



Norges miljø- og
biovitenskapelige
universitet

Master's Thesis 2023 30 ECTS

Faculty of Science and Technology

A study of hourly electrical load consumption forecasting approach for industrial buildings using a recurrent neural network architecture

Atif Fazal

Data Science

May 15, 2023

Acknowledgements

The completion of this project marks the end of my study time at NMBU.

I would like to express my sincerest gratitude to my supervisor Tor Kristian Stevik for his advice throughout the work in this project. Thanks to both Heidi Samuelsen Nygård and Stig Ødegaard Ottesen for providing their advice during joint meetings. I would also like to express my gratitude to Habib Ullah for providing constructive feedback regarding my report during the finalization of this project.

Lastly, i thank my family for their continuous support throughout my years at NMBU. I would like to emphasize the support provided to me by my older sister whose encouraging words and advice was helpful.

Oslo, 14.05.2023

Atif Fazal

Abstract

Increased focus on decarbonization involving electrification and usage of variable renewable energy sources impose problems for the power grids. Power grids may suffer from increased disturbances as the demand for electricity increases. To potentially mitigate power supply disturbances, end users can help regulate the power grid by offering their flexibility in exchange for economic incentives. Industrial buildings such as ASKO are adapting and willing to offer their flexibility on the flexibility market. The methodology for developing flexibility bids to be offered on the flexibility market consist of several steps. One of them being accurate electrical load forecasts, these load forecasts are used as baseline estimates when creating flexibility bids. Accurate electrical load forecasts can also contribute with information during planning and power management leading to cost optimization in both operative and maintenance related tasks.

This study investigates a deep learning approach to forecast hourly total electrical load consumption profiles for ASKO facility, specifically a variant of the recurrent neural network called long short-term memory network (LSTM). Three forecast models for time series analysis were developed and evaluated - a single step recurrent neural network with LSTM layers, a day-ahead recurrent neural network with LSTM layers predicting 24 hours into the future and a day-ahead recurrent neural network with LSTM layers predicting 24 hours into the future for one of the buildings assessing model performance and compared against another similar building. The data was provided by ASKO and consisted of total load bought, sold, produced and total electrical consumption for 11 buildings belonging to ASKO. Consumption profiles were available both in aggregated and disaggregated (separate profiles for all buildings) form. Data had to be treated before it could be fed to the model. A framework for tuning the LSTM networks was implemented. The models were subjugated to an extensive hyperparameter tuning process. The best model developed was the single-step recurrent neural network with LSTM layers achieving the R^2 score of 0.94 and the MAE of 104.69. After obtaining the results the model performance was assessed and the models show promising results. However, it has been concluded that the presented models may not provide sufficiently accurate load forecasts and a number of aspects may need further investigation.

Contents

1	Introduction	1
1.1	Research question	3
2	Related work	4
3	Theory	5
3.1	Machine learning	5
3.1.1	Time-series modelling	6
3.1.2	Universal Machine learning workflow	7
3.2	Neural networks	8
3.2.1	Single Neuron model	9
3.2.2	Multilayer Neural Network (MLP)	11
3.2.3	Activation functions	13
3.2.4	Loss functions and Error metrics	16
3.2.5	Recurrent neural network (RNN)	17
3.2.6	Model complexity	20
4	Methodology	23
4.1	Exploratory data analysis	23
4.2	Preprocessing	26
4.2.1	Splitting data	27
4.3	Deep Learning modelling using RNN architecture	28
4.4	Software	36
5	Results	37
5.1	Exploratory data analysis results	37
5.2	Single-step recurrent neural network (RNN) forecast results	40
5.3	Day-ahead recurrent neural network forecast (RNN) results	43
5.4	Day-ahead recurrent neural network forecast (RNN) results on a single building (disaggregated data)	46
6	Discussion	50
6.1	Interpreting the results	50
6.2	Shortcomings of the work in this project	53
6.3	Further work	54
7	Conclusion	56
	References	57

List of Figures

3.1	Illustration of a Single-layer neural network architecture Figure from [17, chapter 12]	9
3.2	Concept of gradient descent. The algorithm starts from with random weights and proceeds to locate the global minimum of the loss function. The process of finding the global minimum can be described as climbing down a hill Figure from [17, chapter 2]	10
3.3	Illustration of a multilayer neural network architecture, consisting of one input layer, one hidden layer and an output layer Figure from [17, chapter 12, p.388].	12
3.4	A selection of widely used activation functions and graph illustrations Figure from [17, chapter 13, p.469].	14
3.5	Illustration of sigmoid function and its derivative Figure from [22]	15
3.6	Illustration of a Recurrent neural network layer and its looping ability, left side closed version while right side unfolded version Figure from [17, chapter 16]	18
3.7	Illustration of the LSTM memory cell structure. The horizontal line on top of the figure represents the flow of information. Forget, input, candidate and output gates are illustrated as the yellow boxes Figure from [17, chapter 16]	19
3.8	Figure showing the cases of a model underfitting left most figure, overfitting right most figure and a model with a good compromise which would be the most ideal case when developing a machine learning model, middle figure. Figure from [17, chapter 3].	21
3.9	Visualisation of training and validation loss curves of a model overfitting characterised by the validation loss increasing after a certain point, loss along the y-axis and number of iterations along the x-axis Figure from [26]	22
4.1	Visualisation of total load consumption, time along the x-axis and load (kw/h) along the y-axis	24
4.2	Visualisation of total load consumption by year, time along the x-axis and load (kw/h) along the y-axis	25
4.3	Better visualisation of total load consumption using 2 months, time along the x-axis and load (kw/h) along the y-axis	25
4.4	Figure showing how input data is split into a training set, validation set and a test set used when training a machine learning model [17, chapter 6]	27
4.5	Summary of best single-step model showing each layer, input shape and the number of parameters in the model	34
4.6	Summary of best day-ahead model showing each layer, output shape and the number of parameters in the model	35
4.7	Summary of best day-ahead model for building 1 showing each layer, output shape and number of parameters	35
5.1	Heatmap showing daily electrical consumption	37
5.2	Heatmap showing hourly electrical consumption	38
5.3	Heatmap showing monthly electrical consumption	39
5.4	Best results of single step RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.	41

5.5	Best results of single step RNN model after tuning, train set predictions plotted against true values zoomed in observing 2 weeks of hourly electrical consumption. Hour along the x-axis and load (kw/h) along the y-axis. . .	41
5.6	Training and validation loss curves of the best performing single step RNN model. Number of epochs along the x-axis and loss along the y-axis . . .	42
5.7	Best results of single step RNN model after tuning, hourly electrical consumption predictions made on test set. Hour along the x-axis and load (kw/h) along the y-axis.	42
5.8	Best results of multi.step RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.	44
5.9	Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values, closer look at prediction plotting 2 weeks of data. Hour along the x-axis and load (kw/h) along the y-axis.	44
5.10	Training and validation loss curves of the best performing day-ahead RNN model, number of epochs along the x-axis and loss along the y-axis . . .	45
5.11	Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions made on test set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.	45
5.12	Best results of day-ahead RNN model after tuning, hourly electrical consumption for building 1 predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis. . . .	47
5.13	Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions for building 1 made on train set plotted against true values (closer look). Hour along the x-axis and load (kw/h) along the y-axis.	47
5.14	Training and validation loss curves of the best performing day-ahead RNN model trained of building 1 dataset, number of epochs along the x-axis and loss along the y-axis	48
5.15	Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions for building 1 made on test set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis. . . .	48
5.16	Best results of day-ahead RNN model after tuning, hourly electrical consumption for building 7 predictions plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.	49

List of Tables

4.1	Time features included	26
4.2	Python libraries used in this project	36
5.1	Best performing single-step RNN model metrics	40
5.2	Best performing day-ahead RNN model metrics	43
5.3	Best performing day-ahead RNN model metrics for building 1	46

1 Introduction

The EU and Norway sharing similar ambitions, work together committed to reaching the goal of being climate-neutral by 2050 an economy with net-zero greenhouse gas emissions [1]. The worlds fossil-based energy consumption constitute 80 % (as of 2021 without including the electricity sector) [2]. Leading to climate change and environmental degradation posing an existential threat to the world. As an initiative to decarbonize the energy system EU declared the European green deal holding the objective of zero greenhouse emission by 2050 at its heart [3]. Electrification is one solution used to mitigate emissions and decarbonizing the energy system, since the efficiency of electrical technologies are generally much higher than fossil-based [4]. Additionally, electrification goes hand in hand with an increase of renewable energy sources thus reducing the greenhouse gas emission. Eurostat defines renewable energy sources as energy sources that replenish themselves naturally [5]. The main types of renewables are: hydropower, geothermal, wind, solar. These renewables are considered clean energy sources. Hence, renewable energy sources plays a critical role in the transition to clean energy and zero emission. According to [6] they are responsible for over one-third of the CO_2 emission reduction between 2020 and 2030 under the net zero emissions by 2050 scenario. The deployment of renewable energy sources has has to expand in order to get on track with the zero emission goal [6].

Electrification leads to higher demand of electricity and increasing the renewable energy sources impose problems for the power grids. Power grids become more vulnerable to potential faults that can occur leading to issues that disturb the supply of electricity [7]. The variable production from renewable energy sources is increasing, doing so may cause more disturbances in the power grid in the form of increased intermittency. In Norway, imbalances are handled by statnett that regulate the transmissions with reserves. A reserve works as a source that offer up- or down- regulation at transmission level if needed [8]. One solution to help regulate the power grid is to make it more "flexible" e.g. through consumer flexibility. End users in the power system can offer their flexibility (by for example reducing their electrical consumption) on the flexibility market for other participants benefit in exchange for economic incentives thus helping to regulate the power grid [9][10].

Climate changes and the increased focus on electrification including new guidelines for emissions require industrial building complexes such as ASKO to adapt. ASKO wants to provide their flexibility on the flexibility market. The methodology for presenting flexibility bids at the flexibility market consist of several steps. One of those steps are accurate load forecast needed in order to create reliable flexibility bids. ASKO facility are much like other industrial facilities, models capable of providing accurate electrical load forecast are of great value and can contributing with information during planning and power management leading to cost optimization in both operative and maintenance related tasks. Accurate load prediction can also provide useful information in decision making for future plans. Predicting load consumption is challenging, such data are complex and are influenced by a range of factors like seasons, weekly variation with higher consumption during working days as opposed to weekends and outside factors such as temperature.

Electrical load forecasting a type of time series, can be approached using various methods. According to [11], the most frequently used time series forecasting methods can be divided into three subcategories: statistical methods, machine learning methods and hybrid models. In traditional approaches, statistical methods are used. These include *Autoregressive models*, *Moving Average models*, *Autoregressive integrated moving average model*. Machine learning methods include both shallow methods such as *Support Vector Machines* and deep learning approaches such as *Artificial neural network models*. Hybrid models are a combination of the different models utilizing the advantages of the models to improve forecasts [11]. Deep learning a subset of machine learning methods are able to solve complex problem. There are several advantages to applying a deep learning approach for forecasting electrical load a few of them are listed in the following [12]:

- able to automatically learn features from data, useful when features are difficult to define
- capable of handling large and complex data extracting useful insights from big data
- able to uncover non-linear relationships in data that would be difficult using traditional methods

For the work in this project the choice of model falls on a variant of recurrent neural network called long short-term memory (LSTM) network. There are two main reasons for

the choice of investigating LSTM networks in this project. Firstly, the choice is influenced by the constant usage of LSTM networks in related work. Secondly, the choice is influenced after reviewing machine learning theory. Deep learning methods such as the recurrent neural networks and LSTM networks are specifically designed to handle sequential data and model long term dependancies between data points. This allows the models to make predictions using based on past information well suited for the problem of forecasting electrical load consumption for ASKO facility explored in this project.

1.1 Research question

To summarize, in accordance with reducing gas emissions leading to an increased focus on electrification requires industrial facilities such as ASKO to adapt. To help the power grids in regulation related tasks and receiving economic incentives in exchange, ASKO wants to provide their flexibility on the flexibility market. Developing flexibility bids consist of several steps and one of those steps are accurate load forecasts needed in order to create reliable flexibility bids. The main goal of this thesis is to evaluate the potential of deep learning approaches to predict electrical load consumption. Specifically exploring a variant of the recurrent neural network called long short-term memory (LSTM) network. For the work in this project the following research questions are addressed:

- To what degree is the LSTM network able to capture the variance and learn the electrical load consumption patterns of an industrial building complex?
- Does regularization techniques such as dropout increase model performance and ability to generalize?

2 Related work

Reviewing literature addressing similar problems shows the constant appearance of deep machine learning models especially the recurrent neural network variant with long short-term memory layers.

[13] presents a recurrent neural network model using Long Short-Term memory (lstm) layers, predicting the electrical load for a planned smart grid in a city. The proposed model is compared against a traditional shallow machine learning technique, support vector machines. Judging by the results presented in [13], the recurrent neural network model with lstm layers outperforms the support vector machines model.

[14] investigates variables that affect the electrical load consumption. In this paper the influence of weather factors on a residential buildings load consumption is studied. [14] implements a recurrent neural network with lstm layers and measure the forecasting ability of the model with and without weather variables. Findings show that adding weather variables had a positive effect, improving the prediction accuracy of a buildings load consumption.

[15] predicts the hourly electrical load consumption of different kinds of buildings implementing a recurrent neural network with lstm layers. In order to evaluate the model performance, multi-layered perceptron, random forest and support vector machines were implemented. The recurrent neural network was shown to predict better compared to the models, achieving lower prediction errors.

A novel approach for predicting electrical load consumption is described in [16]. The work in this paper presents a hybrid model integrating convolutional neural network (cnn) and a lstm network. The cnn part of the model is used to learn the features and data trends while the lstm part of the network is used to model the long term dependancies in the dataset. The hybrid model performance is compared against a recurrent neural network with lstm layers, a radial basis function network and XGboost. Results show that the proposed hybrid cnn-lstm model yield lower error metrics thus outperforming the other models.

3 Theory

3.1 Machine learning

General machine learning

In the modern age of technology, data is one of the resources we have access to in vast amounts both in structured and unstructured format. From this came the existence of learning algorithms in the field of machine learning that convert data into knowledge. Machine learning, a subfield of artificial intelligence utilizes self-learning algorithms to create predictions. Using data and answers to learn the rules, a more efficient approach as opposed to manually derive the rules and build models to analyze massive amounts of data. The field of machine learning can be broken apart into three main branches: *supervised learning*, *unsupervised learning* and *reinforcement learning* [17].

Supervised learning models capture patterns in a dataset that allow us to make predictions on unseen data or future data. This subfield is defined by a models usage of labeled training data, a set of features and the corresponding solution (target) as input to correctly classify or predict future target values. Unsupervised learning handle unlabeled and unstructured data using machine learning algorithms to analyze and cluster datasets. Here the main goal is to discover patterns or data groups extracting meaningful information without knowing the outcome variable. Reinforcement learning is similar to supervised learning but differs with regard to knowing the solution (target). Models learn by trial and error with the help of a reward function providing a measure of reward. For the work in this thesis, we will focus on supervised learning.

Furthermore, supervised learning can be grouped into [17, chapter 1]:

- classification for predicting class labels - the goal is to predict categorical class labels which can be understood as discrete class lables (e.g. email spam detection, where an email can be classified as either "spam" or "not spam")
- regression - is the prediction of a continuous outcomes (e.g. predicting the math test scores of students based on time spent on studying)

The focus of this thesis which is time series forecasting can be regarded as a supervised

regression problem.

There are a range of different supervised shallow machine learning models applicable for solving classification and regression tasks. Examples of shallow machine learning models are linear regression, logistic regression, support vector machines and tree based learning algorithms such as decision trees. The main advantage of shallow methods over deep learning methods lies in the fact that shallow methods are computationally less expensive to train. However, shallow methods may not be able to capture complex patterns in data. In contrast, deep learning methods are able to capture complex patterns and potentially achieve higher accuracy requiring more computational resources [18].

Deep learning methods are based on neural networks and multiple layers are connected to one another as opposed to shallow methods consisting of one layer. The more layers a model has the deeper the model is considered and is where the name "deep" learning comes from. Deep learning methods are capable to learn meaningful representations of the input data automatically by itself while shallow methods are not [19, chapter 1].

3.1.1 Time-series modelling

Time series data is a type of sequential data where each element in the data includes a dimension for time. This means that each element depend on the previous element and affect subsequent elements - it is therefore crucial to keep the order of the elements as randomising the order would result in loss of information. Time series data such as stock prices, voice or speech recordings are a type of sequential data with a dimension for time [17, chapter 16]. There are types of time series data which do not have a time dimension such as text data and DNA sequences. Non-sequential data are considered to be *independent and identically distributed* (IID). This means that for non-sequential data the order in which the data is presented to the model is irrelevant. However, time series data violates the assumption of independence as subsequent data points are often highly correlated.

Time series data can be divided into univariate or multivariate. Univariate time series data consist of historical data of only one variable, such as historical data of a buildings load consumption. In the second case, the dataset may include several variables that might affect one another. Records of weather data from a weather station containing

multiple variables such as air temperature, atmospheric pressure, humidity, wind direction etc. are measured at regular intervals and is an example of a multivariate dataset. Time series forecasting predicts the future values of a series based on its recent values which can either be univariate containing only one variable to be forecasted or multivariate using additional variables with relation to the value to be predicted. For the work in this thesis we will attempt to forecast ASKO load consumption using recent and current values. Variables such as time features and air temperature are added to the dataset making it a multivariate dataset.

3.1.2 Universal Machine learning workflow

The following subsection presents a universal blueprint that can be applied to any machine learning problem [20].

Defining the problem:

- what is the nature of the problem, is it a binary or multiclass classification problem?
is it a regression problem? is it a clustering problem?

Collecting data:

- what is the data availability?
- collect and label data if needed
- check for data imbalance
- secure domain knowledge

Deciding on an evaluation protocol:

- how to measure the training process?
- maintaining a hold-out validation, if plenty of data is available
- k-fold cross validation, in case of dataset containing few data samples

Preparing and visualizing the data:

The first part of this step is referred to as exploratory data analysis, after collecting data one would like it to be perfect which is not always the case. Therefore, it is important to

visualize data and distributions, identify outliers, handle missing and imbalanced data. Preprocessing data and making sure it is in order is a crucial step as it directly impacts the model development later on. Garbage in means garbage out. Preprocessing can consist of multiple techniques such as feature selection for dimensionality reduction, feature encoding and standardizing data so that each feature in the data is centered and scaled which is done for faster model convergence. Preprocessed data is then split into a train and test set. Using the train set to train and validate the model, and finally tested on the remaining part of the data to assess if the model is able to generalize well on unseen data.

Model development:

Develop a simple model that beats a dumb baseline to achieve some statistical power. After developing a simple model one should ask the following questions:

- is the model powerful enough?
- does it have enough layers and parameters to properly model the problem?

An ideal model would be at the border between underfitting and overfitting. To be able to understand where that border is one should develop a model too complex in the training phase. Identify overfitting by monitoring the training loss and validation loss.

Regularising the model and tuning model hyperparameters:

In this step the goal is to make the model as good as it can by repeatedly training, validating and modifying the model. Machine learning models have several hyperparameters which can be tuned to improve model performance such as activation function, loss function, learning rate and optimization function. Furthermore, the model architecture such as the number of units in a layer and the number of layers. Once a satisfying model is developed, the final model is trained on all the available data and evaluating it once last time on the test data. A final evaluation of the model is done using the predicted values and comparing against the true values using performance metrics chosen.

3.2 Neural networks

Neural networks (NN) are a subset of machine learning and the heart of deep learning algorithms(cite ibm what is a neural network).The concept of *artificial neural netowrks*

(ANN) were built upon hypotheses and models on how the brain works thus came the first implementation in the 1950s namely Rosenblatt's Perceptron model. However interest soon faded due to inadequate solutions for training a Neural Network with multiple layers. Today Neural Networks are more popular than ever thanks to major breakthroughs in the previous decade [17, chapter 12].

3.2.1 Single Neuron model

To understand the multilayer neural network. one must first understand the single neuron model. Using *Adaptive linear Neuron* (Adaline) as an example let us explain the single layer neural network architecture. A visualized representation of the Adaline algorithm can be observed in Figure 3.1

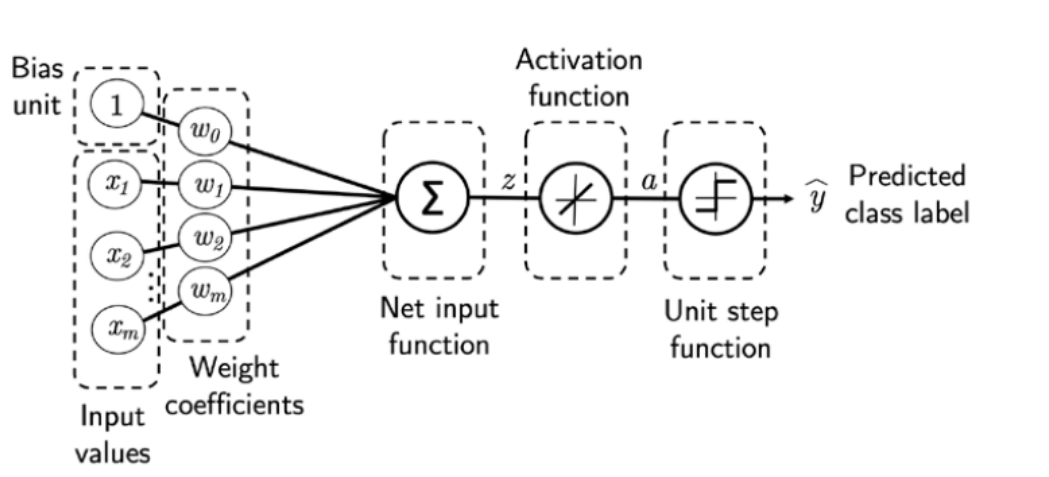


Figure 3.1: Illustration of a Single-layer neural network architecture Figure from [17, chapter 12]

The first step initializes *weights* in vector \mathbf{w} to 0 or small random numbers. Subsequently the *Net input*, z , is computed as a dot product of vector containing m weights \mathbf{w} and \mathbf{x} where \mathbf{x} is a vector of inputs fed to the model containing m features variables including the bias. z can be defined as the following [17, chapter 2]:

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \mathbf{w}^T \mathbf{x} \quad (3.1)$$

Furthermore, the net input is passed through an *activation function* $\phi(z)$ and transformed:

$$\phi(z) = \phi(\mathbf{w}^T \mathbf{x}) \quad (3.2)$$

In Adaline the activation function is an identity function $\phi(z)$. The learning process begins with optimizing the loss function in the case of Adaline defined as [17, chapter 2]:

$$J(\mathbf{w}) = \frac{1}{2} \sum_i (y^i - \phi(z^i))^2 \quad (3.3)$$

$J(\mathbf{w})$ is the sum of squared errors(SSE) between true value and the calculated outcome of the activation function over i observations. To find the optimal weights that minimizes the loss function, a simple optimization algorithm is used called **gradient descent**. The process of gradient descent visualized in Figure 3.2, can be described as climbing down a hill until a local or global minimum is reached.

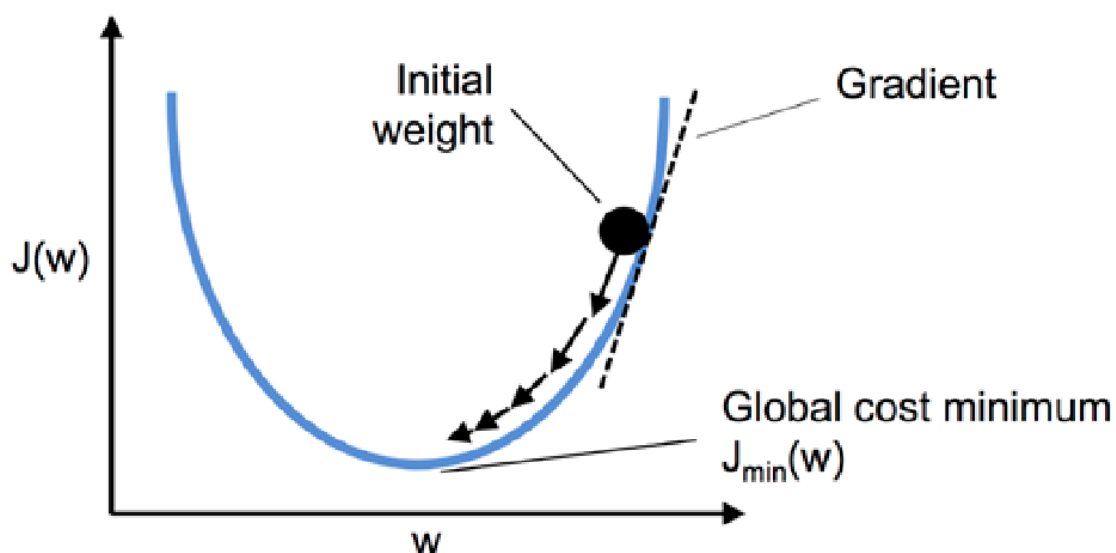


Figure 3.2: Concept of gradient descent. The algorithm starts from with random weights and proceeds to locate the global minimum of the loss function. The process of finding the global minimum can be described as climbing down a hill Figure from [17, chapter 2]

An update of the weight vector is defined as [17, chapter 2]:

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w} \quad (3.4)$$

where:

$$\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}) \quad (3.5)$$

In 3.5, η is the *learning rate* and $J(\mathbf{w})$ is the gradient of the loss function. We take a step in the negative direction of the loss function's gradient until we reach a local or global

minimum, where the learning rate determines the size of the step. Choosing a learning rate too large or small may result in risks of overshooting/undershooting the global minimum. One would prefer to reach the global minimum. A more novel approach to computing the gradient can be done by utilizing another gradient descent algorithm that build upon the basic approach concept discussed. It is possible to advance performance by using for example *stochastic gradient descent* (SGD).

To compute the gradient of the loss function, we take the partial derivative of the loss function with respect to each weight [17, chapter 2]:

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i (y^i - \phi(z^i)) x_j^i \quad (3.6)$$

These steps are then repeated updating the weights in the process until a stopping rule is met for example the number of epochs(each time the data passes through the model). Finally the last activations are passed to a *threshold function* which in this case outputs a binary prediction, if the output value is greater than or equal to 0 a class label of 1 is assigned and -1 otherwise [17, chapter 2].

3.2.2 Multilayer Neural Network (MLP)

Connecting multiple single neurons together forms a neural network, adding hidden layers and output layers produces a multilayer neural network. How deep a network is depends on the number of hidden layers in the network. An overview of how such a network is represented can be observed in Figure 3.3

The multilayer neural network illustrated in Figure 3.3 consist of three layers: *input layer*, *hidden layer* and an *output layer*. Here a_i^l refers to the i-th activation unit in the l-th layer, $w_{i,j}^l$ refers to the weight connecting unit i in l layer - with unit j in l-th layer and $i = 0$ refer to the bias units.

Starting at the input layer data is fed into the network in vector form. Once the input layer is in place, weights are assigned. The weights determine the importance of variables, variables assigned larger weights contribute more to the output. Each unit of one layer is connected to each unit of the next layer through weights, in Figure 3.3 these are the lines denoted by $w_{i,j}^l$. Inputs are multiplied by their respective weights and summed.

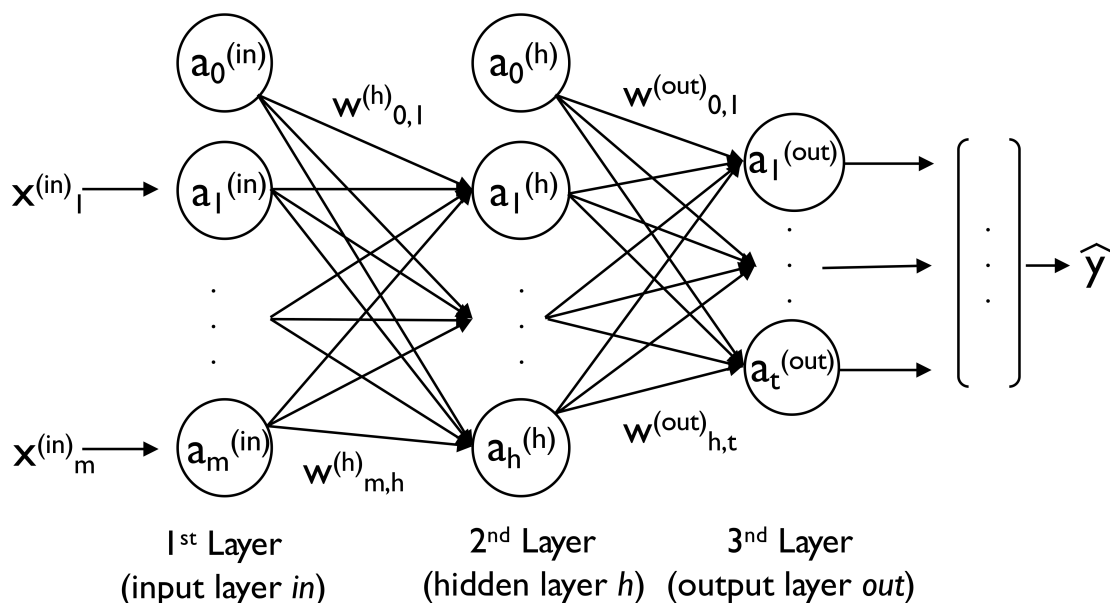


Figure 3.3: Illustration of a multilayer neural network architecture, consisting of one input layer, one hidden layer and an output layer Figure from [17, chapter 12, p.388].

Furthermore, the sum is passed through a non-linear activation function. Passing data from the input layer to the output layer is called *feedforward propagation*. Adding more hidden layers to the network increases its capacity to learn complex non-linear structures. A drawback of this is the increased model complexity prone to overfitting.

Taking the values of activations in the output layer, a prediction error is computed using a loss function and the true values. An important aspect of neural networks such as the MLP is the ability to send error from the last layers (output) to the first layers, called *backpropagation*. Backpropagation is a computationally efficient approach to calculating the partial derivative of complex loss functions (gradients) using the mathematical chain rule [17, chapter 12]. The partial derivatives are used by the optimization algorithm, e.g. gradient descent, to minimize the loss adjusting the weights thus lowering error and increasing predictive performance.

The MLP learning procedure can be summarised in three steps [17, chapter 12]:

1. Feedforward propagation, sending data through the network in order to generate an output
2. Compute error, based on the output an error is computed using a complex loss function

3. Backpropagation, sending error back into the network and computing its derivative with respect to each weight in order to update the model

3.2.3 Activation functions

For the network to be able to capture non-linear dependencies in data, one needs to use non-linear activation function. Without non-linear activation function the network will only perform linear operations such as dot product and addition [19, chapter 3]. An overview of different activation functions can be observed in Figure 3.4. Different functions are used depending on the problem. For example, *sigmoid activation function* is commonly used for binary classification problems, *hyperbolic tangent (tanh)* is similar to sigmoid as both are considered *s-shaped* functions. Sigmoid range of output values are between 0 and 1 while tanh ranges between -1 and 1. Thus its derivative can take on larger values compared to sigmoid allowing for larger weight updates and faster convergence. One popular and commonly used activation function is the *Rectified Linear Unit (ReLU)*, the function returns 0 if it receives any negative value and the same value back for any positive value, it is known for being able to tackle the vanishing gradient problem due to its derivative being 1 for positive values [17, chapter 13] [21].

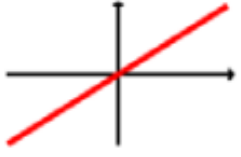
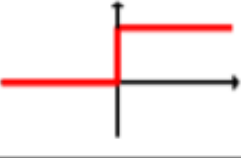
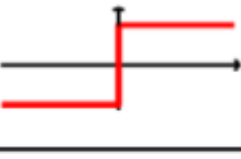

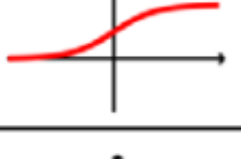
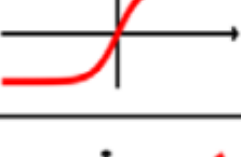
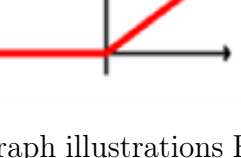
Activation function	Equation	Example	1D graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit step (Heaviside function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, multilayer NN	
Hyperbolic tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Figure 3.4: A selection of widely used activation functions and graph illustrations Figure from [17, chapter 13, p.469].

Vanishing gradient problem

The problem of the vanishing gradient can arise as more layers using certain activation functions are added to the network, the gradients of the loss function will be close to zero resulting in insignificant changes to the weights thus making the network unable to properly learn [22]. Gradients are found using backpropagation mentioned in Chapter 3.2.2 . The vanishing gradient problem is commonly caused by the use of the sigmoid function given by:

$$\phi_{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (3.7)$$

The output generated from this function is a number in a small range between 0 and 1. The sigmoid function and its derivative can be observed in Figure 3.5.

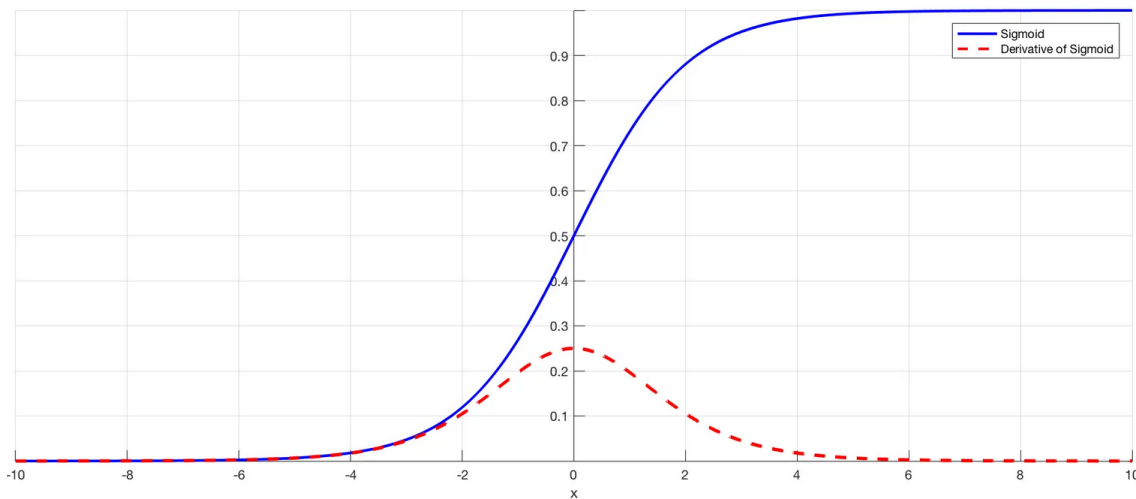


Figure 3.5: Illustration of sigmoid function and its derivative Figure from [22]

As observed from the Figure 3.5, the derivative of the sigmoid function will be close to 0 as inputs become large or small. Therefore, small derivatives are multiplied together and the gradient decreases as we propagate backwards into the network. One simple yet effective solution to the vanishing gradient problem is to change the activation function and use others such as Rectified Linear Unit.

3.2.4 Loss functions and Error metrics

A loss function is a quantity that will be minimized during training to evaluate how well the network models the data *i.e.* a measure of success during training [19, chapter 3]. Taking the predicted values and the actual values, a distance score is computed. The distance score can have a large value if the deviation between the prediction and true target is large. Different loss functions yield different errors. Therefore, choosing the correct loss function is a crucial part when building the network. *Binary crossentropy* for a binary classification problem, *categorical* for a many-class classification problem and *mean squared error (MSE)*, *mean absolute error (MAE)* for a regression problem. For the work in this thesis functions used are described as follow [23]:

- mean squared error (MSE) - is measured as the average of squared difference between predictions and actual observations [24], defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^i - \hat{y}^i)^2, \quad (3.8)$$

where n denotes the number of observations, y^i denotes the true observation and \hat{y}^i denotes the predicted values. Due to squaring, predictions that deviate further from the true value are given larger errors thus given larger weights.

- mean absolute error(MAE) - is measured as the average absolute values of the prediction error, defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^i - \hat{y}^i| \quad (3.9)$$

- root mean squared error (RMSE) - is the square root of MSE:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y^i - \hat{y}^i)^2} \quad (3.10)$$

- mean absolute percentage error (MAPE) - the average of the absolute differences

between actual observations and predictions

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y^i - \hat{y}^i}{y^i} \right| \times 100 \quad (3.11)$$

- coefficient of determination R^2 - the accuracy of predictions:

$$R^2 = \frac{\sum_{i=1}^n (y^i - \hat{y}^i)^2}{\sum_{i=1}^n (y^i - \bar{y})^2} \quad (3.12)$$

if predictions matches the true values perfectly $R^2 = 1$, where \bar{y} is the mean value [25]

3.2.5 Recurrent neural network (RNN)

Standard neural network models such as the MLP assume that the data inputs are independent of each other. Based on this independence assumption, the order in which the data inputs are fed to the network is irrelevant. Such models are not capable to keep information in memory about previously seen training examples (data inputs). Data inputs are passed through the feedforward and backpropagation steps, updating the weights independently of the order in which they are presented to the network [17, chapter 26].

Time series data, a type of sequential data that appear in a certain order, are not independent of each other. Hence, the assumption of independence is not valid and data has to be treated in a sorted order (in this case sorted along the time axis) which is done in order to leverage information about previously seen training examples. The *recurrent neural networks* architecture is specifically designed for modeling sequences, they are networks with loops in them being able to remember past information when processing new events. A simple one hidden layer case is presented in Figure 3.6

The input layer is denoted by \mathbf{x} , hidden layer by \mathbf{h} and output layer by \mathbf{o} . Weight matrices connecting the different layers are denoted by \mathbf{W} along with a subscript which specify layer. The difference between standard neural networks and RNNs lies in the fact that each hidden unit in the hidden layers receives two set of inputs, the preactivation from the input layer and the activations from the same hidden layer from the previous timestep, $t - 1$. The weight matrix \mathbf{w}_{hh} in Figure 3.6 is associated with the *recurrent edge* (loop)

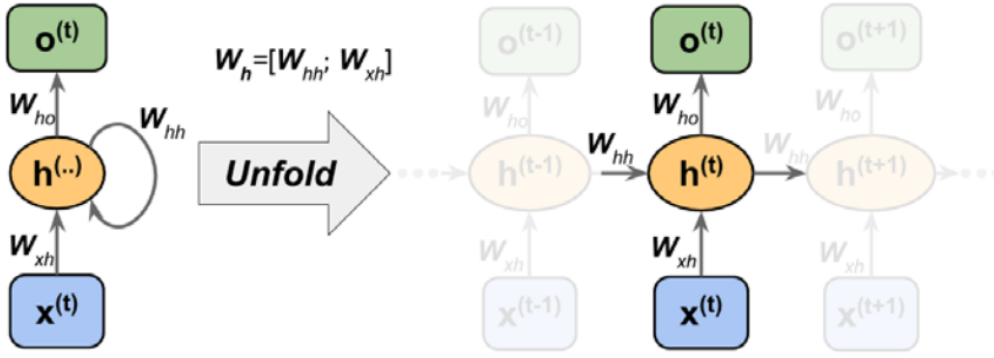


Figure 3.6: Illustration of a Recurrent neural network layer and its looping ability, left side closed version while right side unfolded version Figure from [17, chapter 16]

illustrated as the round arrow. The recurrent edge is the connection between the previous time steps allowing to retain past information. The left hand side of Figure 3.6 presents the architecture in a closed version and the right hand side of the figure in an unfolded version. The activations of the hidden units at time step, t , can be calculated in the following way [17, chapter 16]:

$$\mathbf{h}^t = \phi_h(\mathbf{z}_h^{(t)}) = \phi_h(\mathbf{W}_{xh}\mathbf{x}^t + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h) \quad (3.13)$$

ϕ_h is the activation function and \mathbf{b}_h is the bias in the hidden layer. The backpropagation step in RNNs can become complicated, *i.e.* the error is sent across the layers and the time steps. Overall loss, L , is the sum of all the loss functions at times $t = 1$ to $t = T$. Problems such as the vanish gradient and exploding gradient when computing the gradients of a loss function mentioned in section 3.2.3 could arise leading to poor predictions. A way of tackling such problems is the implementation of a certain layer called *long short-term memory* (LSTM).

In some cases it would be enough to look at recent information to make predictions for the present task. However, there might be relevant information going further back. Simple RNNs might not be capable of handling such long-term dependancies. The solution are LSTM layers, capable of learning information for long periods of time. The core idea behind LSTM layers is the *memory cell* which replaces the standard hidden layer of RNNs. An overview of how such a memory cell is represented can be observed in Figure 3.7

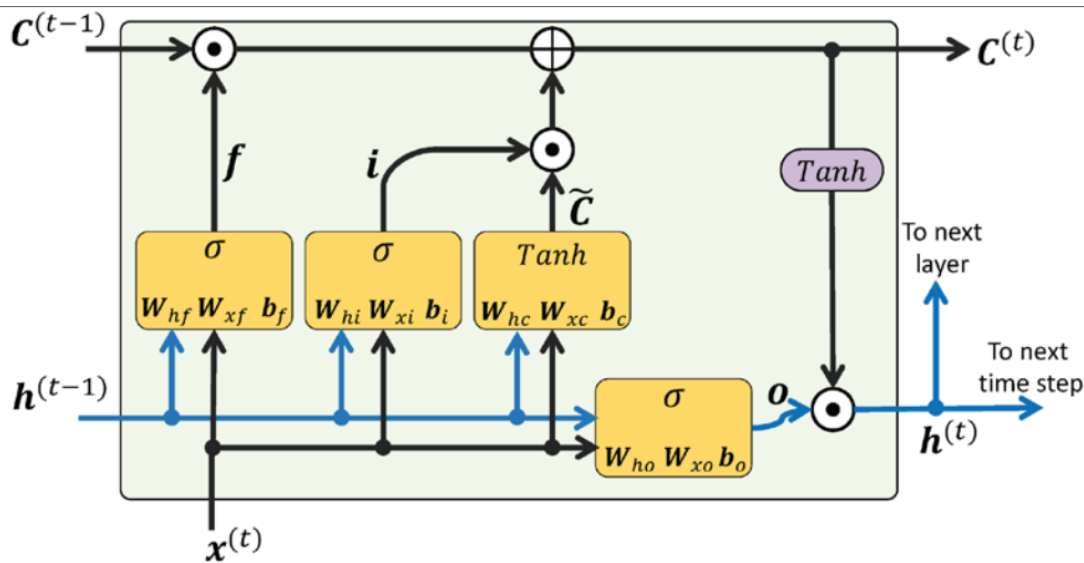


Figure 3.7: Illustration of the LSTM memory cell structure. The horizontal line on top of the figure represents the flow of information. Forget, input, candidate and output gates are illustrated as the yellow boxes. Figure from [17, chapter 16]

The horizontal line at the top of Figure 3.7 is called the *cell state*, C^t , it is an additional flow carrying information across several time steps without being multiplied with any weight factor. Past information is retained [17, chapter 16]. The LSTM has the ability to discard or add information to the cell state, regulated by several *gates*. Gates consist of a sigmoid neural net layer and an element-wise multiplication. The output of the sigmoid layer (between 0 and 1) decides how much information should be let through. A value of 0 discards information while a value of 1 lets information through. An LSTM layer has 3 gates, *forget gate*, *input gate* and *output gate* controlling the cell state.

The first step in a LSTM layer is to decide what information in the cell gate to keep or forget. A sigmoid layer handles the decision, looking at $\mathbf{h}^{(t-1)}$ and \mathbf{x}^t . The forget gate is computed as follows [17, chapter 16]:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}^{(t)} + \mathbf{W}_{hf}\mathbf{h}^{(t-1)} + \mathbf{b}_f) \quad (3.14)$$

The next steps involve the decision on new information to store in the cell. This is done in two parts, a sigmoid layer called input gate layer decides which values to update. Following, a tanh layer that creates a vector of new candidate, \tilde{C}_t values to be added to the cell state. These steps are computed by the following equations [17, chapter 16]:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}^{(t)} + \mathbf{W}_{hi}\mathbf{h}^{t-1} + \mathbf{b}_i) \quad (3.15)$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_{xc}\mathbf{x}^{(t)} + \mathbf{W}_{hc}\mathbf{h}^{t-1} + \mathbf{b}_c) \quad (3.16)$$

The old cell state is updated into a new cell state, forgetting information decided by the forget gate and adding new information decided by both input gate and new candidates, using the following equation [17, chapter 16]:

$$\mathbf{C}^{(t)} = (\mathbf{C}^{(t-1)} \odot \mathbf{f}_t) \oplus (\mathbf{i}_t \odot \tilde{\mathbf{C}}_t) \quad (3.17)$$

where \odot refers to the element-wise multiplication and \oplus refers to element-wise summation. The output is computed which is based on a filtered cell state. Passing through a sigmoid layer deciding on cell state output computed as follows [17, chapter 16]:

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}^t + \mathbf{W}_{ho}\mathbf{h}^{t-1} + \mathbf{b}_o) \quad (3.18)$$

Finally, the cell state is put through a tanh layer and multiplied with the output from the sigmoid layer to only output chosen information with the following equation [17, chapter 16]:

$$\mathbf{h}^{(t)} = \mathbf{o}_t \odot \tanh(\mathbf{C}^{(t)}) \quad (3.19)$$

3.2.6 Model complexity

One important part of any machine learning model development is to make sure that the model is able to capture general trends in the data. A machine learning model can in some cases memorise specific patterns and noise in the training dataset. A direct consequence of this happening would be poor predictions, meaning that the model would not be able to predict well on unseen data. This is called overfitting and the model has high variance, overfitting can be caused by having too many parameters leading to a model that is too complex. Similarly, the opposite situation is called underfitting. The model is not able to

capture the general trends in the training dataset, leading to poor predictions on unseen data. In the case of underfitting the model is not complex enough and suffers from high bias [17, chapter 3]. The problems of overfitting and underfitting is illustrated by Figure 3.8.

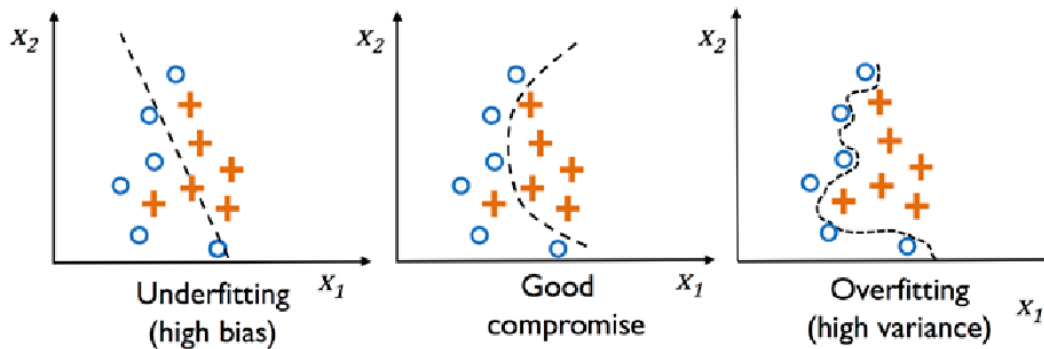


Figure 3.8: Figure showing the cases of a model underfitting left most figure, overfitting right most figure and a model with a good compromise which would be the most ideal case when developing a machine learning model, middle figure. Figure from [17, chapter 3].

Cross-validation techniques are used during the machine learning model training process to estimate the model generalization performance on new data. When tuning the model in search for the best hyperparameter combinations, several models are fitted to the training dataset. These models need to be evaluated on unseen data using the validation dataset, in order to select the best performing model. There are several cross-validation methods such as [17, chapter 6]:

- holdout method - this method involves splitting the input dataset into three different sets. Training, validation and test sets. The training dataset is used for training the model, the validation dataset set is used during training to evaluate model generalization performance and the final evaluation done on the unseen data, test set.
- k-fold cross-validation - in this method the training dataset is split into random k folds without replacement, where $k - 1$ folds are used for training the model and one fold is used for evaluating model performance.
- Leave-one-out cross-validation - a special case of k -fold cross-validation where a single training example is left out and used for testing during each training epoch.

For a model the loss and validation loss can be recorded for the training and validation sets for every epoch during the training process. These values can be used to plot training and validation loss curves to identify if a model is overfitting or underfitting when tuning one or several parameters. Should training loss decrease over time and validation loss decrease until a turning point and start increasing, the model is likely overfitting. Figure 3.9 show the training and validation loss curves for a model overfitting.

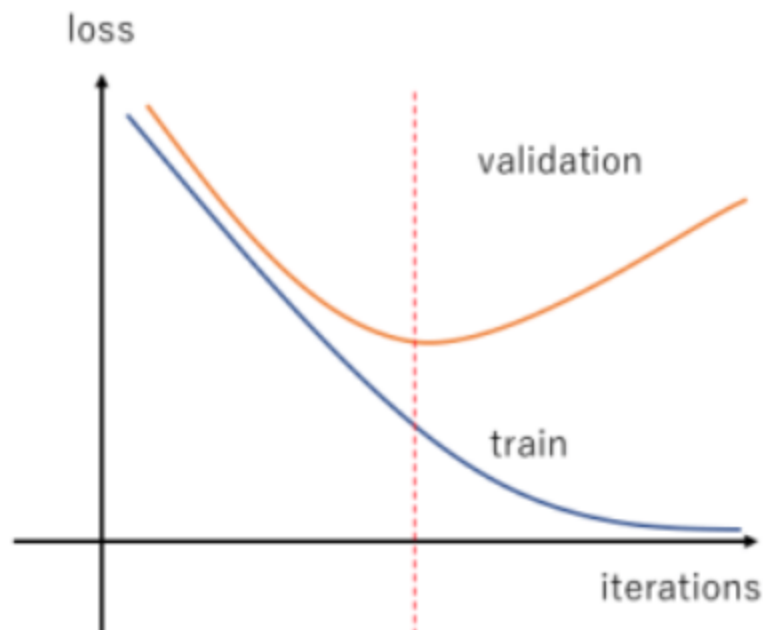


Figure 3.9: Visualisation of training and validation loss curves of a model overfitting characterised by the validation loss increasing after a certain point, loss along the y-axis and number of iterations along the x-axis Figure from [26]

Tuning the model complexity is an important part of model development to achieve a good bias variance trade-off. To tackle the problem of overfitting, one can:

- reduce the number or layers
- reduce the number of units in layers
- add L1 and/or L2 regularisation, regularization penalizes extreme parameters (weight) values [17, chapter 3]
- regularizing a neural network by adding dropout [17, chapter 15]

4 Methodology

4.1 Exploratory data analysis

This section covers the exploratory data analysis performed on the dataset before model development. The goal of this part was to get familiar with the data.

The dataset provided by ASKO consisted of time series representing the hourly load consumption profiles of 11 facilities including electrical load bought, sold, produced and total consumption covering 2 years of historical data points between 01/01/2020 and 31/12/2021 - 17544 data points. The dataset can be divided into two main parts, the aggregated portfolio of 11 buildings and the load profiles for each building separated. Among these 11 buildings several have solar panels that are able to produce electrical load hence the production column.

After importing the data into python from excel was it was viewed. Several columns containing nan values were found, due to importing from excel format. These columns were dropped. Furthermore, a datetime index was created from the timestamp column in the dataset. The last two rows contained minimum and maximum values, these were dropped. Initial investigation showed no missing values. However, after changing the index frequency to hourly, two duplicate rows were discovered. The duplicate rows were investigated and two of those rows did not have values resembling the preceding or succeeding values. The two duplicate entries that had extreme values compared to the other values were dropped. For the whole dataset two missing rows were identified. The missing values were filled with the nearest preceding or succeeding row value for each column. The dataframe consisted of 4 columns (load bought, sold, produced and total consumption) and 17544 rows of data points. In this project the total consumption served as the target value to be predicted and the the remaining columns load bought, sold and produced served as explanatory variables.

The historical temperature values were imported. Temperature values imported from the Frost API were available in 10 minute intervals. Therefore, the 10 minute values ranging from 01/01/2020 and 31/12/2021 had to be resampled to hourly resolution using the mean values of 10 minute resolution. The temperature values dataframe was concatenated

with the ASKO dataset.

In the next step of the analysis, the data were plotted see Figures 4.1, 4.2, 4.3. The total consumption profile is plotted to identify the general trend of this type of data. Inspecting the figures, a seasonal behaviour can be observed. Furthermore, it can be observed that the data is volatile with regular spikes (extreme high or low values) for both years. The maximum consumption is recorded during the warmest period.

A further data investigation of the seasonal behaviour was conducted and plotted in the form of heatmaps. The total consumption was resampled into different frequencies such as monthly, daily and by hour of day. Monthly frequency heatmap was created in order to address the seasonal component of the data, daily heatmap was created in order to address which days had the highest electrical consumption and finally heatmap of hour by day was created in order to address which hours during the day had the highest electrical consumption.

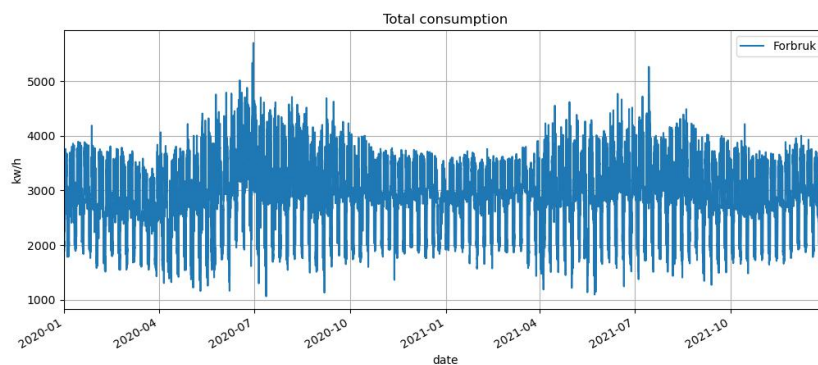


Figure 4.1: Visualisation of total load consumption, time along the x-axis and load (kw/h) along the y-axis

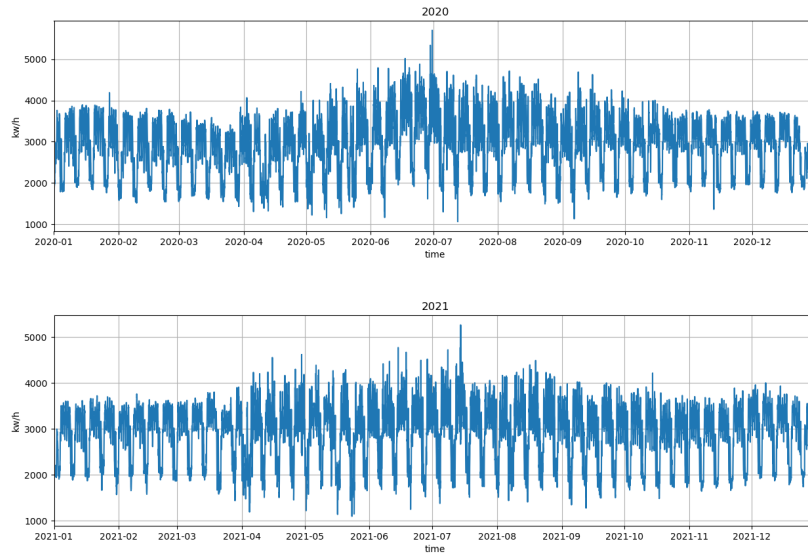


Figure 4.2: Visualisation of total load consumption by year, time along the x-axis and load (kw/h) along the y-axis

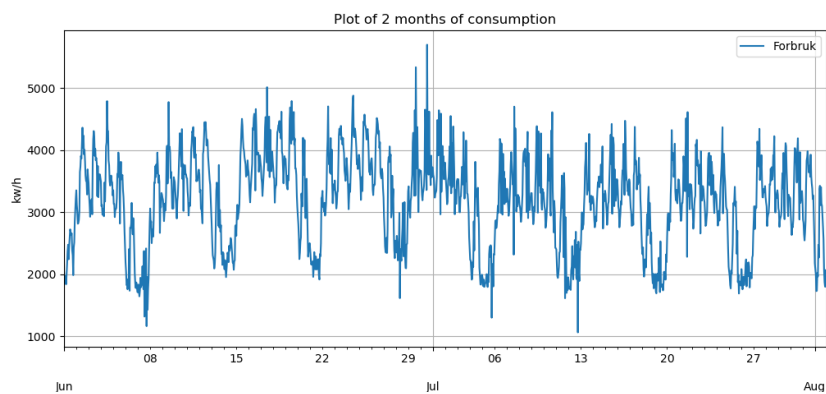


Figure 4.3: Better visualisation of total load consumption using 2 months, time along the x-axis and load (kw/h) along the y-axis

4.2 Preprocessing

After investigating the seasonal behaviour of load consumption the dataset was expanded to include several time feature variables in order to improve data quality fed to the model. Time features assigned values and a brief explanation added to the dataset is presented in table 4.1

Table 4.1: Time features included

Time feature	Value	Description
Hour	0 - 23	0 means 12 am and so forth
Day of the week	0 - 6	Monday = 0, Sunday = 6
Quarter	1 - 4	January, February and March assigned value 1 and so forth
Month	1-12	January assigned 1 and so forth
Year	2020 - 2021	Either year 2020 or 2021
Day of year	1 - 365	First day of the year assigned value 1 and so forth
Day of month	1 - 30 or 31	First day of the month assigned value 1 and so forth
Week of year	1 - 52	First week of the year assigned value 1 and so forth

One problem with time feature variables is that the model may not be able to extract useful information other than the ascending order of data points [27]. Consider for example the first day of a year assigned value 1 in contrast to the last day of the year assigned value 365. The model may consider the last day of higher magnitude than the first. One solution is to one-hot encode the time features increasing the dimensionality of the dataset. To limit the increased dimensionality of the dataset and extract meaningful information from these variables, the time feature variables were converted into sine and cosine values. Each column of time features were converted into two.

The dataset was supplemented with the air temperature values to further enrich information. Air temperature values were collected from a weather station close to ASKO facility. Weather data was collected from the Frost API - a free access archive of historical weather and climate data [28].

4.2.1 Splitting data

A common practice when building a machine learning model is to estimate its performance on data that the model has not seen before. For the model to be able generalize well on unseen data techniques such as **holdout cross-validation** and **k-fold cross-validation** are used [17, chapter 6]. In this project the split was done chronologically, as it is important to keep the order of consecutive data points when analysing time series data. As the chronological order of data matter, the hold out method was considered in this project. The original dataset (illustrated as the top row in Figure 4.4) was split into separate training and test data sets - the former used when training the model encompassing 90%, and the latter used to estimate the models ability to generalize consisting of the remaining 10%. Furthermore, the training set was split into a training and validation set, the validation set encompassing 10% of the training set. The validation set is used during training to evaluate the model.

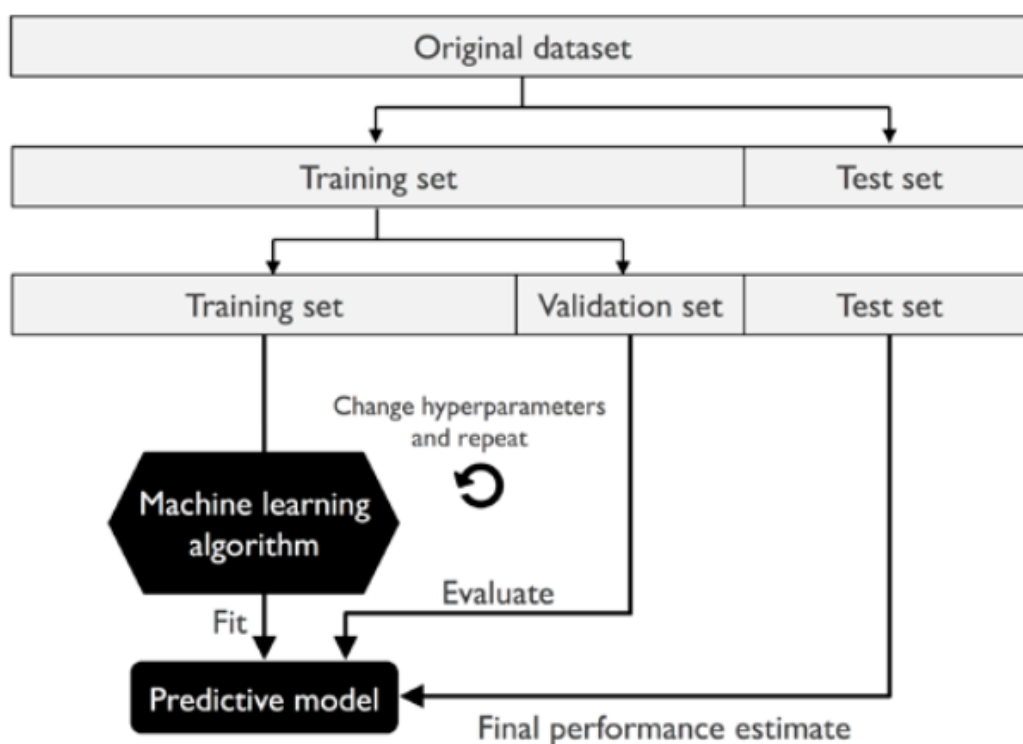


Figure 4.4: Figure showing how input data is split into a training set, validation set and a test set used when training a machine learning model [17, chapter 6]

4.3 Deep Learning modelling using RNN architecture

This section covers the methodology used to develop a machine learning model using the RNN architecture.

After preprocessing and splitting the data, the input data set was scaled. A scaler function was fitted to the training data and subsequently used to scaled the entire dataset. The scaler function applied to the training data set was **MinMaxScaler**, translating each feature individually between zero and one [29]. Feedforward artificial neural networks take two-dimensional matrices of input data in of the shape (time steps, input features), the RNN layer however processes sequences, it therefore requires 3D tensors as input of the shape (batchsize, time steps, input features) [19, chapter 6]. To create 3D tensors from the data set that the RNN layers can process, a generator provided by Keras was used. Using the **TimeSeriesGenerator** the following parameters that determines the 3D tensor shape has to be specified [30]:

- **data** - the data points of input features
- **target** - the targets corresponding to data points in data
- **length/lookback** - the length of the previous time steps that the RNN layer consideres
- **batch size** - the number of time series samples in each batch

The function takes in a sequence of data-points gathered at equal intervals and produces batches of input features and the corresponding targets. Three generators were instantiated to create batches of data from the training, validation and test sets. 3D tensor were made from the scaled data in chronological order for training, validation and test sets of size 80%, 10% and 10%.

After preparing the 3D tensors using **TimeSeriesGenerator**, the data was ready to be fed to the model for training. In the next step a framework for automating the tuning process of the recurrent neural network was set up using the KerasTuner API [31]. The main goal of the tuning process was to develop a hyperparameter combination that yielded the best model performance based on the metrics passed as arguments when instantiating the tuner. The KerasTuner API offer several tuners e.g. **RandomSearch**

and **Hyperband**, however the **BayesianOptimization** tuner was chosen. This tuner is an informed search method able to learn from previous iterations. Instead of testing hyperparameter combinations at random, it investigates the search space and converges to the optimal hyperparameters based on promising hyperparameter combinations from previous iterations [32].

The most important part when defining a **BayesianOptimization** tuner is to define a function that builds the machine learning model to be tuned. The code for defining a search space and building a model is presented in code Listing 1. The function takes an instance of keras **HyperParameter** class **hp** as an argument [33].

```
1 # imports
2 from keras.models import Sequential
3 from keras import layers
4 from tensorflow.keras.optimizers import RMSprop, Adam
5 from sklearn.metrics import mean_absolute_error,
   mean_squared_error, mean_absolute_percentage_error, r2_score
6 from tensorflow.keras.losses import MeanSquaredError
7 from keras_tuner.engine.hyperparameters import HyperParameters
8
9 # Instansiate hyperparameter class
10
11 hp = HyperParameters()
12
13 def build_model(hp):
14 """
15 This function defines the search space for the tuner, and builds
   a network architecture based on the hyperparameters
   combinations passed as an argument. Finally, returns a
   compiled model.
16 """
17 model_lstm = Sequential()
18
19 # This part of the build_model defines the search space with
```

```
the hyperparameter class
20
21 lstm_1_units = hp.Int('lstm_1_units',
22                       min_value = 50, max_value = 350, step = 50)
23 dropout_layer_1 = hp.Choice('dropout_layer_1',
24                              values = [0.0, 0.1, 0.2, 0.3, 0.5])
25 layer_2 = hp.Boolean('layer_2')
26 lstm_2_units = hp.Int('lstm_2_units',
27                       min_value = 50, max_value = 350, step = 50)
28 dropout_layer_2 = hp.Choice('dropout_layer_2',
29                              values = [0.0, 0.1, 0.2, 0.3, 0.5])
30 optimizer = hp.Choice('optimizer',
31                       ['adam', 'RMSprop'])
32 learning_rate = hp.Float('learning_rate',
33                           min_value = 1e-5, max_value = 1e-2, sampling = '
log')
```

Listing 1: Python code for building a model part 1

The **HyperParameter** class was used to define a search space, a search space is a grid of hyperparameters values that the tuner draws hyperparameter combinations from. As shown in the code example above, the search space included several parameters of the RNN layer:

- **lstm_1_units** - the number of units in the first LSTM layer ranging between 50 - 350 in steps of 50
- **dropout_layer_1** - the dropout rates for the dropout layer ranging between 0 for no dropout and 0.5 for max dropout rate
- **layer_2** - a boolean value, True or False, if the values is True another LSTM layer is added to the network
- **lstm_2_units** - the number of units in the second LSTM layer ranging between 50 - 350 in steps of 50
- **dropout_layer_2** - the dropout rates for the second dropout layer ranging between

0 for no dropout and 0.5 for max dropout rate.

- **optimizer** - the optimising algorithm to be used, choices between Adam and RMSprop
- **learning_rate** - the learning rate used by the optimizer minimum value 0.00001 and max value 0.01

```
1 # This part of the build_model code builds the model
   architecture with the specified search space
2
3 model_lstm.add(layers.LSTM(units = lstm_1_units ,
4                             return_sequences = layer_2 ,
5                             input_shape = (lookback , x_train.shape[-1]) ,
6                             activation = 'tanh' ,
7                             recurrent_activation = 'sigmoid' ,
8                             recurrent_dropout = 0 ,
9                             unroll = False ,
10                            use_bias = True))
11
12 model_lstm.add(layers.Dropout(dropout_layer_1))
13
14 if (layer_2 == True):
15     model_lstm.add(layers.LSTM(units = lstm_2_units ,
16                                 activation = 'tanh' ,
17                                 recurrent_activation = 'sigmoid' ,
18                                 recurrent_dropout = 0 ,
19                                 unroll = False ,
20                                 use_bias = True))
21
22     model_lstm.add(layers.Dropout(dropout_layer_2))
23
24 model_lstm.add(layers.Dense(1 , activation = 'sigmoid'))
25
```



```
26 hp_optimizer = optimizer
27
28 if hp_optimizer == 'adam':
29     hp_learning_rate = learning_rate
30
31 if hp_optimizer == 'RMSprop':
32     hp_learning_rate = learning_rate
33
34 # compiling the model
35 model_lstm.compile hp_optimizer ,
36     loss = 'mse' ,
37     metrics = ['mae' , 'mse' ]
38 return model_lstm
```

Listing 2: Python code for building a model part 2

The second part of the function building the model is shown in Listing 2. The code defines at least one LSTM layer. If a second LSTM layer is chosen it is added. Finally, an output layer is added. Furthermore, the model is compiled with the chosen metrics and returned.

Note, some of the layer parameters were fixed. In order to utilize a GPU runtime provided by the Kaggle environment to speed up the training and tuning process, these were held fixed. Due to hardware limitations speeding up the process was important as the tuning process ran for several hours despite utilizing GPU runtime. The LSTM layer activation function had to be fixed to 'tanh' activation, the recurrent activation to 'sigmoid', recurrent dropout rate to 0, unroll to False and use bias set to True according to [34].

The code for defining the **BayesianOptimization** tuner is presented in Listing 3 [35]. Parameters that needs to be specified when creating a tuner are as follows: hypermodel - the machine learning model that should be tuned, the objective function - in this project it was set to the loss computed on the validation set, max trials - the total number of trials to test drawing hyperparameter from the search space, execution per trial - the number of executions per trial for a set of hyperparameters tested.

```
1 from keras_tuner.tuners import BayesianOptimization
2
3 tuner_lstm = BayesianOptimization(build_model,
4                                   objective = "val_loss",
5                                   max_trials = 50,
6                                   executions_per_trial = 1,
7                                   directory = '/kaggle/working',
8                                   project_name = 'rnn-lstm-tuner')
```

Listing 3: Python code for for defining a BayesianOptimization tuner

```
1 from tensorflow.keras.callbacks import EarlyStopping
2
3 early_stopping = EarlyStopping(monitor = 'val_loss', patience =
4                                 16, mode = 'min')
5 # Run the hyperparameter search
6 tuner.search(x_train_gen,
7             epochs = 80,
8             validation_data = x_val_gen,
9             shuffle = False,
10            callbacks = [early_stopping],
11            steps_per_epoch = 200)
```

Listing 4: Python code for starting the tuner search for hyperparameter

The code for running the search is presented in Listing 4. A keras callback object was initialised, **EarlyStopping**. The **EarlyStopping** callback is responsible for stopping the training process when the monitored metric has stopped improving [36], in this project the validation loss. The patience argument is the number of epochs with no improvements after which the training will be stopped, set to 16 epochs.

The arguments the search methods takes are the training and validation data (training and validation data were yielded by the **TimeSeriesGenerator**, the number of epochs to run the training process for in each trial and finally the callback objects. When using the

TimeSeriesGenerator one does not have to specify the steps per epoch or validation step arguments as they are passed by the generator automatically.

Parameters that the **TimeSeriesGenerator** takes which determine the shape of the 3D tensors were manually tested. Lookback, batch size were fixed at 336 and 32.

After testing several hyperparameter combinations, the tuner returns the best hyperparameter combination that yielded the lowest validation loss. The best model itself is also returned and can be saved and loaded to be used for later [37].

Finally, the best model derived from the tuner was re-trained, evaluated and used to make predictions on the training and test data sets. Inverse scaling had to be applied on the output from the model to revert the scaling and restore the original scale. Furthermore, a range of different metrics such as MAPE, RMSE, MAE and R2 score were computed using the ground truth (actual values) and the predicted values. Additionally, results were plotted to visually assess model performance by observing the match between the ground truth and predicted values.

The model summary of the best performing models for the single-step and day-ahead models is provided by Figure 4.5, Figure 4.6 and 4.7 respectively.

```

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
lstm (LSTM)                  (None, 336, 250)         269000
lstm_1 (LSTM)                (None, 250)               501000
dropout (Dropout)           (None, 250)                0
dense (Dense)                (None, 1)                  251
-----
Total params: 770,251
Trainable params: 770,251
Non-trainable params: 0
-----

```

Figure 4.5: Summary of best single-step model showing each layer, input shape and the number of parameters in the model

```

Model: "sequential"

```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 336, 300)	382800
dropout (Dropout)	(None, 336, 300)	0
lstm_1 (LSTM)	(None, 50)	70200
dense (Dense)	(None, 1)	51

```

=====
Total params: 453,051
Trainable params: 453,051
Non-trainable params: 0

```

Figure 4.6: Summary of best day-ahead model showing each layer, output shape and the number of parameters in the model

```

Model: "sequential_2"

```

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 336, 256)	281600
dropout_1 (Dropout)	(None, 336, 256)	0
lstm_3 (LSTM)	(None, 4)	4176
dense_1 (Dense)	(None, 1)	5

```

=====
Total params: 285,781
Trainable params: 285,781
Non-trainable params: 0

```

Figure 4.7: Summary of best day-ahead model for building 1 showing each layer, output shape and number of parameters

4.4 Software

Experiments in this project were conducted in an online python environment provided by Kaggle ¹ and Google Colaboratory². Kaggle and Google Colaboratory provide a jupyter notebook service and access to computational resources such as GPU runtimes.

Experiments were conducted using Python version 3.7.12 and libraries are listed in Table 4.2.

Table 4.2: Python libraries used in this project

Library	Version
Numpy	1.21.6
Matplotlib	3.5.3
Pandas	1.3.5
Tensorflow	2.11.0
Keras	2.11.0
Scikit-learn	1.0.2

¹<https://www.kaggle.com/>

²<https://colab.research.google.com/>

5 Results

This chapter presents the results from the exploratory data analysis, including the prediction results from the modelling and tuning done in this thesis.

5.1 Exploratory data analysis results

Electricity consumption has a seasonal behaviour being highly correlated with seasons. Some days may have high consumption while other days have lower. In order to address this heatmaps were created of data in different frequencies. Data was first resampled and aggregated to daily level and can be observed in Figure 5.1. As expected, working days have a higher electrical consumption than weekdays.

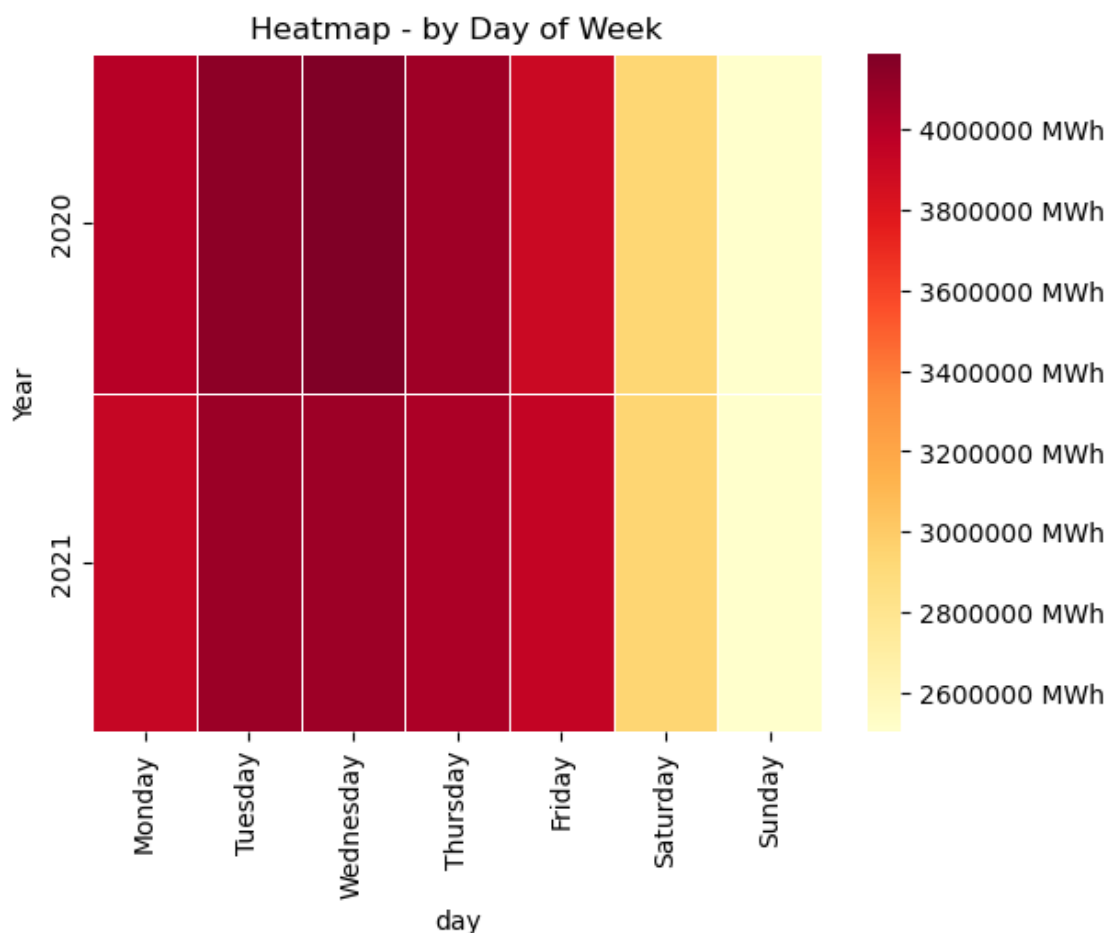


Figure 5.1: Heatmap showing daily electrical consumption

Data was then resampled and aggregated on a hourly frequency. Hourly electrical consumption profiles are visualized in Figure 5.2. The highest electrical consumption can

be observed during business hours from 8 a.m to 4 p.m and peak between 9 a.m to 12 p.m.

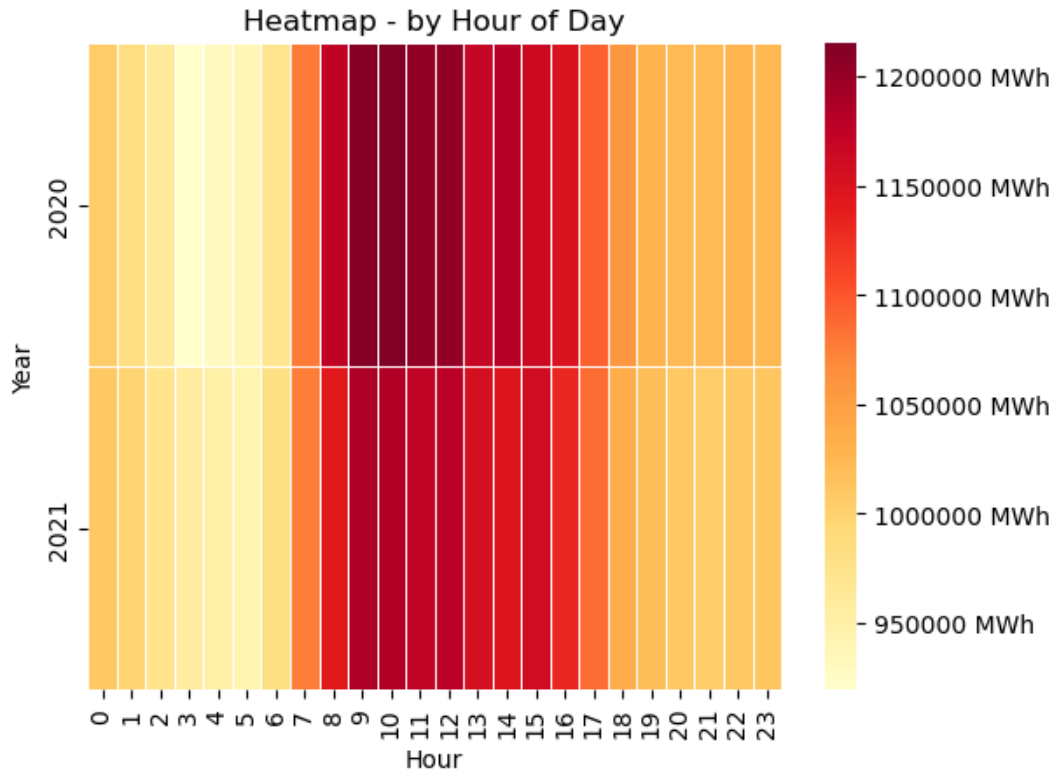


Figure 5.2: Heatmap showing hourly electrical consumption

Finally data was resampled and aggregated on a monthly frequency. Monthly electrical consumption profiles are visualized in Figure 5.3. Seasonal patterns of electrical consumption can be observed from Figure 5.3. In 2020 the peak months were June, July and August, June recording the highest consumption during the hottest period of the year. The same pattern can be observed for 2021. Extremely hot or cold season have the highest electrical consumption.

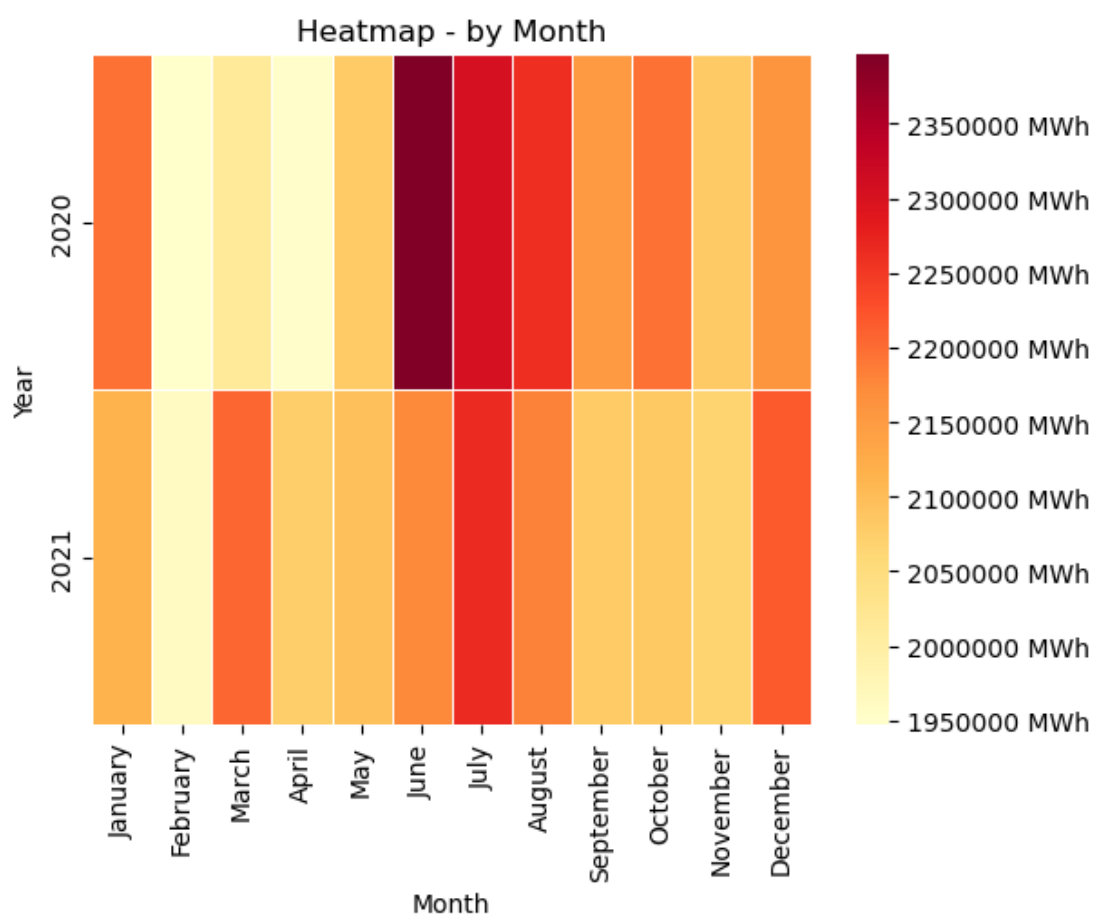


Figure 5.3: Heatmap showing monthly electrical consumption

5.2 Single-step recurrent neural network (RNN) forecast results

The RNN model has been trained to forecast ASKO facility electrical consumption one hour ahead, feature variables include electricity purchased, electricity generated using solar panels, electricity sold, time variables and air temperature values collected from a weather station close to the facility. Performance metrics for the model achieving the lowest validation loss score in the hyperparameter tuning process are presented in Table 5.1, for both train and test data. The model was trained on electrical consumption time series data with a temporal resolution of 1 hour. Batchsize was set to 32 with lookback set to 336, each sequence fed to the model had a length of 336 looking back at 2 weeks of prior data points before making a prediction. The best set of hyperparameters from the model tuning process consist of a LSTM layer with 250 units, tanh activation function and a dropout rate of 0.0. A second LSTM layer was included in the model tuning process with units set to 250, tanh activation function and a dropout layer with rate of 0.5. Optimizer was set to "Adam" with a learning rate of 0.001.

Table 5.1: Best performing single-step RNN model metrics

Data	MAE	MAPE	RMSE	R^2
Train	145.25	0.0506	197.78	0.90
Test	104.69	0.0439	149.09	0.94

Training prediction are visualised in Figure 5.4 and Figure 5.5. Each Figure show the predictions made by the model for electrical consumption time series data. For a more zoomed in visualization see Figure 5.5. Missing predictions at the beginning of the plot is caused by the lookback parameter, the number of samples in each sequence. Target sequence is shifted ahead determined by the lookback parameter, thus predictions will lack 336 (2 weeks) first time steps as well as time steps at the end chosen as validation data. It can be observed that the model struggles to capture spikes in the data (time steps where consumption is either extremely high or low). Time series such as electrical consumption data are highly volatile and predicting spikes accurately may be challenging. The model is however able to follow the seasonal behaviour to some extent observed in Figure 5.5.

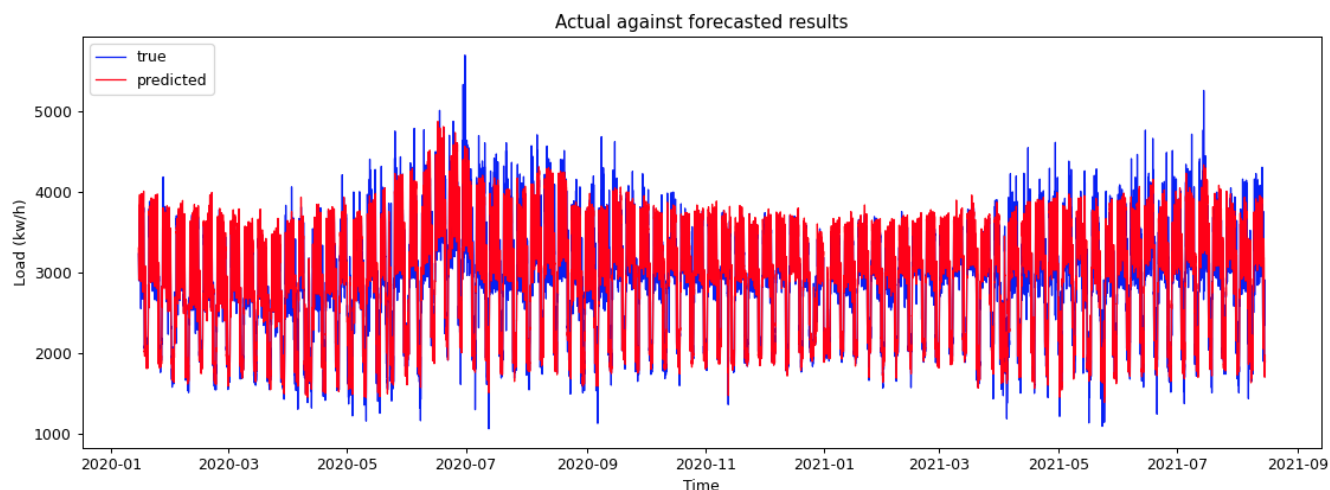


Figure 5.4: Best results of single step RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

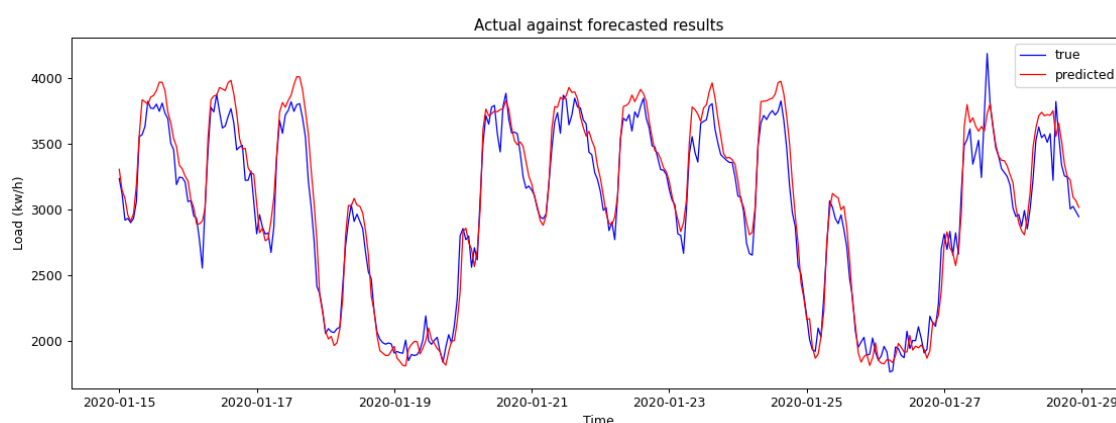


Figure 5.5: Best results of single step RNN model after tuning, train set predictions plotted against true values zoomed in observing 2 weeks of hourly electrical consumption. Hour along the x-axis and load (kw/h) along the y-axis.

Training and validation loss learning curves for the best performing tuned RNN model can be observed in Figure 5.6. Validation loss is slightly lower than the training loss and continuously decreasing per epoch. Number of epoch were set to 200, it can be observed that the model stopped training after 58 epochs. This is due to the **EarlyStopping** callback not recognizing further improvement thus stopping training to prevent overfitting.

Predictions made on the test set are visualised in Figure 5.7, the figure shows hourly electrical consumption predictions made on test set. The model performance on test set were greater than on the train set observed in Table 5.1. Test results observed in figure 5.7 confirms that the model is able to predict patterns and peaks better, overestimating

or underestimating slightly.

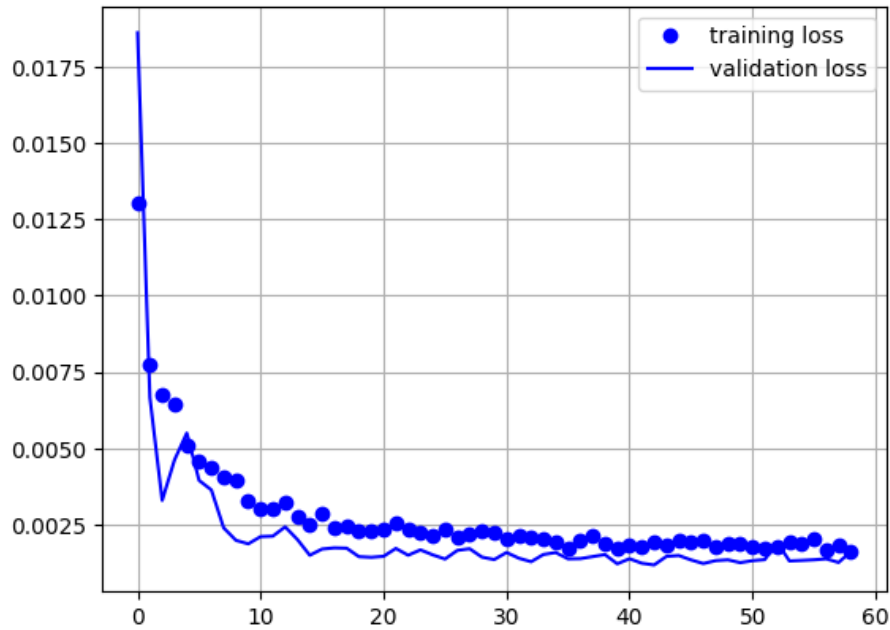


Figure 5.6: Training and validation loss curves of the best performing single step RNN model. Number of epochs along the x-axis and loss along the y-axis

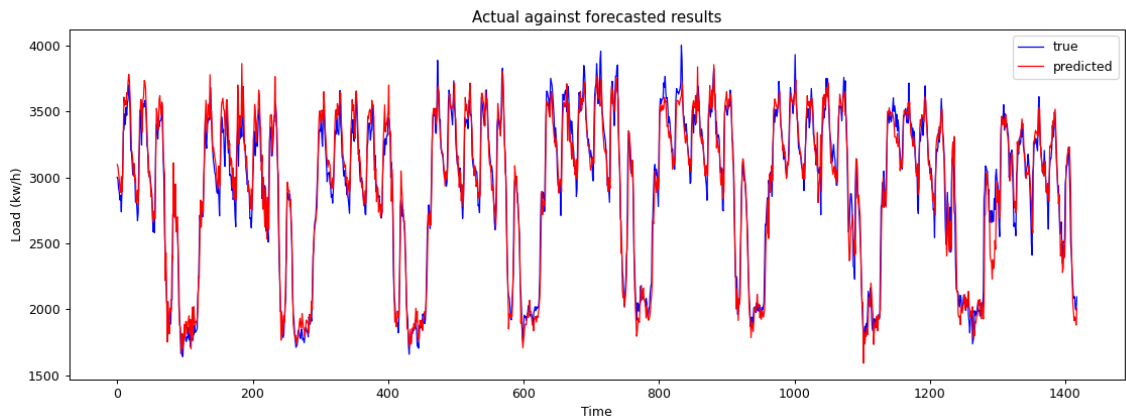


Figure 5.7: Best results of single step RNN model after tuning, hourly electrical consumption predictions made on test set. Hour along the x-axis and load (kw/h) along the y-axis.

5.3 Day-ahead recurrent neural network forecast (RNN) results

The day-ahead RNN model has been trained on the same data set as the single-step RNN model, now forecasting several steps ahead. Parameters outside of the model such as data transformations and LSTM input shape were fixed. Batchsize parameter fixed at 32 and lookback fixed at 336. The model achieving the lowest validation loss score during the hyperparameter tuning process are presented in Table 5.2.

The best set of hyperparameters from the model tuning process for day-ahead RNN included the LSTM layer with 300 units, a dropout layer with dropout rate of 0.3, a second LSTM layer with 50 units, the activation function in both layers were fixed to "tanh" and finally "Adam" optimizer with learning rate set to 0.00010105.

Table 5.2: Best performing day-ahead RNN model metrics

Data	MAE	MAPE	RMSE	R^2
Train	177.64	0.0669	265.73	0.83
Test	202.97	0.0782	268.41	0.78

Predictions made on the training set by the day-ahead RNN model are visualised in figure 5.8 and 5.9. For a closer look at the predictions see figure 5.9. One can notice from Figure 5.9 that the prediction quality is lower as the prediction time frame increases compared to the single-step RNN model. Despite not being able to properly predict peaks in the data, the model is able to follow the general trend to some extent.

The training and validation loss curves for the best performing day-ahead model are visualised in Figure 5.10. Validation loss is slightly lower than training loss as observed from the figure. Training was stopped after 49 epochs by the **EarlyStopping** callback due to inadequate improvement rate. Both training and validation curves in the figure show spikes throughout the training procedure while continuously decreasing.

The predictions on test set is visualised in Figure 5.11. The model performance on the test set is worse than on the training set. The model is not able to capture peaks, overestimating or underestimating target values. This means that the model is not able to generalize well to unseen data. Furthermore, this can be confirmed by performance

metrics shown in Table 5.2, the training set has better performance metrics as opposed to the test set. However, the model is still able to follow trends to some extent.

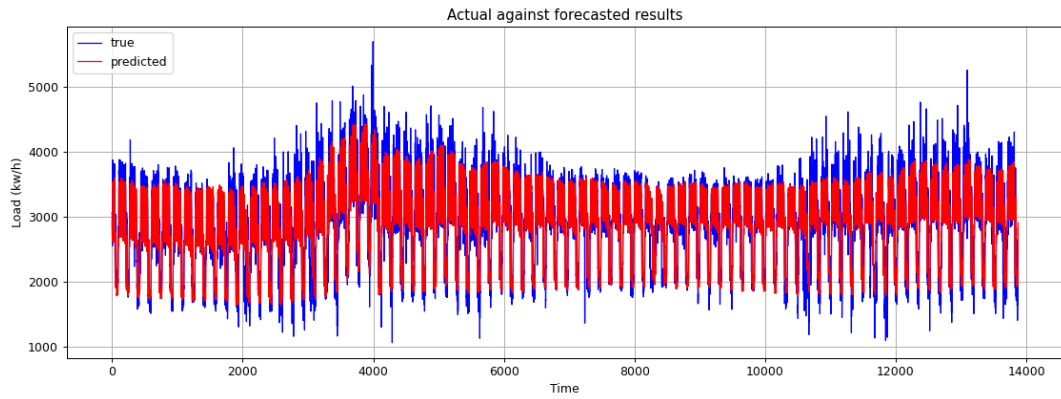


Figure 5.8: Best results of multi.step RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

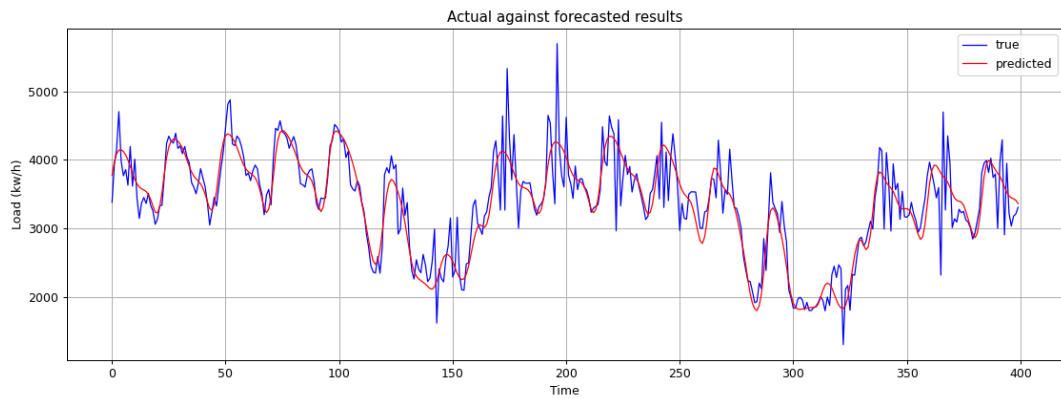


Figure 5.9: Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions made on train set plotted against true values, closer look at prediction plotting 2 weeks of data. Hour along the x-axis and load (kw/h) along the y-axis.

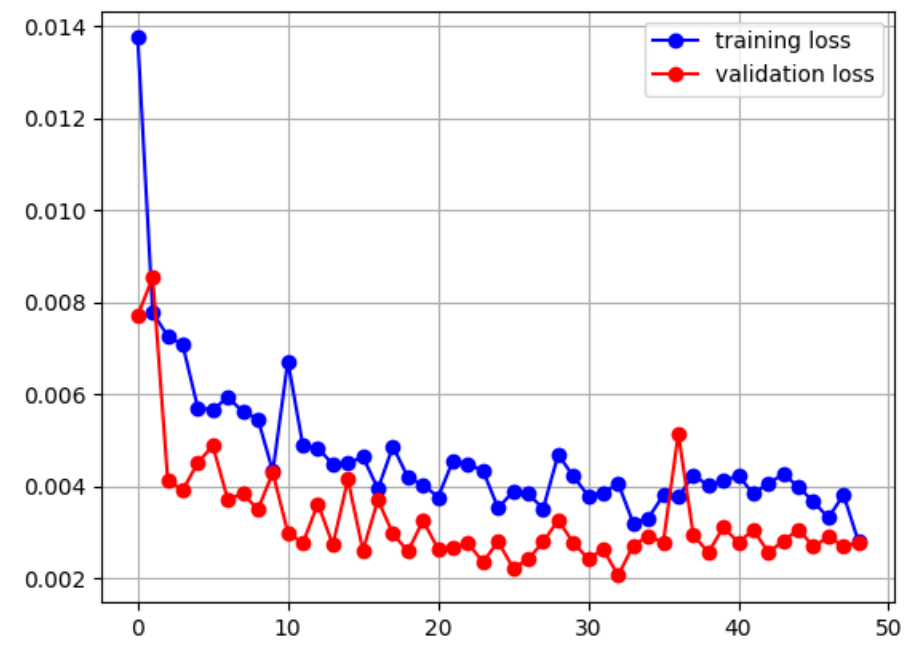


Figure 5.10: Training and validation loss curves of the best performing day-ahead RNN model, number of epochs along the x-axis and loss along the y-axis

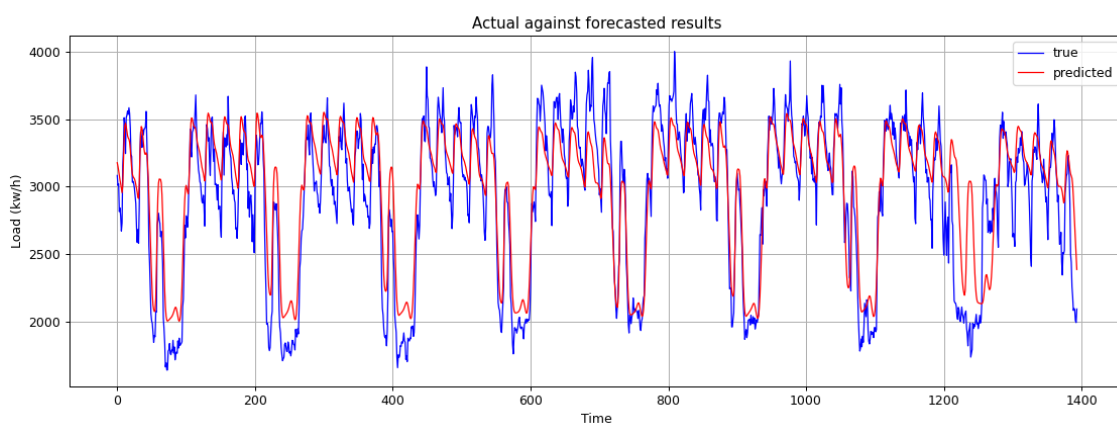


Figure 5.11: Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions made on test set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

5.4 Day-ahead recurrent neural network forecast (RNN) results on a single building (disaggregated data)

The results of hyperparameter tuning of the day-ahead RNN model are presented in Figure 5.12. This model is trained on a different dataset than the aggregated dataset for the 11 buildings. From the 11 buildings, the dataset for building number 1 was chosen to investigate if the predictions improve. Batchsize parameter was fixed at 32 and lookback at 336. The model achieving the lowest validation loss score during the tuning process are presented in table 5.3.

The best combination of hyperparameters derived from the tuning process include the LSTM layer with 256 units, a dropout layer with dropout rate of 0.5, a second LSTM layer with 4 units, activation in both these layers were held fixed to "tanh" and finally "Adam" optimizer with learning rate set to 0.0002776.

Table 5.3: Best performing day-ahead RNN model metrics for building 1

Data	MAE	RMSE	R^2
Train	24.37	37.35	0.82
Test	26.67	34.70	0.85

For a closer look at the training prediction made on the training set see Figure 5.13. The model experiences the same problems as the one trained on aggregated dataset. Observing Figure 5.13, the model is not able to predict peaks. However, it is able to follow the trend in the dataset to some extent.

The training and loss curves for the best performing day-ahead model for building 1 are presented in Figure 5.14. Loss decreased continuously, however, the validation loss exhibits spikes. The **EarlyStopping** callback stopped training after 30 epoch to prevent overfitting.

Predictions made on test set are visualised in Figure 5.15. The model display the same characteristics of the previous model tuned for aggregated dataset, not being able to capture peaks yet follows the trend to some degree.

Another experiment was conducted to investigate if the model tuned on one of the building

datasets is applicable for another building dataset. The results for predicting on the building 7 dataset with the same model parameters is visualised in Figure 5.16. It can be observed from Figure 5.16, the model produces poor predictions. As expected, the model trained on building 1 is unable to properly predict the load consumptions for another building due to the load consumptions between the buildings being very different.

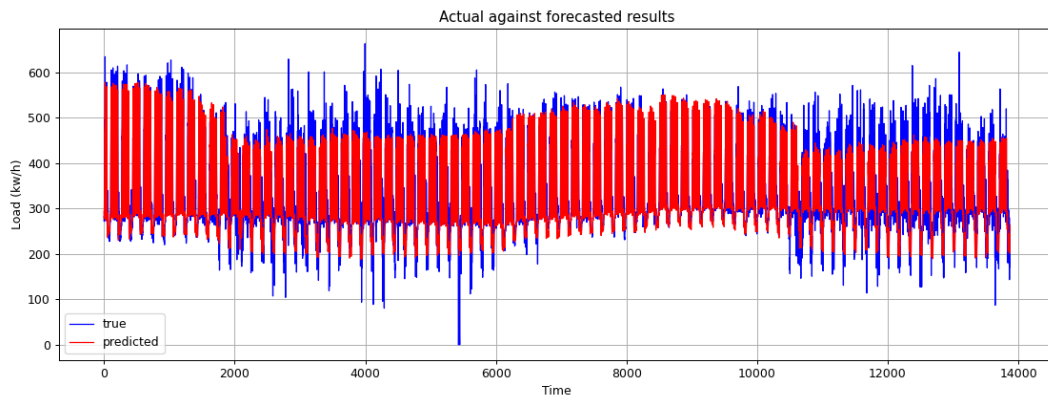


Figure 5.12: Best results of day-ahead RNN model after tuning, hourly electrical consumption for building 1 predictions made on train set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

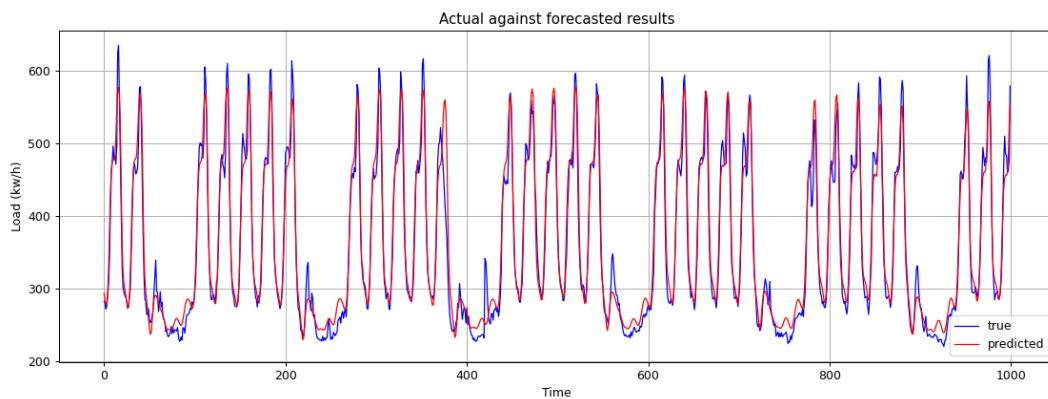


Figure 5.13: Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions for building 1 made on train set plotted against true values (closer look). Hour along the x-axis and load (kw/h) along the y-axis.

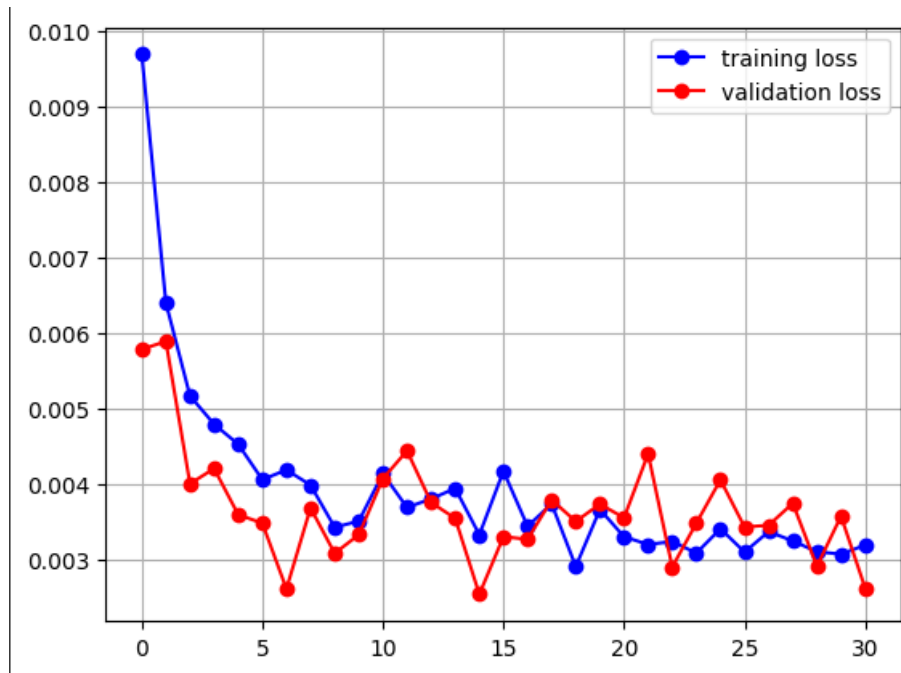


Figure 5.14: Training and validation loss curves of the best performing day-ahead RNN model trained of building 1 dataset, number of epochs along the x-axis and loss along the y-axis

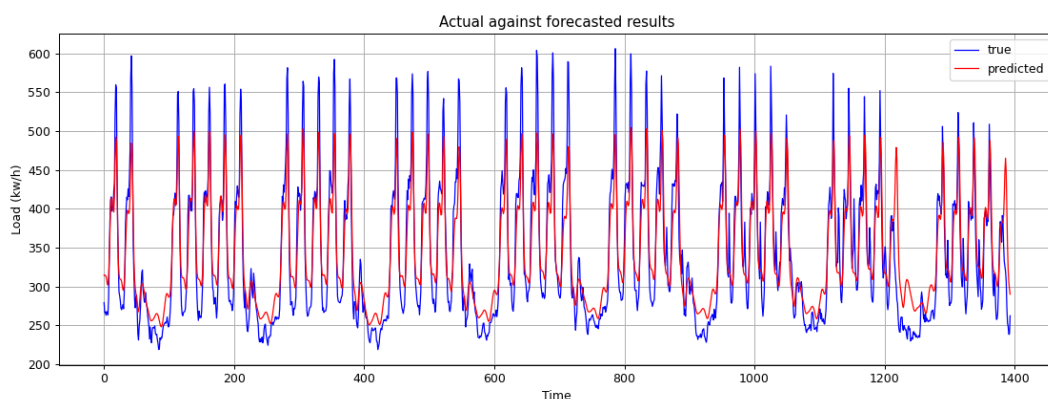


Figure 5.15: Best results of day-ahead RNN model after tuning, hourly electrical consumption predictions for building 1 made on test set plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

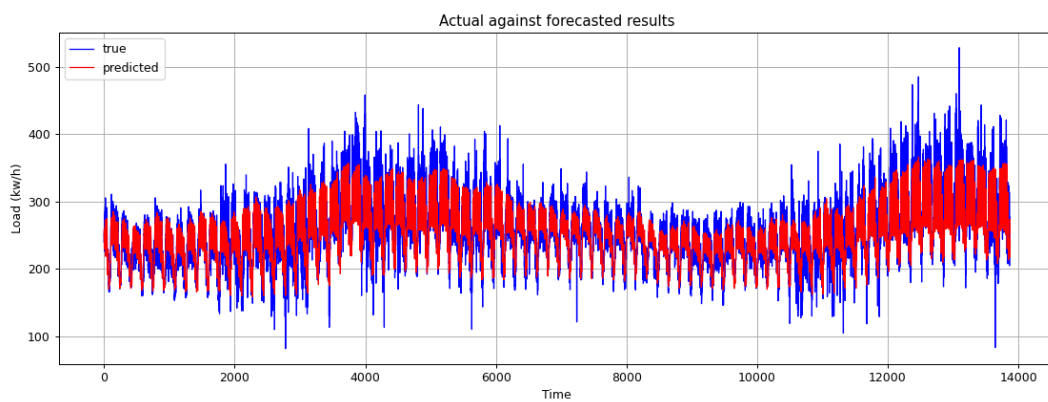


Figure 5.16: Best results of day-ahead RNN model after tuning, hourly electrical consumption for building 7 predictions plotted against true values. Hour along the x-axis and load (kw/h) along the y-axis.

6 Discussion

In the previous chapter, performance of various models were presented using performance metrics such as MAE, RMSE, MAPE and the R^2 score. Assessing model performance with error metrics alone may not be sufficient enough. One needs to observe the match between the predicted values and the observed values. Evaluating model train score versus the test score tells us if the model has been able to learn the trends in the data and if the model can generalize well. A good model would yield better test scores compared to training scores. Contrary, a better train score than test score indicates overfitting and leads to poor predictions.

The dataset had to be treated before it could be fed to the model. Duplicate rows that had the same time stamp were removed. The duplicate values chosen for removal were the ones not resembling the nearest preceding and/or succeeding values. Consecutive time steps are often highly correlated and these extreme values were therefore dropped. Furthermore, temperature values were only available in 10 minute resolution. These were resampled to hourly using the mean values. To further enrich information about seasonality several time feature variables were extracted from the date time index. Considering that the model may not be able to extract useful information other than the ascending order of the data points these had to be transformed. One choice was to create dummy variables, this would greatly increase the dimensionality of the dataset (several columns for one time feature variable). To limit this increase it was decided to use another approach which would leverage the seasonal information without only extracting the information of ascending data points. The time feature columns were transformed into sine and cosine values limiting the increase to two columns per time feature and leveraging useful information about the seasonal variability in the data.

6.1 Interpreting the results

The model that preformed the best in this project was the single-step RNN model. The train scores seem good and when predicting on the test set the scores improves. A desired outcome when developing machine learning models. However, observing Figure 5.4 the model is not able to predict peaks well. Looking at the plot, the model is able to predict

the colder months well. The warmer months contain more volatile time steps. This is probably due to ASKO facility having to use more power because of cooling equipment to e.g. cool frozen food products. Prediction accuracy drops for these months as the model is not able to properly capture these volatile values. This may explain the good test scores, note the prediction on the test set falls on a period that the model is able to predict well. The model predicts better on a period of time that maintains a constant trend. Furthermore, the loss plot presented in Figure 5.6 demonstrates a good fit. The loss curves decrease to a point of stability and there is a minimal gap between both curves. Good forecast are of importance to ASKO facility, they are to be used as baseline to create accurate flexibility bids. Inaccurate forecasts will lead to inaccurate flexibility bids. These flexibility bids may require forecast estimates several hours ahead. A model that predicts further into the future was therefore developed.

Considering the day-ahead forecasts, for a given hour t , the prediction are now made for hour $t + \text{shift}$ (shift = hours ahead to forecast in this case 24). The model predicts 24 hours into the future. Time series data points are often highly correlated with the previous time steps. Considering that the prediction time frame increases, the correlation between the previous time steps decreases. The dependancies between the previous time steps and the time step to be predicted becomes less clear and unstable leading to unstable predictions. Observing the day-ahead forecasts confirms this, the scores worsen compared to the single-step model. The results look similar to the previous model however, now much more inaccurate. It can be observed that the prediction performance drops significantly despite tuning the model predicting 24 hours ahead. Overall performance decreases as the prediction time frame increases even for the colder periods that the single-step model was able to predict well. The model struggles to capture peaks and valleys throughout training and test sets. The performance metrics for the training set is better than the test set, this may indicates that the model is overfitting and has become too familiar with the patterns and noise in the training set unable to generalize well to unseen data. Observing the loss plot presented in Figure 5.10, there is no clear indication of overfitting. The curve exhibits spikes and a small gap between the loss and validation loss can be observed closing towards the end. Based on the metrics, the model seem to overfit despite using techniques such as dropout and **EarlyStopping** to tackle the problem. The best model

predicting 24 hours ahead was therefore subjected to manual tuning in an attempt to reduce overfitting. Several approaches to reduce the complexity of the model were tested such as reducing the number of units in the layers, removing layers, adding more dropout and different learning rates. However, doing so would worsen model performance. Again, this might be due to the fact that the prediction time frame is increased leading to unstable predictions.

A model for predicting load consumption for one of the buildings was developed in an attempt to estimate if the forecasting performance increased. Judging by the results presented in 5.4 they improved albeit by small numbers. The model trained on the dataset for building 1 show improved metrics on both the train and test set and better test set results, this could be coincidental and retraining the model may show otherwise. It can be observed from Figure 5.13, that the model is able to predict peaks better. Looking at the whole plot presented in Figure 5.12, the model is exhibiting the same characteristics as the previous model. The model is not able to predict extreme values occurring during the summer months well, however, still able to make good predictions when the trend is constant. Furthermore, observing Figure 5.15 the model is able to predict peaks and the seasonal behaviour to some extent but not fully reaching any peak or valley.

6.2 Shortcomings of the work in this project

A few shortcomings of the current approach that may affect the model performance of the machine learning models come to mind. First of all the train, validation and test split. In this project the max possible amount of training data (90%) was chosen. The reason behind this choice was that we wanted the model to be able to have as much data as possible to train and learn the patterns in the data set. During warmer months the electrical load becomes more volatile. For the model to be able to see this type of variation twice during training 90% was chosen for training. Spikes in the validation loss curves can be observed for the day-ahead models in both Figure 5.10 and Figure 5.14. These spikes can be a direct consequence of the increased prediction time frame. However, several other reasons come to mind. These spikes may occur if the validation set is not representative for the whole training set. This means that the validation set has too few samples compared to the training set. In this case the model performance can suffer because the validation set does not provide enough information to evaluate the models ability to generalize during training. Finally, the spikes could be due to the batch size being set to 32.

For the work in this project the inputs features fed to the model were load bought, sold, produced, sine and cosine time feature variables and temperature values. The actual historical total load consumption was solely used as the target feature to be predicted and not included in the feature set of the model. It might be useful to supplement the input dataset with more weather related features than the outside temperature e.g. solar irradiance.

One of the disadvantages of deep learning models is that these model can become computationally expensive. The experiments in this project were conducted in an online jupyter notebook environment provided by Kaggle and Google Colaboratory to speed up the training and tuning process utilizing gpu runtimes. However, the runtime on the gpus were limited and the tuning process had to be adjusted accordingly. In the early stages of training and tuning, it was observed that computations were switched to local cpu runtime instead of the faster gpu runtime. This led to an increased training and tuning process, to counter this some parameters were held fixed in order to properly utilize the gpu runtime [34].

An experiment to estimate the model performance of a model trained and tuned on one of the datasets (on building 1) and applied on another (on building 7) was conducted. As expected a model trained and tuned on one of the datasets is not applicable to another dataset. The reason for this is that the load consumption profiles of all the buildings are very different. Some of the buildings have extremely low consumption while other have high consumption. Some have solar panels that produce electricity while other do not. If one were to consider predicting on a disaggregated level (each dataset separately), 11 different models have to be trained and tuned. Developing 11 different models for each of the datasets is a time consuming process.

Finally, it could be that the problem lies with the dataset. The dataset with only the historical load consumption of a building might not be sufficient enough for a model to learn and predict properly. Adding input variables that directly impact the target feature to be predicted can improve model performance significantly.

6.3 Further work

Starting with the train, validation and test split. A potential improvement of the current approach would be to investigate different data splits. The training, validation and test set are split chronologically, this results in the test set falling onto a period that has a constant trend compared to warmer months having more volatile time steps. As stated before it would be interesting to test the model performance on the periods it is not able to predict well. This can be done by increasing the dataset keeping two years for training and the remaining full year for testing to fully assess model performance. The work in this project handled time series data and the hold out method for cross validation was therefore used. However, a different cross validation technique could be explored such as cross-validation on a rolling basis described in [38]. The spikes in the validation curves for the day-ahead models could be due to a smaller validation set compared to training set. Increasing the validation set may improve model performance.

The work in this project did not include the target feature in the input feature set when training the model. The historical total load consumption was treated exclusively as the target to be predicted. Load bought, sold and produced were used to explain the total load consumption that served as the target. Including the target feature in the input

feature space may provide additional information to the model and thus improve model performance.

Further extension of the work in this project could be to change and/or expand the hyperparameter search space. This might yield a more robust model able to generalize better at the cost of requiring computational resources. Some of the hyperparameters such as the activation functions in the LSTM layers for the models were held fixed to properly utilize gpu runtimes. Additionally, it was not possible to use recurrent dropout in the LSTM layers (used for tackling overfitting problems). Using the proposed framework to automatically tune the hyperparameters with a more expansive search space may provide a more robust model. In the tuning process the max trial per hyperparameter combination was set to 1 to reduce training time. For a more robust model the tuning process should run a combination of hyperparameters more than once.

None of the models presented in this project were able to predict peaks and valleys well, especially with the increased prediction time frame. The problem could lie with the dataset, a potential extension would be to gather meaningful feature variables that explain the target value properly. An extension of the input feature space discussed during the work in this project was the gate activity at the facility. The gate activity at the facility is believed to be able to explain some of the load consumption for the buildings and might provide valuable information for the input to the models.

Finally, it could be worth exploring other forecasting approaches. Traditional statistical approaches such as ARIMA (Autoregressive integrated moving average). A novel deep learning approach combining convolutional layers with RNN LSTM layers to create a hybrid neural network described in [16].

7 Conclusion

The work in this project explored a deep learning approach, specifically a variant of recurrent neural network called long short-term memory network capable of learning long term dependancies. This model was applied to the problem of forecasting hourly total electrical load consumption for 11 industrial buildings belonging to ASKO, using the historical load consumption profiles consisting of load bought, sold, produced and total load consumption. Three modelling approaches were investigated - a single-step recurrent neural network, a day-ahead recurrent neural network predicting 24 hours ahead into the future and a day-ahead model predicting 24 hours ahead into the future for one of the buildings to estimate forecasting ability and compared against another. A framework for tuning the hyperparameters using a predefined search space of hyperparameter combinations was implemented. Despite subjecting the models to an extensive hyperparameter tuning process, none were able to predict periods of volatile time steps well. The best model developed was the single-step recurrent neural network reaching the R^2 score of 0.94, MAE of 104.69, MAPE of 0.0439 and RMSE of 149.09. Hence, the models developed may not provide sufficiently accurate predictions to be used as baseline estimations for flexibility bid creations even more so if the goal is to predict further into the future. However, the models developed show promising results when forecasting hourly electrical load consumption. Finally, potential improvements of the current approach and alternative approaches are suggested.

Bibliography

- [1] 2050 long-term strategy. URL https://climate.ec.europa.eu/eu-action/climate-strategies-targets/2050-long-term-strategy_en. Accessed : 2023-05-10.
- [2] Total energy consumption. URL <https://yearbook.enerdata.net/total-energy/world-consumption-statistics.html>. Accessed : 2023-05-10.
- [3] A european green deal. URL https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en. Accessed : 2023-05-10.
- [4] Electrification. URL <https://www.iea.org/reports/electrification>. Accessed : 2023-05-10.
- [5] Glossary:renewable energy sources. URL https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Renewable_energy_sources. Accessed : 2023-05-10.
- [6] Renewables. URL <https://www.iea.org/reports/renewables>. Accessed : 2023-05-10.
- [7] Evaluation of fast frequency reserves, 2018. URL <https://www.statnett.no/om-statnett/nyheter-og-pressemedlinger/Nyhetsarkiv-2018/fleksibelt-forbruk-bidrar-til-stabilitet-og-verdiskapning-i-det-nordiske-kraftsystemet/>. Accessed : 2023-05-10.
- [8] The power market - energifakta norge. URL <https://energifaktanorge.no/norsk-energiforsyning/kraftmarkedet/>. Accessed : 2023-05-10.
- [9] What is flexibility and how can it contribute to the power grid system?, . URL <https://blogg.sintef.no/sintefenergy-nb/smartgrids/hva-er-fleksibilitet-og-hvordan-kan-det-bidra-til-nytte-i-kraftsystemet/>. Accessed : 2023-05-10.
- [10] Local flexibility solutions in statnetts regulating power market. URL <https://www.statnett.no/for-aktorer-i-kraftbransjen/nyhetsarkiv/lokale-fleksibilitetslosninger-testes-i-statnetts-regulerkraftmarked/>. Accessed : 2023-05-10.
- [11] Mahmoud A. Hammad, Borut Jereb, Bojan Rosi, and Dejan Dragan. Methods and models for electric load forecasting: A comprehensive review. *Logistics Sustainable Transport*, 11:51–76, 02 2020. doi: 10.2478/jlst-2020-0004.
- [12] Advantages and disadvantages of deep learning. URL <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-deep-learning/>. Accessed : 2023-03-04.
- [13] Md. Rashidul Islam, Abdullah Al Mamun, Md. Sohel, Md. Lokman Hossain, and Md. Mofij Uddin. Lstm-based electrical load forecasting for chattogram city of bangladesh. In *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pages 188–192, 2020. doi: 10.1109/ESCI48226.2020.9167536.
- [14] Yizhen Wang, Ningqing Zhang, and Xiong Chen. A short-term residential load forecasting model based on lstm recurrent neural network considering weather features. *Energies*, 14(10), 2021. ISSN 1996-1073. doi: 10.3390/en14102737. URL <https://www.mdpi.com/1996-1073/14/10/2737>.

-
- [15] Xin Wang, Fang Fang, Xiaoning Zhang, Yajuan Liu, Le Wei, and Yang Shi. Lstm-based short-term load forecasting for building electricity consumption. In *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, pages 1418–1423, 2019. doi: 10.1109/ISIE.2019.8781349.
- [16] Shafiu Hasan Rafi, Nahid-Al-Masood, Shohana Rahman Deeba, and Eklas Hossain. A short-term load forecasting method using integrated cnn and lstm network. *IEEE Access*, 9:32436–32448, 2021. doi: 10.1109/ACCESS.2021.3060654.
- [17] Sebastian Raschka and Vahid Mirjalili. *Python machine learning : machine learning and deep learning with python, scikit-learn, and tensorflow 2 / Sebastian Raschka, Vahid Mirjalili*. Expert insight. Packt Publishing, Limited, Birmingham, third edition edition, 2019 - 2019. ISBN 9781789958294.
- [18] What are the differences between a shallow network and a deep network. URL <https://www.mlexpert.io/machine-learning/interview-questions/shallow-and-deep-networks>. Accessed : 2023-03-30.
- [19] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017. ISBN 1617294438.
- [20] Oliver Tomic. Applied machine learning ii, lecture notes: Dat300 - 5 - universal workflow of machine learning. November 2021.
- [21] Sebastian Ruder. An overview of gradient descent optimization algorithms. URL <https://arxiv.org/abs/1609.04747>.
- [22] The vanishing gradient problem - chi - feng wang. URL <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>. Accessed : 2023-01-10.
- [23] Time series forecast error metrics you should know, . URL <https://towardsdatascience.com/time-series-forecast-error-metrics-you-should-know-cc88b8c67f27>. Accessed : 2023-01-10.
- [24] Common loss functions in machine learning. URL <https://towardsdatascience.com/time-series-forecast-error-metrics-you-should-know-cc88b8c67f27>. Accessed : 2023-01-13.
- [25] Evaluation metrics for regression models, . URL <https://medium.com/analytics-vidhya/evaluation-metrics-for-regression-models-c91c65d73af>. Accessed : 2023-01-10.
- [26] What is a learning curve in machine learning. URL <https://www.baeldung.com/cs/learning-curve-ml>. Accessed : 2023-03-27.
- [27] Cyclical features encoding, it’s about time!, . URL <https://towardsdatascience.com/cyclical-features-encoding-its-about-time-ce23581845ca>. Accessed : 2023-03-15.
- [28] Frost api. URL <https://frost.met.no/index.html>. Accessed : 2023-03-14.
- [29] Minmaxscaler documentation. URL <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>. Accessed : 2023-03-15.
- [30] Timeseriesgenerator documentation. URL https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/TimeSeriesGenerator. Accessed : 2023-03-15.

- [31] Keras api documentation. URL https://keras.io/api/keras_tuner/. Accessed : 2023-03-15.
- [32] Grid search vs random search vs bayesian optimization. URL <https://towardsdatascience.com/grid-search-vs-random-search-vs-bayesian-optimization-2e68f57c3c46>. Accessed : 2023-03-16.
- [33] Keras hyperparameter documentation. URL https://keras.io/api/keras_tuner/hyperparameters/. Accessed : 2023-03-16.
- [34] Lstm layer documentation. URL https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM. Accessed : 2023-03-16.
- [35] Bayesian optimization tuner documentation. URL https://keras.io/api/keras_tuner/tuners/bayesian/. Accessed : 2023-03-16.
- [36] Earlystopping documentation. URL https://keras.io/api/callbacks/early_stopping/. Accessed : 2023-03-16.
- [37] Keras guide documentation. URL https://keras.io/guides/keras_tuner/getting_started/#query-the-results. Accessed : 2023-03-16.
- [38] Cross validation in time series. URL <https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4>. Accessed : 2023-04-15.



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitenskapelige universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway