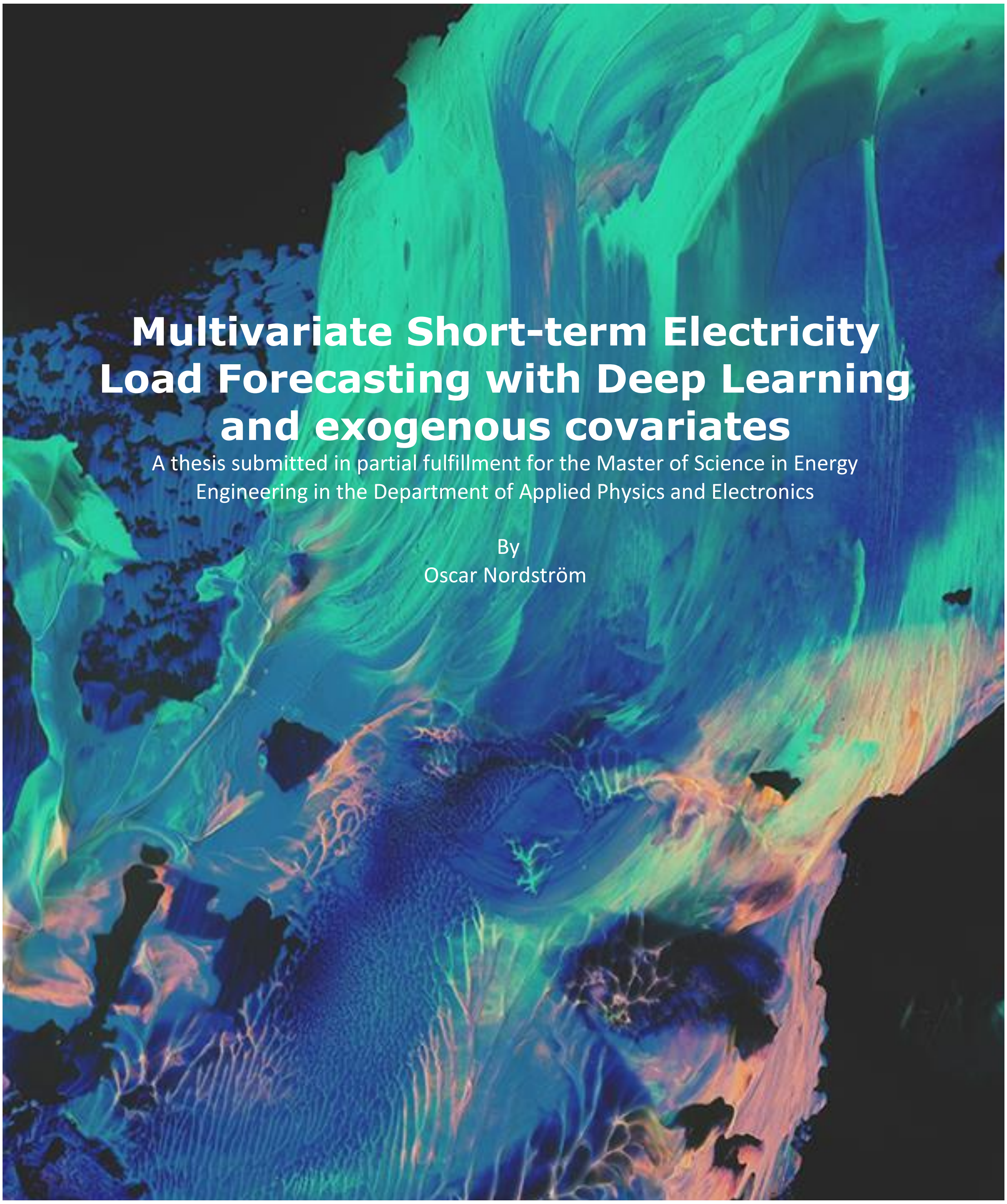
UMEÅ UNIVERSITY



Multivariate Short-term Electricity Load Forecasting with Deep Learning and exogenous covariates

A thesis submitted in partial fulfillment for the Master of Science in Energy Engineering in the Department of Applied Physics and Electronics

By

Oscar Nordström

Master thesis, 30 hp

MSc program in Energy Engineering, 300 hp

Department of Applied Physics and Electronics. Autumn 2020

UMEÅ UNIVERSITY

Abstract

Department of Applied Physics and Electronics

Master of Science in Energy Engineering

by

Maintaining the electricity balance between supply and demand is a challenge for electricity suppliers. If there is an under or overproduction, it entails financial costs and affects consumers and the climate. To better understand how to maintain the balance, can the suppliers use short-term forecasts of electricity load. Hence it is of paramount importance that the forecasts are reliable and of high accuracy. Studies show that time series modeling moves towards more data-driven methods, such as Artificial Neural Networks, due to their ability to extract complex relationships and flexibility. This study evaluates the performance of a multivariate Deep Autoregressive Neural Network (DeepAR) in a short-term forecasting scenario of electricity load, with forecasted weather parameters as exogenous covariates. This thesis's goal is twofold: to test the performance in terms of evaluation metrics of day-ahead forecasts in exogenous covariates' presence and examine the robustness when exposing DeepAR to deviations in input data. We perform feature selection on given covariates to identify and extract relevant parameters to facilitate the training process and implement a feature importance algorithm to examine which parameters the model considers essential. To test the robustness, we simulate two cases. In the first case, we introduce Quarantine periods, which mask data prior to the forecast range, and the second case introduces an artificial outlier. An exploratory analysis displays significant annual characteristic differences between seasons, therefore do we use two test sets, one in winter and one in summer. The result shows that DeepAR is robust against potential deviations in input data and that DeepAR surpassed both benchmark models in all of the tested scenarios. In the ideal test scenario where weather parameters had the most significant impact (winter), do DeepAR achieve a Normalized Deviation (ND) of 2.5%, compared to the second-best model, with an ND of 4.4%.

Acknowledgements

First and foremost, I would like to thank my supervisor at Vitec Energy, Christer Modin, for giving me this opportunity and guidance through this project. I would also like to thank my supervisor at Umeå University, Tomas Nordström, for valuable suggestions that improved this work's quality.

Contents

1 Introduction

The global energy demand will increase as the world's population continues to grow, and we face the challenge of finding alternative energy resources, compared to traditional generation technologies (fossil fuels such as coal, oil, and gas), to cover future energy needs [1], [2]. An alternative could be renewable energy resources such as wind power, hydropower, and photovoltaics. However, there is a downside to renewable resources such as wind and photovoltaics; they depend on weather conditions, which are stochastic by nature, to generate energy. Weather dependence is not only present in the generation of energy but also affects the temporal energy consumption. Suppose one looks at a household's electricity load during a long-term observation, then the household will have a greater demand for electricity during the winter than during the summer (especially in the northern hemisphere). Weather parameters such as clouds, wind, and rain can also influence the energy need if one looks at a short time interval. With the introduction of these variations in the electricity grid, it can be challenging to balance energy load and production [3].

According to Swedish law, electricity suppliers are obliged to deliver as much electricity as their customers consume. If there is an over-or under production: an imbalance, the electrical system does not work. The assignment to maintain the balance can be accepted by either the electricity supplier or outsourced to another organization, and it is up to Svenska kraftnät (SVK) to follow up on how well the balance is maintained. If there is an imbalance in the system, SVK will add or remove electricity so that equilibrium in the system occurs. The financial costs that arise for SVK must be reimbursed by the organization responsible for the balancing [4], [5]. The organization responsible for the balance tries to maintain a balance by planning their production based on short-term forecasts of the electricity load. Therefore, the forecast models must deliver a high degree of accuracy, both due to the economic benefits and from a climate point of view.

If the electricity load is logged in chronological order, it is a time series. A time series can display trends, seasonality, and patterns unique to the specific series. Thus to achieve high forecast accuracy, one must select a suitable model and tune it for that particular series, which can be a time-consuming process. Moreover, if there are also exogenous variables available, a feature selection procedure is usually necessary, either by domain expertise or the use of appropriate algorithms. Short term load electricity forecasting is a vast research subject, where proposed models span both the traditional statistical approach such as sARIMAX [6], and machine learning models such as support vector machines [7], or XG-boost [8]. Furthermore, most of the proposed models aim to improve forecasts for the univariate case.

As we move towards a more data-driven society, where the amount of data increases exponentially, we face challenges such as streamlining the production environment; can we efficiently produce multiple forecasts simultaneously and at the same time utilize external information to enhance the accuracy of the forecast? Are those methods robust against potential deviations in input data? The machine learning class Deep learning has revolutionized many research fields, including time series forecasting. Artificial neural networks (ANN) are a powerful tool in modeling high-dimensional data as they are compositional models, i.e., they aim at modeling complex relations by solving more simplistic equations [9], which makes them relatively robust against the curse of dimensionality. Research also shows that ANNs can effectively model noisy data [10]. The features of compositionality and noise resistance make models such as ANNs a practical approach to multivariate forecasting as ANNs aim to distinguish the underlying relationships between inputs and targets based on actual data [11].

A large aspect of a time series forecasting is to handle unexpected forecast deviations linked to the data. These deviations often stem from anomalies such as missing values or outliers prior to the forecast window. Even if a model, in theory, yields accurate forecasts, it may be that they are not optimal in production due to the consequences that can arise when the model receives data of a deviating nature. An example could be that a measurement error causes extreme or missing values in the input data, causing the forecast to follow the same deviating pattern, with severe consequences for decisions based on the forecast.

1.1 Purpose

The purpose of this thesis is to evaluate the performance of an autoregressive multivariate neural network in a realistic demand forecasting scenario of day-ahead forecasts of 28 independently collected electricity load series when exogenous covariates are present. Specifically, we will evaluate the proposed model in *Probabilistic Forecasting with Autoregressive Recurrent Networks* (DeepAR) [12]. The goal of this thesis is twofold: test the performance in terms of evaluation metrics of the forecast in exogenous covariates' presence, under the ideal condition that all necessary data is available and no interference is present, and to investigate the robustness of how DeepAR handles potential outliers and missing values in the conditioning range. Further interesting observations is computation time and general observations during the implementation of models. We measure the performance in terms of evaluation metrics root mean square error and normalized deviation and compare the metrics to two benchmark univariate models, one in the class of traditional time series models, the sARIMAX, and a machine learning model, which is provided by the client Vitec Energy.

1.2 Thesis outline

The structure of the thesis is as follows. We start by providing a background of time series analysis and the methods used to approach time-series forecasts. We account for the concept of artificial neural networks and the components that comprise DeepAR. The background is followed up by the experimental setup, which contains an in-depth explanation of the available dataset, on what system we run the experiments, and how we will evaluate the performance of DeepAR. We then move on to the experiments, where we perform feature selection on the covariates, account for the training procedure, hyperparameter tuning, and benchmark models. Following the experiment, we move on to results and discussion, where we discuss the practical challenges and strengths of DeepAR and how they relate to forecasting with exogenous covariates and robustness. And finally, a conclusion where we state our findings and future work.

2 Background

This chapter aims to provide relevant background information on time series analysis and its methodology when approaching time series forecasting. We begin by giving a general introduction to time series analysis and the theory behind one of the most widely used traditional methods, one of the benchmark models used in the thesis. We then go through the concept of Deep learning and the architectures of the neural networks and components that comprise DeepAR. Lastly, we introduce the metrics used to evaluate performance.

2.1 Time series and its methods

Time series are observations of data that have been logged in chronological order. A time series can manifest itself in many ways, from the simplest case where observations are entirely unrelated and evenly distributed. Other time series can have underlying characteristics such as trends, seasonal dependencies, or time-dependent variance. The objectives of time series analysis is to try to model such underlying characteristics to draw inference about the series. Below in figure 2.1 can one observe a decomposition of a simple simulated time series.

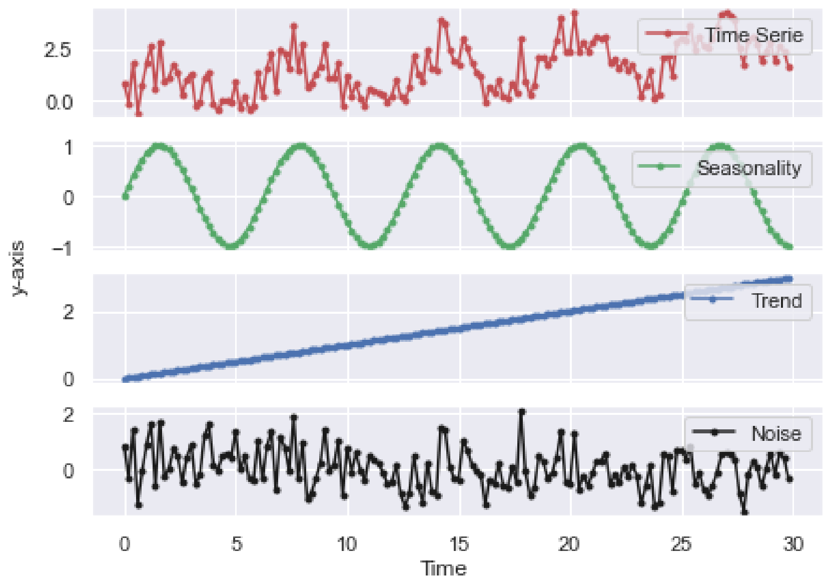


FIGURE 2.1: Example of time series, the seasonality is represented by a Sinus curve with an periodicity of 2π and amplitude 1, linear Trend: $y = 0.1t$ and Noise$\sim N(0, 0.5)$.

Time series forecasting can be divided into two submethods: traditional time series prediction methods and machine learning methods[13]. Traditional time series models strive, by definition, to find the joint distributions of a sequence of random variables, where the observed historical values in a time series are postulated as a realization of the random variables [14]. Brockwell and Davis express time series analysis as: "An important part of the analysis of a time series is the selection of a probability model (or class of models) for the data ".[14, p. 6]. There are several models within the traditional time series methods, such as Simple Exponential Smoothing, Holt Winter's Exponential Smoothing, or Vector Autoregression, but probably the best known is the Autoregressive Moving-Average (ARMA) model. From the definition, one can say that a time series $\{z_t\}$ is an ARMA process of order p and q if it is stationary and for every t,

$$z_t - \phi_1 z_{t-1} - ... - \phi_p z_{t-p} = \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} \tag{2.1}$$

where $\epsilon_t \sim WN(0, \sigma^2)$, and the polynomials $\phi(\epsilon) = 1 - \phi_1\epsilon - ... - \phi_p\epsilon^p$ and $\theta(\epsilon) = 1 + \theta_1\epsilon + ... + \theta_p\epsilon^p$ has no common factors [14]. As the name suggests, the ARMA model comprises two parts: the autoregressive- (AR) and moving average (MA) part. The AR part states that realization X_t is the sum of a linear combination of previous realizations and is represented on the left side of equation 2.1 by the coefficients ϕ. The MA part states that the realization X_t is a linear function of previous error terms ϵ and is represented on the right side by coefficients θ. The models within traditional time series forecasting require that the time series is weakly stationary, i.e., that the statistical properties are not dependent on time; for example, the mean of the time series is increasing/decreasing, or a seasonal component is present. To deal with potential temporal dependencies, one can instead model an adjusted time series that is differentiated to achieve stationarity and use a modified ARMA model: Seasonal Autoregressive Integrated Moving Average (sARIMA). The following is the general form of the sARIMA model, with the addition of linear regression for exogenous covariates, which yields the notation sARIMAX:

$$\Phi(B)_p \phi(B^s)_P \Delta^d \Delta_s^D z_t = \Theta(B)_q \theta(B^s)_Q \epsilon_t + \sum_{i=1}^{N} \beta_i x_i, \tag{2.2}$$

where s denotes the frequency of seasonality, Δ is the integration operator, B is the lag operator, x_i is the independent covariates with corresponding coefficients β_i, P and Q parameters refers to the order of seasonal- autoregressive and moving average terms, Φ and Θ refers to the coefficients of the seasonal autoregressive and moving average terms, D and s refers to the seasonal difference and seasonal frequency respectively [15].

Even though time series models provide a factual theoretical statistical basis for use, it is not necessarily the models that can generate the best accuracy. In a study of published methods for forecasting energy- demand in research articles, between 2005 and 2015, it was found that the most cited studies used neural networks to predict energy consumption [16]. It is worth mentioning that the authors also note the remarkable performance of the neural networks in found papers.

2.2 Deep learning

It is estimated that of the total amount of data available in the world 2019, 90% has been generated during the two previous years [17]. Furthermore, the amount of data growth does not appear to decrease soon, as Kaisler et al. state [18, p. 1]: "The current growth rate in the amount of data collected is staggering". A new trend for forecast modeling has emerged, where one is challenged with forecasting large datasets in the size of thousands to millions of time series [19]. In recent years, the concept of machine learning, especially Deep Learning, has drawn attention in both industry and academia. Deep learning has contributed to significant advances in image and speech recognition, natural language comprehension, autonomous vehicles, and more [12]. However, it is not self-evident that ML exceeds traditional statistical time series models in forecasting time series. Makridakis et al.[20] display a certain skepticism towards ML algorithms' performance in the field of forecasting time series. The authors argue that many of the published papers of forecasting ML algorithms lack objective relevance in ML's performance: ML algorithms' papers' performance is not compared to benchmark alternative time series models. The models are often trained and evaluated on a one-step-ahead forecast, and conclusions are drawn from a few time-series.

Deep Learning is a subclass of the machine learning genre, which utilizes stacked layers of Artificial Neural Networks (ANN) to model data. The word "Deep" in deep learning refers to the large number of stacked layers in a network. The design of a neural network is to, in a sense, to mimic the human brain where neurons are interconnected to form a complex network structure [21]. The network can then be trained to approximate specific problems using mathematical and statistical formulations. Rosenblatt designed the first artificial neural network in 1958 called the Perceptron [22], and it is still a fundamental building block in most ANN architectures. However, with some modifications introduced by Hopfield [23]. The Perceptron is a feed-forward neural network, and the name feed-forward originates in that the signals always move in one direction, from the input layer to the output layer. Figure 2.2 illustrates the structure of a Perceptron.

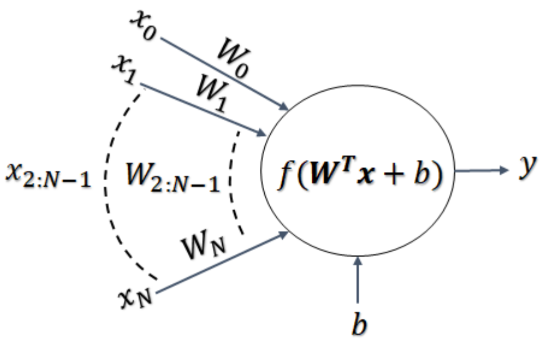


FIGURE 2.2: The neuron (Circle) receives the sum of the linear representation of the input data $\mathbf{W^T x}$, adds a constant bias term b, and lastly, it applies the nonlinear activation function f to the product. The activation function's purpose is to squeeze the outputs to the value-range of 0 to 1 or -1 to 1, and the value of the output determines whether the neuron is active or not.

A single perceptron can perform simple regression and classification tasks. In the classification task, we pass the signal through the activation function f to output probabilities, used to determine the class. For the regression task, we remove the activation function, thus outputting a linear regression. However, Deep learning's real strength lies in the fact that it is possible to connect networks to create larger model that can handle more complex tasks. Figure 2.3 below displays a typical architecture of a fully stacked feed-forward neural network, a Multi-Layered Perceptron (MLP), consisting of an input layer, two hidden layers, and finally an output layer.

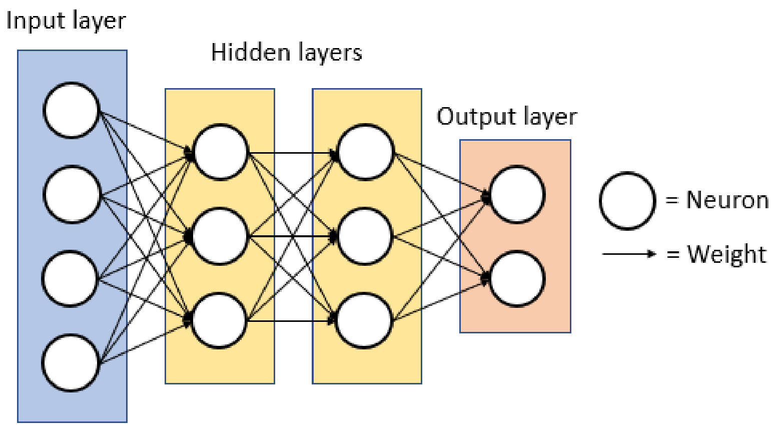


FIGURE 2.3: An example of a Artificial Neural Network, the Multi Layered Perceptron. The network contains one input layer with 4 neurons (blue), two hidden layers (yellow) with 3 neurons respectively and last an output layer (red) with 2 neurons.

The training of a feed-forward neural network is an iterative process where we start with a forward pass. The forward pass is when the network computes an output by sending the inputs through the network layers. The obtained output is then compared with the actual values, i.e., we calculate an error function (loss value). The last step is to perform a backward propagation of errors with an optimizer such as the gradient descent. The gradient descent method refers to the process of when we adjust the network's weights and biases by adjusting the respective matrices in an iterative stepwise process to minimize the loss function. The rate at which we adjust the parameters is defined as the learning rate and is a crucial hyperparameter of a neural network's performance. Large learning rates can cause the network to perform poorly due to the optimizer tend to step past a loss minimum. If the learning rate is too low, it may lead to more iterations to reach a converged state and the loss minima. One iteration of the process forward pass, calculate loss value, and perform the backward propagation is called an epoch [24]. The weights are normally initialized randomly.
Training of neural networks is a complex process, where there are numerous types of suggested optimizers and methods, and an in-depth explanation can be considered redundant in this thesis. For further information, see appropriate textbooks on neural networks like [25].

MLP networks can be used in time series modeling, assuming that all inputs are independent [26]. If the input is a sequence, the MLP cannot memorize how the order elements in the sequence are connected; for example, after a Monday comes a Tuesday. Thus, they may not be the optimal choice of the model when dealing with time-series data.

2.3 Recurrent Neural Network

The MLP network propagates the input data through the cells without considering what the network has previously processed. One way to create an artificial memory for MLP is to feedback information from previous inputs to a later time step - A Recurrent Neural Network (RNN). The RNN computes a hidden state (at the hidden state in the MLP) for each input and then propagate the hidden state forward to the next time step. Below in figure 2.4 is an illustration of a simple vanilla RNN. On the right side of the figure is the corresponding unfolded RNN. The unfolded RNN illustrates how the single RNN cell processes the elements in a sequence one at a time, and the folded RNN is how one could present a compact form of the vanilla RNN.

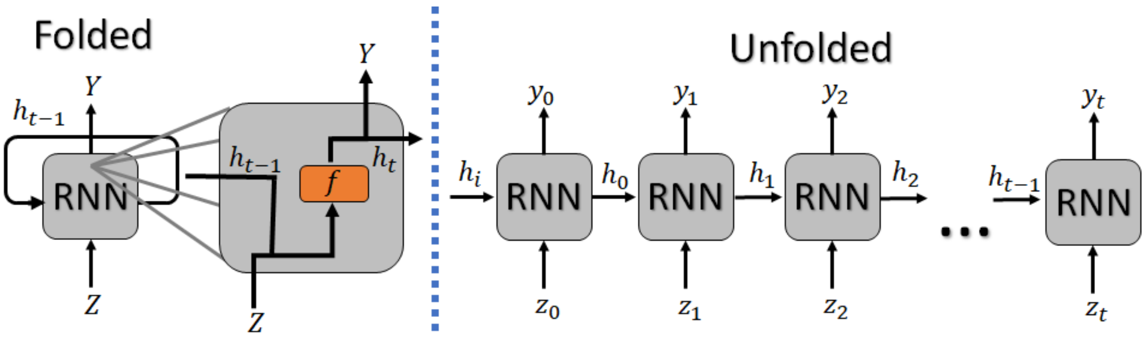


FIGURE 2.4: Example of a vanilla Recurrent Neural Network. The image is supposed to illustrate how the folded network is unfolded to be able to process input sequence z. The network takes the inputs $Z \in \{z_0, z_1...z_t\}$ one at a time and the outputs the corresponding predictions $Y \in \{y_0, y_1...y_t\}$. Note the hidden state h that is passed from previous to next time step.

The hidden state is a vector representation of how the network interprets previously seen information in the input data and is calculated by:

$$h_t = f(W_{hh}h_{t-1} + W_{zh}z_{t-1} + b), \tag{2.3}$$

where f is a non-linear activation function, W_{hh} is the hidden state weight matrix, W_{zh} is the weight matrix with respect to the inputs and b is the bias term. The output at time step t is computed by the following equation:

$$y_t = W_{yh}h_t \tag{2.4}$$

where W_{yh} is the weight matrix with respect to the output. Suppose that we are dealing with a time series and want to predict the next day of the week, using the previous six days as input, then the folded network on the left in figure 2.4 is unfolded in an iterative process for a total of 6 times. At the first time step $t = 0$, the network receives the initial hidden state h_i and the first input z_0, computes h_1 according to equation 2.3, use obtained h_0 to calculate y_0 with 2.4. Before ending the first iteration, we save the hidden state h_0 and compute the loss value L_0. At $t = 1$ we fetch the saved previous hidden state h_0 and update to h_1 to compute y_1 and L_1. We repeat the procedure for six iterations, and in the end, we compute the final loss by summing the current time step loss L_6 with the previous time steps $\sum_{i=1}^{5} L_i$; thus, we calculate the cumulative loss by stepping backward in time - backward propagation through time. Note that the weight matrices is shared through all iterations.

2.4 Long Short Term Memory network

The Long Short Term Memory(LSTM) is a recurrent neural network architecture that was first invented in 1997 [27]. LSTM aims to overcome the problem of vanishing gradient problem that can occur in the training process of a vanilla RNN. The vanishing gradient problem derives from the backpropagation through time algorithm where each neuron in a layer calculates its gradient with respect to the previous adjacent layer. If the adjustments of the weights in the adjacent layer are small, it follows that the adjustments in the current layer are even smaller. Thus the gradient exponentially shrinks towards zero. The LSTM negates the vanishing gradient problem by introducing a cell state, C, and input-, output- and forget gates, denoted i, o and f, respectively. The cell state acts as the memory for LSTM, where information segments can be removed or added using regularizing gates. These gates act as a kind of filter that helps the LSTM cells choose between which information is essential to the outcomes prediction power. Figure 2.5 shows the structure of a standard LSTM cell.

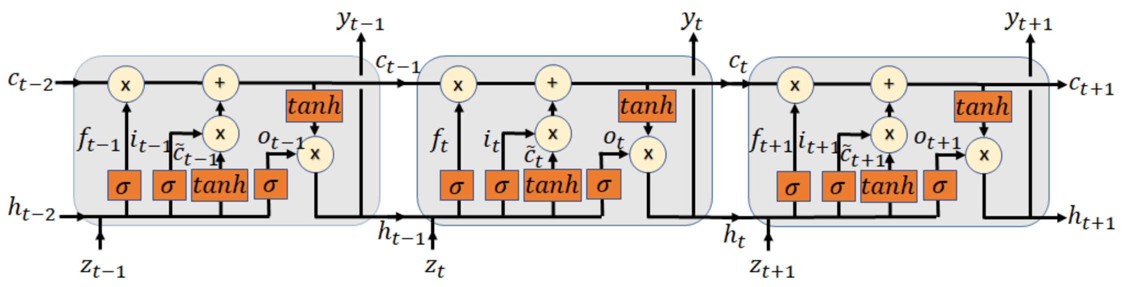


FIGURE 2.5: The structure of a standard LSTM cell. The figure is redone with inspiration from [28]

Figure 2.5 illustrates how information from the previous and current time step flows from left to right, through the gates, and then removed/added to the cell state by element-wise multiplication and addition. The governing equations for the cell state, C_t, candidate state, $\tilde{C}_t$, and hidden state h_t in the LSTM cell is given by:

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t, \tag{2.5}$$

$$\tilde{C} = tanh(W_{\tilde{C}h} h_{t-1} + W_{\tilde{C}z} z_t + b_{\tilde{C}}), \tag{2.6}$$

$$h_t = o_t \cdot tanh(C_t), \tag{2.7}$$

where $W_{\tilde{C}}$ is the weight matrix with respect to the candidate state, $b_{\tilde{C}}$ is the bias with respect to the candidate state.

The governing equations for the gates is stated as:

$$f_t = \sigma(W_{fh} h_{t-1} + W_{fz} z_t + b_f), \tag{2.8}$$

$$i_t = \sigma(W_{ih} h_{t-1} + W_{iz} z_t + b_i), \tag{2.9}$$

$$o_t = \sigma(W_{oh} h_{t-1} + W_{oz} z_t + b_o), \tag{2.10}$$

where W_f, W_i and W_o, the respective weight matrices to the gates f, i and o, σ is the Sigmoid function that squeezes the outputs to the range 0 to 1 and b_f, b_i and b_o is the respective bias [29]. The outputs at each time step are computed similarly to a vanilla RNN by sending the hidden state through an output neuron.

The LSTM network has received much attention as a reliable forecast model with several articles that can prove its performance [30], [31]. However, some studies show the opposite [20], so it is advisable to be careful when evaluating LSTM models on time series prediction.

2.5 Encoder-Decoder architecture

Recall from the sections [2.3, 2.4] how a vanilla RNN and a standard LSTM network have the property that they unfold as many times as the length of the input sequence and give a prediction at each fold. With this unfolding property, RNN and LSTM have modeling problems where input and output sequences differ in shape.

The Encoder-Decoder architecture was proposed by Sutskever et al.[32] and the intended purpose is the sequence to sequence problems like language translation. However, it has been shown that it can also be applied to time series forecasting, with promising results [33], [34]. The encoder structure consists of stacked LSTM layers, which reads the input data, one at a time, and recode it into a vector format with a fixed dimension. After the encoder has processed the input data (returning a vector of fixed-length), it passes it through to the decoder structure. The decoder takes the vector representation, decodes, and returns the desired representation. Figure 2.6 below, displays the encoder-decoder structure with stacked LSTM layers. The input is represented with the notation z, prediction with y and C and h is the hidden- and cell state, respectively, within the LSTM units.

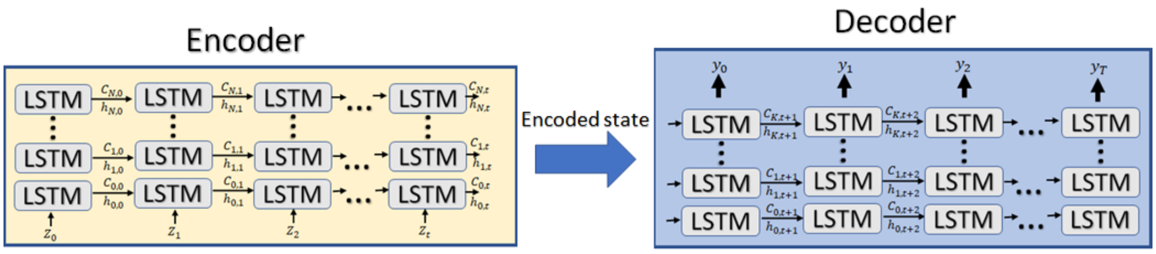


FIGURE 2.6: The figure illustrates Encoder-Decoder architecture. The Encoder reads the inputs $z_i \in \{z_0, z_1 ... z_t\}$ one at a time and propagates the internal states forward until the end of the input sequence. The obtained predictions in the Encoder is discarded. The Decoder is initialized by the encoded state and unfolds T times where T denotes the length of the sought after output sequence and predicts $y_i \in \{y_0, y_1 ... y_T\}$. Note that N and K denotes the number of layers in the Encoder and Decoder respectively.

One can view the Encoder-Decoder as two separate models. The Encoder aims to perform feature extraction on the input sequence while the Decoders purpose is to interpret the extracted features and output the target sequence. The two models do not necessarily share weights.

2.6 Deep Autoregressive Neural Network

Salinas, Flunkert & Gasthaus [12] proposed a multivariate deep learning time series model: Deep Autoregressive Neural Network (DeepAR). The proposed model aims to create a global model of a multivariate data set, containing related time series instead of creating individual models for each time series. In this way, the model can extract interrelationships between the time series to generate high accuracy forecasts. The model can also extract underlying dependencies from related time series and covariates, thus minimizing feature engineering. DeepAR takes as an input an adjustable conditioning range for all available series and generates a multi-step horizon forecast for each series, with the help of exogenous covariates. Below in figure 2.7 illustrates how the model receives previous values, covariates, and outputs a multi-step forecast.

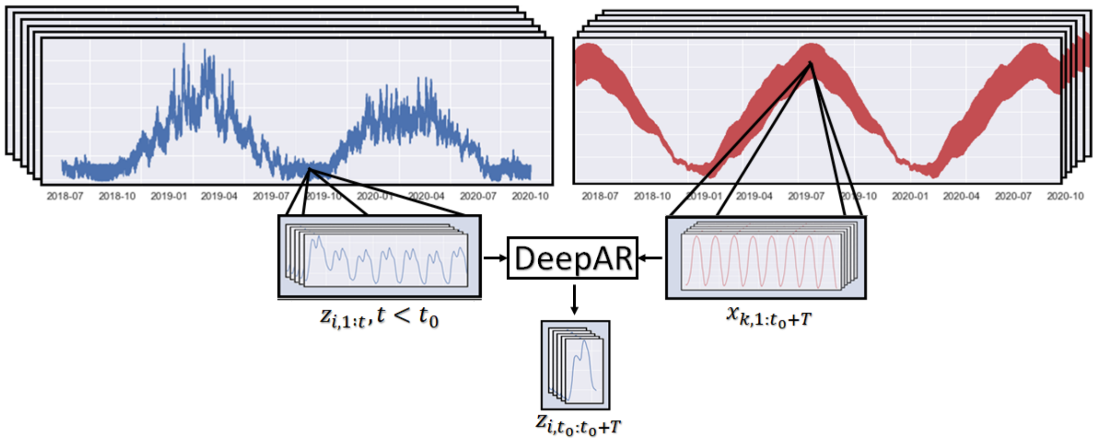


FIGURE 2.7: The figure illustrates a simplification of what DeepAR takes as input and generates a multi-step forecast. t_0 is the start time of the forecast range and T is the end.

DeepAR is a probabilistic forecasting model due to that the model aims to estimate the conditional distribution for each time series value $z_{i,t}$, at time t:

$$P(z_{i,t_0:T}|z_{i,1:t_0-1}, X_{i,1:T}), \tag{2.11}$$

where T is the end of the forecast window, $z_{i,1:t_0-1}$ is the past values, t_0 is the start of forecast range, $X_{k,1:T}$ is the covariates (note that the covariates is known for the whole range of the time span). In the continuous domain, the conditional distribution can be considered an continuous probability distribution. Equation 2.11 above can be described as if we have a set of realizations of previous continuous values $z_{i,1:t_0-1}$ and associated covariates $X_{k,1:T}$, we want to find the conditional probability density distribution that maximises the likelihood of observing the future values $z_{i,t_0:T}$. To obtain the conditional distribution, one replaces the cost function for the recurrent neural network with the following likelihood functions:

$$Q_\Theta(z_{i,t_0:T}|z_{i,1:t_0-1}, X_{i,1:T}) = \prod_{t=t_0}^{T} \ell(z_{i,t_t}|\theta(h_{i,t}, \Theta)) \tag{2.12}$$

where $h_{i,t} = h(h_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta)$ is a parametrization of an autoregressive recurrent neural network and Θ denotes the model parameters.

Recall the Encoder-Decoder structure presented in section 2.5 where we train two modules, one to encode and one to decode. DeepAR's core architecture builds on the same concept as the Encoder-Decoder structure. Instead of designing two separate modules, did the authors propose a single module for both phases, with shared weight matrices and parameters. The module is trained to output a one-step-ahead forecast at each unfolding of the LSTM. During the encoding phase, do the module receive the values in the conditioning range, one at a time, of the previous time step $z_{i,t-1}$ and covariates at current time step $x_{k,t}$ and outputs a one step ahead forecast $\hat{z}_{i,t}$. The model is autoregressive in that it uses past values as inputs to the next layer to generate future values in the inference phase. DeepAR also incorporates an item/group dependent embedding vector, which picks up item-specific properties for each time series. A summary of the model is displayed below in figure 2.8.

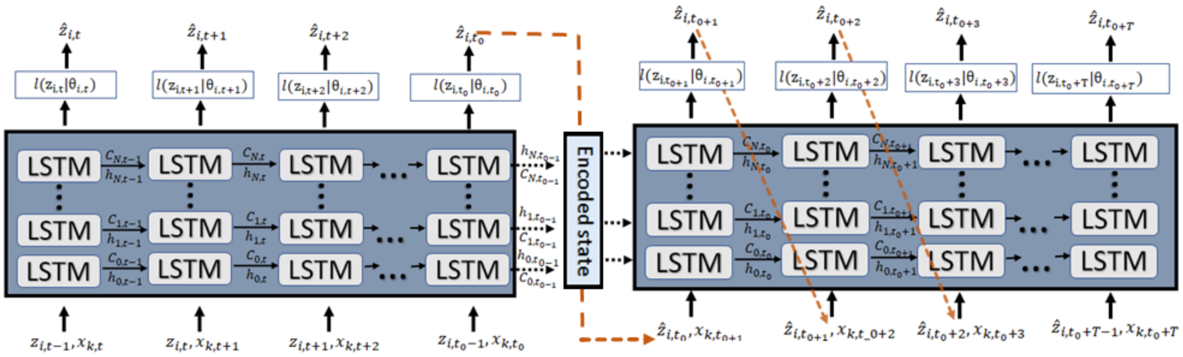


FIGURE 2.8: Summary of the proposed model Deep Autoregressive Neural Network. The left section of the figure displays the training phase (Encoder), where the network receives the covariates $x_{k,t}$, the previous target values $z_{i,t}$, where $t < t_0$ and outputs the hidden state $h_{t,i}$ which is used to compute the one-step forecast $\hat{z}_{i,t+1}$. During the training, are the outputs $\hat{z}_{i_t}$ used to compute the loss function and tune the parameters Θ of the model. The right part of the figure displays the inference phase (Decoder), $t \geq t_0$. DeepAR receives information of previous values through the Encoded state, and output the parameters of a distribution, draw a sample $\hat{z}_{i,t}$ and pass that sample forward to the next LSTM layer until the end of the forecast window is reached. Note that the last output in the encoder $\hat{z}_{i,t_0}$ is part of of the forecast range. A pass of the above process is called a sample trace. By performing Monte-Carlo sampling, we can sample multiple traces to estimate the joint predicted distribution, which yields the target median, confidence intervals, and quantiles of interest.

To tune the model in the training phase, i.e., try to find the optimal values of the model parameters Θ, one takes the logarithm of the likelihood and applies the stochastic gradient descent to maximize the log-likelihood function:

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{t=t_0}^{T} log(\ell(z_{i,t}|\mu(h_{i,t}), \sigma(h_{i,t})). \tag{2.13}$$

DeepAR handles data from various distributions, such as Binomial, Bernoulli, Beta, or Gaussian. Under the assumption that we are dealing with Gaussian data, the likelihood function is equivalent to the probability density function:

$$\ell(z|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(z-\mu)^2}{2\sigma^2}}. \tag{2.14}$$

2.7 Overfitting and Dropout

A model in the class of machine learning is defined as overfitting when it fails to generalize on new data for which it has not previously been presented. If a network is overfitting, the network has learned to explain training data at a detailed level but fails to explain unseen data on a similar detail-level. One solution to prevent overfitting is to train multiple models with the same parameter configuration, on the same training set, and aggregate their predictive capabilities (possible due to the stochastic nature of the initialization of the weights). However, this is a time-consuming and computationally demanding process.
Dropout is a regularization technique used to prevent overfitting for neural networks. Applying dropout in a neural network introduces a probability p that a neuron is masked (dropped out) during the training phase, which facilitates the network's ability to generalize on new data. One can view the introduction of dropout to train multiple models within the training session; it is not as computationally demanding as training multiple models and aggregate the predictions but achieves the same objective at a smaller cost [35]. Dropout is a recognized control method but may not be the optimal choice for RNNs because when one masks neurons in an RNN, noise is introduced to the signal. The introduced noise is then amplified as the signal propagates through the network, which may impair the RNN's ability to predict. Instead of applying dropout to each neuron in the network, one can instead apply recurrent dropout to the input, and output layers of the recurrent layers in the model [36].

2.8 Evaluation metrics

The metrics used to evaluate selected models' performance are the Normalized Deviation (ND):

$$ND = \frac{\sum_{i,t} |z_{i,t} - \hat{z}_{i,t}|}{\sum_{i,t} |z_{i,t}|}, \tag{2.15}$$

and Root Mean Squared Error ($RMSE$):

$$RMSE = \sqrt{\frac{1}{N} \sum_{i,t} (z_{i,t} - \hat{z}_{i,t})^2}. \tag{2.16}$$

The main difference between the two measurement values is that $RMSE$ is more sensitive to extreme forecast errors than ND, which yields an overall percentage error and is less receptive to whether each error is of high or low magnitude.

3 Experimental setup

This chapter intends to give the reader an outline of the dataset we use by an exploratory analysis that provides an increased understanding of important covariates and the different characteristics within the time series we are going to model. The chapter also discusses which system we use, how to preprocess data, and a particular sampling scheme we use during training. We also develop problem statements that are interesting to answer in a realistic forecast scenario.

3.2 Data and exploratory analysis

On November 1, 2011, Svenska Kraftnät divided Sweden into four electricity areas. Electricity area 2 includes the County's: Jämtland, Västernorrland, Gävleborg, Västerbotten, and parts of Dalarna. The purpose of the division was to counteract restrictions and to be able to ensure continuous energy supply to the inhabitants of these regions. The division was necessary as Sweden has an uneven energy production, which means an electricity deficit in southern Sweden arises. Electricity area 2 is further divided into grid areas. In each grid area, a unique supplier is responsible for maintaining the balance between electricity supply and demand.

Electricity Load in Sweden is logged every hour for the outlet points with a magnitude above 63 amps on their main fuse (note that a typical household manages with a 16 amp fuse). All outlet points below 63 amps are logged once a month. The Electricity Load dataset used in this thesis is obtained by calculating the residual of all supplied, logged energy, and eventual losses in the electricity grid per hour. Thus is the obtained residual logged per hour. One can summarize the dataset as the Netto residual of all non-hourly logged electricity outlet points under 63 amps (thus, it consists mostly of households) in electricity area 2. The data consists of 28 independently logged Load series, representing different grid areas within electricity area 2 with a time range of 2016-10-12 to 2020-09-01, logged on an hourly resolution. Associated with the electricity load is the covariate dataset. The covariate dataset consists of day-ahead forecasts of various weather parameters and represents the geographical area that electricity area 2 spans. The weather forecasts consist of the parameters: wind speed in $[m/s]$, global radiation $[W/m2]$ (can be interpreted as all short-wave radiation, both direct and diffuse, that hits a horizontal surface), Solar Angle [°] and temperature [C°]. Temperature, wind speed, and global radiation are represented by 33 locations, while the solar angle is given by 28. All of the provided covariates span the same time range as the Electricity Load dataset. An unknown number of external weather institutes provide the weather forecasts, and there is no information on how weather forecasts correlate to the actual values. All data used in this thesis are encoded to preserve confidentiality and is provided by the client Vitec Energy.

By inspecting the Load series in the dataset, it is concluded that all show a similar pattern and are correlated. Therefore, the exploratory analysis will only focus on Load series 1, and drawn conclusions are assumed to be true for all series. Figure 3.9 below displays Load series 1, with a sample of three weather forecasts for each parameter.

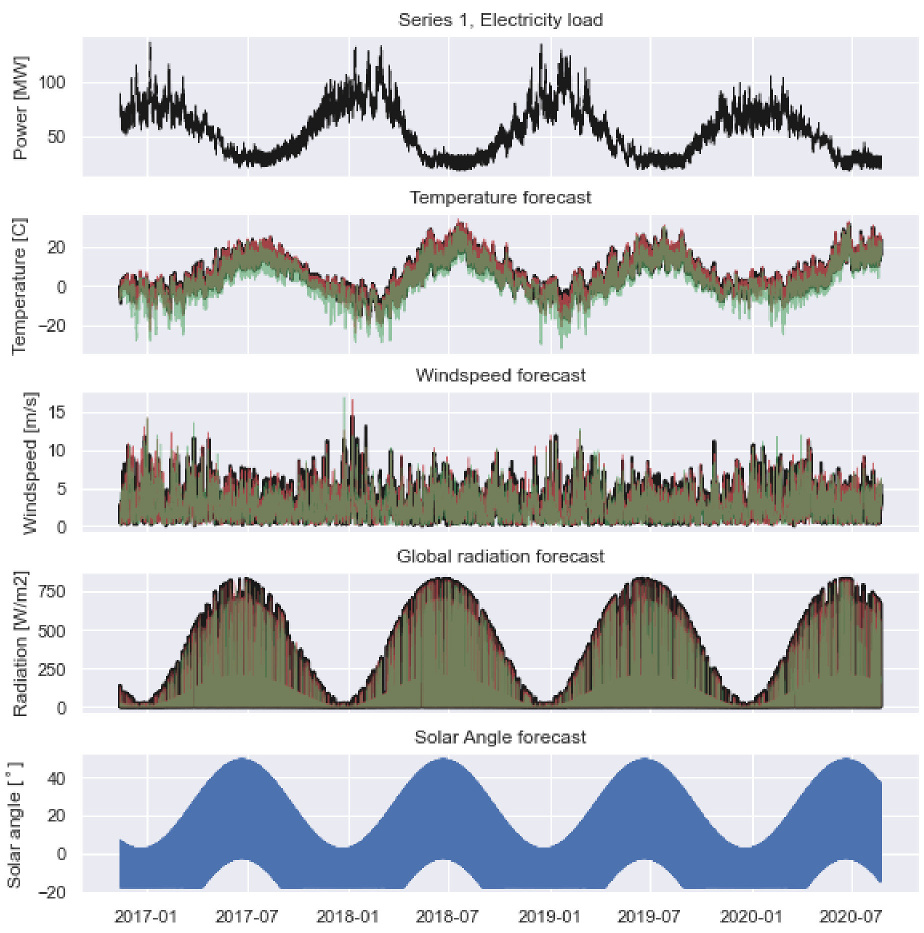


FIGURE 3.9: The figure displays Load series 1, with a sample of three weather parameter for each quantity; Temperature, Wind speed, Global radiation and Solar angle.

The annual Electricity load follows a clear seasonal dependence, where the maximum occurs during the winter season (136 MW in 2017-01-05 16:00:00) and the minimum during the summer season (18 MW in 2020-06-26 04:00:00). As expected, one can also observe that if it is a high load period, the temperature is low and vice versa for the summer. The only parameter that deviates from a clear annual seasonality is the wind forecast, which only shows a slight dependence. One theory as to why we see this phenomenon is that the weather forecast represents Sweden's hinterland, where the sea does not significantly impact the seasons, resulting in a stationary trend with only the wind's stochastic nature as a variation. Figure 3.10 displays one week during august of Load series 1, with three arbitrarily selected forecasts for each weather parameter. In the figure, we can note that the electricity load is stationary around 27 MW and whose load does not differ much daily, only a slight level shift during the first two days. Simultaneously, we can observe that temperature and wind speed forecast have their peaks during the observed period.

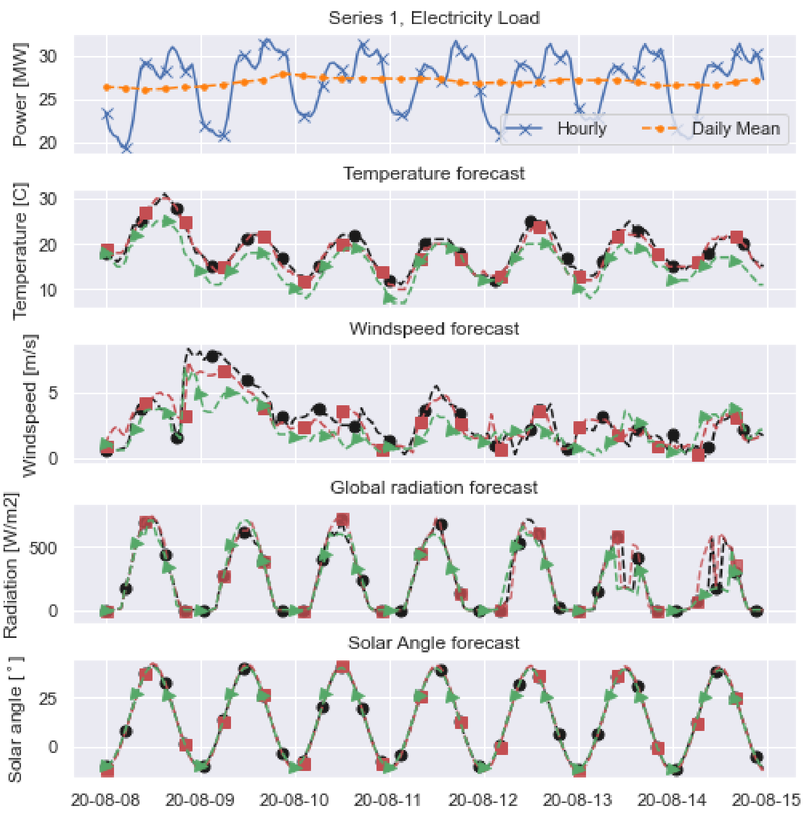


FIGURE 3.10: Load series 1 for a week during summer, with three arbitrarily selected forecasts for each weather parameter.

During the winter, electricity load displays a distinct characteristic difference compared to the summer, especially the daily average. The daily average in the winter season does not exhibit stationary behavior but can vary significantly from one day to another, from here on denoted level shift, as seen between the dates 2020-02-09 to 2020-02-12 in figure 3.11. A probable explanation for these variations is that the temperature increases or decreases, which follows that the household heating supplied varies. One can also observe that the hourly variations also differ between the two seasons, indicating that the consumers' typical energy pattern also differs. There can be many factors that can affect the daily temporal pattern of the load. For example, there could be fewer sunshine hours during the winter, which means that the light sources are turned on earlier or that the households' social routines are altered.

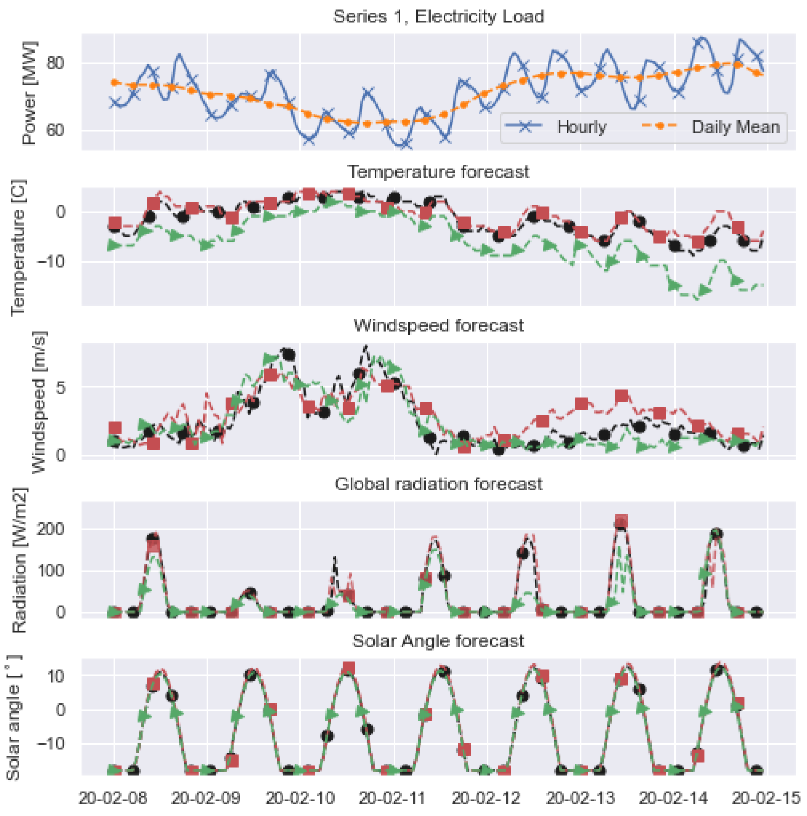


FIGURE 3.11: Load series 1 for one week during winter, with three arbitrarily selected forecasts for each weather parameter.

The plausible distributions between the seasons are furthermore explored in figure 3.12, in which one can observe that there is a clear distribution difference between the two seasons. The winter season is similar to a Gaussian distribution with a large standard deviation, while the summer season is suggestive of an exponential Gaussian distribution. The overlap between the distributions may indicate that it is further possible to divide the data into autumn and spring seasons.

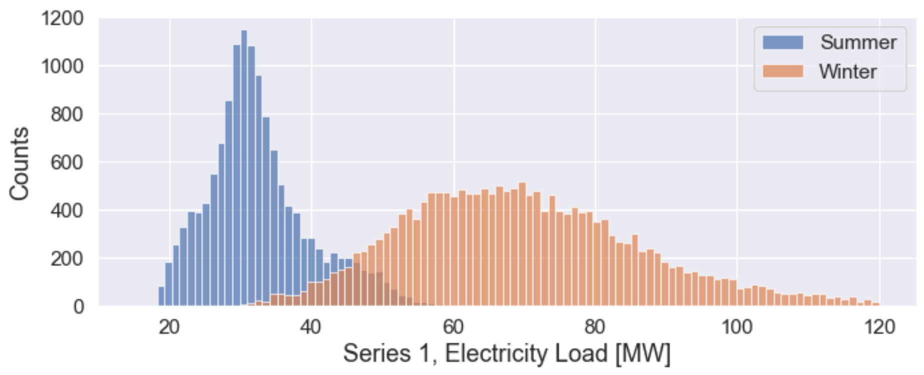


FIGURE 3.12: The figure shows the seasonal count distributions for winter and summer for Load series 1. The summer season is defined as the months of May-September and winter the rest.

The electricity load dataset represents mostly households. Therefore there should be an apparent weekly periodicity, where electricity use is higher during weekends and holidays than during the days that count as a working week. However, when examining possible seasonal dependence, figure 3.13, it was discovered that there was no noticeable difference in median value. Though there is a difference in load variation between the days of the week, and one can observe that it is more likely that extreme values occur during Thursdays, Fridays, and Saturdays.

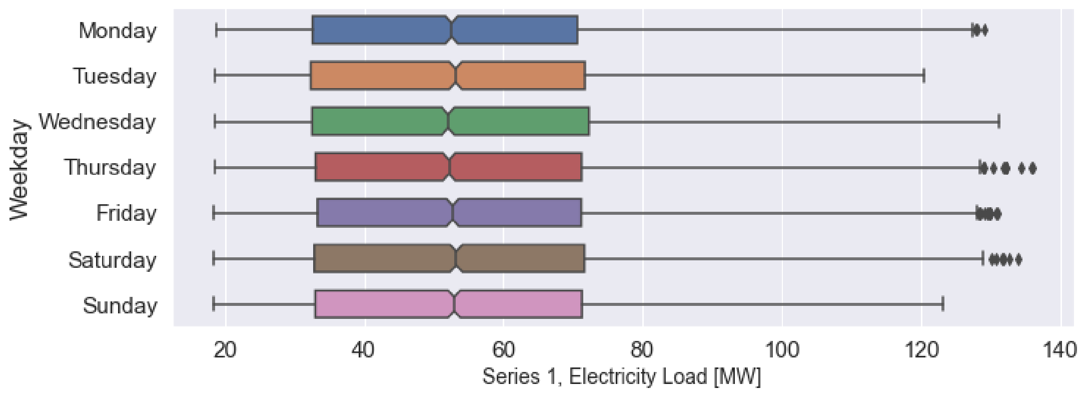
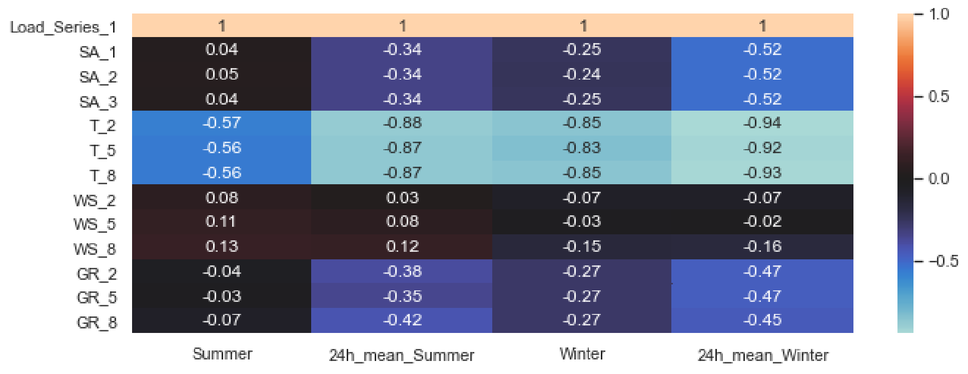


FIGURE 3.13: Boxplot of Load series 1. where the values has been divided categorically according to the day of the week, to visualize potential weekly seasonality.

We perform a correlation analysis on the weather parameters to investigate which of them is significant to use in the training of DeepAR, see figure 3.14. The correlation of the daily average value may indicate whether any of the parameters affect the load series's level shift. We calculate the correlation for three arbitrarily chosen weather parameters. Not unexpectedly, the parameter Temperature displays by far the highest correlation with the electricity load for both seasons, and the wind speed has a limited degree of explanation.



	Summer	24h_mean_Summer	Winter	24h_mean_Winter
Load_Series_1	1	1	1	1
SA_1	0.04	-0.34	-0.25	-0.52
SA_2	0.05	-0.34	-0.24	-0.52
SA_3	0.04	-0.34	-0.25	-0.52
T_2	-0.57	-0.88	-0.85	-0.94
T_5	-0.56	-0.87	-0.83	-0.92
T_8	-0.56	-0.87	-0.85	-0.93
WS_2	0.08	0.03	-0.07	-0.07
WS_5	0.11	0.08	-0.03	-0.02
WS_8	0.13	0.12	-0.15	-0.16
GR_2	-0.04	-0.38	-0.27	-0.47
GR_5	-0.03	-0.35	-0.27	-0.47
GR_8	-0.07	-0.42	-0.27	-0.45

FIGURE 3.14: Correlation table between Load series 1 and randomly selected covariates, 3 from each parameter, for the seasons winter and summer. The correlation between the cases is displayed, hourly correlation and 24h rolling mean. The hourly correlation indicates whether variations within one of the covariates can explain short-term deviations in energy load. The rolling 24h rolling mean indicates if any of the covariates displays a broader dependency. SA denotes Solar Angle, GR is Global Radiation, T is Temperature and WS is Wind Speed.

3.3 Standardization and batch sampling

The covariates are standardized with a Z-score to achieve inputs with zero mean and unit variance:

$$Z_{k,t} = \frac{x_{k,t} - \bar{x}_k}{\sigma_{x_k}}, \tag{3.17}$$

where σ_{x_k} is the standard deviation and $\bar{x}_k$ is the mean value for covariate k. Missing values is excluded in the calculation of the Z-score. Instead of applying a normalization process to the data before forwarding it to the model, we instead normalize each training window with an item dependent scale factor v_i:

$$v_i = \frac{1}{t_0}\sum_{t=1}^{t_0} z_{i,t} + 1. \tag{3.18}$$

This form of item dependent normalization with a scale factor is performed due to two reasons. The first is that the nonlinearities in the network cause a scaling problem between input and outputs. The second argument is that with the obtained scaling factors v_i we can apply a weighted sampling scheme that facilitates DeepAR's ability to model high-scale time series.

3.3.1 Weighted batch sampling

Suppose that high-precision forecasts for Load series with a large scale is of greater importance in a production scenario than accurate forecasts of series with a lower scale. By introducing a weighted sampler, we let DeepAR sample non-uniformly on training samples with a higher amplitude, facilitating the model's ability to recognize trends and patterns present in a large-amplitude load series. The weighted sampler utilizes a multinomial probability distribution, with replacement, where the probability of occurring is proportional to the normalization factor v_i. For each batch iteration, we call the weighted sampler with a weight matrix $W = |v_i| / \sum_{i=1}^{N} |v_i|$, where N represents the number of training windows in the training set. The weighted sampler then draws a sample, via the multinomial distribution, and returns n-number of training windows, where n denotes the batch size.

3.4 Computerware

We use the programming language Python to perform the experiments. More specifically, we use PyTorch [37] as our neural network framework. We do not implement DeepAR's architecture but utilizes a re-implementation, found at the GitHub-repository [38]. We run the experiments on a system consisting of the Central Processing Unit (CPU) Ryzen 7 3800x (8 cores, 16 threads), and the Graphics Processing Unit (GPU) Nvidia Geforce RTX 2070.

3.5 Evaluation method

To test the robustness of DeepAR, i.e., test the performance when we expose DeepAR to temporal anomalies in the input data, we simulate two scenarios. The first scenario introduces an extreme value, an outlier, into the conditioning range and measure how a fully trained model's performance is affected in the inference phase. However, how do we define what an outlier is? There is no unified definition of how an anomaly should be classified, but in general, one can consider values that deviate from a time series's normal behavior [39]. In this thesis, we introduce an anomaly under the assumption that data follows a Gaussian distribution that deviates by three times the standard deviation (the equivalent of 2.7% probability of observing the value) from the mean in the conditioning range.

The second case is to simulate how DeepAR handles missing data. That data is missing at the time of forecast is not uncommon. This thesis focuses on a joint model of electricity load for 28 grid areas, resulting in 28 unique balance responsible organizations. Each organization is responsible for validating the data quality and then entering it into the database before the forecast. That necessary data is always available can be considered naive; thus, one can introduce a quarantine period to the input data. The quarantine period implies that we mask the data closest to the forecast range and assume that the latest available input data is a couple of days behind. However, exogenous covariates are assumed to be known throughout the conditioning range. In this thesis, we test the robustness of DeepAR to missing values by masking two time periods prior to the forecast range, 24 and 48 hours, Quarantine period 2 and 3, denoted Q_2 and Q_3, respectively. Quarantine period 1, Q_1, is considered the ideal case and implies that the forecast range's target data is always unknown to the model and that all data in the conditioning range is known. A clarification: if we assume that the conditioning range is 168 hours and we want to forecast the next 24 hours (total of 192 hours), then the hours 144 to 192 are masked in Q_2 and 120 to 192 for Q_3 as illustrated in figure 3.15.

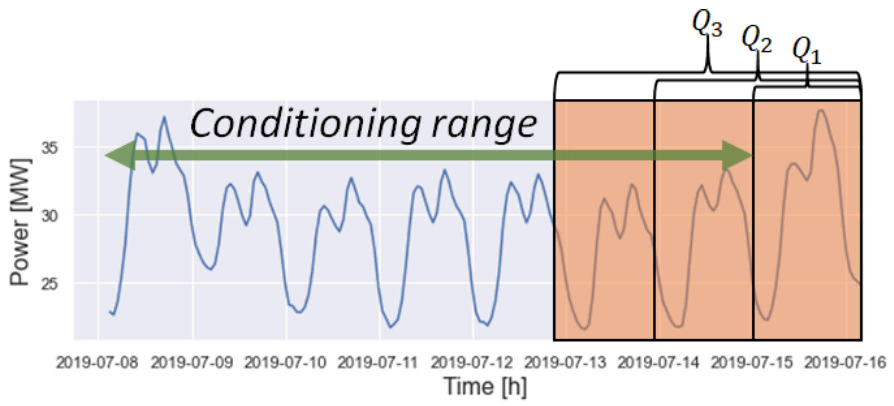


FIGURE 3.15: Illustration of masked values in the quarantine periods. Q_1 denotes that target data always is unknown in the forecast range, Q_2 the forecast range and the previous 24 hours, Q_3 the forecast range minus the 48 previous hours.

4 Experiments

This chapter will present the methodology used to obtain benchmark models, go through hyperparameter tuning for DeepAR, which covariates we use, how we set up the training procedure and a feature importance algorithm for further interpretation of the used covariates.

4.1 sARIMAX

We assume that all series exhibit similar properties in seasonality- and trend components due to the complexity and infeasibility of finding the adequate order of autoregressive and moving average terms in the sARIMAX models by manual investigating the corresponding auto- and partial autocorrelation functions. Using the above assumption of similarity, we remove the trend by first-order differentiation ($d = 1$) and the seasonal component by differentiation of order ($D = 1$) and periodicity of 24, to ensure stationarity on all series, following the residuals in figure 4.16.

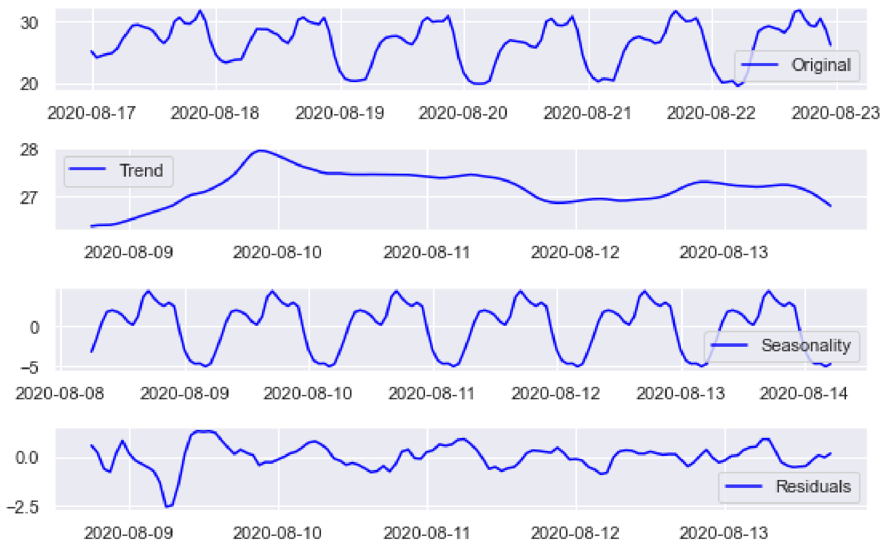


FIGURE 4.16: Time series decomposition during the summer season for Load series 1. The decomposition displays a clear 24-hour seasonality and a trend component. The residuals do appear to be approximately stationary and centered around zero.

Finding the order of autoregressive (p and P) and moving average terms (q and Q) of the sARIMAX models is set up as an iterative process, where each unique Load series is fitted with the Auto-Arima package PMDARIMA. Auto-Arima uses a stepwise optimization technique that returns the order of parameters concerning the minimum Akaike information criterion (AIC). The AIC value is a statistical measure for how well a model captures the variance in a dataset. The input to PMDARIMA is set to 30 days of training data prior to the test set(2020-08-16 00:00 to 2020-08-22 23:00 for summer and 2020-02-15 00:00 to 2020-02-21 23:00 for winter). The exogenous covariates used for each Load series are the day-ahead temperature forecasts after it has been shown that they have the highest correlation with the Load variation, see figure 3.14. We use the first three temperature parameters from the covariate dataset that has no missing values for test period. With known orders of each sARIMAX model's parameters, we then tune the parameters by fitting the model in a rolling forecast manner on two weeks of data prior to the forecast window of the day-ahead energy load. The quarantine period's missing values are substituted by a previous forecast. The respective sARIMAX models' tuning and inference take about 4 minutes for the 28 series.

4.2 Baseline

In this work, we use two models to compare the performance of DeepAR, the first is sARIMAX, and the second is an LSTM + XG boost hybrid, from now on denoted Baseline. The Baseline is provided by the client Vitec Energy and is used internally as a benchmark model. It is a recognized method in forecast modeling of electricity load, with several studies that prove its performance, [40], [41], [42]. Therefore, we do not give a detailed explanation of the methodology used in the implementation of the Baseline but refer to the provided articles.

4.3 DeepAR

The provided covariates represent weather forecasts for grid areas within electricity area two. Some of these grid areas border on each other. Since the weather forecast and the grid areas have differently defined geographical regions, cases may arise where the same weather forecast overlap two grid areas, i.e., some of the weather parameters in the dataset are duplicates. To use all of the covariates during the training process can be made an analogy with the introduction of noise to the training process since not all covariates contribute to a predictive explanatory degree. See figure 3.14 where the Wind speed displays a relatively low correlation with the electricity load. Initial tests show that when all covariates are available for DeepAR, the training converges, however unstable, thus indicating that more epochs are required. DeepAR can handle noisy data, but we choose to exclude the wind speed as their use entails longer training time. To further reduce the covariates' dimension is only one parameter of the Solar angle used because the difference between the forecasts is a slight change in phase, see figure 3.11. In addition to weather parameters, we add date features such as time of day, day of the week, age, and month of the year. We exclude covariates that exceed 80% of missing values. The feature selection reduces the covariates' dimension from 131 to 61, consisting of 28 Temperature and Global radiation parameters, 1 Solar Angle, and the date features. Figure 4.17 displays how DeepAR interprets the presence of the weather parameters during a training session. The model has been trained with the same parameter configuration two times, using the same seed, with and without weather parameters, and evaluated on two held out validation sets, one for winter and one for summer. Note that weighted batch sampling is active in both cases.

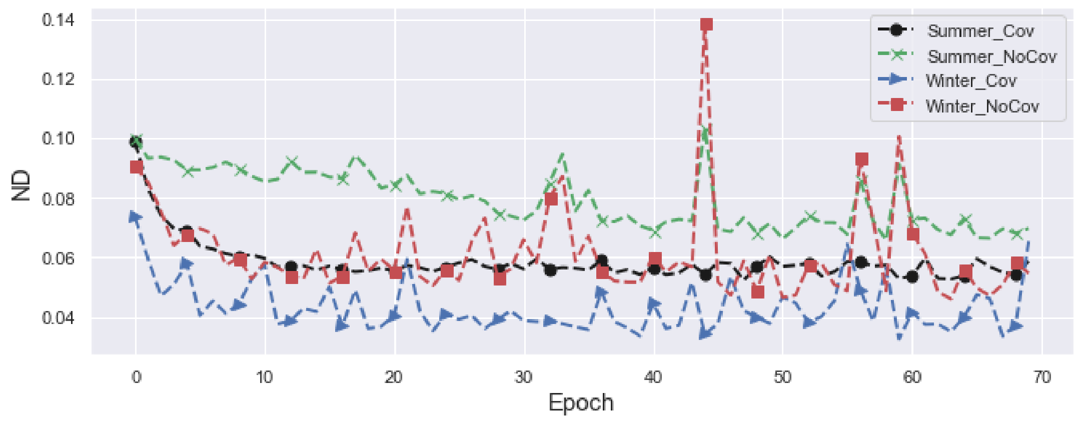


FIGURE 4.17: Training session to compare the impact of weather parameters on the predictive power of electricity load. The trained model is evaluated on two held out validation sets, one in winter and one in summer. When weather parameters are present, they contribute to a more stable convergence and results in a lower percentage deviation in both cases.

The hyperparameter tuning is performed with the python package Optuna. Optuna is a hyperparameter optimization framework where one defines an objective function to minimize with respect to defined hyperparameter space. In this case, we use ND as our objective function. The optimization algorithm utilized is the Multivariate Tree-structured Parzen Estimator (TPE) algorithm, with ten independent samples as initial search space, see [43] for more details. During the search, the covariates available are temperature forecasts (excluded duplicates) and time-explaining parameters such as Day of the Month, Month of Year, Age, and Hour of Day. The training set's date range is set to 90% of the available data and validated on the remaining 10%, and 50 epochs are used to ensure that the configurations converge. Note that the test sets are excluded from the training and validation sets. The rand points and step size of chosen hyperparameters are stated in table 4.1. The learning rate is fixed to 1e-3 and the batch size to 32 since experiments displayed that the values provide stable convergence.

TABLE 4.1: Rand points of search space for hyperparameters tuning

Parameter	**Lower bound**	**Upper bound**	**Step size**
lstm_layers	2	4	1
lstm_hidden_dim	30	70	10
embedding_dim	18	25	1
lstm_dropout	0.1	0.5	0.1

The best performing hyperparameter configuration can be seen in table 6.

We evaluate DeepAR on two test sets, one week in the summer and one week in the winter due to that electricity load has noticeable property differences depending on the season, see section 3.2. The winter set is held out from the training data and substituted by the previous week. We utilize the log-likelihood function, under the Gaussian assumption, as the loss function during training, see equation 2.13 and Adam as the optimizer. When we use the log-likelihood to train the model, there is no guarantee that the obtained validation likelihood yields the best forecast. Therefore we extract the best model according to how it performs in the term ND on the validation set, following [44]. The training length constitutes all available data except test sets and the validation set, consisting of two weeks of data prior to the summer test set. Missing target values are set to zero and masked during training. A softmax activation function is applied to the estimate of σ, restricting the standard deviation to positive values. Target data are normalized with a scale factor v_i, according to equation 3.18, and the covariates by Z-score, equation 3.17. We use the weighted batch sampler explained in 3.3.1 to sample the training data. The choice of learning speed and batch size is a trade-off between stability and training speed. A lower learning rate and batch size would lead to a more stable convergence but increased training time. We utilize the recurrent dropout explained in section 2.7 to prevent overfitting. We train two sets of DeepAR, one with Q_1 active and one with Q_3 active. Figure 23 in appendix A displays the respective training progression. A full overview of used hyperparameters can be seen in table 6.

4.3.1 Hardware and execution time

DeepAR architecture builds upon stacked layers of LSTM, which makes it challenging to parallelize during training due to their sequential structure. A training session of 70 epochs, with hyperparameter settings stated in table 6, takes approximately 10 hours when trained on the CPU stated in 3.4. The computation time in the inference phase is dependent on the number of sample traces and takes approximately 4-6 seconds to compute for 200 traces, elapsed time is with loading and prepossessing. Attempts to train DeepAR on a GPU were unsuccessful. The probable bottleneck is that each transfer of data to the GPU is limited by the transfer rate between the CPU and GPU memories at each time step, which means that a large fraction of the GPU's computing power is idle while waiting to receive new data.

4.3.2 Feature importance

To evaluate how DeepAR interprets the presented covariates, we implement a feature importance algorithm 1. As input, the algorithm takes a fully trained model. A trained model has created potential temporal connections between the covariates it considers to significantly impact the electricity load. Thus, if we break these connections, the model will give a lower degree of prediction accuracy. The feature importance algorithm is inspired by [45]. In this setting with a temporal dependent model, we restrict the permuted data set to the test set's conditioning range.

```
Algorithm 1: Feature importance
Input: Trained model: $\boldsymbol{h_{i,t}} = h(h_{i,t-1}, z_{i,t-1}, x_{k,t}, \Theta)$
Output: Relative change in likelihood loss $\tilde{\epsilon}(k)$ for covariate $k$
Data: Test set: conditioning range $z_{i,t}$ and covariates $x_{k,t}$, where
        $t = \{1, 2, 3, ..., t_0 - 1\}$
for $n$ in num samples do
    Original loss: $\mathcal{L}_o = \ell(z_{i,t}|\theta(\boldsymbol{h_{i,t}})$
    for $k$ in num covariates do
        Randomly permute covariate $x_{k,t}$ in each conditioning range and draw a
         sample: $\tilde{x}_{k,t} = \{\tilde{x}_{k,1}, \tilde{x}_{k,2}, ..., \tilde{x}_{k,t}\}$
        Permutation loss: $\mathcal{L}_k = \ell(z_{i,t}|\theta(h(h_{i,t-1}, z_{i,t-1}, \tilde{x}_{k,t})$
    Relative change: $\epsilon(n, k) = |\frac{\mathcal{L}_o - \mathcal{L}_k}{\mathcal{L}_o}| \cdot 100$
Return sorted average relative change: $\tilde{\epsilon}(k) = sort(\frac{\sum_1^n \epsilon(k)}{n})$
```

5 Result

This chapter will present the obtained result from the evaluation methods stated in 3.5, on the electricity load dataset explored in section 3.2. We use the definition of DeepAR in section 2.6, with the hyperparameter configuration that minimized ND on the validation set and trained according to the procedure in section 4.3.

We evaluate DeepAR against two benchmark models, the sARIMAX and the Baseline for quarantine periods Q_1, Q_2, and Q_3 during the summer and winter seasons. As proof of the concept, we evaluate the case with the introduced outlier on a 24-hour forecast on the last day of the summer test set. Neither test set contains any known holidays. Note that we use the same weights to evaluate Q_1 and Q_2 for both seasons. Table 5.2 displays obtained results of day-ahead forecasts during the summer season. Of the compared models, do DeepAR yield the best performance in terms of evaluation metrics in all of the compared cases.

Table 5.2: Result for summer season.

Test	Q_1		Q_2		Q_3		Outlier	
Metrics	ND	$RMSE$	ND	$RMSE$	ND	RMSE	ND	$RMSE$
sARIMAX	3.8%	0.69	5.5%	0.91	-	-	6.4%	1.33
DeepAR	3.3%	0.60	4.2%	0.71	4.3%	0.72	2.5%	0.47
Baseline	5.4%	0.91	4.9%	0.80	6.2%	1.04	-	-

Table 5.3 displays obtained results for day-ahead forecasts for a week during the winter season, and of the compared models do DeepAR deliver the lowest ND and $RMSE$. DeepAR should deliver better forecasts for the summer than during the winter due to the stationary level in the summer. However, a comparison shows that performance is worse during the summer. The probable origin of the differences is that the weighted sampler is active during training and the difference in the annual distribution of the number of training windows between summer and winter periods. In the exploratory analysis, we define the summer as 5/12 of the yearly period, so a shorter period combined with a low scale means that we sample training windows in the summer to a lesser extent.

TABLE 5.3: Result for the winter season.

Test	Q_1		Q_2		Q_3	
Metrics	*ND*	*RMSE*	*ND*	*RMSE*	*ND*	RMSE
sARIMAX	6.2%	2.58	16.3%	6.58	-	-
DeepAR	2.5%	0.98	3.8%	1.47	3.3%	1.29
Baseline	4.4%	1.73	4.6%	1.96	4.3%	1.69

From table 5.2 and 5.3, we can see that the forecasts are less accurate when we mask values in the condition range, which is logical as when we mask values, we also remove valuable information. However, DeepAR still manages to capture the underlying dependencies needed to provide satisfactory accuracy for Q_2 and Q_3. The sARIMAX model yields the worst performance during the winter but is acceptable during the summer. The linear structure of sARIMAX does not appear to capture the dependence needed to explain level shifts during the winter season, while both machine learning models excel at this task. The findings of the hyperparameter tuning confirms that DeepAR is adaptable to new datasets with relatively few hyperparameter changes. The main difference between our configuration and the proposed one is that we use 2 LSTM layers and 60 neurons in each, as opposed to 3 layers and 40 neurons [12]. The use of 2 layers, compared to 3, lowered the training time per epoch by approximately 60 seconds. The Baseline has the longest training time, with 14 hours, while DeepAR has 10 hours. A comparison of inference computation time shows that DeepAR is by far the fastest with 6 seconds against the Baseline with 2 minutes and sARIMAX, which computes in 4 minutes.

Figure 5.18 below shows how DeepAR handles the case of an extreme value in the conditioning range. From the figure, we can conclude that the outlier does not significantly affect the forecasting ability. It is possible that the confidence interval is affected but it has not been explored.

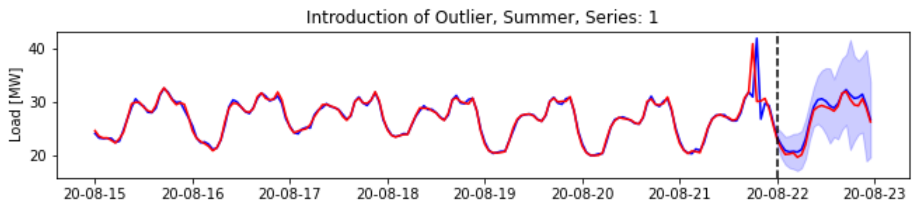


FIGURE 5.18: Example of how DeepAR handles temporal outliers in the conditioning range. The artificial outlier is introduced 6 hours prior to the forecast window and is set to be 3 standard deviations from the conditioning range's mean value. The introduction of an outlier does not entail any clear divergences in the forecast range.

Figure 5.19 illustrates how DeepAR handles missing values in the conditioning range for Q_3. In the figure we can observe how the Encoder receives the missing values in the target sequence and interpolates values forward using one-step forecasts up to time step 2020-02-22 23:00:00. The bottom part of the figure is a sample of three weather parameters. Load series 1 displays an increase in amplitude around 21 of February. Simultaneously, all of the observed weather parameters display a notable decrease in quantity during the same period. Note that this is not an evidence of how DeepAR has interpreted the covariates, but more an indication that there is a connection between the target series and covariates.

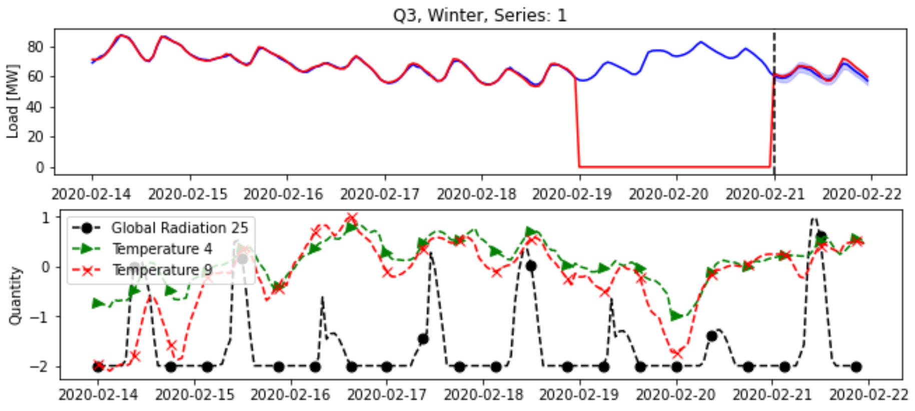


FIGURE 5.19: Day-ahead forecast of Quarantine period 3 for Load series 1 during winter. Target values in the forecast range are hidden for DeepAR and displayed here to illustrate the forecast accuracy and level shift during the winter. One can observe how the encoder interpolates the missing values forward and how the network interprets previous values and presented covariates. The bottom part of the figure displays a sample of presented weather parameters. The quantities of the weather parameters have been normalized and shifted in the figure for visual purposes.

Figure 5.20 shows a sample of forecasts for Q_1 during the summer. The three figures on the left show high forecast accuracy and those on the right low. We can see that the model well captures the daily variations for the left graphs, while the fit for Series 13 and 6 is of lower quality. A clear difference between the series on the left and right is the magnitude. Note the extreme value on August 18 for Load series 4. The outlier's origin is unclear and can not be explained here without domain knowledge of the data. However, the outlier does not seem to affect the median values of the next coming day's forecasts but does seem to influence the enlargement of corresponding confidence intervals.

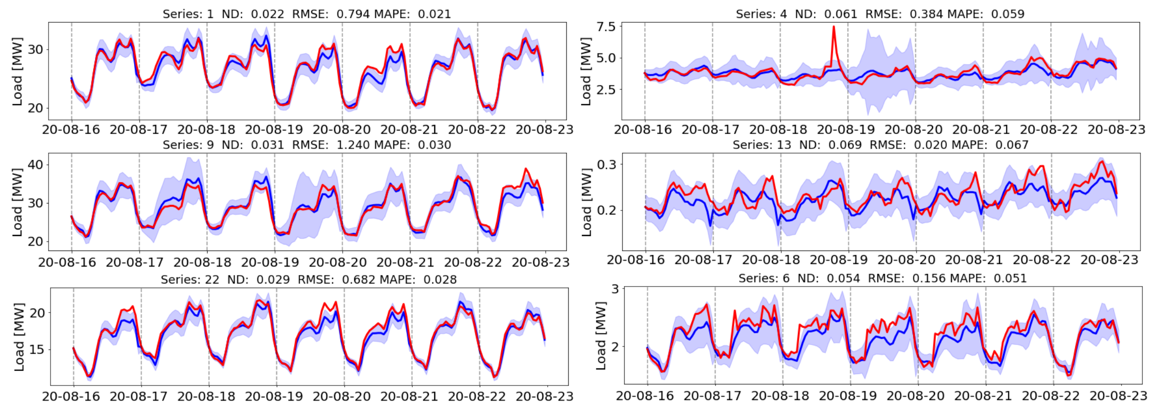


FIGURE 5.20: Sample of high versus load forecast accuracy for Q_1 during the test week in summer. The three figures on the left display high accuracy forecasts, while those on the left display low accuracy. One can observe that the main difference between the left and right graphs is the magnitude of the series, indicating that the weighted sampler works as intended. DeepAR is also generating significant smaller confidence intervals for the graphs on the left.

Figure 5.21 shows a sample of high and low forecasts for Q_3 during winter. The three graphs on the left show high forecast accuracy and those on the right low. We can see that the model well captures the daily seasonality for the left graphs. The figure shows that DeepAR captures the level shifts that appear around 21 of February for all series except 7 and 13.

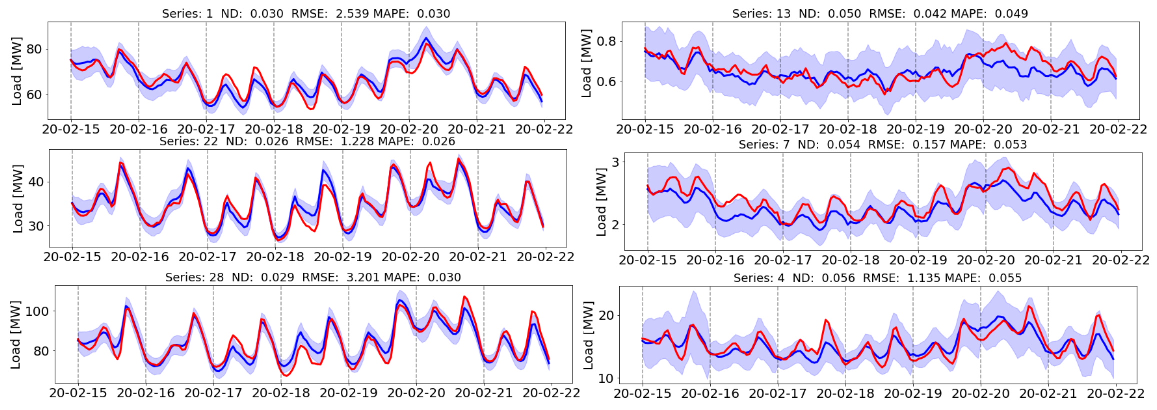


FIGURE 5.21: Sample of high versus load forecast accuracy for Q_3 during the test week in winter. The three figures on the left display high accuracy forecasts, while those on the left display low accuracy. One can observe that the main difference between the left and right graphs is the magnitude of the series, indicating that the weighted sampler works as intended. DeepAR is also generating significant smaller confidence intervals for the graphs on the left.

According to DeepAR is Hour of the day the most crucial covariate as seen in figure 5.22, which confirms Salinas et al.'s [12] statement that DeepAR learns the hourly seasonal dependence within the target series from covariates. The figure also shows a significant connection between the day of the week and the electricity load, which shows a clear underlying weekly seasonality within the target series. Interestingly, global radiation seems to be of higher importance than temperature. The feature importance is computed for Q_1 during the test set in winter.

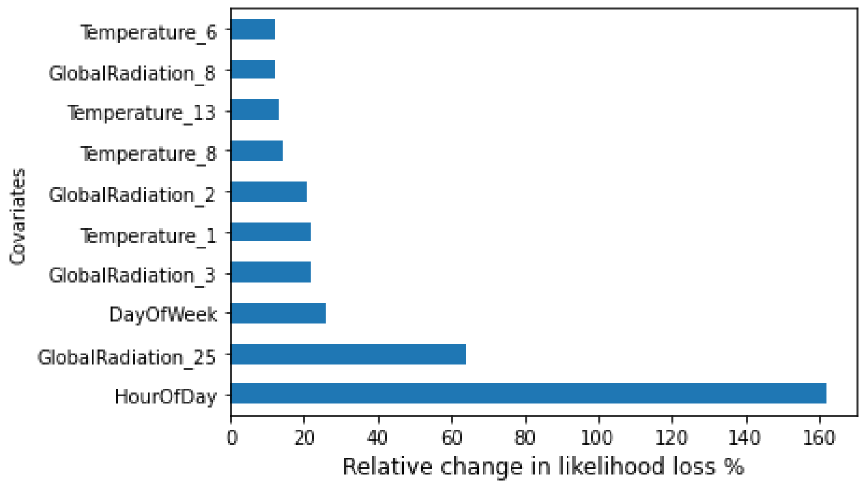


FIGURE 5.22: Feature importance of presented covariates. The feature importance is computed with the test set during winter. The figure displays that Hour of day is the most important feature and that global radiation is more important than temperature.

6 Discussion

The obtained performance, shown in table 5.2 and 5.3 shows the advantages of DeepAR, and the strength of multivariate models that creates a global model to extract interrelationships and complex correlations from covariates. During the project's execution, we discovered that if we train DeepAR without a quarantine period, the network does not yield reliable results when evaluating cases with a quarantine period. One theory is that the network focuses more on learning the series' temporal patterns (due to the autoregressive nature of DeepAR) and fails to capture dependencies that may explain level shifts depending on presented covariates. For example, the temperature drops dramatically, leading to a rise in electricity load. This phenomenon occurs mainly during the evaluation of the winter season. So the flexibility that the autoregressive Encoder possesses, which can adapt to situations where data is missing, is somewhat lost as one still has to consider the use of DeepAR from case to case, depending on the insertion rate of data. One can argue that the correct way to handle Q_3 would be to train the model to output a 72h forecast and use the last 24 hours as the day-ahead forecast. However, then the problem remains that DeepAR does not capture the various dependencies required to explain level shifts. We can also observe that the linear structure of sARIMAX yields reliable results for time series where weather parameters have less impact.

A factor that had a significant impact on the result for the winter was the weighted sampler. Suppose one considers the scale difference between summer and winter for Load series 1, see figure 5.20 and 5.21. In that case, there is a higher probability that we sample a window during the winter than during the summer, resulting in better performance and faster convergence during training. The weighted sampler can be considered a powerful tool in forecast modeling of time series with substantial amplitude variations that depends on the annual season. One could further explore applying a non-affine transformation to the normalization scales v_i to increase the probability of sampling a window in the middle range, i.e., the summer periods.

The feature importance in figure 5.22 shows that DeepAR considers that global radiation is more influential than temperature, which contradicts the correlation analysis in section 3.2 and the findings in [46]. A possible explanation is that algorithm 1 permutes only one set of covariates, which could introduce bias. We might obtain different importance if we chose another conditioning range. Another reason may be structural differences between temperature and global radiation. Global radiation has more apparent daily variations than temperature, see figure 5.19, so when we permute global radiation, we change the structure to a greater degree, which has a more notable effect on interpretation. Therefore, the importance should not be considered entirely true but instead to be interpreted as an indication of what DeepAR considers essential. Note that we do not consider collaborative relationships between the covariates. For a more detailed explanation, we could instead consider a gradient-based approach [47].

Another factor that may have influenced the result is that many of the given weather parameters contained large amounts of missing values. Over 60% of the values were missing in 51% of the weather parameters. We chose only to exclude missing values above 80%, which may have contributed to slower and more unstable convergence. The inability to use the GPU during training limited the number of hyperparameter configurations tested, especially the dimension of the embedding vector. The embedding vector greatly affects the number of parameters in the model, which delimits training time but is an important parameter to explore as the dimension is a part of how the model captures complex patterns. If one were to increase the number of target series, one could also increase the batch size, thus increasing the ability to parallelize during training. The comparison of inference time shows that DeepAR (with data loading and preparation) is by far, the shortest time with 6 seconds, compared to the second shortest, 2 minutes. This result shows a multivariate model's effectiveness in a production scenario and is an essential factor to consider when choosing a model.

7 Conclusion

This thesis has evaluated the possibilities of applying an Deep Autoregressive Neural Network (DeepAR) in a forecasting scenario of electricity load where forecasted weather parameters are used as exogenous covariates. This thesis's findings shows that DeepAR achieves high accuracy when forecasting electricity load in areas with annual variations and that it handles cases where complex relationships between the covariates and target series are unknown. The result shows that the network is robust against potential deviations in input data and that DeepAR surpassed both benchmark models for all tested cases. In the ideal test scenario, Q_1, where weather parameters had the most significant impact (winter), do DeepAR achieve an Normalized Deviation (ND) of 2.5%, compared to the second-best model, the Baseline, with an ND of 4.4%. As a recommendation, we can conclude that DeepAR can be used as an aid for electricity suppliers to reduce the error between supply and demand of electricity load. Further research investigates the possibilities of implementing attention layers to the model and compare the current performance of DeepAR to alternative multivariate direct Deep learning models, such as Deep factors [48] and Temporal fusion transformer [49], which have proven to surpass DeepAR on benchmark data sets.

Appendix

Table 4: sARIMAX parameters for summer season

Series	**(p,d,q)s(P,D,Q)f**	**Series**	**(p,d,q)s(P,D,Q)f**
1	(2,1,0)(2,1,0)[24]	15	(1,1,2)(0,1,1)[24]
2	(1,1,1)(2,1,0)[24]	16	(1,1,0)(1,1,0)[24]
3	(0,1,0)(0,1,2)[24]	17	(1,1,0)(0,1,1)[24]
4	(2,1,1)(2,1,1)[24]	18	(2,1,3)(1,1,1)[24]
5	(1,1,0)(0,1,2)[24]	19	(0,1,1)(1,1,1)[24]
6	(0,1,1)(0,1,1)[24]	20	(1,1,1)(1,1,0)[24]
7	(3,1,0)(1,1,1)[24]	21	(3,1,0)(0,1,1)[24]
8	(2,1,0)(2,1,0)[24]	22	(3,1,0)(0,1,2)[24]
9	(3,1,0)(0,1,2)[24]	23	(1,1,1)(2,1,0)[24]
10	(1,1,0)(2,1,1)[24]	24	(2,1,0)(0,1,1)[24]
11	(1,1,0)(0,1,1)[24]	25	(1,1,0)(2,1,0)[24]
12	(2,1,1)(1,1,2)[24]	26	(0,1,0)(0,1,1)[24]
13	(1,1,2)(0,1,1)[24]	27	(1,1,0)(2,1,0)[24]
14	(3,1,1)(2,1,0)[24]	28	(3,1,2)(2,1,0)[24]

Table 5: sARIMAX parameters for winter season

Series	**(p,d,q)s(P,D,Q)f**	**Series**	**(p,d,q)s(P,D,Q)f**
1	(2,1,0)(0,1,1)[24]	15	(2,1,1)(2,1,1)[24]
2	(3,1,0)(0,1,1)[24]	16	(2,1,2)(2,1,0)[24]
3	(2,1,0)(2,1,2)[24]	17	(1,1,1)(0,1,1)[24]
4	(3,1,0)(0,1,1)[24]	18	(2,1,2)(0,1,1)[24]
5	(1,1,1)(2,1,1)[24]	19	(3,1,0)(0,1,1)[24]
6	(1,1,0)(2,1,1)[24]	20	(2,1,0)(2,1,0)[24]
7	(1,1,3)(0,1,1)[24]	21	(3,1,0)(2,1,0)[24]
8	(1,1,2)(2,1,0)[24]	22	(2,1,0)(0,1,1)[24]
9	(2,1,0)(2,1,0)[24]	23	(2,1,0)(0,1,1)[24]
10	(0,1,1)(2,1,0)[24]	24	(3,1,1)(2,1,0)[24]
11	(3,1,3)(2,1,0)[24]	25	(1,1,1)(2,1,0)[24]
12	(3,1,0)(0,1,1)[24]	26	(1,1,0)(1,1,1)[24]
13	(2,1,0)(2,1,2)[24]	27	(0,1,1)(2,1,0)[24]
14	(2,1,1)(0,1,1)[24]	28	(3,1,0)(0,1,1)[24]

Table 6: Configuration of hyperparameters used to evaluate DeepAR

Hyperparameters			
LSTM Units	60	Dropout	0.2
LSTM Layers	2	Embedding dim	20
Learning Rate	1e-3	Conditioning range	168
Batch Size	32	Covariate dim	61
Epochs	70	Inference traces	200

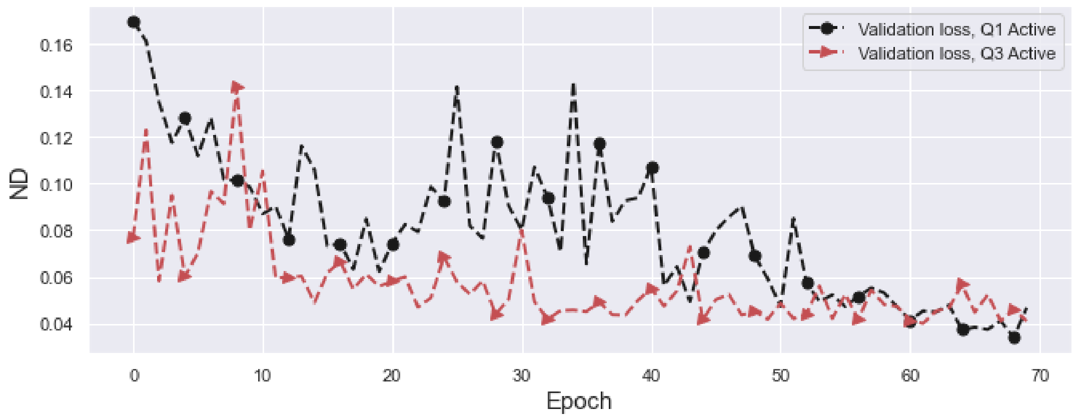


Figure 23: The figure shows the validation loss for two training sessions for DeepAR with Q_1 and Q_3. We extract the best model at the epoch that yields the lowest ND on the validation set. With Q_3 active do it occur at epoch 61 and at Q_1 at epoch 69.

Bibliography

[1] V. Lizunkov, E. Politsinskaya, E. Malushko, A. Kindaev, and M. Minin, "Population of the world and regions as the principal energy consumer," *International Journal of Energy Economics and Policy*, vol. 8, no. 3, p. 9, 2018. [Online]. Available: http://zbw.eu/econis-archiv/bitstream/11159/2120/1/1028134991.pdf.

[2] T. Ahmad and D. Zhang, "A critical review of comparative global historical energy consumption and future demand: The story told so far," *Energy Reports*, vol. 6, pp. 1973–1991, Nov. 1, 2020, Publisher: Elsevier, ISSN: 2352-4847. DOI: 10.1016/j.egyr.2020.07.020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352484720312385 (visited on 01/12/2021).

[3] I. Hadjipaschalis, A. Poullikkas, and V. Efthimiou, "Overview of current and future energy storage technologies for electric power applications," *Renewable and Sustainable Energy Reviews*, vol. 13, no. 6, pp. 1513–1522, Aug. 2009, ISSN: 1364-0321. DOI: 10.1016/j.rser.2008.09.028. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032108001664 (visited on 09/24/2020).

[4] *Drift och elmarknad*. Svenska Kraftnät. [Online]. Available: https://www.svk.se/drift-av-transmissionsnatet/drift-och-elmarknad/ (visited on 09/23/2020).

[5] *Balansansvar*. Svenska Kraftnät. [Online]. Available: https://www.svk.se/aktorsportalen/elmarknad/balansansvar/ (visited on 09/23/2020).

[6] N. Elamin and M. Fukushige, "Modeling and forecasting hourly electricity demand by SARIMAX with interactions," *Energy*, vol. 165, pp. 257–268, Dec. 2018, ISSN: 03605442. DOI: 10.1016/j.energy.2018.09.157. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360544218319297 (visited on 12/26/2020).

[7] A. Yang, W. Li, and X. Yang, "Short-term electricity load forecasting based on feature selection and least squares support vector machines," *Knowledge-Based Systems*, vol. 163, pp. 159–173, Jan. 2019, ISSN: 09507051. DOI: 10.1016/j.knosys.2018.08.027. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950705118304234 (visited on 12/26/2020).

[8] H. Zheng, J. Yuan, and L. Chen, "Short-term load forecasting using EMD-LSTM neural networks with a xgboost algorithm for feature importance evaluation," *Energies*, vol. 10, no. 8, p. 1168, Aug. 2017, Number: 8 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/en10081168. [Online]. Available: https://www.mdpi.com/1996-1073/10/8/1168 (visited on 12/26/2020).

[9] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, Aug. 21, 2018, ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1718942115. arXiv: 1707.02568. [Online]. Available: http://arxiv.org/abs/1707.02568 (visited on 12/18/2020).

[10] B. Huang, C.-L. Lin, W. Chen, C.-F. Juang, and X. Wu, "Signal frequency estimation based on RNN," in *2020 Chinese Control And Decision Conference (CCDC)*, ISSN: 1948-9447, Aug. 2020, pp. 2030–2034. DOI: 10.1109/CCDC49329.2020.9164504.

[11] C. Fan, Y. Sun, Y. Zhao, M. Song, and J. Wang, "Deep learning-based feature engineering methods for improved building energy prediction," *Applied Energy*, vol. 240, pp. 35–45, Apr. 2019, ISSN: 03062619. DOI: 10.1016/j.apenergy.2019.02.052. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0306261919303496 (visited on 01/03/2021).

[12] D. Salinas, V. Flunkert, and J. Gasthaus, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *arXiv:1704.04110 [cs, stat]*, Feb. 2019. [Online]. Available: http://arxiv.org/abs/1704.04110 (visited on 09/17/2020).

[13] F. Wang, M. Li, Y. Mei, and W. Li, "Time series data mining: A case study with big data analytics approach," *IEEE Access*, vol. 8, pp. 14 322–14 328, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2966553.

[14] P. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, 3rd ed. Springer texts in Statistics, Apr. 2016.

[15] M. M. Fathi, A. G. Awadallah, A. M. Abdelbaki, and M. Haggag, "A new budyko framework extension using time series SARIMAX model," *Journal of Hydrology*, vol. 570, pp. 827–838, Mar. 1, 2019, ISSN: 0022-1694. DOI: 10.1016/j.jhydrol.2019.01.037. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022169419301040 (visited on 11/11/2020).

[16] I. Ghalehkhondabi, E. Ardjmand, G. R. Weckman, and W. A. Young, "An overview of energy demand forecasting methods published in 2005–2015," *Energy Systems*, vol. 8, no. 2, pp. 411–447, May 2017, ISSN: 1868-3967, 1868-3975. DOI: 10.1007/s12667-016-0203-y. [Online]. Available: http://link.springer.com/10.1007/s12667-016-0203-y (visited on 09/21/2020).

[17] R. H. Hariri, E. M. Fredericks, and K. M. Bowers, "Uncertainty in big data analytics: Survey, opportunities, and challenges," *Journal of Big Data*, vol. 6, no. 1, p. 44, Jun. 2019, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0206-3. [Online]. Available: https://doi.org/10.1186/s40537-019-0206-3 (visited on 09/26/2020).

[18] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money, "Big data: Issues and challenges moving forward," in *2013 46th Hawaii International Conference on System Sciences*, Jan. 2013, pp. 995–1004. DOI: 10.1109/HICSS.2013.645.

[19] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long- and short-term temporal patterns with deep neural networks," *arXiv:1703.07015 [cs]*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1703.07015 (visited on 09/21/2020).

[20] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," *PLOS ONE*, vol. 13, no. 3, e0194889, Mar. 2018, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0194889. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889 (visited on 09/24/2020).

[21] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006, ISBN: 978-0-387-31073-2.

[22] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, p. 386, Publisher: US: American Psychological Association, ISSN: 1939-1471. DOI: 10.1037/h0042519. [Online]. Available: https://psycnet-apa-org.proxy.ub.umu.se/fulltext/1959-09865-001.pdf (visited on 11/20/2020).

[23] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Jan. 15, 1982. DOI: https://doi.org/10.1073/pnas.79.8.2554.

[24] A. Goh, "Back-propagation neural networks for modeling complex systems," *Artificial Intelligence in Engineering*, vol. 9, no. 3, pp. 143–151, Jan. 1995, ISSN: 09541810. DOI: 10.1016/0954-1810(94)00011-S. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/095418109400011S (visited on 09/29/2020).

[25] A. Géron, "Hands-on machine learning with scikit-learn, keras, and TensorFlow," p. 1150,

[26] H. Weytjens, E. Lohmann, and M. Kleinsteuber, "Cash flow prediction: MLP and LSTM compared to ARIMA and prophet," *Electronic Commerce Research*, Jul. 2019, ISSN: 1389-5753, 1572-9362. DOI: 10.1007/s10660-019-09362-7. [Online]. Available: http://link.springer.com/10.1007/s10660-019-09362-7 (visited on 10/02/2020).

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, p. 1735, Nov. 1997, ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://proxy.ub.umu.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=9710215021&site=ehost-live&scope=site (visited on 09/29/2020).

[28] S. Varsamopoulos, K. Bertels, and C. G. Almudever, "Designing neural network based decoders for surface codes," *ResearchGate*, p. 13, Dec. 6, 2018. DOI: https://www.researchgate.net/profile/Savvas_Varsamopoulos/publication/329362532_Designing_neural_network_based_decoders_for_surface_codes/links/5c09270da6fdcc494fde1c08/Designing-neural-network-based-decoders-for-surface-codes.pdf.

[29] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, May 21, 2019, Publisher: MIT Press, ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. [Online]. Available: https://doi.org/10.1162/neco_a_01199 (visited on 11/16/2020).

[30] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A comparison of ARIMA and LSTM in forecasting time series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018, pp. 1394–1401. DOI: 10.1109/ICMLA.2018.00227.

[31] S. Bouktif, A. Fiaz, A. Ouni, and M. A. Serhani, "Optimal deep learning LSTM model for electric load forecasting using feature selection and genetic algorithm: Comparison with machine learning approaches †," *Energies*, vol. 11, no. 7, p. 1636, Jul. 2018. DOI: 10.3390/en11071636. [Online]. Available: https://www.mdpi.com/1996-1073/11/7/1636 (visited on 09/30/2020).

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv:1409.3215 [cs]*, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1409.3215 (visited on 09/23/2020).

[33] S. Du, T. Li, Y. Yang, and S.-J. Horng, "Multivariate time series forecasting via attention-based encoder–decoder framework," *Neurocomputing*, vol. 388, pp. 269–279, May 2020, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.12.118. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231220300606 (visited on 09/29/2020).

[34] I.-F. Kao, Y. Zhou, L.-C. Chang, and F.-J. Chang, "Exploring a long short-term memory based encoder-decoder framework for multi-step-ahead flood forecasting," *Journal of Hydrology*, vol. 583, p. 124 631, Apr. 2020, ISSN: 0022-1694. DOI: 10.1016/j.jhydrol.2020.124631. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022169420300913 (visited on 09/30/2020).

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 2014, p. 30, DOI: https://dl.acm.org/doi/pdf/10.5555/2627435.2670313.

[36] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv:1409.2329 [cs]*, Feb. 19, 2015. arXiv: 1409.2329. [Online]. Available: http://arxiv.org/abs/1409.2329 (visited on 11/20/2020).

[37] (). "PyTorch," [Online]. Available: https://www.pytorch.org (visited on 01/09/2021).

[38] Y. Zhang, Q. Jiang, and X. Ma. (). "GitHub - zhykoties/TimeSeries: Implementation of deep learning models for time series in PyTorch.," [Online]. Available: https://github.com/zhykoties/TimeSeries (visited on 12/14/2020).

[39] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *arXiv:2002.04236 [cs, stat]*, Feb. 11, 2020. arXiv: 2002.04236. [Online]. Available: http://arxiv.org/abs/2002.04236 (visited on 12/04/2020).

[40] C. Li, Z. Chen, J. Liu, D. Li, X. Gao, F. Di, L. Li, and X. Ji, "Power load forecasting based on the combined model of LSTM and XGBoost," in *Proceedings of the 2019 the International Conference on Pattern Recognition and Artificial Intelligence - PRAI '19*, Wenzhou, China: ACM Press, 2019, pp. 46–51, ISBN: 978-1-4503-7231-2. DOI: 10.1145/3357777.3357792. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3357777.3357792 (visited on 01/09/2021).

[41] M. Fan, Y. Hu, X. Zhang, H. Yin, Q. Yang, and L. Fan, "Short-term load forecasting for distribution network using decomposition with ensemble prediction," in *2019 Chinese Automation Congress (CAC)*, ISSN: 2688-0938, Nov. 2019, pp. 152–157. DOI: 10.1109/CAC48633.2019.8997169.

[42] F. Fahiman, S. M. Erfani, and C. Leckie, “Robust and accurate short-term load forecasting: A cluster oriented ensemble learning approach,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, ISSN: 2161-4407, Jul. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852460.

[43] (). “Optuna: A hyperparameter optimization framework — optuna 2.3.0 documentation,” [Online]. Available: https://optuna.readthedocs.io/en/stable/index.html (visited on 11/24/2020).

[44] K. Astrom, “Maximum likelihood and prediction error methods,” *IFAC Proceedings Volumes*, vol. 12, no. 8, pp. 551–574, Sep. 1979, ISSN: 14746670. DOI: 10.1016/S1474-6670(17)53976-2. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474667017539762 (visited on 12/17/2020).

[45] A. Fisher, C. Rudin, and F. Dominici, “All models are wrong, but many are useful: Learning a variable’s importance by studying an entire class of prediction models simultaneously,” *arXiv:1801.01489 [stat]*, Dec. 23, 2019. arXiv: 1801.01489. [Online]. Available: http://arxiv.org/abs/1801.01489 (visited on 12/01/2020).

[46] S. D. Atalay, G. Calis, G. Kus, and M. Kuru, “Performance analyses of statistical approaches for modeling electricity consumption of a commercial building in france,” *Energy and Buildings*, vol. 195, pp. 82–92, Jul. 15, 2019, ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2019.04.035. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378778818338647 (visited on 01/02/2021).

[47] A. Ghorbani, A. Abid, and J. Zou, “Interpretation of neural networks is fragile,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3681–3688, Jul. 17, 2019, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33013681. [Online]. Available: https://www.aaai.org/ojs/index.php/AAAI/article/view/4252 (visited on 01/02/2021).

[48] Y. Wang, A. Smola, D. C. Maddix, J. Gasthaus, D. Foster, and T. Januschowski, “Deep factors for forecasting,” *arXiv:1905.12417 [cs, stat]*, May 28, 2019. arXiv: 1905.12417. [Online]. Available: http://arxiv.org/abs/1905.12417 (visited on 12/30/2020).

[49] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” *arXiv:1912.09363 [cs, stat]*, Sep. 27, 2020. arXiv: 1912.09363. [Online]. Available: http://arxiv.org/abs/1912.09363 (visited on 12/30/2020).