top k accuracy and the second one a performance estimate based on the MSE. Both methods are described in the following.

Table 7.6: Parameters of the feedforward combiner.

Scenario	With input	# Layers	# Neurons/layer
HMM	no	1	21
	yes	2	64, 21
NLP	no	8	$7 \times 128, 6723$
	yes	8	$7 \times 256, 6723$
Hard	no	1	22
	yes	1	22
Easy	no	8	$7 \times 64, 6723$
	yes	4	$3 \times 64, 6723$
Bad	no	8	$7 \times 64, 6723$
	yes	8	$7 \times 64, 6723$
All	no	2	32, 6723
	yes	2	32, 6723

Weighting Based on Top k Accuracy. Each expert e_p is assigned its mean top k accuracy, which is defined in Section 3.5.4 as an evaluation metric, on the last M input symbols as the weight. The top k accuracy is determined after $M + 1$ symbols have been processed such that the correct predictions are known. Weighting based on the top k accuracy is implemented for $k = 1, 5, 10, 20$.

Weighting Based on Mean Squared Error. The authors of [29] suggest a weighting based on the MSE of each expert in comparison to random guessing. The symbols used by problem p are assumed to be uniformly distributed such that the MSE for random guessing is given by

$$MSE_{rand}^p = \left(1 - \frac{1}{C_p}\right)^2. \quad (7.6)$$

For each expert e_p , the MSE corresponds to

$$MSE_{e_p} = \frac{1}{M} \sum_{m=1}^M \left(1 - \hat{y}_{m,y_m}^{e_p}\right)^2, \quad (7.7)$$

with M the number of processed symbols and $\hat{y}_{m,y_m}^{e_p}$ the probability assigned by expert e_p to the correct next symbol in the sequence, where y_m denotes the label of example m . The MSE is determined when symbol $M + 1$ arrives such that the correct predictions for the last M symbols are known. The weight of expert e_p is then proportional to

$$\hat{w}_p = \max(MSE_{rand}^p - MSE_{e_p}, 0). \quad (7.8)$$

Both variants of the online accuracy-based weighting assign the same weight $\frac{1}{|S|}$ to all experts at initialisation. Also, the weights are normalised to sum up to 1 and the final output of the ensemble is calculated on the top 20 outputs of each expert.

The choice of M is important when considering concept drift. Two configurations for both variants are examined, which represent two extreme cases. In the first, all symbols processed so far are used to compute the MSE, i.e. M grows with time. In the second configuration, the weights are reset when the concept changes, i.e. the weights are only calculated on the symbols originating from a single concept and M corresponds to the stability period. This configuration can be applied in situations where a concept drift can be reliably detected.

Methods for detecting concept drift usually analyse how the accuracy of the model varies, e.g. [122]. Recently, a method based on an independence test between windows on the data stream was suggested [33]. Moreover, context knowledge about the data might be available and used for drift detection. For example, in the application of predicting radar emissions, start and end of a deinterleaved PDW sequence are known due to the signal sorting process. This context knowledge can be used to perform a weight reset at the end of a sequence.

7.3.2.5 Model Averaging

Like in the other architectures, the top 20 predictions of the experts are averaged. All symbols not contained in the top 20 prediction are assigned a probability of 0 since it is assumed that their influence on the final result is negligible. Consequently, the computational efficiency is increased. After averaging the outputs, the final prediction is obtained by computing the top 20 predictions on the new symbol probabilities.

7.3.3 Experimental Results

All architectures are tested with several constant stability periods per scenario, namely 1, 5, 10, 50, 100, 200, 400, 600, 800, 1200, 1600, and 2000 symbols. Although the experts are only trained with a stability period of up to 1000 symbols, longer periods are included because some effects are better visible for those. For each stability period, a sequence with alternating data from the different problems is randomly generated from the test data that is defined in the SPiCe dataset. The sequence generation is done analogously to Algorithm 1, but with a fixed stability period, i.e. $len_{min} = len_{max}$. The generated sequences are stored such that each architecture is tested with the exact same data. Sequences from problems with less data are repeated such that the test data contains an equal number of periods per problem. With a batch size of 128, at least 128 stability periods per problem are used for evaluation. Each example in a batch starts with a different offset into the sequential data of the problem such that each sequence is different.

In some applications, a concept drift can be detected by using a drift detector, e.g. [33, 122], or by exploiting context knowledge. Each architecture is therefore tested with and without a state reset of the LSTM experts at the end of a stability period.

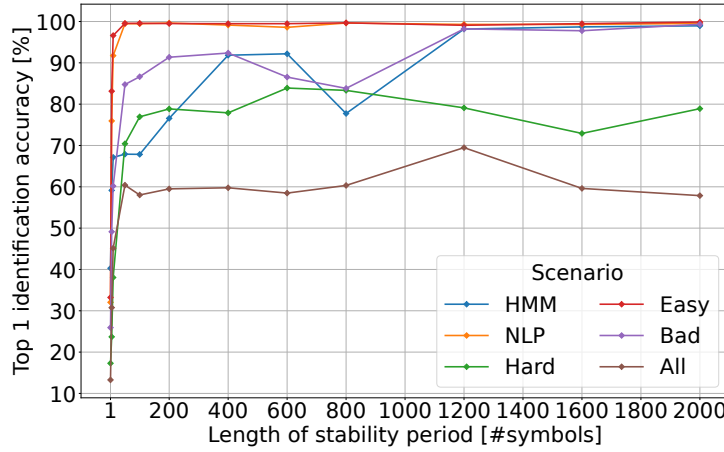


Figure 7.8: Top 1 identification accuracies of the gating networks used in the MoE variants for the different scenarios [63].

Moreover, the two variants of the online accuracy-based weighting architecture are evaluated with and without a weight reset at this point.

To give a reference for the prediction accuracy of the individual experts, the sparsely-gated MoE architecture is also included with $d = 1$ and perfect identification. This means that the gating network outputs the labels and hence, the expert that was trained on the current concept is always selected for the prediction. Of course, this architecture cannot be used in a real application as the labels are not available.

To show what the identification difficulty of a scenario means, the identification accuracies of the gating networks in the different scenarios are shown in Figure 7.8. The problems in the *NLP* and *easy* scenario can be identified with a probability close to 100 %, also with a short stability period. For the problems in the *HMM* and *bad* scenario, longer stability periods are needed to reliably distinguish between them. The problems in the *hard* and *all* scenario cannot be identified with the high accuracies observed for the other scenarios.

For better readability, the results of the methods performing best in general are displayed in Figure 7.9, while showing only the results of selected values for d in the sparsely-gated MoE architecture. The results of the methods with lower accuracies are shown in Figure 7.10. Both figures contain the results of the sparsely-gated MoE with perfect identification as a reference. Detailed result tables can be found in Appendix A.2, as well as a single figure containing all results (Figure A.3). The accuracies are reported at the end of a stability period, i.e. after all parts of the ensemble have processed all symbols originating from the current concept.

As can be seen from the Figures 7.9 and 7.10, the MoE variants and the MSE-based weighting with a weight reset after a stability period provide better results than all other methods in all scenarios. An exception is the *HMM* scenario, in which no clear best method can be identified. However, also in this scenario the MSE-based weighting with weight reset performs slightly better than the other methods for most stability periods.

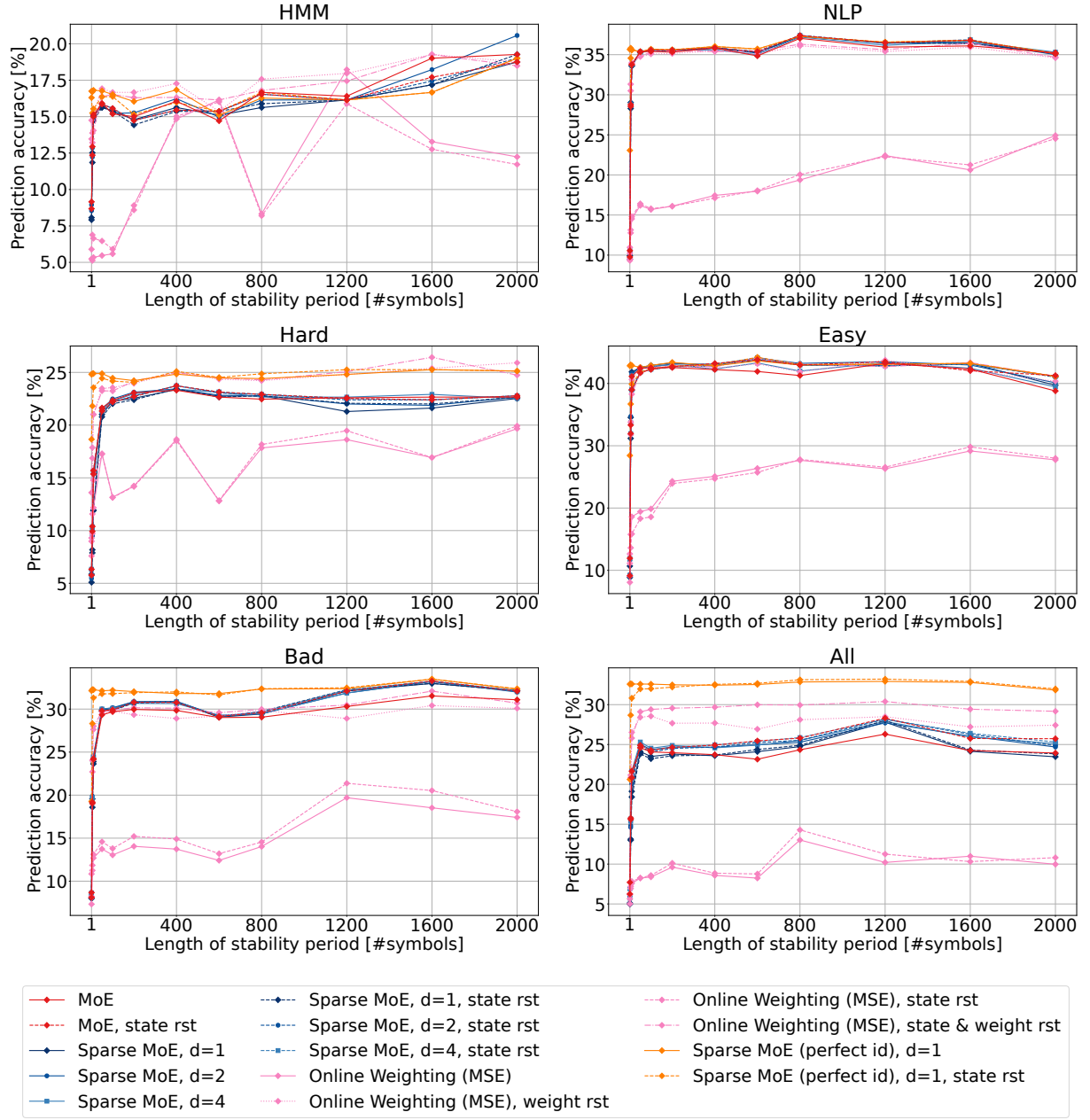


Figure 7.9: Top 1 prediction accuracies of the MoE variants and the MSE-based weighting [63].

In situations in which a drift detection is possible and the identification accuracy is low, the MSE-based weighting with weight reset clearly outperforms every other method (*hard*, *all* in Figure 7.9). Furthermore, this method has the advantage that it does not need to be trained. However, if the detection of a concept drift is not possible and therefore no weight reset can be performed at the end of a stability period, the results of the MSE-based weighting are much worse. In these situations, the MoE or sparsely-gated MoE are to be preferred. If the identification accuracy is high (*NLP*, *easy* in Figure 7.9), the MoE variants perform better, but there is not much difference to the MSE-based weighting with weight reset.

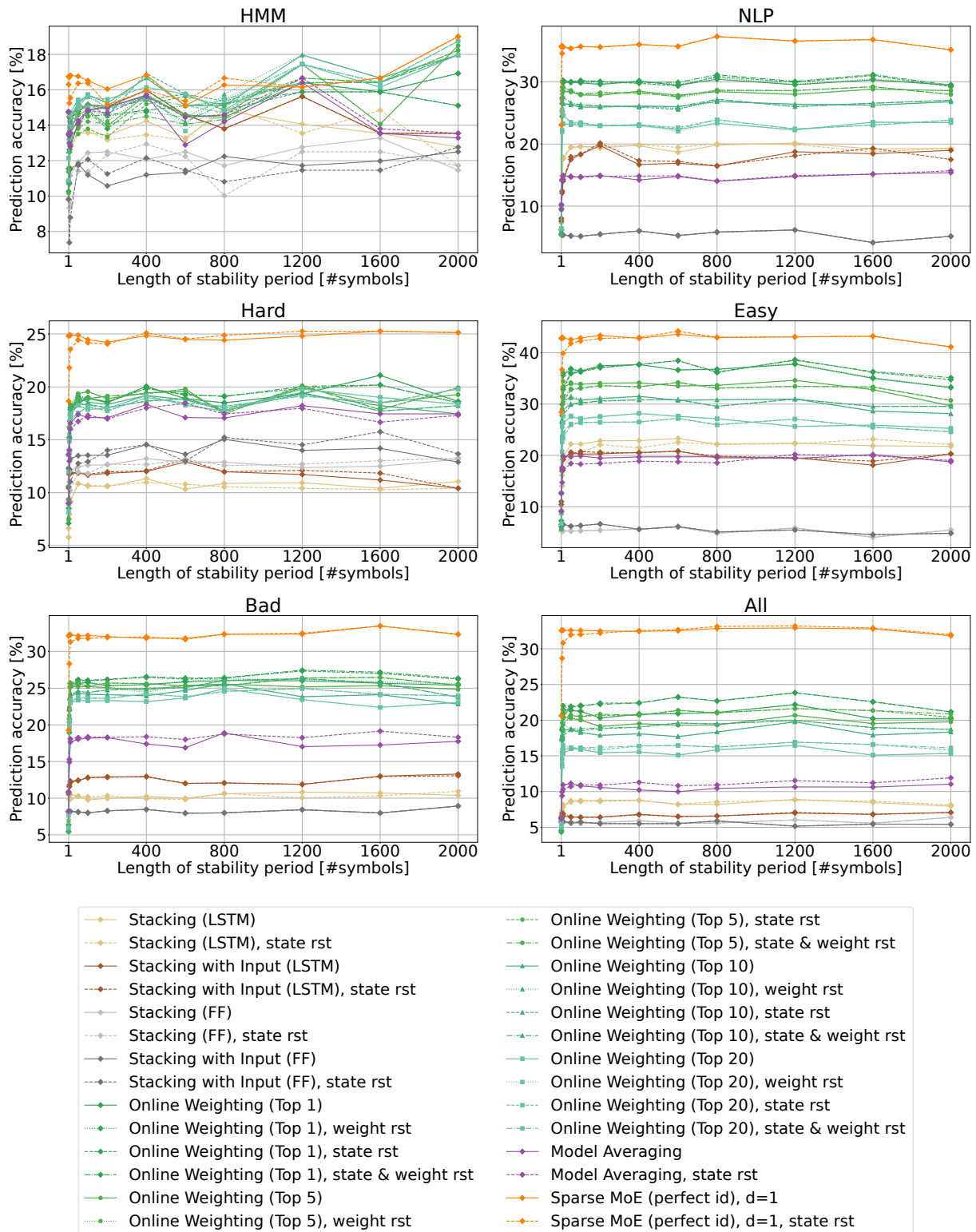


Figure 7.10: Top 1 prediction accuracies of the stacking variants, the online weighting based on top k accuracy, the model averaging, and the sparsely-gated MoE with perfect identification [63]. FF = feedforward.

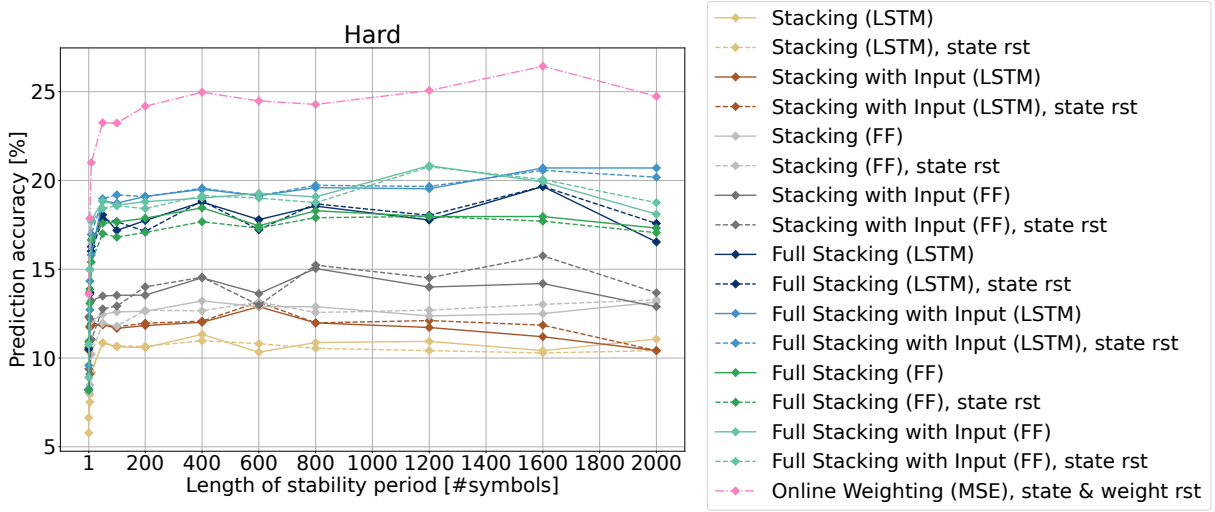


Figure 7.11: Comparison of the top 1 prediction accuracies of the different stacking variants for the *hard* scenario [63]. The results of the MSE-based weighting are shown for reference. FF = feedforward.

Also in situations with low identification accuracy (*hard, all*), it is better to use the sparsely-gated MoE with some $d > 1$ instead of the full MoE if no drift detection is available. With drift detection and a state reset at the end of a stability period, the sparsely-gated MoE with $d > 1$ and the full MoE perform more or less the same.

The online weighting based on top k accuracy provides the best performance for $k = 1$ (see Figure 7.10). However, it is outperformed by the MoE variants and the MSE-based weighting with weight reset. With increasing k , the prediction accuracy even decreases.

Stacking does not perform well for the scenarios in general as shown by the results in Figure 7.10. The LSTM combiner provides better results than the feedforward combiner in most of the scenarios, but is still not comparable to the results of the best methods. To see whether this is due to the decision to train on the top 20 predictions of the experts, the evaluation is repeated with combiners that are trained on the full output of the experts. However, this could not be done for every scenario since in most of them the input size is very large and training would have taken months. The size of the full output of the experts equals the sum of the individual vocabulary sizes. Nevertheless, to give an impression on the performance of full stacking, the combiners for the *hard* scenario are trained since here the sum of the vocabulary sizes is only 98 and not 7559 as for example in the *NLP* scenario. The results are shown in Figure 7.11, with the accuracies of the MSE-based weighting with state and weight reset given for reference. Full stacking performs significantly better than stacking based on the top 20 predictions, however, the accuracy is still much lower than that of the MoE variants and the MSE-based weighting with weight reset.

The results achieved by model averaging are also much worse than those of the best methods (see Figure 7.10). This could be expected since this architecture does not adapt to the estimated accuracies of the experts.

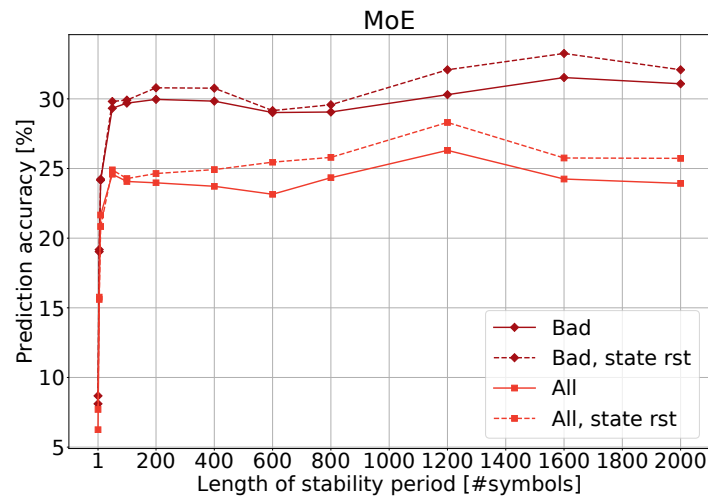


Figure 7.12: Top 1 prediction accuracies of the MoE architecture for the *bad* and *all* scenario with and without state reset [63].

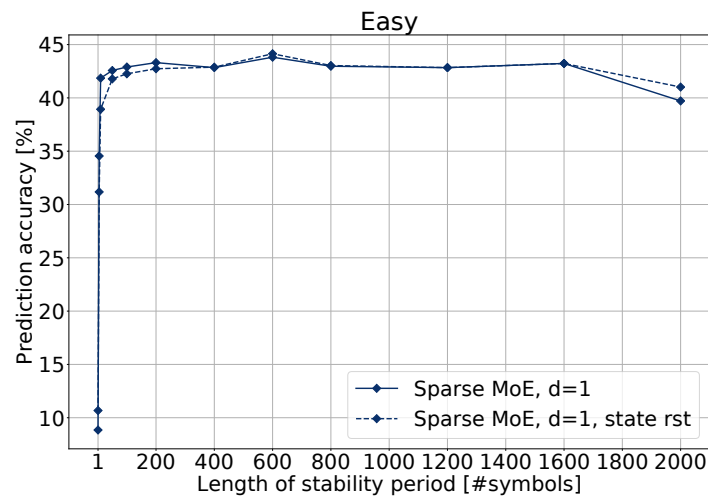


Figure 7.13: Top 1 prediction accuracy of the sparsely-gated MoE with $d = 1$ for the *easy* scenario with and without state reset [63].

When looking at the accuracy of the MoE with and without a state reset of the LSTM experts, one can see that a state reset is especially beneficial in the *bad* and the *all* scenario (plotted individually in Figure 7.12). In these scenarios, the internal states of the LSTM experts are corrupted by the irrelevant data. The *bad* scenario was chosen such that it is particularly challenging for the experts to predict symbols of other concepts. Consequently, the variety among the sequences is much higher, which therefore leads to a worse corruption of the internal state. In the other scenarios, the effect is not as big since there are less problems or the problems are more similar.

However, a state reset is not always advantageous as can be seen from the top 1 accuracy of the sparsely-gated MoE with $d = 1$ for the *easy* scenario in Figure 7.13.

Here, the identification accuracy is high also with short stability periods. Hence, the LSTM experts only process the data from the problem that they were trained on when employing the sparsely-gated MoE with $d = 1$. Therefore, keeping the LSTM states improves the performance since the experts are provided more information about the current sequence. This effect is visible up to a stability period length of 400 symbols. If the stability period becomes even longer, it already contains enough information such that a state reset does not cause harm.

The influence of the length of the stability period on the prediction accuracy decreases with increasing length for all methods except for the MSE-based weighting without weight reset (see Figures 7.9 and 7.10, as well as the result tables in the appendix). For very short periods of only 1 or 5 symbols, the prediction accuracy of all methods is much lower than for longer periods, but rapidly increases until it converges to a maximum. The low accuracy with short stability periods was to be expected since less symbols from the same concept provide less information for the weighting and the predictions made by the experts.

7.4 Ensembles of Predictive Radar Models

This section presents different approaches for ensembles of predictive radar models. Based on the results obtained in the previous section, selected architectures are implemented and evaluated. Section 7.4.1 provides details on the implementation of the architectures and Section 7.4.2 presents the experimental results.

7.4.1 Implementation

In Chapter 5, predictive radar models are developed using LSTMs and MCs with the hierarchical emission model. It is shown that syllables and words are the most interesting modelling levels because for letters, commands, and functions, simple prediction strategies like repeating the last symbol provide results comparable to MCs and LSTMs. For syllables, the MC approach provides better results while LSTMs are to be preferred for words. Therefore, in this section, ensembles of MC experts for syllables and LSTM experts for words are implemented and evaluated. The results of the previous section show that the MoE, the sparsely-gated MoE, and the MSE-based weighting with weight reset provide the best results from all architectures tested. Hence, the evaluation in this section is restricted to these approaches.

The MC experts for syllables are those developed in Section 5.2.2. For the MoE and sparsely-gated MoE architecture, the $\text{LSTM}_{\text{scen}}$ for identification as presented in Section 6.2.1 is employed as the gating network. Although the MC provides better results for the identification based on syllables, it is not used here since it is not robust with respect to missing and additional syllables. As the MC experts do not have a memory, a state reset is not meaningful. For the same reason, the weighted mean of the prediction accuracies of the MCs $\overline{\text{acc}}_w$ is taken as a reference for the expected ideal case instead of the sparsely-gated MoE with perfect identification and $d = 1$.

The weights are set to the amount of test data available for each emitter type e in the set of emitters \mathcal{E} , $\text{data}(e)$, as given in Table 5.2, i.e.

$$\overline{\text{acc}}_w = \frac{\sum_{e \in \mathcal{E}} \text{data}(e) \cdot \text{acc}_e(\hat{Y}, Y, k)}{\sum_{e \in \mathcal{E}} \text{data}(e)}, \quad k = 1, \quad (7.9)$$

with $\text{acc}_e(\hat{Y}, Y, k)$ being the top k accuracy as defined in (3.78).

For words, the experts are the LSTMs presented in Section 5.2.1. In the MoE and sparsely-gated MoE architecture, the $\text{LSTM}_{\text{scen}}$ developed in Section 6.2.1 is employed as the gating network because it is more robust with respect to additional words than the $\text{LSTM}_{\text{rand}}$, which provides the best results with ideal data. Both MoE variants, as well as the MSE-based weighting, are evaluated with and without a state reset after the sequence length. The MSE-based weighting is implemented as described in Section 7.3.2.4, both for syllables and words, where C_p corresponds to the number of syllables and words used by each emitter, respectively.

7.4.2 Experimental Results

The following sections present the results for the different ensemble architectures for syllables and words. At first, the approaches are evaluated under ideal conditions and afterwards with corrupted data that contains missing and additional symbols. The top 1 prediction accuracies are given as defined by (3.78) with $k = 1$.

7.4.2.1 Evaluation Under Ideal Conditions

A mistake in the estimated probabilities for the experts leads to the selection of the wrong emitter model in the sparsely-gated MoE network with $d = 1$ and a disadvantageous assignment of weights in the other architectures. To see how much impact a wrong identification has on the prediction performance, the emitter models are tested with the data of all three radar variants. The results are shown in Figure 7.14. On the

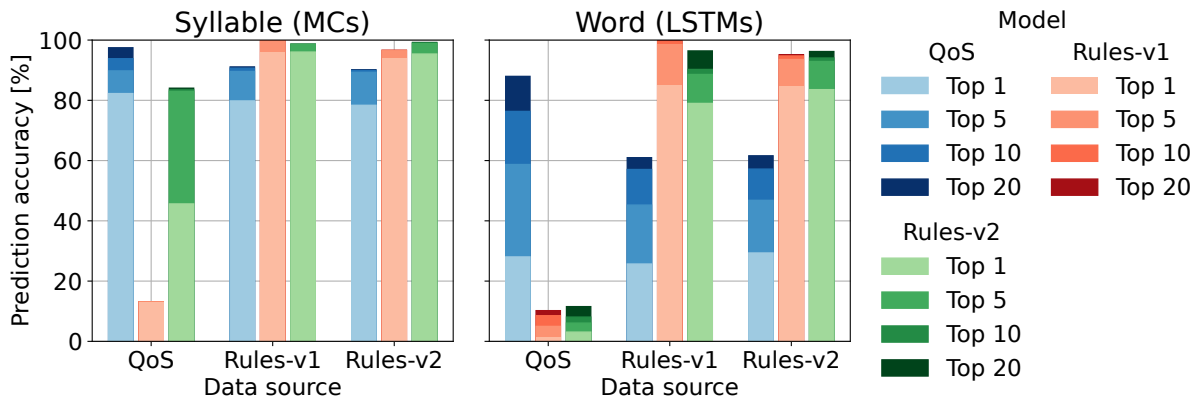


Figure 7.14: Prediction accuracies of the emitter models on the data of all emitters [64].

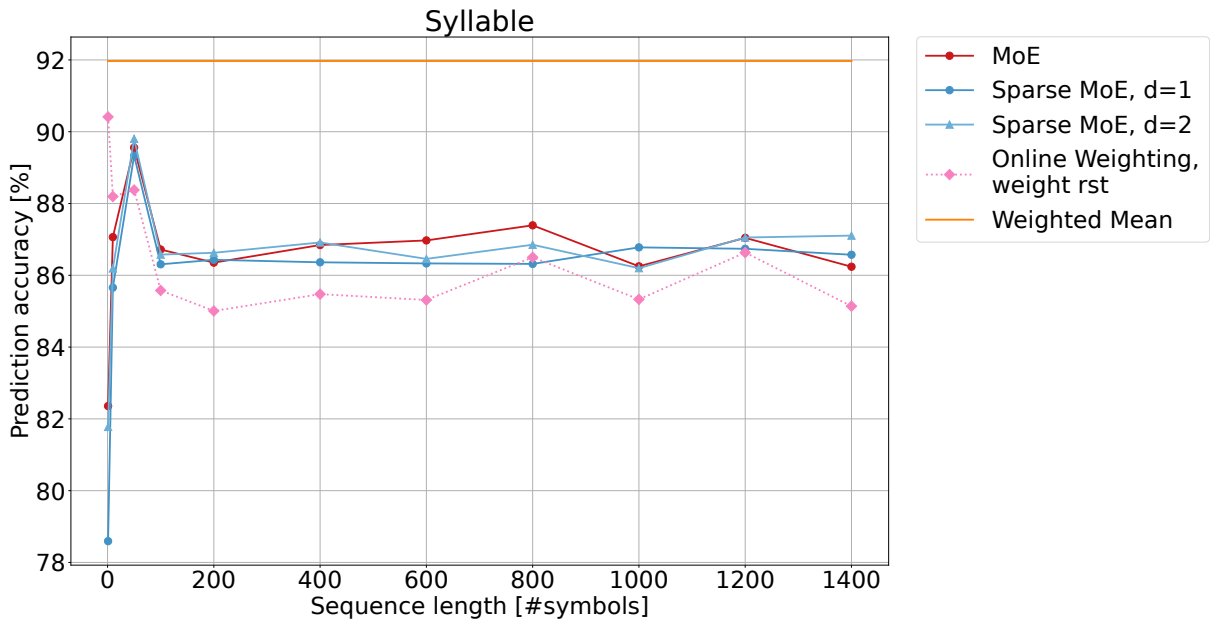


Figure 7.15: Top 1 prediction accuracy of the different ensemble architectures with MC experts for syllables [64].

x-axis of each plot, the emitter that is the source of the data is shown. The colour of the bars indicates which emitter model is used to make the prediction. For example, the leftmost red bar in each plot shows the prediction accuracy of the model for the Rules-v1 radar on the data of the QoS radar.

Although the Rules-v2 radar uses many more syllables and words than the Rules-v1 radar, the Rules-v1 model is able to predict the Rules-v2 emissions with a top 1 accuracy of 94 % for syllables and 84.84 % for words. The Rules-v2 model also achieves a high accuracy for the symbols of the Rules-v1 radar, especially for the syllables, for which the difference to the top 1 accuracy of the Rules-v2 model is just -0.14 percentage points. The results of Chapter 5 already suggested that although the Rules-v2 radar can emit many different syllables and words, the most frequently used symbols dominate the data. The fact that the prediction accuracy is that high also implies that the two rule-based radars are hard to distinguish. Since the behaviour of the QoS radar regarding syllables and words is much different to the rule-based approaches, none of the rule-based emitter models is capable of predicting the syllables and words of the QoS radar and also the QoS model achieves much lower prediction accuracies for the data of the other radars than the corresponding models. Consequently, a confusion between the two rule-based emitters in an ensemble does not cause much harm, however, the correct identification of the QoS radar is important.

Syllables with MC Experts. Figure 7.15 shows the results for the different architectures and Table 7.7 provides the detailed values. The accuracies of all ensembles are very similar at longer sequence lengths. The MSE-based weighting achieves the lowest accuracy with 85.1 % at a sequence length of 1400 syllables, while the sparsely-

Table 7.7: Top 1 prediction accuracy [%] of the ensemble architectures (arch.) in different configurations (cfg.) with MC experts for syllables. S = State, W = Weight.

Arch.	Cfg.	Reset		Sequence length										
		S	W	1	10	50	100	200	400	600	800	1000	1200	1400
MoE (LSTM _{scen})				82.4	87.1	89.6	86.7	86.4	86.8	87.0	87.4	86.2	87.0	86.2
Online Weighting	MSE		x	90.4	88.2	88.4	85.6	85.0	85.5	85.3	86.5	85.3	86.6	85.1
Sparse MoE (LSTM _{scen})	d=1			78.6	85.7	89.3	86.3	86.4	86.4	86.3	86.3	86.8	86.7	86.6
	d=2			81.8	86.2	89.8	86.6	86.6	86.9	86.5	86.9	86.2	87.1	87.1
Weighted Mean				92.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0

gated MoE with $d = 2$ achieves the highest accuracy with a value of 87.1 %. Only for a sequence length of one symbol, the MSE-based weighting clearly outperforms the other methods with an accuracy of 90.4 % compared to the second best result of 82.4 % obtained by the MoE. In general, the sparsely-gate MoE with $d = 2$ provides slightly better results than the other architectures. As seen from Figure 7.14, the two rule-based radar models are able to predict each other's emissions very well and consequently, an ensemble of these models is beneficial for the accuracy. For the same reason, the difference to the configuration with $d = 1$ is very small, although the LSTM_{scen} employed as the gating network assigns about 60 % to 80 % of the syllables of the Rules-v1 radar to the Rules-v2 radar, depending on the sequence length. However, none of the methods achieves the theoretically best value estimated by the weighted mean of the prediction accuracies of the emitter models of 92 %. The drop in accuracy from a sequence length of 50 to 100 syllables is due to misclassifications with high confidence by the LSTM_{scen}. With a sequence length of 100 syllables and more, the QoS radar is given a much higher weight. If now the two rule-based radars are misclassified as the QoS emitter, the prediction of the expert for the QoS radar dominates the weighted sum provided in the output of the MoE variants. As seen in Figure 7.14, the QoS emitter model achieves a lower prediction accuracy on the syllables of the rule-based emitters and hence, the accuracy of the overall ensemble decreases.

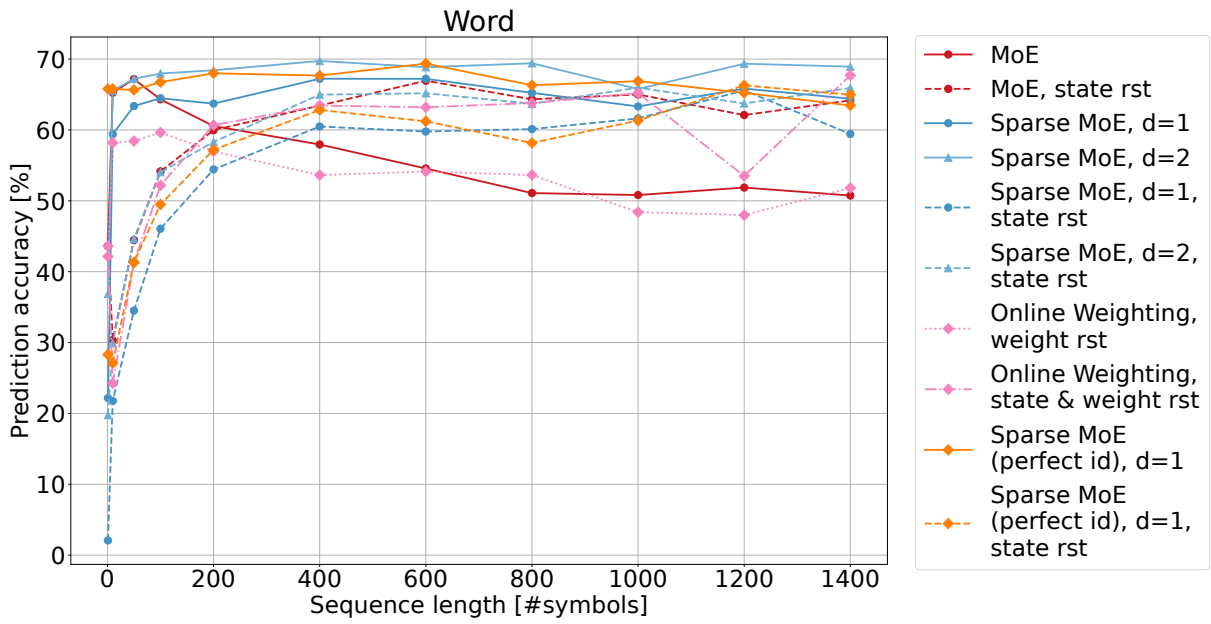


Figure 7.16: Top 1 prediction accuracy of the different ensemble architectures with LSTM experts for words [64].

Words with LSTM Experts. Figure 7.16 shows the results of the different ensemble architectures with LSTM experts for words. In Table 7.8, the detailed values are given. As is the case for syllables, the sparsely-gated MoE with $d = 2$ outperforms the other methods in general. However, the difference to the other architectures is larger than for syllables. The highest accuracy of 68.9 % with 1400 words is given by the sparsely-gate MoE without a state reset and $d = 2$. This is even higher than the accuracy of 63.5 % for the sparsely-gated MoE with perfect identification. Here, the positive effect of an ensemble of experts, in this case the two rule-based models, is observed.

The impact of a state reset after a sequence length can clearly be seen. At shorter sequence lengths of up to 100 words, a state reset results in a large decrease in the prediction accuracies for the MoE architecture. With a sequence length of 10 words, the state reset causes a drop from 65.2 % to 30.3 % prediction accuracy. At a sequence length of 200 words, the results with and without a state reset are about the same (60.0 % vs. 60.5 %). With longer sequences, the variant with state reset clearly outperforms the MoE without reset, for which the accuracy decreases with increasing sequence length. Also for the MSE-based weighting, this behaviour can be observed. The sparsely-gated MoE, however, does not benefit from a state reset. This matches the observations made in the previous section with the SPiCe data. With shorter sequences, it is better to keep the LSTM state since then, the experts receive more information. As seen from the results of the sparsely-gated MoE with perfect identification and state reset, the accuracies of the radar models improve with increasing sequence lengths. At the same time, however, the state is corrupted by irrelevant data in the classifier fusion architectures, which are MoE and MSE-based weighting in this case. An improvement might be achieved by “warming up” the LSTMs with parts of the training data after a state reset.

Table 7.8: Top 1 prediction accuracy [%] of the ensemble architectures (arch.) in different configurations (cfg.) with LSTM experts for words. S = State, W = Weight.

Arch.	Cfg.	Reset		Sequence length										
		S	W	1	10	50	100	200	400	600	800	1000	1200	1400
MoE (LSTM _{scen})				43.6	65.2	67.2	64.3	60.5	57.9	54.6	51.1	50.8	51.9	50.7
		x		43.6	30.3	44.4	54.1	60.0	63.4	66.9	64.3	65.0	62.1	64.2
Online Weighting	MSE		x	42.2	58.2	58.4	59.6	57.0	53.6	54.1	53.6	48.4	48.0	51.8
	MSE	x	x	43.6	24.2	41.3	52.2	60.7	63.4	63.2	63.8	65.2	53.5	67.7
Sparse MoE (LSTM _{scen})	d=1			22.2	59.4	63.4	64.5	63.7	67.2	67.2	65.2	63.3	65.8	64.4
	d=2			36.9	65.5	67.2	68.0	68.4	69.7	68.9	69.4	65.8	69.3	68.9
	d=1	x		2.1	21.8	34.5	46.1	54.4	60.5	59.8	60.1	61.6	65.6	59.4
	d=2	x		19.8	29.9	44.6	54.0	58.3	65.0	65.2	63.7	66.0	63.7	65.9
Sp. MoE (perfect id)	d=1			65.8	65.8	65.6	66.7	68.0	67.7	69.4	66.3	66.9	65.2	63.5
	d=1	x		28.3	27.1	41.3	49.5	57.2	62.8	61.2	58.2	61.3	66.3	65.0

7.4.2.2 Evaluation with Missing and Additional Symbols

This section presents the relative results with missing or additional symbols and words with respect to the results obtained for ideal data. Here, the scenario with 20 % corrupted data is exemplarily shown as this is the worst case scenario for the predictive models and the identification approaches in the Chapters 5 and 6. For improved readability, only the results at a sequence length of 1400 symbols are given. The relative differences are defined by

$$\text{acc}_{rel} = \frac{\text{acc}(\hat{Y}_{corrupt}, Y_{corrupt}, k) - \text{acc}(\hat{Y}, Y, k)}{\text{acc}(\hat{Y}, Y, k)}, \quad k = 1, \quad (7.10)$$

with $\text{acc}(\hat{Y}, Y, k)$ as defined in (3.78).

Syllables with MC Experts Figure 7.17 shows the accuracies as well as the relative differences with missing or additional syllables. It is seen from the figure that all ensemble architectures are more or less equally robust and exhibit less decrease of the accuracy than the weighted mean of the emitter models' results. Still, the absolute accuracy of the weighted mean is higher. For additional syllables, the accuracy decrease is the same for all architectures. For missing symbols, the MSE-based weighting is the most robust, however, the decrease of 6 % is very close to the worst loss of 8 % by the two sparsely-gated MoE configurations.

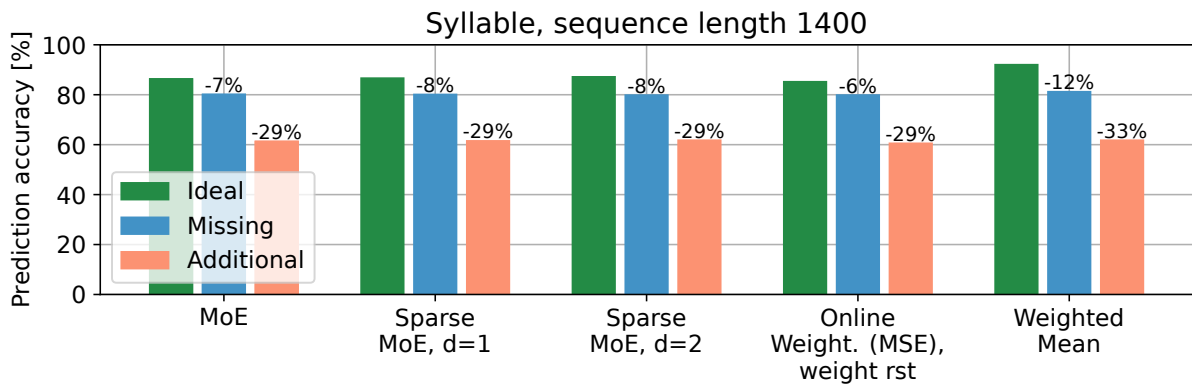


Figure 7.17: Top 1 prediction accuracy of the different ensemble architectures with MC experts for syllables and 20 % corrupted data at a sequence length of 1400 symbols [64].

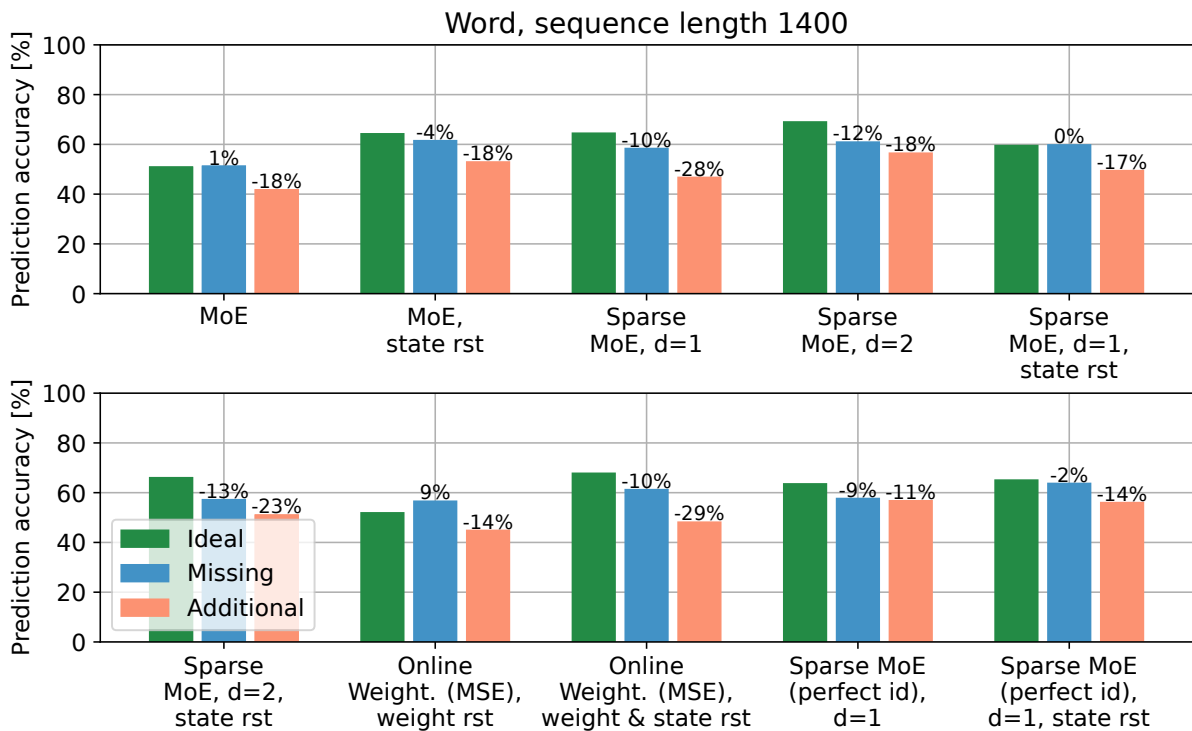


Figure 7.18: Top 1 prediction accuracy of the different ensemble architectures with LSTM experts for words and 20 % corrupted data at a sequence length of 1400 symbols [64].

Words with LSTM Experts Figure 7.18 shows the results of the ensembles of LSTM experts with 20 % corrupted data and at a sequence length of 1400 words. For missing words, the MSE-based weighting without a state reset exhibits an increase of 9 % relative to the results with ideal data. Nevertheless, the absolute accuracy of 56.48 %

is worse than the best result with missing words, which is 61.4 % and achieved by the MoE with state reset. However, with a state reset, the MSE-based weighting comes close with an accuracy of 61.14 %, followed by the sparsely-gated MoE with $d = 2$ and 60.83 %. With additional words, the sparsely-gated MoE with $d = 2$ clearly outperforms the other methods with an accuracy of 56.37 %, although it is not the most robust architecture, which is again the MSE-based weighting without state reset. The second best result with additional words is provided by the MoE with 52.84 %.

7.5 Summary

This chapter presents and evaluates ensemble architectures of predictive models. In the first part, the architectures are compared for processing streaming data via LSTMs with sudden, recurring concept drift under different conditions and configurations with a public dataset. Based on the insights gained from this evaluation, the second part presents three selected approaches for the predictive radar models developed in Chapter 5. In the considered application, the data from different concepts, which correspond to the emitters, is alternating. Due to the properties of the ELINT processing chain, start and end of a deinterleaved PDW sequence are known, i.e. a concept drift can be detected. Therefore, it is possible to perform a state and a weight reset after a sequence length or stability period.

The evaluations of the first part show that the sparsely-gated MoE architecture is in general a good choice when employing LSTMs, also if the change of the concept cannot be detected. However, the number of experts d that should be activated for the prediction needs to be chosen and a gating network needs to be trained. In situations with high identification accuracy, d can be smaller than in situations with low identification accuracy. This finding is confirmed by the evaluations with predictive radar models in the second part, for which the sparsely-gated MoE with $d = 2$ and without a state reset provides the overall best results. In the case of the radar models, the MoE is equal to the sparsely-gated MoE with $d = 3$, for which the results for words are worse than with $d = 2$. Since the gating network is able to identify the QoS radar with high accuracy but only confuses the two rule-based emitters, $d = 2$ is a good choice.

If drift detection is available and the identification of the current concept is not possible with high accuracy, the MSE-based weighting appears to be the most effective method on the SPiCe data. Here, the benefits of classifier fusion can clearly be seen. With a weight reset at the end of a stability period, the MSE-based weighting outperforms every other method and has the advantage that no additional training is required. Also in situations with high identification accuracy, it comes close to the results of the MoE variants. This is as well observed for the ensembles of MC radar models for syllables, for which the identification accuracy is lower than that of words. For ensembles of LSTM experts for words, however, the identification accuracy is rather high and the sparsely-gated MoE with $d = 2$ provides much better results.

The evaluation on the SPiCe data shows that the internal states of the LSTM experts are clearly affected by the data of different concepts. Therefore, a state reset

after a stability period is beneficial in situations where a concept drift can be detected and classifier fusion is employed. If no drift detection can be performed, a method that uses classifier selection like the sparsely-gated MoE without a state reset might provide better results. With classifier selection, the corruption of the internal states is minimised since not all of the experts process all data. In the case of the predictive radar models, it is observed that the two rule-based emitters are able to predict each other's emissions very well, which means that they behave in a very similar manner. Consequently, the state is not corrupted as badly as with the data of the QoS radar. Since the QoS radar can be identified with high accuracy, the sparsely-gated MoE with $d = 2$ performs a fusion of the results of the two rule-based approaches if the input belongs to one of them, which is beneficial for the prediction accuracy.

With corrupted data, all ensembles of MC radar models for syllables are equally robust, while the LSTM ensembles exhibit a larger variety. The most robust model is the MSE-based weighting with weight reset, however, the absolute results are worse than those of the others. With missing words, the MoE with a state reset provides the best absolute accuracy, while the sparsely-gated MoE with $d = 2$ is the best method for additional words. However, for missing words the result of the sparsely-gated MoE is also very close to that of the MoE.

Chapter 8

Recognition of Unknown Radar Emitters

This chapter presents several approaches for recognising if a received signal belongs to an unknown emitter. It provides an introduction and a literature review on the general topic of open-set recognition, as well as detailed descriptions of the different methods and a thorough evaluation under several conditions. Parts of this chapter have been published in [65].

8.1 Introduction

As the goal of ELINT is to collect information about radar systems, recognising unknown emitters is of great importance. If unknown radars are detected, the corresponding signals need to be prioritised and recorded such that further analysis can be performed and possibly models of the emitters' behaviour can be learnt. However, classifiers like neural networks are generally trained to identify a set of known classes, which is sometimes called the closed-set assumption. Recognising if an input does not belong to any of the known classes is referred to as open-set or open-world recognition, e.g. [51–56]. Table 8.1 provides a characterisation of open-set recognition in comparison to classification and anomaly detection. The major differences are the goal, the available training data, and the output that the classifier provides in each task. Approaches from the area of anomaly detection could as well be used to classify input as unknown, however, an anomaly detector does not distinguish between

Table 8.1: Characterisation of classification, anomaly detection, and open-set recognition [55].

	Classification	Anomaly detection	Open-set recognition
Goal	Discrimination between known classes	Detection of abnormal data	Identification of data from known classes
Training data	Data of all known classes	Typical data and few or no outliers	Data of known classes and few or none of unknown classes
Output	Label of a known class	Anomaly: yes/no	Label of a known class or “unknown”

several known classes. A combination of an anomaly detector with a classification approach therefore results in a method for open-set recognition. The authors of [52] define the different types of input classes that a classifier for open-set recognition needs to handle as:

- **Known classes:** The classes that the classifier should recognise and identify. The set of known classes is denoted by \mathbb{K} in the following.
- **Known unknown classes:** Input belonging to known unknown classes, denoted by \mathbb{V} , should be classified as unknown/rejected. Type and structure are known and either training data is available or can be generated.
- **Unknown unknown classes:** Data of unknown unknown classes, denoted by \mathbb{U} , should as well be classified as unknown, but is unavailable at training time and only encountered at test time.

As an example, consider a classifier that should detect and identify traffic signs in images for an autonomous driving application. The known classes are the different traffic signs, for which training data is available. The known unknown class consists of example images without traffic signs. The classifier can as well be trained with these images. However, the known unknown class cannot cover the complete range of images without traffic signs encountered in the real world, which are the members of the unknown unknown class.

Open-set recognition can be regarded as a learning task with these properties [55]:

- If the input x belongs to a known class $k_c \in \mathbb{K}$, which is denoted by $\text{class}(x) = k_c$, then the label y corresponds to the class index $\text{idx}(k_c) = c$, otherwise $y = \emptyset$, with \emptyset meaning “unknown”.
- The goal of the learning process is to train a classifier \hat{c} that returns $\hat{c}(x) = c$ for all x with $\text{class}(x) = k_c \in \mathbb{K}$ and $\hat{c}(x) = \emptyset$ otherwise.
- Predicting the class $\hat{c}(x) = \emptyset$ is referred to as the reject option. The learning process is supposed to adjust the selection of this option by the classifier.

8.1.1 Related Work

There are two alternative principles employed for open-set recognition. The first one is to compare the output values $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_C)$ for each class $k_c \in \mathbb{K} = \{k_1, \dots, k_C\}$ to a threshold δ and reject the input as unknown if none of the values exceeds it, i.e.

$$\hat{c}(x) = \begin{cases} T_1(\hat{\mathbf{y}}) & \text{if } \max(\hat{\mathbf{y}}) \geq \delta, \\ \emptyset & \text{else,} \end{cases} \quad (8.1)$$

where $T_1(\hat{\mathbf{y}})$ denotes the index of the top 1 entry in $\hat{\mathbf{y}}$ as defined in (3.77), which corresponds to the index c of the most probable known class k_c . The second approach is to add an extra class with index $C + 1$ called “unknown” to the possible output

classes. The classifier can be trained on the known and known unknown classes, but obviously not on the unknown unknown classes.

The work presented in [53] introduces *OpenMax*, which estimates the probability that the input belongs to an unknown class, to replace the commonly used softmax layer in a neural network (see (3.66)). The basis is the analysis of the logits, i.e. the activations \tilde{y} of the final dense layer before softmax. Per class, the logits of every correctly classified training input, which consists of images in [53], are combined into a mean activation vector to yield a single representation of the class. Based on the logits and the corresponding mean activation vector, a Weibull distribution is fitted for each class with

$$f_{Weibull}(x) = \lambda\kappa(\lambda x)^{\kappa-1} \exp(-(\lambda x)^\kappa), \quad (8.2)$$

where λ and κ need to be estimated from the data. OpenMax then adapts the top k logits by scaling them according to the Weibull probability of the distance between the input x and the mean activation vector of the class. In addition, a pseudo-activation of the unknown class is determined from the original logits and the Weibull model. Afterwards, the revised logits and the pseudo-activation are normalised through the softmax function. If now the probability for the unknown class is the highest or all values are below a threshold, the input is rejected as unknown.

Although achieving good performance in comparison to directly using the softmax function with a threshold, it is not clear how to apply OpenMax to sequential data. The mean activation vector would need to be calculated for many sequences with different offsets and lengths to capture the general distribution of the logits for a certain class, which does not seem to be practical.

A network to generate data of the known unknown classes is proposed in [123], which is designed to find data samples that are close to the known classes but that are still rejected. Afterwards, the classifier's training data is augmented with this known unknown data, which, together with the defined loss function, results in a regularisation of the network by increasing the uncertainty for unknown input. The work presented in [56] follows a similar idea and introduces a new loss function, which is called the *entropic open-set loss*, to evenly distribute the probability across known classes if the input belongs to an unknown class. This loss function is employed in this chapter and therefore, the details are described in Section 8.3. The same applies for the approach called *deep open classification* described in [54], which replaces the final softmax layer by a layer of independent sigmoid functions and combines it with a new loss function. In addition to the entropic open-set loss, the work [56] introduces the *objectosphere loss* as an extension. The application in [56] is image classification and hence, a convolutional neural network specially designed for processing images is employed. The objectosphere loss targets the "deep feature layer" that exists in all convolutional neural networks, but not in the LSTM architecture of this thesis and therefore, the evaluations of this chapter are restricted to the entropic open-set loss.

Literature on recognising an unknown emitter or signal for an ELINT application is very sparse. Moreover, most of the existing papers either use methods that have not established themselves and were replaced by neural network types like LSTMs [57, 124, 125] or exhibit a lack of details on how training for the unknown

class is performed [57, 58]. In [59], the *class probability output network* as introduced in [126] is employed. It is a method for normalising the output of a classifier in order to obtain probability values, i.e. an alternative to using the softmax function.

8.1.2 Contributions

This chapter provides a thorough investigation of six approaches in several configurations to recognise unknown emitters based on the hierarchical emission model presented in Chapter 4. Four of the methods employ LSTMs and two are based on MCs. The LSTMs are implemented with three different loss functions, which are cross-entropy, entropic open set, and deep open classification. To the best of the author's knowledge, this is the first implementation of the entropic open-set loss with LSTMs in general and the first implementation of deep open classification with unidirectional LSTMs. Moreover, it is also the first application of LSTMs and MCs for the task of unknown emitter recognition.

In addition, an MC-based method for generating known unknown data is proposed. All considered classifiers for unknown emitter recognition are trained with five training cases, which differ in the contained known unknown data. An additional training case that resembles the conventional training for classification in terms of the closed-set assumption is employed for the LSTM with cross-entropy loss and the MC as described in Chapter 6. The evaluation is performed with ten different test cases, which consist of several combinations of unknown classes. It is investigated how the generated known unknown data influences the rejection accuracy for unknown input.

A compromise between the true and false rejection rate, as well as the accuracy for discriminating between known classes, needs to be found. This is achieved by selecting the configuration of the method, which consists of the training case as well as the value of the threshold δ , if a threshold is used. This chapter provides estimates on the expected performance for the distinction between known and unknown input when choosing the best configuration for the identification of known classes and vice versa. Moreover, this chapter investigates whether a single classifier is enough for a good accuracy for all metrics or a hierarchical combination of different classifiers is to be preferred. An evaluation with corrupted data additionally provides estimates on the robustness of the classifiers.

Section 8.2 describes the cases employed for training the approaches introduced in Section 8.3. Section 8.4 presents the results and Section 8.5 provides the summary.

8.2 Training Cases

Table 8.2 gives an overview of the different training cases defined for the classifiers considered in this chapter. Cases 0 and I contain all three example emitters as introduced in Section 4.3 as the known classes, while case 0 corresponds to the conventional training as employed in Chapter 6 and case I contains additional known

Table 8.2: Training cases employed for unknown emitter recognition.

		Training Case						
		Emitter	0	I	II	III	IV	V
known	QoS	x	x	x	x	x	x	x
	Rules-v1	x	x	x	x	x	x	x
	Rules-v2	x	x					
known unknown	UNKs		x	x			x	x
	Random		x	x			x	x
	QoS _{alt}					x	x	x
	Rules-v1 _{alt}					x	x	

unknown data. Cases II to V do not include the Rules-v2 radar in the set of known emitters because it is used as an unknown unknown for testing later on. Therefore, the dictionary used by the open-set recognition approach does not contain the symbols exclusively emitted by the Rules-v2 radar in these cases, which are about 12 % of its syllables and 38 % of its words (see Table 6.1 and Table 6.2 of Chapter 6). For simplicity, the unknowns are also referred to as “emitters” although the data might be artificially generated. Several combinations of known unknown emitters are defined in the different training cases. The generation of the corresponding data is described in the following section.

8.2.1 Generation of Known Unknown Emitters

There are three possibilities for the signal of an unknown emitter,

1. it only contains unknown symbols,
2. it contains known and unknown symbols,
3. it only contains known symbols.

In the first case, the input sequence to the open-set recognition method always consists of UNK symbols introduced by the word embedding (see Chapter 4) and is hence known in advance. Consequently, this case falls in the known unknown category and can be included in the training of the classifier. If a completely unknown signal is received, the symbol extraction step (see Figure 6.2 of Chapter 6) cannot connect it to any of the symbols in the global dictionary Ω^l at modelling level l and needs to assign new symbols to it. As these are not part of the dictionary, no vector representations exist and the word embedding layer of the LSTMs presented in Figure 6.3 maps them to the special symbol UNK. Also the MC approach fails to find the input string in

the dictionary and it therefore maps it to UNK. Consequently, a signal that does not contain any known symbols is always represented as a sequence consisting of UNK symbols only. In Table 8.2, it is referred to as “UNKs”. Of course, recognising a sequence of UNK symbols as unknown does not require a sophisticated method, however, it is included here for completeness.

Based on the data from the known unknown classes, the classifier is supposed to learn to distinguish between known and unknown input. If the unknown input is similar to the known classes, this task is more complicated. In the application considered in this thesis, the classifier should also be able to tell that an emitter is unknown if it uses the same symbols as the known classes but with different frequencies and agility. Therefore, known unknown emitters that make use of known symbols are needed. The most straightforward way to achieve this is to generate random sequences of known symbols. This approach is referred to as “Random” in Table 8.2. To make the known unknown data even more similar to the known classes, altered versions of the QoS and the Rules-v1 radar are created. The data of the Rules-v2 radar is not used here since it is employed as an unknown unknown emitter later on. In Table 8.2, the altered versions are called “QoS_{alt}” and “Rules-v1_{alt}”, respectively. The altered data is created from the MC emitter models developed in Chapter 5 by modifying the symbol transition matrices and sampling from the new distribution $\hat{p}_e^{alt}(\omega_j|\omega_i)$ with $\omega_i, \omega_j \in \Omega_e^l$ being symbols in the dictionary of emitter e . Each row of the transition matrix is altered by randomly choosing one of three different operations. The first one reverses the order of the n most probable entries, while n is chosen randomly in $\left[2, 3, \dots, \left\lfloor \frac{1}{2} |\Omega_e^l| \right\rfloor\right]$. For example, with $n = 2$, the second most probable next symbol is assigned the probability of the most probable symbol and vice versa. With $n = 3$, reversing the order of the entries equals swapping the first and the third entry. The second operation reweights the entries by adding random values between 0 and N , followed by a normalisation of the row. Due to the normalisation, the choice of N is irrelevant as long as $N > 0$. The third operation leaves the row unchanged. The transition matrices are modified 16 times per radar version and afterwards, data is sampled from the altered distributions to create the known unknowns.

8.3 Approaches

As described in the introduction, two alternative approaches can be applied to perform open-set recognition, which are employing a threshold or adding an extra unknown class. In this thesis, both approaches are investigated, as well as a classifier that only distinguishes between known and unknown, i.e. an anomaly detector. According to the classification given in Table 8.1, this approach is not a method for open-set recognition, but it can be turned into one in combination with a method that performs classification. An overview of the methods is given in Table 8.3, while the details are provided in the following sections. Since the identification accuracies for letters, commands, and functions are not satisfactory for the LSTMs even without unknown emitters (see Chapter 6), only syllables and words are considered here,

Table 8.3: Overview of the methods employed for unknown emitter recognition.
Rv1 = Rules-v1.

Method	Case	Dictionary	Thres.	Classes
LSTM - Cross-Entropy	0	Ω^l	yes	QoS, Rules-v1, Rules-v2
	I	Ω^l	no	QoS, Rules-v1, Rules-v2, unknown
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	no	QoS, Rules-v1, unknown
LSTM - Entropic Open Set	I	Ω^l	yes	QoS, Rules-v1, Rules-v2
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	yes	QoS, Rules-v1
LSTM - Deep Open Class.	I	Ω^l	yes	QoS, Rules-v1, Rules-v2
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	yes	QoS, Rules-v1
LSTM - Unknown Gate	I	Ω^l	no	known, unknown
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	no	known, unknown
MC	0	Ω^l	yes	QoS, Rules-v1, Rules-v2
	I	Ω^l	no	QoS, Rules-v1, Rules-v2, unknown
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	no	QoS, Rules-v1, unknown
MC - Unknown Gate	I	Ω^l	no	known, unknown
	II-V	$\Omega_{QoS}^l \cup \Omega_{Rv1}^l$	no	known, unknown

i.e. $l \in \{\text{syllables}, \text{words}\}$. For the same reason, only LSTM- and MC-based methods are examined and the dictionary lookup is not included.

The LSTM-based methods are similar to those described in Chapter 6 with the general architecture as shown in Figure 6.3. They are trained with a batch size of

$$\left\lfloor \frac{120}{|\mathbb{K} \cup \mathbb{V}|} \right\rfloor \cdot |\mathbb{K} \cup \mathbb{V}|, \quad (8.3)$$

where each batch is split into chunks of size $|\mathbb{K} \cup \mathbb{V}|$ such that several emitters are represented in each batch and on average, the same amount of data for each emitter is used during training (see Section 6.2.1). The Adam optimiser is employed with a learning rate of 0.0002, and training, validation, and test sets for the example emitters are the same as in the previous chapters. A basic batch size of 120 is chosen because it is a common factor of the number of simulation runs in each of these sets (see Chapter 5). By training several networks with different parameters, the number of layers and number of LSTM cells per layer are found. During training, checkpoints are created when the current validation loss is lower than all previous values. The final architecture then corresponds to the checkpoint with the lowest loss on the validation set. For all methods, also the MC-based approaches, the dictionary only contains the symbols from the known emitters and consequently also from the known unknown emitters as the data is generated from the symbols of the known emitters.

Table 8.4: Parameters of the LSTM architecture with cross-entropy loss.

Symbol	Parameter	Training Case					
		0	I	II	III	IV	V
Syllables	# layers	1	1	1	1	2	1
	# cells/layer	8	16	16	8	32	16
Words	# layers	1	1	1	1	1	1
	# cells/layer	4	8	8	16	16	8

8.3.1 Long Short-Term Memory with Cross-Entropy Loss

The LSTM using the cross-entropy loss defined by (3.68) is considered in two variants. The first one is equal to the network used for identification developed in Chapter 6, which corresponds to the training case 0 defined in Table 8.2. In this case, a threshold δ needs to be applied for rejection. The second one, described by the training cases I to V, contains an extra “unknown” class in the output. It is trained with the cross-entropy loss as well, while the data from the known unknown emitters is labelled to belong to the unknown class. Therefore, the training data consists of pairs $(\omega, \text{idx}(e))$ where $\omega \in \Omega^l$ is a symbol at modelling level l and $\text{idx}(e)$ is the class index c of the emitter $e \in \mathbb{K} \cup \mathbb{V}$. Here, $\text{idx}(\text{QoS}) = 1$ and $\text{idx}(\text{Rules-v1}) = 2$. In training case 0 and I, $\text{idx}(\text{Rules-v2}) = 3$ and $\text{idx}(e) = 4 \forall e \in \mathbb{V}$. In the other training cases, $\text{idx}(e) = 3 \forall e \in \mathbb{V}$.

Section 6.2.1 presents the architectures and training procedure with different sequence lengths employed for the LSTMs in the training case 0. For syllables, the $\text{LSTM}_{\text{scen}}$ trained on the complete scenarios is shown to provide the best results and is hence used here. For words, the $\text{LSTM}_{\text{rand}}$, which is trained with random sequence lengths, achieves the highest accuracies with ideal data. However, it is not as robust with respect to corrupted data as the $\text{LSTM}_{\text{scen}}$. The $\text{LSTM}_{\text{rand}}$ is nevertheless employed for words in this chapter because the conducted investigations serve as a feasibility study for recognising a very similar emitter, like the Rules-v2 radar, as unknown and therefore require high accuracies with ideal data. Table 8.4 lists the parameters of the LSTMs in the different training cases, while the parameters for training case 0 are repeated here for convenience.

8.3.2 Long Short-Term Memory with Entropic Open-Set Loss

The authors of [56] introduce the entropic open-set loss for training a classifier that can recognise if something is unknown. This new loss function forces the classifier to uniformly distribute the class probabilities for unknown input such that it is possible

Table 8.5: Parameters of the LSTM architecture with entropic open-set loss.

Symbol	Parameter	Training Case				
		I	II	III	IV	V
Syllables	# layers	1	1	1	1	2
	# cells/layer	16	16	16	16	16
Words	# layers	1	1	1	1	1
	# cells/layer	8	8	32	8	8

to define a threshold on the confidence for rejection. The function of the entropic open-set loss J_E is given by

$$J_E(\mathbf{x}) = \begin{cases} -\log \hat{y}_c & \text{if } \text{class}(\mathbf{x}) = k_c \in \mathbb{K}, \\ -\frac{1}{|\mathbb{K}|} \sum_{k_c \in \mathbb{K}} \log \hat{y}_c & \text{if } \text{class}(\mathbf{x}) \in \mathbb{V}, \end{cases} \quad (8.4)$$

where \hat{y}_c is the estimated probability value obtained by the softmax function for class $k_c \in \mathbb{K}$ as defined in (3.66). For an input \mathbf{x} with $\text{class}(\mathbf{x}) = k_c \in \mathbb{K}$, the entropic open-set loss is equal to the cross-entropy loss as defined in (3.68). Hence, it is not trained with training case 0. Table 8.5 lists the parameters of the architecture for the cases I to V.

8.3.3 Long Short-Term Memory with Deep Open Classification Loss

Usually, the logits $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_C)$ of a neural network are normalised by the softmax function as defined in (3.66) such that the output can be interpreted as a probability distribution. By this definition, the network is forced to output a probability greater than 0 for at least one class. However, if the input does not belong to any of the known classes, the correct output might be a probability of 0 for all the known classes. Therefore, [54] replaces the softmax layer by a layer of independent sigmoids $\sigma(\tilde{y}_c)$ $\forall c \in \{1, \dots, C\}$ as given by (3.60). In this approach, the probability assigned to each known class k_c is considered individually and therefore, the probabilities of all classes do not need to sum to 1. The corresponding loss function J_{DOC} for training the network is defined by

$$J_{DOC}(\mathbf{x}) = \sum_{k_c \in \mathbb{K}} -\mathbb{1}_{k_c}(\text{class}(\mathbf{x})) \log(\sigma(\tilde{y}_c)) - (1 - \mathbb{1}_{k_c}(\text{class}(\mathbf{x}))) \log(1 - \sigma(\tilde{y}_c)), \quad (8.5)$$

where $\mathbb{1}_a(x)$ is a variation of the indicator function from (3.79) to determine equality instead of set membership,

$$\mathbb{1}_a(x) = \begin{cases} 1 & \text{if } x = a, \\ 0 & \text{otherwise.} \end{cases} \quad (8.6)$$

Table 8.6: Parameters of the LSTM architecture with deep open classification loss.

Symbol	Parameter	Training Case				
		I	II	III	IV	V
Syllables	# layers	2	1	2	1	1
	# cells/layer	64	64	16	64	128
Words	# layers	1	1	1	1	1
	# cells/layer	256	256	512	512	512

This loss function corresponds to the binary cross-entropy loss with the result of the indicator function $\mathbb{1}_{k_c}(\text{class}(x))$ as the label. For an input x with $\text{class}(x) \in \mathbb{V}$, the desired output of the network consists of all 0. To reject an input as unknown, a threshold needs to be defined.

Unfortunately, the LSTMs are hard to train using the deep open classification loss in some cases, as will be seen as well in the evaluation below. For example, the best validation loss for training case I with syllables was achieved directly after random initialisation of the weights and diverged afterwards. No architecture could be found that solved the problem. Moreover, the mean probability assigned to the correct class is below 0.1 during testing. Table 8.6 provides details on the number of layers and number of LSTM cells per layer for each training case. The networks are much larger than with the other architectures. Also the LSTMs with deep open classification loss are not trained with case 0 since it is specially designed to handle unknown input and the experience with the complicated training suggests that it would not provide better results than the LSTMs with cross-entropy loss for data without unknown emitters.

8.3.4 Long Short-Term Memory as Unknown Gate

Adding an extra output class might decrease the accuracy on the known emitters. Therefore, a classifier that only distinguishes between known and unknown is trained.

Table 8.7: Parameters of the LSTM architecture employed as unknown gate.

Symbol	Parameter	Training Case				
		I	II	III	IV	V
Syllables	# layers	1	1	1	1	1
	# cells/layer	16	16	8	16	16
Words	# layers	1	1	1	1	1
	# cells/layer	4	8	16	32	8

It can act as a gate that only passes the known data to the classifier that distinguishes between different known classes. The LSTM-based unknown gate is trained with the cross-entropy loss, while the training data consists of pairs $(\omega, \text{idx}(e))$ with $\text{idx}(e) = 1$ if $e \in \mathbb{K}$ and $\text{idx}(e) = 2$ if $e \in \mathbb{V}$. It is only trained with the cases I to V since case 0 does not contain any known unknown emitters and hence $\text{idx}(e) = 1$ for the complete training data. Table 8.7 provides the number of layers and the number of LSTM cells per layer in the architecture of the unknown gate.

8.3.5 Markov Chain

Like in the previous chapters, an MC-based approach is implemented in comparison to the neural networks. The basic principle is the same as described in Section 6.2.2 and the MC trained with case 0 is equal to the approach from Section 6.2.2. In this training case, a threshold δ is applied to reject an input as unknown. For the other training cases, the output classes of the MCs are extended to contain an “unknown” class, like for the LSTMs with cross-entropy loss. For every known emitter $e \in \mathbb{K}$, the transition matrix $\hat{P}_e(\omega_j|\omega_i)$ that describes the probability that symbol ω_j is the next symbol having observed ω_i , is estimated as defined in (5.3). Based on this matrix, the probability that the emitter generated the input sequence $\bar{\omega} = \omega_1\omega_2\ldots\omega_{|\bar{\omega}|}$ is determined by Bayes’ rule, as given in (6.3) and repeated here for convenience,

$$\hat{P}(e_i|\bar{\omega}) = \frac{\hat{P}(\bar{\omega}|e_i) \cdot \hat{P}(e_i)}{\hat{P}(\bar{\omega})}. \quad (8.7)$$

The probability of the unknown class is obtained in the same way,

$$\hat{P}(\emptyset|\bar{\omega}) = \frac{\hat{P}(\bar{\omega}|\emptyset) \cdot \hat{P}(\emptyset)}{\hat{P}(\bar{\omega})}. \quad (8.8)$$

Here, $\hat{P}(\bar{\omega}|\emptyset)$ needs to be estimated based on the data from the known unknown emitters. A common symbol transition matrix is learnt for the known unknown emitters because calculating $\sum_{e \in \mathbb{V}} \hat{P}(\bar{\omega}|e)$ to obtain $\hat{P}(\bar{\omega}|\emptyset)$ would result in a probability of 0 for most of the unknown unknown emitters since they did not actually generate the data. With a common transition matrix, the data of the unknown unknown emitter does not need to specifically match one of the known unknown emitters, but only the common distribution, which is what is desired. The different training cases hence influence the estimated unknown distribution. The probabilities $\hat{P}(\emptyset)$ and $\hat{P}(e)$ for $e \in \mathbb{K}$ are set to $\frac{1}{|\mathbb{K}|+1}$. Note that obtaining $\hat{P}(\emptyset|\bar{\omega})$ by $1 - \sum_{e \in \mathbb{K}} \hat{P}(e|\bar{\omega})$ without estimating $\hat{P}(\bar{\omega}|\emptyset)$ is not possible since $\hat{P}(\bar{\omega})$ is computed by marginalisation

$$\hat{P}(\bar{\omega}) = \hat{P}(\bar{\omega}|\emptyset) + \sum_{e \in \mathbb{K}} \hat{P}(\bar{\omega}|e). \quad (8.9)$$

8.3.6 Markov Chain as Unknown Gate

Analogous to the LSTM employed as an unknown gate, an MC is learnt for the same purpose. The probability for the unknown class is determined as described above,

while the probability for the known class is calculated by summing the probabilities of the known emitters,

$$\hat{P}(\text{known}|\bar{\omega}) = \sum_{e \in \mathbb{K}} \hat{P}(e|\bar{\omega}). \quad (8.10)$$

Therefore, the MC is basically identical to the one described in the section before, except for the summation of the known emitters' probabilities.

8.4 Experimental Results

The classifiers are tested with several combinations of known unknown and unknown unknown emitters. The test cases 1a to 5a for the classifiers trained with the cases 0 and I are displayed in Table 8.8 and the test cases 1b to 5b for the other classifiers are given in Table 8.9. The difference is in the role of the Rules-v2 radar, which belongs to the known classes in the test cases 'a' and to the unknown unknown emitters in the test cases 'b'. Test case 2a corresponds to the evaluation performed in Chapter 6 and is included to see whether training with an extra unknown class decreases the performance in a scenario without unknown emitters. The only known unknown emitter contained in the test cases is the sequence consisting only of the UNK symbol since this is expected to actually appear during deployment of the system, in contrast to the other known unknown emitters. In addition to the Rules-v2 radar, four other unknown unknown emitters are employed, which are artificially

Table 8.8: Test cases for classifiers trained with cases 0 or I.

		Test Case				
	Emitter	1a	2a	3a	4a	5a
known	QoS	x	x	x	x	x
	Rules-v1	x	x	x	x	x
	Rules-v2	x	x	x	x	x
known unk	UNKs	x		x	x	x
unknown unknown	Unk-1			x		x
	Unk-2			x		x
	Unk-3				x	x
	Unk-4				x	x

Table 8.9: Test cases for classifiers trained with cases II to V.

		Test Case				
	Emitter	1b	2b	3b	4b	5b
known	QoS	x	x	x	x	x
	Rules-v1	x	x	x	x	x
known unk	UNKs	x		x	x	x
unknown unknown	Rules-v2	x	x	x	x	x
	Unk-1			x		x
	Unk-2			x		x
	Unk-3				x	x
	Unk-4				x	x

generated. They make use of known and unknown syllables and words, while emitters that employ more than one symbol randomly switch between them. The four emitters are defined by:

- Unk-1: One known word consisting of one known syllable, $w_{26101507} = (s_{40338}, s_{40338}, s_{40338}, s_{40338})$.
- Unk-2: One known word consisting of eight known syllables, $w_{59399} = (s_{113}, s_{354}, s_{595}, s_{836}, s_{1077}, s_{1318}, s_{1559}, s_{1800})$.
- Unk-3: Two known words $w_{24501141} = (s_{37607}, s_{37607}, s_{37607}, s_{37607})$ and $w_{6749} = (s_1, s_{242}, s_{483})$ consisting of one or three known syllables and one unknown word $w_{unk1} = s_{unk1}$ consisting of one unknown syllable.
- Unk-4: One known word $w_{24901379} = (s_{38290}, s_{38290}, s_{38290}, s_{38290})$ consisting of one known syllable and two unknown words $w_{unk2} = (s_1, s_{499}, s_{772})$, $w_{unk3} = (s_{346}, s_{595}, s_{828}, s_{1559}, s_{1792})$ consisting of three or five known syllables.

Note that the Unk-1 emitter makes use of the most common word in the data of the known radars, and hence their syllables.

The methods that need a threshold for accepting the current input (see Table 8.3) are evaluated with the values $\delta \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. The minimum is chosen to be 0.4 as with three known emitters, a lower confidence value comes close to random guessing. For the methods with an extra unknown class in the output, the threshold can be interpreted as being equal to 0.0. Therefore, each approach exists in several configurations, which consist of a training case and an acceptance threshold. As it is done in Chapter 6, the methods are tested at different sequence lengths in the set $\mathcal{S} = \{1, 10, 50, 100, 200, 400, 600, 800, 1000, 1200, 1400\}$. Two different evaluation metrics are employed because in several test cases there are more unknown emitters than known ones, resulting in a high overall accuracy by rejecting every input as unknown. The metrics are called the *distinction* and the *identification accuracy*. The distinction accuracy acc_{dist} is defined as the mean accuracy of distinguishing between known and unknown, while ignoring confusions between known emitters. It is composed of two parts, which are called the *acceptance accuracy* acc_{acpt} and the *rejection accuracy* acc_{rej} , with

$$\text{acc}_{acpt}(s) = \frac{1}{|\mathbb{K}|} \sum_{e_i \in \mathbb{K}} \sum_{e_j \in \mathbb{K}} \text{acc}(\hat{Y}_s^{e_i}, Y_{e_j}, k), \quad k = 1, \quad (8.11)$$

and

$$\text{acc}_{rej}(s) = \frac{1}{|\mathbb{V} \cup \mathbb{U}|} \sum_{e \in \mathbb{V} \cup \mathbb{U}} \text{acc}(\hat{Y}_s^e, Y_{unk}, k), \quad k = 1. \quad (8.12)$$

Here, \hat{Y}_s^e corresponds to the output of the classifier for the data of emitter e after a sequence of s symbols and Y_e is the set of labels containing only $\text{idx}(e)$, while Y_{unk}

is the set of labels only containing the index of the unknown class. The top k accuracy $\text{acc}(\hat{Y}, Y, k)$ is given by (3.78). The distinction accuracy is then defined as

$$\text{acc}_{\text{dist}}(s) = \frac{1}{2} (\text{acc}_{\text{apt}}(s) + \text{acc}_{\text{rej}}(s)). \quad (8.13)$$

The identification accuracy acc_{id} is given by the mean classification accuracy for the emitters in the known classes,

$$\text{acc}_{\text{id}}(s) = \frac{1}{|\mathbb{K}|} \sum_{e \in \mathbb{K}} \text{acc}(\hat{Y}_s^e, Y_e, k), \quad k = 1. \quad (8.14)$$

8.4.1 Evaluation Under Ideal Conditions

Distinction Accuracy. Table 8.10 provides an overview of each method's configurations that achieve the highest mean distinction accuracy. These are obtained by averaging the accuracies for each configuration over the test cases 1a to 5a and 1b to 5b, respectively, and choosing the configuration with the best performance. The sequence lengths of 1, 600, and 1400 symbols are exemplarily shown. For most of the methods using a threshold, a higher value of $\delta = 0.5$ is beneficial to distinguish between known and unknown input (see also Figure 8.1). The majority of the approaches achieves the best results using the same configuration with the sequence lengths of 600 and 1400 symbols. Only in two cases the best configurations are different for these sequence

Table 8.10: Configurations (training case, δ) that achieve the highest distinction accuracies, averaged over the test cases 1a to 5a and 1b to 5b, respectively.

		Syllable			Word			
		Sequence length			Sequence length			
Method		1	600	1400	1	600	1400	
Test cases 'a'	LSTM	Cross-Entropy	(I, 0.0)	(0, 0.5)	(0, 0.5)	(I, 0.0)	(I, 0.0)	(I, 0.0)
		Entropic Open Set	(I, 0.5)	(I, 0.5)	(I, 0.5)	(I, 0.5)	(I, 0.5)	(I, 0.5)
		Deep Open Class.	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.5)	(I, 0.5)	(I, 0.4)
		Unknown Gate	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)
	MC	MC	(0, 0.4)	(0, 0.4)	(0, 0.5)	(0, 0.4)	(0, 0.5)	(0, 0.5)
		Unknown Gate	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)	(I, 0.0)
Test cases 'b'	LSTM	Cross-Entropy	(IV, 0.0)	(IV, 0.0)	(IV, 0.0)	(II, 0.0)	(III, 0.0)	(III, 0.0)
		Entropic Open Set	(IV, 0.6)	(IV, 0.8)	(IV, 0.7)	(II, 0.6)	(IV, 0.8)	(IV, 0.8)
		Deep Open Class.	(IV, 0.4)	(IV, 0.4)	(IV, 0.4)	(II, 0.4)	(IV, 0.6)	(II, 0.5)
		Unknown Gate	(II, 0.0)	(IV, 0.0)	(IV, 0.0)	(V, 0.0)	(V, 0.0)	(V, 0.0)
	MC	MC	(III, 0.0)	(IV, 0.0)	(IV, 0.0)	(II, 0.0)	(IV, 0.0)	(IV, 0.0)
		Unknown Gate	(III, 0.0)	(IV, 0.0)	(IV, 0.0)	(II, 0.0)	(IV, 0.0)	(IV, 0.0)

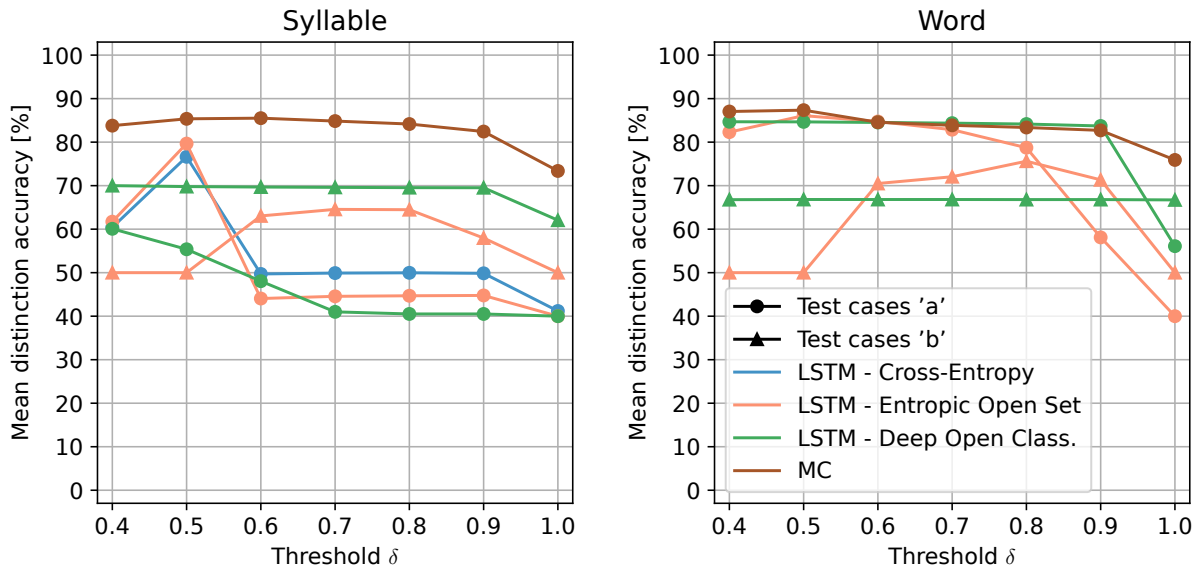


Figure 8.1: Dependency of the distinction accuracy on the threshold at a sequence length of 1400 symbols [65].

lengths, both for syllables and words. For syllables, this concerns the MC with test cases 'a' and the LSTM with entropic open-set loss with test cases 'b'. For words, it applies to the best configurations of the LSTM with deep open classification loss, both in cases 'a' and 'b'. For syllables, the best configuration for a sequence length of one symbol differs from the configuration with 600 symbols in five out of twelve cases. For words, it is different in six scenarios, which corresponds to 50 % of the cases. The LSTM with cross-entropy loss and the MC are the only methods that are trained with case 0. For both, the best results in the test cases 'a' are most often obtained with training case 0 when using syllables. For words, however, the training case I works best for the LSTM with cross-entropy loss but not for the MC.

For the test cases 1b to 5b, the highest distinction accuracies with syllables are most often achieved with training case IV. This shows that to distinguish between known and unknown, training with all known unknown emitters provides an advantage. The results are not as clear for words as for syllables. However, four of the six methods provide the best results with training case IV at a sequence length of 600 words and three methods at 1400 words. The LSTM with cross-entropy loss achieves the highest accuracies with training case III for words, which only contains the altered version of the Rules-v1 and the QoS radar as known unknowns, but not the UNKs or random sequences. Still, it is able to reject the UNKs sequence as shown in Figure 8.10 below. The LSTM trained with entropic open-set loss applies high thresholds of up to 0.8. The relation between threshold and distinction accuracy is shown in Figure 8.1 for a sequence length of 1400 symbols. Only the results of the methods with a best configuration that employs a threshold are depicted for the training case of that best configuration. As is seen, the variation of the accuracy with respect to the selected threshold depends very much on the method.

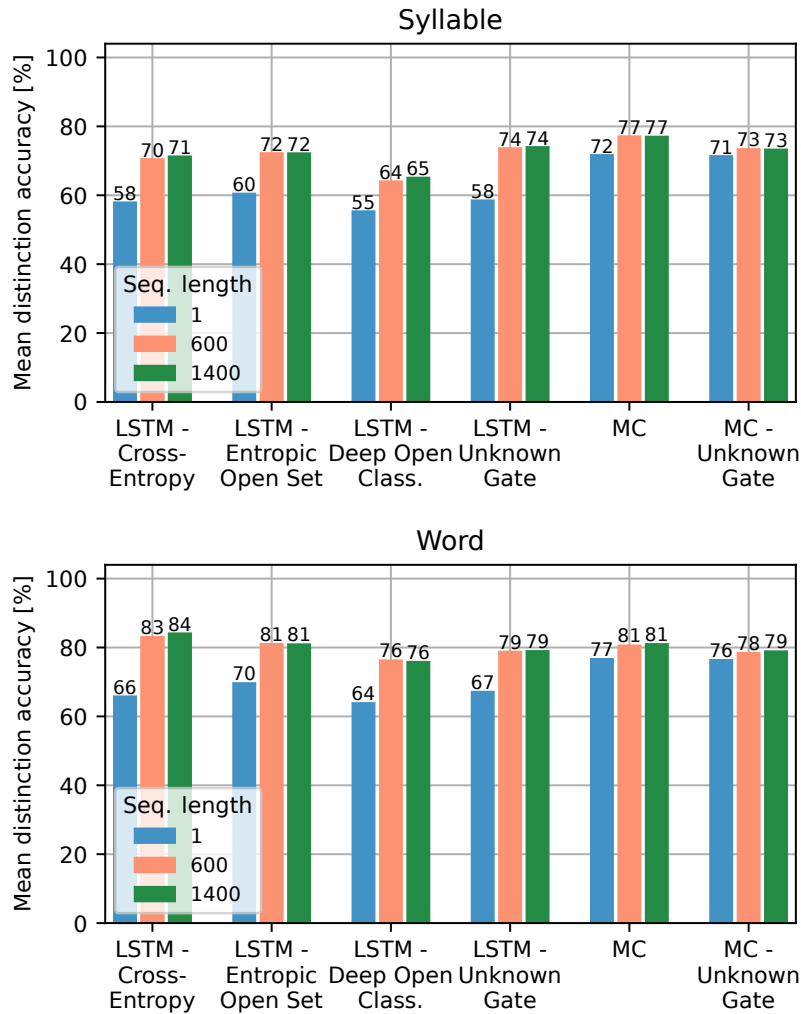


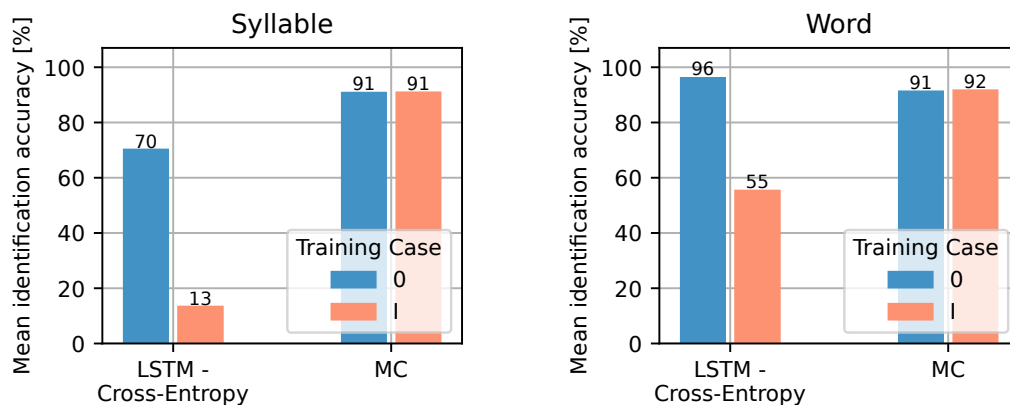
Figure 8.2: Mean distinction accuracies of the best configurations [65].

Figure 8.2 presents the mean distinction accuracies of the best configurations, averaged over all test cases 1a to 5a and 1b to 5b. As also observed in Chapter 6, the MC outperforms all other methods for syllables, especially for a sequence length of only one symbol. However, the LSTM-based unknown gate provides higher accuracies than the MC-based unknown gate with longer sequence lengths. For words, the MC provides the highest accuracies with only one symbol, but the MC-based unknown gate comes very close. With increasing sequence lengths, the LSTM with cross-entropy loss is the best method on average. The LSTM with entropic open-set loss and the MC achieve about the same results. Also the LSTM-based and the MC-based unknown gate provide very similar accuracies. The LSTM with deep open classification loss achieves slightly lower accuracies than the other methods. The values for the distinction accuracies per test case are provided in Tables A.7 to A.10 of Appendix A.3.

Overall, the distinction accuracies are higher when using words. The reason is that the unknown emitters have less words than syllables in common with the known emitters, which shows the benefit of the hierarchical emission model.

Table 8.11: Configurations (training case, δ) that achieve the highest identification accuracies, averaged over the test cases 1a to 5a and 1b to 5b, respectively.

		Syllable Sequence length			Word Sequence length		
Method		1	600	1400	1	600	1400
Test cases 'a'	LSTM	Cross-Entropy	(0, 0.4)	(0, 0.4)	(0, 0.4)	(0, 0.4)	(0, 0.4)
		Entropic Open Set	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)
		Deep Open Class.	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)
	MC	(0, 0.6)	(I, 0.0)	(I, 0.0)	(0, 0.4)	(I, 0.0)	(I, 0.0)
Test cases 'b'	LSTM	Cross-Entropy	(III, 0.0)	(III, 0.0)	(II, 0.0)	(II, 0.0)	(II, 0.0)
		Entropic Open Set	(II, 0.4)	(II, 0.4)	(II, 0.4)	(IV, 0.4)	(IV, 0.4)
		Deep Open Class.	(II, 0.4)	(IV, 0.4)	(II, 0.4)	(II, 0.4)	(II, 0.4)
	MC	(II, 0.0)	(II, 0.0)	(II, 0.0)	(II, 0.0)	(II, 0.0)	(II, 0.0)

**Figure 8.3:** Comparison of the mean identification accuracies in the test cases 'a' with training case 0 or I at a sequence length of 1400 symbols [65].

Identification Accuracy. Table 8.11 depicts the configurations that provide the best identification accuracies. In contrast to the distinction accuracy for syllables, the identification accuracy is highest with training cases that do not contain all known unknown emitters in the test cases 'b' for most of the methods. Especially for the LSTM with cross-entropy loss, the training case 0, which is the conventional training, provides the highest identification accuracies. A comparison between the accuracies of the training cases 0 and I is shown in Figure 8.3. Note that these training cases are only employed in the test cases 'a' and therefore, the results in this figure are different from the mean over all test cases 'a' and 'b' presented in Figure 8.5 below.

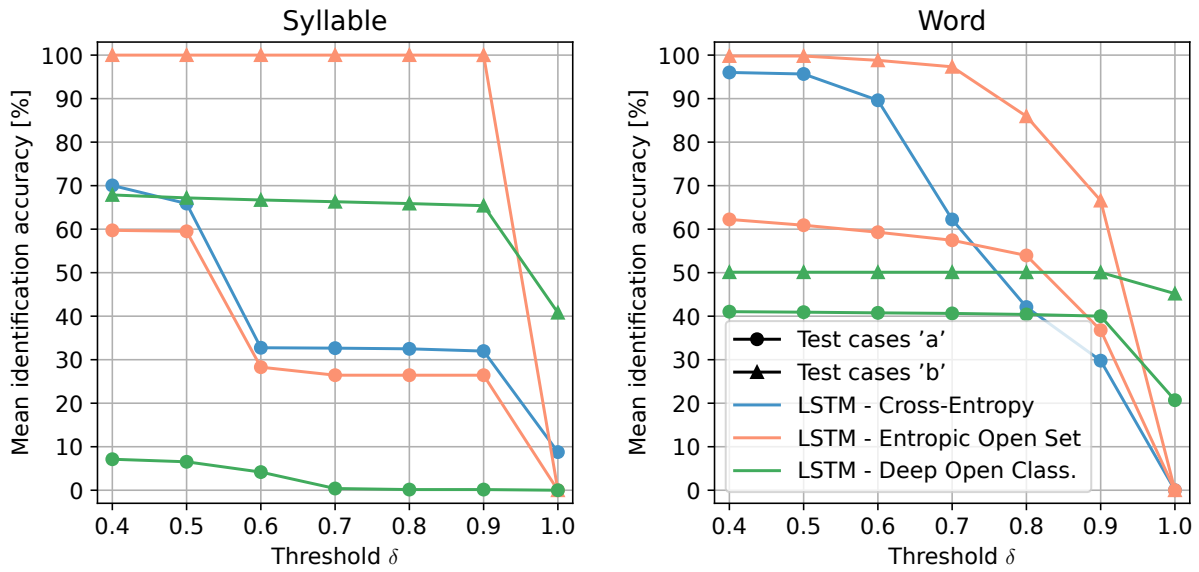


Figure 8.4: Dependency of the identification accuracy on the threshold at a sequence length of 1400 symbols [65].

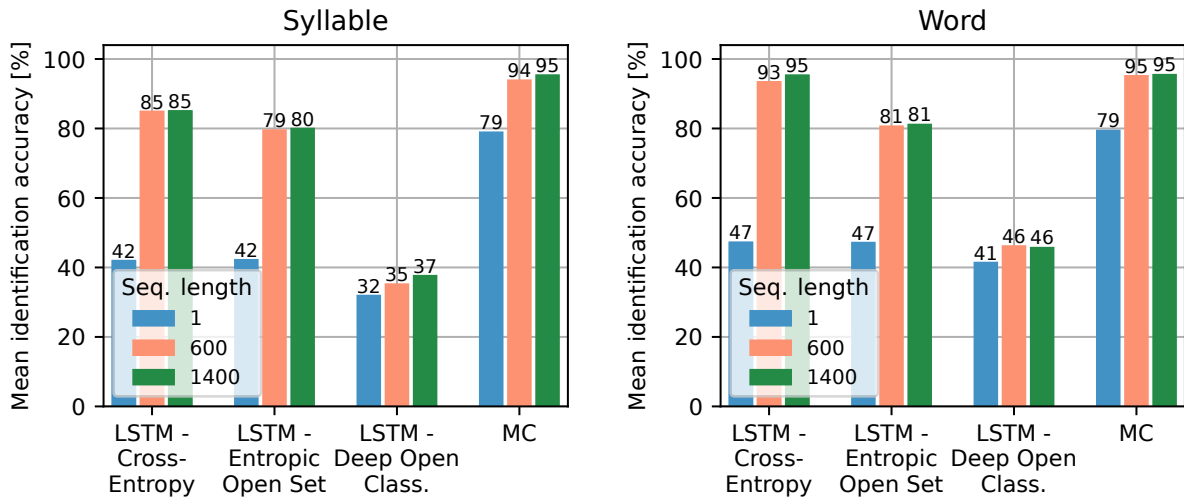


Figure 8.5: Mean identification accuracies of the best configurations [65].

With only one exception, all methods that apply a threshold for rejection use the smallest value of 0.4 for a high identification accuracy, which reduces the false rejection rate. Figure 8.4 shows the dependency of the identification accuracy on the threshold at a sequence length of 1400 symbols. The training case that is part of the best configuration is depicted for each method that applies a threshold. Most of the methods reject every input with $\delta = 1.0$. Especially the LSTM with entropic open-set loss shows a distinctive behaviour for syllables in the test cases 'b'. Both the QoS and

the Rules-v1 radar are correctly identified with a confidence of at least 0.9, but they are rejected as unknown with a threshold of 1.0.

For the identification accuracy, the MC achieves much higher results than all other approaches (see Figure 8.5) when considering syllables. The difference is even bigger than observed for the distinction accuracy. For words, the LSTM with cross-entropy loss is the only method that achieves about the same accuracy as the MC at a sequence length of 1400 words. In the other cases, the MC significantly outperforms all other methods, especially for a sequence length of only one symbol. This confirms the results from Chapter 6. Both methods clearly outperform the other two LSTM-based approaches. The LSTM with deep open classification loss achieves much lower accuracies than the other methods. The values for the identification accuracies per test case are provided in Tables A.11 to A.14 of Appendix A.3.

Rejection and Acceptance Accuracy. A high rejection accuracy might be accompanied by a low acceptance accuracy and vice versa. Figure 8.6 depicts these accuracies for each method and each test case at a sequence length of 1400 symbols. In an ELINT application, there is no need for a short reaction time and hence, longer sequences can be processed before making a decision. The exact values are provided in Tables A.7 to A.10 of Appendix A.3. Test case 2a does not contain any unknown emitters, therefore its rejection accuracy is set to 100 %. It is clearly seen from the figure that most of the methods provide an acceptance accuracy of more than 90 % for syllables and more than 80 % for words, while the rejection accuracy varies greatly between the test cases. The LSTM with deep open classification loss, however, tends to reject nearly every input sequence for the test cases ‘a’ when using syllables and also achieves a lower acceptance accuracy than the other methods in the test cases ‘b’. Nevertheless,

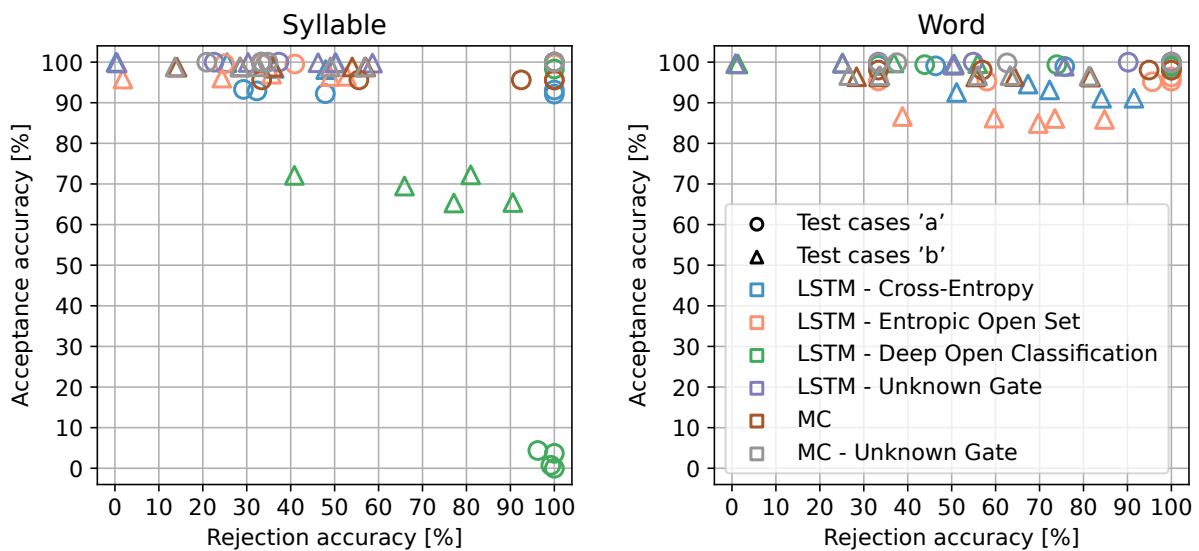


Figure 8.6: Rejection and acceptance accuracies for the different test cases at a sequence length of 1400 symbols [65].

it achieves much higher rejection accuracies for syllables. As some of the unknown emitters are very similar to the known ones, correctly rejecting the unknown emitters also results in a false rejection of some of the known sequences. The LSTM with entropic open-set loss applies a high threshold of 0.8 in the test cases 'b' for words, which leads to a lower acceptance accuracy. However, it is seen that more than 80 % of the input sequences are accepted with a value greater or equal 0.8. Overall, higher rejection accuracies are achieved with words than with syllables. For the majority of the test cases and methods, the rejection accuracies are above 25 % with words, while the majority lies between 10 % to 60 % for syllables.

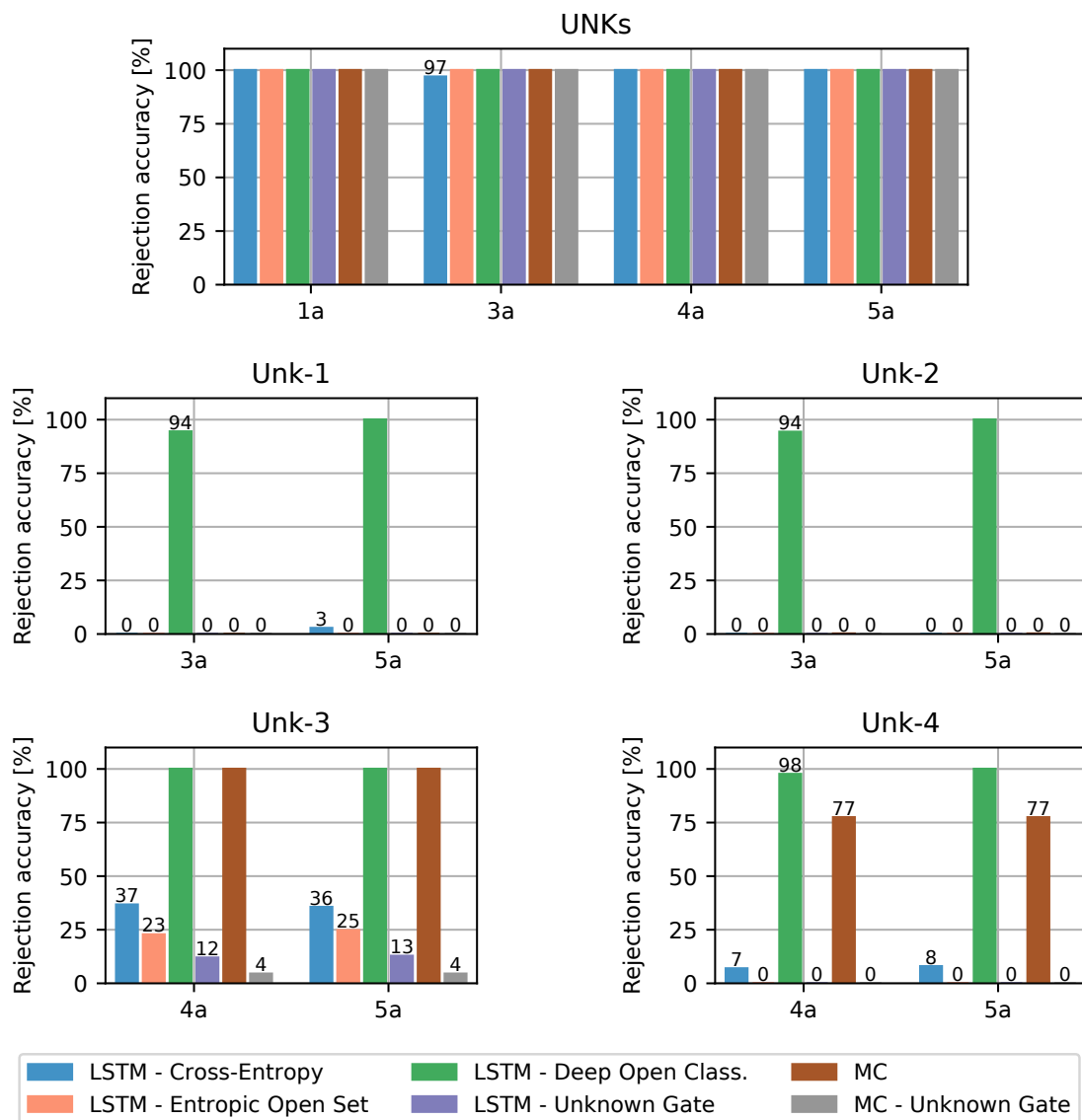


Figure 8.7: Rejection accuracies for the unknown emitters in the test cases 'a' at a sequence length of 1400 syllables [65].

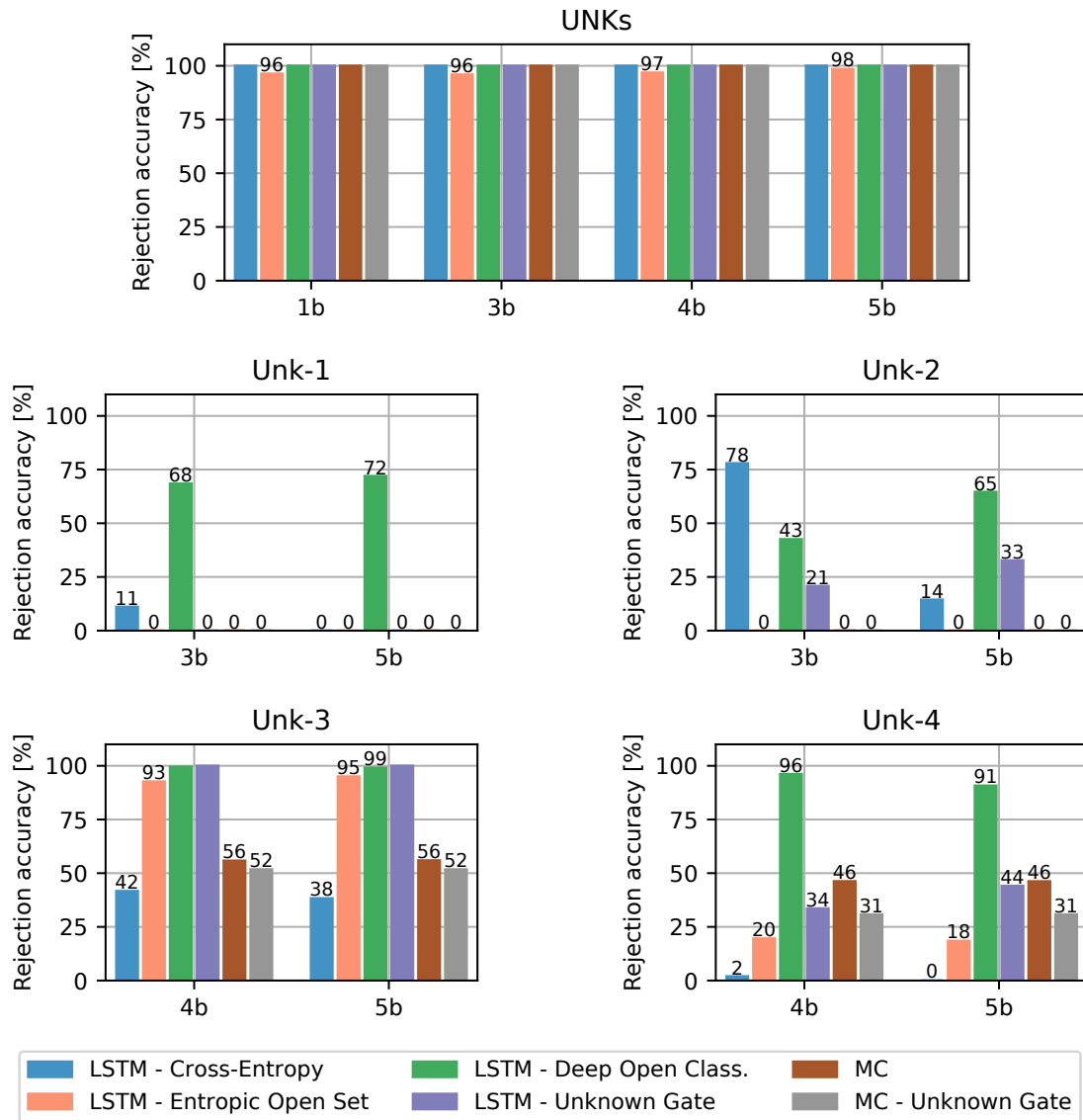


Figure 8.8: Rejection accuracies for the emitters UNKs, Unk-1, Unk-2, Unk-3, and Unk-4 in the test cases ‘b’ at a sequence length of 1400 syllables [65].

Rejection Accuracies for the Unknown Emitters. Figure 8.7 shows the rejection accuracies with syllables individually for each unknown emitter in the test cases ‘a’ and Figure 8.8 provides the same metrics for the test cases ‘b’. This excludes the Rules-v2 radar, for which the results are shown separately in Figure 8.11 (top). The figures consider a sequence length of 1400 syllables and the configurations that work best for the distinction accuracy. The bars without a label represent an accuracy of 100 %. All methods reliably reject the UNKs sequences with at least 96 % accuracy in all test cases. As the LSTM with deep open classification loss classifies nearly every input sequence as unknown in the test cases ‘a’, it provides a high rejection accuracy for all unknown emitters. For the Unk-3 and the Unk-4 emitter, the MC achieves high accuracies as well. The Unk-1 and the Unk-2 emitter, however, are not rejected. Both

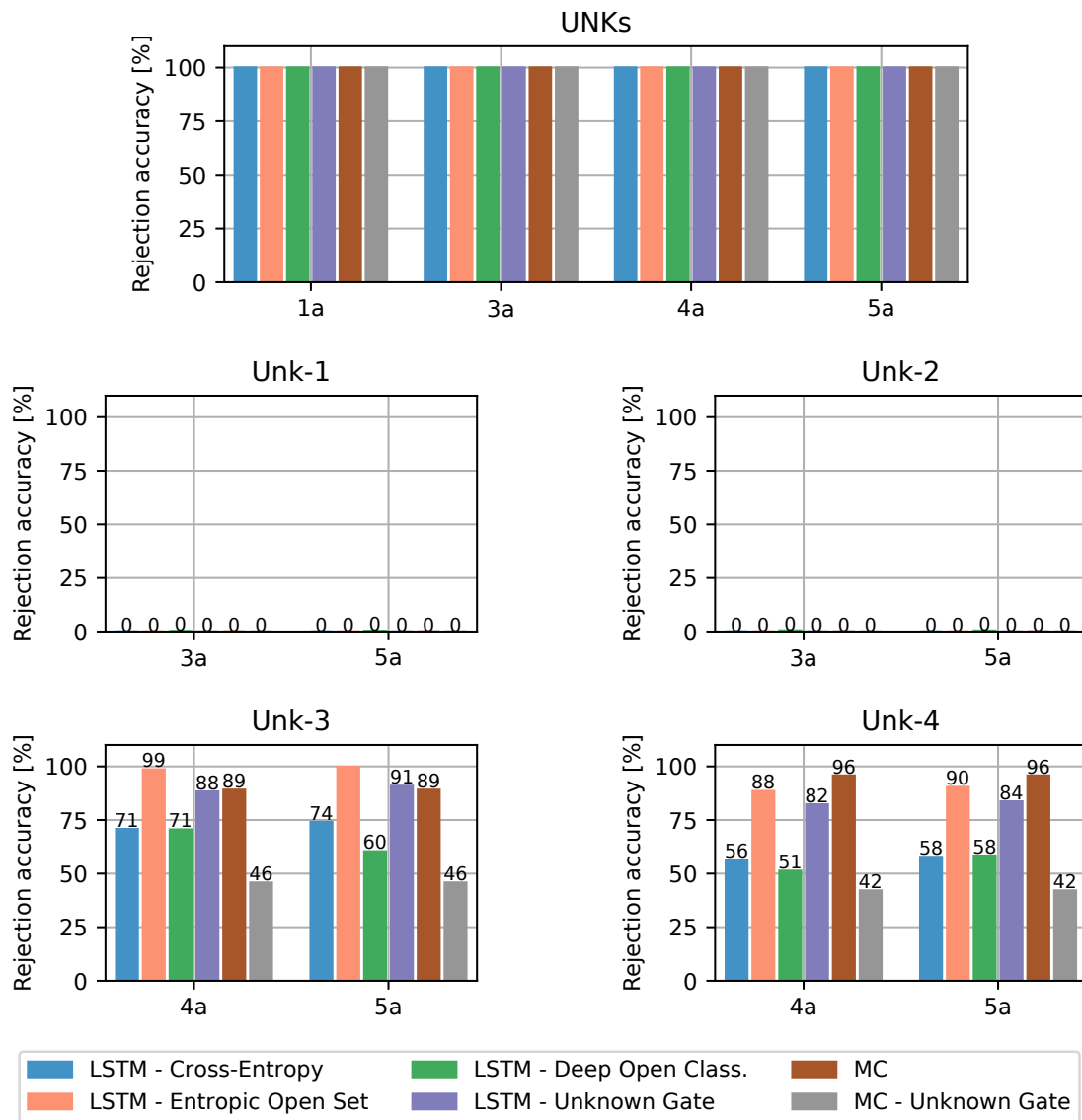


Figure 8.9: Rejection accuracies for the unknown emitters in the test cases ‘a’ at a sequence length of 1400 words [65].

make only use of known syllables, while the Unk-3 and the Unk-4 emitter also emit unknown syllables, which makes the rejection easier.

As seen in Figure 8.8, the rejection accuracies are much higher in the test cases ‘b’ than in the cases ‘a’. Only the accuracies of the MC significantly decrease. The Rules-v2 emitter is considered as an unknown unknown here, which helps the LSTM-based approaches to distinguish known from unknown. The rejection accuracies for the Rules-v2 radar, however, are low (see Figure 8.11, top). Only the LSTM with deep open classification loss achieves accuracies of more than 40 %, while wrongly rejecting at least 25 % of the known sequences.

Figures 8.9 and 8.10 provide the individual rejection accuracies with words of the emitters UNKs, Unk-1, Unk-2, Unk-3, and Unk-4 per method for the test cases ‘a’

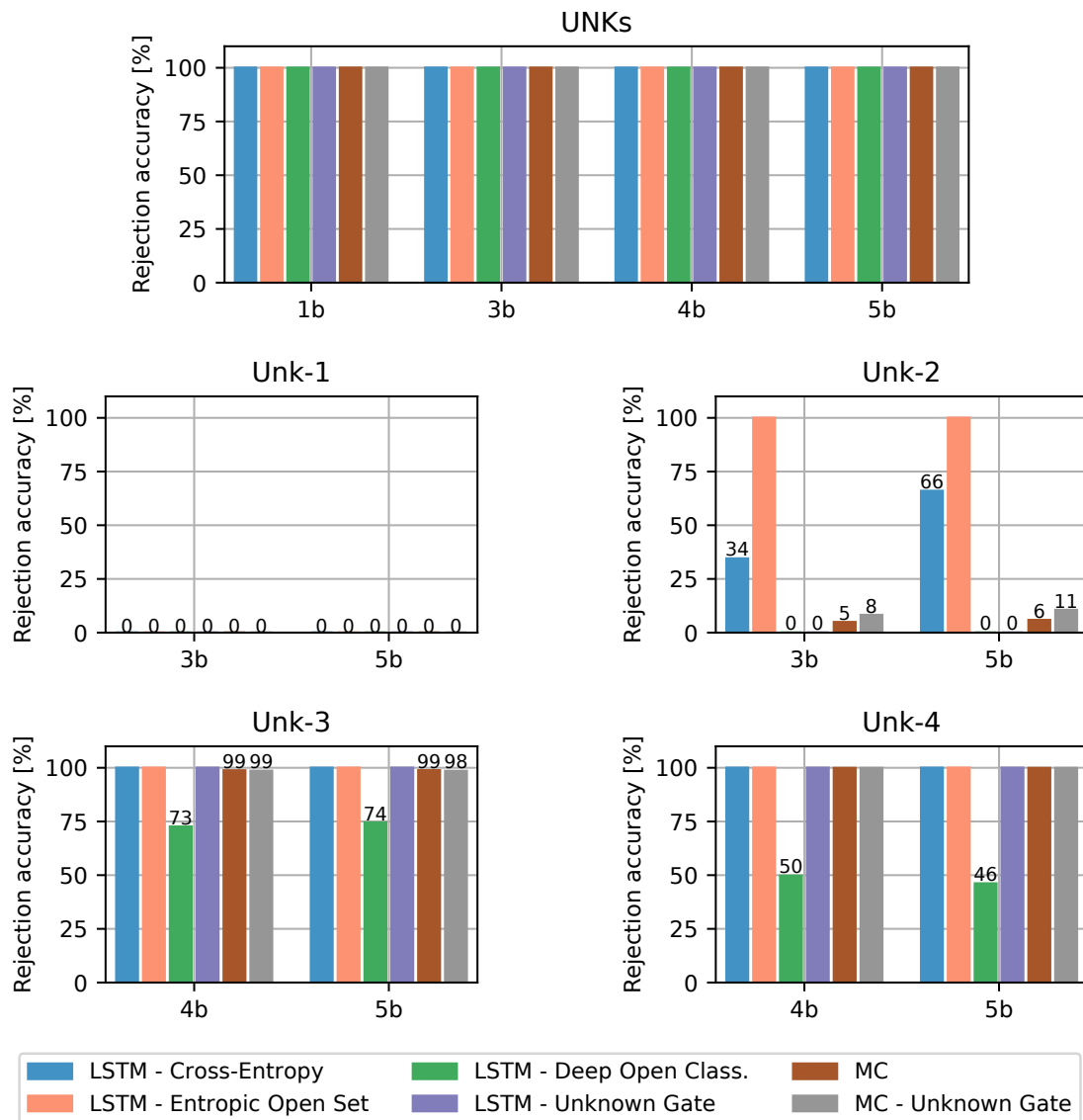


Figure 8.10: Rejection accuracies for the emitters UNKs, Unk-1, Unk-2, Unk-3, and Unk-4 in the test cases 'b' at a sequence length of 1400 words [65].

and 'b', respectively. The UNKs sequences are rejected with an accuracy of 100% by all methods. Also when using words, the Unk-1 and Unk-2 emitters are not rejected in the test cases 'a'. In the cases 'b', the LSTM with entropic open-set loss reliably rejects the Unk-2 emitter and the LSTM with cross-entropy loss also achieves higher accuracies than in the test cases 'a'. All methods are able to classify the Unk-3 and the Unk-4 emitters as unknown with more than 40% accuracy in all test cases. The Unk-1 and Unk-2 emitters make use of known words only, while the Unk-3 and the Unk-4 emitters also use some unknown words.

Figure 8.11 provides a comparison of the rejection accuracies for the Rules-v2 radar with syllables or words. It is observed that words are much better suited to reject this emitter than syllables. This is also true for the other unknown unknown

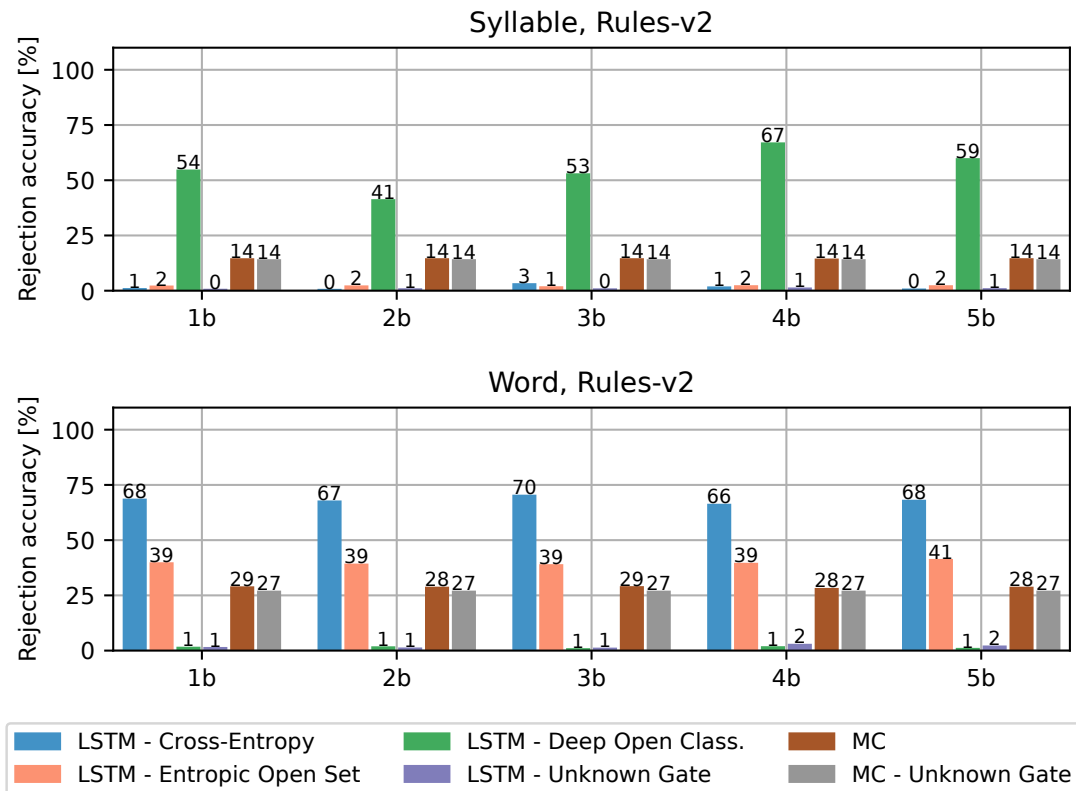


Figure 8.11: Rejection accuracies for the Rules-v2 emitter at a sequence length of 1400 words or syllables [65].

radars. Although the Rules-v2 radar is very similar to the known emitters, it is rejected with an accuracy of about 70 % by the LSTM with cross-entropy loss, which is enough to recognise that there is something unknown in the intercepted data. With syllables, the best rejection accuracy is 3 % for this method. One reason for this higher accuracy is that the Rules-v2 radar has less words than syllables in common with the other known emitters and therefore more words are mapped to UNK. This highlights again the benefits of the hierarchical structure of the emission model.

The overall results show that the rejection accuracies of the LSTM-based methods for the unknown emitters depend on the test case. In contrast, the performance of the MC-based approaches is stable except for small statistical deviations due to some randomness in the test case generation. As an MC does not have a memory, processing the data of other emitters in the scenario does not influence the classification decision. For the LSTMs, however, also the past sequences affect their internal state.

Impact of the Configuration. The results discussed above represent the accuracies that can be achieved if the best configuration for the specific evaluation metric is chosen for each method. Table 8.12 provides “evaluation matrices” for syllables that depict the mean accuracies obtained by different combinations of configuration and evaluation metric. The row of each matrix represents the configuration that works

Table 8.12: Evaluation matrices at a sequence length of 1400 syllables. The rows represent the configuration that works best for the specified metric and the columns denote the metric that it is evaluated with. All values in [%].

(a) LSTM with cross-entropy loss.					(b) LSTM with entropic open-set loss.				
Best config.	Evaluation metric				Best config.	Evaluation metric			
		acc_{dist}	acc_{id}	mean			acc_{dist}	acc_{id}	mean
	acc_{dist}	71.14	75.34	73.24		acc_{dist}	72.11	76.07	74.09
	acc_{id}	56.25	84.88	70.56		acc_{id}	55.86	79.86	67.86
	mean	64.35	82.80	73.57		mean	71.46	79.74	75.60
(c) LSTM with deep open classification loss.					(d) MC.				
Best config.	Evaluation metric				Best config.	Evaluation metric			
		acc_{dist}	acc_{id}	mean			acc_{dist}	acc_{id}	mean
	acc_{dist}	65.04	37.49	51.27		acc_{dist}	76.94	93.31	85.13
	acc_{id}	65.04	37.49	51.27		acc_{id}	71.80	95.24	83.52
	mean	65.04	37.49	51.27		mean	76.86	94.53	85.70

best for the specified metric while the columns denote the metric that the configuration is evaluated with. For example, the row “ acc_{dist} ” in combination with the column “ acc_{id} ” provides the mean identification accuracy of the configuration that is best for the distinction accuracy. The values on the diagonal are the accuracies of “matching” configuration and evaluation metric. The accuracies are displayed for a sequence length of 1400 syllables and averaged over all test cases ‘a’ and ‘b’. The label “mean” refers to the mean of the identification and the distinction accuracy. The corresponding best configurations are given in Table 8.13. For the LSTM with deep open classification loss, whose results are shown in Table 8.12c, the best configuration is always the same and hence, the results do not change when choosing a different evaluation metric. For the other LSTM-based methods, the distinction accuracy decreases especially when the configuration that is best for the identification accuracy is chosen. By selecting a configuration, a decision between true and false rejection rate needs to be made. As new information is supposed to be collected in an ELINT application, it is most probably better to reject some of the known sequences than to classify something unknown as known. However, if too many of the known sequences are rejected, the amount of required time-consuming manual analysis increases.

Table 8.13: Configurations (training case, δ) that achieve the highest mean of distinction and identification accuracy, averaged over the test cases 1a to 5a and 1b to 5b, respectively.

	Method	Syllable Sequence length			Word Sequence length		
		1	600	1400	1	600	1400
Test cases 'a'	LSTM						
	Cross-Entropy	(I, 0.0)	(0, 0.5)	(0, 0.5)	(I, 0.0)	(0, 0.5)	(0, 0.5)
	Entropic Open Set	(I, 0.5)	(I, 0.5)	(I, 0.5)	(I, 0.4)	(I, 0.5)	(I, 0.5)
	Deep Open Class.	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)	(I, 0.4)
Test cases 'b'	MC	(0, 0.4)	(0, 0.5)	(0, 0.5)	(0, 0.4)	(0, 0.5)	(0, 0.5)
	LSTM						
	Cross-Entropy	(II, 0.0)	(III, 0.0)	(III, 0.0)	(II, 0.0)	(III, 0.0)	(III, 0.0)
	Entropic Open Set	(IV, 0.6)	(II, 0.9)	(II, 0.9)	(II, 0.6)	(IV, 0.6)	(IV, 0.7)
	Deep Open Class.	(II, 0.4)	(IV, 0.4)	(IV, 0.4)	(II, 0.4)	(II, 0.9)	(II, 0.5)
	MC	(II, 0.0)	(IV, 0.0)	(IV, 0.0)	(II, 0.0)	(IV, 0.0)	(IV, 0.0)

Table 8.14: Evaluation matrices at a sequence length of 1400 words. The rows represent the configuration that works best for the specified metric and the columns denote the metric that it is evaluated with. All values in [%].

(a) LSTM with cross-entropy loss.					(b) LSTM with entropic open-set loss.				
Best config.	Evaluation metric				Best config.	Evaluation metric			
		acc_{dist}	acc_{id}	mean			acc_{dist}	acc_{id}	mean
	acc_{dist}	83.96	73.84	78.90		acc_{dist}	80.85	73.42	77.14
	acc_{id}	64.41	95.15	79.78		acc_{id}	66.15	81.01	73.58
	mean	82.30	94.07	88.19		mean	79.07	79.11	79.09
(c) LSTM with deep open classification loss.					(d) MC.				
Best config.	Evaluation metric				Best config.	Evaluation metric			
		acc_{dist}	acc_{id}	mean			acc_{dist}	acc_{id}	mean
	acc_{dist}	75.75	45.57	60.66		acc_{dist}	80.92	93.55	87.24
	acc_{id}	75.73	45.57	60.65		acc_{id}	77.03	95.36	86.19
	mean	75.75	45.57	60.66		mean	80.92	93.55	87.24

Table 8.14 shows the evaluation matrices with different metrics at a sequence length of 1400 words. Similar patterns are observed as for syllables. The configuration that is best for the identification accuracy provides much worse results for the distinction accuracy for the LSTMs with cross-entropy and entropic open-set loss. However, the configuration optimised for the average of both metrics seems to be a good compromise if no hierarchical combination of an unknown gate and a classifier for the known classes is desired. For the MC, the decrease of the distinction accuracy is less visible. The LSTM with deep open classification loss obtains its best results for all metrics with the same configuration in most cases and therefore, the results do not change much.

8.4.2 Evaluation with Missing and Additional Symbols

Since in the real world, the data is not ideal, the methods are tested with corrupted data. As an example, the evaluation on sequences of length 1400 is presented with 20 % missing or additional symbols, which is the worst case scenario as shown in Chapter 6. The symbols are either removed or inserted randomly, while the additional symbols are chosen from the global dictionary that contains the symbols of all emitters. Therefore, also symbols of other emitters might be inserted into the sequences.

Distinction Accuracy. Figure 8.12 shows the results of the best configurations for the distinction accuracy with missing or additional syllables in comparison to the results obtained with ideal data. The labels of the bars represent the accuracies relative to the ideal case, analogous to (6.11). All LSTM-based methods are very robust with respect to missing syllables, while the LSTMs trained with cross-entropy loss and entropic open-set loss only exhibit a small accuracy decrease with additional syllables as well. As also observed in Chapter 6, the MC is not as robust as the LSTM-based methods. The MC-based unknown gate is much more robust than the MC, but still provides worse results than the LSTMs. All methods are able to reliably reject the UNKs sequences, even if they contain 20 % known syllables. Hence, if the symbol extraction step wrongly maps an unknown input sequence to a known syllable, it does not change the result as long as there are sufficiently many UNK symbols. The rejection accuracy of the LSTM-based methods is still 100 %, while the accuracies of the MC and the MC-based unknown gate in test case 5b reduce slightly to 99.71 % and 99.77 %, respectively.

Figure 8.13 depicts the distinction accuracies using words with corrupted data in comparison to the ideal case. All LSTM-based methods are trained with random sequence lengths because this variant provides the best accuracy for ideal data. However, as also observed in Chapter 6, it is not as robust with respect to additional words as the LSTM trained with the complete scenarios. This can be seen for the LSTM with cross-entropy and entropic open-set loss, but the other two approaches are also very robust with additional words. Missing words cause no accuracy loss for any of the LSTM-based methods. Also the MC-based approaches only exhibit a slight decrease.

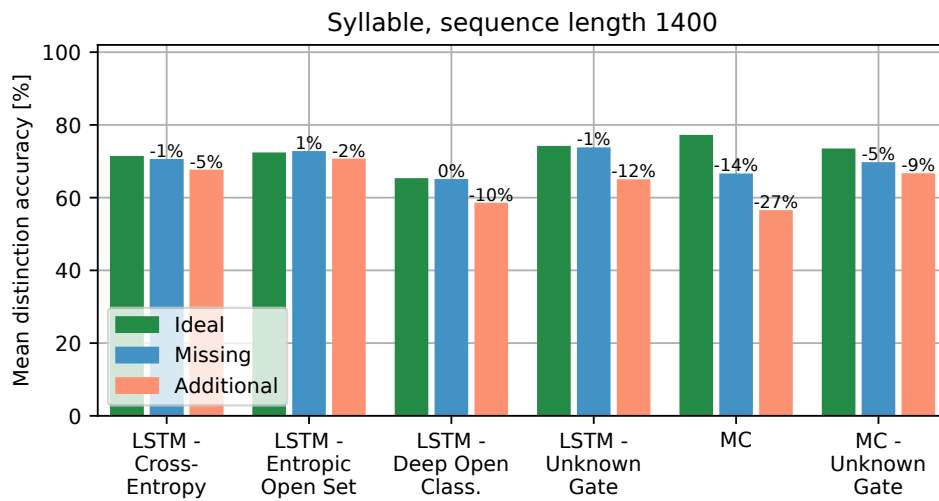


Figure 8.12: Mean distinction accuracies of the best configurations with 20 % missing or additional syllables, respectively [65].

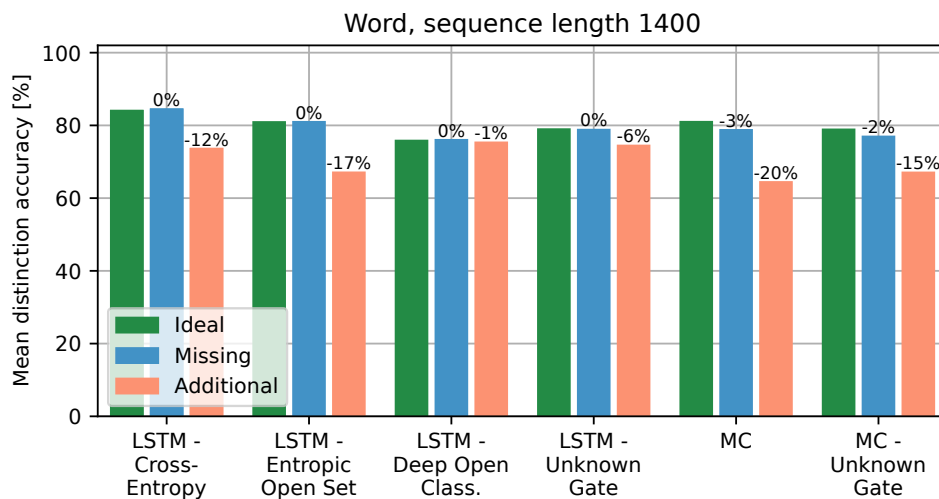


Figure 8.13: Mean distinction accuracies of the best configurations with 20 % missing or additional words, respectively [65].

With additional words, however, the accuracy decreases significantly. This was to be expected when considering the results obtained in Chapter 6. With additional words and the distinction accuracy, the LSTM with cross-entropy loss is outperformed by the LSTM with deep open classification loss and the LSTM-based unknown gate. With missing words, it still provides the best average performance. Although the UNKs sequences contain 20 % known symbols, they are still rejected with an accuracy of 100 % by the LSTM with cross-entropy loss. The rejection accuracy of the MC decreases by 0.32 percentage points in the test case 5b, which is negligible.

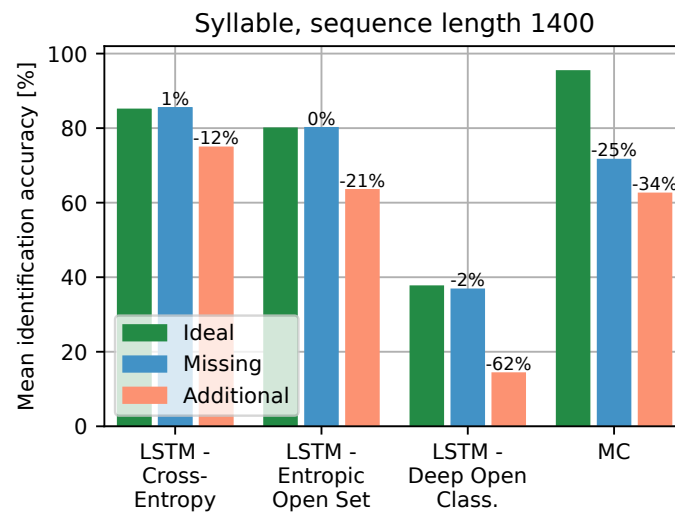


Figure 8.14: Mean identification accuracies of the best configurations with 20 % missing or additional syllables, respectively [65].

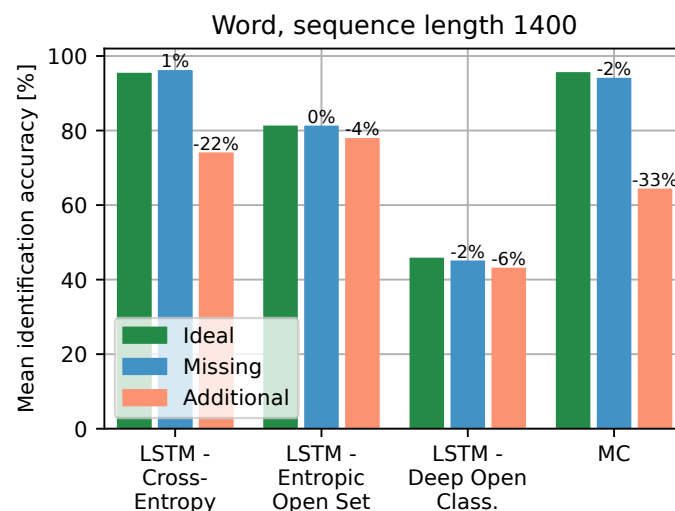


Figure 8.15: Mean identification accuracies of the best configurations with 20 % missing or additional words, respectively [65].

Identification Accuracy. Figure 8.14 displays the results for syllables of the best configurations for the identification accuracy with corrupted data. As before, the labels of the bars correspond to the accuracies relative to the results obtained with ideal data. Also in this case, missing syllables do not cause much difference in the accuracies for the LSTM-based methods, but the accuracy of the MC decreases significantly. This is mainly because of a confusion between the known emitters. With additional syllables, the MC also rejects some of the known sequences, while at the same time accepting about 25 % of the UNKs sequences in some test cases. The LSTM with cross-entropy

loss is the most robust method and also provides higher average accuracies than the MC. The other two LSTM-based methods exhibit a higher accuracy decrease with additional syllables. Especially the LSTM with deep open classification loss is not robust and rejects the complete input in some of the test cases.

The results for words of the methods' best configurations for the identification accuracy are shown in Figure 8.15. Also here, missing words do not cause much harm. For additional words, however, the MC exhibits a very large accuracy decrease of 33 %. This is partly due to a higher false rejection rate, but also because the known emitters are more often confused. Also the accuracy of the LSTM with cross-entropy loss decreases significantly with additional words because it wrongly assigns 19.48 % of the QoS radar's sequences to the Rules-v1 radar. The best configuration for the identification accuracy of the LSTM with entropic open-set loss is much more robust than the variant for the distinction accuracy and it provides the highest accuracy with missing and additional words, but its performance is much lower than the best values with ideal data.

8.5 Summary

This chapter investigates six methods in several configurations for determining if an input does not belong to any of the known emitter classes. This is especially important for an ELINT application because the goal of ELINT is to collect new information about radar emitters. In the literature, this problem is referred to as open-set recognition and several solutions are suggested. The general challenge in open-set recognition is that no training data for the "unknown unknown" classes exists but the classifier needs to be trained with "known unknown" data, which hopefully allows for the rejection of the unknown unknown input. This chapter suggests several approaches for the generation of known unknown data, which includes sampling random sequences from the emitters' dictionaries or altering an MC that is trained to predict the emissions of the radar (see Chapter 5). It is shown that to improve the capability of the classifier to distinguish between known and unknown, training with all available known unknown sequences provides the best results for all approaches when using syllables. For words, the best training case is not as clear, but with a sequence length of 600 and 1400 words, at least half of the methods also provide the highest distinction accuracy when trained with all available known unknown data.

The evaluation shows that all of the methods are able to reject sequences consisting only of the special symbol UNK when employed in their best configuration, even though not all of them are trained with it. Rejecting the UNKs sequences is also successful with the LSTM trained with cross-entropy loss and only the known classes, which is the conventional training. It reaches an accuracy of more than 97 % in the test cases 1a to 5a for syllables, while obtaining 100 % in all but one of them. This is achieved with a threshold on the network's confidence values of 0.5. Although "thresholding softmax" is claimed to be problematic because the softmax function can cause neural networks to be overconfident [56], it works very well for this kind of

input. Consequently, a completely unknown emitter can be reliably detected as such, also if the classifier is not trained to do so. The results with additional symbols show that this is even possible when the UNKs sequences contain 20 % known symbols.

Emitters that are more similar to the known classes are less reliably recognised as unknown, while higher rejection accuracies are achieved based on words. For four of the five unknown unknown emitters, the best rejection rates are above 65 % with words, which is sufficient to tell that there are active unknown emitters.

For syllables, the MC achieves the highest average accuracy with ideal data, both for the identification and the distinction accuracy. The LSTM with cross-entropy loss provides the best performance for words with the distinction accuracy at longer sequence lengths, while the MC achieves better results with a sequence length of only one word. However, the evaluation shows that there is no method that outperforms every other approach in all test cases.

Both LSTMs trained with the loss functions that are specially designed for recognising unknown input provide a lower average accuracy for classifying known classes than the LSTM trained with cross-entropy loss. Also the configurations that work best for the distinction between known and unknown provide a lower identification accuracy. However, most of the configurations which are optimal for the average of the identification and the distinction accuracy provide a good compromise between the two metrics. Therefore, it depends on the situation whether it is worth the additional computational effort to combine a classifier to distinguish between known and unknown with a classifier for the known classes.

With corrupted data, the performance naturally decreases. As also found in the previous chapters, missing symbols only have a small impact while additional symbols can cause a very large accuracy decrease. Especially the MC is not robust, while most of the LSTM-based methods exhibit less accuracy decrease. Both for syllables and words, the LSTM outperforms the MC with corrupted data at longer sequence lengths, although the MC's accuracy in the ideal case is much higher for syllables and comparable for words.

Chapter 9

Conclusions

This thesis proposes a new electronic intelligence (ELINT) processing chain for radar emitter identification and emission prediction. It employs machine learning for handling the signals of agile multifunction radars that traditional processing approaches are not able to manage effectively. The new processing chain provides solutions to four of the five problems of electronic warfare (EW) as defined in Section 2.4, namely the classification or identification of radars, the prediction of emissions, the learning or training of behavioural models, and the recognition of unknown emitters.

The basis of all developed approaches is the representation of radars as systems that speak a language, which is divided into the five modelling levels called letters, syllables, words, commands, and functions. A similar kind of hierarchical emission model was first introduced and extended in [12–21] with the four levels letters, words, commands, and tasks, but the emission model is adapted in the present work to better suit the needs of ELINT. This representation as a radar language allows for taking advantage of methods developed in the field of natural language processing (NLP). In this thesis, it is suggested to use word embeddings, which are dense vector representations of the radar language symbols, i.e. the different letters, syllables, words, commands, and functions. These representations are learnt by a neural network from the context that the symbols appear in and they have been shown to capture semantic relationships in NLP [25].

The two main approaches compared in this thesis are the Markov chain (MC) without a memory about the past and a variant of a recurrent neural network (RNN), the Long Short-Term Memory (LSTM), which is especially designed to remember previous inputs. Both are employed for predicting the emissions of the example emitters, identifying the emitter type based on its emissions, and recognising if an input belongs to an unknown emitter. It is shown that for emission prediction, the MC and the LSTM perform almost equally, while the LSTM outperforms the MC only for the most complicated task of predicting words. Both approaches clearly provide better results for syllables and words than simple methods like random guessing or repeating the last symbol, with an improvement of up to 120 %. Due to the structure of the data, though, simple methods provide similar results for letters, commands, and functions.

For identification of the radar emitter type, the MC provides notably better results than the LSTM, but only with ideal data. If the data is corrupted, i.e. symbols are missing or additional symbols are inserted, the accuracy of the MC rapidly decreases, while the LSTM is very robust due to its memory. It is shown how the identification accuracy depends on the number of consecutive symbols of a single emitter, both for training the LSTM as well as testing the LSTM and the MC. Although the symbols

used by the example emitters highly overlap, both methods are able to identify the emitter type, while longer symbol sequences are needed to distinguish between very similar radars. The results show that words are the modelling level best suited for identification. In addition, it is demonstrated that for syllables and words, both the MC and the LSTM clearly outperform a simple dictionary lookup, which resembles the conventional approach of comparing the received waveform parameters to entries in a database.

To actually employ the emitter models for prediction, they need to be combined into a multi-model system or an ensemble, in which the individual predictive models are called experts in the literature. There are two general principles for a combination, which are classifier fusion and classifier selection. This thesis investigates six architectures in several configurations, which either perform classifier fusion or selection. To increase the informative value, the architectures are first compared based on a public dataset and afterwards, the three best methods are implemented with the predictive emitter models as experts. In two of these architectures, the same LSTM which is developed for identification of the radar type is employed to select the emitter model(s) that should make the prediction for the current input. Since the experts might be LSTMs, it is investigated how the prediction accuracy is influenced if they need to process data that they were not trained with. It is shown that processing the data of other experts corrupts the state of the LSTMs and therefore decreases the prediction accuracy. There are two possibilities to solve this problem. The first one is to perform a state reset of the experts when the concept has changed. However, this requires the detection of such a change, which is called drift detection in the context of machine learning. In an ELINT application, drift detection is available from context knowledge because after deinterleaving, start and end of the sequences are known. The second option is to employ classifier selection, which avoids the corruption of the LSTM states by only routing the input to the expert that was most probably trained with the correct data. However, this approach requires a high identification accuracy. As a result of these evaluations, this thesis provides guidelines for the selection of an architecture that takes the identification accuracy, the availability of drift detection, and the length of the stability period, i.e. the number of consecutive symbols from the same concept, into account.

As the goal of ELINT is to gather information about radars, recognising an unknown emitter is very important. The classifiers are trained on the “known unknown” data, which should help them to also reject the “unknown unknown” input. This thesis suggests methods to generate known unknown training data and it shows that in most cases, training with this data increases the accuracy of the methods to distinguish between known and unknown input. The evaluation is performed with several unknown emitters, which differ in their similarity to the known radars. The most different emitter, which does not have any symbols in common with the known radars, is reliably identified as unknown, even with corrupted data. The more similar the unknown radars are to the known classes, the lower the rejection accuracy. As for identification, words are best suited to recognise an emitter as unknown. With words, four of the five unknown unknown emitters are recognised as such with an accuracy

of more than 65 % and up to 100 %. This is high enough to at least be able to tell that unknown signals are received. As is the case for identifying the emitter type, MCs achieve the highest accuracies for syllables, but only with ideal data. The LSTMs outperform the MC with syllables if the data is not ideal and are also to be preferred for words.

Considering the radar emissions as a language with an inherent hierarchical structure provides advantages for all considered tasks. Since the example emitter is airborne, it integrates many pulse repetitions, which leads to very long sequences in the order of 10 000 repetitions of the same letter. Neither the MC nor the LSTM is able to predict the next letter if it is different from the current one. However, the next syllable, which is composed of letters, can be predicted with high accuracy. Also the identification based on letters is not possible, but high accuracies are achieved for syllables and words. In addition, a comparison between emission prediction with the sparse vector representation of the symbols via one-hot encoding and the dense representation via word embeddings shows that word embeddings speed up the training process, while at the same time increasing the stability and the achieved performance.

9.1 Future Work

This thesis provides solutions to several problems, but it also gives rise to new questions and opens new directions of research. First of all, this thesis shows that word embeddings can be used to represent radar signals and that these representations provide advantages in further processing the symbols with neural networks. However, the evaluation is restricted to the word2vec algorithm. A possible future direction of research is therefore to investigate which other methods from the field of representation learning are applicable and provide the most advantages. These methods might as well be applied to the raw signal or pulse descriptor word (PDW) sequences and might therefore be useful in the deinterleaving step, too.

The five problems of EW, as defined in Section 2.4, also include the decoding of the internal state of the radar, which is not addressed in this thesis. With the adapted hierarchical emission model, it could be realised in a straightforward way. In this approach, the decoding of the internal state corresponds to mapping the letters, syllables, words, or commands to functions.

For the simulated airborne radar employed in this thesis, switching to a different letter cannot be predicted because of the high number of repetitions. However, predicting the next letter, which corresponds to a pulse, is important in applications like adaptive jamming. As syllables can be predicted very well, a possible solution to this problem is to predict the next letter based on the syllables, but a very large number of letters needs to be received in order to identify the current syllable. Therefore, an approach that keeps several hypotheses about the current syllable while receiving the letters is a future direction of research.

Another interesting direction is to investigate ensembles of different methods for recognising unknown emitters in order to increase the rejection accuracy. Several

approaches can be considered to achieve this goal, e.g. majority voting or (weighted) averaging.

Since the example radar used in this thesis is simulated, as much data as needed can be generated for training the MCs and LSTMs. In an actual application, this is probably not the case. Hence, investigating how well the methods work with less data is of interest. Finally, an evaluation of the presented approach with real data is desirable. Unfortunately, the data from real multifunction radars is hard to obtain for reasons of military secrecy.

Appendix A

Appendix: Additional Material

A.1 Identification of the Radar Emitter Type

This section contains the confusion matrices for syllables of the $LSTM_{scen}$ and the MC to support the findings of Chapter 5.

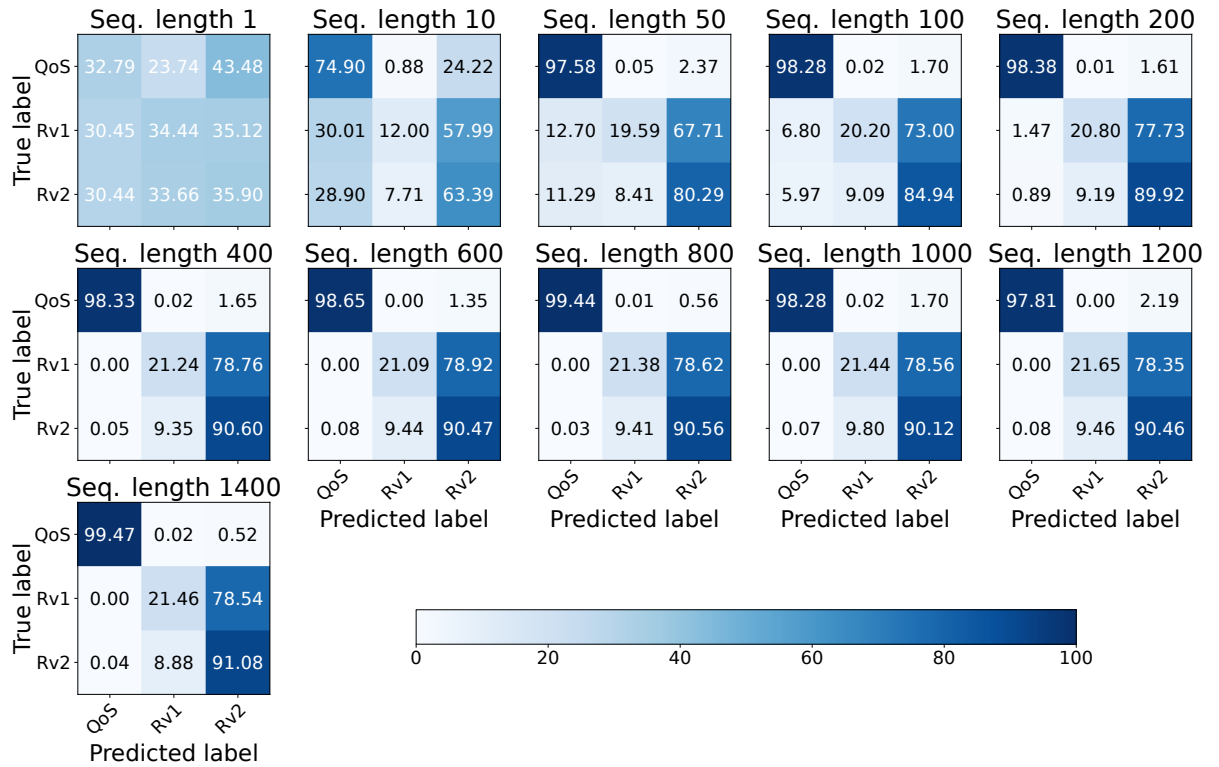


Figure A.1: Confusion matrices of the $LSTM_{scen}$ at different sequence lengths for syllables.

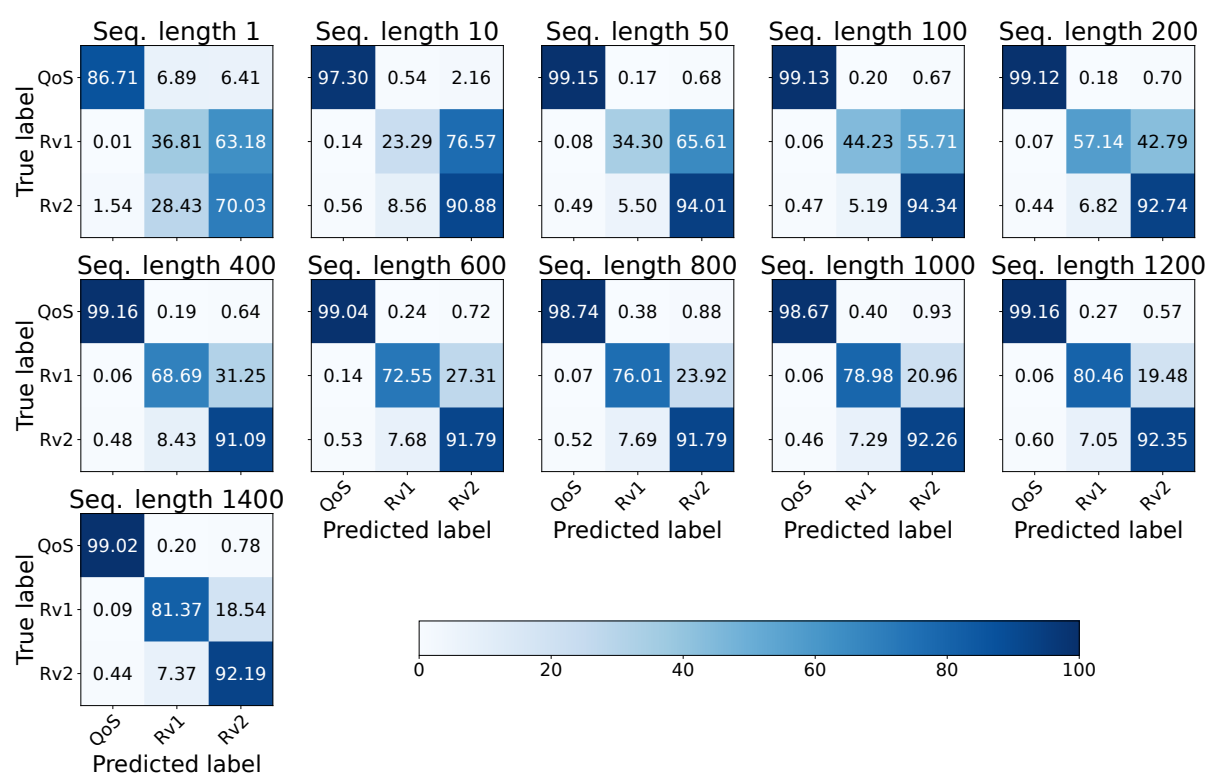


Figure A.2: Confusion matrices of the MC at different sequence lengths for syllables.

A.2 Ensembles of Predictive Models

This section of the appendix provides detailed results for each ensemble architecture and each scenario. The highest prediction accuracies per stability period, i.e. per column in the table, are marked in bold. The sparsely-gated Mixture of Experts (MoE) architecture with perfect identification is excluded since this is not an ensemble that could be actually used, but is only given for reference. The section also contains additional figures that show the complete results of all architectures, the impact of a state reset on the MoE architecture, as well as an individual comparison of the results for the online weighting based on top k accuracy for different k .

Table A.1: Top 1 prediction accuracies [%] for the *HMM* scenario. FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				8.7	12.9	15.1	15.9	15.2	15.0	16.1	14.7	16.7	16.4	19.0	19.3
			x	9.2	12.4	15.0	15.9	15.6	14.7	15.5	15.4	16.5	16.1	17.7	18.8
Sparse MoE	d=1			7.9	12.5	15.1	15.7	15.4	14.8	15.6	15.1	15.6	16.1	17.2	18.8
	d=2			8.5	12.9	14.9	15.9	15.2	15.3	16.2	15.0	16.1	16.1	18.2	20.6
	d=1	x		8.1	11.9	14.7	15.6	15.4	14.4	15.4	15.4	15.9	16.1	17.2	19.3
	d=2	x		9.0	12.2	14.9	15.9	15.6	14.7	15.5	15.4	16.5	16.1	17.4	19.0
Stacking	LSTM			10.8	11.6	12.6	13.6	13.6	13.4	14.2	13.3	14.8	14.1	13.5	12.8
	LSTM	x		11.4	13.1	13.3	13.9	13.7	13.2	13.5	13.2	15.1	13.5	14.8	11.7
	FF			9.8	10.7	11.2	11.9	12.4	12.5	12.1	12.5	11.7	12.8	13.3	11.5
	FF	x		9.8	9.3	9.5	11.4	11.4	12.3	12.9	12.2	10.0	12.5	12.5	11.7

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Stacking with Input	LSTM			14.8	14.5	14.4	14.8	15.2	15.1	16.0	14.6	13.8	15.6	13.5	13.5
	LSTM	x		14.8	14.5	14.4	14.8	15.2	15.1	16.0	14.6	13.8	15.6	13.5	13.5
	FF			9.8	10.7	11.5	11.8	11.2	10.6	11.2	11.3	12.2	11.7	12.0	12.5
	FF	x		10.8	7.4	8.8	11.7	12.1	11.2	12.2	11.5	10.8	11.5	11.5	12.8
Online Weighting	k=1			10.3	12.6	14.2	15.2	14.9	15.1	15.6	14.6	14.5	16.4	15.9	16.9
	k=5			10.2	13.3	13.8	14.6	14.9	14.0	15.7	14.5	14.6	17.4	14.1	18.5
	k=10			10.8	14.3	14.5	15.3	15.7	15.3	16.7	15.1	15.1	18.0	16.7	18.0
	k=20			10.9	14.6	14.7	15.4	15.6	15.5	16.1	15.8	14.8	17.4	16.1	18.0
	k=1	x		11.5	12.5	12.9	14.0	14.8	13.8	15.5	15.8	14.8	15.9	15.9	15.1
	k=5	x		11.4	13.1	13.1	13.5	13.8	13.3	15.2	15.6	15.1	16.7	16.4	18.2
	k=10	x		12.5	13.5	13.5	14.9	15.2	14.5	16.9	15.8	15.4	16.4	16.4	18.0
	k=20	x		12.2	13.1	13.4	15.0	15.0	15.1	16.8	15.1	14.7	17.4	16.4	18.8
	k=1		x	13.5	13.6	14.4	15.2	14.8	14.5	14.8	14.5	15.5	16.4	15.9	16.9
	k=5		x	13.5	13.8	14.1	14.7	14.5	14.2	15.7	14.6	14.3	17.4	14.1	18.5
	k=10		x	13.5	13.8	14.3	15.1	15.6	15.0	15.5	14.1	15.8	18.0	16.7	18.0
	k=20		x	13.5	13.9	14.5	15.2	15.7	15.2	15.9	13.7	15.5	17.4	16.1	18.0
	k=1	x	x	14.8	13.2	13.1	14.0	14.5	14.6	14.8	15.1	15.2	15.9	15.9	15.1
	k=5	x	x	14.8	13.6	13.3	13.8	14.2	14.1	14.5	14.1	14.3	16.7	16.4	18.2
	k=10	x	x	14.8	13.5	13.5	15.0	15.2	14.9	15.5	14.2	15.2	16.4	16.4	18.0
	k=20	x	x	14.8	12.9	13.4	15.0	15.7	14.9	15.5	15.1	15.4	17.4	16.4	18.8
	MSE			5.2	5.1	5.3	5.5	5.6	8.9	14.8	16.1	8.3	18.2	13.3	12.2
	MSE	x		5.9	6.9	6.6	6.5	5.9	8.6	15.0	16.0	8.2	15.9	12.8	11.7
	MSE		x	13.5	13.9	15.1	16.9	16.7	16.7	17.3	15.4	17.6	18.0	19.3	18.8
	MSE	x	x	14.8	13.2	14.1	16.4	16.5	16.3	16.3	16.1	16.8	17.4	19.3	18.5
Model Averaging				13.5	13.4	13.6	14.3	14.8	15.0	15.7	12.9	14.2	16.4	13.5	13.3
		x		14.8	13.0	12.8	14.1	14.8	14.7	15.6	14.5	14.6	16.7	13.8	13.5
Sparse MoE (perfect id)	d=1			16.8	16.7	16.8	16.8	16.5	16.0	16.8	15.1	16.3	16.1	16.7	19.0
	d=1	x		16.3	15.2	15.5	16.4	16.4	15.2	16.0	15.4	16.7	16.1	16.7	19.0

Table A.2: Top 1 prediction accuracies [%] for the *NLP* scenario. FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				9.8	28.5	33.8	35.4	35.4	35.3	35.8	34.8	37.0	35.9	36.1	35.2
		x		10.6	28.8	33.8	35.4	35.6	35.6	35.8	35.4	37.4	36.5	36.8	35.1
Sparse MoE	d=1			9.5	29.0	33.6	35.3	35.6	35.5	35.9	35.3	37.3	36.5	36.4	35.2
	d=2			9.8	29.0	33.8	35.3	35.6	35.5	35.8	35.2	37.3	36.4	36.6	35.0
	d=3			9.8	28.8	33.7	35.3	35.6	35.6	35.9	34.9	37.4	36.3	36.6	35.1
	d=4			9.8	28.6	33.8	35.3	35.6	35.6	35.6	35.0	37.2	36.2	36.6	35.3
	d=1	x		9.9	28.3	33.6	35.3	35.6	35.5	35.9	35.3	37.3	36.5	36.4	35.2
	d=2	x		10.4	28.7	33.8	35.3	35.6	35.6	35.8	35.4	37.4	36.5	36.8	35.1
	d=3	x		10.5	28.8	33.8	35.4	35.6	35.6	35.8	35.4	37.4	36.5	36.9	35.1
	d=4	x		10.6	28.8	33.8	35.4	35.6	35.6	35.8	35.4	37.4	36.5	36.9	35.1
Stacking	LSTM			7.3	14.1	17.6	19.5	19.5	19.6	19.7	18.7	19.8	20.2	19.2	19.3
	LSTM	x		7.8	14.8	17.9	19.5	19.6	19.3	19.9	19.5	20.1	19.9	18.8	19.3
	FF			5.4	5.4	5.4	5.2	5.2	5.5	6.0	5.3	5.8	6.2	4.1	5.2
	FF	x		5.4	5.4	5.4	5.2	5.2	5.5	6.0	5.3	5.8	6.2	4.1	5.2
Stacking with Input	LSTM			7.6	12.1	14.0	17.5	18.3	19.7	16.7	16.9	16.5	18.8	18.4	19.0
	LSTM	x		8.0	12.4	14.3	17.9	18.3	20.1	17.3	17.2	16.5	18.1	19.3	17.5
	FF			5.4	5.4	5.4	5.2	5.2	5.5	6.0	5.3	5.8	6.2	4.1	5.2
	FF	x		5.4	5.4	5.4	5.2	5.2	5.5	6.0	5.3	5.8	6.2	4.1	5.2

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Online Weighting	k=1			6.2	24.4	29.9	29.9	29.8	29.6	30.1	29.5	30.4	29.5	30.3	29.5
	k=5			5.8	26.5	28.5	28.3	27.9	27.8	28.5	27.9	28.4	28.0	28.8	28.6
	k=10			5.9	24.9	26.5	26.3	25.9	25.9	26.1	26.0	26.8	26.4	26.2	26.8
	k=20			6.4	22.0	23.3	23.1	23.0	22.9	23.0	22.3	23.3	22.3	23.5	23.4
	k=1	x		6.2	25.3	29.9	29.9	30.1	30.0	29.8	29.3	30.9	29.9	31.0	29.2
	k=5	x		5.4	27.1	28.5	28.5	28.0	28.2	28.2	27.7	28.6	28.6	29.2	28.0
	k=10	x		5.6	25.3	26.4	26.4	26.3	26.1	25.9	25.5	27.2	26.0	26.6	27.0
	k=20	x		6.4	22.2	23.2	23.2	23.3	23.0	23.1	22.7	23.9	22.4	23.0	23.8
	k=1		x	9.5	26.5	30.2	30.0	30.0	29.7	30.2	29.6	30.7	29.6	30.5	29.5
	k=5		x	9.5	28.6	29.2	28.4	27.9	27.8	28.5	27.5	28.4	28.0	28.8	28.6
	k=10		x	9.5	27.8	27.7	26.5	26.0	25.9	26.1	25.7	26.8	26.4	26.2	26.8
	k=20		x	9.5	25.2	24.6	23.3	23.1	22.9	23.0	22.0	23.3	22.3	23.5	23.4
	k=1	x	x	10.2	27.4	30.2	30.0	30.2	30.1	29.9	29.9	31.1	30.1	31.2	29.5
	k=5	x	x	10.2	29.1	29.1	28.6	28.0	28.2	28.2	27.6	28.6	28.6	29.2	28.0
	k=10	x	x	10.2	28.3	27.6	26.6	26.3	26.1	26.0	26.0	27.2	26.0	26.6	27.0
	k=20	x	x	10.2	25.5	24.6	23.5	23.5	23.0	23.1	22.4	23.9	22.4	23.0	23.8
	MSE			9.3	12.8	14.5	16.2	15.7	16.1	17.4	18.0	19.4	22.4	20.6	24.9
	MSE	x		11.0	13.1	14.8	16.4	15.8	16.1	17.1	18.0	20.1	22.3	21.2	24.5
	MSE		x	9.5	30.5	33.9	34.8	35.1	35.1	35.5	35.1	36.1	35.4	35.9	34.6
	MSE	x	x	10.2	31.3	34.0	34.8	35.3	35.5	35.4	35.5	36.3	35.6	36.4	34.7
Model Averaging				9.5	14.1	14.9	14.7	14.7	14.9	14.2	14.8	14.0	14.8	15.2	15.4
		x		10.2	14.3	14.9	14.8	14.6	14.8	14.8	14.9	14.1	14.9	15.2	15.7
Sparse MoE (perfect id)	d=1			35.6	35.8	35.6	35.4	35.7	35.6	36.0	35.7	37.3	36.6	36.8	35.2
	d=1	x		23.1	34.6	35.6	35.4	35.7	35.6	36.0	35.7	37.3	36.6	36.8	35.2

Table A.3: Top 1 prediction accuracies [%] for the *hard* scenario. FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				6.3	9.9	15.4	21.6	22.3	23.1	23.3	22.7	22.5	22.6	22.7	22.7
		x		5.8	10.4	15.7	21.3	22.2	22.7	23.7	23.1	22.9	22.5	22.4	22.8
Sparse MoE	d=1			5.9	7.9	11.9	21.0	22.3	22.5	23.4	22.7	22.8	21.3	21.6	22.5
	d=2			6.2	9.5	14.8	21.5	22.5	23.1	23.4	22.8	22.8	22.0	21.9	22.7
	d=3			6.3	9.9	15.6	21.7	22.5	22.9	23.3	23.0	22.2	22.5	22.3	23.0
	d=4			6.3	10.0	15.6	21.5	22.3	23.0	23.4	23.1	22.7	22.7	22.9	22.5
	d=5			6.3	9.9	15.5	21.6	22.3	23.0	23.4	22.8	22.4	22.5	22.8	22.7
	d=1	x		5.1	8.2	11.9	20.8	22.0	22.4	23.4	22.7	22.8	22.1	22.0	22.7
	d=2	x		5.5	9.7	14.8	21.3	22.2	22.8	23.7	23.0	22.9	22.4	22.4	22.8
	d=3	x		5.7	10.2	15.5	21.3	22.2	22.7	23.7	23.1	22.9	22.5	22.4	22.8
	d=4	x		5.7	10.4	15.7	21.3	22.2	22.7	23.7	23.1	22.9	22.5	22.4	22.8
	d=5	x		5.8	10.4	15.7	21.3	22.2	22.7	23.7	23.1	22.9	22.5	22.4	22.8
Stacking	LSTM			6.6	7.9	9.2	10.9	10.6	10.6	11.3	10.3	10.9	10.9	10.4	11.1
	LSTM	x		5.8	7.5	9.1	10.9	10.7	10.7	11.0	10.8	10.5	10.4	10.3	10.4
	FF			8.0	11.0	12.1	12.5	12.6	12.6	13.2	12.9	12.9	12.4	12.5	13.2
	FF	x		8.1	8.5	10.2	12.0	11.8	12.7	12.7	13.2	12.6	12.7	13.0	13.3
Stacking with Input	LSTM			10.6	11.8	11.8	11.9	11.7	11.8	12.0	12.9	12.0	11.7	11.2	10.4
	LSTM	x		10.4	11.7	11.9	11.9	11.8	12.0	12.1	13.2	12.0	12.1	11.8	10.4
	FF			9.4	12.2	13.2	13.5	13.5	13.5	14.5	13.6	15.0	14.0	14.2	12.9
	FF	x		12.3	9.1	11.0	12.8	12.9	14.0	14.6	13.0	15.2	14.5	15.8	13.7

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Online Weighting	k=1			7.1	14.0	17.5	19.0	18.9	18.5	20.1	18.9	18.5	19.3	21.1	18.6
	k=5			7.4	15.3	17.9	19.3	19.6	18.9	19.4	19.6	17.6	19.9	18.5	19.3
	k=10			7.7	15.4	17.8	19.0	18.6	18.3	19.2	18.8	17.8	19.3	18.2	17.4
	k=20			8.3	14.9	17.4	18.2	17.9	18.0	19.2	18.3	17.7	19.8	19.0	18.4
	k=1	x		8.5	15.1	17.3	18.4	18.9	18.5	19.9	19.2	19.1	19.9	20.2	18.5
	k=5	x		9.2	16.2	17.9	18.7	18.9	19.1	19.3	19.8	17.4	20.1	17.8	19.9
	k=10	x		10.7	15.5	17.4	18.4	18.3	18.0	18.7	18.9	18.0	19.5	17.7	18.2
	k=20	x		12.0	13.8	16.8	17.5	18.1	17.7	18.9	18.5	18.2	19.1	18.5	19.8
	k=1		x	9.0	15.2	18.0	19.1	19.0	18.6	20.0	18.9	18.5	19.4	21.1	18.6
	k=5		x	9.0	16.0	18.4	19.3	19.6	18.9	19.4	19.6	17.6	19.9	18.5	19.3
	k=10		x	9.0	15.5	18.0	19.0	18.5	18.3	19.2	18.8	17.8	19.3	18.2	17.4
	k=20		x	9.0	14.9	17.5	18.3	17.8	18.0	19.2	18.3	17.7	19.8	19.0	18.4
	k=1	x	x	13.6	16.2	17.9	18.5	18.9	18.6	19.9	19.3	19.1	20.1	20.2	18.5
	k=5	x	x	13.6	16.8	18.2	18.8	18.9	19.1	19.4	19.8	17.4	20.1	17.8	19.9
	k=10	x	x	13.6	15.7	17.6	18.4	18.3	18.0	18.7	18.9	18.0	19.5	17.7	18.2
	k=20	x	x	13.6	13.8	16.9	17.5	18.1	17.7	19.0	18.5	18.2	19.1	18.5	19.8
	MSE			7.6	9.7	12.2	17.3	13.2	14.2	18.7	12.8	17.8	18.6	16.9	19.7
	MSE	x		9.3	11.6	14.8	17.3	13.1	14.2	18.5	12.8	18.2	19.5	16.9	19.9
	MSE		x	9.0	16.9	21.0	23.5	23.6	24.0	25.0	24.3	24.2	24.8	25.4	25.9
	MSE	x	x	13.6	17.9	21.0	23.2	23.2	24.2	25.0	24.5	24.3	25.1	26.4	24.7
Model Averaging				9.0	14.1	16.6	17.4	17.1	17.1	18.3	17.1	17.1	18.2	17.4	17.4
		x		13.6	13.0	16.0	16.7	17.3	17.0	18.0	18.4	17.4	18.0	16.7	17.3
Sparse MoE (perfect id)	d=1			24.8	24.9	24.9	24.9	24.5	24.2	24.8	24.5	24.4	24.8	25.3	25.1
	d=1	x		18.7	21.8	23.6	24.4	24.2	24.1	25.1	24.5	24.9	25.3	25.3	25.1

Table A.4: Top 1 prediction accuracies [%] for the *easy* scenario. FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				9.1	33.3	41.1	42.4	42.5	42.6	42.2	41.9	41.3	43.2	42.3	38.8
		x		11.9	32.0	39.0	41.7	42.3	42.7	43.2	43.8	43.0	43.5	42.1	41.3
Sparse MoE	d=1			8.8	34.5	41.9	42.6	42.9	43.3	42.8	43.8	43.0	42.8	43.2	39.7
	d=2			9.3	34.7	41.6	42.6	42.7	43.1	43.1	44.1	43.2	43.5	43.0	40.1
	d=3			9.3	34.1	41.4	42.6	42.7	43.0	42.8	43.3	42.3	43.0	43.4	39.3
	d=4			9.2	33.7	41.3	42.4	42.7	43.0	42.3	43.3	42.0	43.4	42.4	39.5
	d=5			9.2	33.4	41.1	42.5	42.7	42.7	42.5	43.2	42.2	43.8	41.9	39.6
	d=1	x		10.7	31.2	38.9	41.8	42.3	42.7	42.9	44.1	43.0	42.8	43.2	41.0
	d=2	x		11.6	31.9	39.0	41.8	42.3	42.7	42.8	44.1	43.0	42.8	43.1	41.1
	d=3	x		11.8	32.0	39.0	41.8	42.3	42.7	42.8	44.1	43.0	42.8	43.1	41.1
	d=4	x		11.9	32.0	39.0	41.8	42.3	42.7	42.8	44.1	43.0	42.8	43.1	41.1
	d=5	x		11.9	32.0	39.0	41.8	42.3	42.7	42.8	44.1	43.0	42.8	43.1	41.1
Stacking	LSTM			8.5	16.1	19.7	22.2	22.2	22.8	22.9	23.3	22.2	22.4	21.9	21.7
	LSTM	x		10.1	14.0	17.4	19.9	20.5	22.1	21.5	22.5	22.1	22.3	23.2	22.1
	FF			5.3	5.3	5.3	5.2	5.3	5.4	5.6	6.1	4.8	5.9	4.0	5.5
	FF	x		5.3	5.3	5.3	5.2	5.3	5.4	5.6	6.1	4.8	5.9	4.0	5.5
Stacking with Input	LSTM			11.0	16.9	19.2	20.6	20.2	20.3	20.6	20.8	19.5	19.5	18.1	20.3
	LSTM	x		10.5	17.3	19.2	20.3	20.7	20.6	20.5	20.8	19.9	19.5	18.9	20.3
	FF			6.5	6.5	6.5	6.2	6.3	6.6	5.6	6.1	5.1	5.5	4.6	4.8
	FF	x		6.5	6.5	6.5	6.2	6.3	6.6	5.6	6.1	5.1	5.5	4.6	4.8

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Online Weighting	k=1			6.1	27.8	35.6	36.8	36.4	37.4	37.7	36.7	36.8	37.8	35.0	33.2
	k=5			5.6	28.9	33.4	33.9	33.9	34.0	34.2	33.5	33.7	34.6	32.8	29.7
	k=10			6.0	27.0	30.7	31.2	30.7	31.0	31.5	30.7	30.9	31.0	28.6	28.1
	k=20			6.7	23.3	26.4	27.5	27.1	27.5	28.2	27.6	27.1	25.7	25.9	25.3
	k=1	x		7.3	26.4	32.9	35.9	36.3	37.1	37.8	38.5	36.1	38.7	36.2	34.8
	k=5	x		6.5	26.5	30.4	32.8	33.0	33.6	33.4	34.2	33.1	33.5	33.3	30.7
	k=10	x		7.2	24.1	27.7	29.9	30.2	30.6	30.8	30.9	29.6	31.0	29.6	29.6
	k=20	x		8.8	20.7	23.4	25.9	26.4	26.4	26.5	27.2	26.0	27.1	25.5	24.6
	k=1		x	9.1	29.9	36.1	36.9	36.6	37.5	37.7	36.7	36.8	37.9	35.2	33.3
	k=5		x	9.1	31.4	34.4	34.2	33.9	34.1	34.2	33.5	33.7	34.6	32.8	29.7
	k=10		x	9.1	29.4	31.9	31.5	30.9	31.2	31.5	30.7	30.8	31.0	28.6	28.1
	k=20		x	9.1	25.6	27.7	27.7	27.2	27.6	28.2	27.7	27.1	25.7	25.9	25.3
	k=1	x	x	12.6	28.3	33.3	36.1	36.4	37.2	37.7	38.5	36.3	38.5	36.3	35.2
	k=5	x	x	12.6	28.8	31.3	33.0	33.2	33.6	33.4	34.2	33.1	33.5	33.3	30.7
	k=10	x	x	12.6	26.5	28.9	30.0	30.3	30.7	30.8	30.9	29.6	31.0	29.6	29.6
	k=20	x	x	12.6	22.6	24.5	26.2	26.4	26.5	26.5	27.2	26.0	27.1	25.5	24.6
	MSE			8.1	15.8	18.6	19.4	19.9	24.3	25.1	26.4	27.7	26.3	29.2	27.7
	MSE	x		11.1	13.6	15.8	18.3	18.6	23.9	24.7	25.7	27.8	26.6	29.8	28.0
	MSE		x	9.1	33.8	40.6	42.4	42.6	42.8	42.4	43.2	41.9	43.8	42.1	40.4
	MSE	x	x	12.6	31.8	38.3	41.7	42.3	42.9	42.7	44.1	43.1	42.7	43.4	41.1
Model Averaging				9.1	17.6	19.9	19.7	19.9	19.5	19.7	19.7	19.7	19.4	20.2	18.8
		x		12.6	14.7	17.2	18.4	18.3	18.4	18.9	18.8	18.6	20.2	19.9	19.0
Sparse MoE (perfect id)	d=1			42.8	43.0	42.9	42.6	42.9	43.4	42.8	43.6	43.0	43.1	43.2	41.1
	d=1	x		28.4	36.7	39.9	41.8	42.3	42.8	43.0	44.2	43.0	43.1	43.2	41.1

Table A.5: Top 1 prediction accuracies [%] for the *bad* scenario. FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				8.1	19.0	24.2	29.3	29.7	30.0	29.8	29.0	29.1	30.3	31.5	31.1
		x		8.7	19.2	24.2	29.8	29.9	30.8	30.8	29.2	29.6	32.1	33.3	32.1
Sparse MoE	d=1			8.0	19.6	23.9	29.9	30.1	30.8	30.8	29.1	29.4	32.1	33.0	32.1
	d=2			8.3	19.9	24.6	30.0	30.2	30.9	30.9	29.2	29.5	31.9	33.2	32.0
	d=3			8.3	19.8	24.6	30.0	30.1	30.8	30.9	29.2	29.4	31.9	33.3	32.1
	d=4			8.3	19.7	24.5	30.0	30.1	30.7	30.6	29.3	29.5	31.9	33.2	32.0
	d=5			8.3	19.5	24.3	29.9	30.1	30.7	30.6	29.3	29.4	31.9	33.0	32.0
	d=6			8.2	19.2	24.2	29.8	30.1	30.7	30.8	29.3	29.4	31.7	32.8	32.1
	d=1	x		8.0	18.6	23.6	29.8	29.8	30.7	30.8	29.1	29.7	32.3	33.0	32.1
	d=2	x		8.5	19.1	24.2	29.8	29.9	30.8	30.9	29.2	29.7	32.2	33.2	32.1
	d=3	x		8.6	19.2	24.2	29.8	29.9	30.8	30.9	29.1	29.7	32.2	33.2	32.1
	d=4	x		8.6	19.2	24.2	29.8	29.9	30.8	30.9	29.1	29.7	32.2	33.3	32.1
	d=5	x		8.7	19.2	24.2	29.8	29.9	30.8	30.9	29.1	29.7	32.2	33.3	32.1
	d=6	x		8.7	19.2	24.2	29.8	29.9	30.8	30.9	29.1	29.6	32.2	33.3	32.1
Stacking	LSTM			8.0	9.8	9.9	10.3	9.8	10.0	10.2	10.0	10.6	10.8	10.7	10.4
	LSTM	x		7.1	10.4	10.5	10.3	10.2	10.3	9.9	9.8	10.6	10.0	10.3	10.9
	FF			8.2	8.2	8.2	8.1	8.0	8.3	8.5	8.0	8.0	8.4	8.0	8.9
	FF	x		8.2	8.2	8.2	8.1	8.0	8.3	8.5	8.0	8.0	8.4	8.0	8.9
Stacking with Input	LSTM			10.6	11.8	12.1	12.4	12.8	12.9	12.9	12.0	12.1	11.9	13.0	13.3
	LSTM	x		10.8	11.6	12.3	12.4	12.8	12.8	13.0	12.0	12.1	11.9	12.9	13.1
	FF			8.2	8.2	8.2	8.1	8.0	8.3	8.5	8.0	8.0	8.4	8.0	8.9
	FF	x		8.2	8.2	8.2	8.1	8.0	8.3	8.5	8.0	8.0	8.4	8.0	8.9

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Online Weighting	k=1			5.5	18.9	25.2	25.6	25.8	25.3	25.5	25.8	26.0	26.0	25.7	25.4
	k=5			5.4	21.3	25.2	25.2	25.2	25.1	24.7	25.4	25.6	25.2	25.3	24.8
	k=10			5.5	20.1	23.9	24.3	24.2	24.1	24.0	24.7	25.6	23.8	24.1	22.8
	k=20			5.9	19.3	23.0	23.3	23.3	23.3	23.2	23.7	24.9	23.4	22.4	23.0
	k=1	x		6.4	20.3	25.2	26.1	26.0	26.1	26.5	26.2	26.4	27.3	27.0	26.2
	k=5	x		6.4	22.2	25.2	25.6	25.3	25.7	25.6	25.3	25.3	26.4	26.5	25.4
	k=10	x		7.0	20.8	24.0	24.6	24.4	24.7	25.0	25.1	26.0	26.3	25.7	23.8
	k=20	x		7.6	19.7	23.1	23.6	23.6	23.6	24.2	23.8	24.6	24.9	24.2	23.9
	k=1		x	8.2	20.8	25.6	25.8	25.8	25.4	25.4	25.9	26.3	26.2	25.8	25.6
	k=5		x	8.2	22.4	25.6	25.3	25.2	25.1	24.4	25.4	25.6	25.2	25.3	24.8
	k=10		x	8.2	21.1	24.3	24.3	24.2	24.1	23.9	24.7	25.6	23.8	24.1	22.8
	k=20		x	8.2	20.4	23.3	23.4	23.3	23.3	23.2	23.7	24.9	23.4	22.4	23.0
	k=1	x	x	10.8	22.0	25.7	26.2	26.1	26.2	26.6	26.4	26.4	27.5	27.2	26.3
	k=5	x	x	10.8	23.5	25.7	25.7	25.3	25.7	25.6	25.3	25.3	26.4	26.5	25.4
	k=10	x	x	10.8	22.1	24.3	24.6	24.4	24.7	25.0	25.1	26.0	26.3	25.7	23.8
	k=20	x	x	10.8	20.7	23.4	23.7	23.7	23.6	24.2	23.8	24.6	24.9	24.2	23.9
	MSE			7.3	11.3	12.7	13.7	13.0	14.1	13.7	12.4	14.0	19.7	18.5	17.4
	MSE	x		8.1	11.8	13.1	14.6	13.8	15.2	14.9	13.2	14.5	21.4	20.5	18.1
	MSE		x	8.2	22.7	27.6	29.4	29.7	29.3	28.9	29.2	29.8	28.9	30.4	30.1
	MSE	x	x	10.8	24.0	28.0	29.7	29.9	30.2	30.1	29.6	30.0	30.5	32.1	30.7
Model Averaging				8.2	14.9	17.6	18.0	18.2	18.3	17.4	16.9	18.9	17.0	17.2	17.7
		x		10.8	15.3	18.1	18.2	18.4	18.3	18.4	18.0	18.8	18.2	19.1	18.3
Sparse MoE (perfect id)	d=1			32.1	32.3	32.2	32.1	32.2	32.0	31.8	31.8	32.3	32.4	33.5	32.3
	d=1	x		19.3	28.3	31.3	31.8	31.8	31.9	32.0	31.6	32.4	32.5	33.5	32.4

Table A.6: Top 1 prediction accuracies [%] for the scenario containing all problems.
FF = feedforward.

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
MoE				6.2	15.8	21.7	24.6	24.1	24.0	23.7	23.2	24.3	26.3	24.2	23.9
	x			7.7	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.3	25.8	25.7
Sparse MoE	d=1			5.0	13.1	19.1	24.1	23.5	23.8	23.6	24.1	24.7	27.8	24.1	23.5
	d=2			5.6	14.8	21.0	25.0	24.3	24.7	24.7	25.0	25.3	27.7	26.0	24.7
	d=3			5.8	15.4	21.5	25.2	24.5	24.7	24.9	24.9	25.3	27.8	25.8	24.8
	d=4			5.9	15.7	21.7	25.3	24.5	24.9	24.6	24.9	25.5	27.9	25.9	25.1
	d=5			6.0	15.8	21.7	25.3	24.5	24.7	24.7	24.7	25.5	27.7	25.9	25.1
	d=6			6.0	15.8	21.7	25.2	24.6	24.7	24.7	24.7	25.5	27.6	25.8	25.4
	d=7			6.1	15.9	21.7	25.2	24.6	24.8	24.6	24.7	25.2	27.3	26.0	25.3
	d=8			6.1	15.8	21.8	25.2	24.4	24.8	24.7	24.6	25.5	27.2	25.9	25.1
	d=9			6.2	15.9	21.8	25.0	24.4	24.6	24.5	24.5	25.2	27.1	25.8	25.1
	d=10			6.2	15.8	21.8	25.0	24.4	24.6	24.6	24.3	25.2	27.0	25.4	25.2
	d=11			6.2	15.8	21.8	25.0	24.5	24.5	24.7	24.3	24.8	26.9	25.2	25.1
	d=12			6.2	15.8	21.8	24.9	24.1	24.5	24.6	24.4	24.7	26.5	24.9	24.9
	d=13			6.2	15.8	21.7	24.8	24.2	24.3	24.2	24.0	24.8	26.6	24.7	24.9
	d=14			6.3	15.8	21.7	24.7	24.2	24.1	24.0	23.5	24.7	26.5	24.7	24.2
	d=1	x		5.1	13.0	18.4	23.8	23.2	23.6	23.7	24.4	24.9	28.0	24.3	23.8
	d=2	x		6.0	14.6	20.1	24.7	24.1	24.5	24.7	25.2	25.5	28.2	26.3	24.9
	d=3	x		6.6	15.1	20.6	24.9	24.2	24.6	24.9	25.3	25.7	28.2	26.2	25.2
	d=4	x		6.9	15.4	20.7	24.9	24.3	24.6	25.0	25.4	25.9	28.2	26.4	25.3
	d=5	x		7.1	15.5	20.8	25.0	24.3	24.7	25.0	25.4	25.8	28.2	26.3	25.4
	d=6	x		7.2	15.5	20.8	24.9	24.3	24.6	25.0	25.4	25.8	28.2	26.2	25.5
	d=7	x		7.4	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.2	26.2	25.5
	d=8	x		7.4	15.6	20.9	24.9	24.3	24.6	24.9	25.5	25.8	28.3	26.2	25.6
	d=9	x		7.5	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.2	26.1	25.6
	d=10	x		7.6	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.2	26.2	25.5
	d=11	x		7.6	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.2	26.2	25.5
	d=12	x		7.7	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.4	26.2	25.6
	d=13	x		7.7	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.4	26.2	25.6
	d=14	x		7.7	15.6	20.8	24.9	24.3	24.6	24.9	25.5	25.8	28.4	26.2	25.6

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Stacking	LSTM			5.7	7.1	7.8	8.7	8.8	8.8	8.8	8.2	8.2	8.9	8.5	7.9
	LSTM	x		6.7	7.4	8.2	8.5	8.6	8.6	8.8	8.2	8.6	8.8	8.6	8.2
	FF			5.7	5.7	5.7	5.6	5.5	5.7	5.9	5.6	5.6	6.0	5.6	6.4
	FF	x		5.7	5.7	5.7	5.6	5.5	5.7	5.9	5.6	5.6	6.0	5.6	6.4
Stacking with Input	LSTM			6.4	6.6	6.6	6.4	6.4	6.4	6.8	6.5	6.6	7.0	6.8	7.1
	LSTM	x		6.4	6.9	6.9	6.5	6.4	6.4	6.8	6.5	6.6	7.1	6.8	7.1
	FF			5.8	5.9	5.8	5.7	5.7	5.5	5.5	5.5	5.9	5.2	5.4	5.4
	FF	x		5.8	5.9	5.8	5.7	5.7	5.5	5.5	5.5	5.9	5.2	5.4	5.4
Online Weighting	k=1			4.3	17.3	21.5	21.4	21.2	20.3	20.8	20.9	21.1	22.2	20.2	20.2
	k=5			4.5	17.7	20.4	20.3	20.0	19.1	19.5	19.3	19.3	20.7	19.5	19.8
	k=10			4.8	16.2	18.7	18.6	18.3	17.9	18.1	17.7	18.4	19.8	17.9	18.3
	k=20			5.1	13.9	16.0	16.0	15.9	15.4	15.6	15.1	15.9	16.5	15.1	15.3
	k=1	x		4.5	17.2	20.5	21.9	21.9	22.4	22.4	23.2	22.7	23.8	22.6	21.1
	k=5	x		4.9	17.3	19.5	20.6	20.5	20.7	20.7	21.4	21.0	21.6	21.4	20.4
	k=10	x		5.4	15.8	17.8	18.7	18.5	18.8	19.0	19.6	19.5	20.0	19.0	18.7
	k=20	x		5.7	13.5	15.3	16.0	16.1	15.9	16.4	16.4	16.2	16.9	16.6	15.8
	k=1		x	6.2	18.8	22.0	21.4	21.3	20.3	20.9	21.0	21.1	22.2	20.2	20.3
	k=5		x	6.2	19.6	21.2	20.4	20.1	19.1	19.5	19.3	19.3	20.7	19.5	19.8
	k=10		x	6.2	17.9	19.5	18.7	18.3	17.9	18.1	17.7	18.3	19.8	17.9	18.4
	k=20		x	6.2	15.4	16.5	16.1	15.9	15.5	15.6	15.1	15.9	16.5	15.1	15.3
	k=1	x	x	6.2	18.6	21.0	21.9	22.1	22.2	22.4	23.2	22.7	23.9	22.6	21.2
	k=5	x	x	6.2	19.0	20.2	20.7	20.5	20.8	20.7	21.4	21.0	21.6	21.4	20.9
	k=10	x	x	6.2	17.4	18.5	18.8	18.8	18.9	19.0	19.6	19.5	20.0	19.0	18.7
	k=20	x	x	6.2	14.6	15.8	16.1	16.1	16.2	16.4	16.5	16.2	16.9	16.6	16.1
	MSE			5.6	6.9	7.6	8.2	8.4	9.6	8.6	8.2	13.0	10.2	11.0	10.0
	MSE	x		5.0	7.2	7.8	8.3	8.6	10.1	8.9	8.8	14.3	11.2	10.3	10.8
	MSE		x	6.2	21.2	26.5	28.4	28.6	27.7	27.7	26.9	28.1	28.5	27.2	27.4
	MSE	x	x	6.2	20.9	25.8	29.1	29.4	29.6	29.7	30.0	29.9	30.4	29.4	29.2

continued on next page

continuation from previous page

Arch.	Cfg.	Reset		Stability period [#symbols]											
		S	W	1	5	10	50	100	200	400	600	800	1200	1600	2000
Model Averaging				6.2	10.0	10.9	11.2	10.8	10.6	10.2	10.0	10.5	10.7	10.6	11.0
		x		6.2	9.4	10.3	10.7	10.9	10.9	11.3	10.8	10.9	11.5	11.2	11.9
Sparse MoE (perfect id)	d=1			32.6	32.6	32.6	32.6	32.6	32.5	32.4	32.5	32.8	32.9	32.8	31.8
	d=1	x		20.6	28.7	30.8	32.0	32.0	32.2	32.5	32.7	33.1	33.2	32.9	32.0

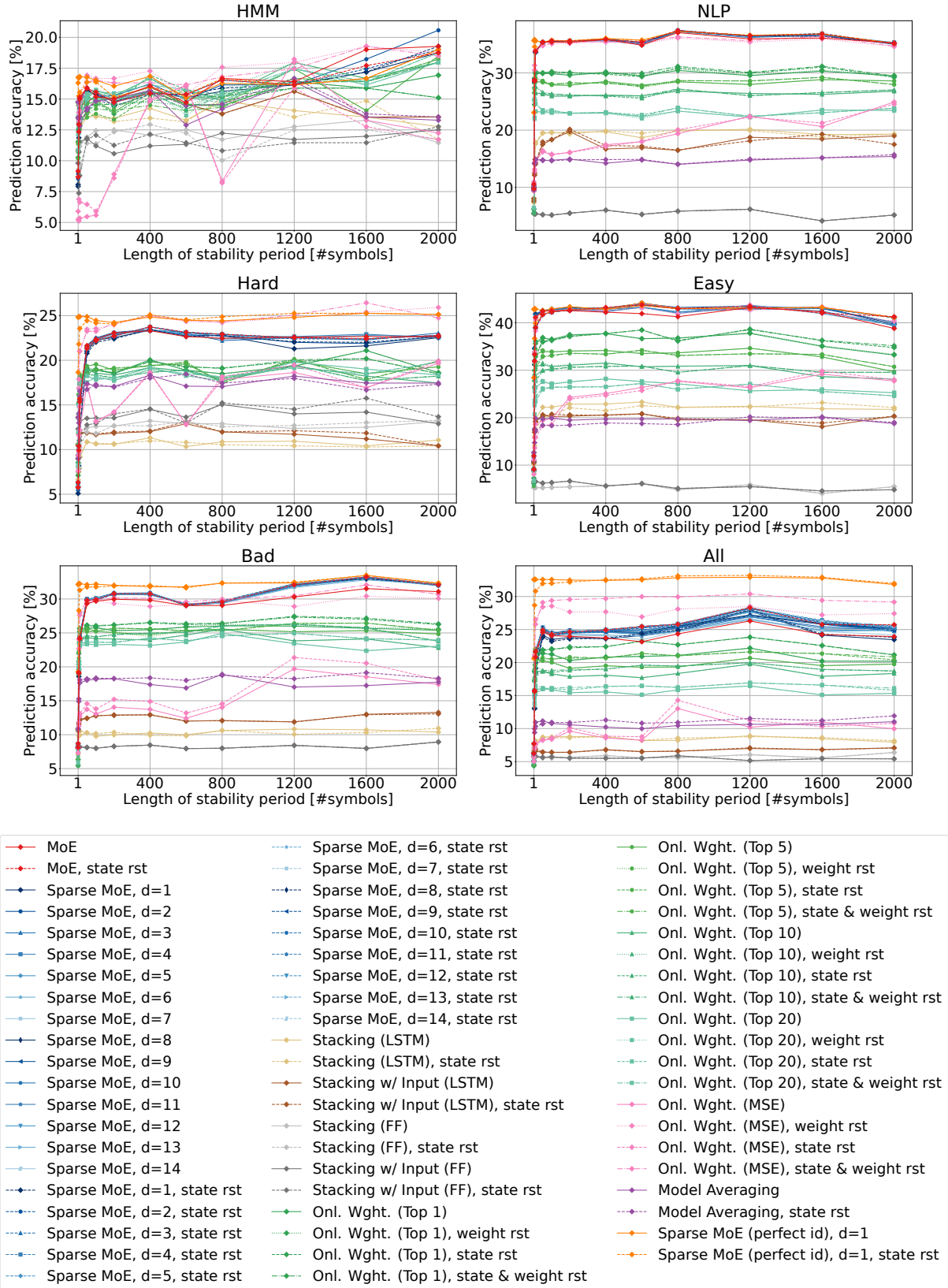


Figure A.3: Top 1 prediction accuracies of the different ensemble architectures [63].
FF = feedforward.

A.3 Recognition of Unknown Radar Emitters

This section contains the detailed values of the results presented in Chapter 8. The methods are sorted by highest distinction or identification accuracy, respectively.

Table A.7: Distinction, acceptance, and rejection accuracies in the test cases 1a to 5a at a sequence length of 1400 syllables.

Case	Method	Accuracy [%]		
		acc_{dist}	acc_{acpt}	acc_{rej}
1a	LSTM - Unknown Gate	100.00	100.00	100.00
	MC - Unknown Gate	100.00	99.99	100.00
	LSTM - Entropic Open Set	99.78	99.56	100.00
	MC	97.81	95.62	100.00
	LSTM - Cross-Entropy	96.06	92.12	100.00
	LSTM - Deep Open Class.	51.84	3.67	100.00
2a	LSTM - Unknown Gate	100.00	100.00	–
	MC - Unknown Gate	99.99	99.99	–
	LSTM - Entropic Open Set	99.62	99.62	–
	LSTM - Deep Open Class.	98.31	98.31	–
	MC	95.62	95.62	–
	LSTM - Cross-Entropy	93.13	93.13	–
3a	LSTM - Unknown Gate	66.67	100.00	33.33
	MC - Unknown Gate	66.66	99.99	33.33
	LSTM - Entropic Open Set	66.44	99.54	33.33
	MC	64.51	95.62	33.40
	LSTM - Cross-Entropy	62.62	92.91	32.32
	LSTM - Deep Open Class.	50.31	4.38	96.24
4a	MC	94.04	95.62	92.47
	LSTM - Entropic Open Set	70.23	99.53	40.93
	LSTM - Cross-Entropy	70.06	92.24	47.88
	LSTM - Unknown Gate	68.66	99.99	37.32
	MC - Unknown Gate	67.41	99.99	34.82
	LSTM - Deep Open Class.	49.97	0.72	99.21
5a	MC	75.57	95.62	55.52
	LSTM - Entropic Open Set	62.27	99.60	24.95
	LSTM - Unknown Gate	61.27	99.98	22.56
	LSTM - Cross-Entropy	61.27	93.28	29.25
	MC - Unknown Gate	60.44	99.98	20.89
	LSTM - Deep Open Class.	50.00	0.00	100.00

Table A.8: Distinction, acceptance, and rejection accuracies in the test cases 1b to 5b at a sequence length of 1400 syllables.

Case	Method	Accuracy [%]		
		acc_{dist}	acc_{acpt}	acc_{rej}
1b	MC	77.93	98.80	57.05
	MC - Unknown Gate	77.84	98.84	56.84
	LSTM - Cross-Entropy	75.04	99.79	50.30
	LSTM - Unknown Gate	75.00	99.86	50.15
	LSTM - Entropic Open Set	72.75	96.46	49.04
	LSTM - Deep Open Class.	71.22	65.31	77.14
2b	LSTM - Deep Open Class.	56.48	72.12	40.85
	MC	56.44	98.75	14.12
	MC - Unknown Gate	56.26	98.84	13.69
	LSTM - Unknown Gate	50.15	99.79	0.51
	LSTM - Cross-Entropy	50.11	99.95	0.26
	LSTM - Entropic Open Set	48.83	95.84	1.82
3b	LSTM - Cross-Entropy	73.06	98.15	47.96
	LSTM - Deep Open Class.	67.69	69.48	65.90
	LSTM - Unknown Gate	65.12	99.92	30.31
	MC	63.66	98.78	28.53
	MC - Unknown Gate	63.63	98.85	28.42
	LSTM - Entropic Open Set	60.20	96.07	24.34
4b	LSTM - Unknown Gate	79.19	99.77	58.62
	LSTM - Deep Open Class.	77.99	65.43	90.56
	MC	76.40	98.81	54.00
	LSTM - Entropic Open Set	74.63	96.51	52.74
	MC - Unknown Gate	73.94	98.84	49.04
	LSTM - Cross-Entropy	67.35	98.45	36.24
5b	LSTM - Deep Open Class.	76.60	72.24	80.97
	LSTM - Unknown Gate	73.04	99.86	46.22
	MC	67.40	98.76	36.04
	LSTM - Entropic Open Set	66.33	97.03	35.64
	MC - Unknown Gate	65.77	98.84	32.69
	LSTM - Cross-Entropy	62.69	99.86	25.52

Table A.9: Distinction, acceptance, and rejection accuracies in the test cases 1a to 5a at a sequence length of 1400 words.

Case	Method	Accuracy [%]		
		acc_{dist}	acc_{acpt}	acc_{rej}
1a	LSTM - Unknown Gate	100.00	100.00	100.00
	MC - Unknown Gate	99.93	99.85	100.00
	LSTM - Deep Open Class.	99.49	98.98	100.00
	LSTM - Cross-Entropy	99.46	98.91	100.00
	MC	99.02	98.05	100.00
	LSTM - Entropic Open Set	97.66	95.32	100.00
2a	LSTM - Unknown Gate	100.00	100.00	–
	MC - Unknown Gate	99.81	99.81	–
	LSTM - Cross-Entropy	99.38	99.38	–
	LSTM - Deep Open Class.	99.16	99.16	–
	MC	98.05	98.05	–
	LSTM - Entropic Open Set	96.35	96.35	–
3a	LSTM - Unknown Gate	66.67	100.00	33.33
	MC - Unknown Gate	66.58	99.82	33.33
	LSTM - Deep Open Class.	66.49	99.47	33.52
	LSTM - Cross-Entropy	66.27	99.20	33.33
	MC	65.69	98.05	33.33
	LSTM - Entropic Open Set	64.35	95.36	33.33
4a	MC	96.49	98.05	94.92
	LSTM - Entropic Open Set	95.42	95.17	95.68
	LSTM - Unknown Gate	95.08	99.98	90.17
	LSTM - Cross-Entropy	87.32	98.92	75.73
	LSTM - Deep Open Class.	86.65	99.37	73.94
	MC - Unknown Gate	81.22	99.82	62.61
5a	MC	77.50	98.05	56.95
	LSTM - Unknown Gate	77.45	100.00	54.91
	LSTM - Entropic Open Set	76.67	95.32	58.02
	LSTM - Cross-Entropy	72.71	99.08	46.35
	LSTM - Deep Open Class.	71.62	99.38	43.87
	MC - Unknown Gate	68.71	99.84	37.57

Table A.10: Distinction, acceptance, and rejection accuracies in the test cases 1b to 5b at a sequence length of 1400 words.

Case	Method	Accuracy [%]		
		acc_{dist}	acc_{acpt}	acc_{rej}
1b	LSTM - Cross-Entropy	87.61	91.10	84.11
	MC	80.35	96.44	64.26
	MC - Unknown Gate	79.99	96.70	63.28
	LSTM - Entropic Open Set	77.29	84.89	69.70
	LSTM - Deep Open Class.	75.06	99.52	50.59
	LSTM - Unknown Gate	74.88	99.23	50.53
2b	LSTM - Cross-Entropy	80.99	94.59	67.38
	LSTM - Entropic Open Set	62.71	86.61	38.81
	MC	62.38	96.40	28.35
	MC - Unknown Gate	61.63	96.70	26.56
	LSTM - Deep Open Class.	50.45	99.54	1.36
	LSTM - Unknown Gate	50.23	99.62	0.83
3b	LSTM - Entropic Open Set	72.93	86.22	59.64
	LSTM - Cross-Entropy	71.84	92.55	51.12
	MC - Unknown Gate	65.17	96.65	33.69
	MC	64.86	96.36	33.37
	LSTM - Deep Open Class.	62.46	99.79	25.13
	LSTM - Unknown Gate	62.45	99.70	25.19
4b	LSTM - Cross-Entropy	91.29	91.11	91.47
	MC	88.96	96.31	81.61
	MC - Unknown Gate	88.90	96.55	81.25
	LSTM - Unknown Gate	87.31	98.99	75.62
	LSTM - Entropic Open Set	85.34	85.89	84.79
	LSTM - Deep Open Class.	77.78	99.65	55.92
5b	LSTM - Cross-Entropy	82.72	93.17	72.28
	LSTM - Entropic Open Set	79.81	86.13	73.48
	MC - Unknown Gate	76.28	96.65	55.91
	MC	75.92	96.35	55.48
	LSTM - Unknown Gate	74.83	99.38	50.29
	LSTM - Deep Open Class.	68.33	99.79	36.87

Table A.11: Identification accuracies in the test cases 1a to 5a at a sequence length of 1400 syllables.

Case	Method	acc_{id} [%]
1a	MC	90.78
	LSTM - Cross-Entropy	69.35
	LSTM - Entropic Open Set	62.51
	LSTM - Deep Open Class.	0.76
2a	MC	90.80
	LSTM - Cross-Entropy	70.60
	LSTM - Entropic Open Set	55.14
	LSTM - Deep Open Class.	32.56
3a	MC	90.81
	LSTM - Cross-Entropy	70.50
	LSTM - Entropic Open Set	56.63
	LSTM - Deep Open Class.	1.59
4a	MC	90.81
	LSTM - Cross-Entropy	69.54
	LSTM - Entropic Open Set	62.75
	LSTM - Deep Open Class.	0.72
5a	MC	90.83
	LSTM - Cross-Entropy	70.24
	LSTM - Entropic Open Set	61.62
	LSTM - Deep Open Class.	0.00

Table A.12: Identification accuracies in the test cases 1b to 5b at a sequence length of 1400 syllables.

Case	Method	acc_{id} [%]
1b	LSTM - Entropic Open Set	100.00
	LSTM - Cross-Entropy	99.70
	MC	99.69
	LSTM - Deep Open Class.	64.87
2b	LSTM - Entropic Open Set	100.00
	LSTM - Cross-Entropy	99.69
	MC	99.65
	LSTM - Deep Open Class.	68.06
3b	LSTM - Entropic Open Set	100.00
	LSTM - Cross-Entropy	99.80
	MC	99.66
	LSTM - Deep Open Class.	69.34
4b	LSTM - Entropic Open Set	100.00
	MC	99.67
	LSTM - Cross-Entropy	99.66
	LSTM - Deep Open Class.	65.41
5b	LSTM - Entropic Open Set	100.00
	LSTM - Cross-Entropy	99.70
	MC	99.68
	LSTM - Deep Open Class.	71.64

Table A.13: Identification accuracies in the test cases 1a to 5a at a sequence length of 1400 words.

Case	Method	acc_{id} [%]
1a	LSTM - Cross-Entropy	95.69
	MC	91.65
	LSTM - Entropic Open Set	61.94
	LSTM - Deep Open Class.	41.89
2a	LSTM - Cross-Entropy	95.18
	MC	91.58
	LSTM - Entropic Open Set	62.53
	LSTM - Deep Open Class.	38.57
3a	LSTM - Cross-Entropy	96.38
	MC	91.54
	LSTM - Entropic Open Set	62.30
	LSTM - Deep Open Class.	39.33
4a	LSTM - Cross-Entropy	96.30
	MC	91.47
	LSTM - Entropic Open Set	62.28
	LSTM - Deep Open Class.	41.33
5a	LSTM - Cross-Entropy	96.52
	MC	91.53
	LSTM - Entropic Open Set	62.14
	LSTM - Deep Open Class.	44.07

Table A.14: Identification accuracies in the test cases 1b to 5b at a sequence length of 1400 words.

Case	Method	acc_{id} [%]
1b	LSTM - Entropic Open Set	99.86
	MC	99.07
	LSTM - Cross-Entropy	95.39
	LSTM - Deep Open Class.	50.03
2b	LSTM - Entropic Open Set	99.81
	MC	99.16
	LSTM - Cross-Entropy	87.71
	LSTM - Deep Open Class.	50.14
3b	LSTM - Entropic Open Set	99.79
	MC	99.20
	LSTM - Cross-Entropy	95.00
	LSTM - Deep Open Class.	50.17
4b	LSTM - Entropic Open Set	99.81
	MC	99.41
	LSTM - Cross-Entropy	96.32
	LSTM - Deep Open Class.	50.06
5b	LSTM - Entropic Open Set	99.68
	MC	98.94
	LSTM - Cross-Entropy	97.05
	LSTM - Deep Open Class.	50.14

List of Acronyms

AESA	Active electronically scanned array
Amp	Amplitude
AOA	Angle of arrival
BW	Bandwidth
CBOW	Continuous bag-of-words
COMINT	Communication intelligence
CPI	Coherent processing interval
DES	Discrete event system
ELINT	Electronic intelligence
EM	Expectation-maximisation
EW	Electronic warfare
FSM	Finite state machine
GRU	Gated Recurrent Unit
HMM	Hidden Markov model
HPRF	High pulse repetition frequency
LSTM	Long Short-Term Memory
MC	Markov chain
MoE	Mixture of Experts
MOP	Modulation on pulse
MPRF	Medium pulse repetition frequency
MSE	Mean squared error
MTI	Moving target indication
NLP	Natural language processing
OOM	Observable operator model
PDW	Pulse descriptor word
PRF	Pulse repetition frequency

PRI	Pulse repetition interval
PSR	Predictive state representation
PW	Pulse width
QoS	Quality of Service
Radar	Radio detection and ranging
ReLU	Rectified linear unit
RF	Radio frequency
RNN	Recurrent neural network
SPiCe	Sequence Prediction Challenge
TOA	Time of arrival

List of Symbols

A	Transition probability matrix of a hidden Markov model (HMM)
\mathcal{A}	Observable operator model (OOM)
A	Set of arcs in a Petri net
b	Bias
B	Emission probability matrix of an HMM
B_S	Signal bandwidth
\mathcal{B}_t	Batch of symbols at index t
c	Speed of light, $c = 299\,792\,458\,\text{m s}^{-1}$
χ	Empty word
C	Number of classes
D	System-dynamics matrix of a predictive state representation (PSR)
ΔR	Range resolution
η	Learning rate
E	Error
\mathbb{E}	Expected value
\mathcal{E}	Set of emitters
E	Set of experts
f	Activation function in a neural network
f_{χ}	State transition function
$f_{\mathcal{T}}$	Transition function of a Petri net
f_d	Doppler shift
\mathcal{F}	Set of events in a finite state machine (FSM)
G	Formal grammar
G_p	Petri net graph
G_p^m	Marked Petri net graph
H	Deterministic FSM

\mathcal{H}	Set of core histories in PSRs
\mathbb{H}	Cross-entropy loss
id	Identity matrix
i, j	General indices
k	Variable for top k accuracy
\mathbb{K}	Set of known emitters
l	Modelling level, $l \in \{\text{letters, syllables, words, commands, functions}\}$
λ	HMM
μ	OOM matrix
\mathcal{N}	Set of nonterminal symbols
ω	Symbol
Ω^l	Set of symbols at modelling level l
π	Permutation
π	Probability distribution of the initial state of an HMM
P	Matrix or vector of probability distributions
\mathbb{P}	Set of production rules of a formal grammar
$\mathcal{P}(E)$	Power set of E
\mathbb{P}	Set of places in a Petri net
P, Q	Probability distributions
\hat{P}	Estimated probability distribution
\mathcal{Q}	Set of core tests in PSRs
R	Range
R_u	Maximum unambiguous range
s	Input sequence
$\sigma(x)$	Sigmoid function
S	Start symbol of a formal grammar
\mathcal{S}	Set of sequence lengths
S	Scenario
Σ	Set of terminal symbols of a formal grammar
t	Index in a time series
t_e, t_r	Emission and reception time
τ_v	Linear operators of a OOM
T	Operator in HMM view
\mathbb{T}	Set of transitions of a Petri net

T_k	Top k elements
u	Weight
\mathbf{U}	Set of unknown unknown emitters
V	Stream of one-hot encoded vectors
\mathbf{V}	Set of known unknown emitters
v_r	Relative velocity
v_u	Maximum unambiguous velocity
v_w	Word vector of word w
w	Word
\mathbf{W}	Weight matrix
x	Input vector
\mathbf{x}	Marking of the places in a Petri net
\mathcal{X}	Set of states
\mathcal{X}_m	Set of accepting states
$\{X_t\}_{t \in \mathcal{T}}$	Family of random variables indexed by \mathcal{T}
y	Label, i.e. the desired output
\hat{y}	Estimated label
\mathcal{Y}	Set of labels
$\hat{\mathcal{Y}}$	Set of estimated labels
\mathcal{Y}	Alphabet of the observations
$\{Y_t\}$	Set of emissions of an HMM

List of Figures

1.1	The evolution of radars from non-agile to agile.	1
2.1	The traditional ELINT processing chain for emitter identification.	10
2.2	Visualisation of the waveform parameters TOA, RF, PW, and PRI.	10
2.3	Schematic representation of the deinterleaving process.	11
2.4	Comparison of the delta-T and the complex delta-T histogram for finding possible pulse repetition intervals (PRIs) in a stream of PDWs.	12
2.5	Examples of different PRI modulation types.	13
2.6	The processing chain for emitter identification and emission prediction proposed in this thesis.	15
3.1	Visualisation of the Chomsky hierarchy for classes of formal grammars.	18
3.2	A state diagram of an FSM.	20
3.3	Example of a Petri net graph in different states.	23
3.4	The relationship between Petri net languages and the Chomsky hierarchy.	23
3.5	Example of an MC that describes the transition probabilities of the weather between “sunny”, “cloudy”, and “rainy”.	25
3.6	Example of an HMM based on the MC from Figure 3.5.	26
3.7	The HMM corresponding to the stochastic grammar defined in (3.6) to (3.11).	30
3.8	The HMM view of the model trajectory as a sequence of states and the OOM view as a sequence of linear operators [88].	31
3.9	A simple feedforward neural network with one hidden layer.	37
3.10	Examples for activation functions.	38
3.11	A function and its gradient.	40
3.12	RNNs have connections which form a feedback from their own output to their input [94].	43
3.13	An RNN unrolled over time [94].	43
3.14	Internal structure of an LSTM cell, unrolled over three time steps [94].	44
3.15	Internal structure of a Gated Recurrent Unit (GRU) cell, unrolled over three time steps.	46
4.1	Hierarchical model of the emissions of a multifunction radar.	47
4.2	Examples for the numerical representation of words using one-hot encoding and word embedding.	49

4.3	The architectures for learning word embeddings contained in word2vec.	50
4.4	Example for modelling the different PRI levels as syllables [60].	51
4.5	The adapted hierarchical emission model.	52
5.1	General architecture of the networks for prediction.	63
5.2	Prediction accuracies of the different methods.	66
5.3	With a missing or additional symbol, at least one or two prediction errors, respectively, are expected to be made.	68
5.4	Comparison of the impact of missing and additional syllables and words on the top 1 accuracies for the Quality of Service (QoS) radar.	71
5.5	Course of training and validation loss for syllables of the QoS radar with word embedding and one-hot encoding [60].	72
6.1	The input to the identification method is a stream of sequences that consist of deinterleaved pulses from potentially different emitters. . . .	75
6.2	Symbols are extracted from the deinterleaved PDW sequences, which are then input to the identification method.	78
6.3	General architecture of the networks for identification.	79
6.4	Identification accuracies of the different methods on the test data of all emitters.	83
6.5	Confusion matrices of the LSTM _{scen} and the MC at a sequence length of 1400 syllables.	84
6.6	Confusion matrices at different sequence lengths of the LSTM _{rand} for words [61].	85
6.7	Confusion matrices at different sequence lengths of the MC for words. .	85
6.8	Transition matrices for the commands of the different emitter types. . .	86
6.9	Identification accuracies of the LSTM _{scen} , the MC, and the MC _ε with missing or additional symbols.	89
6.10	Identification accuracies of the different MC variants with missing and additional symbols.	91
7.1	Mixture of Experts [63].	95
7.2	Sparsely-Gated Mixture of Experts [63].	96
7.3	The two variants of the stacking architecture [63].	97
7.4	Online Accuracy-Based Weighting [63].	98
7.5	Model Averaging [63].	98
7.6	The general architecture of the experts [63].	100
7.7	The general architecture of the feedforward combiner [63].	102
7.8	Top 1 identification accuracies of the gating networks used in the MoE variants for the different scenarios [63].	106
7.9	Top 1 prediction accuracies of the MoE variants and the mean squared error (MSE)-based weighting [63].	107
7.10	Top 1 prediction accuracies of the stacking variants, the online weighting based on top k accuracy, the model averaging, and the sparsely-gated MoE with perfect identification [63].	108

7.11	Comparison of the top 1 prediction accuracies of the different stacking variants for the <i>hard</i> scenario [63].	109
7.12	Top 1 prediction accuracies of the MoE architecture for the <i>bad</i> and <i>all</i> scenario with and without state reset [63].	110
7.13	Top 1 prediction accuracy of the sparsely-gated MoE with $d = 1$ for the <i>easy</i> scenario with and without state reset [63].	110
7.14	Prediction accuracies of the emitter models on the data of all emitters [64].	112
7.15	Top 1 prediction accuracy of the different ensemble architectures with MC experts for syllables [64].	113
7.16	Top 1 prediction accuracy of the different ensemble architectures with LSTM experts for words [64].	115
7.17	Top 1 prediction accuracy of the different ensemble architectures with MC experts for syllables and 20 % corrupted data at a sequence length of 1400 symbols [64].	117
7.18	Top 1 prediction accuracy of the different ensemble architectures with LSTM experts for words and 20 % corrupted data at a sequence length of 1400 symbols [64].	117
8.1	Dependency of the distinction accuracy on the threshold at a sequence length of 1400 symbols [65].	135
8.2	Mean distinction accuracies of the best configurations [65].	136
8.3	Comparison of the mean identification accuracies in the test cases ‘a’ with training case 0 or I at a sequence length of 1400 symbols [65]. . . .	137
8.4	Dependency of the identification accuracy on the threshold at a sequence length of 1400 symbols [65].	138
8.5	Mean identification accuracies of the best configurations [65].	138
8.6	Rejection and acceptance accuracies for the different test cases at a sequence length of 1400 symbols [65].	139
8.7	Rejection accuracies for the unknown emitters in the test cases ‘a’ at a sequence length of 1400 syllables [65].	140
8.8	Rejection accuracies for the emitters UNKs, Unk-1, Unk-2, Unk-3, and Unk-4 in the test cases ‘b’ at a sequence length of 1400 syllables [65]. . .	141
8.9	Rejection accuracies for the unknown emitters in the test cases ‘a’ at a sequence length of 1400 words [65].	142
8.10	Rejection accuracies for the emitters UNKs, Unk-1, Unk-2, Unk-3, and Unk-4 in the test cases ‘b’ at a sequence length of 1400 words [65]. . . .	143
8.11	Rejection accuracies for the Rules-v2 emitter at a sequence length of 1400 words or syllables [65].	144
8.12	Mean distinction accuracies of the best configurations with 20 % missing or additional syllables, respectively [65].	148
8.13	Mean distinction accuracies of the best configurations with 20 % missing or additional words, respectively [65].	148
8.14	Mean identification accuracies of the best configurations with 20 % missing or additional syllables, respectively [65].	149

8.15	Mean identification accuracies of the best configurations with 20 % missing or additional words, respectively [65].	149
A.1	Confusion matrices of the LSTM _{scen} at different sequence lengths for syllables.	157
A.2	Confusion matrices of the MC at different sequence lengths for syllables.	158
A.3	Top 1 prediction accuracies of the different ensemble architectures [63].	172

List of Tables

2.1	Purpose of different PRI modulation types [1].	13
2.2	Simplified example of an emitter mode database.	14
3.1	Chomsky hierarchy of formal grammars with suitable models.	20
4.1	Possible combinations of words of the “Mercury” emitter and the connection to commands [20].	48
4.2	Number of symbols used by the different resource management methods.	53
4.3	Letters of the example radar.	54
4.4	Syllables of the example radar.	54
4.5	Words and commands of the example radar.	55
4.6	word2vec parameters used. Values for the QoS and Rules-v2 radars are shown before and those for the Rules-v1 after the slash. Batch size and skip window are the same for all.	56
5.1	Training, validation, and test set per emitter type.	61
5.2	Amount of data in the sets of each emitter.	62
5.3	LSTM parameters for the different modelling levels of the example radars.	63
5.4	Prediction accuracies [%] of the MCs, the LSTMs, and the repeating strategy. The best results are marked in bold.	67
5.5	Prediction accuracies [%] of the MCs, the LSTMs, and the strategy that predicts the most frequent symbols. The best results are marked in bold.	68
5.6	Mean \pm standard deviation of the relative top 1 prediction accuracy $\overline{\text{acc}}_{rel}(\mathcal{E})$ [%] with missing or additional syllables.	69
5.7	Mean \pm standard deviation of the relative top 1 prediction accuracy $\overline{\text{acc}}_{rel}(\mathcal{E})$ [%] with missing or additional words.	70
6.1	“Overlap matrix” for syllables [%]. Rv1 = Rules-v1, Rv2 = Rules-v2. . .	78
6.2	“Overlap matrix” for words [%]. Rv1 = Rules-v1, Rv2 = Rules-v2. . . .	78
6.3	LSTM parameters for the different modelling levels and network types.	79
6.4	Identification accuracies [%] of the MCs and LSTMs at different sequence lengths for selected modelling levels.	84
6.5	Mean \pm standard deviation of the relative top 1 identification accuracy $\overline{\text{acc}}_{rel}(\mathcal{S})$ [%] with missing or additional syllables.	88

6.6	Mean \pm standard deviation of the relative top 1 identification accuracy $\overline{\text{acc}}_{\text{rel}}(\mathcal{S})$ [%] with missing or additional words.	90
7.1	Overview of the problems defined in Sequence Prediction Challenge (SPiCe).	99
7.2	Definition of the scenarios as subsets of the problems.	99
7.3	LSTM parameters of the experts.	100
7.4	LSTM parameters of the gating networks.	101
7.5	Parameters of the LSTM combiner.	103
7.6	Parameters of the feedforward combiner.	104
7.7	Top 1 prediction accuracy [%] of the ensemble architectures in different configurations with MC experts for syllables.	114
7.8	Top 1 prediction accuracy [%] of the ensemble architectures in different configurations with LSTM experts for words.	116
8.1	Characterisation of classification, anomaly detection, and open-set recognition [55].	121
8.2	Training cases employed for unknown emitter recognition.	125
8.3	Overview of the methods employed for unknown emitter recognition.	127
8.4	Parameters of the LSTM architecture with cross-entropy loss.	128
8.5	Parameters of the LSTM architecture with entropic open-set loss.	129
8.6	Parameters of the LSTM architecture with deep open classification loss.	130
8.7	Parameters of the LSTM architecture employed as unknown gate.	130
8.8	Test cases for classifiers trained with cases 0 or I.	132
8.9	Test cases for classifiers trained with cases II to V.	132
8.10	Configurations (training case, δ) that achieve the highest distinction accuracies, averaged over the test cases 1a to 5a and 1b to 5b, respectively.	134
8.11	Configurations (training case, δ) that achieve the highest identification accuracies, averaged over the test cases 1a to 5a and 1b to 5b, respectively.	137
8.12	Evaluation matrices at a sequence length of 1400 syllables.	145
8.13	Configurations (training case, δ) that achieve the highest mean of distinction and identification accuracy, averaged over the test cases 1a to 5a and 1b to 5b, respectively.	146
8.14	Evaluation matrices at a sequence length of 1400 words.	146
A.1	Top 1 prediction accuracies [%] for the HMM scenario.	159
A.2	Top 1 prediction accuracies [%] for the NLP scenario.	161
A.3	Top 1 prediction accuracies [%] for the hard scenario.	163
A.4	Top 1 prediction accuracies [%] for the easy scenario.	165
A.5	Top 1 prediction accuracies [%] for the bad scenario.	167
A.6	Top 1 prediction accuracies [%] for the scenario containing all problems.	169
A.7	Distinction, acceptance, and rejection accuracies in the test cases 1a to 5a at a sequence length of 1400 syllables.	173
A.8	Distinction, acceptance, and rejection accuracies in the test cases 1b to 5b at a sequence length of 1400 syllables.	174

A.9	Distinction, acceptance, and rejection accuracies in the test cases 1a to 5a at a sequence length of 1400 words.	175
A.10	Distinction, acceptance, and rejection accuracies in the test cases 1b to 5b at a sequence length of 1400 words.	176
A.11	Identification accuracies in the test cases 1a to 5a at a sequence length of 1400 syllables.	177
A.12	Identification accuracies in the test cases 1b to 5b at a sequence length of 1400 syllables.	178
A.13	Identification accuracies in the test cases 1a to 5a at a sequence length of 1400 words.	179
A.14	Identification accuracies in the test cases 1b to 5b at a sequence length of 1400 words.	180

List of Publications

Published

S. Apfeld, A. Charlish, and W. Koch, "An Adaptive Receiver Search Strategy for Electronic Support," *Sensor Signal Processing for Defence (SSPD)*, 2016

—, "Quality of Service Resource Management for Search Strategy Design in Electronic Support," *Sensor Signal Processing for Defence (SSPD)*, 2017

F. Katsilieris, S. Apfeld, and A. Charlish, "Correlation based classification of complex PRI modulation types," *Sensor Signal Processing for Defence (SSPD)*, 2017

S. Apfeld, A. Charlish, and G. Ascheid, "Modelling, Learning and Prediction of Complex Radar Emitter Behaviour," *IEEE International Conference on Machine Learning and Applications*, 2019

—, "Identification of Radar Emitter Type with Recurrent Neural Networks," *Sensor Signal Processing for Defence (SSPD)*, 2020

—, "The Value of Memory: Markov Chain versus Long Short-Term Memory for Electronic Intelligence," *IEEE Radar Conference*, 2021

S. Apfeld and A. Charlish, "Recognition of Unknown Radar Emitters with Machine Learning," *IEEE Transactions on Aerospace and Electronic Systems (Early Access)*, 2021

Accepted for Publication

S. Apfeld, A. Charlish, and G. Ascheid, "Ensembles of Long Short-Term Memory Experts for Streaming Data with Sudden Concept Drift," *IEEE International Conference on Machine Learning and Applications (accepted)*, 2021

Under Review

S. Apfeld and A. Charlish, "Ensembles of Predictive Radar Models for Electronic Intelligence," *IEEE Radar Conference (under review)*, 2022

Bibliography

- [1] R. Wiley, *ELINT: The Interception and Analysis of Radar Signals*. Artech House, 2006.
- [2] S. P. Sira, Y. Li, A. Papandreou-Suppappola, D. Morrell, D. Cochran, and M. Rangaswamy, "Waveform-agile sensing for tracking," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 53–64, 2009.
- [3] A. O. Hero and D. Cochran, "Sensor management: Past, present, and future," *IEEE Sensors Journal*, vol. 11, no. 12, pp. 3064–3075, 2011.
- [4] A. Charlish, "Autonomous agents for multi-function radar resource management," Ph.D. dissertation, University College London, 2011.
- [5] F. Katsilieris, "Sensor management for surveillance and tracking," Ph.D. dissertation, TU Delft, 2015.
- [6] A. Charlish and F. Hoffmann, "Cognitive radar management," in *Novel Radar Techniques and Applications Volume 2: Waveform Diversity and Cognitive Radar, and Target Tracking and Data Fusion*. Institution of Engineering and Technology, 2017, pp. 157–193.
- [7] M. Skolnik, *Radar Handbook*, 3rd ed. McGraw-Hill, 2008.
- [8] M. A. Richards, J. A. Scheer, and W. A. Holm, *Principles of Modern Radar: Basic principles*. Institution of Engineering and Technology, 2010.
- [9] T. M. Mitchell, *Machine Learning*. McGraw-Hill, Inc., 1997.
- [10] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008.
- [11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [12] N. Visnevski, V. Krishnamurthy, S. Haykin, B. Currie, F. Dilkes, and P. Lavoie, "Multi-Function Radar Emitter Modelling: A Stochastic Discrete Event System Approach," *IEEE Conference on Decision and Control*, 2003.

- [13] N. A. Visnevski, F. A. Dilkes, S. Haykin, and V. Krishnamurthy, "Non-self-embedding context-free grammars for multi-function radar modeling - electronic warfare application," *IEEE International Radar Conference*, 2005.
- [14] N. A. Visnevski, S. Haykin, V. Krishnamurthy, F. A. Dilkes, and P. Lavoie, "Hidden Markov models for radar pulse train analysis in electronic warfare," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005.
- [15] N. A. Visnevski, "Syntactic Modeling of Multi-Function Radars," Ph.D. dissertation, McMaster University, 2005.
- [16] I. Arasaratnam, S. Haykin, T. Kirubarajan, and F. Dilkes, "Tracking the Mode of Operation of Multi-Function Radars," *IEEE Conference on Radar*, 2006.
- [17] I. Arasaratnam, "Tracking the mode of operation of multi-function radars," Master's thesis, McMaster University, 2006.
- [18] N. A. Visnevski, V. Krishnamurthy, A. Wang, and S. Haykin, "Syntactic modeling and signal processing of multifunction radars: A stochastic context-free grammar approach," *Proceedings of the IEEE*, vol. 95, no. 5, pp. 1000–1025, 2007.
- [19] A. Wang and V. Krishnamurthy, "Threat estimation of multifunction radars: Modeling and statistical signal processing of stochastic context free grammars," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2007.
- [20] —, "Signal Interpretation of Multifunction Radars: Modeling and Statistical Signal Processing With Stochastic Context Free Grammar," *IEEE Transactions on Signal Processing*, vol. 56, no. 3, pp. 1106–1119, 2008.
- [21] —, "Modeling and interpretation of multifunction radars with stochastic grammar," *IEEE Aerospace Conference*, 2008.
- [22] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.
- [23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *International Conference on Learning Representations (ICLR)*, 2013.
- [24] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *Advances in Neural Information Processing Systems (NIPS)*, 2013.
- [25] T. Mikolov, W.-T. Yih, and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations," *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013.

- [26] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," *Conference on Empirical Methods in Natural Language Processing*, 2014.
- [27] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.
- [28] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.
- [29] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining Concept-Drifting Data Streams Using Ensemble Classifier," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.
- [30] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.
- [31] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 81–94, 2014.
- [32] Y. Lu, Y.-M. Cheung, and Y. Y. Tang, "Adaptive Chunk-Based Dynamic Weighted Majority for Imbalanced Data Streams With Concept Drift," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 8, 2020.
- [33] F. Hinder, A. Artelt, and B. Hammer, "Towards non-parametric drift detection via Dynamic Adapting Window Independence Drift Detection (DAWIDD)," *International Conference on Machine Learning (ICML)*, 2020.
- [34] J. Ou, Y. Chen, F. Zhao, J. Liu, and S. Xiao, "Novel Approach for the Recognition and Prediction of Multi-Function Radar Behaviours Based on Predictive State Representations," *Sensors*, vol. 17, no. 3, mar 2017.
- [35] M. L. Littman, R. S. Sutton, and S. Singh, "Predictive Representations of State," *Advances in Neural Information Processing Systems (NIPS)*, 2001.
- [36] Z.-M. Liu and P. S. Yu, "Classification, Denoising, and Deinterleaving of Pulse Streams With Recurrent Neural Networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 4, pp. 1624–1639, 2019.
- [37] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Conference on Empirical Methods in Natural Language Processing*, 2014.

- [38] S. A. Shapero, A. B. Dill, and B. O. Odelowo, "Identifying Agile Waveforms with Neural Networks," in *International Conference on Information Fusion (FUSION)*, 2018.
- [39] P. Notaro, M. Paschali, C. Hopke, D. Wittmann, and N. Navab, "Radar Emitter Classification with Attribute-specific Recurrent Neural Networks," *pre-print arXiv: 1911.07683*, 2019.
- [40] Z. M. Liu, "Recognition of Multifunction Radars Via Hierarchically Mining and Exploiting Pulse Group Patterns," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 6, pp. 4659–4672, 2020.
- [41] X. Li, Z. Liu, Z. Huang, and W. Liu, "Radar Emitter Classification with Attention-Based Multi-RNNs," *IEEE Communications Letters*, vol. 24, no. 9, pp. 2000–2004, 2020.
- [42] W. Zhang, X. Yin, X. Cao, Y. Xie, and W. Nie, "Radar Emitter Identification Using Hidden Markov Model," *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference*, 2019.
- [43] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.
- [44] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, pp. 1–39, 2010.
- [45] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.
- [46] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," *International Conference on Learning Representations (ICLR)*, 2017.
- [47] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.
- [48] G. Giacinto and F. Roli, "Methods for dynamic classifier selection," *International Conference on Image Analysis and Processing*, 1999.
- [49] A. H. Ko, R. Sabourin, and A. S. Britto, "From dynamic classifier selection to dynamic ensemble selection," *Pattern Recognition*, vol. 41, no. 5, 2008.
- [50] B. Balle, F. M. Luque, A. Quattoni, and S. Verwer, "Results of the Sequence Prediction Challenge (SPiCe): a Competition on Learning the Next Symbol in a Sequence," *International Conference on Grammatical Inference*, 2016.
- [51] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boulton, "Toward Open Set Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013.

- [52] W. J. Scheirer, L. P. Jain, and T. E. Boulton, "Probability Models for Open Set Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.
- [53] A. Bendale and T. E. Boulton, "Towards Open Set Deep Networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1563–1572, 2016.
- [54] L. Shu, H. Xu, and B. Liu, "DOC: Deep Open Classification of Text Documents," *Conference on Empirical Methods in Natural Language Processing*, 2017.
- [55] D. O. Cardoso, J. Gama, and F. M. França, "Weightless neural networks for open set recognition," *Machine Learning*, vol. 106, no. 9-10, pp. 1547–1567, 2017.
- [56] A. R. Dhamija, M. Günther, and T. E. Boulton, "Reducing Network Agnostophobia," *Neural Information Processing Systems (NeurIPS)*, 2018.
- [57] L. Anjaneyulu, N. Murthy, and N. Sarma, "Radar emitter classification using self-organising neural network models," *International Conference of Recent Advances in Microwave Theory and Applications*, 2008.
- [58] P. Fitch, "Pulse signal and source identification using fuzzy-neural techniques," *IEEE Aerospace and Electronic Systems Magazine*, vol. 28, no. 1, pp. 22–33, 2013.
- [59] L. S. Kim, H. B. Bae, R. M. Kil, and C. H. Jo, "Classification of the trained and untrained emitter types based on class probability output networks," *Neurocomputing*, vol. 248, no. C, pp. 67–75, jul 2017.
- [60] S. Apfeld, A. Charlish, and G. Ascheid, "Modelling, Learning and Prediction of Complex Radar Emitter Behaviour," *IEEE International Conference on Machine Learning and Applications*, 2019.
- [61] —, "Identification of Radar Emitter Type with Recurrent Neural Networks," *Sensor Signal Processing for Defence (SSPD)*, 2020.
- [62] —, "The Value of Memory: Markov Chain versus Long Short-Term Memory for Electronic Intelligence," *IEEE Radar Conference*, 2021.
- [63] —, "Ensembles of Long Short-Term Memory Experts for Streaming Data with Sudden Concept Drift," *IEEE International Conference on Machine Learning and Applications (accepted)*, 2021.
- [64] S. Apfeld and A. Charlish, "Ensembles of Predictive Radar Models for Electronic Intelligence," *IEEE Radar Conference (under review)*, 2022.
- [65] —, "Recognition of Unknown Radar Emitters with Machine Learning," *IEEE Transactions on Aerospace and Electronic Systems (Early Access)*, 2021.
- [66] S. Apfeld, A. Charlish, and W. Koch, "An Adaptive Receiver Search Strategy for Electronic Support," *Sensor Signal Processing for Defence (SSPD)*, 2016.

- [67] —, “Quality of Service Resource Management for Search Strategy Design in Electronic Support,” *Sensor Signal Processing for Defence (SSPD)*, 2017.
- [68] F. Katsilieris, S. Apfeld, and A. Charlish, “Correlation based classification of complex PRI modulation types,” *Sensor Signal Processing for Defence (SSPD)*, 2017.
- [69] A. Charlish and F. Katsilieris, “Array radar resource management,” in *Novel Radar Techniques and Applications Volume 1: Real Aperture Array Radar, Imaging Radar, and Passive and Multistatic Radar*. Institution of Engineering and Technology, 2017, pp. 135–171.
- [70] B. Dutertre, “Dynamic Scan Scheduling,” *IEEE Real-Time Systems Symposium*, 2002.
- [71] I. V. L. Clarkson, E. D. El-Mahassni, and S. D. Howard, “Sensor scheduling in electronic support using Markov chains,” *IEE Proceedings - Radar Sonar and Navigation*, vol. 153, no. 4, pp. 325–332, 2006.
- [72] I. V. L. Clarkson, “Optimisation of Periodic Search Strategies for Electronic Support,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 3, pp. 1170–1784, 2011.
- [73] C. Winsor and E. J. Hughes, “Optimisation and evaluation of receiver search strategies for electronic support,” *IET Radar, Sonar & Navigation*, vol. 6, no. 4, 2012.
- [74] K. Nishiguchi and M. Kobayashi, “Improved algorithm for estimating pulse repetition intervals,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 2, pp. 407–421, 2000.
- [75] C. Davies and P. Hollands, “Automatic processing for ESM,” *IEE Proceedings F (Communications, Radar and Signal Processing)*, vol. 129, no. 3, p. 164, 1982.
- [76] Y.-J. J. Ryoo, K.-H. H. Song, and W.-W. W. Kim, “Recognition of PRI Modulation Types of Radar Signals Using the Autocorrelation,” *IEICE Transactions on Communication*, vol. E90-B, no. 5, pp. 1290–1294, 2007.
- [77] M. Ahmadi and K. Mohamedpour, “A new method for recognizing pulse repetition interval modulation,” *International Conference on Signal Processing Systems*, 2009.
- [78] A. Mahdavi and A. M. Pezeshk, “A robust method for PRI modulation recognition,” *International Conference on Signal Processing*, 2010.
- [79] G. Hu and Y. Liu, “An efficient method of pulse repetition interval modulation recognition,” *International Conference on Communications and Mobile Computing*, 2010.

- [80] P. Lavoie, "Hidden Markov Modeling for Radar Electronic Warfare," Tech. Rep., 2001.
- [81] N. Chomsky, "On certain formal properties of grammars," *Information and Control*, vol. 2, no. 2, pp. 137–167, 1959.
- [82] G. Wachsmuth and E. Visser, "Formal Grammars - language specification," *Lecture Slides*, TU Delft, 2017.
- [83] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.
- [84] N. Chomsky, "A note on phrase structure grammars," *Information and Control*, vol. 2, no. 4, pp. 393–395, 1959.
- [85] M. Parigot and E. Pelz, "A logical Approach of Petri Net Languages," *Theoretical Computer Science*, vol. 39, pp. 155–169, 1985.
- [86] J. L. Peterson, *Petri net theory and the modeling of systems*. Prentice Hall, Englewood Cliffs, 1981.
- [87] G. Žitković, "Introduction to Stochastic Processes - Lecture Notes," *Department of Mathematics, The University of Texas at Austin*, 2010.
- [88] H. Jaeger, "Observable Operator Models for Discrete Stochastic Time Series," *Neural Computation*, vol. 12, pp. 1371–1398, 2000.
- [89] S. Singh, M. James, and M. Rudary, "Predictive State Representations: A New Theory for Modeling Dynamical Systems," *Conference on Uncertainty in Artificial Intelligence*, 2004.
- [90] B. Wolfe, M. R. James, and S. Singh, "Learning predictive state representations in dynamical systems without reset," *International Conference on Machine Learning (ICML)*, 2005.
- [91] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [92] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015.
- [93] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
- [94] C. Olah, "Understanding lstm networks," [Online; accessed on March 13, 2019], <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [95] D. Zipser and R. J. Williams, "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity," in *Back-propagation: Theory, Architectures and Applications*. Hillsdale, NJ: Erlbaum, 1995, pp. 433–486.

- [96] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.
- [97] —, "On The Computational Power of Neural Nets," *Annual workshop on Computational Learning Theory*, 1992.
- [98] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Diplomarbeit, Technische Universität München, 1991.
- [99] F. Beaufays, "The neural networks behind google voice transcription," [Online; accessed on March 13, 2019], <https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html>.
- [100] P. Khaitan, "Chat smarter with allo," [Online; accessed on March 13, 2019], <https://ai.googleblog.com/2016/05/chat-smarter-with-allo.html>.
- [101] A. Efrati, "Apple's machines can learn too," [Online; accessed on March 13, 2019], <https://www.theinformation.com/articles/apples-machines-can-learn-too>.
- [102] W. Vogels, "Bringing the magic of amazon ai and alexa to apps on aws." [Online; accessed on March 13, 2019], <https://www.allthingsdistributed.com/2016/11/amazon-ai-and-alexa-for-all-aws-apps.html>.
- [103] A. Robinson and F. Fallside, *The Utility Driven Dynamic Error Propagation Network*. University of Cambridge Department of Engineering, 1987.
- [104] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," *International Conference on Machine Learning (ICML)*, 2006.
- [105] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *International Conference on Artificial Neural Networks*, 1999.
- [106] F. Gers and J. Schmidhuber, "Recurrent nets that time and count," *IEEE International Joint Conference on Neural Networks*, 2000.
- [107] F. A. Gers and J. Schmidhuber, "LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.
- [108] F. Gers, "Long Short-Term Memory in Recurrent Neural Networks," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2001.
- [109] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," *International Conference on Machine Learning (ICML)*, 2015.

- [110] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.
- [111] Google Brain, "TensorFlow API Documentation, version 1.13." [Online; accessed on March 26, 2020], <https://www.tensorflow.org/>.
- [112] Python Software Foundation, "Python language reference, version 3.7." [Online; accessed on March 26, 2020], <https://docs.python.org/3.7/reference/>.
- [113] G. B. Willson, "Radar classification using a neural network," *SPIE Applications of Artificial Neural Networks*, 1990.
- [114] I. Jordanov and N. Petrov, "Intelligent Radar Signal Recognition and Classification," in *Recent Advances in Computational Intelligence in Defense and Security*, ser. Studies in Computational Intelligence. Springer International Publishing, 2016, vol. 621.
- [115] L. Cain, J. Clark, E. Pauls, B. Ausdenmoore, R. Clouse, and T. Josue, "Convolutional Neural Networks for Radar Emitter Classification," *IEEE Annual Computing and Communication Workshop and Conference*, 2018.
- [116] H. Li, W. Jing, and Y. Bai, "Radar Emitter Recognition Based on Deep Learning Architecture," *CIE International Conference on Radar*, 2016.
- [117] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *International Conference on Learning Representations (ICLR)*, 2015.
- [118] C. Chen, Y. Wang, J. Zhang, Y. Xiang, W. Zhou, and G. Min, "Statistical Features-Based Real-Time Detection of Drifted Twitter Spam," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 914–925, 2017.
- [119] B. Tang, M. Heywood, and M. Shepherd, "Input partitioning to mixture of experts," *IEEE International Joint Conference on Neural Networks*, pp. 227–232, 2002.
- [120] J. Goodband, O. Haas, and J. Mills, "A mixture of experts committee machine to design compensators for intensity modulated radiation therapy," *Pattern Recognition*, vol. 39, no. 9, pp. 1704–1714, 2006.
- [121] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: A literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.
- [122] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," *SIAM International Conference on Data Mining*, 2007.
- [123] I. Jo, J. Kim, H. Kang, Y.-D. Kim, and S. Choi, "Open Set Recognition by Regularising Classifier with Fake Data Generated by Generative Adversarial Networks," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

- [124] E. Granger, S. Grossberg, P. Lavoie, and M. A. Rubin, "Comparison of classifiers for radar emitter type identification," *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 9, pp. 3–11, 1999.
- [125] E. Granger, M. A. Rubin, S. Grossberg, and P. Lavoie, "A What-and-Where fusion neural network for recognition and tracking of multiple radar emitters," *Neural Networks*, vol. 14, no. 3, pp. 325–344, 2001.
- [126] W. J. Park and R. M. Kil, "Pattern Classification With Class Probability Output Network," *IEEE Transactions on Neural Networks*, vol. 20, no. 10, pp. 1659–1673, 2009.

Curriculum Vitae

Name Sabine Apfeld
Date of birth 23. November 1988
Place of birth Cologne, Germany

since 08/2014 Research Associate
Fraunhofer Institute for Communication, Information
Processing and Ergonomics (FKIE), Wachtberg

06/2014 Degree Master of Science in Computer Science

„Development and implementation of an FPGA-based signal
processing system with adaptive waveform control of a
30 GHz radar for distance measurements“

Master thesis prepared at Fraunhofer Institute for High
Frequency Physics and Radar Techniques (FHR), Wachtberg

01/2012 Degree Bachelor of Science in Computer Science

„Weiterentwicklung des AccelKit PCIe Moduls zur
Unterstützung von nebenläufigen Downstream-Kanälen“

10/2008 - 06/2014 Study of Computer Science
University of Bonn

06/2008 Abitur
Paul-Klee-Gymnasium, Overath