FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Long Short-Term Memory Neural Networks for One-Step and Multi-Step Time Series Forecasting

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Constantin Fränkel
Matrikelnummer 1128002

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Hon.-Prof. Dr. Wolfgang Ernst Katzenberger

Wien, 28.04.2018

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Long Short-Term Memory Neural Networks for One-Step and Multi-Step Time Series Forecasting

MASTER THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Business Informatics

by

Constantin Fränkel
Registration Number 1128002

To the Faculty of Informatics
at the Technical University of Vienna

Advisor: Hon.-Prof. Dr. Wolfgang Ernst Katzenberger

Vienna, 28.04.2018

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Constantin Fränkel, BSc.
Lohnsteinstraße 26C, 2380 Perchtoldsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Constantin Fränkel)

Acknowledgments

Besides one's own will and dedication, I know that all achievements in life are also a product of lucky circumstances. I am deeply grateful for the circumstances that allowed me to study and write my master thesis at the Technical University of Vienna. I especially want to thank my family for their loving care and help on my path in life. Further, I want to express my thankfulness to my advisor Hon.-Prof. Dr. Wolfgang Ernst Katzenberger for his philanthropist attitude to imparting knowledge to his students and for encouraging me since my bachelor studies, giving me the opportunity to deepen my knowledge in the most interesting fields. Last but not least, I am thankful for my dear friends, many of whom I also found during my studies at the TU Vienna, for their joyful company and support.

Abstract

This master thesis aims to employ Long Short-Term Memory (LSTM) neural networks for one-step and multi-step time series forecasts. For this endeavor, a software stack, including a deep learning framework is selected and different machine learning- and statistical models are implemented. The performance of the LSTM approaches is compared to carefully chosen benchmark methods on an exemplary real-world problem and the experiments are run on powerful, cloud-based machines. To provide a methodological framework for time series forecasting projects, a seven-phase process model is elaborated. Further, to allow for model selection of computationally intensive deep learning methods under limited resources, a modified form of blocked cross-validation, together with a multi-stage Bayesian hyperparameter optimization approach is proposed. The proof of concept of the proposed methodology is conducted on the real-world problem in the domain of electricity demand forecasting. The implemented LSTM model clearly outperformed the benchmark models on all performance measures in the one-step walk forward out-of-sample test and showed a roughly 10% lower root-mean square error than the second-best model which utilized double seasonal Holt-Winters exponential smoothing. Inspired by work in the area of natural language processing, for the multi-step scenario, an encoder-decoder LSTM neural network was implemented, as simpler architectures showed disappointing results. Also, the multi-step LSTM forecaster proofed to be a competitive approach, but the purely statistical model had the lead. However, due to resource constraints, it was not possible to retrieve statements on the same validity level as for the one-step case. By comparing the LSTM-forecaster to the predictive performance of simple recurrent neural networks, the added value of the more complex, gated cell architecture of the LSTM has been indicated. A downside of LSTM neural networks is the relatively long training time which can be a problem for exhaustive hyperparameter searches. On the other hand, LSTM neural networks showed to have good generalization ability and needed comparably infrequent retraining.

Kurzfassung

Das Ziel dieser Masterarbeit ist es, neuronale Netze, mit sogenannten Long Short-Term Memory (LSTM) – Blöcken, für Ein- und Mehrschritt- Zeitreihenprognosen anzuwenden und deren Performance im Vergleich zu sorgfältig ausgewählten Benchmark-Methoden anhand von Daten eines ausgesuchten Problemfeldes, exemplarisch zu evaluieren. Für dieses Vorhaben wird ein Software-Stack selektiert und verschiedene maschinelle Lernverfahren, sowie ein statistisches Vorhersagemodell implementiert werden. Ein siebenphasiges Prozessmodel zur Durchführung und Evaluierung von Projekten zur Zeitreihenvorhersage wurde konzipiert und im empirischen Teil dieser Arbeit angewandt. Des Weiteren wurde eine Methode entwickelt, um aufwendige Modellselektionen mit vielen verschiedenen, zu optimierenden Hyperparametern unter begrenzten Ressourcen durchzuführen. Dafür wird eine modifizierte Form einer geblockten Kreuzvalidierung, mit einer mehrstufigen, Bayesschen Hyperparameteroptimierung kombiniert. Die vorgestellte Methodik wurde im empirischen Teil mit Hilfe von Daten aus dem Bereich der Elektrizitätsverbrauchsvorhersage validiert.

Das implementierte neuronale LSTM- Einschrittvorhersagemodell lieferte im walk forward out-of-sample Test eindeutig die besten Ergebnisse betreffend aller angewandten Metriken. So konnte etwa ein um 10,3% geringerer Root Mean Square Error (RMSE) als beim zweitbesten Modell, einem doppel-saisonalen, exponentiellen Glättungsverfahren nach Holt-Winters, erzielt werden. Inspiriert durch Fortschritte auf dem Gebiet der maschinellen Sprachverarbeitung, wurde für das Mehrschritt-Szenario ein neuronales LSTM-Encoder-Decoder-Netzwerk implementiert, da einfachere Architekturen unbefriedigende Resultate lieferten. Auch die Mehrschrittprognose mittels neuronalem LSTM-Encoder-Decoder-Netzwerks erwies sich als vielversprechender Ansatz, jedoch lieferte das statistische Modell die genauesten Vorhersagen. Aufgrund von Ressourcenbeschränkungen war es jedoch nicht möglich, diese Experimente mit derselben Aussagekraft wie für die Einschrittprognosen durchzuführen. Durch Vergleich der Güte von Prognosen mittels neuronaler LSTM-Netze einfacherer, rekurrenter neuronaler Netzwerke wurde der Mehrwert der komplexeren, Architektur der LSTM angezeigt. Ein Nachteil von neuronalen LSTM-Netzen ist die relativ lange Trainingszeit, welche ein Hindernis für ausgiebige Hyperparameteroptimierungen darstellen kann. Dagegen stellen die gute Generalisierungsfähigkeit und das vergleichsweise selten benötigte Retraining einen klaren Vorteil dar.

Contents

1. Introduction

1.1. Problem Statement and Objectives

Being able to make precise time series forecasts is indispensable to many sectors such as medicine, demography, finance, energy, meteorology, and geodynamics, just to name a few. Furthermore, with the recent developments in the domain of Internet of Things (IoT) the amount of time series data generated by different sensory devices is growing fast and predictive solutions are gaining even more importance. The prediction of time series is a regression problem, as regression deals with the estimation of the relation between a dependent output variable and one or more independent input variables which are also called predictors. There exist different approaches to perform a regression on time series data which can be roughly categorized in statistical and machine learning approaches. For a long time artificial neural networks (ANNs) were widely used in machine learning but some downsides such as the vanishing gradient problem and a long learning time when using backpropagation to train networks with many layers led to a shift to Support Vector Machines (SVM), which proofed to be suitable for many applications and could reduce the learning time significantly. The problem with the SVM is that it uses a fixed kernel function and is therefore not flexible regarding the input features. A possible countermeasure is to use expert knowledge about the input data to improve the kernel functions. [1]
Recently, a type of machine learning models that also tackle the above-mentioned problems led to a renaissance of neural network models. These models are categorized under the term deep learning, which refers to a set of algorithms and architectural improvements that enhance a machine's ability to learn complex coherences. Still, there is a lot of ambiguity when it comes to the terms machine learning, deep learning as well as other related concepts such as artificial intelligence and computational neuroscience, therefore a disambiguation and categorization is provided in the introduction of this thesis. However, deep learning models proofed to be superior over more conventional models in the areas of speech- and text processing, computer vision, and many others. Nevertheless, the majority of research in deep learning has been done on static data and less on time-dependent data (compare with [2], [3]). The type of deep learning models that will be used in this thesis are so-called Long Short-Term Memory (LSTM) neural networks which are especially interesting for working with time series, as they have a special cell architecture that allows for retaining learned characteristics of a series for a long time. To further motivate the usage of LSTMs and to put deep learning into a historical context, the evolution from the first artificial neuron to the modern architectures is explained.
In the empirical part, a state of the art hardware and software stack shall be used to conduct the experiments. In the case of one-step-ahead predictions, already the last historical value can be a good estimate for the value to be predicted, and it is often not trivial to beat this naive forecasting approach. With an increased number of time steps to forecast, the uncertainty increases, making it more difficult to produce good forecasts. Therefore, the performance of the LSTM models is evaluated in a one-step- and a multi-step scenario. As a benchmark, other and simpler regression techniques will be used. This is an important point, since the comparison to computationally less expensive techniques sometimes seems to be forgotten in deep learning research. Further, especially in machine learning research in the time series forecasting domain, a lack of a clearly defined methodology for working through a forecasting project and evaluating the outcomes seems to be prevalent, as it is also mentioned in a review by Hippert et al. [4]. Thus, a process model with a clearly defined methodology shall be elaborated for the

empirical part of this thesis. Moreover, when it comes to hyperparameter optimization and model selection, working with deep learning models and time series can be very resource intensive and often a minimalistic approach is chosen to optimize the parameters due to costly evaluation runs. Hence, another objective of this work is to develop an approach that reduces the amount of required resources.

1.2. Structure of the Work

In this section, a short description of the thesis structure shall be given.

- Chapter 1 gives an introduction to the thesis by stating the problem and defining the objectives. The remainder of this chapter gives an overview of artificial neural networks and its biological sources of inspiration, to introduce the most important terminologies used in the subsequent chapters. Further, a disambiguation and classification of terms like machine learning, artificial intelligence, deep learning and computational neuroscience will be conducted to establish a common understanding of these.

- Chapter 2 outlines the development from the first artificial neuron models towards deep learning as known nowadays, based on a categorization into three development waves. Moreover, this chapter introduces the most important architectures such as feedforward and recurrent neural networks and contains the theory needed to understand the most important concepts for deep learning such as the backpropagation algorithm. This chapter concludes with a description of the latest advances in the field of deep learning such as Deep Belief Networks, Convolutional Neural Networks, and LSTM Recurrent Neural Networks.

- Chapter 3 covers related work that was sighted during the literature research and describes the used methodological approaches and documents the outcomes. It concludes with an analysis of potential implications and research questions that are tackled within this study.

- Chapter 4 elaborates on the methodological approach that was used in this study. A process model for working through time series forecasting projects is developed and different aspects such as data partitioning, model selection, hyperparameter optimization, model evaluation and the chosen benchmark methods are documented in detail.

- Chapter 5 contains the empirical part of this study. The forecasting problem is described, an exploratory data analysis is conducted and the necessary pre-processing steps are described. Further, the selection criteria for the soft- and hardware stack are listed and the chosen setups are presented. The experimental setup for the one-step and the multi-step scenario is explained and the process of hyperparameter optimization is documented. Finally, a thorough quantitative and qualitative analysis of the outcomes and comparison to the benchmark methods is provided by the means of carefully chosen performance measures.

- Chapter 6 summarizes the outcomes, points out the limitations of this study and gives an outlook for further research in this field.

1.3. Introduction to Artificial Neural Networks

1.3.1. The biological neuron and biological neural networks

Artificial neural networks are inspired by the functioning of the biological brain and its smallest building parts the biological neurons (see figure 1).

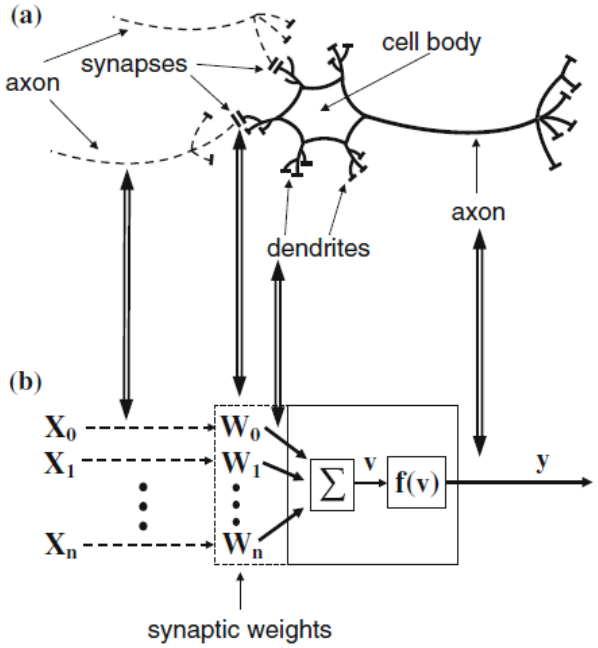

Figure 1 A schematic figure of a biological neuron (a) and its artificial counterpart (b) [5]

A biological neural network is a highly sophisticated web of neurons in which information is transmitted via electrochemical signals. These signals are being received by dendrites which form the input wires to the neuron. On a high abstraction level, one can say that the cell body (soma) of a neuron will fire an output signal as soon as a certain threshold of activation through its dendrites is reached. The output signals travel through so called axons to synapses which can link an axon of one neuron to the dendrite of another neuron.

1.3.2. The artificial neuron

The artificial counterpart to the biological neuron, depicted in figure 1, has one or more input links (x_0 …x_n), which are weighted by the synaptic weights(w_0..w_n), and transforms the input signals to an output signal y which describes the activation state of the neuron. The activation state is given by a transfer function that is often a step or sigmoid function of the weighted sum of the inputs subtracted by a threshold. [6] One can imagine the synaptic weights as values that assign a relative importance to an input compared to the other inputs. In accordance, the threshold then sets a level of importance that has to be reached for the neuron to fire.

1.3.3. Artificial neural networks and deep architectures

An artificial neural network consists of a net of interconnected artificial neurons and usually, those networks have a 3-layer architecture as you can see in figure 2.

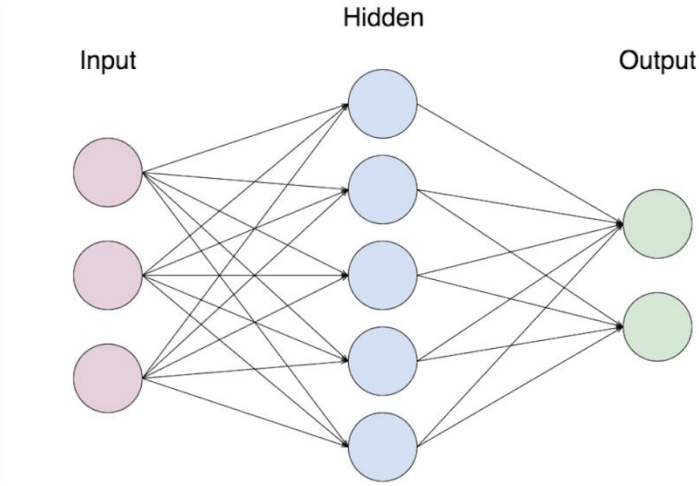

Figure 2 Example of artificial neural network architecture

The neurons in the both outer layers, the input and the output layer build the interface for processing input signals and supplying output signals respectively. The intelligence happens in the in-between layer, which is referred as hidden layer, since it is hidden from the outside. Now, the hidden layer can have more than one layer itself and broadly speaking, while networks with one hidden layer are shallow architectures by definition, all artificial neural networks with more than one hidden layer can be considered as deep neural networks. Nevertheless, a deep architecture and deep learning are not interchangeable terms. As the term deep learning already suggests it is not only referring to a type of architecture but more to a set of algorithms and architectural improvements to facilitate learning.

1.3.4. Classification of Neural Networks

Supervised Learning, Unsupervised Learning

As already explained in the previous section, the intelligence of artificial neural networks lies in the hidden layer, to be more precisely, in the allocation of the synaptic weights. Now, there are many different approaches how these weights can be learned during training, but generally one can differ between supervised- and unsupervised learning.

In supervised learning, the network is trained on input data and at the same time provided with the expected results regarding the shown inputs. The discrepancy between the input and the desired output is then used to readjust the weights, to minimize the error. [7] A very famous representative of these types of algorithms is the backpropagation algorithm. This thesis will mostly deal with supervised learning in the empirical part, since for the training for regression problems the desired output (classes or historical values) are known. Supervised learning problems can further be categorized into classification and regression problems:

- Classification: Provides categorical outputs
- Regression: The output is in real valued form

In contrast, in settings with unsupervised learning, the network has to be trained on problems where no reference data of correct outputs can be provided. Thus, the neural network has to be self-organizing and work on the input data without further guidance with the aim to model the underlying data structure/distribution to be able to conduct clustering or association[8]:

- Clustering: Uncover inherent groupings in the data
- Association: The goal of association rule learning is to find rules that are highly distinguishing concerning the data. That means a good association rule distinguishes data with different characteristics as precisely as possible and at the same time is able to sum up large groups with similar characteristics.

Lastly, there is also a third setting which is referred to as semi-supervised learning and as the name suggests, it is a mixture of both methods where in most cases an unsupervised learning phase is followed by a supervised learning phase. This method proofed to be extremely powerful for big datasets where the majority of data is unlabeled and only a minor part can be used for supervised training. Deep Belief Networks, introduced in section 2.4.2, used for discrimination tasks are a very famous example of successful models using that type of learning.

Feedforward and Recurrent Neural Networks

Artificial Neural Networks (ANNs) are often architecturally categorized by the way their neurons and layers are connected. The two most important categories are *Feedforward Neural Networks (FNNs)* and *Recurrent Neural Networks* (*RNNs)*. For example, the network shown in figure 2 is a FNN. Thus, the information/signal flow is only directed forward from the input neurons in the direction of the output neurons. In section 2.3.1, these architectures are covered in more detail. In contrast, in RNNs there can be feedback loops, forming cycles within the network graph. These types of models will be covered in section 2.3.2 and further in the deep learning part, in section 2.4.3., with the so-called Long Short Term Memory Recurrent Nets. For this thesis, the RNNs are of special interest since they allow for incorporation of temporal dependencies.

Activation Functions

The neurons in a neural network can have different activation functions and in general, one can differ between linear and non-linear activation functions. However, a neural network consisting of only linear units has very limited learning capacity, since also a conglomerate of linear units can only grasp linear dependencies. Therefore, nearly all modern neural networks have some non-linearity in their activation functions. In the following figures, three graphs of very common activation functions, specifically

the ReLU-, sigmoid- and tanh function, are depicted. The choice of the activation function is very important and strongly depends on the task. A very important property of a good activation function is its differentiability. All three functions are differentiable but building the gradient of the tanh function is computationally less expensive than building it for the sigmoid function, further it tends to converge faster. However, the problem with both functions is that they get saturated for large activation values, which means that the gradient will be zero and for learning algorithms this can be a serious problem (see section 2.3.1).

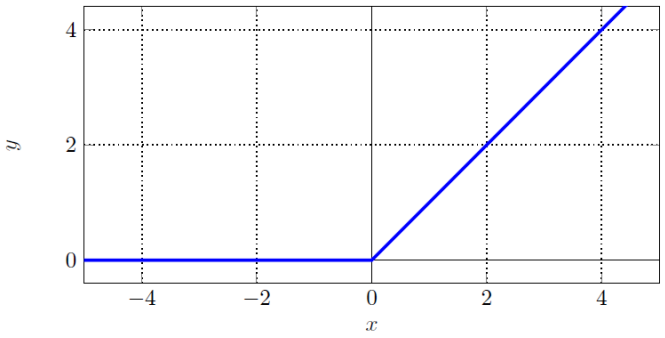

Figure 3 Rectified Linear Unit (ReLU) activation function

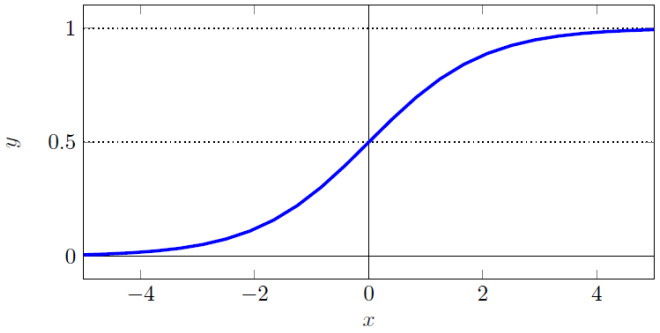

Figure 4 Sigmoid activation function

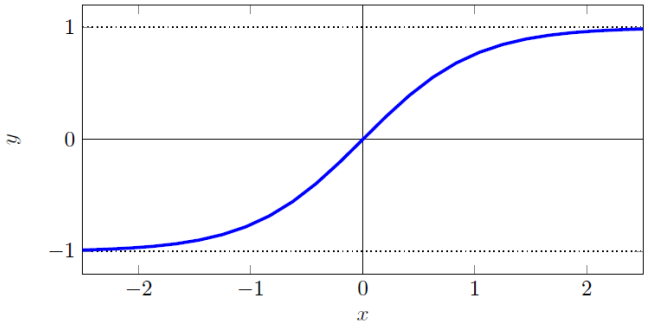

Figure 5 Tanh activation function

1.4. Classification of Terms

1.4.1. Deep Learning, Artificial Intelligence and Machine Learning

The terms artificial intelligence (AI), Machine learning, artificial neural networks and deep learning are often mixed up and used for referring to all kinds of systems, therefore in the following a disambiguation shall be provided. The explanation will be based on the Venn diagram in figure 6 and the classification in figure 7. Artificial intelligence is the most general term and sums up all areas of computer science where a machine has cognition abilities that allow for simulation of intelligent behavior such as learning and/or solving problems.
A type of AI systems are knowledge bases, which were invented at the Stanford University in an approach to implement the Dendral expert system. This system helped in finding new organic molecules by incorporation of their mass spectra and inferring conclusions from chemistry knowledge. [5] So what those systems basically do is trying to mimic a human expert by drawing conclusions from a database that contains the knowledge of experts in the regarding field with as much precision as possible. The next, narrower term, is machine learning (ML) - in contrast to a knowledge base as a representative of AI systems that are not capable of learning, those systems can learn from the data they are shown and gain experience in the course of training. What discerns them from the more advanced representation learning systems is the lack of automated feature engineering. That means that the inputs to those models are handcrafted features. Finally, deep learning levels the automated feature engineering capabilities of AI-systems to new grounds by making use of different abstraction levels. Those abstractions make use of complex representations which are built out of simpler representations, which also explains why deep learning is a subcategory of representation learning.[9]

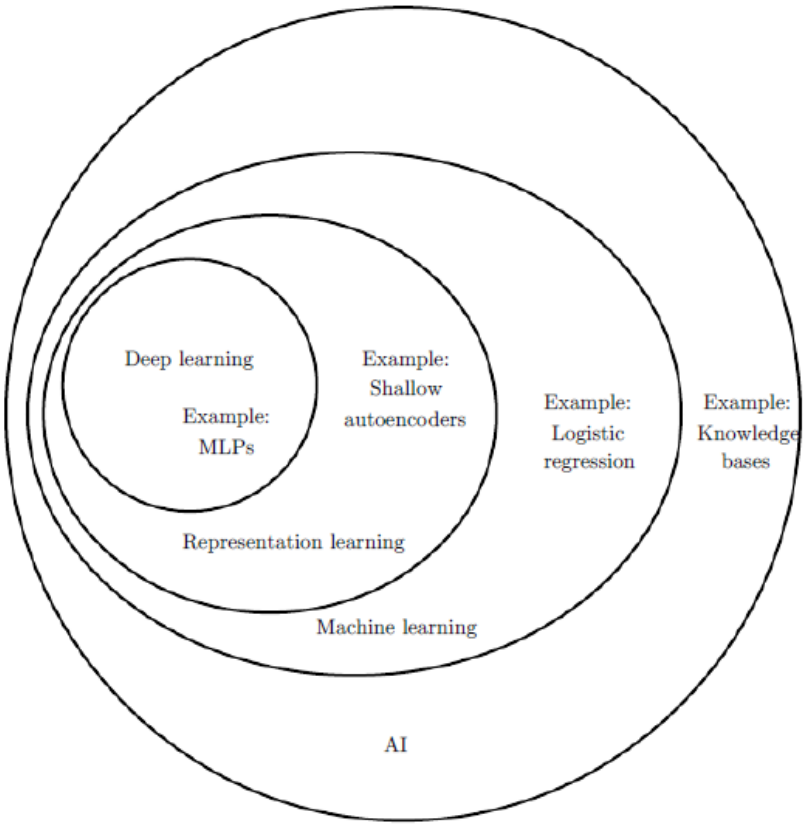

Figure 6 Venn diagram for sorting the terms around deep learning [9]

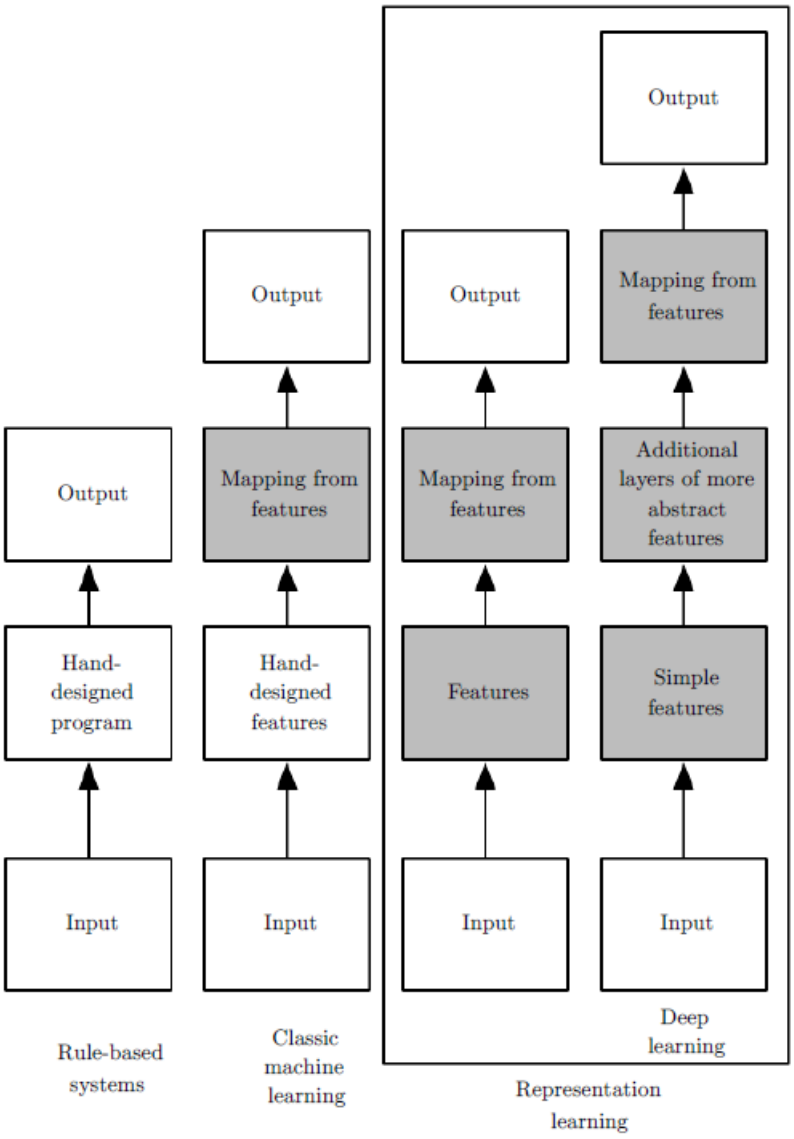

Figure 7 Difference between Rule- based systems, machine learning and representation learning. The grey shaded boxes indicate which steps of the feature mapping and engineering part are conducted automatically by the respective systems. [9] Rule-based approaches for expert systems like knowledge bases draw their intelligence completely from a human-defined set of rules which produce the output. In classic machine learning systems, features are extracted manually from the input which is often a very tedious task and automation is limited to the feature mapping. In systems inhibiting representation learning, also the feature engineering part is automated, which reduces the need for human interaction significantly and, even more for deep learning, often leads to superior models.

1.4.2. Deep Learning and Computational Neuroscience

As shown in the next chapter, deep learning originated from models that are inspired by biological neurons and biological brains, nevertheless the functioning of the biological brain is to a large extent still unknown and therefore it can't be used as a blueprint for the development of artificial neural networks. On the other hand, artificial neural networks can also help to gain more knowledge of the functioning of the biological counterparts, since with computational neuroscience experiments in a bottom up approach can deliver useful insights.
Hereto Goodfellow et al. write "The field of deep learning is primarily concerned with how to build computer systems that are able to successfully solve tasks requiring intelligence, while the field of computational neuroscience is primarily concerned with building more accurate models of how the brain actually works." And further they define deep learning very precisely like in the following:
"The modern term "deep learning" goes beyond the neuroscientific perspective on the current breed of machine learning models. It appeals to a more general principle of learning *multiple levels of composition*, which can be applied in machine learning frameworks that are not necessarily neurally inspired."[9]

2. Development of Deep Learning

As shown in figure 8 Goodfellow et al. sort the development of neural networks into three waves which finally emerged to deep learning:

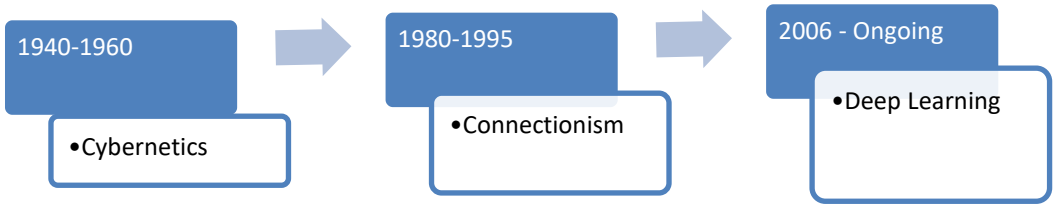

Figure 8 Three waves of development - Cybernetics, Connectionism, Deep Learning

Cybernetics, 1940-1960: A transdisciplinary research field that Norbert Wiener defined as "the scientific study of control and communication in the animal and the machine". [10] In this era the simplest and though till nowadays highly relevant architectures that resemble biological neurons were proposed by Mc Culloch and Pitts [11]. However, those structures were not capable of learning in the common sense. In 1949 the psychologist Donald O. Hebb published his book "Organization of Behavior" in which he described the process of learning in the biological brain as a process of modification of synapses and electrochemical signals [12]. Inspired by the work of Hebb, in the late 1950s, Frank Rosenblatt developed the perceptron [13] which was the first neural structure that was capable of learning artificial synaptic weights.

In 1969 the momentum in artificial neural networks rapidly slowed down as a consequence of a book published by Marvin Minsky and Papert [14]. In this book, he demonstrated the systematically immanent downside of the perceptron, as it can only learn linear functions and thus fails to solve problems which are non-linearly separable.

A famous example for this is the XOR-function which is a two dimensional problem, but there does not exist a linear separation for the solution, thus a third dimension is needed to be able to draw a linear separation hyperplane. The third dimension can be introduced by an additional layer to the perceptron. Minsky and Papert also were aware of that fact but the Perceptron learning algorithm introduced in section 2.2 was not able to learn the connection weights in a multilayer setting and Minsky and Papert did not believe in the possibility of finding a general rule for this problem.[15]

By academia, their findings were falsely generalized as an insuperable obstacle for artificial neural sciences. As a consequence, the development of artificial neural networks came to a halt and it took till the 1980s to gain momentum again, also because "negative opinions of eminent AI authorities caused limited financing of research into neural networks" [5].

At this place, a very interesting historical fact from the book "Artificial Intelligence – A Modern Approach" by Russel and Norvig [16] shall be cited : "Marvin Minsky and Dean Edmonds, built the first neural network computer in 1951. The SNARC, as it was called, used 3000 vacuum tubes and a surplus

automatic pilot mechanism from a B-24 bomber to simulate a network of 40 neurons. Minsky's Ph.D. committee was skeptical whether this kind of work should be considered mathematics, but von Neumann was on the committee and reportedly said, "If it isn't now it will be someday." Ironically, Minsky was later to prove theorems that contributed to the demise of much of neural network research during the 1970s."

This quote nicely shows how much Von Neumann, already at this early stage, was an advocate of neural computing because he saw the potential of introducing stochastics into computing and he reportedly defended his stance several times against resistance of the mainstream (Boolean-branch) computer scientists.

Connectionsim, 1980-1995: A subfield of cognitive science which deals with complex, emergent phenomena that occur by the interplay of interconnected, simple units. The term itself was already introduced by Hebb in the 1940s [17]. During this period, the power of interconnecting the simple structures already discovered in the cybernetics-era was employed to learn more complex, nonlinear functions via interconnected artificial neurons. Two important pioneers in this area are Rumelhart and McClelland who, together with other famous researchers such as Geoffrey Hinton, formed the Parallel Distributed Processing Research Group and published their much-recognized work under the title "Parallel-Distributed Processing: Explorations in the Microstructure of Cognition"[18]. In their book, Goodfellow et al. point out some of the key concepts that were discovered at this time which remain relevant also in modern deep learning research such as distributed representation and the backpropagation algorithm.

The concept of distributed representation boils down to the idea that different neurons in the hidden layer have very specialized abilities (for example the recognition of the color red) and with this granularity of specialization there comes the ability of generalization. That means that a neuron in the hidden layer can make valuable cognition for many different input features. Goodfellow et al. give the example of the neuron with the recognition of the color together with neurons which each can identify one simple object. In an undistributed representation, the number of neurons in the hidden layer would equal to a single neuron for every different object, times the number of colors each object can have. In contrast, in a distributed representation one would have only one neuron for every color and one neuron for every object, so the color neurons represent general knowledge that can be used for many different objects. This shows that the effectivity of the concept of distributed representations comes from the power of generalization through specialization.

Another important contribution of the research on artificial neural networks dating back to this time is the backpropagation algorithm which is used to train a neural network with more than one layer and will be covered in more detail, since it is so essential to all further developments and is still one of the predominant algorithms used for learning. Nevertheless, algorithms as like backpropagation have downsides such as the vanishing gradient problem and a long learning time for many-layered networks, which make the usage for these computationally unfeasible. Together with the progress made in other areas of machine learning such as Support Vector Machines (SVMs), Goodfellow et al. state that unrealistically high expectations led to a sinking popularity of ANN-research: "Ventures based on neural networks and other AI technologies began to make unrealistically ambitious claims while seeking investments. When AI research did not fulfill these unreasonable expectations, investors were disappointed." [9].

Deep Learning, 2006 – Ongoing: As already mentioned, also SVMs have their downsides due to the fixed kernel function that limits the flexibility of such models since manual feature engineering and optimization of the kernel function is necessary. Further, although the backpropagation algorithm allows for training of multi-layered networks, Mo [1] sums up the most significant, remaining downsides of multilayer – neural networks prevalent in the connectionism era:

1. Lack of training unlabeled data: The ability to train on unlabeled data is extremely important for problems that require unsupervised learning.
2. Backpropagation of the correcting signal through many layers causes its weakening (vanishing gradient problem)
3. Unfeasible learning time for multi-layer networks
4. Backpropagation algorithm can get stuck in poor local optima

In 2006, Hinton et al. published their work on Deep Belief Networks (DBNs) to tackle the problems listed above. The main difference to previous training algorithms is a greedy, layerwise training procedure and the layers itself are built from stacked Restricted Boltzmann Machines instead of stacked autoencoders [19]. Other very prominent representatives of deep learning models are Convolutional Neural Networks (CNNs) and Long-short term memory neural networks (LSTMs).

2.1. The neuron model of McCulloch-Pitts

In 1943, in the Bulletin of Mathematical Biophysics, the neuroscientist Warren S. McCulloch and the logician Walter Pitts published their paper "A logical calculus of the ideas immanent in nervous activity" where they proposed the first mathematical model of a neuron. The inputs and outputs of the McCulloch-Pitts neuron are binary only. In an analogy to biological neural cells, they have a threshold and additionally an inhibitory input. The input signals are summed up and compared to the threshold value. As shown in equation (2.1), if the arithmetic sum of the input signals is greater than the threshold and there is no inhibitory signal, then the neuron's binary output equals to 1 otherwise it is 0. An important thing to mention is that the weights are all the same for the excitatory inputs, nevertheless, an input can be fed into the neuron more than once so through this method the weight can take the value of every positive integer. In any case, the McCulloch-Pitts neuron is not able to learn those weights, so they have to be set manually. [20] [21]
The inputs to the McCulloch-Pitts neuron can be seen as the truth values of propositions and the neuron itself combines those to inputs to calculate the truth value of another proposition.[22]

$$c_{t+1} = \begin{cases} 1 \; if \sum_{i=0}^{n} a_{i,t} \geq \theta \; and \; b_{1,t} = \ldots = b_{m,t} = 0 \\ 0 \; Otherwise \end{cases} \tag{2.1}$$

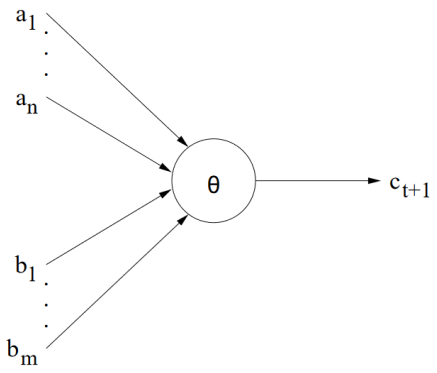

Figure 9 Mc Culloch-Pitts Neuron [21]

2.2. The Perceptron

In 1958 Frank Rosenblatt published a paper in which he introduced the perceptron. The major difference to the previous neuron models was the ability to learn the weights of the Perceptron with the so-called delta rule. In contrast to the McCulloch-Pitts neuron, the perceptron has now different weights per-se and they can be positive or negative.

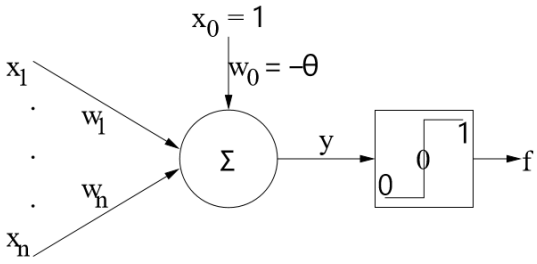

Figure 10 The Perceptron model by Frank Rosenblatt [21]

Below, figure 10 is described mathematically by defining the input- and the weight vector as well as the input function which is the scalar product of the input and output vector and the output function. The threshold is simply treated as a synaptic weight to an activated synapse.

$$\text{Input: } \vec{x} = (x_o = 1, x_1, ..., x_n) \tag{2.2}$$

$$\text{Weight: } \vec{w} = (w_o = -\theta, w_1, ..., w_n), \theta = bias \tag{2.3}$$

$$\text{Net input: } y = \vec{w}\vec{x} = \sum_{i=0}^{n} w_i x_i \tag{2.4}$$

$$\text{Output: } f(\vec{x}) = g(\vec{w}\vec{x}) = \begin{cases} 0 \text{ if } \vec{w}\vec{x} < 0 \\ 1 \text{ if } \vec{w}\vec{x} \geq 0 \end{cases} \tag{2.5}$$

As already mentioned, the biggest advance of the Perceptron is its ability to learn a set of weights by the delta rule. The delta rule can be derived quite intuitively starting from the error function E as the squared residuals summed over all training cases, multiplied by ½ (simplifies the term after the derivative): [22]

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2 \tag{2.6}$$

To obtain the error derivatives for the weights, it is only needed to calculate the partial derivative of the error E with respect to the weights under the usage of the chain rule. The chain rule says that the error will change with respect to a weight change like the output changes with respect to this weight change times the change of the error when the output changes.

$$\frac{\delta E}{\delta w_i} = \frac{1}{2} \sum_n \frac{\delta y^n}{\delta w_i} \frac{dE^n}{dy^n} = -\sum_n x_i^n (t^n - y^n) \tag{2.7}$$

As we build the sum of the errors over all training cases (n) we get in equation (2.8) as a result the so-called batch delta rule, since the set of training cases is also referred to as a batch. The symbol ε stands for the learning rate which determines how fast the algorithm learns and can take any real number between 0 and 1.The correct choice of the learning rate is crucial, as a high learning rate can lead to oscillation and therefore not finding the optimum solution, whilst a very low learning rate may lead to extremely slow learning. Commonly, the learning rate will be chosen by empirical tests.

$$\Delta w_i = -\varepsilon \frac{\delta E}{\delta w_i} = \sum_n \varepsilon x_i^n (t^n - y^n) \tag{2.8}$$

2.3. Classical Neural Network Architectures

2.3.1. Feedforward Neural Network (stacked Autoencoders)

The most widespread artificial neural network architecture is the feedforward neural network (see figure 12 for a schematic depiction of the architecture). Roughly speaking, different hidden layers work on different levels of abstraction and transform the input until the data reaches the output transformation. One can imagine the transformations in a way, considering image classification, that in the first layers very simple geometrical forms are recognized from the input pixels – like lines and edges. In the next layers some neurons may infer shapes from those lines and edges and in the upper most layers the neurons are for example able to decide, based on the representation delivered by the previous layer, whether an image shows a landscape or a technical apparatus.

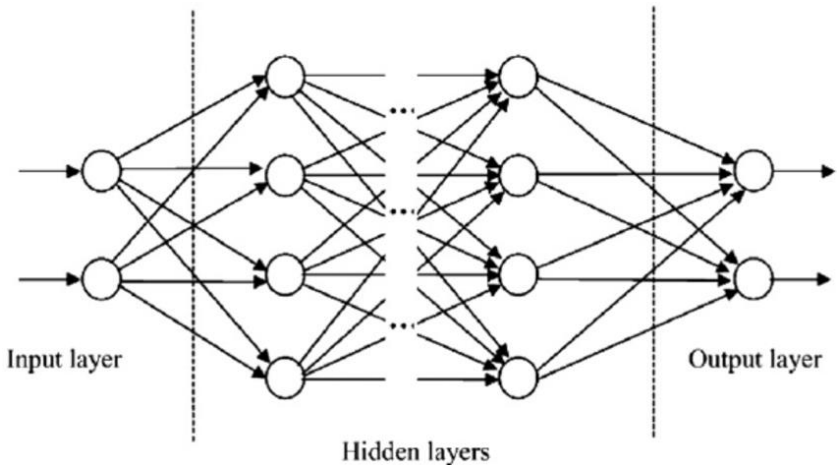

Figure 11 A feedforward neural network with one input layer, several hidden layers and one output layer [23]

Backpropagation

As already mentioned a very famous algorithm for deriving the weights in a neural network with more than one hidden layer is backpropagation. Since this algorithm widely used, it will be derived in the following. Again, like in the case of the delta-rule, the starting point is the quadratic error term E:

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2 \tag{2.9}$$

Then the error derivatives with respect to the output of the output nodes y_j , j∈ output are built. We have to use the error derivatives of the output units since we don't know what the target values within the hidden units should be.

$$\frac{\delta E}{\delta y_j} = -(t_j - y_j) \tag{2.10}$$

The core idea to backpropagation is that the error derivatives in the output layer are propagated one layer downwards (back) and used to compute the error derivatives in this layer. This procedure recursively follows till the algorithm reaches the first network layer.

In figure 12, a schematic neural network is depicted where the node j is a node in the output layer and y_j refers to the output of node j, the same nomenclature goes for node i and its output. The sum of all the inputs that come into the node j from the preceding layer is z_j.

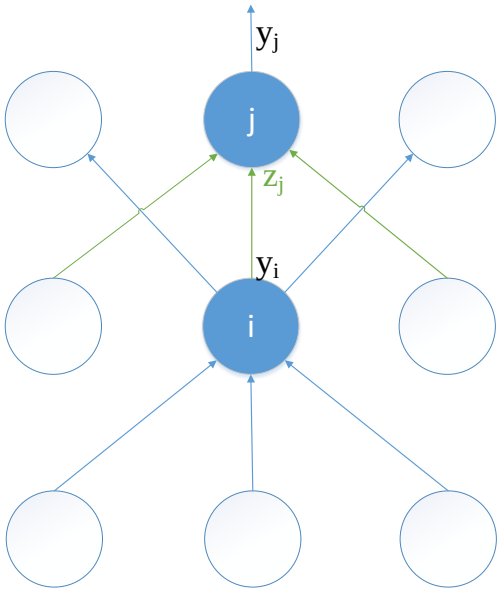

Figure 12 Schematic neural network for explanation of backpropagation adapted from [22]

By making use of the chain rule, in equation (2.11) we can obtain the following expression for the partial derivative of E with respect to the input to node j:

$$\frac{\delta E}{\delta z_j} = \frac{dy_j}{dz_j}\frac{\delta E}{\delta y_j} = y_j(1-y_j)\frac{\delta E}{\delta y_j} \tag{2.11}$$

To calculate the change in the error with respect to a change in the output of unit i, one has to consider all the connections that link the neuron i to the neurons in the layer above. Thus, this derivative can be written as the sum of all the connections from unit i to the above layer in the form of the multiplication between the derivative of the total input to the neurons in the above layer with respect to the output of neuron i and the partial error derivative with respect to the total input to the respective neurons of the above layer. Since the first term of the multiplication simply evaluates to the respective neural connection weights of neuron i to the neural units of the above layer, the result of equation (2.12) is quite intuitive:

$$\frac{\delta E}{\delta y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\delta E}{\delta z_j} = \sum_j w_{ij}\frac{\delta E}{\delta z_j} \tag{2.12}$$

Finally, the missing key to being able to perform the backpropagation algorithm to find a good set of weights for a multilayer neural network is the calculation of the error derivative with respect to the connection weights. Again, via applying the chain rule, it can be computed with the expression in equation (2.13), that boils down to a multiplication of the output of the neuron i, namely y_i, and the expression calculated in equation number (2.11).

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta z_j}{\delta w_{ij}}\frac{\delta E}{\delta z_j} = y_i\frac{\delta E}{\delta z_j} \tag{2.13}$$

In conclusion, that means that with the help of equation number (2.12) it is possible to compute the error changes with respect to the output of a neuron, not only in the layer below the output layer but

in virtually any layer, which allows the evaluation of the formula in equation (2.11). So finally, by back-propagation of the error derivative from the very top layer, we can calculate the error derivatives w.r.t. the weights.
As already mentioned in the sections above, one of the most severe problems with backpropagation is that the gradients at one layer can be seen as the product of the gradients of the previous layer and therefore, if gradients in the product are smaller than 1, the result can get very small. This problem is called *vanishing gradients,* whereas the opposite, when the gradients are big and tend to blow up, it is called exploding gradients. Since general feedforward neural networks, usually have different connection weights at each layer, there are ways to cope with that problem. Though, for Recurrent Neural Networks (RNNs), which are introduced in the next section, this is a serious issue.[9]

2.3.2. Simple Recurrent Neural Networks (RNNs)

Recurrent ANN architectures are extremely important for modeling time series and will therefore, in more elaborate forms, be very prominent in the practical part of this thesis. More general, recurrent neural networks are the architecture of choice when it comes to processing sequential data, which not necessarily is time-dependent. Such time invariant sequences are character sequences and handwriting and regarding the character sequences an important field of research is character generation from an input sequence, where one feeds a text into the net and the output shall be a readable text with, in the best case, grammatical and even contextual fit. [9]

The architecture differs from feedforward networks in the way that there exist loops, forming directed cycles in the connection graph. Those loops can form a feedback on a neuron itself, within the neurons of the hidden layers as well as between the neurons of the output layer and the neurons of the hidden layers.

Bianchini et al. state that the recursive dynamics, coming with these internal loops also allow for delayed activation dependencies between the neurons. The possibility to encode information not only locally distributed by the activation state of different neurons, but also temporally distributed via their time-varying activation, also lets RNNs be known as spatiotemporal Networks.[24]
Both, the feedforward and the recurrent neural networks have some kind of memory. The feedforward network produces a static model of the data from which it has learnt the weights and it outputs solely a function of its input and the weights. Through the feedback loops, the recurrent neural network's memory is more of dynamic nature since its output is a function of the weights, the input, and the previous state. [25]
A very important idea for the understanding of recurrent neural networks is parameter sharing. In ordinary feedforward neural networks, the network would learn the set of parameters for every time step /position of the sequence and thus the ability to generalize for sequences of various length and other time indices is not given. [9]
Figure 13 shows an example of a possible RNN architecture. The variable x stands for the input sequence, while the variable U is the matrix which parametrizes the connection weights to the hidden units h. The connections within the hidden layer are parametrized by the weight matrix W and the weights from the hidden neurons to the output neurons o, by matrix V, respectively.

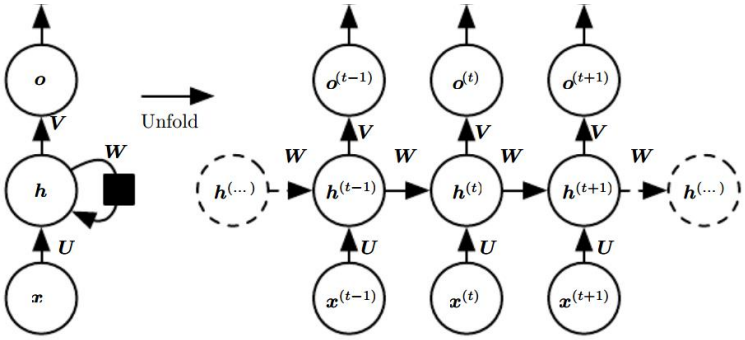

Figure 13 An example of a recurrent neural network architecture as a computational graph (right), unfolded from a circuit diagram (left) [9]

By unfolding the recurrent structure, the following equation can be used to define the state of the hidden units h, depending on time t:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \ldots, x^{(2)}, x^{(1)}) \tag{2.14}$$

The function $g^{(t)}$ takes the current input and all past inputs and builds the current hidden state $h^{(t)}$, by recursive reformulation the next equation is yielded.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{2.15}$$

The unfolding of the recurrent architecture results in a model always having the same input size, regardless of the input sequence, since it is driven by state transitions – one to another- (from $h^{(t-1)}$ to $h^{(t)}$) and not by inputs of different lengths of historical sequences. The formulation of equation (2.15) allows using the identical transition function f with the identical parameter Θ for every time step/position in the sequence (=parameter sharing). Further, Goodfellow et al. point out that in prediction scenarios, the recurrent neural network "learns to use $h^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t". This comes from the fact that $h^{(t)}$ is a fixed length vector, while the input to the neural network can be of any length, thus by keeping important details of the past and leaving less important ones aside, a summary is achieved.[9]

The training of an RNN can again be done with a slightly modified version of backpropagation namely *backpropagation through time*. This boils down to unfolding the recurrent architecture so that we have a normal feedforward architecture which can be trained with backpropagation. The thing is that recurrent architectures can get very deep and the deeper the neural network, the more likely one has to deal with the vanishing/exploding gradients problem. This is even more an issue for RNNs since they share the same weights for every layer, through one cycle of backpropagation.

RNNs have the ability to remember information in the hidden state for a long time. However, as explained for the reasons above, it is very difficult to train them in the way to be able to make use of that. [22] So only the most recent types of RNNs have these capabilities and will be described in the section about Long- Short-Term Memory Networks.

2.4. Deep Architectures

2.4.1. Deep Belief Networks (DBNs)

Deep Belief Nets overcome well-known issues that come up when applying backpropagation to many-layered neural networks: The need for a big labeled data set, long learning times and poor local minima.[26] As one of the first models they allowed training of deep architectures, without using convolution techniques and their introduction in 2006 led to the rise of deep learning.[9] Therefore, this section will explain some of the ideas in a little more detail.

Deep Belief nets are a type of so-called energy-based models which by their nature are probabilistic generative models. The neural networks introduced in the previous sections are of discriminative nature, which means that they can model the dependence of an unobserved variable (a label) on an observed variable. More precisely, after training, a discriminative model can estimate P(Label|Observation). Generative models, in contrast, model a joint probability distribution over the observable and unobservable data, thus allowing to estimate P(Label|Observation) as well as P(Observation|Label). This also explains why this class of models is named generative models, since they can generate observations, given a label, based on the joint probability gathered during the training phase.
Hinton explains the benefits of a generative model over a solely discriminative model with the fact that for example, training with backpropagation only results in models which inhibit how the output depends on the input, but information about the structure of the input is neglected. He further states that this is in particular a weak point if the output can potentially be better modeled by features that capture the structure of the input than by only providing the raw, highly structured input data.[27]
To explain the function of Deep Belief networks it is necessary to understand the building blocks of energy-based models. A very simple energy-based model is the Hopfield network. This neural net is formed by the recurrent interconnection of binary threshold units. The binary threshold units are neurons with a nonlinear activation function that can be in one of two states (mostly -1 and 1). Recurrent networks composed of nonlinear units can either settle to a stable state or end up in inconvenient states such as oscillation or chaotic behavior. John Hopfield discovered that making the connections between the units symmetric, one can overcome this inconvenience and further, the network's state can be easily calculated by what is called a global energy function. That means all possible binary states of the neurons in the network are assigned an energy and these energies sum up to the global energy of the network. Moreover, by a clever choice of the energy function, by conducting the binary threshold decision rule, the network will settle to an energy minimum. Hopfield found out that the binary configurations in an energy minimum can be used to store memories by utilization of a very simple learning rule for the weights, where the weight increment is determined by the product of the binary states of the connected neurons. Further Hopfield noted that this architecture gives a content addressable memory, which can also restore memories by having only partial information about it.[22]
Though I wanted to mention these interesting facts about the properties of a Hopfield net, at this point, the learning rule for the weights will not be further explained, since the theory of the Hopfield net shall mainly be provided for introducing the energy function.

Equation (2.16) shows a possible energy function of a Hopfield neural network. It consists of the sum of the products of the bias terms b_i with the state s_i of the neuron i and the sum of the product of the states (s_i and s_j) of every two interconnected neurons i and j weighted by their connection weight w_{ij}.

$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij} \tag{2.16}$$

The lower the energy, the better, and the binary threshold decision rule can be seen from the viewpoint of calculating the so-called energy gap ΔE_i in equation (2.17) which is nothing else than the difference between the global energy when a neuron (i) is in its on-state and when the same neuron is in its off-state. So the quadratic global energy function allows a neuron to locally compute if turning on/off will cause the global energy to fall or to rise and thus to set its global energy minimizing state.

$$\Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij} \tag{2.17}$$

An issue with Hopfield nets is that they can get trapped in poor local minima which differ from the global minimum. This comes from the fact that due to its energy minimizing decision rule it won't be able for example in figure 14 to escape from the local energy minimum in point A and find the global minimum B.

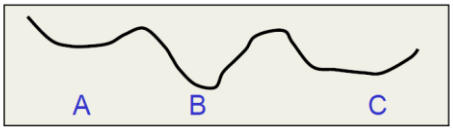

Figure 14 Local and global energy minima[22]

The solution to the local minima trap is to add some random noise to the system which allows jumping out of such traps. Therefore, the binary threshold units are replaced by stochastic binary threshold units. This means that the state of a neuron is now stochastically determined by equation (2.18), where the temperature T controls how much noise is added. By gradually reducing the amount of noise the network will settle in a global minimum. This method is inspired by physics and is referred to as simulated annealing.

$$p(s_i = 1) = \frac{1}{1 + e^{-\Delta E_i / T}} \tag{2.18}$$

[22]

Another way to utilize Hopfield nets, besides storing memories, is to let the net create interpretations of input shown to the net. This can be done by adding a layer of visible units in front of the network (input layer) and connecting these with the hidden layer neurons which then, after learning, represent the interpretation of the inputs. Stochastic Hopfield nets with hidden units are often referred as Boltzmann machines. But as Goodfellow et al. point out, all energy-based models that follow a Boltzmann probability distribution can be called Boltzmann machines.[9]

Further, for the following examinations, the temperature will be constantly set to 1, which equals to thermal equilibrium, which boils down to the settlement of the probability distribution over the model to the stationary distribution. Nevertheless, the model is still of stochastic nature since the probability

with which a neuron turns on is following the Boltzmann distribution, thus allowing the model to escape from local minima.[28]
Instead of further explaining general Boltzmann machines, I directly want to come to Restricted Boltzmann machines (see figure 15), as they are the key components of Deep Belief Networks. They differ from general Boltzmann machines in the way that neurons in the same layer have no connections amongst each other. This leads to nicer mathematical properties which allow for easier computation during learning (see Goodfellow et al.[9]).

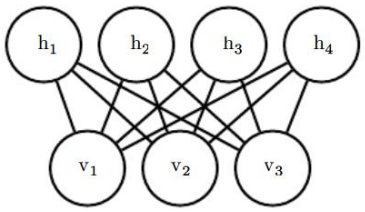

Figure 15 Restricted Boltzmann machines have only one hidden layer behind the visible layer and their interconnections form a bipartite undirected graph, meaning that there are no connections between units of the same layer. Usually, every hidden unit is connected to all visible units and vice versa but as for example for convolutional RBMs, they can also be sparsely connected.[9]

So for restricted Boltzmann machines the energy function looks like in equation (2.19), where the variable v_i denotes the i-th neuron of the visible layer while the variable h_k denotes the k-th element of the hidden layer and accordingly, w_{ik} stands for the connection weight between the i-th visible and the k-th hidden neuron [22]:

$$E(v,h) = -\sum_{i \in vis} v_i b_i - \sum_{k \in hid} h_k b_k - \sum_{i,k} v_i h_k w_{ik} \tag{2.19}$$

The probability that the vector v of a configuration of visible units and the vector h of hidden units occur in a joint configuration, can be expressed by the energy of that joint configuration in relation to the sum of all other possible joint configurations. The sum in the denominator is known under the name *partition function*.

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}} \tag{2.20}$$

As easily follows, the probability to get a certain configuration of the visible units (vector v), is then given by the sum of all joint configurations that have v in it, again divided by the partition function.

$$p(v) = \frac{\sum_{h} e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}} \tag{2.21}$$

At this point, it is necessary to have a closer look how a restricted Boltzmann machine performs learning.The goal of this machine is to construct an interpretation of the input, more precisely to grasp its probability distribution. Therefore the learning algorithm has to find a way to maximize the probabilities that the Boltzmann machine assigns to the vectors in the training set. As already seen in previous sections, learning in neural networks is always done through changing the connection weights and as it turns out, there is a very simple learning rule for restricted Boltzmann machines. By differentiation of

the log-probability of p(v) in equation (2.21), as Hinton[22] shows, one yields the simple update rule written down in equation (2.22).

$$\Delta w_{ij} \propto \left\langle s_i s_j \right\rangle_{\text{data}} - \left\langle s_i s_j \right\rangle_{\text{model}} \tag{2.22}$$

So the weight update boils down to the difference of two correlations which quantify how often two neurons are simultaneously in the on-state under two different circumstances. These circumstances are a) when a data vector is clamped to the visible units and b) when there is no data vector clamped to the visible units, which are in equation (2.22) referred to as data- and model state respectively. Hinton gives an intuitive explanation of this equation, in a way that the first term (data) strengthens connections between neurons according to the Hebbian learning rule (fire together wire together) and the second term makes sure that these connections don't get too strong since otherwise no learning and escaping of spurious minima would be possible.

Now, what really makes restricted Boltzmann machines so much preferable to general Boltzmann machines is that Geoffrey Hinton found out that the collection of the correlation statistics can be performed in a one-step manner in restricted Boltzmann machines whereas in general ones, a lot of steps had to be conducted. The algorithm that allows for efficient computation of the learning rule is called persistent *contrastive divergence*.

In figure 16, the idea for the contrastive divergence algorithm is depicted: It divides into two phases, namely the positive and the negative phase. In the positive phase, the data vectors of the training set (data) are clamped to the visible units (one after another) and the hidden units are updated according to the weights and their update rules, all in parallel (which can be done since there are no inter-layer connections, thus implying independence between two units of the same layer). The correlation statistics are collected by simply averaging over the number of simultaneous activations the neurons of the hidden and the visible layer have together, through the whole set of training vectors. In the negative phase, the states of the of the hidden units after the positive phase are used to reconstruct the state of the visible units and again, the correlation statistics are measured similar to the positive phase (compare [22]).

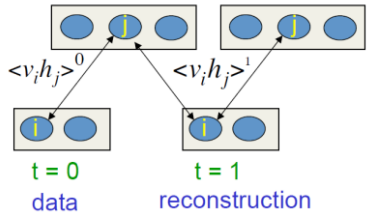

Figure 16 Contrastive Divergence for Restricted Boltzmann machines [22]

After examining the basic theory of energy based models and restricted Boltzmann machines, the motivation to Deep Belief nets shall be provided. Like in many developments of machine learning, Geoffrey Hinton again had a leading role in discovering the potential of Deep Belief Nets. He stacked up restricted Boltzmann machines so that the hidden layer of the first machine is the visible layer of the second one and so on.

The idea behind it is, that the output of Boltzmann machines extracts features and models correlations within the input data in an unsupervised manner and by using these features as input to the next layer

of another restricted Boltzmann machine, the system is able to learn higher order features and correlations.[26] [29]
Though DBNs can be constructed in a way that they work as purely generative models, generating data according to the observed training data, in the following, an architecture is considered, involving discriminative learning, which proofed to be useful in classification tasks. For this purpose, on top of the network, a final layer of neurons with a so-called softmax activation function is added, which allows for discrimination between classes if class-labels are provided for training (the same final layer is often used for classification tasks in traditional artificial neural networks). The stacked-up restricted Boltzmann machines are pre-trained in a greedy, layer-wise unsupervised manner, using contrastive divergence as explained above. Then after pre-training the weights in the network get fine-tuned using backpropagation by utilizing the labeled data. So the model learns in an unsupervised manner with an usually large unlabeled dataset during pre-training and gets fine-tuned on much less labeled data during supervised learning.
That means in pre-training good representations for the unlabeled data are learnt which boost the ability to solve the supervised learning task.[9]
Raina et al. give a very nice example for this [30]: The pixels of unlabeled images are fed as input vectors to the DBN and in the unsupervised pre-training phase, correlations between rows of pixels are detected, which learns the DBN that most pictures consist of lots of edges. This allows the network to represent images as constructs of edges than as sole conglomerates of pixel intensity values. Further, Raina et al. state that using the learnt abstraction together with the labeled data, yields in a higher level representation of the labeled data too, which finally results in a less complex supervised learning task.
The combination of unsupervised pre-training and supervised fine-tuning is referred to as semi-supervised learning. Goodfellow et al. sum up some of the reasons why this type of learning can be very beneficial [9]:

- Supervised learning on a relatively small labeled data set compared to a big unlabeled data set may lead to overfitting, which can often be significantly reduced by incorporating the unlabeled data
- Initializing the parameters of a DBN by unsupervised learning, can have a regularizing effect on the model and may be explained in a way that pretraining helps to find features that “relate to the underlying causes that generate the observed data”[9] and “separate features or directions in feature space corresponding to different causes, so that the representation disentangles the causes from one another.”[9]
- Another, already mentioned point in the paragraphs above is that information about the input distribution can be used to generate better input- to output mappings.

From these findings, Goodfellow et al. infer that unsupervised pretraining enhances the model most likely when the unlabeled dataset is big and the initial, raw data representation leaves a lot of space for improvement. [9]

2.4.2. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are a type of Feedforward Neural Networks with sparsely connected layers. They are now widely used for tasks like classifying images and processing speech. More general, they are especially favorable when the data shows a grid-form like 2D images. The name of the network comes from the mathematical term *convolution* since CNNs apply convolution instead of a matrix multiplication in some layers. For example with images it is an important observation that there is a high correlation between neighbored pixels, meaning there is a spatial relationship that can be exploited to construct more efficient learning architectures. [9] CNNs incorporate these observations into their architecture by forcing "the extraction of local features by restricting the receptive fields of hidden units to be local" [31]
Further, LeCun et al. state that one of the biggest issues of traditional neural networks for processing speech and images is the lack of built-in invariance to local distortions of the input. This is especially a problem since, in the course of input normalization, which is needed so that the neural network can work with the input data, some errors can be caused. [31]
Goodfellow et al. sum up the three concepts that allow CNNs to overcome some of the limitations quoted above:

1. Sparse interactions
2. Parameter sharing
3. Equivariant representations

With the help of the block diagram of a Convolutional Neural Network shown in figure 17, in the following those concepts will be explained. A convolutional layer typically consists out of three stages: A convolution stage, a detector stage, and a pooling stage. For simplification, the pixel vector of a 2D-image will be considered as the input vector and the task is to classify the image into categories.

In figure 18, the process inside the convolution stage is depicted, where the input vector gets convolved with a trainable set of filters. One can think of those filters as feature detectors and convolution extracts the information whether a certain pattern is present or not. One and the same filter is applied to every position in the input and may take a small region around this position to detect the feature in question. The result of this stage is a set of convolutions, with the same dimensions as the initial image but instead an image it is now a map that shows to which extent different areas of the input image conform to a certain pattern.
To allow for parallel processing, the architecture looks like indicated in figure 19, showing locally connected feature detectors. So there are many copies of the same feature detector, all with the same weights, connected to different fields of pixels. This stage creates a set of linear activations by combining two of the above-listed principles, namely sparse interaction by locally connected feature detectors and parameter sharing between those, the third principle is a direct way of the described way of parameter sharing: Equivariant representation means that if the input changes (for example due to a shift of pixels in the image), there will be a change of the same magnitude in the output.
The next stage, the detector stage, is needed for normalization to simplify learning and for that, the linear activations of the first layer are transformed by a non-linear activation function like rectified linear activation function (gets rid of negative values).

The third and last stage utilizes a very clever trick called *Pooling:* It is a type of subsampling where the elements of a window of a certain width and height (often2x2) is transformed for example by the MAX-function so that only the maximal value that means the area with the highest conformity to the filter pattern gets into the reduced feature map, which has then roughly only one quarter of the original size. This procedure not only allows for a much less resource intensive further processing than with the original size, but also leads to the nice property, that for recognition or classification tasks, the network is not as sensitive to the exact location of a pixel, which is for example very helpful for handwriting recognition.
Now, after all the three main stages of a convolutional layer are run through, the feature maps can be further reduced and abstracted by passing them through a stack of subsequent convolutional layers.
Finally, having gone through all convolutional layers, the elements of the feature maps are the inputs to a fully connected neural network. If we consider again an image classification task of two classes- a technical apparatus and a landscape, then there are some features like certain features that speak more for the apparatus and others that make an image of a landscape more likely to be the original input image. However, this neural network gets trained by backpropagation and in the end of the training process, the knowledge learnt about the discriminative power of the extracted features is incorporated into the weights of the neural network. [9] [26] [31] [32]

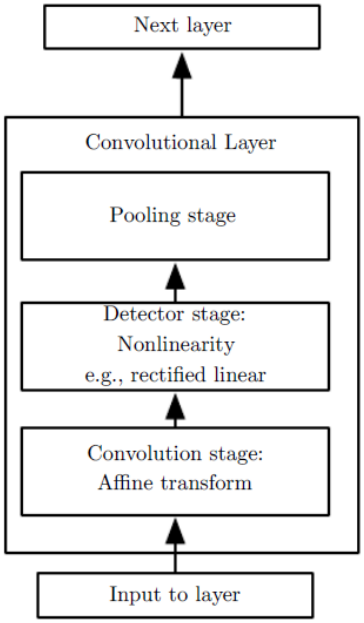

Figure 17 Block diagram of a CNN [9]

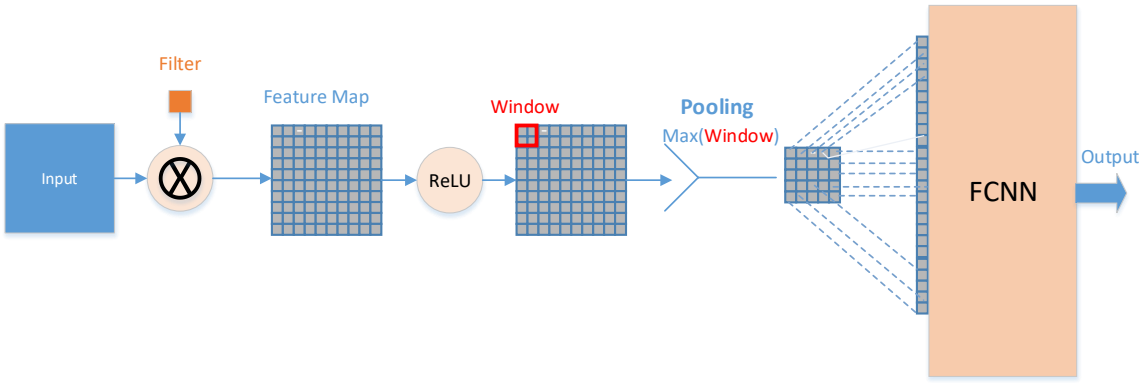

Figure 18 Process diagram of a CNN

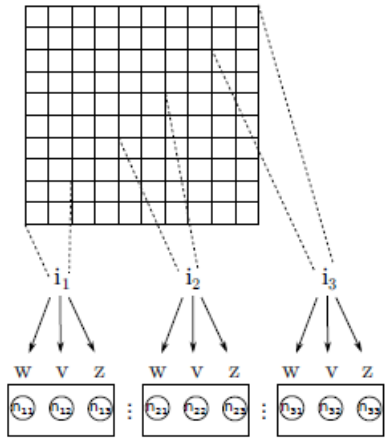

Figure 19 Feature detectors in a CNN [32]

2.4.3. LSTM Recurrent Neural Network

As introduced in the previous sections, it is very difficult to impossible to train general RNNs to work well with sequence predictions where it is needed to look back a lot of timesteps. Goodfellow et al. write to that: "Specifically, whenever the model is able to represent long-term dependencies, the gradient of a long-term interaction has exponentially smaller magnitude than the gradient of a short-term interaction. It does not mean that it is impossible to learn, but that it might take a very long time to learn long-term dependencies because the signal about these dependencies will tend to be hidden by the smallest fluctuations arising from short-term dependencies."[9]

Long-Short Term Memory Recurrent Neural Networks are an approach to solve the problem of remembering and learning from information that occurred a long time ago. For that in the LSTM architecture, the simple hidden units of RNN are replaced by more complex units, also called LSTM memory blocks, which are equipped with a memory cell, which state is controlled by multiplicative gates. In figure 20, a LSTM block is shown: It basically consists of 3 different gates, namely the input-, forget- and the output gate (the black circles indicate multiplicative joints), one input neuron and the memory cell with a linear feedback loop containing the current state. The gates are usually neurons with a non-linear sigmoid activation function, with their input weights controlling their output. So for example, the input gate has a logistic sigmoid activation function, thus its activations vary between zero (gate com-

pletely closed) and one (gate open), controlling to which extent new information that is fed in via the input neuron shall be gated through and considered for computing the current state of the cell. Accordingly, the forget gate controls whether the state of the cell shall be completely retained, modified or completely overwritten with the newly gated input. Finally, the output gate decides which parts (considering a vector) or with what intensity the state of the cell shall be propagated to the next cell. [33] [32] [34]

In figure 21 an inside view of the mathematical operations within an LSTM layer is given. Equation (2.23) describes the function of the forget gate f_t which has sigmoid activation and takes into consideration the last hidden state h_{t-1}, which was propagated by the preceding layer, and the input, both weighted with the weights in W_f, as well as a bias b_f.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.23}$$

The two other gate functions (the input-gate (2.24) and the output gate (2.25)) are formed analogously:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.24}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.25}$$

The update candidate for the cell state gets calculated with a tanh activation function as shown in equation (2.26):

$$C_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C] \tag{2.26}$$

Equation (2.27) is very important, since it shows that the current state is built by a linear operation, in particular, the sum of the parts of the previous cell state that shall remain (as decided by the forget gate) and the parts of the update candidate that are considered as relevant by the input gate. Without further going into the mathematical details, realizing the propagation of information through time with a linear operation is a main reason for the success of LSTMs.[9]

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t \tag{2.27}$$

Finally, as described above, the hidden state which is propagated to the next cell is determined by the product of the result of the output gate function with the output of the tanh-unit that is fed with the current cell state C_t:

$$h_t = o_t \cdot \tanh(C_t) \tag{2.28}$$

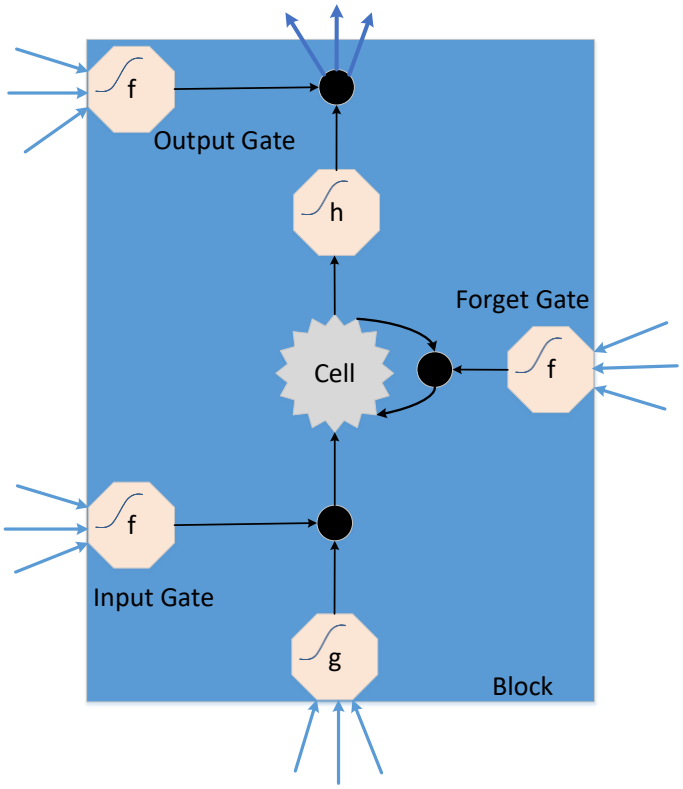

Figure 20 Diagram of an LSTM cell block adapted from [33]

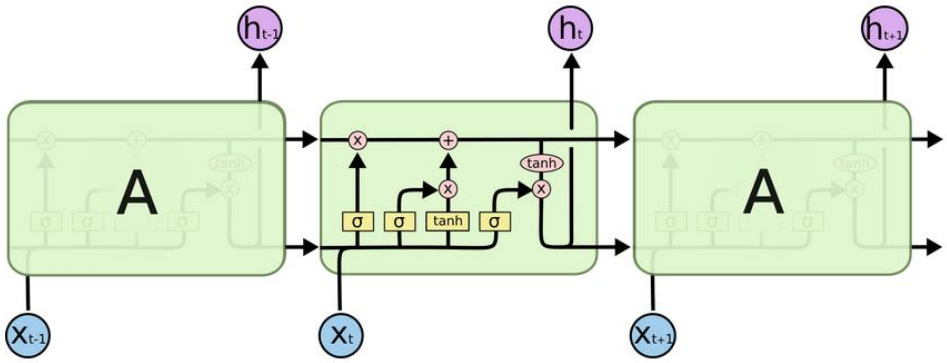

Figure 21 Recurrently connected cell-layers of an LSTM [35]

3. Related Work

The initial search for the related work literature was conducted via Google Scholar given the following search terms:

Forecasting literature with machine learning approaches:

- “Deep Learning for Time Series Forecasting/Prediction”
- “LSTM/Long Short Term Memory time series forecasting/prediction”

State-of-the-art assessment in electricity demand forecasting:

- “LSTM electricity/energy demand forecasting”
- “Electricity /energy demand forecasting”
- “Demand forecasting”

The articles were retrieved from the respective journal databases and considered for research given the following criteria:

- Relevance to the topic:
  - Time series or sequence prediction related research
  - Deep Learning / machine learning approach or statistical model that incorporates multiple seasonalities
- Quantitative Studies
- Peer Reviewed

Gers and Schmidhuber [36] state that LSTM’s are able to outperform more traditional approaches on complex time series but fail to do so when applied to simpler problems, like the Mackey-Glass series or a chaotic laser data set from a contest at the Santa Fe Institute. They treat the LSTM as a solely autoregressive model with only the very last time-step as input at every time step in a walk forward prediction scenario. In contrast, they train and test the competing models e.g. a multilayer perceptron with a time window, thus letting them directly access several past time-steps for prediction. Further, they evaluate stepwise versus iterated training and prediction. Gers and Schmidhuber explain that the autoregressive LSTM approach is inferior to the time window approaches since it cannot access historical data as part of its inputs but instead has to learn the extraction and representation of a Markov state.

In the paper of Qiu et al.[37] an ensemble model of deep belief networks is proposed for regression and time series forecasting. Amongst other datasets, they compare the one-step-ahead prediction performance of the ensemble model on an electricity load dataset and the Mackey-Glass time series. The competitor models consisted out of Support Vector Regression (SVR), a Feedforward Neural Network(FNN), a conventional Deep Belief Network and an ensemble model of FNNs. The findings induced that the ensemble models have an edge on the non-ensemble models and that deep networks outperform the more shallow architectures, as well as the SVR on the Mackey-Glass time series. For both datasets, they chose a train-to-test split of three to one. The performance measures used to compare the results were the RMSE and the MASE. Regarding the load demand dataset, findings where not so clear as the SVR performed significantly better than the neural network models, only the proposed DBN-ensemble model could beat the SVR. For the energy load dataset half-hourly time steps were

used with a time window approach, considering the last 24 hours as input and the single next time step as output. In their outlook, they also state that more advanced optimization algorithms for parameter selection have to be developed to yield better results.

Another evaluation of deep learning methods in the area of short-term load forecasting was conducted by Busseti et al. [38]. Hourly data and a not clearly specified input time-window for predicting the next hour was used. Their empirical analysis resulted in deep recurrent neural networks showing the best performance with regards to the benchmark models used and the RMSE as error measure.

Cinar et al.[39] employed sequence-to-sequence prediction with content attention LSTM networks for univariate and multivariate time series forecasting. The content attention is a modification to the normal LSTM and shall improve the LSTM's ability to reuse input sequences for the output prediction by learning attention weights that let the LSTM focus on the most promising parts of the input history for predicting the output. In this paper the methodology is very well documented and Cinar et al. use six different datasets of univariate and multivariate nature. The use of a training-validation-test split is described and 75% of the data is used for training-validation and the last 25% are used for testing. Within the training-validation set, again a fraction of three quarters is used for training and the remaining quarter for validation.
They concentrate on short-term forecasts up to 7 time steps at the given sample rate into the future. They do a grid search over several LSTM parameters over a not closely defined grid and do not optimize the architecture besides the number of neurons in their work.

Bianchi et al.[24] performed a comparative study amongst various Recurrent Neural Network- architectures (Simple RNNs ,Echo State Networks, LSTMs, Gated Recurrent Units, Nonlinear AutoRegressive with eXogenous inputs). They conduct the analysis on 3 artificial datasets and 3 real-world datasets for short-term load forecasting, two of them comprising out of electricity demand series. Bianchi et al. provided a quite clear description of their methodology which shall be presented with the time series describing the electricity demand of Rome and neighboring regions at a resolution of 10 minutes. They train the RNNs to predict 24 hours ahead, which resolves to 144 time steps at the given sampling rate. They conducted minimal preprocessing of the data by differencing the series by the daily period and standardized the series' values. They use four months of data for training and hyperparameter optimization (three for training and one month for validation) After finding the best hyperparameter settings they use these four months to train the final models and compare the performance on the fifth month. The training approach was not clearly specified (e.g. window size, method). The prediction accuracy was evaluated on basis of the Normalized Root Mean Squared Error. Summed up, their analysis showed that depending on the time series, different methods have the lead. Their experiments further showed that the more complex RNNs like the LSTM did not show significantly better performance than the basic RNN architecture. In the conclusion, they theorize that gated RNNs like LSTM and GRUs are excelling in settings "where temporal dependencies are more irregular than in the dynamical systems underlying the load time series". [24]

In the book "Modeling and Forecasting Electricity Loads and Prices", Weron et al. review papers published and consider both, statistical and machine learning approaches. Further, a case-study on the

usage of statistical methods is given on the example of load time series. They noted that the reviewed literature delivered contradictory results on the usefulness of Artificial Neural Networks in time series forecasting: While Hippert et al.[40] find that the large feedforward neural network had the best out-of-sample performance compared to more conventional methods, like exponential smoothing or regression, Taylor et al.[41] found a simple statistical method (Double Seasonal Holt-Winters (DSHW) / Double Exponential Smoothing) to outperform the neural network model.

The above-mentioned paper by Taylor et al.[41] from 2006, is especially interesting since it compares widely used univariate methods for forecasting electricity demand up to a day ahead. Regarding the choice of univariate methods for forecasting they write “In the short run, the load is mainly influenced by meteorological conditions, seasonal effects (daily and weekly cycles, calendar holidays) and special events. Weather-related variation is certainly critical in predicting electricity demand for lead times beyond a day-ahead… when the interest is in shorter lead times, a univariate model will be sufficient. Indeed, univariate models are the norm for lead times up to about six hours ahead, and, due to the lack of readily available weather forecasts, they are sometimes used for longer lead times.” A very nice aspect of this work is that it uses the Naive forecast(with and without error model) as a benchmark. Other methods considered besides the neural network and DSHW, were Seasonal ARMA and Regression with Principal Component Analysis. The two load time series used for the forecaster evaluation are 30 weeks of hourly electricity demand data for Rio and half-hourly demand data for England and Wales. For both, the first 20 weeks were used for parameter estimation and the test-set consisted out of the remaining 10 weeks. The forecast-accuracy up to 24 hours ahead was reported in terms of the MAPE.

Concluding the literature research, the available literature for time series forecasting with deep learning seems to indicate an improved performance compared to the benchmarks in the evaluation. The research of Taylor et al. considered a single layer feedforward neural network for forecasting, which makes curious about the outcomes if one used an RNN or an LSTM. Load time series forecasting is a popular field of research, not only for its practical relevance but seemingly also for the reason that such data is easily accessible and publicly available. This is also the reason why, in the empirical part of this thesis, also a load time series is used for the demonstration of the methodological approach and the performance evaluation.
Considering the paper of Gers and Schmidhuber[36], an analysis of LSTM’s trained with a time-window approach is of interest, as also the benchmark methods get trained in this way. There are lots of promising results in the literature for applications of sequence-to-sequence recurrent neural networks (see Cinar et al.[39], Lipton et al.[42], Graves et al.[43]) and the training in a time-window approach with a multistep-forecast is a comparable problem. Thus, in this thesis, a time-window approach will be used for training, as explained in chapter 4.
As reported, Qiu et al. [37] state that more advanced optimization algorithms for parameter selection have to be developed to yield better results. Regarding the parameter tuning in general, in several papers, it is not how exactly the optimization has been conducted and if the optimization of the parameters of the selected benchmark methods has been done with the same efforts as for the deep learning methods. Further, for the parameter search the computational burden seems to be still quite high, which often results in a relatively narrow parameter grid for the search. As a result of the literature research, this thesis lays importance on a methodological and objective way to tune the parame-

ters of the deep learning methods as well as the conventional approaches. Moreover, not only the conventional model parameters shall be tuned but also the number of layers, which is rarely done in the sighted literature.

Regarding the used benchmark methods, considering energy demand forecasting, it seems that often better benchmarks than the applied could have been used, as often ARIMA is used as a benchmark but as also the studies of Taylor et al.[41] show, that often double seasonal exponential smoothing seems to be better suited for forecasting this type of time series. Further, it would be valuable to have the naive forecast as a standard benchmark for the one-step-ahead forecasts, since its creation is easily understood by non-experts and is extremely simple to produce (see 4.5.1 - Naive Forecast).

Hippert et al., who conducted an exhaustive study on conventional (no deep architectures) neural networks, observed two significant lacks in the academic literature that impact the credibility of the results and write to that: “First, most of the papers proposed NN architectures that seemed to be too large for the data samples they intended to model, i.e., there seemed to be too many parameters to be estimated from comparatively too few data points. These NNs apparently overfitted their data and one should, in principle, expect them to yield poor out-of-sample forecasts. Secondly, in most papers the models were not systematically tested, and the results of the tests were not always presented in an entirely satisfactory manner.” [4] While during the literature research for deep learning the first point of critique was at least not obvious, the second point of critique seems to still remain, especially regarding the methodology where they further criticize the use of benchmarks, seems to remain in modern literature.

4. Methodology

4.1. Process Model

In the following, the wider framework is developed as a hybrid model (figure 23) from Hyndman and Shmueli (figure 22). Starting with the process model after Hyndman, in the first stage the problem needs to be clearly defined, which is especially important if more than one person is working on the problem or the forecast is produced for one or more stakeholders. This step involves communication with stakeholders of the forecasting project and specification of the usage scenario of the forecast. Important details of the specification include the usage scenario, time horizon of the forecast, available data, forecast interval, time and resource constraints.
The second stage addresses the data collection, which refers to the collection of the available data for feature selection and input for the forecasting method on the one side, and to the collection of the expertise of all stake holders on the other side.
During the third step, the previously collected data has to be analyzed and characterized. Regarding time series, especially seasonality, trend, correlations between variables and explanatory power of variables, as well as outliers are of interest.
Step four deals with model selection and model fitting. During the selection process, the parameters of different parameter sets of models and/or different models are fitted to the historical training data in the train-validation set and their performance is then validated on the validation data. Subsequently, the best performing model/ best performing parameter set is then selected for evaluation on the test data set in the next step.
In the fifth and last step, the previously selected model and its parameter set is evaluated on the unseen test data with a set of selected performance measures.

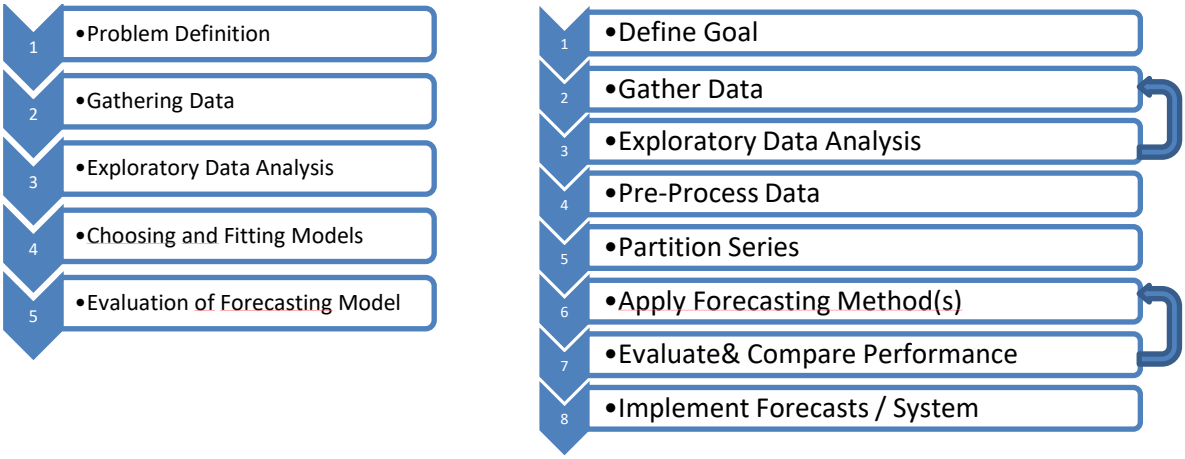

Figure 22 Left: The 5 Steps of a Forecasting Project adapted from Hyndman [44], Right: Time Series Forecasting Process in 8 steps adapted from Shmueli [45]

In the process of Hyndman there is no step for data pre-processing and data partitioning into train-valid and test dataset, as it is present in the 8-step process of Shmueli. Moreover, the Shmueli process model indicates by the backward errors that after initial data exploration it can occur that more data is needed. On the other hand, the Shmueli process model has no model fitting and selection step. Since the data pre-processing and the partitioning of the historical data into a train-validation set and a test data set are as important for this thesis as the model fitting and selection step, a hybrid model of the Hyndman and the Shmueli model is used as depicted in figure 23.

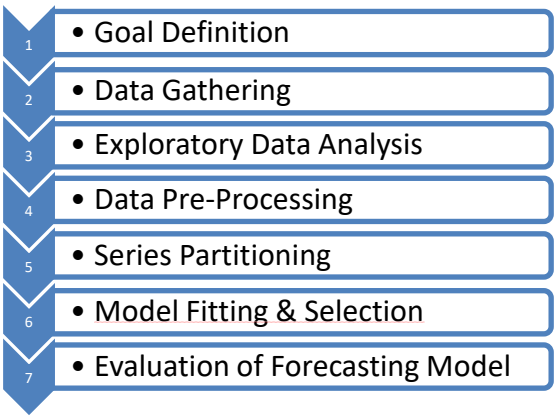

Figure 23 Overall Time Series Forecasting Process in 7 steps used in this thesis

The sixth step in the used process model in figure 23 is of special importance to this thesis since choosing and fitting the model for deep learning methods in the time series forecasting domain is due to resource and time constraints not always an easy task since the parameter space that has to be searched for fitting the models can be very large and there is not much literature on model selection in this domain yet and this step is often not the focus of the available related literature in this area as outlined in chapter 4.

4.2. Data Partitioning

The historical data gets partitioned into the train-valid-set and the test-set. The train-valid partition consists of the train- and the validation subset. The train-valid-set gets chosen for model selection as explained in the next section and the test-set is withheld for the evaluation of the winner model in the model selection phase. The nomenclature is not always clear in the literature but this seems to be the most consistent one and aligns, along with many other papers and books, with Ripley in [46] , as well as Russel & Norvig in [16].
For this thesis, the data is partitioned as specified here: 80% of the data is used for the train-valid-set and the remaining 20% of the data is used for final evaluation as test-set. Within the train-valid-set, the data is again partitioned for every block with 80% for the train-set and 20% of the valid-set (See modified blocked cross-validation in the next section).

4.3. Model Selection

The biggest challenge in the model selection and evaluation of deep learning models in the time series forecasting domain is that the models are computationally demanding to train and that the time series can be very long, which makes also the evaluation a resource-intensive task. In the next paragraphs, an approach to deal with these issues is presented.
As already described in the previous chapter, in the model selection phase a set of one or models with different parameter sets gets evaluated against each other. In conventional machine learning scenarios, it is best practice to use cross-validation(CV) for model selection. Cross-validation (figure 25) has the advantage over the last block validation (figure 24) that the whole available data gets used for training and evaluation and the results of this method tend to be more robust than with last block validation. Also in the time series domain cross-validation gets used for evaluating auto regressions on time series. Bergmeier et al.[47] state, that in many cases standard cross-validation is not suitable for time series since it assumes that the data is independent and identically distributed, which is not the case if the training and validation sets are randomly chosen (as is common in CV) . Furthermore, they criticize, that time series might be the result of a process that evolves over time. According to Bergmeier et al., last block validation would solve these issues but has the already mentioned downside of not utilizing the whole data set, which can be problematic in terms of robustness, limited validity, thus eventually leading to a model choice with little ability to generalize on unseen data.
But even last block evaluation does not solve all data dependency issues, since data from the test set may depend up to a number of lags on the training data.
A possible approach to deal with this form of dependency is not to use the part from the training set for training up to a distance at which independence for the test data is guaranteed.[48] In the study of Bergmeier et al. such approaches are referred to as non-dependent cross-validation.

Another very important consideration that must be taken when dealing with time series is stationarity, which in the weak form can be broken down into a time series with constant mean and variance throughout the series. As one of the CV's assumptions is the identical distribution of the data, the time series has to be stationary. Sometimes non-stationarity can be removed by differentiation and the Dickey-Fuller unit root test can be used for checking the series for stationarity.[47]

However, often non-stationarity cannot be removed and further, the Dickey-Fuller unit root test also has its limitations regarding computability with very long lags.
For using CV on non-stationary datasets a reasonable approach is to train and validate the model on small enough subsets which are stationary (see [48]).

Figure 24 Last Block-Validation

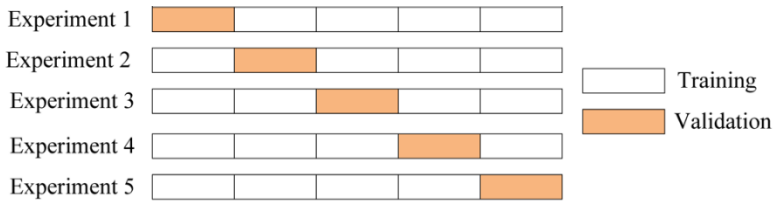

Figure 25 Cross-Validation

Bergmeier et al. have conducted an exhaustive study where they analyzed different CV strategies for time series predictor evaluation on stationary time series. The strategies investigated in this study were, amongst others, standard 5-fold cross-validation (data gets randomly partitioned into 5 sets), last block evaluation, non-dependent cross-validation and so-called blocked cross-validation. For blocked cross-validation, the data is not randomly partitioned but sequentially, to make use of all available data, the set is further partitioned in a canonical way (see figure 26).

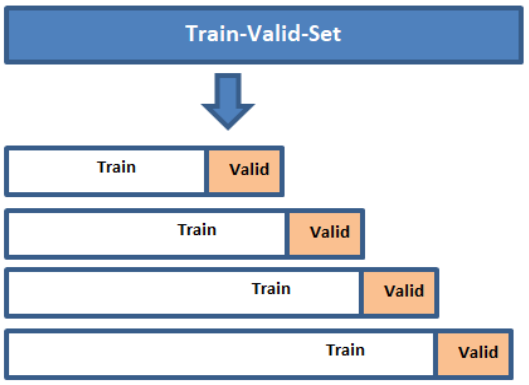

Figure 26 Canonical Blocked Cross-Validation

4.3.1. Hyperparameter Optimization

Machine learning models have various hyperparameters that need to be tuned to yield good prediction results. There are different approaches to tune these parameters, the most widely used are:

- Manual Tuning: The model gets tested with a certain parameter set against different metrics, often also with the help of graphics such as a plot of the loss function and the prediction error. This can be a good method if an expert conducts this procedure since with the help of expert's experience one can often significantly cut down the parameter space to a meaningful one that can then be evaluated in relatively little time.

- Grid search: With this procedure, the parameter space is defined by a search grid, which usually defines a value range and a step size for every parameter that shall be optimized. Subsequently, every possible parameter combination gets (usually not manually) evaluated against the given performance metrics. The biggest disadvantage of this method is that the search is the very resource intensive since an exhaustive search over the whole parameter space is conducted. On the other hand, this exhaustive method is effective by nature in the sense that it finds the optimal solution for sure. Nevertheless, as with all methods, there is still expert knowledge necessary to provide the algorithm with a meaningful grid so that grid search can be done within the time constraints and the parameter grid is not too restrictive so a globally optimal solution can be found.
- Random search: Tries to tackle the problem of the high resource demands of grid search by choosing parameter combinations from a random distribution. Many types of research have been done in this area and Bergstra and Bengio state "Granting random search the same computational budget, random search finds better models by effectively searching a larger, less promising configuration space."[49]

Since the problem is that with long time series and deep learning the computational complexity gets high very quickly and this thesis has to be conducted within a reasonable time and with given resource constraints, it is problematic, that with random hyperparameter optimization, potentially a large fraction of the time the search is executed in regions which are non-optimal and the search does not get more effective over time.
Therefore, in this thesis, another approach is used, namely Bayesian Optimization (BO). The idea behind BO is to estimate the parameter response surface, which tells how the objective function that shall be optimized reacts to different parameter settings and then go the directions that have the maximum likelihood to arrive at a global minimum/maximum of the objective function. There is no closed form of the objective function available but by trying different parameter settings one can get observations of it. To allow the algorithm to do a first estimation of the response surface, an initial number of runs with different hyperparameter settings $\lambda_1 \ldots . \lambda_n$ are evaluated and the yielded accuracies are recorded. A probabilistic regression model is fit to recorded accuracies and the respective parameter set is learned and in the next steps used as a surrogate of the response surface. The probabilistic regression model, which is often a Gaussian Process, allows to systematically explore the parameter space and choose hyperparameter settings for the next evaluation that are likely to improve the accuracy.[50] [51]

Modified Blocked Cross-Validation

With the canonical CV as illustrated in figure 26, the training set for fitting a parameter configuration gets consecutively larger, this is problematic since with increasing size of the training set also the time needed for training increases. Especially in the deep learning domain and with a myriad of parameter combinations, this type of training can create a significant bottleneck.
To circumvent this issue and still being able to get a degree of generalizability by utilization of the whole available test data, a modified form of the above presented blocked CV is used:
It works like depicted in figure 27, where the different train-valid sets are not ordered canonical but side by side, without any overlapping regions. The number of blocks and the block-size shall be chosen in a way that parallel hardware can be maximally exploited and that the training and validation time

stays feasible. The rationale behind this in simple words is that if a model performs very badly in comparison to other models on a smaller subset of the data, it is unlikely that it will outperform the others on a larger dataset. The core idea, namely that the evaluation results of a machine learning algorithm on a subset of a large dataset can be used to cut down the model runtime, is already reported to be useful by authors in other fields of research such as neural language processing and others (see [52] , [50]).

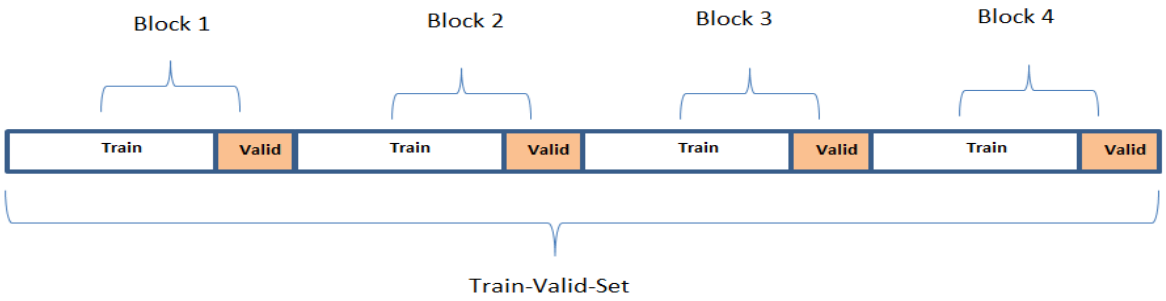

Figure 27 Modified Blocked Cross-Validation

4.4. Model Evaluation

For the model evaluation of the deep learning models the three criteria described by Adya and Collopy[53] for evaluation of neural network models are used:

1) Neural network forecasts have to be compared to well-accepted reference models
2) The comparison has to be done on an out-of-sample set
3) Enough predictions (at least 40) have to be done to make valuable conclusions.

Further, the checklist which was developed by Armstrong[54] is cited in the following and is obeyed and referred to in the subsequent sections:

Using reasonable alternatives
• Compare reasonable forecasting methods
Testing assumptions
• Use objective tests of assumptions
• Test assumptions for construct validity
• Describe conditions
• Match tests to the problem
• Tailor analysis to the decision
Testing data and methods
• Describe potential biases
• Assess reliability and validity of data
• Provide easy access to data
• Disclose details of methods
• Find out whether clients understand the methods
Replicating outputs
• Use direct replication to identify mistakes

- Replicate studies to assess reliability
- Extend studies to assess generalizability
- Conduct extensions in realistic situations
- Compare with forecasts obtained by different methods

Assessing outputs

- Examine all important criteria
- Prespecify criteria
- Assess face validity
- Adjust error measures for scale
- Ensure error measures are valid
- Avoid error measures sensitive to degree of difficulty
- Avoid biased error measures
- Avoid sensitivity to outliers
- Do not use R2to compare models
- Do not use RMSE
- Use multiple error measures
- Use ex ante tests for accuracy
- Use statistical significance only to test accuracy of reasonable models
- Use ex post tests for policy effects
- Obtain large samples of independent forecast errors
- Conduct explicit cost/benefit analysis

4.4.1. Out-of-Sample Testing

The evaluation of the forecasting models in the hold-out test data is done in a fixed-size, rolling-window walk-forward scenario as defined by Tashman [55]. The forecast rolls forward by the forecasting horizon together with the rolling window at every forecasting step. That means, with progressing time more historical data becomes available and subsequently, this data gets into the rolling window which is a FIFO-Queue (First In - First Out), as the oldest data at the beginning of the window gets pushed out as newer data comes in. The forecast horizon can have one to multiple steps. The described procedure is shown in figure 28.

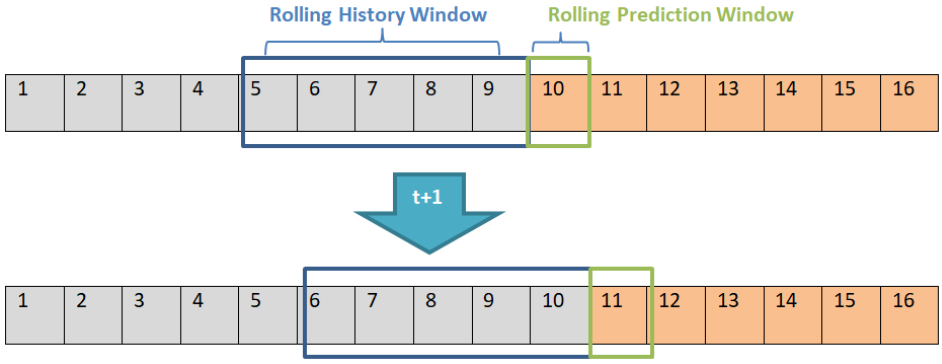

Figure 28 Rolling Window Forecast

4.4.2. Multistep Forecasting Strategies

In this section, the used strategies for multistep forecasting are specified according to Bontempi et al.[56]. In essence, they are the recursive strategy and the Multi-Input-Multi-Output (MIMO) strategy. With the recursive strategy, the output of a one-step-ahead forecast is fed back as input, constituting the last element of the history window, for the next forecast. A disadvantage of this method is that the estimation error gets amplified, due to the feedback loop.
In contrast, the MIMO strategy forecasts multiple steps at once and its biggest advantage is that it does not neglect the stochastic dependencies between the outputs and gets rid of the accumulation of errors.[57]

4.4.3. Performance Metrics

The choice of performance metrics is a crucial task for a meaningful performance evaluation. According to the literature research in chapter 3 and Tofallis[58], two of the most widespread error measures (absolute and relative error measures) for time series forecasting are used, namely the Root Mean Squared Error (4.1) and the Mean Absolute Percentage Error (4.2). Further, a relatively little-used performance indicator, measuring the Mean Directional Accuracy (4.3) is applied, since it allows to intuitively compare the models regarding the ability to follow the direction of the trend and to predict directional changes. As Armstrong suggests, the RMSE shall not be used because it is a scale-dependent error measure. If forecasting models on several different time series are evaluated it is important to use scale independent performance measures for comparison only. In the case of this thesis all models are compared by evaluation on the same time series, therefore scale independency has not to be taken into consideration. For the formulas listed below, the $\hat{y}$ denotes the predicted value and y the actual value. Though it is often done, using solely the MAPE alone as selection criteria is not advisable since it prefers forecasts that under-forecast as Tofallis describes in his analysis.[58] Nevertheless, the MAPE has become an industry standard in load forecasting since it is easily interpretable as it "captures the proportionality between the forecast error and the actual load"[41].
By using different error measures for evaluation, the procedure is in-line with Armstrongs' checklist.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.1}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4.2}$$

$$MDA = \frac{1}{N}\sum_{t} I \tag{4.3}$$

$$I = \begin{cases} 0 & if \quad sign(y_t - y_{t-1}) \neq sign(\hat{y}_t - \hat{y}_{t-1}) \\ 1 & if \quad sign(y_t - y_{t-1}) \equiv sign(\hat{y}_t - \hat{y}_{t-1}) \end{cases}$$

4.5. Benchmark Methods

According to the checklist by Armstrong [54] and the evaluation criteria by Adya and Colopy [53], the benchmark methods need to be well-accepted and reasonable. Therefore, the selection of the benchmark methods was done by using methods that are used as benchmarks in deep learning research and methods that show top performance in the energy forecasting domain (see the related literature in chapter 3).

4.5.1. Naive Forecast

The naive forecast is the simplest form of forecasting method and its rationale is that the next datapoint (the one to be forecasted) will not be too far away from the very last datapoint. Thus, without any further knowledge incorporated this method uses the last datapoint as the forecast for the next time step. Though its simplicity, especially for one-step-ahead forecasts, for many problems this can actually be a very appropriate method and much more complex and resource intensive methods often have a hard time beating the naive forecast. Nevertheless, in most papers, this simplest form of a forecasting method is not considered as a comparison measure, although it is a very intuitive and valuable benchmark. As depicted in figure 29, the naive forecast for the one-step-ahead forecast is nothing but the time series to be predicted, shifted backward by one time step.

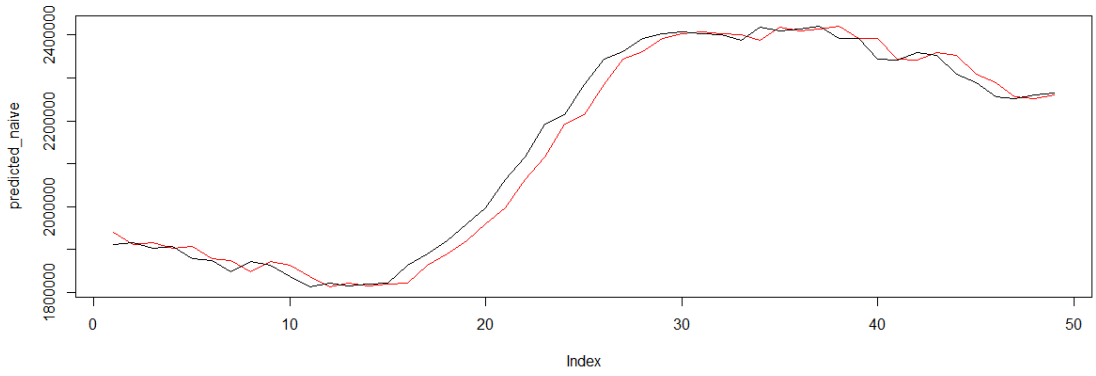

Figure 29 Naive Forecast(Red)

4.5.2. Support Vector Regression

Support Vector Regression is done with Support Vector Machines which theoretical foundation reaches back to the work of Vapnik in the 1960's. Also due to the lack of computing power, it took around 30 years till Vapnik and his colleagues presented the SVM in the form we know it today.
SVMs are based on statistical learning theory "which characterizes properties of learning machines which enable them to generalize well on unseen data". [59]
To explain the origins of the name of an SVM, I refer to the more common task for SVMs than regression, which is clustering/classification. In figure 30 a problem is shown where the two clusters can be linearly separated by a line into the two clusters w1 and w2. The SVM's goal is to find a separation line which has maximum distance to the two clusters and is equidistant to them. This separation line is shown as the solid line between the pair of dashed lines. The both dashed lines, also called support hyper-planes, are supported on the points marked with a circle around them, which are the points of their respective cluster which are closest to the other cluster. From the fact that the circled points are the feature vectors with which help the hyperplanes are constructed, they are called *support vectors*.[5]

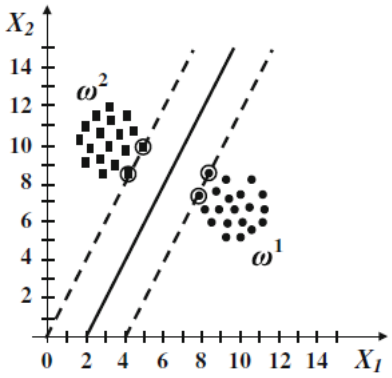

Figure 30 Two clusters and their supporting vectors [5]

Goodfellow et al. [9] state that for class prediction, the SVM uses a linear function in the form of equation (4.4), where *x* denotes the training examples and *w* the weight vector. It predicts the positive class, when *f(x)* is positive and the negative class, when *f(x)* is negative.

$$f(x) = w^T x + b \tag{4.4}$$

An essential finding to SVM's is that the function *f(x)* can be rewritten in the form of a dot product of *x* only, where α is a vector of coefficients and $x^{(i)}$ is a single training example:

$$w^{\mathrm{T}} x + b = b + \sum_{i=1}^{m} \alpha_i x^{\mathrm{T}} x^{(i)} \tag{4.5}$$

Further, Goodfellow et al. show that by substitution of *x* in the above equation with the so-called *kernel k* from equation (4.6), which replaces x with the output of a feature function ϕ, one yields the formula in (4.7) :

$$k(x, x^{(i)}) = \phi(x)^T \phi(x^{(i)}) \tag{4.6}$$

$$f(x) = b + \sum_i \alpha_i k(x, x^{(i)}) \tag{4.7}$$

Equation (4.7) allows the SVM to transform the input x with the nonlinear kernel function k into an m-dimensional feature space, in which it creates a linear model. This procedure is referred to as kernel-trick and for better insight I want to cite Goodfellow et al. explaining the effectiveness of the trick: ”First, it allows us to learn models that are nonlinear as a function of x using convex optimization techniques that are guaranteed to converge efficiently. This is possible because we consider ϕ fixed and optimize only α, i.e., the optimization algorithm can view the decision function as being linear in a different space. Second, the kernel function k often admits an implementation that is significantly more computationally efficient than naively constructing two $\phi(x)$ vectors and explicitly taking their dot product.”[9]

Now, Support Vector Regression (SVR) works very similarly to the above-described case, with the difference that the outputs of *f(x)* are not distinct classes anymore but can take a value out of a continuous value range. SVR tries to find a function that has at most a deviation of ε from the targets y_i in the training data and is at the same time as flat as possible. Which means that the loss function (right side of figure 31) will ignore errors that are less than ε and only errors greater than this threshold will contribute to the costs.[59]

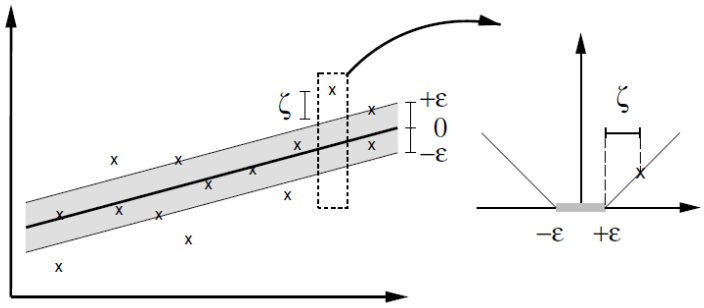

Figure 31 Linear SVM with soft margin loss function (also ε-insensitive loss function) [59]

For the mathematical formulation, in order to give an intuitive explanation of Support Vector Regression, only the linear case is described as it is done by Smola et al. in [59] and summarized in the next paragraphs. To extend the following explanation to the nonlinear case, one would have to formulate it as a dual problem by means of the kernel function as shown in (4.7). For the linear case again, starting from equation (4.4), the term flatness can be explained as seeking for a small weight vector *w*, which can be obtained by minimizing its Euclidean norm $\|w\|^2$. Including the restrictions for errors not to exceed ε, the following convex optimization problem can be stated (adopted from Smola et al. [59] to the notation of Goodfellow et al. [9] with f(xi)=yi for the training examples) :

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|w\|^2 \\ &\text{subject to} \begin{cases} y_i - w^T x_i - b \le \varepsilon \\ w^T x_i + b - y_i \le \varepsilon \end{cases} \end{aligned} \tag{4.8}$$

Since the there are problems where not all pairs (x_i,y_i) can be approximated with a function with a smaller error than ε, or a bigger error than that should be tolerated, it is common to introduce so-called slack variables (ξ_i and ξ_i^*).[59] This leads to the SVR model that is developed in the original paper of Vapnik et al. [60]:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{subject to} \begin{cases} y_i - w^T x_i - b \le \varepsilon + \xi_i \\ w^T x_i + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \end{aligned} \tag{4.9}$$

The formulation in (4.9) also introduces the, together with ε, most important parameter for parametrizing the SVR in the practical section. Namely, the constant C which can take values greater 0 and controls to which extent flatness is preferred over smaller deviations from ε.

4.5.3. Double Seasonal Holt-Winters

The Double Seasonal Holt-Winters (DSHW) method is an adaptation to the standard Holt-Winters method to allow for two seasonalities. As described in chapter 3, the DSHW forecasts have been identified as the best performing method in the univariate load forecasting scenario. Thus, it qualifies as a method to provide a reasonable benchmark.

The following formulas show the Holt-Winters method as described by Winters [61]:
The k-step ahead forecast in equation (4.13) has an additive trend component T and a multiplicative seasonal component as the series level *L* is multiplied by the seasonality index *S*.
The trend's local slope is estimated in equation (4.11) by smoothing the difference of the current level L_t (4.10) and the previous one (L_{t-1}) with the smoothing parameter γ. The local m-period seasonal index S_t (4.12) is the δ-smoothed ratio of the current observation X_t to the current local level L_t. [62]

$$L_t = \alpha\frac{X_t}{S_{t-s}} + (1-\alpha)(L_{t-1} + T_{t-1}) \tag{4.10}$$

$$T_t = \gamma(L_t - L_{t-1}) + (1-\gamma)T_{t-1} \tag{4.11}$$

$$S_t = \delta\frac{X_t}{L_t} + (1-\delta)S_{t-m} \tag{4.12}$$

$$X_t(k) = (L_t + kT_t)S_{t-m+k} \tag{4.13}$$

The above presented Holt-Winters method was extended by Taylor[62] by a second seasonal component, where the first component is denominated by *D* and the second one by *W*, whereas *D* could denote the within day - and *W* the within week seasonalities for example.

The level and trend component are calculated analogously to the standard Holt-Winters method (4.10) and (4.11). Sticking to the exemplary denomination of daily and weekly seasonalities, the s_1-period is the within-day seasonality of the local seasonal index D_t , which equals to the smoothed ratio of the current observation X_t, to the product of the local level (L_t), and the within week seasonality W_{t-s2} (4.14).
Accordingly, the within week seasonality is estimated in equation (4.15). The parameters for this model are commonly estimated by minimization of the sum of squared 1-step-ahead errors from an estimation sample and an initial set of values for level, trend, and seasonalities.

$$D_t = \delta \frac{X_t}{L_t W_{t-s_2}} + (1-\delta) D_{t-s_1} \tag{4.14}$$

$$W_t = \omega \frac{X_t}{L_t D_{t-s_1}} + (1-\omega) W_{t-s_2} \tag{4.15}$$

$$X_t(k) = (L_t + kT_t) D_{t-s_1+k} W_{t-s_2+k} \tag{4.16}$$

4.5.4. Simple Recurrent Neural Networks

The use of a simple recurrent network is of special interest since the LSTM's as described in the previous chapters is used as the deep learning regressor and as explained earlier, it is an advanced form of a recurrent neural network itself. So the comparison of the LSTM to its simpler pendant can give insights to the added value of such networks in the time series domain. Comparing the simpler predecessors is with the more sophisticated approaches is a good rule in general and should be done in all deep learning research. For a detailed introduction to RNNs please refer to chapter 2.3.2.

5. Empirical Part

5.1. Problem Description & Exploratory Analysis

The empirical part of this thesis demonstrates the training, optimization and application of LSTM's for time series prediction on a real-world example and exemplarily compares the performance to more traditional approaches. As motivated in the literature chapter, the short-term prediction of a univariate energy demand time series will be used for this case-study.
The prediction horizon of short-term predictions in the energy sector ranges from 15 minutes up to a day ahead (compare [63]). In the industry, the short-term forecasts are essential for planning the generation unit commitment, energy transfer schedules, and load dispatch. Bunn and Farmer[64] have estimated that with a 1% higher forecasting error, the operating costs rise by £10 million and Pai et al. state on the importance of precise short-term forecasting methods: "Hence, over estimation [sic] of future load results in unnecessary spinning reserve and, furthermore, is not accepted by international energy networks owing to excess supply. In contrast, under estimation [sic] of load causes failure in providing sufficient reserve and implies high costs in the peaking unit. Because buying at the last minute from other suppliers is expensive, international electricity production cooperation requires accurate forecasting of the needs of all participants. However, forecasting the electricity load is difficult, primarily due to the various influences, such as climate factors, social activities, and seasonal factors."[65]

In this study, the forecasting models predictive power will be assessed on one-step and multi-step forecasts, whereby a single step equals to 15 minutes and the multi-step forecasts are 96 step-ahead forecasts, which are equal to a one-day-ahead forecast.
The data used for this endeavor is publicly available and is drawn from the data pool of swissgrid [66]. It consists out of 17372 observations of aggregated energy demand in kilowatt-hours (kwh) with a resolution of 15 minutes from 1.Januray 2017 00:00 till 1.July 2017 00:00. As can be seen in figure 32, the energy demand peaks in end of January and from then on it shows a clear downtrend till mid of April with overall lower weekly highs and lower weekly lows, then a break of the downtrend at the end of April is followed by relatively stable demand lows and highs till the end of June. So to sum up the big picture regarding the trending behavior, there is an upward trend in the first month, followed by a longer downtrend till the end of April and a more-or-less sideward trending, last period.
As it can be inferred from the monthly mean and standard deviations ($\sqrt{Variance}$) listed in table 1, the series is non-stationary according to the weak form of the stationary definition, which postulates that a time series is stationary, when it shows constant mean and variance throughout the time series. Also, the commonly used pre-processing step of first-order differencing did not remove the non-stationarity.

Month	Mean [Mwh]	Standard Deviation [Mwh]
January	1991.96	265.05
February	1839.28	239.15
March	1681.77	239.40
April	1524.60	244.85
May	1493.67	237.51
June	1428.65	240.62

Table 1 Monthly Mean and Standard Deviation for the assessment of stationarity

In figure 33 the various seasonalities that are apparent in this time series can be easily seen. The most important cycles are the weekly and the daily cycle. This is also obvious in the autocorrelation plots in figure 34 and figure 35, where the multiples of 96 (one day) and especially the value of the week before (672 time-steps) show high correlations.

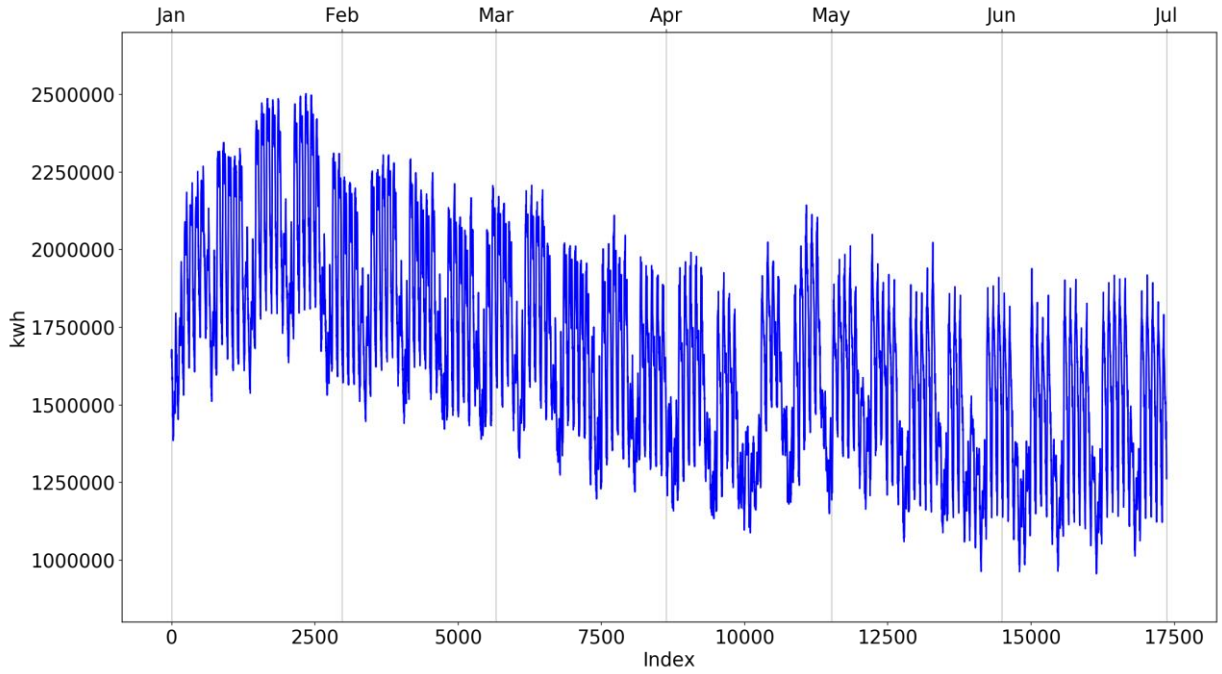

Figure 32 Electricity load time series in kwh of the Swiss energy sector from first of January till first of July 2017 (17372 time steps at a sample rate of 15 minutes)

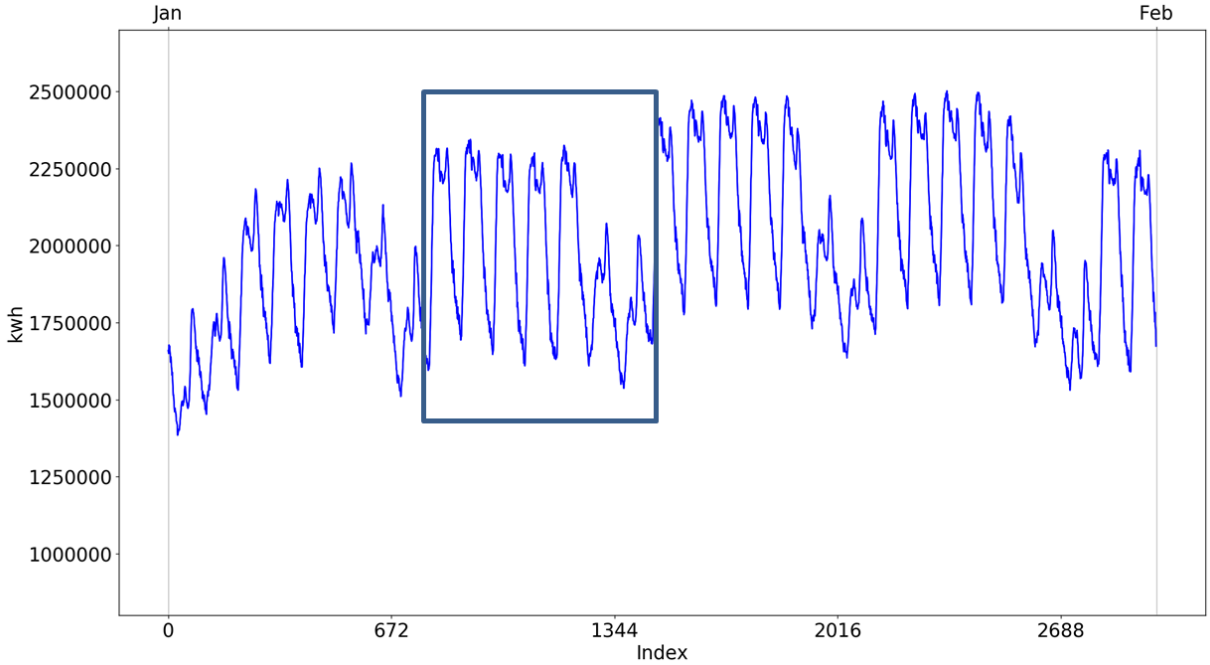

Figure 33 Magnification of the electricity load time series for the first month. The square highlights a week cycle with 5 workdays with a higher energy demand and Saturday and Sunday with a lower overall demand.

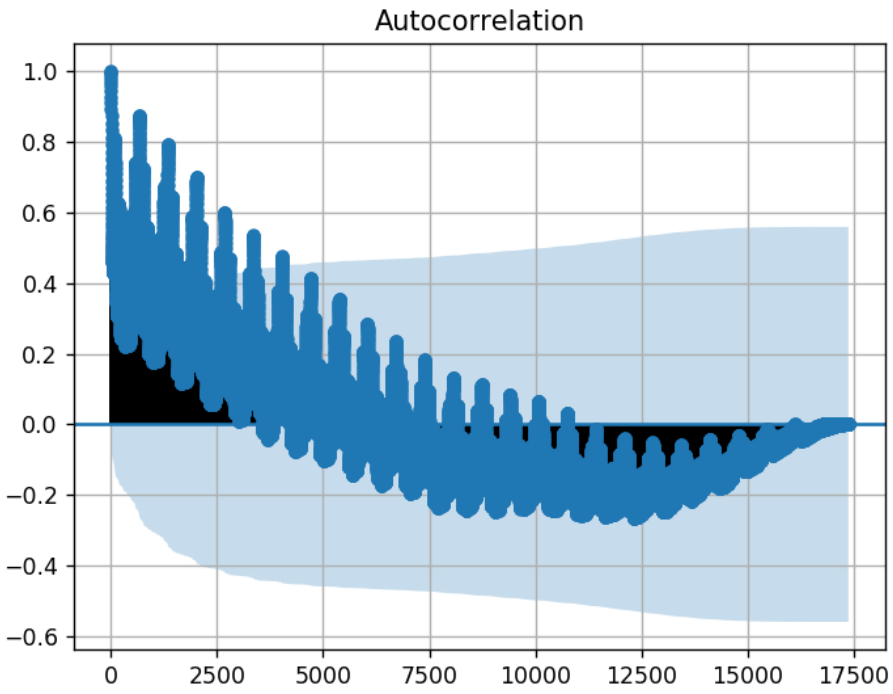

Figure 34 Autocorrelations for all lags of the engergy demand time series, showing the significant lags exceeding the 95% confidence band

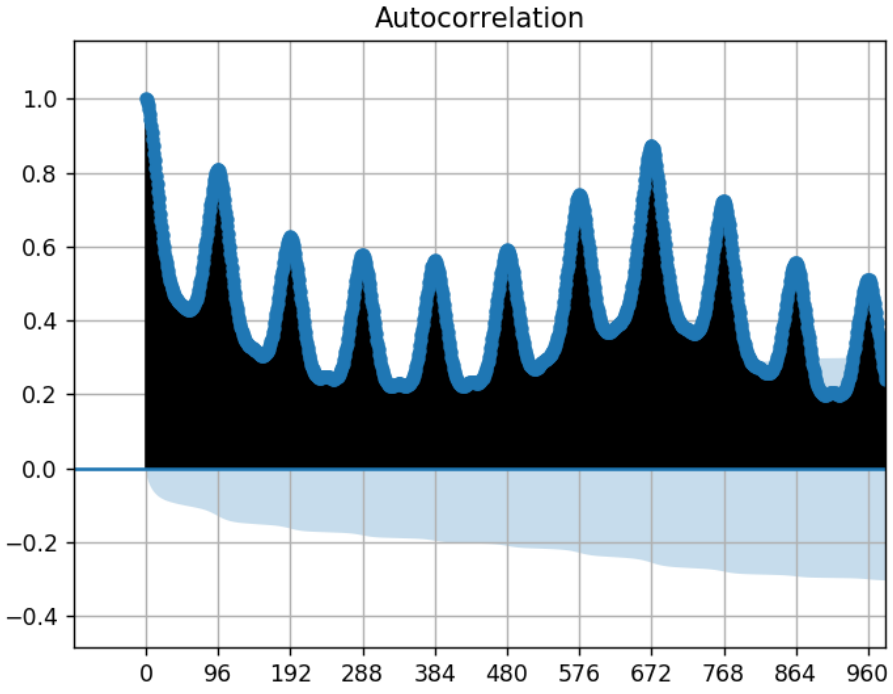

Figure 35 Zoomed Autocorrelations for the first 10 days, indicating a higher correlation with the last day (96 time steps) and the value one week ago (672 time steps), with all lags that are multiples of 96 clearly exceeding the 95% confidence band

5.2. Pre-Processing

The empirical part of this thesis tries to apply as few pre-processing steps as possible, as the experiments shall also show to which extent the machine learning methods are capable of capturing patterns in the data with as less help from a human operator via introducing auxiliary variables or certain pre-processing steps which could introduce a bias in favor for one or the other model.
First-order differencing between consecutive time steps is a common approach for de-trending the time series and making it stationary. This would result in a series of absolute changes in electricity demand on a 15 minutes basis. Since this pre-processing step could not remove non-stationarity and further in some initial experiments neither showed to be helpful for the machine learning methods nor for the conventional methods, this pre-processing step was not applied.
After inspection of the autocorrelations in figure 34 and figure 35, a reasonable pre-processing step could have been to apply seasonal differencing to reduce seasonality in the time series.
Maimon and Rokach state that research has not agreed on the necessity of de-seasonalizing : “For example, in modeling and forecasting seasonal time series, some researchers (Gorr, 1994) believe that data preprocessing is not necessary because the neural network is a universal approximator and is able to capture all of the underlying patterns well. Recent empirical studies (Nelson, Hill, Remus & O’Connor, 1999; Zhang and Qi, 2002), however, find that pre-deseasonalization of the data is critical in improving forecasting performance.” [67]
However, by differencing the series to remove for example the weekly cycle, the forecasting models “would be trained on the residuals of the load at the same time and day in two consecutive weeks and therefore could not learn the similarities in consecutive days at a particular time.”[68]
Thus, by differencing some information granularity may be lost and therefore differencing was not used.
Amongst many others, it has been shown by LeCun et al.[69] that the training of neural networks converges faster, if the mean of the training data is close to zero. Thus, the only pre-processing step that was carried out is data standardization (5.1), which transforms the training data to a distribution with a zero mean and a standard deviation of 1.

$$y = \frac{x - mean}{\sqrt{Var}} \tag{5.1}$$

5.3. Software Stack

In essence, the software stack consists out of a programming environment, a programming language, a deep learning framework, a low-level deep learning backend and different packages that allow the reuse of pre-implemented concepts in the area of machine learning and statistics. While the deep learning backend operates on a low level and defines, for example, different models of an artificial neuron and deals with the mathematics, the deep learning framework provides a higher level of abstraction and allows using pre-implemented architectures which can then be modified and parametrized.
The software stack used for the empirical part of this thesis had to fulfill certain main requirements:

- Wide-spread programming language for statistical and machine learning models
- GPU-computation support
- Reference implementations in the area of deep learning available
- Good community support
- A Development environment that can be run on a Linux-based server in the cloud

Two of the most widespread programming languages for statistical computing and machine learning are *R* and *Python*. While R is a programming language with a strong focus on statistics, Python is more versatile and the evaluation of available deep learning libraries showed that Python clearly has the lead here over R. Since there is a well-documented and extensively tested package (forecast package by Roby Hyndman [70]) for R available that includes the double seasonal Holt-Winters (DSHW) method, R is chosen as the programming language for the purely statistical model and Python is used for the machine learning models. For the evaluation of the Support Vector Machine, the famous Python package scikit-learn has been used. The software stack that is further described in the following lines, is summarized in figure 36.

Different available open-source deep learning frameworks including Keras, Torch (Facebook), Microsoft Cognitive Toolkit (CNTK) and Caffee have been evaluated and the decision fell for Keras, since at the time of the evaluation (July 2017), this framework seemed to have the largest user-base and reference implementations regarding recurrent neural networks. Another important point for this choice was that the underlying deep learning backend could be switched between the Theano and the Tensorflow library and both had their pro's and con's at the time of the evaluation, such as Theano offered better capabilities of speeding computations up on the CPU by massive parallelization of instructions, whereas the GPU- parallelization seemed to be better with Tensorflow. So, by choosing Keras, maximum flexibility in this matter could be achieved and the decision between the deep learning libraries could be postponed. Finally, during the first experiments within the empirical part of this thesis, it became clear that Tensorflow provided superior GPU-support over Theano and a more active community. Furthermore, Yoshua Bengio (the father of Theano and a leading deep learning researcher), announced on Sept. 28, 2017 that Theano won't be further maintained. Nevertheless, using Tensorflow was a good choice and is eagerly maintained by a team at Google and additionally the supporting and supported backends of Keras are growing as now it is possible to use the CNTK by Microsoft as a backend and Deeplearning4j. At the time of writing also some experiments with the CNTK have been conducted, but for the recurrent neural networks and especially for the stateful LSTM networks, Tensorflow is a step ahead.

Figure 36 Overview of the Software Stack used in the empirical part. On the left side, the Python-based stack is shown, whereas, on the right side, the R-based stack is depicted

5.4. Hardware Stack

All experiments were conducted on cloud-machines. Three different machines were used, the first one for the experiments with R, the second and most powerful machine for the optimizations of the machine learning algorithms, and the third one for the final evaluations with python. The different machines were mainly chosen for their cost-profile and the computing power needed for the respective task. The GPU capacity is needed in the case of the neural networks, where parallel data processing in batches can tremendously speed up training, by utilization of the parallel cores of the GPU. Each GPU allows for around 4,8 $*10^{12}$ floating point operations per second (TFLOPS).

Machine Type	Task	#CPUs	CPU Freq	RAM	#GPUs	GPU Memory	Parallel GPU Cores	GPU Type
1	R-Computations	24	2,4 GHz	120 GB	-	-	-	-
2	Optimization of ML-Algos	32	2,7 GHz	244 GB	2	16 GB	4096	NVIDIA Tesla M60
3	Final Evaluation of ML-Algos	16	2,7Ghz	122 GB	1	8 GB	2048	NVIDIA Tesla M60

Table 2 The three machine types used for the different tasks during the experiments

5.5. Experimental Setup and Hyperparameter Optimization

As explained in chapter 4.2 – Model Selection, the data is separated into the train-valid set and the test set with 80% of the data constituting the train-valid set and the remaining 20% build the test data. The partitioning of the previously presented energy demand dataset, obtained from swissgrid is documented in table 3.

	% of total data	Time steps	Start date	End date	Total length
Train-valid	80%	1-13897	01.01.2017 00:15	25.05.2017 19:15	13897
Test	20%	13898-17372	25.05.2017 19:30	01.07.2017 00:00	3475

Table 3 Partitioning of the energy demand dataset for the experiments

5.5.1. One-Step Models

For the neural networks the number of neurons, the number of layers, the learning rate, the optimizer, the batch-size and the number of training epochs were considered as hyper parameters to tune during the Bayesian optimization. The activation function for the hidden units was chosen as the tanh which is a reasonable choice and no big changes in performance can be expected by exchanging it against other common activation functions, like a logistic function, according to the survey paper of Gamboa et al.[32]. Thus, the activation function was not object to the optimization.

The neural network is trained for a number of epochs and in each epoch it tries to improve its ability to forecast the next time-step (target) by considering a number of historical time-steps (source sequence with its length defined by the lookback-window). The training samples (source-target-pairs) for the one-step-ahead prediction are created with a rolling-window like it is depicted in figure 28. To speed up the training, a number of training examples can be shown to the network simultaneously and due to the large time series used in this study, such an approach is needed to allow for training in a feasible time. The number of examples that are shown to the neural network in parallel is defined by the batch-size. This results in a data matrix as it is exemplarily visualized in figure 37 for a lookback-window length of four and a batch-size of four. The sequences A to D are shown in parallel to the neural network and temporal dependencies can only be exploited within a sequence within the batch. To exploit temporal dependencies that extend over the lookback-window, so-called *stateful* recurrent neural networks / LSTMs are employed. That means that at the beginning of every training epoch the cell state of the network at the end of the previous training epoch is used as its initial state.

A difficulty that is often overseen, when using stateful networks, is that the input to the neural network needs a special form, which can also be dependent on the deep learning framework used. The statefulness in Keras is implemented in the way that successive batches are treated as logical successive data junks. That means sequence A in batch 1 is the predecessor of sequence A in batch 2. This implies that the batches need to have all the same batch-size (length) and that the batch-size must equal to the length of the lookback-window. If this structure is filled with the data from a rolling-window on a time-series, as it is shown in figure 37, this results in a Hankel-matrix, which is a square-matrix where the elements of the main skew diagonals are constant. This also has the implication that the training data sometimes has to be cut so that it can be fit into batches of a fixed length.

During the sighting of available reference implementations, it seemed that this pre-processing step is circumvented by training stateful architectures with a batch-size of one. This might be okay for single runs with small time-series, but for repeated runs with different hyperparameter settings and long time-series, this is computationally not feasible.

		Source (lookback-window)				Target	Batches
Training - Sequences	A	1	2	3	4	5	Batch 1
	B	2	3	4	5	6	
	C	3	4	5	6	7	
	D	4	5	6	7	8	
	A	5	6	7	8	9	Batch 2
	B	6	7	8	9	10	
	C	7	8	9	10	11	
	D	8	9	10	11	12	
	A	9	10	11	12	13	Batch 3
	B	10	11	12	13	14	
	C	11	12	13	14	15	
	D	12	13	14	15	16	

Figure 37 Structure of the training data for the one-step neural-network forecasters: The network is trained in a way that it learns to transform the source sequence into the target value. The matrix within each source-batch has the structure of a Hankel-matrix (a square-matrix where elements of the main skew diagonals are equal).

For the LSTM and the RNN a two-stage Bayesian Optimization approach has been chosen, where a wider range of hyperparameter settings get tested on smaller subsets of the training data and more promising parameter ranges are more closely evaluated in a follow-up stage to a first evaluation stage. Related approaches are already described by Swersky [52] and Wang [50].

In the first stage of this experiment, a wide parameter range was evaluated with Bayes hyperparameter optimization, to allow for a more exhaustive parameter search in a more promising range with a narrowed parameter grid and extended training data during the second stage.
For this endeavor, four blocked folds, each with a length of 3474 time-steps and a cutoff at 2500 with 80% for training and 20% for validation were fed to the neural networks. The size of the cutoff was chosen in a way that the parameter combinations, which were likely to result in long training times (such as a small batch-size which results in less parallelization) could be trained within 2500 seconds (roughly 42 minutes). This was done to meet time and financial resource constraints, resulting in a maximum training and evaluation time below 48 hours for the 50 trials. Nevertheless, very early in the experiments, it was obvious that the factor which increases the training time the most, is the number of network layers – the more layers, the longer the training time. Since the literature suggests that mostly 2 layers are enough to estimate all types of functions[4] [32], the first stage was divided into two parallel sub-stages: One investigating the, according to the literature, more promising 1-to 3-layer neural networks and the other evaluating an extreme case with 6 layers (table 4). Therefore, two machines of type 2 were set-up, which each allowed for training the four folds in parallel.
In the following, the search grids for the Bayesian hyperparameter optimization are documented. The values for these grids have been chosen by sighting related literature for common and promising parameter settings (see the related work in chapter 3) and some initial manual experiments with extreme values. This procedure may provide the Bayesian optimization with a starting point that is not too far-off and thus allows for a relatively low number of trials.

First Stage		
Number of layers	1,2,3	6
Number of neurons	10,20,50,100	10,20,50,100
Optimizer	Adam , Rmsprop	Adam , Rmsprop
Learn rate	0.0001, 0.001,0.01,0.1,0.2,0.5	0.0001, 0.001,0.01,0.1,0.2,0.5
Batch-size	64,96,256	64,96,256
Number of epochs	100,200,400	100,200,400
Number of folds	4	4
Fold length	2500	2500
Trials	50	50

Table 4 Parameter grid for the first hyperparameter optimization Stage with two parallel runs, one with a maximum depth of one to three hidden layers, and the other with six hidden layers

The experiments of the first stage clearly showed that one- and two-layer models are superior over three- and six-layer models, which also aligns with the literature (see [32] and [4]). The difference between one- and two-layer models was not so clear, therefore, these two architectures were also assessed in the final stage. Moreover, smaller learning rates clearly showed better results than the larger ones and the maximum number of epochs showed to be sufficient with 100. In the second stage, the models were trained on folds without a cut-off to utilize all available training data, since, in this stage, the search is already conducted in promising parameter ranges. An exhaustive search would take 96 trials with the given parameter grid in table 5. Again, with the Bayesian optimization, only a fraction of trials need to be evaluated to find nearly optimal results. To save resources and since the search is already conducted in a promising grid but with longer folds, the number of trials was reduced to 30.

Final (Second) Stage	
Number of layers	1, 2
Number of neurons	10,20,50,100
Optimizer	Adam, Rmsprop
Learn rate	0.0001,0.001
Batch-size	64,96,256
Number of epochs	100
Number of folds	4
Fold length	3474
Trials	30

Table 5 Parameter grid for the second hyperparameter optimization stage for LSTM and RNN

Finally, for both the LSTM and the RNN, the best performing setup showed to be a two-layer architecture like shown in figure 38 with 100 neurons, a batch-size of 96, the Adam optimizer, and a maximum of 100 training epochs.

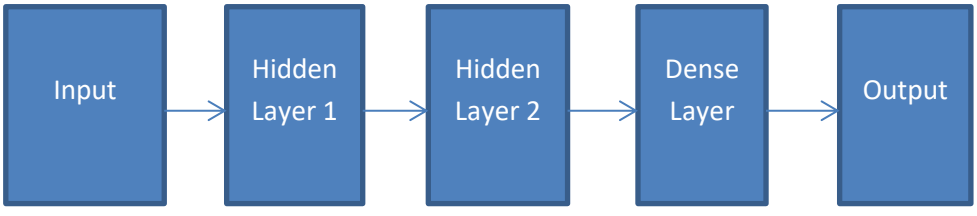

Figure 38 Architecture of the LSTM and RNN for One-Step Predictions

For the SVM, the training samples were generated with a rolling window with step-size one, in the same way as for the neural network models. Also Bayesian hyperparameter tuning has been applied. The kernel function was not object to the optimization as the radial bias function (rbf) chosen, since extensive research has been done on the choice of the kernel function and the rbf seems to be the best choice regarding accuracy and training time (compare [71] and [72]). The rbf introduces the parameter γ which controls the smoothness of the support vector, where a higher value for γ implies a more sophisticated solution.[73] For a detailed explanation of the SVM and the parameter C please refer to section 4.5.2 – Support Vector Regression. In Table 6, the parameter grid for the Bayesian hyperparameter optimization is documented. Again, 4 folds were evaluated to achieve better generalization, since the training time of SVM's with an rbf kernel was much lower than that of neural networks, the folds could use the whole available training data, without hurting time and resource restrictions. Further, initial experiments have shown that the SVM is very sensitive to changes in the magnitude within the training data, thus a retraining of the SVM was executed every 200 time steps, which showed to be a good compromise between runtime and precision.

SVM Hyperparameter Optimization	
Kernel	rbf
C	Uniform(10,100)
γ	Uniform(0.001,0.01)
lookback	48,64,96,128
Number of folds	4
Fold length	3474

Table 6 Grid for the Bayesian hyperparameter optimization with the values for the parameter C and γ sampled from a uniform distribution

The optimization led to a model with a lookback-length of 48 time steps, and values C=66 and γ= 0.0036178.

For the optimization of the DSHW model, the two main seasonal periods were identified with the autocorrelation plot in figure 35 as the daily (96 time steps) and weekly (672 time steps) seasonality. Only the lookback size needed to be tuned since all other model parameters are estimated by the forecast package [70] in R by minimization of the sum of squared 1-step-ahead errors (see section 4.5.3 – Double Seasonal Holt-Winters). The optimal lookback length showed to be around 2000 time steps. This seems reasonable, as DSHW is a purely statistical method and as explained in the exploratory analysis, the training and test data is non-stationary and if the parameters are estimated on a long historical fraction of the time series which has very different statistical properties than the more recent data, the results for the parameter estimation will be poor. The parameters for this model were, due to the computationally uncostly calculations, re-estimated after every time step.

5.5.2. Multi-Step Models

In the multi-step scenario, the next 96 time-steps (24 hours) are predicted in a walk-forward manner, with a step size of one day. Regarding the neural network methods, the MIMO-(Multi-Input-Multi-output) strategy was chosen (see section 4.4.2 - Multistep Forecasting Strategies) because in preliminary experiments the implementations with the recursive strategy showed to be inferior to MIMO-implementations. The first approach was to use the standard neural network architecture also used in the one-step scenario and instead of a single-valued target sequence for each training sample, the networks were trained on a multi-step target window of size 96, by applying a slightly changed architecture with a time-distributed dense layer as depicted in figure 39. The downside of this sequence-to-sequence prediction approach, with the target sequence being longer than one, is that the source sequence length must equal the target sequence length. This results in a lookback with the length of 96 time steps. A Bayesian hyperparameter optimization had been performed on the number of neurons, the number of layers, the learning rate, the optimizer, the batch-size and the number of training epochs. The out-of-sample prediction results of the optimized models were not very promising since they did not capture the characteristics of the time series and were more or less copying the last sequence, even though stateful neural networks were used. From these results, it was clear, that the networks needed longer input sequences for the training samples than only the last day to predict the next day. With the above-described restrictions for the source and target length, another architectural approach had to be used. Inspired by the research in the field of machine translation, the so-called encoder-decoder architecture, shown in figure 40 encoder-decoder architecture, was used. This architecture was proposed by Cho et al. and "consists of two recurrent neural networks (RNN) that act as an encoder and a decoder pair. The encoder maps a variable-length source sequence to a fixed-length vector, and the decoder maps the vector representation back to a variable-length target sequence. The two networks are trained jointly to maximize the conditional probability of the target sequence given a source sequence."[74] Instead of a simple RNN, also LSTM's can be used for the encoder-decoder model.

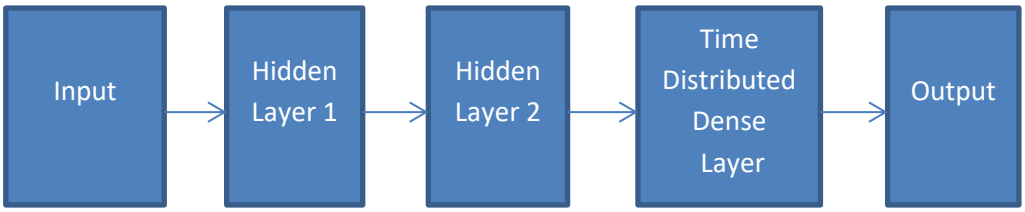

Figure 39 Architecture for LSTM and RNN for multi-step predictions

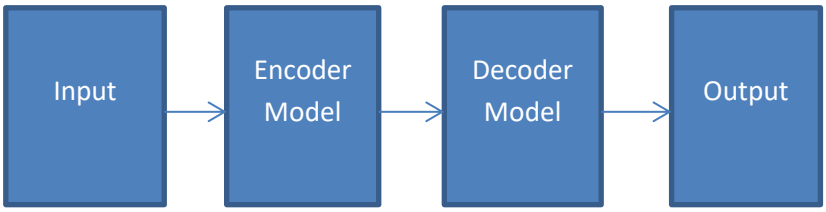

Figure 40 Encoder-Decoder Architecture

Since using stateful neural networks did not show good performance in exhaustive initial experiments and due to the limitations of the input format along with the batch-size and increased training time

that come with stateful neural networks, stateless neural networks were used in the multistep scenario. This allowed to use a batch size of 1024 and significantly reduced the training time. As already explained, the length of the target sequence equaled to one day (96 time steps) and the training data was generated by a rolling window with a step-size of one.

The initial experiments with the encoder-decoder model showed promising results, but at the same time they indicated that a similar approach like with the standard architecture together with an extensive hyperparameter search was not feasible with the resources available for this thesis: While in the one-step scenario and the multistep scenario with the standard architecture, the best results could already be yielded within the first 100 training epochs, in the case of multistep predictions, many hundred training epochs were necessary to evaluate a single set of parameters. Moreover, the models were trained on the unfolded training data because the models showed to generalize badly when trained on smaller subsets, which also increased the time necessary to assess a parameter setup. 15 experiments with manually chosen parameter sets have been conducted. As well as for the LSTM as the RNN, the following set of parameters yielded the best results during the manual hyperparameter tuning: 20 neurons in the encoder and in the decoder with a single hidden layer in both, a learning rate of 0.001 and a maximum of 500 training epochs. Alternative trials have been performed with differing values for the number of neurons in the hidden layers (10,30,50,100), the learning rate (0.0001,0.01) and the lookback period (96,192,288,672). An increase in the number of hidden layers was tested, which didn't result in an increased performance but longer training times.

In opposition to the neural network models, a single SVM does not have the ability to do MIMO-forecasts, therefore the recursive strategy was used, where the output of a one-step-ahead forecast is fed back as input, constituting the last element of the history window, for the next forecast. Due to the SVM's sensitivity to changing amplitudes in the data, like in the one-step case also in these experiments, the SVM was retrained frequently (after every forecast / 96 time steps). A major advantage of the SVMs is the short training time which allowed for an extensive Bayesian hyperparameter search, even in the multi-step scenario. For the optimization, the same search grid as in Table 6 has been used. The best results were obtained with a lookback-length of 384 time steps (4 days), C=91 and γ=0.01339945. Compared to the best solution for the one-step solution, it shows that for a multi-step forecast, for the SVM a longer lookback period is of advantage (which also coincides with the multi-step forecasts with neural networks) and the higher value of C and γ indicate a more complex solution, which is also reasonable since also the target time series are longer and more complex.

Also for the DSHW-method the iterative multi-step forecasting approach was implemented. The tuning for the DSHW did show similar as for the one-step scenario and the optimal lookback length lay at around 2000 time steps. The model's parameters were re-estimated after every forecast (one day/ 96 time steps).

5.6. Analysis of Forecasting Results

5.6.1. One-Step Forecasts

In table 7, the statistical error measures for the experiments are summarized. Due to the stochastic nature of the neural networks, the average of the 15 evaluations in table 8 is shown. The results suggest that the LSTM has the overall best performance:
The LSTM has a 10.33% lower RMSE, and shows an improvement by 4.75% in terms of MAPE and 2,67% in terms of MDA compared to the second-best forecaster in this experiment, namely the DSHW. The DSHW benchmark was identified as the best performing univariate method for electricity load forecasting by Talyor et al.[41].
As demonstrated further below, the mean RMSE of the LSTM was tested as significantly smaller than the RMSE of the DSHW, with the RMSE of the latter as the comparison value in a one-tailed, one-sample t-test. Regarding the other neural network forecaster (RNN), the LSTM clearly shows a lower standard deviation (s) and a lower mean RMSE. Further below, the procedure for the significance tests is documented exemplarily for the RMSE-values.

Method	**RMSE (s)** [kwh]	**MAPE (s)** [%]	**MDA (s)** [%]
LSTM	12884.39 (750.0029)	0.7233 (0.0389)	0.7541(0.0092)
DSHW	14368.44	0.7708	0.7274
RNN	14571.95 (4307.544)	0.827 (0.2305)	0.7317 (0.0226)
SVM	15509.658	0.904	0.698
Naive	24181.05	1.3299	0.6489

Table 7 Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Directional Accuracy (MDA) of one-step-ahead forecasts for the LSTM and the four benchmark methods. The values in brackets are the empirical standard deviations (s) of the respective error measure.

	LSTM			**RNN**		
Run	**RMSE** [kwh]	**MAPE** [%]	**MDA** [%]	**RMSE** [kwh]	**MAPE** [%]	**MDA** [%]
1	12191.641	0.697	76.0	13222.799	0.76	74.3
2	13139.907	0.745	74.7	13853.402	0.788	74.4
3	12540.587	0.706	76.3	13228.914	0.756	75.2
4	11718.419	0.662	76.0	13833.385	0.783	74.2
5	12261.432	0.689	76.1	13775.014	0.784	72.7
6	13362.383	0.744	74.2	13491.6	0.768	73.6
7	13866.4	0.74	74.0	13363.259	0.761	72.1
8	12058.708	0.685	75.9	30105.982	1.658	65.6
9	12668.46	0.702	75.1	13527.378	0.774	73.4
10	14184.875	0.789	76.4	12973.98	0.741	74.2
11	12893.651	0.728	74.5	13217.39	0.751	74.0
12	13098.455	0.745	75.2	13547.036	0.781	73.0
13	14030.112	0.8	74.9	13293.545	0.753	74.5
14	13049.083	0.731	77.1	13992.908	0.797	72.5
15	12201.796	0.686	74.7	13152.642	0.75	73.9

Table 8 RMSE, MAPE,MDA for each of the 15-times repeated RNN and LSTM one-step forecasting experiments

Since one of the assumptions of the t-test is a normal distribution of the test sample, the Shapirow-Wilkinson-test is used to assess, whether the null-hypothesis that the sample is drawn from a normal distribution has to be rejected. As shown in table 9, the p-value indicates that the null hypothesis cannot be rejected at a 1%-significance level. With the restrictions that come with the small sample size and the power of the Shapirow-Wilkinson test, the assumption of normality for the t-test is to the best knowledge, not hurt, and the t-test can be performed. The hypotheses for this test are formulated relative to the RMSE of the second-best performing method (DSHW):
H_0: The results come from a normal distribution with the mean of 14368,44
H_1: The results come from a normal distribution with a mean significantly lower than 14368,44
As documented in table 9, the null-hypothesis can be rejected on a 1% significance level, indicating, that the results of the LSTM are significantly better than the non-neural forecasters.

Shapirow-Wilkinson		One-tailed, one-sample t-test	
W	p-value	t	p-value
0.95492	0.605	-7.6613	1.129e-06

Table 9 Test statistics for Shapirow-Wilkinson and t-test, with respect to LSTM and DSHW in the one-step scenario

For the mean-comparison of the LSTM and the RNN, a variant of the t-test for unequal variances has to be performed, but checking the results of the 15 runs of the RNN with the Shapirow-Wilkinson-test led to a rejection of the null-hypothesis (W=0.34575 and p=2.524e-07). Thus, a normal distribution of the results of the RNN-runs cannot be assumed and the assumptions of the Welch-test are not met. However, if one removes the outlier (30105.982), with an RMSE double as big as the other 14 results, the standard deviation of the sample reduces to s=307.1852 and the mean equals to 13462.38 kwh. With the outlier-cleaned dataset, the null-hypothesis of the Shapiro-Wilkinson test cannot be rejected and a Welch-test may be performed (see Table 10):
H_0: True difference in means is equal to 0
H_1: True difference in means is not equal to 0

Shapirow-Wilkinson		Welch-test	
W	p-value	t	p-value
0.94747	0.5221	-2.7479	0.01286

Table 10 Shapirow-Wilkinson-test for outlier-remove RNN-runs and Welch-test for comparison of the means of the LSTM-runs and the outlier-removed RNN-runs

The result of the Welch-test portends a significant lower mean RMSE of the 15 LSTM-runs compared to the 14, outlier-removed RNN-runs on a 1% significance level.

Figure 41 visualizes a comparison between the training times needed to produce the forecasting results. More specific, the visualization incorporates not only the training times but also the times needed to re-estimate the model parameters after the respective number of time steps for the DSHW and the SVM. A single parameter re-estimation for the DSHW is timely rather inexpensive with only 16 seconds and for the SVM it amounts to as little as three seconds.

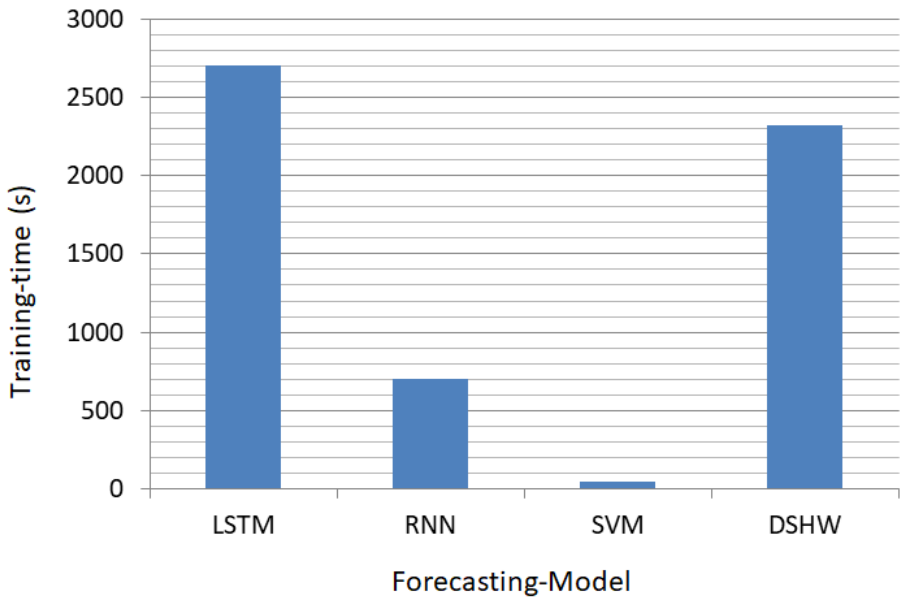

Figure 41 Time in seconds needed for the training-/parameter re-estimation for the different forecasting models in the one-step-ahead out-of-sample forecasting scenario (LSTM: 2700s , RNN: 700s , SVM: 51s, DSHW: 2320s)

In figure 42 and figure 43 a magnification of the out-of-sample forecasts for the DSHW,SVM,LSTM and RNN is depicted. The first magnifies the forecasts for the 200 time steps between 5th of June 5:15 and 7th of June 7:15. The second one shows the 200 time steps between 13th of June 13:15 and 15th of June 15:15. The visualizations do not differ too much from each other, which can be expected since the performance metrics are not too far off from each other.

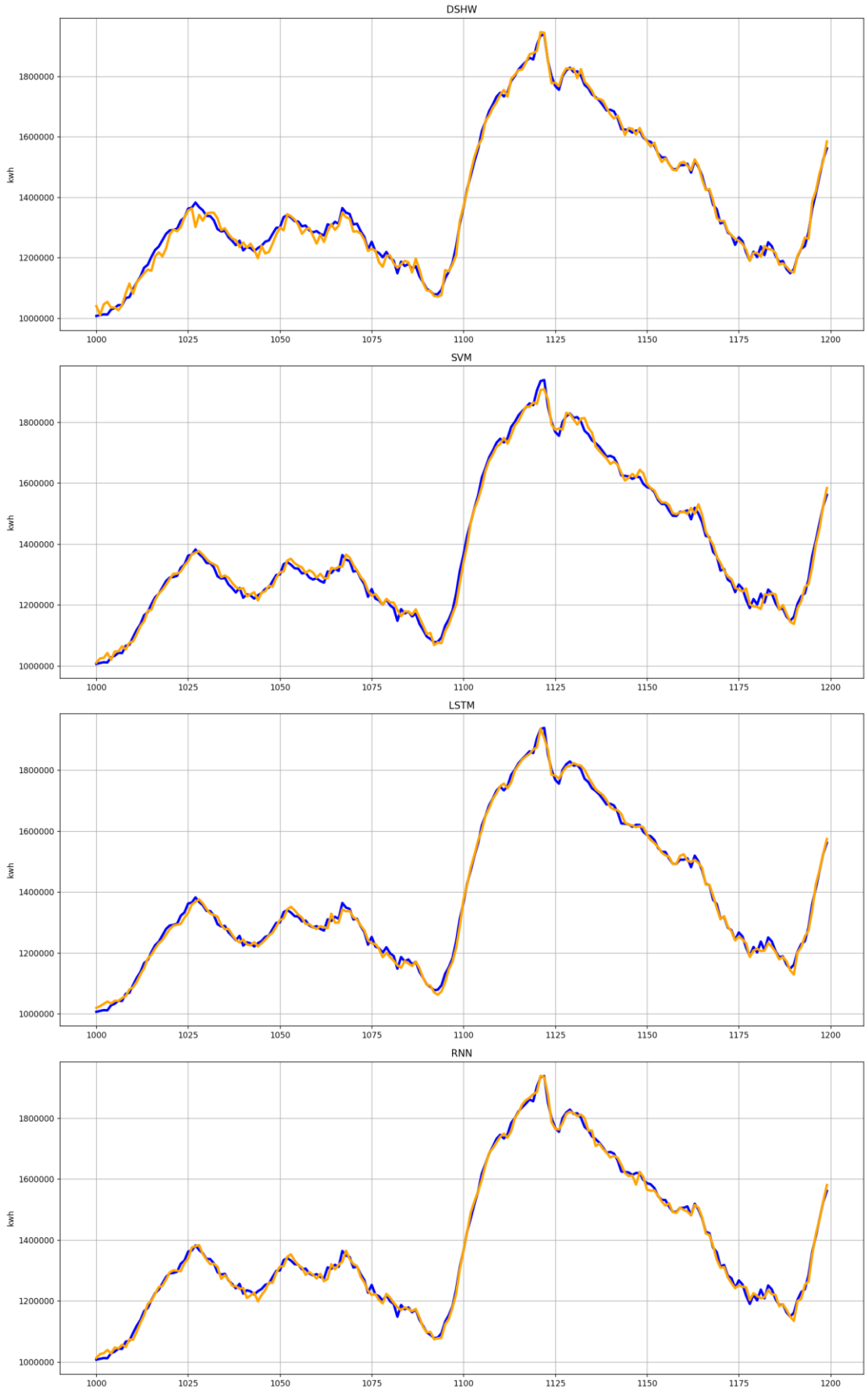

Figure 42 Out-of-sample one-step forecasts from 5.6.2017 5:15 – 7.6.2017 7:15 (time steps 1000-1200 of the test data)

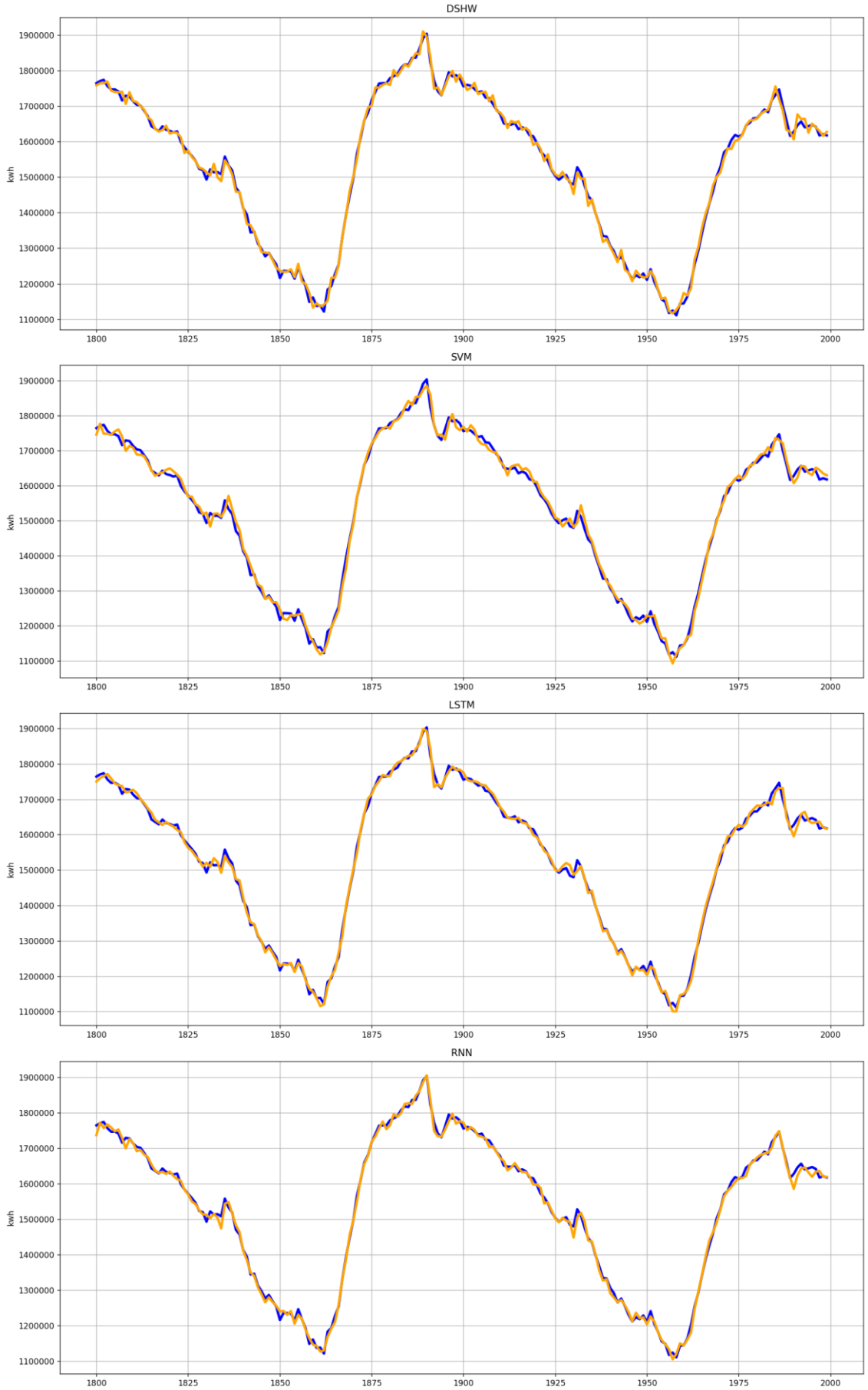

Figure 43 Out-of-sample one-step forecasts from 13.6.2017 13:15 – 15.6.2017 15:15 (time steps 1000-1200 of the test data)

5.6.2. Multi-Step Forecasts

In table 11, the performance measures for the different multi-step forecasters are reported. Like in the one-step case, also in the multi-step scenario the neural network forecasts were repeated 15 times (see table 12) to average over the stochastic results. The LSTM could not outperform the DSHW-method. In terms of RMSE, the LSTM was the best performing machine learning method, but interstingly, the SVM showed significantly better accuracy regarding the directional forecasts (MDA). Further below, the procedure for the significance tests is documented exemplarily for the RMSE-values.

Method	**RMSE (s)** [kwh]	**MAPE (s)** [%]	**MDA (s)** [%]
DSHW	123726.2	5.273001	79.70
LSTM	134388.8 (10726.71)	7.516 (0.5055)	67.4867 (0. 86)
SVM	150462.096	7.654	79.1
RNN	272231.5 (8531.788)	17.94047 (0.6009)	17.9 (2.92)

Table 11 Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Mean Directional Accuracy (MDA) of 96-step-ahead forecasts for the LSTM and the benchmark methods. The values in brackets are the empirical standard deviations (s) of the respective error measure.

	LSTM			**RNN**		
Run	**RMSE [kwh]**	**MAPE [%]**	**MDA [%]**	**RMSE [kwh]**	**MAPE [%]**	**MDA [%]**
1	154058.855	8.294	67.9	262828.887	17.386	18.2
2	129112.164	7.335	66.8	280325.231	18.543	15.0
3	127213.302	7.258	68.4	274881.773	18.037	15.9
4	122214.392	7.008	66.8	267066.932	17.678	18.2
5	157750.333	8.649	66.7	273035.95	17.681	17.1
6	131003.143	7.144	68.3	272335.311	18.142	24.4
7	136391.03	7.611	67.4	272876.742	17.978	21.7
8	129172.855	7.319	68.2	273383.121	18.198	17.4
9	125464.875	7.121	68.1	251658.433	16.431	22.1
10	141093.501	7.967	66.6	262085.504	17.157	18.1
11	142034.065	8.053	65.7	278340.603	18.376	18.6
12	127621.628	7.141	67.3	278154.739	18.409	17.7
13	135925.64	7.501	68.8	271231.934	17.932	15.1
14	121150.567	6.928	67.2	280078.207	18.502	14.3
15	135624.92	7.411	68.1	285189.0	18.657	14.7

Table 12 RMSE, MAPE,MDA for each of the 15-times repeated RNN and LSTM multi-step forecasting experiments

Again, before comparing the mean values of the LSTM and the DSHW, the normality assumption of the test sample for the t-test has to be assessed with the Shapirow-Wilkinson-test. As shown in table 13, the p-value indicates that the null hypothesis cannot be rejected at a 1%-significance level. Thus, with an unrefuted normality assumption, the t-test can be performed with the following null (H0) - and alternative (H1) hypothesis:

H0: The results come from a normal distribution with the mean of 123726.2

H1: The results come from a normal distribution with a mean significantly greater than 123726.2

As documented in table 13, the null-hypothesis has to be rejected, from which follows that the DSHW-method is performing significantly better than the LSTM.

Shapirow-Wilkinson		One-tailed, one-sample t-test	
W	p-value	t	p-value
0.90529	0.1147	3.8498	0.0008841

Table 13 Test statistics for Shapirow-Wilkinson and t-test, with respect to LSTM and DSHW in the multi-step scenario

Table 14 summarizes the test statistics for the Shapirow-Wilkinson-test for the 15 RNN-runs and the Welch-test for the mean comparison between the LSTM and the RNN. The Shapirow-Wilkinson-test did not deliver a contraindication to the normality assumption at a 1% significance level. Therefore the Welch-test could be performed with the following null- and alternative hypothesis:
H_0: True difference in means is equal to 0
H_1: True difference in means is not equal to 0
As expected, the Welch-Test showed that there is a significant difference of the mean-values of the LSTM and RNN runs and thus that the LSTM is performing significantly better than the RNN.

Shapirow-Wilkinson		Welch-Test	
W	p-value	t	p-value
0.93693	0.3453	-38.951	2.2e-16

Table 14 Shapirow-Wilkinson-test for RNN-runs and Welch-test for comparison of the RMSE mean of the LSTM-runs and the outlier-removed RNN-runs

Lastly, also a Welch-test was performed to compare the mean RMSE of the LSTM runs with the RMSE of the SVM and the LSTM with the hypothesis H_o and H_1:
H0: The results come from a normal distribution with the mean of 150462.096
H1: The results come from a normal distribution with a mean significantly less than 150462.096
In table 15, the small p-value is reported which led to a rejection of the null-hypothesis at a 1% significance level. Thus, the mean RMSE of the LSTM can be assumed as significantly smaller than the RMSE produced with the SVM forecasts.

Welch-Test	
t	p-value
-5.8034	2.288e-05

Table 15 Welch test for mean comparison of the RMSE values of LSTM and SVM

The total training- and re-estimation times needed for the out-of-sample forecasts are depicted in figure 44. A single re-estimation took about 14 seconds for the SVM and 16 seconds for the DSHW. The overall required training time increased for the LSTM by more than 76% compared to the single-step case, for the RNN the increase amounted to 114%. This increase in training time of the neural networks is due to the more sophisticated architecture (encoder-decoder-layers) and the larger lookback-lengths. In case of the SVM, the nearly 10-fold training time resulted out of more frequent re-estimations (after every day) and increased computational complexity due to a longer lookback period.

Only the statistical method, namely the DSHW, even decreased the time necessary for producing the forecasts in the multi-step scenario as against the single forecasts. This is because the re-estimations which are the time-intensive task with the DSHW-method, had to be done less frequently (daily instead on a 15 minutes basis).

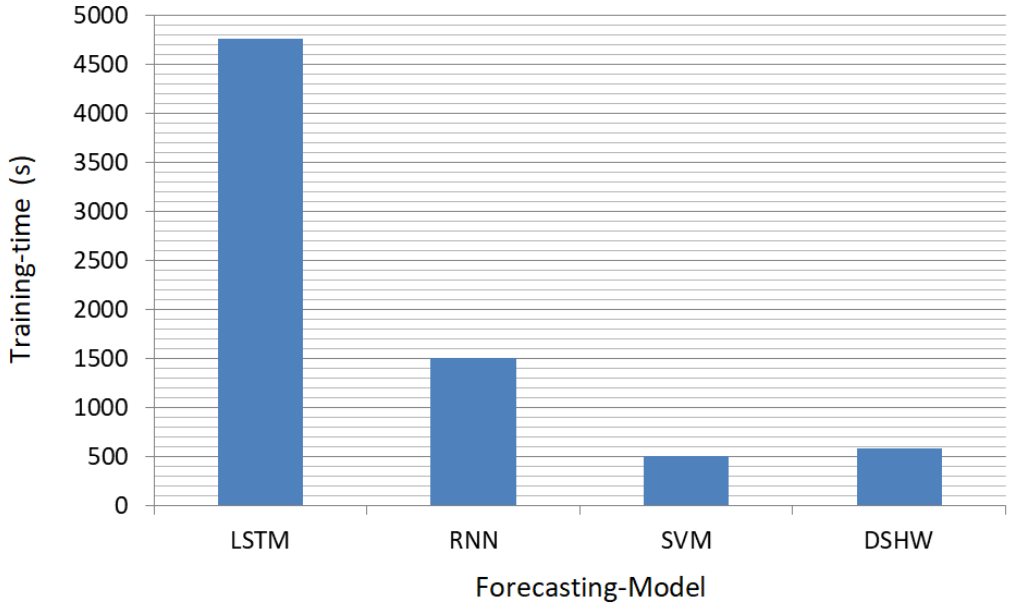

Figure 44 Time in seconds needed for the training-/parameter re-estimation for the different forecasting models in the multi-step-ahead out-of-sample forecasting scenario (LSTM: 4760s , RNN: 1500s , SVM: 507s , DSHW: 579s)

In figure 45, the 96-step-ahead rolling-window forecasts are visualized for all four forecasting models. From a qualitative point of view, the statistical results from above can be confirmed as the DSHW-, SVM- and LSTM- forecasters seem to capture the characteristics of the original series to some extent. The only method that obviously fails to deliver useful predictions is the RNN-forecaster. This is interesting as the RNN and the LSTM performed very similar in the hyperparameter tuning stage and delivered comparable RMSE scores when evaluated on the validation part of the train-valid data. By expanding the training set with the data used for validation during the hyperparameter optimization and evaluating the model on the withhold test data, the previously observed satisfying results vanished. A possible explanation for this behavior could be the regime shifts inherent in the energy demand dataset (downtrend shifting to a short upwards/sideward trend), another explanation could be that the extended training set needs a much higher number of epochs (the performance with an additional maximum epoch add-on of 400 epochs was tested but did not improve the forecast). Further the visualization show that the SVM has less predictive power when it comes to demand changes from workdays to weekends and vice-versa. This is probably due to the chosen lookback period, which is below one week and for being able to predict these changes, at least a week of historical data for the lookback-window would be necessary. However, the chosen setup showed to deliver the highest accuracy regarding the chosen performance measures.

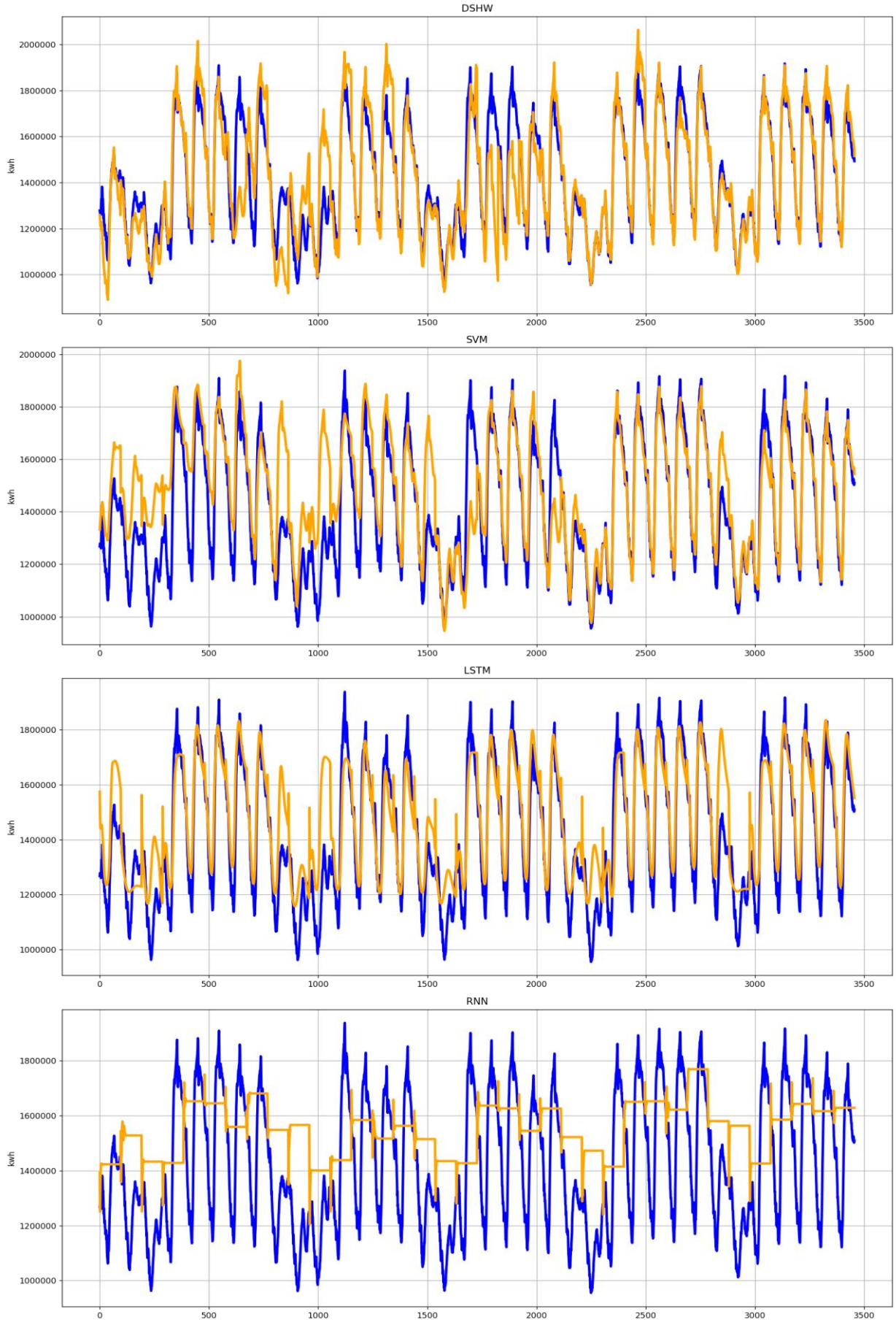

Figure 45 the 96-step-ahead rolling-window forecasts with DSHW,SVM, LSTM and RNN for the whole available test data

6. Conclusion

The applicability of Long Short-Term Memory (LSTM) neural networks for one- and multi-step time series forecasts was evaluated on an exemplary real world problem.
To motivate the use of LSTM's for time series forecasting, the historical development from the first artificial neuron until recent advances in deep learning have been outlined and a disambiguation and classification of related terms like artificial intelligence, machine learning, computational neuroscience and others has been provided.
Time series forecasting with neural networks is, in many aspects, treated as an art - especially when it comes to the approach for conducting experiments. Therefore, a seven-step process model for time series forecasting experiments has been evolved and a clear methodological approach that allows for efficient hyperparameter optimization of machine learning forecasters through making use of a modified form of blocked cross validation and Bayesian hyperparameter optimization has been developed. Further, the performance of the LSTM was compared with machine learning and conventional benchmark models with carefully selected performance measures. An electricity demand dataset was used as the real world problem, since energy demand time series are popular in the forecasting literature for their interesting characteristics and public availableness.

To sum up, the LSTM showed to be a competitive method for time series forecasts: It captured the characteristics of the electricity demand time series very well and proofed to be significantly more effective than simple recurrent neural networks. In the case of the one-step-ahead forecasting scenario, the LSTM even outperformed the Double Seasonal Holt-Winters (DSHW) method which is frequently reported as top performer in comparable scenarios. Though, in the multi-step forecasting scenario the statistical method (DSHW) showed to be superior over all other approaches. However, the LSTM showed the best accuracy amongst the machine learning models in terms of most performance metrics. By comparing the LSTM-forecaster to the predictive performance of simple recurrent neural networks, the added value of the more complex, gated memory block architecture of the LSTM is indicated. A downside of LSTM neural networks is that they take the longest time for training which can be a problem for exhaustive hyperparameter searches with limited hardware resources. On the other side, unlike other forecasting approaches, LSTM nets need comparably infrequent retraining so that once the optimal hyperparameters are found, the long training times are not a problem.

Limitations of this study

The results obtained from the experiments in this thesis have only limited generalized to other, maybe even multivariate datasets, since solely univariate data has been the subject to the evaluation and different dataset can have totally different characteristics which could be learnt with more or less success than the ones inherent in the used electricity demand data.
Another point is that due to the limited hardware, time and financial resources available for this thesis, a proof-of-concept of the developed approach for model search and hyperparameter optimization could be delivered by applying it to the one-step case but the resources were not sufficient to conduct the exhaustive parameter search in the multi-step scenario. Thus, it is likely that even better results for the neural network models could have been obtained.

Outlook

The limitations of this study already give a linking point for further research such as an analysis in multivariate scenarios and in different problem domains. Then, increased resources would be necessary to allow for more profound assertions about the performance of deep learning methods for multi-step-ahead time series forecasts. Further, it would be interesting to evaluate a wider range of modified architectures and algorithms like LSTM's with an attention mechanism as described in Cinar et al. [39] or with peephole connections as introduced by Gers et al.[75].

7. Bibliography

[1] D. Mo, “A survey on deep learning: one small step toward AI,” *Dept Comput. Sci. Univ N. M. USA*, 2012.

[2] M. Längkvist, L. Karlsson, and A. Loutfi, “A review of unsupervised feature learning and deep learning for time-series modeling,” *Pattern Recognit. Lett.*, vol. 42, pp. 11–24, Jun. 2014.

[3] M. Längkvist, “Modeling time series with deep networks,” Örebro University, 2014.

[4] H. S. Hippert, C. E. Pedreira, and R. C. Souza, “Neural networks for short-term load forecasting: A review and evaluation,” *IEEE Trans. Power Syst.*, vol. 16, no. 1, pp. 44–55, 2001.

[5] M. Flasiński, *Introduction to artificial intelligence*. Springer, 2016.

[6] H. Adeli and S.-L. Hung, *Machine learning: neural networks, genetic algorithms, and fuzzy systems*. John Wiley & Sons, Inc., 1994.

[7] L. Medsker, E. Turban, and R. R. Trippi, “Neural network fundamentals for financial analysts,” *J. Invest.*, vol. 2, no. 1, pp. 59–68, 1993.

[8] J. Brownlee, “Supervised and Unsupervised Machine Learning Algorithms,” *Machine Learning Mastery*, 16-Mar-2016. .

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. 2016.

[10] N. Wiener, “Cybernetics, or Communication and Control in the Animal and the Machine,” *N. Y. Ffiley*, vol. 23, 1948.

[11] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[12] D. Hebb, “The organisation of behavior Wiley,” *N. Y.*, 1949.

[13] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[14] M. Minsky and S. Papert, *Perceptron (expanded edition)*. MIT Press, Cambridge, MA. Original edition published in, 1969.

[15] J. L. McClelland and D. E. Rumelhart, *Explorations in parallel distributed processing: A handbook of models, programs, and exercises*. MIT press, 1989.

[16] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Englewood Cliffs, N.J: Prentice Hall, 1995.

[17] J. L. Elman, *Rethinking innateness: A connectionist perspective on development*, vol. 10. MIT press, 1998.

[18] J. L. McClelland, D. E. Rumelhart, P. R. Group, and others, *Parallel distributed processing*, vol. 2. MIT press Cambridge, MA:, 1987.

[19] 2015 at 11:30am Posted by ajit jaokar on July 29 and V. Blog, “Evolution of Deep learning models.” [Online]. Available: http://www.datasciencecentral.com/profiles/blogs/evolution-of-deep-learning-models. [Accessed: 07-May-2017].

[20] H. Wachtel, “McCullogh-Pitts and Perceptron Models,” *University of Colorado*, 2017. [Online]. Available: http://ecee.colorado.edu/~ecen4831/lectures/NNet2.html. [Accessed: 16-May-2017].

[21] A. Ngom, “McCullogh-Pitts Uwindsor,” *University of Windsor*, 2017. [Online]. Available: http://angom.myweb.cs.uwindsor.ca/teaching/cs574/le2.pdf. [Accessed: 16-May-2017].

[22] G. Hinton, “Neural Networks for Machine Learning,” *Coursera*, 2017. [Online]. Available: https://www.coursera.org/learn/neural-networks. [Accessed: 25-May-2017].

[23] A. Ş. Şahin and H. Yazıcı, “Thermodynamic evaluation of the Afyon geothermal district heating system by using neural network and neuro-fuzzy,” *J. Volcanol. Geotherm. Res.*, vol. 233, pp. 65–71, 2012.

[24] M. Bianchini, M. Maggini, and L. C. Jain, *Handbook on neural information processing*. Springer, 2013.

[25] J. Patterson and A. Gibson, *Deep Learning*. O' Riley, 2017.
[26] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]," *IEEE Comput. Intell. Mag.*, vol. 5, no. 4, pp. 13–18, Nov. 2010.
[27] G. E. Hinton, "To recognize shapes, first learn to generate images," *Prog. Brain Res.*, vol. 165, pp. 535–547, 2007.
[28] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, May 2007.
[29] A. Fischer and C. Igel, "An introduction to restricted Boltzmann machines," *Prog. Pattern Recognit. Image Anal. Comput. Vis. Appl.*, pp. 14–36, 2012.
[30] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 759–766.
[31] Y. LeCun, Y. Bengio, and others, "Convolutional networks for images, speech, and time series," *Handb. Brain Theory Neural Netw.*, vol. 3361, no. 10, p. 1995, 1995.
[32] J. C. B. Gamboa, "Deep Learning for Time-Series Analysis," *ArXiv Prepr. ArXiv170101887*, 2017.
[33] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, vol. 385. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[35] "Understanding LSTM Networks -- colah's blog." [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 20-Jul-2017].
[36] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying LSTM to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*, Springer, 2002, pp. 193–200.
[37] X. Qiu, L. Zhang, Y. Ren, P. N. Suganthan, and G. Amaratunga, "Ensemble deep learning for regression and time series forecasting," in *Computational Intelligence in Ensemble Learning (CIEL), 2014 IEEE Symposium on*, 2014, pp. 1–6.
[38] E. Busseti, I. Osband, and S. Wong, "Deep learning for time series modeling," *Tech. Rep. Stanf. Univ.*, 2012.
[39] Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Aït-Bachir, and V. Strijov, "Position-based content attention for time series forecasting with sequence-to-sequence rnns," in *International Conference on Neural Information Processing*, 2017, pp. 533–544.
[40] H. Hippert, D. Bunn, and R. Souza, "Large neural networks for electricity load forecasting: Are they overfitted?," *Int. J. Forecast.*, vol. 21, no. 3, pp. 425–434, 2005.
[41] J. W. Taylor, L. M. de Menezes, and P. E. McSharry, "A comparison of univariate methods for forecasting electricity demand up to a day ahead," *Int. J. Forecast.*, vol. 22, no. 1, pp. 1–16, Jan. 2006.
[42] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *ArXiv Prepr. ArXiv150600019*, 2015.
[43] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv Prepr. ArXiv13080850*, 2013.
[44] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2014.
[45] G. Shmueli and K. C. Lichtendahl, *Practical Time Series Forecasting with R: A Hands-On Guide*. Axelrod Schnall Publishers, 2016.
[46] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.
[47] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Inf. Sci.*, vol. 191, pp. 192–213, May 2012.
[48] S. Arlot, A. Celisse, and others, "A survey of cross-validation procedures for model selection," *Stat. Surv.*, vol. 4, pp. 40–79, 2010.
[49] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.

[50] L. Wang, M. Feng, B. Zhou, B. Xiang, and S. Mahadevan, “Efficient hyper-parameter optimization for NLP applications,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 2112–2117.
[51] E. Brochu, V. M. Cora, and N. De Freitas, “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning,” *ArXiv Prepr. ArXiv10122599*, 2010.
[52] K. Swersky, J. Snoek, and R. P. Adams, “Multi-task bayesian optimization,” in *Advances in neural information processing systems*, 2013, pp. 2004–2012.
[53] M. Adya and F. Collopy, “How effective are neural networks at forecasting and prediction? A review and evaluation,” *J Forecast.*, vol. 17, pp. 481–495, 1998.
[54] “Principles of Forecasting: A Handbook for Researchers and Practitioners - 2001-armstrong-principlesforecasting.pdf.” [Online]. Available: https://www.gwern.net/docs/predictions/2001-armstrong-principlesforecasting.pdf. [Accessed: 07-Oct-2017].
[55] L. J. Tashman, “Out-of-sample tests of forecasting accuracy: an analysis and review,” *Int. J. Forecast.*, vol. 16, no. 4, pp. 437–450, 2000.
[56] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, “Machine Learning Strategies for Time Series Forecasting,” in *Business Intelligence*, vol. 138, M.-A. Aufaure and E. Zimányi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 62–77.
[57] S. Ben Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, “A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition,” *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7067–7083, Jun. 2012.
[58] C. Tofallis, “A better measure of relative prediction accuracy for model selection and model estimation,” *J. Oper. Res. Soc.*, vol. 66, no. 8, pp. 1352–1362, Aug. 2015.
[59] A. J. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Stat. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.
[60] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[61] P. R. Winters, “Forecasting sales by exponentially weighted moving averages,” *Manag. Sci.*, vol. 6, no. 3, pp. 324–342, 1960.
[62] J. W. Taylor, “Short-term electricity demand forecasting using double seasonal exponential smoothing,” *J. Oper. Res. Soc.*, vol. 54, no. 8, pp. 799–805, 2003.
[63] K. Metaxiotis, A. Kagiannas, D. Askounis, and J. Psarras, “Artificial intelligence in short term electric load forecasting: a state-of-the-art survey for the researcher,” *Energy Convers. Manag.*, vol. 44, no. 9, pp. 1525–1534, 2003.
[64] D. Bunn and E. D. Farmer, “Comparative models for electrical load forecasting,” 1985.
[65] P.-F. Pai and W.-C. Hong, “Support vector machines with simulated annealing algorithms in electricity load forecasting,” *Energy Convers. Manag.*, vol. 46, no. 17, pp. 2669–2688, Oct. 2005.
[66] “Swissgrid - Energy Statistic Switzerland.” [Online]. Available: https://www.swissgrid.ch/swissgrid/en/home/experts/topics/energy_data_ch.html. [Accessed: 26-Feb-2018].
[67] O. Maimon and L. Rokach, Eds., *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2010.
[68] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, “An overview and comparative analysis of recurrent neural networks for short term load forecasting,” *ArXiv Prepr. ArXiv170504378*, 2017.
[69] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*, Springer, 1998, pp. 9–50.
[70] R. J. Hyndman, M. O’Hara-Wild, C. Bergmeir, S. Razbash, E. Wang, and M. R. Hyndman, “Package ‘forecast,’” 2017.

[71] W.-C. Wang, K.-W. Chau, C.-T. Cheng, and L. Qiu, “A comparison of performance of several artificial intelligence methods for forecasting monthly discharge time series,” *J. Hydrol.*, vol. 374, no. 3–4, pp. 294–306, Aug. 2009.
[72] C.-W. Hsu, C.-C. Chang, C.-J. Lin, and others, “A practical guide to support vector classification,” 2003.
[73] C. J. Burges, “A tutorial on support vector machines for pattern recognition,” *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.
[74] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” 2014, pp. 1724–1734.
[75] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with LSTM recurrent networks,” *J. Mach. Learn. Res.*, vol. 3, no. Aug, pp. 115–143, 2002.