NTNU

Norwegian University of
Science and Technology

Short-Term Electric Load Forecasting Using Artificial Neural Networks

Torfinn Skarvatun Tyvold

Master of Science in Physics and Mathematics
Submission date: June 2018
Supervisor: Bob Ohara, IMF
Co-supervisor: Christian Aleksander Jansen Oshaug, Nord-Trøndelag Elektrisitetsverk AS

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This thesis was written in fulfillment of the course TMA4900 Industrial Mathematics, Master's Thesis at the Norwegian University of Science and Technology (NTNU) during Spring 2018. I would like to thank Professor Bob O'Hara, my supervisor at NTNU, for guidance on writing the thesis and my co-supervisor Christian Aleksander Oshaug Jansen at Nord-Trøndelag Elektrisitetsverk for providing the data set that was used.

Torfinn Skarvatun Tyvold, June 2018

Abstract

Short-term forecasting of power consumption is an important tool for decision makers in the energy sector. Power load forecasting is a challenging multi-step ahead time series forecasting problem since the power load is dependent on a number of different factors, e.g. temperature, time of day, time of week and recent power consumption. The goal of this master thesis was to develop a model that predicts the power consumption in Nord-Trøndelag for each hour of the next day. Several different models based on artificial neural networks were developed and tested on a historical data set consisting of hourly observations of power loads and temperatures in Nord-Trøndelag from 2011 to 2017. The data set was provided by Nord-Trøndelag Elektrisitetsverk (NTE). The performance of the models was compared to NTE's current model and several other types of models that are commonly used for short-term load forecasting. The final proposed model is an ensemble average of the two best performing multilayer perceptrons tested and a time-varying linear regression model that uses Kalman filtering for weight estimation. The proposed model is very efficient in terms of time usage and a large improvement compared to NTE's current model.

Sammendrag

Prediksjon av strømforbruk er et viktig verktøy for beslutningstakere i energisektoren. Strømforbrukspredikering er et utfordrende problem ettersom strømforbruket er avhengig av en rekke ulike faktorer, b.a. temperatur, tidspunkt på dagen, hvilken ukedag det er og hva strømforbruket var de siste par dagene. Målet med denne masteroppgaven var å utvikle en metode som predikerer strømforbruket i Nord-Trøndelag for hver time neste dag. Flere ulike modeller basert på kunstige nevrale nettverk ble utviklet og testet på et historisk datasett som består av timeobservasjoner av strømforbruket i Nord-Trøndelag fra 2011 til 2017 og temperaturdata fra den samme perioden. Datasettet kommer fra Nord-Trøndelag Elektrisitetsverk (NTE). Nøyaktigheten til modellene ble sammenlignet med NTEs nåværende modell og et par andre typer modeller som ofte brukes til å produsere strømforbruksprognoser. Den foreslåtte modellen bruker gjennomsnittet av prediksjonene fra de to beste nevrale nettverkene som ble testet og en tidsvarierende lineær regresjon modell som bruker Kalmanfiltrering. Den foreslåtte modellen er tidsmessig effektiv og en stor forbedring sammenlignet med NTEs nåværende modell.

Contents

1 Introduction

In Norway electricity provision and distribution within a region was historically handled by the same company. In 1991 the power system was liberalized, meaning that consumers in Norway got to freely choose who their energy provider is, regardless of who owns the power distribution network in their local area. Today consumers in Norway pay two separate actors for electricity. They pay their power provider for the electricity they use and they pay their distribution network operator for the cost of transporting the electricity to their house. Most power companies in Norway still provide both services, but they are legally required to not give preferential treatment to their own electricity. For instance the distribution price is legally required to be the same regardless of who the power provider is.

In addition to the consumer market where power providers sell electricity to consumers there also exists a wholesaler market where power producers sell electricity to power providers. Most of this trading is done through Nord Pool, an energy exchange that operates in multiple countries in Northern Europe. Most of the trade on Nord Pool is done on the day-ahead market, which closes at noon each day. Before this deadline power producers and providers lay in bids to Nord Pool that specify how much power they are willing to sell or buy at different price levels for each hour of the next day. Nord Pool then calculates the price at each hour based on the supply and demand and settles the trades. Starting at midnight the power is then physically delivered from the power producers to the power providers according to the agreed upon contracts.

When the hour of delivery comes the power provider may discover that the customers are using dP more electricity than the power provider bought for that hour. This deficit then has to be made up for by buying electricity on the reserve market, also called the regulating power market, at a price R that is different from the price S of the day-ahead market. If $R > S$ the money lost for the supplier will be $dP \cdot (R–S)$, whereas if $R < S$ the supplier will earn the same amount. Similarly, if the power provider's customers use less electricity than anticipated the excess electricity has to be sold on the regulating power market, which will result in a net gain for the power provider if $R > S$ or a net loss if $R < S$. Although the power supplier may sometimes earn money by buying or selling on the reserve market instead of on the day-ahead market, on average they can expect to loose money by doing so and the larger the imbalance of the power provider is the more likely they are to loose money. Part of the reason for this is that the imbalance of the power provider impacts the imbalance of the market. E.g. if the power supplier has an excess supply of electricity that they need to sell on the reserve market, then the more excess electricity they have, the higher the total supply in the market becomes, which reduces the price. Because of this a power supplier ideally wants to predict the power load as accurately as possible. Since the amounts of electricity traded between power producers and providers is typically very large, even a relatively small reduction in the forecast error can result in large

financial savings for the company.

Nord-Trøndelag Elektrisitetsverk (NTE) acts both as a distribution network operator that runs the power grid in Nord-Trøndelag, an area that is populated by around 130.000 people, and as the primary power provider in the same area. The main goal of this thesis was to develop a model that accurately predicts the power consumption in Nord-Trøndelag for each hour of the next day based on information available on the morning of the current day. E.g. if the current time is Tuesday morning then we want to predict the power load on each hour of Wednesday. Ideally this model should be able to significantly outperform NTE's current model, which is based on Kalman filtering and is known to be inaccurate at times. Another goal of the thesis was to investigate whether a model based on neural networks would be a significant improvement to one based on Kalman filtering or not.

The data set used to test the different models was provided by NTE. It contains hourly observations of the power consumption in Nord-Trøndelag from 2011 to 2017 as well as weather data from the same period.

Power load forecasting can be viewed as a multi-step ahead time series forecasting problem. It is a challenging problem since power loads are not only heavily dependent on temperatures, but also exhibit both daily, weekly and yearly cyclical patterns. Special days like public holidays also have to be accounted for.

Section 2 of this thesis will give a detailed description of the data set provided by NTE, while section 3 will provide the necessary theoretical background to understand the models that are presented in section 4. Section 5 describes the test approaches used to test the different models and the results are reported in section 6. The results are discussed in section 7.

2 Data set

As mentioned in the introduction, the data set was provided by Nord-Trøndelag Elektrisitetsverk (NTE) and consists of hourly observations of the power consumption in Nord-Trøndelag, Norway from January 1st 2011 to December 31st 2017. The unit of measurement is MWh/h, meaning the average power load in MW over the preceding hour, although from now on it will simply be referred to as MW. The data set also contains weather data from the same period in the form of hourly measurements of temperature, wind speed and solar irradiation from three of the largest town in Nord-Trøndelag, namely Steinkjer, Stjørdal and Namsos. Additionally, since day-ahead forecasts must be made before the exact weather conditions of the next day are known, the data set also includes day-ahead weather forecasts made the morning before for the same three locations. Finally the data set also contains the day-ahead power load predictions made by NTE's current model, which is based on Kalman filtering. In total the data set contains 20 time series, each of which consists of 61.368 measurements. The data set is very close to complete, with only a dozen or so missing values from the power and temperature time series, which was rectified by using the average of the previous and subsequent measurements to fill in the gaps.

2.1 Cyclical patterns

Figure 1 shows the entire power load time series. The power load includes all public power usage in Nord-Trøndelag except for a handful of industrial actors. We observe that there is a strong yearly seasonality in the time series. Power consumption is much higher in the winter than in the summer. This is primarily because a lot more electricity is needed to heat buildings during the winter when the weather is cold.

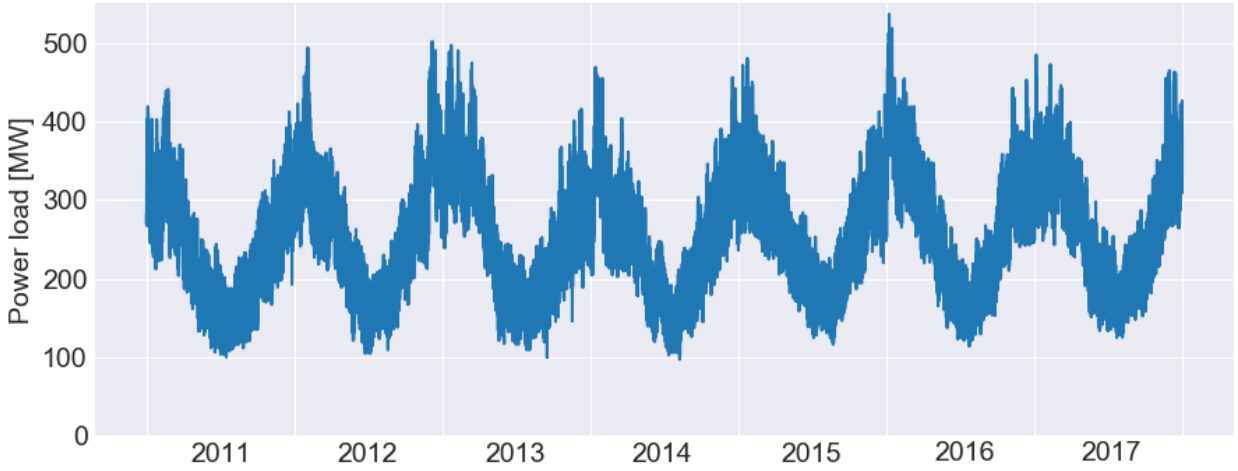

Figure 1: Hourly power consumption in Nord-Trøndelag from 2011 to 2017.

Note that there has been a very slight increasing trend in the power load over the seven years. In general power load patterns will slowly change over time due to changes in population size and consumer patterns, which is something that must be considered when fitting a forecasting model, although in the case of this data set the changes have been very small.

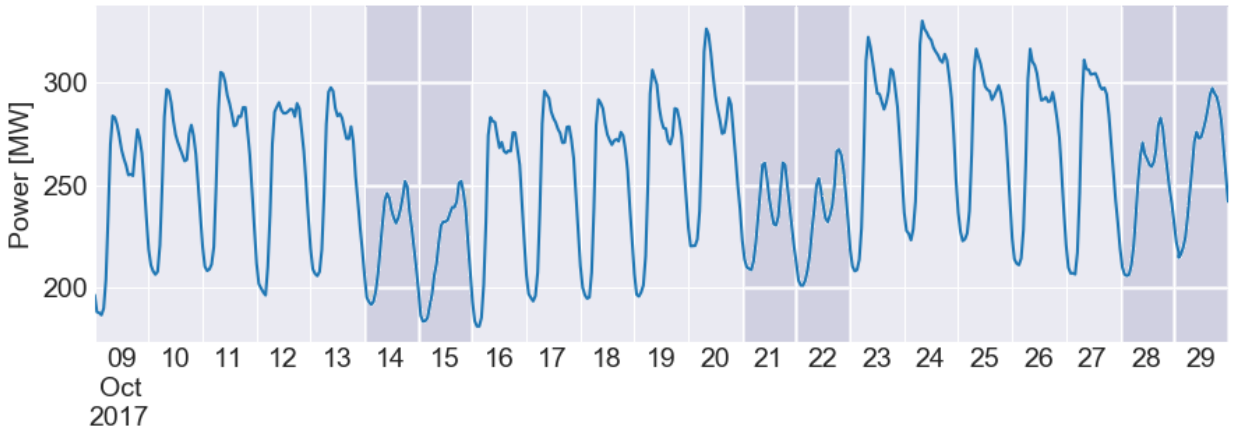

Figure 2: Close-up of the power load from Monday October 9th to Sunday October 30th, 2017. Weekends are shaded with a darker background than workdays.

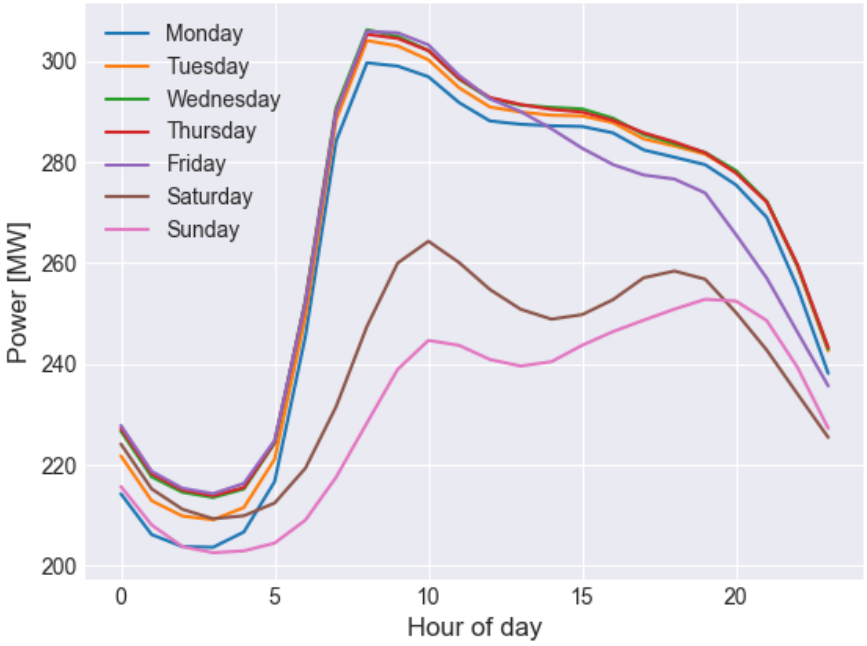

Figure 3: Mean daily power load pattern for each weekday over the entire data set.

Figure 2 shows a close-up of the power load from Monday October 9th to Sunday October

30th, 2017, while figure 3 plots the mean daily power load pattern for each weekday over the entire data set. We observe that in addition to the yearly cyclical pattern in the power loads there are also strong daily and weekly cyclical patterns. Power consumption is generally much higher in the daytime than during the night and higher on workdays than during the weekend. Another thing to note is that the shape of the daily power load curve changes throughout the year. In the winter half of the year workdays usually have a big spike in power consumption in the morning and a smaller one in the afternoon around the time people come home from work, whereas in the summer half of the year there is no second spike during the afternoon (see e.g. figure 4), which is why the shape of the power load curve on workdays is a bit different in figures 2 and 3.

2.2 Public holidays

An additional factor that complicates power demand forecasting is public holidays. There are 12 public holidays in Norway each year. Table 1 lists them all. Five of the holidays fall on the same date each year and thus the weekday they fall on varies from year to year, while the other seven are always a set amount of days from Easter Sunday and fall on the same weekday each year.

Public holiday	**Date**
New Year's Day	January 1st
Maundy Thursday	Three days before Easter Sunday
Good Friday	Two days before Easter Sunday
Easter Sunday	First Sunday after first new moon after spring equinox
Easter Monday	Day after Easter Sunday
Labor Day	May 1st
Norwegian Constitution Day	May 17th
Ascension Thursday	40 days after Easter Sunday
Whit Sunday	50 days after Easter Sunday
Whit Monday	51 days after Easter Sunday
First day of Christmas	December 25th
Second day of Christmas	December 26th

Table 1: List of Norwegian public holidays

Figure 4 shows a close-up of the power load from Monday April 29th to Monday May 21st, 2013. This period contains five public holidays, namely Labor Day on Wednesday May 1st, Ascension Thursday on May 9th, Constitution Day on Friday May 17th and Whit Sunday and Mon-

day on May 20th and 21st. As can be seen in the plot public holidays experience power demand curves similar to those of Saturdays and Sundays regardless of which day of the week they fall on.

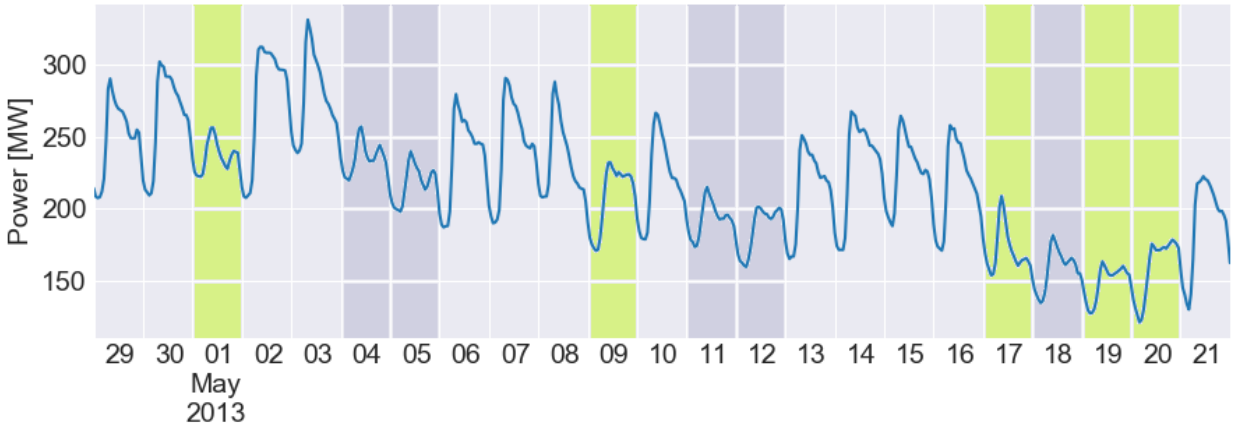

Figure 4: Close-up of power load from Monday April 29th to Tuesday May 21st, 2013. Weekends are shaded darker than workdays and public holidays are shaded green.

2.3 Temperature dependence

The temperature data that is included in the data set is the same temperature data that NTE's current model uses. NTE's current model uses a weighted average of the weather measurements from each site. The formula for the weighted average temperature time series is

$$T(t) = 0.51 \cdot T_{Stjørdal}(t) + 0.3 \cdot T_{Steinkjer}(t) + 0.19 \cdot T_{Namsos}(t). \tag{1}$$

These weights are motivated by the fact that Namsos is the least populous of the three towns and lies in the northern part of Nord-Trøndelag, which is less populated than the south, while Stjørdal is the most populous of the three and lies in the south. Figure 5 is a scatter plot that shows the relationship between the daily average temperature and the daily average power load over the entire data set. The temperatures used are those from formula 1, although the scatter plots for each of three individual temperature time series are practically identical. Note that only regular workdays (Mondays to Fridays) are shown due to the different load pattern exhibited during weekends. We observe that the relationship between power and temperature is relatively linear with power consumption decreasing as the temperature increases. However, upon closer inspection we observe that the relationship is a bit more complicated than that. In particular, if the temperature is already above $15°C$ then increasing it further will not have much of an impact on the power load. This is presumably because heating of buildings no longer becomes a factor. In fact, in countries with warmer climates than Norway it is common to see something closer to a reverse j-shape or even a u- or v-shape where power loads increase when air temperatures go

beyond room temperature because of the large amount of energy consumed by air conditioning systems in hot weather.

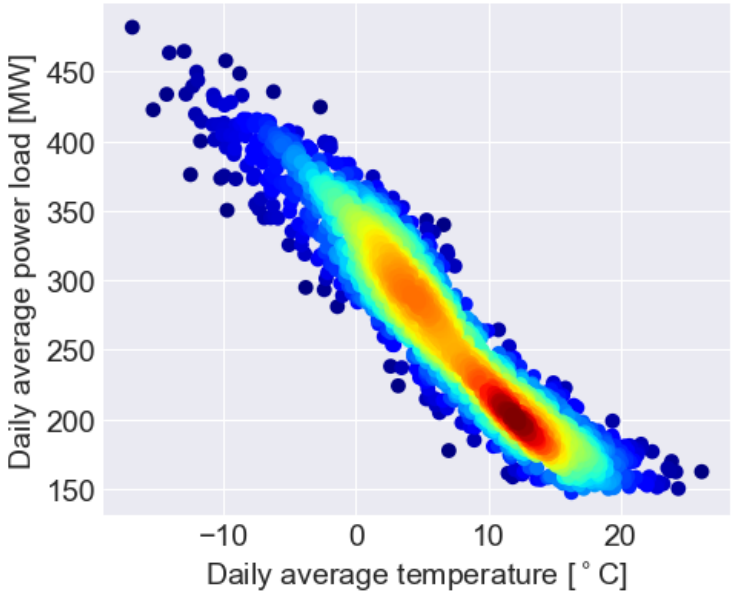

Figure 5: Scatter plot of daily average power loads versus daily average temperatures. Only regular workdays are included. Warmer colors indicate a higher density of observations.

2.4 Temperature forecasts

Figure 6 shows four histograms that display the residuals between the actual and predicted temperatures for each of the three locations in the data set as well as for the weighted average temperature time series. A positive residual means that the actual temperature is higher than the prediction. The temperature predictions were made the morning before at approximately 10 am. E.g. the temperature predictions for April 16th from 1 am to midnight were made on April 15th at 10 am. The forecasting residuals from Namsos and Steinkjer follow a normal distribution with a mean close to zero, but the forecasts from Stjørdal are a bit skewed. A somewhat surprising thing which is not shown in the figure is that the uncertainty of the forecasts does not increase significantly from 1 am to midnight even though the forecasts are made further and further into the future.

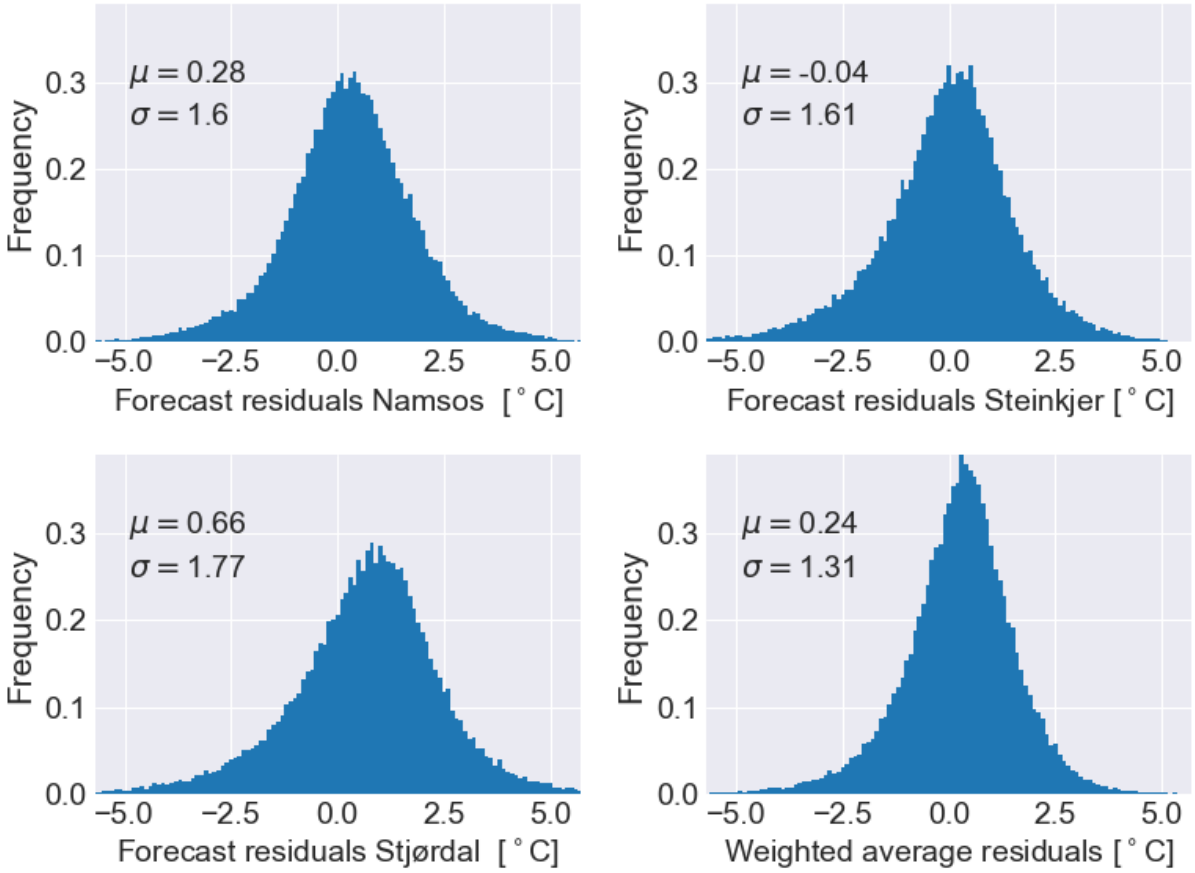

Figure 6: Histograms of the differences between predicted and actual temperatures for each of the three locations as well as the aggregate time series.

3 Background

3.1 Power load forecasting

According to Hong and Fan [1] approximately 2500 journal papers in the field of power load forecasting were published from 1970 to 2014. Out of these around 1300 papers were published between 2010 and 2014. The field of electric power load forecasting can be roughly grouped into three categories: point load forecasting, spatial/hierarchical load forecasting and probabilistic load forecasting.

Point load forecasting is usually grouped into short-term or long-term load forecasting, with short-term typically meaning that the forecast horizon is less than one-two weeks, although there is no universally agreed upon cut-off point between the two subcategories. Approximately 800 out of the 2500 papers found by by Hong and Fan were in the field of short-term point load forecasting, while around 1150 concerned long-term point load forecasting. Note that short-term load forecasting is sometimes further divided up into very short-term and short-term forecasting with very-short term typically meaning less than a day ahead.

Spatial/hierarchical load forecasting concerns methods that can be used in situations where the electric load can be split up into several geographical subareas, while probabilistic load forecasting deals with methods that forecast a probability distribution of the expected power demand. Around 300 of the papers found by Hong and Fan concerns the former, while around 250 concerns the latter. Although the data set used in this thesis contains weather data from several geographic locations the historical power loads are only available for the whole of Nord-Trøndelag. Thus spatial load forecasting methods will not be relevant in this case study. Generating a probability distribution for the power demand may sometimes be useful, but in the context of energy exchanges point forecasts is what we are primarily interested in, thus probabilistic forecasts will not be dealt with in this thesis.

Hong and Fan [1], Srivastava et. al. [2], and Chheepa and Manglani [3] give detailed reviews of the different types of models that are commonly used for short-term point load forecasting. They are often divided into two groups, namely statistical models and artificial intelligence-based models, although the distinction between the two groups is very loose. Common statistical models used include multiple linear regression, adaptive load forecasting models based on Kalman filtering, semi-parametric additive models and traditional time series models like ARIMA models and exponential smoothing. Although ARIMA and exponential smoothing models tend to perform well compared to other types of models in situations where temperature information is not available or where there is little temperature fluctuation they are in most situations ill-suited for short-term power load forecasting because they do not incorporate temperature information into the model. Instead ARIMAX models, an extension of ARIMA models

that are able to incorporate exogenous time series, are sometimes used. Among artificial intelligence methods artificial neural networks (ANN) have received the most attention, especially in later years as computing power has increased, although other methods such as support vector regression are also sometimes used. Multi-layer perceptrons are the most commonly used type of ANN. Recurrent neural network are a type of ANN that are particularly well suited to problems where the data is of a sequential nature. Despite this they have not been used much for load forecasting, although they have received some attention lately, see e.g. Bianchi et. al. [4]. Finally, hybrid methods that combine several different types of models are also commonly used.

3.2 Multi-step ahead time series forecasting strategies

One way to view short-term power load forecasting is as a multi-step ahead time series forecasting problem where we want to predict the next H observations $y_{t+1}, ..., y_{t+H}$ of a time series using the previous L observations $y_t, ..., y_{t-L+1}$ of the same time series and a vector of exogenous information $\mathbf{x}_t$. In this case the time series y is the power load and $\mathbf{x}_t$ is a vector of external factors that affect future power loads, e.g. past temperatures, forecasts of future temperatures, the current hour and the current day of the week.

Subsections 3.2.1 to 3.2.4 will present a number of general strategies that can be used to produce multi-step ahead time series forecasts.

3.2.1 Recursive strategy

Perhaps the simplest strategy for producing a H step ahead forecast is to fit a single model that produces one-step ahead forecasts and use this model recursively H times to produce the desired predictions. Stated more precisely: we fit a function f so that that the k-step ahead predictions are given by the formula

$$\hat{y}_{t+k} = \begin{cases} f(y_t, ..., y_{t-L+1}, \mathbf{x}_t) & \text{if } k = 1 \\ f(\hat{y}_{t+k-1}, ..., \hat{y}_{t+1}, y_t, ..., y_{t+k-L}, \mathbf{x}_t) & \text{if } 1 < k \leq L \\ f(\hat{y}_{t+k-1}, ..., \hat{y}_{t+k-L}, \mathbf{x}_t) & \text{if } L < k \leq H. \end{cases} \tag{2}$$

The biggest issue with this strategy is that it is sensitive to accumulating forecast errors, especially when H is large [5, p.7]. For instance, if the model has a slight tendency to overpredict in each step, then this overprediction might propagate leading to increasingly worse overpredictions the further ahead that we predict.

3.2.2 Direct strategy

Alternatively we can fit a separate function f_k for each of the future data points that we want to predict. That is, we fit the models

$$\hat{y}_{t+k} = f_k(y_t, ..., y_{t-L+1}, \mathbf{x}_t), \quad 1 \leq k \leq H. \tag{3}$$

This is called a direct strategy. The direct strategy avoids the error propagation problem that the recursive strategy can suffer from, but fitting the model is often more computationally expensive. Additionally the predicted data points will be conditionally independent, unlike the data points in the actual time series, since we are using a separate model to predict each future data point, which may lead to decreased accuracy in certain situations [5, p.8].

3.2.3 Multiple-input multiple-output (MIMO) strategy

Unlike the two first strategies where each fitted model only has a single ouput the idea of the MIMO strategy is to fit a single model that outputs all the predictions that we want. That is, we fit the model

$$\hat{y}_{t+1}, ..., \hat{y}_{t+H} = f(y_t, ..., y_{t-L+1}, \mathbf{x}_t). \tag{4}$$

This model does not have the accumulating error problem that the recursive strategy suffers from nor the conditionally independent predictions problem that the direct strategy suffers from. However, the fact that we are using only one model to predict all future data points may make the model less flexible than if we used a direct strategy [5, p.10].

3.2.4 Hybrid strategies

Finally, a hybrid of two or more of the above strategies can be used. E.g. we can use a direct-recursive hybrid strategy where a separate model is fitted for each time step that we want to predict, but the model that predicts $\hat{y}_{t+k}$ takes $\hat{y}_{t+k-1}$ as an input [5, p.8-9].

3.3 Supervised learning

In addition to choosing a strategy for producing the multi-step ahead forecasts we also need to choose the method for fitting the forecasting function f. One approach is to view the search for f as a supervised learning problem. Let $\{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$ be a training set that consists of N samples where each sample consists of a feature vector $\mathbf{x}_i$ and a corresponding response vector $\mathbf{y}_i$. E.g. if we are using the MIMO strategy from equation 4 then $\mathbf{x}_i = [y_t, ..., y_{t-L+1}, \mathbf{x}_t]^T$ and $\mathbf{y}_i = [y_{t+1}, ..., y_{t+H}]^T$. A supervised learning algorithm seeks a function f from the space of feature vectors to the space of response vectors. The function is chosen by solving the optimization problem

$$\underset{f \in F}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} E(\mathbf{y}_i, f(\mathbf{x}_i)), \tag{5}$$

where E is an error function and F is the hypothesis space, meaning that it is the space of possible functions f.

A supervised learning problem where the values in the response vector are discrete is called a classification problem. If the values are continuous it is called a regression problem. In the rest of this theory section we will focus solely on regression problems where $\mathbf{x}_i = [x_{i1}, ..., x_{im}] \in \mathbb{R}^m$ and $\mathbf{y}_i = [y_{i1}, ..., y_{in}] \in \mathbb{R}^n$. Common choices for the error function in regression problems include the mean squared error

$$\text{MSE} = \frac{1}{n} \sum_{j=1}^{n} (y_{ij} - f(\mathbf{x}_i))^2 \tag{6}$$

and the mean absolute error

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^{n} |y_{ij} - f(\mathbf{x}_i)|. \tag{7}$$

3.4 Multiple linear regression

Multiple linear regression (MLR) is a supervised learning model where the hypothesis space F is the space of all possible linear functions and the error function E is the mean squared error. MLR seeks a linear function

$$f(\mathbf{x}_i) = \beta_0 + x_{i1}\beta_1 + ... + x_{im}\beta_m = \mathbf{x}_i^T \boldsymbol{\beta} \tag{8}$$

that approximates $y_i \in \mathbb{R}$. For the sake of convenience the notation $\mathbf{x}_i = [1, x_{i1}, ..., x_{im}]^T$ is used. The parameters $\boldsymbol{\beta} = [\beta_0, ..., \beta_m]^T \in \mathbb{R}^{m+1}$ are found by minimizing the mean squared error

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2. \tag{9}$$

To derive an estimator for $\boldsymbol{\beta}$ first note that equation 8 can be rewritten using matrix notation as

$$f(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}, \tag{10}$$

where $f(\mathbf{X}) = [f(\mathbf{x}_1), ..., f(\mathbf{x}_N)]^T$ and $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$. Similarly equation 9 can be rewritten as

$$\text{MSE} = \frac{1}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}), \tag{11}$$

where $\mathbf{y} = [y_1, ..., y_N]^T$. The coefficients that minimize the MSE are found by setting the derivative of the MSE with respect to $\boldsymbol{\beta}$ equal to 0, as such:

$$\frac{\partial \text{MSE}}{\partial \boldsymbol{\beta}} = \frac{1}{N}(-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}) = 0 \tag{12}$$

and thus the value of $\boldsymbol{\beta}$ that minimizes the MSE is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{13}$$

[6, p.68-69].

3.5 Artificial neural networks

3.5.1 Single-layer perceptrons

Artificial neural networks (ANNs) are a large and important group of supervised learning models. One of the simplest types of ANN is the single-layer perceptron (SLP). A single-layer perceptron is a supervised learning model that fits a function of the form $f(\mathbf{x}_i) = g(\mathbf{b} + \mathbf{w}\mathbf{x}_i))$, where g is called the activation function and can be any arbitrary function, while $\mathbf{w} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^{n \times 1}$ are called the weight matrix and the bias vector and are parameters in the model. Note that a multiple linear regression model is a special case of single-layer perceptron where $n = 1$ and the activation function is the identity function $g(x) = x$. Just like MLR models the parameters are typically found by minimizing the mean squared error, although other error functions may also be used. However, unlike with MLR models there is in general no analytical formula that can be used to find the exact parameters that minimize the error function. Instead an optimization algorithm must be used to find an approximate solution. In section 3.5.4 one such algorithm, stochastic gradient descent (SGD), is described in detail. Two extensions of SGD, namely SGD with momentum and the Adam algorithm, will be shown in sections 3.5.6 and 3.5.7.

3.5.2 Multilayer perceptrons

A multilayer perceptron (MLP) is an artifical neural network that consists of several single-layer perceptrons arranged in succession, where the output of each SLP is used as input to the next one. Each SLP is said to constitute a layer. The final layer in the chain is called the output layer, whereas the others are called hidden layers. In particular: a multilayer perceptron with one hidden layer is a function of the form

$$f(\mathbf{x}_i) = g_2(\mathbf{b}^{(2)} + \mathbf{w}^{(2)} g_1(\mathbf{b}^{(1)} + \mathbf{w}^{(1)}\mathbf{x}_i)), \tag{14}$$

where g_1 and g_2 are the activation functions of the hidden and output layers respectively

and $\boldsymbol{\theta} = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\}$ are the parameters of the model.

Figure 7 shows an illustration of a multilayer perceptron with one hidden layer visualised as a directed computational graph. To the left in the graph there is one node for each of the m inputs $x_{i1}, ..., x_{im}$ to the MLP. The number of nodes in the hidden layer is equal to the output dimension of the first SLP in the chain. The input nodes and the nodes in the hidden layer are fully connected with one edge directed from each of the input nodes to each of the nodes in the hidden layer. Each edge between the two layers is associated with a weight $w^{(1)}_{jk}$, which corresponds to the element in row j and column k of $\mathbf{w}^{(1)}$. The output of node k in the hidden layer is $g_1(b^{(1)}_k + \sum_{j=1}^{m} w^{(1)}_{jk} x_{ij})$, where b_k is the k-th element of the bias vector $\mathbf{b}^{(1)}$. That is, each node in the hidden layer takes the weighted sum of all nodes that have an edge pointed at it and applies an activation function to this sum to produce its output. The next layer then performs a similar operation using the outputs of the hidden layer as input to produce the final outputs of the model, which in the figure are labeled $y_{i1}, ..., y_{in}$ [7, p.227-230].

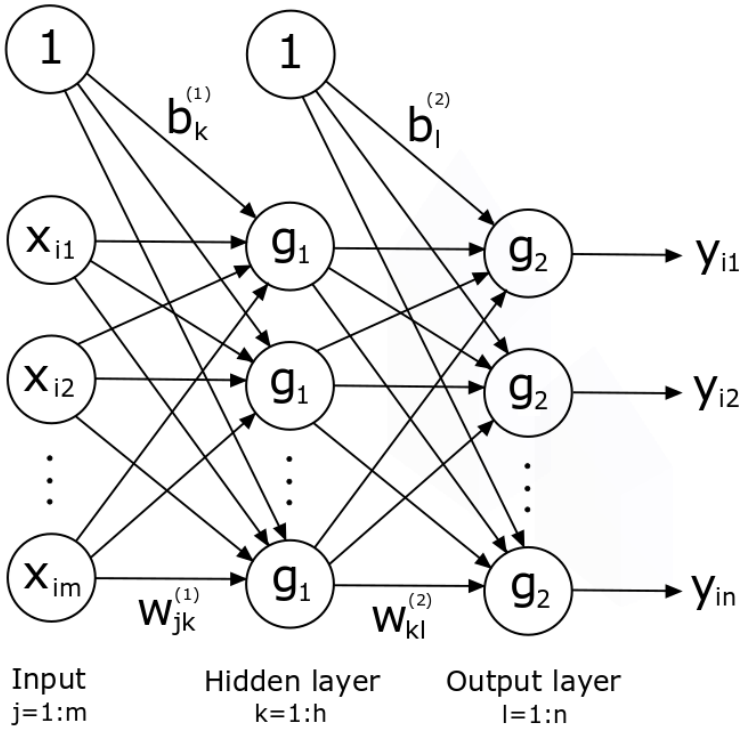

Figure 7: Illustration of a multilayer perceptron with one hidden layer.

Note that in the context of artificial neural networks the nodes are often referred to as neurons, while the edges are often referred to as synapses, due to vague superficial similarities they have with the neurons and synapses in the human brain. These superficial similarities is also where the name artificial neural network comes from.

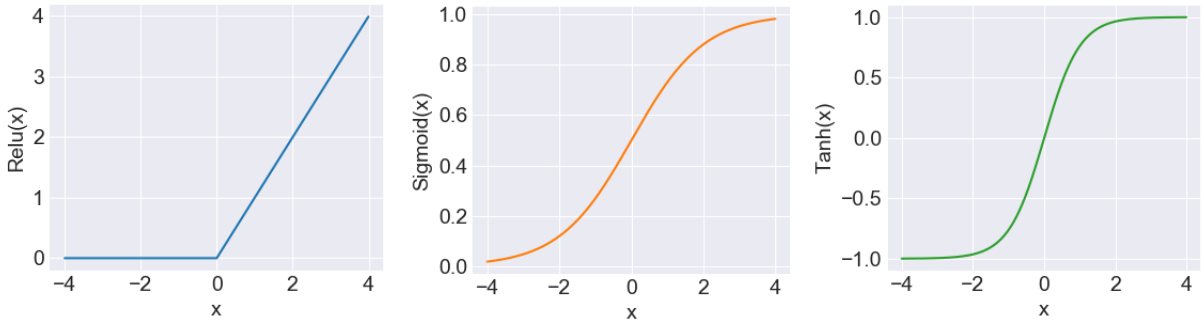

Figure 8: Plots of three commonly used activation functions. Left: rectified linear unit function, center: sigmoid function, right: hyperbolic tangent.

3.5.3 Activation functions

Figure 8 plots three of the most commonly used activation functions: the Sigmoid function $h(x) = \sigma(x) = e^x/(e^x + 1)$, the hyperbolic tangent $h(x) = \tanh(x) = (1 - e^{-2x})/(1 + e^{-2x})$ and the rectified linear unit function $h(x) = \max(0, x)$, commonly called the ReLU function.

There are several reasons why these particular activation functions are commonly used. First of, they are non-linear. From equation 14 it is obvious that if we were to only use linear activation functions in the hidden layers then the model would simplify to a single-layer perceptron. Thus the entire point of stacking several hidden layers after each other would be gone. Furthermore if we also use a linear activation function in the output layer then the model will simplify to ordinary linear regression and the fitted function will only be able to represent linear combinations of the input.

The second reason that these three functions are commonly used is that they all have derivatives that are very easy to calculate. The derivative of $\sigma(x)$ is $\sigma(x)(1 - \sigma(x))$, the derivative of $\tanh(x)$ is $1 - \tanh^2(x)$ and the derivative of the ReLU function is 0 if x is negative and 1 otherwise. As we shall see in sections 3.5.4 and 3.5.5, training a MLP involves calculating the derivatives of the outputs of the activation functions in the network with respect to different inputs many many times over. Thus to ensure that training times remain reasonably low it is very helpful if these gradients are easy to calculate. Note however that the ReLU function is slightly faster than the other two as it and its derivative consists only of a simple if-statement, whereas the Sigmoid and hyperbolic tangent functions require exponential functions to be computed.

The third reason is that all three of these functions squash their input to within a certain range. As can be seen in figure 8 the output of the Sigmoid function is always between 0 and 1, the output of the hyperbolic tangent is always between -1 and 1 and the output of the ReLU function is always non-negative. This can be useful both to ensure that the output of the model is within a certain range and to increase training stability.

Note that it is common to use different activation functions in different layers, but it is very

rare to use different activation functions in the same layer due to potential training instability problems that this may cause. Although linear activation functions cannot be used in the hidden layers, they are often used in the output layer in regression problems where we want the output to be able to represent any real number.

3.5.4 Stochastic gradient descent

Stochastic gradient descent (SGD) is the optimization algorithm that is most commonly used to fit the parameters $\boldsymbol{\theta}$ in multilayer perceptrons. SGD is an iterative algorithm where in each epoch the training data is randomly split into B non-overlapping and roughly equal-sized subsets called batches. Then for each batch the errors of the training examples in the batch are calculated and used to calculate the gradient of the total error of the batch with respect to the parameters $\boldsymbol{\theta}$. The algorithm then updates the parameters by taking a step in the direction of the gradient, where the step size is dependent on the hyperparameter η, which is called the learning rate. Pseudocode for the algorithm is shown below.

Algorithm 1 Stochastic gradient descent

```
Choose η (learning rate/step size)
Initialize parameters θ_0
Set initial iteration number: t = 0
for epoch e = 1 to n_epochs do
    Randomly split the training data into B batches of roughly equal size
    for batch b = 1 to B do
        Initialize the search direction: d_t = 0
        for element i = 1 to n_b in batch do
            Forward pass: Calculate the error E_i = E(f(x_i), y_i)
            Backward pass: Calculate the gradient of the error ∇_{θ_t} E_i
            Update the search direction: d_t = d_t + ∇_{θ_t} E_i
        end for
        Update the parameters: θ_t = θ_{t-1} − η d_t
    end for
end for
return θ_t
```

There are a few things that should be noted about the above algorithm. First off, in the special case where B is equal to 1 the algorithm simplifies to regular gradient descent. Secondly, some authors only use the term stochastic gradient descent when the batch size is equal to 1 and otherwise call the above method mini-batch gradient descent. Thirdly, if the size of the training set is not a multiple of the batch size then the last batch will be smaller than the others. E.g. if N, the training set size, is 1050 and the batch size B is 100, then the training set will be split into 10 batches with 100 elements each and one batch with 50 elements. Fourthly, note

that one epoch constitutes one forward pass of all the training examples and is different from an iteration, which constitutes one parameter update. E.g. if the training data is split into 11 batches, then each epoch consists of 11 iterations. Finally, the convergence criterion that was used in the pseudocode was that a certain number of epochs has been performed, but other convergence criteria can also be used, like stopping training when the total error of an epoch become lower than a certain threshold. [8, p.290-292]

3.5.5 Backpropagation

The most computationally expensive part of stochastic gradient descent is the backward pass where the partial derivatives of the error of a training example with respect to each of the weights in the neural network are calculated. The most common way to calculate the derivatives is to use the backpropagation algorithm. The backpropagation algorithm first calculates the partial derivatives of the error with respect to the weights in the output layer. Then it uses the chain rule to work its way backwards through the neural network layer by layer while computing the partial derivatives of the error with respect to the weights in each layer.

In the derivation below we will for the sake of simplicity assume that the error function used is the mean squared error. Let us number the layers from 1 to M, where layer 1 is the first hidden layer and layer M is the output layer. Furthermore let $a_k^{(L)}$ denote the input to the activation function of neuron k in layer L. The formula for $a_k^{(L)}$ is

$$a_k^{(L)} = \sum_{j=1}^{n_{L-1}} w_{jk}^{(L)} o_j^{(L-1)} + b_j^{(L)} = \sum_{j=0}^{n_{L-1}} w_{jk}^{(L)} o_j^{(L-1)}, \tag{15}$$

where n_{L-1} is the number of neurons in layer $L-1$ and $o_j^{(L-1)}$ is the output of neuron j in layer $L-1$. Note that in the second step we simplified the expression a little bit by writing $b_k^{(L)}$ as $w_{0k}^{(L)}$. The derivative of the above expression with respect to one of the network weights is

$$\frac{\partial a_k^{(L)}}{\partial w_{jk}^{(L)}} = o_j^{(L-1)}. \tag{16}$$

Using the chain rule we can now write the derivative of the error of training sample number i with respect to a certain weight in the neural network as

$$\frac{\partial E_i}{\partial w_{jk}^{(L)}} = \frac{\partial E_i}{\partial a_k^{(L)}} \cdot \frac{\partial a_k^{(L)}}{\partial w_{jk}^{(L)}} = \delta_k^{(L)} \cdot o_j^{(L-1)}, \tag{17}$$

where we have introduced the notation

$$\delta_k^{(L)} \equiv \frac{\partial E_i}{\partial a_k^{(L)}}. \tag{18}$$

The $o_j^{(L-1)}$'s can easily be computed by propagating the feature vector of the training sample forward through the network. Thus to compute 17 we now only need to find a way to calculate $\delta_k^{(L)}$ for all neurons in the network. The easiest ones to compute are those in the output layer. Let g_L denote the activation function of layer L. If the mean squared error is used as the error function then the error of training sample number i can be written as

$$E_i = \frac{1}{2}\sum_{k=1}^{n_M}(o_k^{(M)} - y_{ik})^2 = \frac{1}{2}\sum_{k=1}^{n_M}(g_M(a_k^{(M)}) - y_{ik})^2 \tag{19}$$

and thus

$$\begin{aligned}\delta_k^{(M)} = \frac{\partial E_i}{\partial a_k^{(M)}} &= (g_M(a_k^{(M)}) - y_k)\cdot g_M'(a_k^{(M)}) \\ &= (o_k^{(M)} - y_k)\cdot g_M'(a_k^{(M)}).\end{aligned} \tag{20}$$

To calculate the $\delta_j^{(L)}$'s in the other layers we first use the chain rule to find that

$$\delta_j^{(L)} = \frac{\partial E_i}{\partial a_j^{(L)}} = \sum_{k=1}^{n_{L+1}}\frac{\partial E}{\partial a_k^{(L+1)}}\frac{\partial a_k^{(L+1)}}{\partial a_j^{(L)}} = \sum_{k=1}^{n_{L+1}}\delta_k^{(L+1)}\frac{\partial a_k^{(L+1)}}{\partial a_j^{(L)}} \tag{21}$$

for $1 \leq k < m$. To find the partial derivative at the end of the last expression we observe from equation 15 that

$$a_k^{(L+1)} = \sum_{j=0}^{n_L} w_{jk}^{(L+1)}\cdot o_j^{(L)} = \sum_{j=0}^{n_L} w_{jk}^{(L+1)}\cdot g_L(a_j^{(L)}) \tag{22}$$

and thus

$$\frac{\partial a_k^{(L+1)}}{\partial a_j^{(L)}} = w_{jk}^{(L+1)} g_L'(a_j^{(L)}). \tag{23}$$

By plugging this into equation 21 we get the backpropagation formula

$$\delta_j^{(L)} = g_L'(a_j^{(L)})\sum_{k=1}^{n_{L+1}}\delta_k^{(L+1)} w_{jk}^{(L+1)}. \tag{24}$$

The backpropagation algorithm consists of two main parts. The first part is called the forward pass and is shown in algorithm 2 and the second part is called the backward pass and is shown in algorithm 3. In the forward pass the feature vector $\mathbf{x}_i = [x_{i1}, ..., x_{im}]^T$ of a training sample is propagated through the network and the inputs $a_k^{(L)}$ and outputs $o_k^{(L)}$ of all the neurons in the network are calculated using equation 15 and the formula $o_k^{(L)} = g_L(a_k^{(L)})$, starting with the first hidden layer and proceeding forward layer by layer to the output layer. Then in the back-

ward pass the algorithm first calculates $\delta_j^{(L)}$ for all neurons in the output layer using equation 20, then it uses equation 24 to work its way backwards through the network layer by layer calculating the $\delta_j^{(L)}$'s for all neurons in each layer until it reaches the input layer. Finally, now that all the $\delta_j^{(L)}$'s and $o_j^{(k)}$ have been calculated, the partial derivatives of the error function with respect to each layer in the neural network can easily be computed from formula 17.

For batch methods the derivatives of the total batch error E_b can be calculated by repeating the above algorithm for all the training examples in the batch and summing over the errors E_i of each training sample, in other words by using the formula

$$\frac{\partial E_b}{\partial w_{jk}^{(L)}} = \sum_{i=1}^{n_b} \frac{\partial E_i}{\partial w_{jk}^{(L)}} \tag{25}$$

[7, p.241-245] [9].

Algorithm 2 Forward pass in backpropagation algorithm

for neuron $k = 1$ to n_1 **do**
  $a_k^{(1)} = \sum_{j=0}^{m} w_{jk}^{(1)} x_{ij}$
  $o_k^{(1)} = g_1(a_l^{(1)})$
end for
for layer $L = 2$ to M **do**
  for neuron $k = 1$ to n_L **do**
    $a_k^{(L)} = \sum_{j=0}^{n_{L-1}} w_{jk}^{(L)} o_j^{(L-1)}$
    $o_k^{(L)} = g_L(a_k^{(L)})$
  end for
end for

Algorithm 3 Backward pass in backpropagation algorithm

for output neuron $j = 1$ to n_M **do**
  $\delta_j^{(M)} = (o_j^{(M)} - y_j) \cdot g'_M(a_j^{(M)})$
  $\partial E_i / \partial w_{jk}^{(M)} = \delta_j^{(M)} o_k^{(M)}$
end for
for layer $L = M - 1$ to 1 **do**
  for neuron $j = 1$ to n_L **do**
    $\delta_j^{(L)} = g'_L(a_j^{(L)}) \sum_{k=1}^{n_{L+1}} \delta_k^{(L+1)} w_{jk}^{(L+1)}$
    $\partial E_i / \partial w_{jk}^{(L)} = \delta_j^{(L)} o_j^{(L)}$
  end for
end for

3.5.6 SGD with momentum

The black line in figure 9 illustrates a weakness of regular stochastic gradient descent. When navigating through a ravine (an area where the gradient of the error is much steeper in one dimension than in another) the algorithm will oscillate from one side of the ravine to the other, while only slowly making progress towards the bottom.

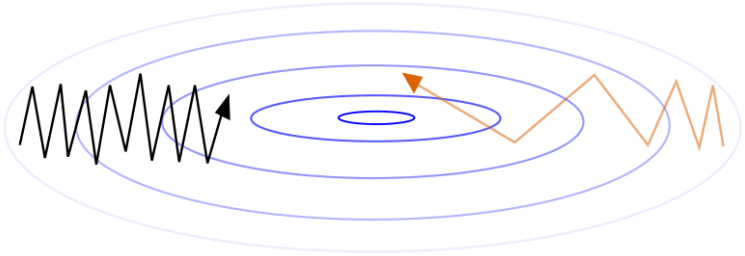

Figure 9: Comparsion of convergence of stochastic gradient descent with and without momentum. The black line illustrates regular SGD and the orange line illustrates SGD with momentum. Darker contour lines indicate lower errors. Note that the effect of adding momentum is exaggerated for illustrative purposes.

One way to deal with this problem is to introduce momentum [8, p.292-296] into the algorithm. Let t denote the current iteration. For regular stochastic gradient descent the update vector is

$$\mathbf{m}_t = \eta \nabla_{\boldsymbol{\theta}_t} E(\boldsymbol{\theta}_t), \tag{26}$$

while for stochastic gradient descent with momentum it is

$$\mathbf{m}_t = \gamma \mathbf{m}_{t-1} + \eta \nabla_{\boldsymbol{\theta}_t} E(\boldsymbol{\theta}_t), \tag{27}$$

where the hyperparameter γ is called the momentum term. In both cases the new parameters are given by the equation

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \mathbf{m}_t. \tag{28}$$

The potential benefit and main drawback of adding momentum are both shown in figure 9 by the orange line. Since the x-component of the update vector points in the same direction each step the learning algorithm gradually builds up momentum in the x-direction leading to faster convergence. A drawback of this is that adding momentum to SGD can make the model prone to overshooting its target if it builds up too much momentum.

3.5.7 Adam algorithm

Algorithm 4 Adam algorithm

Choose exponential decay rates $\beta_1, \beta_2 \in [0, 1]$
Choose learning rate η (≥ 0)
Choose ϵ (≥ 0)
$\mathbf{m}_0 = \mathbf{0}$ (Initialize 1st moment vector)
$\mathbf{v}_0 = \mathbf{0}$ (Initialize 2nd moment vector)
$t = 0$ (Set initial iteration number)
Initialize the parameters $\boldsymbol{\theta}_0$
for epoch $e = 1$ to n_epochs **do**
  Randomly split the training data into B batches of roughly equal size
  for batch $b = 1$ to B **do**
    $t = t + 1$ (Increment iteration number)
    $\mathbf{g}_t = \mathbf{0}$ (Initialize batch gradient)
    for training example $i = 1$ to n_b in batch **do**
      Calculate $E_i = E(f(\mathbf{x}_i), \mathbf{y}_i)$ (Forward pass)
      Calculate $\nabla_{\boldsymbol{\theta}_t} E_i$ (Backward pass)
      $\mathbf{g}_t = \mathbf{g}_t + \nabla_{\boldsymbol{\theta}_t} E_i$ (Update batch gradient)
    end for
    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$ (Update biased 1st moment estimate)
    $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \boldsymbol{\beta}_2)\mathbf{g}_t^2$ (Update biased 2nd moment estimate)
    $\hat{\mathbf{m}}_t = \mathbf{m}_t / (1 - \beta_1^t)$ (Compute bias-corrected 1st moment estimate)
    $\hat{\mathbf{v}}_t = \mathbf{v}_t / (1 - \beta_2^t)$ (Compute bias-corrected 2nd moment estimate)
    $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$ (Update parameters)
  end for
end for
return $\boldsymbol{\theta}_t$

The adaptive moment estimation algorithm, popularly called the Adam algorithm [10], takes the idea behind SGD with momentum one step further by constantly keeping track of exponentially decaying averages of previous gradients $\mathbf{m}_t$ and squared gradients $\mathbf{v}_t$. These moving averages are estimates of the 1st moment (the mean/momentum) and the 2nd moment (the variance) of the gradient. The details are shown in algorithm 4. When the algorithm starts the moment estimates are initialized to zero vectors. This causes the moment estimates to be biased towards 0, especially during the first few iterations and when the decay rates β_1 and β_2 are close to 1. Thus each iteration the moment estimates have to be corrected for bias. The derivation of the

bias correction formulas will not be shown here, instead see [10, p.3] for a full proof. The step size in direction i is the learning rate η times the i-th component of the biased mean vector estimate $\hat{\mathbf{m}}$ divided by the square root of the i-th component of the biased variance vector estimate $\hat{\mathbf{v}}$ plus a small constant ϵ. The basic idea behind dividing the moment by the standard deviation is that the smaller the variance is in one direction, the larger the step taken in that direction is. The purpose of ϵ is to prevent the step size from becoming infinitely large when the variance is close to zero. The authors of the Adam algorithm suggest using $\epsilon = 10^{-8}, \eta = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.999$ as default settings for the different hyperparameters. [10, p.2]

3.5.8 Recurrent neural networks

As mentioned in section 3.5.2, a neural network can be thought of as a directed computational graph. A directed computational graph contains a cycle if there exists at least one node that is reachable from itself, meaning that by starting in that node and following arrows it is possible to return to the same node. A neural network without any cycles is called a feedforward neural network. By looking at figure 7 we can see that multilayer perceptrons are a type of feedforward neural network.

A recurrent neural network (RNN) is a neural network that contains at least one cycle. The cycles allow recurrent neural networks to have internal state or "memory". Recurrent neural networks are commonly used to solve problems where the data has a sequential property such as machine translation, speech recognition, handwriting recognition or time series forecasting.

3.5.9 Elman and Jordan RNNs

The two most basic types of RNNs are Elman and Jordan networks, both of which are commonly refered to as simple RNNs. Let $\{\mathbf{x}_t\}_{t=1}^N$, where $\mathbf{x}_t \in \mathbb{R}^m$, and $\{\mathbf{y}_t\}_{t=1}^N$, where $\mathbf{y}_t \in \mathbb{R}^n$, be two time series and assume that we want to fit a function $\hat{\mathbf{y}}_t = f(\{\mathbf{x}_i\}_{i=1}^t)$, meaning that we want to predict the value of time series $\mathbf{y}$ at time step t based on the values of time series $\mathbf{x}$ at all time steps up to and including t. The basic idea of an Elman RNN is to fit a multilayer perceptron with one hidden layer from $\mathbf{x}_t$ to $\mathbf{y}_t$ with the twist that the hidden layer $\mathbf{h}_{t-1}$ from the previous time step is also fed into the hidden layer of the current time step $\mathbf{h}_t$. The formula for the hidden layer in an Elman RNN is thus

$$\mathbf{h}_t = g_h(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}_h), \tag{29}$$

where $\mathbf{h}_t \in \mathbb{R}^h, \mathbf{U} \in \mathbb{R}^{mxh}, \mathbf{W} \in \mathbb{R}^{hxh}$ and $\mathbf{b}_t \in \mathbb{R}^h$, while the formula for the output layer is

$$\mathbf{y}_t = g_y(\mathbf{V}\mathbf{h}_t + \mathbf{b}_y), \tag{30}$$

where $\mathbf{V}_t \in \mathbb{R}^{hxn}$. Figure 10 shows an illustration of an Elman RNN. Note that to avoid clutter each node in the graph represents a single layer, not a single neuron, which is different from the notation used in figure 7.

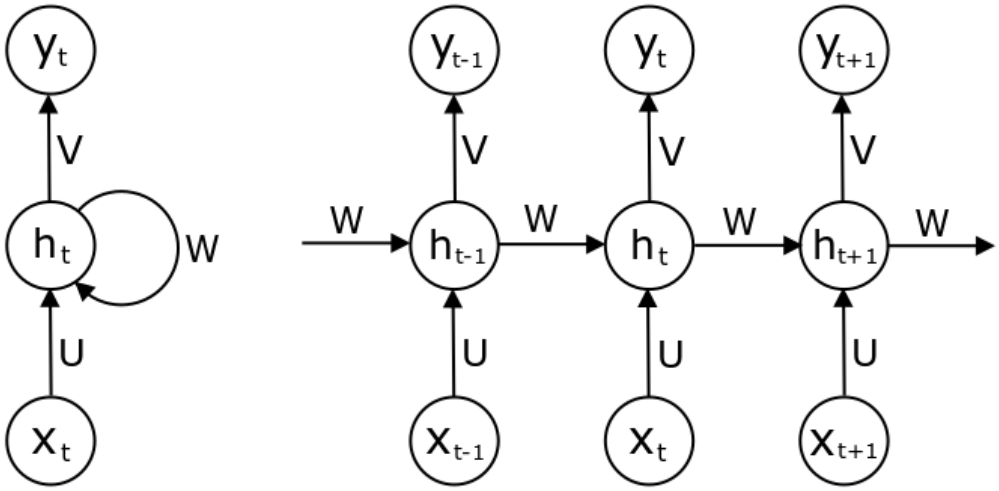

Figure 10: Left: Illustration of an Elman RNN. Right: Unrolled version of the illustration to the right.

Jordan RNNs are similar to Elman RNNs, but with one twist. While Elman RNNs feed the output of the hidden layer of the previous timestep into the hidden layer of the current timestep, Jordan RNNs instead feed the output of the output layer of the previous timestep into the hidden layer of the current timestep. Thus the formula for the hidden layer in a Jordan RNN is

$$\mathbf{h}_t = g_h(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{y}_{t-1} + \mathbf{b}_h), \tag{31}$$

where $\mathbf{W}$ now has the dimension $\mathbb{R}^{nxh}$, while the formula for the output layer is the same as for an Elman RNN.

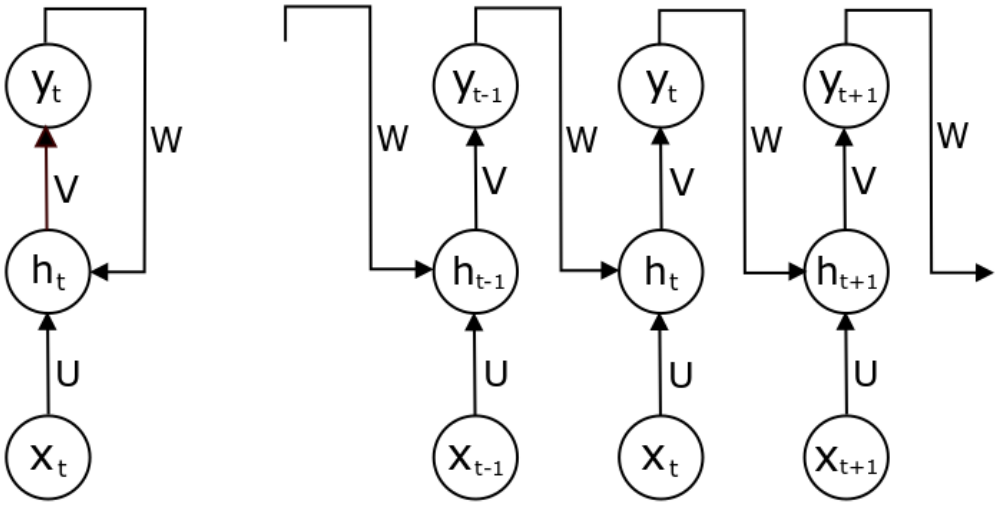

Figure 11: Left: Illustration of a Jordan RNN. Right: Unrolled version of the illustration to the right.

Recurrent neural networks can be trained by unrolling them into feedforward neural networks as shown in figures 10 and 11 and training them just like you would for a many-layered feedforward neural network.

3.5.10 LSTM RRNs

Although Elman and Jordan RNNs theoretically have the ability to remember information over an infinite number of timesteps, in practice they struggle with remembering long-term dependencies. Long short-term memory (LSTM) networks are a type of RNN that were designed primarily with this problem in mind.

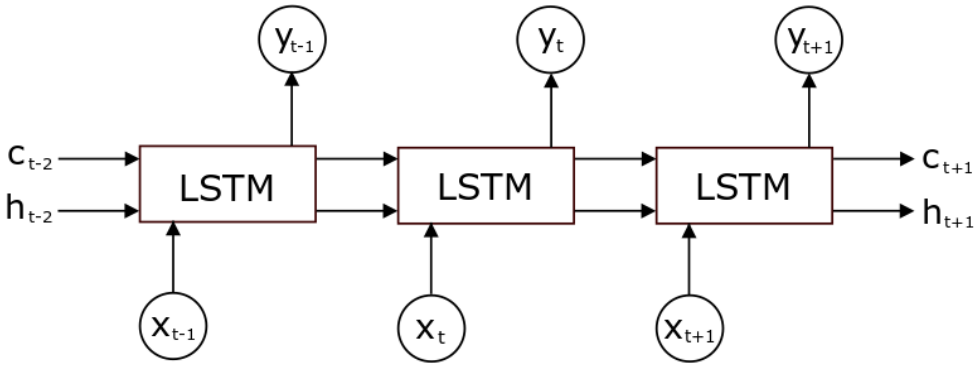

Figure 12: Illustration of an unrolled LSTM recurrent neural network.

Figure 12 shows an illustration of an unrolled LSTM recurrent neural network while figure 13 shows the inside of a LSTM cell. The key difference between a LSTM and an Elman network is the addition of the cell state $\mathbf{c}_t \in \mathbb{R}^h$. The cell state acts as a motorway where information

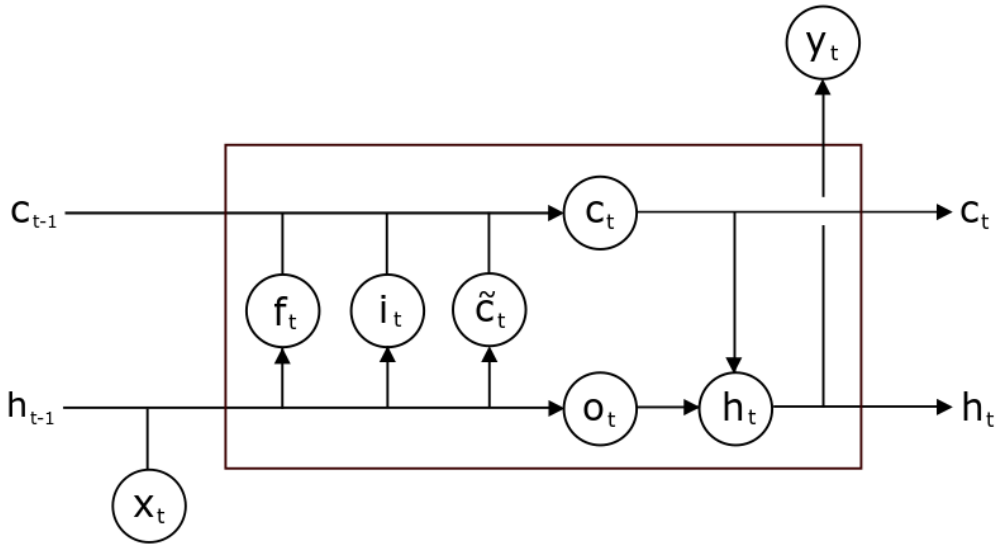

Figure 13: Close-up of a LSTM cell.

can flow through many time steps with minimal degradation. To update the cell state the LSTM utilizes three gates, namely the forget gate $\mathbf{f}_t$, the input gate $\mathbf{i}_t$ and the candidate gate $\tilde{\mathbf{c}}_t$. The cell state has the formula

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t, \tag{32}$$

where the operator $\circ$ denotes the Hadamard product, which is an operation that takes two equally sized matrices and returns a matrix of the same dimension as the inputs where the element at row i and column j in the output matrix is the product of the elements at the same position in the input matrices. As an example:

$$\begin{bmatrix} 4 & 3 \\ 3 & 6 \end{bmatrix} \circ \begin{bmatrix} 1 & 3 \\ 8 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 9 \\ 24 & 30 \end{bmatrix}.$$

The forget gate has the formula

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{x}_t + \mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{b}_f), \tag{33}$$

where σ denotes a Sigmoid activation function. The forget gate outputs a matrix with the same size as $\mathbf{c}_t$ where each element is a number between 0 and 1. 1 means that the corresponding element in $\mathbf{c}_t$ is to remain unchanged, while 0 means that the element is to be completely forgotten. The input gate has the formula

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{x}_t + \mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{b}_i), \tag{34}$$

while the candidate gate has the formula

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{U}_c\mathbf{x}_t + \mathbf{W}_c\mathbf{h}_{t-1} + \mathbf{b}_c). \tag{35}$$

The input gate decides which values in the cell state to update and the candidate gate generates candidates for new additions.

The hidden state h_t of each time step is computed by the formula

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t), \tag{36}$$

where $\mathbf{o}_t$ is (confusingly) called the output gate and has the formula

$$\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{x}_t + \mathbf{W}_o\mathbf{h}_{t-1} + \mathbf{b}_o). \tag{37}$$

The hidden state $\mathbf{h}_t$ is thus a combination of the newest input $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$ and the current cell state $\mathbf{c}_t$.

Finally, just as for an Elman or a Jordan RNN the final ouput of the RNN at each time step is simply

$$\mathbf{y}_t = \sigma_y(\mathbf{V}\mathbf{h}_t) \tag{38}$$

[8, p.404-407] [11] [12].

3.6 Support vector regression

Let $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ be a set of N training samples where each sample consists of a feature vector $\mathbf{x}_i \in \mathbb{R}^n$ and a response $y_i \in \mathbb{R}$. Support vector regression is a supervised learning method where the basic idea is to fit a linear function $f(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i + b$ on the data set, where the regression weights $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are found by solving the following convex optimization problem:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{subject to} \quad & |y_i - \mathbf{w}^T\mathbf{x}_i - b| \leq \epsilon, \quad i = 1, \ldots, m, \end{aligned} \tag{39}$$

where ϵ is a parameter that specifies the maximum deviation that $f(\mathbf{x}_i)$ can have from y_i. [13, p.6] [14, p.1-2] Obviously, sometimes a solution to the above problem will not exist. That is, sometimes it is not possible to fit a function where the deviation between $f(\mathbf{x}_i)$ and y_i is less than ϵ for all i. To account for situations like that we add a non-negative slack variable ξ_i to each inequality and add a penalty term to the objective function that punishes potential solutions for each non-zero slack variable as shown below:

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & |y_i - \mathbf{w}^T\mathbf{x}_i - b| \leq \epsilon + \xi_i, \\ & \xi_i \geq 0, \quad i = 1,\ldots,N. \end{aligned} \tag{40}$$

The parameter C decides how harshly non-zero slack variables are punished. It can be shown [14, p.2-3] [15] that the above problem has a different but equivalent formulation called the dual problem:

$$\begin{aligned} \underset{\alpha,\alpha^*}{\text{minimize}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i^T\mathbf{x}_j(\alpha_j - \alpha_j^*) + \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) - \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0, \\ & 0 \leq \alpha_i, \alpha_i^* \leq C, \quad 1 \leq i \leq N, \end{aligned} \tag{41}$$

where the solution of the dual problem is related to the solution of the original problem through the formula $\mathbf{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i$.

An obvious limitation of linear support vector regression is that it can only represent linear functions. However, support vector regression can be used to fit non-linear functions by using the "kernel-trick", where we replace the dot product $\mathbf{x}_i^T\mathbf{x}_j$ with a non-linear inner product $K(\mathbf{x}_i,\mathbf{x}_j)$. This inner product, which is called a kernel, is a distance function that calculates how "different" $\mathbf{x}_i$ is to $\mathbf{x}_j$. The formula of the fitted function for non-linear support vector regression is thus

$$f(\mathbf{x}) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)K(\mathbf{x}_i,\mathbf{x}) + b \tag{42}$$

[13, p.6]. A commonly used kernel function for non-linear support vector regression is the radial basis function

$$K(\mathbf{x}_i,\mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2), \tag{43}$$

where γ is a free parameter.

A number of different algorithms can be used to fit support vector regression models. One of the main issues in solving the non-linear version of the dual problem in equation 41 is that the kernel matrix $\mathbf{K}$ is dense and may be too large to be stored. As a workaround to this issue the open-source LIBSVM library uses a decomposition method called sequential minimal optimization (SMO) that only modifies two α's per iterations so that only two elements of $\mathbf{K}$ are needed each iteration. Then in each iteration the algorithm only needs to solve a simple two-variable convex optimization problem. See [13, p.11-15, 26-29] for details.

3.7 Kalman filtering

3.7.1 Kalman filter algorithm

The Kalman filter algorithm

Let us consider a state space model where the observation $\mathbf{Y}_t \in \mathbb{R}^{m\times 1}$ (e.g. the power load at a certain hour or the 24 power load measurements for a certain day) at timestep t is a linear combination of an internal system state $\mathbf{S}_t \in \mathbb{R}^{n\times 1}$ and a measurement error $\mathbf{b}_t \in \mathbb{R}^{n\times 1}$, that is, it is given by the equation

$$\mathbf{Y}_t = \mathbf{H}\mathbf{S}_t + \mathbf{b}_t, \tag{44}$$

where $\mathbf{H} \in \mathbb{R}^{m\times k}$ is a fixed matrix. Furthermore let us assume that the internal system state is represented in such a way that it follows an AR(1) process, meaning that it is a linear combination of the previous system state and a system error $\mathbf{a}_t \in \mathbb{R}^{n\times 1}$. In other words that it follows the equation

$$\mathbf{S}_{t+1} = \mathbf{A}\mathbf{S}_t + \mathbf{G}\mathbf{a}_{t+1}, \tag{45}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{G} \in \mathbb{R}^{k\times n}$ are fixed matrices. Finally, assume that both $\mathbf{a}_t$ and $\mathbf{b}_t$ are independent and identically distributed Gaussian white noise processes with zero mean and covariance matrices $\mathbf{\Sigma}$ and $\mathbf{\Omega}$ respectively. It follows from equation 45 that the k-step ahead forecast of the state vector is given by the recursion

$$\hat{\mathbf{S}}_{t+k|t} := \mathrm{E}(\mathbf{S}_{t+k}|\mathbf{Y}_1, ..., \mathbf{Y}_t) = \mathbf{A}\hat{\mathbf{S}}_{t+k-1|t}, \tag{46}$$

while the k-step ahead variance of the state vector can be calculated from

$$\mathbf{V}_{t+k|t} := \mathrm{Var}(\mathbf{S}_{t+k}|\mathbf{Y}_1, ..., \mathbf{Y}_t) = \mathbf{A}\mathbf{V}_{t+k-1|t}\mathbf{A}^T + \mathbf{G}\mathbf{\Sigma}\mathbf{G}^T. \tag{47}$$

Furthermore it then follows from equation 44 that the k-step ahead forecast of the observation vector is given by the equation

$$\hat{\mathbf{Y}}_{t+k|t} := \mathrm{E}(\mathbf{Y}_{t+k}|\mathbf{Y}_1, ..., \mathbf{Y}_t) = \mathbf{H}\hat{\mathbf{S}}_{t+k|t} = \mathbf{H}\mathbf{A}^k\hat{\mathbf{S}}_{t|t} \tag{48}$$

and that the uncertainty of the k-step ahead forecast is

$$\mathrm{Var}(\hat{\mathbf{Y}}_{t+k}|\mathbf{Y}_1, ..., \mathbf{Y}_t) = \mathbf{H}\mathbf{V}_{t+k|t}\mathbf{H}^T + \mathbf{\Omega}. \tag{49}$$

Kalman filtering is an algorithm for iteratively estimating the system state and updating it as more data becomes available. Each iteration consists of two steps. The first step is the pre-

diction step where the next state $\hat{\mathbf{S}}_{t+1|t}$ is forecasted and the variance $\mathbf{V}_{t+1|t}$ of the forecast is calculated based on the current system state estimate $\hat{\mathbf{S}}_{t|t}$ and the current state variance $\mathbf{V}_{t|t}$ using equations 46 and 47. The second step is the update step, also called the filtering step, where the system state estimate $\hat{\mathbf{S}}_{t+1|t+1}$ and the system variance $\mathbf{V}_{t+1|t+1}$ are updated based on the newest observation $\mathbf{Y}_{t+1}$. The details are shown in algorithm 5. [16, p.478-482]

Algorithm 5 Kalman filter

Choose initial state estimate $\hat{\mathbf{S}}_{0|0}$ and variance $\mathbf{V}_{0|0}$.
for $t = 0$ to $n-1$ **do**
  Forecast the next state: $\hat{\mathbf{S}}_{t+1|t} = \mathbf{A}\hat{\mathbf{S}}_{t|t}$
  Calculate the variance of the state forecast: $\mathbf{V}_{t+1|t} = \mathbf{A}\mathbf{V}_{t|t}\mathbf{A}^T + \mathbf{G}\Sigma\mathbf{G}^T$
  Calculate the Kalman gain: $\mathbf{K}_{t+1} = \mathbf{V}_{t+1|t}\mathbf{H}^T(\mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \mathbf{\Omega})^{-1}$
  Update the state estimate using new observation: $\hat{\mathbf{S}}_{t+1|t+1} = \hat{\mathbf{S}}_{t+1|t} + \mathbf{K}_{t+1}(\mathbf{Y}_{t+1} - \mathbf{H}\hat{\mathbf{S}}_{t+1|t})$
  Update the state variance: $\mathbf{V}_{t+1|t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})\mathbf{V}_{t+1|t}$
end for

3.7.2 Derivation of the filtering equations

This section is dedicated to deriving the update equations for the state estimate $\hat{\mathbf{S}}_{t+1|t+1}$ and the state variance $V_{t+1|t+1}$ that are shown in algorithm 5.

The one step ahead observation forecast error is given by the equation

$$\begin{aligned}\mathbf{e}_{t+1} &= \mathbf{Y}_{t+1} - \hat{\mathbf{Y}}_{t+1|t} \\ &= \mathbf{Y}_{t+1} - \mathbf{H}\hat{\mathbf{S}}_{t+1|t} \\ &= \mathbf{Y}_{t+1} - \mathbf{H}\mathbf{A}\hat{\mathbf{S}}_{t|t} \\ &= \mathbf{H}\mathbf{S}_{t+1} + \mathbf{b}_{t+1} - \mathbf{H}\mathbf{A}\hat{\mathbf{S}}_{t|t} \\ &= \mathbf{H}(\mathbf{S}_{t+1} - \mathbf{A}\hat{\mathbf{S}}_{t|t}) + \mathbf{b}_{t+1}\end{aligned} \tag{50}$$

and thus the distribution of the one step ahead observation forecast error is

$$(\mathbf{e}_{t+1}|\mathbf{S}_{t+1}, \mathbf{Y}_t) \sim N(\mathbf{H}(\mathbf{S}_{t+1} - \mathbf{A}\hat{\mathbf{S}}_{t|t}), \mathbf{\Omega}). \tag{51}$$

Assume that the distribution of the next system state $\mathbf{S}_{t+1}$ if the observation forecast error $\mathbf{e}_{t+1}$ is known is

$$(\mathbf{S}_{t+1}|\mathbf{e}_{t+1}, \mathbf{Y}_t) \sim N(\hat{\mathbf{S}}_{\mathbf{t+1|t+1}}, \mathbf{V}_{t+1|t+1}). \tag{52}$$

We want to find formulas for $\hat{\mathbf{S}}_{t+1|t+1}$ and $\mathbf{V}_{t+1|t+1}$. To do this we first note that two variables $\mathbf{X}_1$ and $\mathbf{X}_2$ follow a joint normal distribution

$$\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{bmatrix} \sim N\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right) \tag{53}$$

if and only if

$$\begin{cases}\mathbf{X}_1 \sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11})\\ \mathbf{X}_2|\mathbf{X}_1 \sim N(\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{X}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}).\end{cases} \tag{54}$$

See [17] for a full proof. Now, let $\mathbf{X}_1$ correspond to $\mathbf{S}_{t+1}$ and $\mathbf{X}_2$ correspond to $\mathbf{e}_{t+1}$. Clearly $\boldsymbol{\mu}_1 = \mathbf{A}\hat{\mathbf{S}}_t$ and $\boldsymbol{\Sigma}_{11} = \mathbf{V}_{t+1|t}$. Furthermore, by equating the two expressions for the mean of $(\mathbf{e}_{t+1}|\mathbf{S}_{t+1})$ from equations 51 and 54 we get that $\boldsymbol{\mu}_2 = \mathbf{0}$, $\boldsymbol{\Sigma}_{12} = \mathbf{V}_{t+1|t}\mathbf{H}^T$ and $\boldsymbol{\Sigma}_{21} = \mathbf{H}\mathbf{V}_{t+1|t}$. Similarly, by equivalating the two expressions for the variance of $\mathbf{e}_{t+1}$ we get that $\boldsymbol{\Sigma}_{22} = \mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \boldsymbol{\Omega}$. Thus the joint distribution of $\mathbf{S}_{t+1}$ and $\mathbf{e}_{t+1}$ is

$$\left(\begin{bmatrix}\mathbf{S}_{t+1}\\ \mathbf{e}_{t+1}\end{bmatrix}\middle|\mathbf{Z}_t\right) \sim N\left(\begin{bmatrix}\mathbf{A}\hat{\mathbf{S}}_t\\ \mathbf{0}\end{bmatrix}, \begin{bmatrix}\mathbf{V}_{t+1|t} & \mathbf{V}_{t+1|t}\mathbf{H}^T\\ \mathbf{H}\mathbf{V}_{t+1|t} & \mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \boldsymbol{\Omega}\end{bmatrix}\right). \tag{55}$$

Obviously, the opposite of equation 54, obtained by interchanging $\mathbf{X}_1$ with $\mathbf{X}_2$, also holds. And thus by comparing equation 52 with equation 54 we obtain that

$$\begin{aligned}\hat{\mathbf{S}}_{t+1|t+1} &= \mu_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{X}_2 - \boldsymbol{\mu}_2)\\ &= \hat{\mathbf{S}}_{t+1|t} + \mathbf{V}_{t+1|t}\mathbf{H}^T(\mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \boldsymbol{\Omega})^{-1}\mathbf{e}_{t+1}\\ &= \hat{\mathbf{S}}_{t+1|t} + \mathbf{K}_{t+1}(\mathbf{Y}_{t+1} - \mathbf{H}\hat{\mathbf{S}}_{t+1|t})\end{aligned} \tag{56}$$

and

$$\begin{aligned}\mathbf{V}_{t+1|t+1} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}\\ &= \mathbf{V}_{t+1|t} - \mathbf{V}_{t+1|t}\mathbf{H}^T(\mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \boldsymbol{\Omega})^{-1}\mathbf{H}\mathbf{V}_{t+1|t}\\ &= (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H}_t)\mathbf{V}_{t+1|t},\end{aligned} \tag{57}$$

where for convenience we have introduced the shorthand

$$\mathbf{K}_{t+1} := \mathbf{V}_{t+1|t}\mathbf{H}^T(\mathbf{H}\mathbf{V}_{t+1|t}\mathbf{H}^T + \boldsymbol{\Omega})^{-1} \tag{58}$$

[16, p.480-482].

4 Models

This section describes the different models that were tested on the data set provided by NTE. All the models were implemented using the programming language Python, except NTE's current Kalman filter model. The neural networks were implemented using Keras, a neural network library for Python, while the linear regression and support vector regression models were implemented using sklearn, a general machine learning library for Python. Sklearn itself uses the ϵ-SVR implementation from the LIBSVM library [13]. The direct Kalman filter model was written from scratch, although it relies on the NumPy library for matrix multiplications. The pandas library was used for reading data and for general data manipulation and analysis, while the matplotlib and seaborn libraries were used to generate the plots in this thesis. All of these libraries are open-source. The exception to all of this is NTE's present Kalman filter model, which was not programmed by me. It is proprietary and not owned by NTE or me, thus I only had access to the predictions made by the model for 2011-2017, not to the actual source code.

4.1 Multilayer perceptrons

4.1.1 MIMO model

Let us denote the current hour by t and the most recent power load measurement by P_{t-K}, where K is the number of hours since it was made. The lag K is included because it may take a few hours before accurate power load measurements become available. Next, assume that we want to predict the power loads for each hour of the next day: $P_{t+M}, P_{t+M+1}, ..., P_{t+M+23}$, where $1 \leq M \leq 24$. As an example: if the current time is 10 am on a Tuesday and the most recent power load measurement was made five hours ago at 5 am and we want to predict tomorrow's (Wednesday's) power loads from 1 am to 12 pm then $K = 5$ and $M = 15$.

The first model presented is a multilayer perceptron that uses a multiple-input multiple-output (MIMO) strategy. It outputs the predictions $\hat{P}_{t+M}, \hat{P}_{t+M+1}, ..., \hat{P}_{t+M+23}$ as a single vector. The inputs to the neural network are:

- The 24 most recent power load measurements available: $P_{t-K}, ..., P_{t-K-23}$.
- Temperature measurements for the current and previous $K+23$ hours: $T_t, ..., T_{t-K-23}$.
- Temperature predictions for the next $M+23$ hours: $\hat{T}_{t+1}, ..., \hat{T}_{t+M+23}$.
- 7 binary variables, one for each day of the week, where the current weekday is equal to 1 and the others are equal to 0: $D_{1,t}, ..., D_{7,t}$.

The power, temperature and temperature prediction time series were standardized before they were fed into the network, meaning that they were transformed by subtracting the mean

and dividing by the standard deviation of the time series. The temperature time series used was a weighted sum of the temperature measurements from Namsos, Steinkjer and Stjørdal using formula 1. Note that the data set provided by NTE only includes day-ahead temperature predictions from 1 am to 12 pm based on information available at 10 am on the current day. This means that the temperature predictions for 11 am to 12 pm on the current day that the model uses were made the previous day at 10 am and are thus "outdated". This may seem like a problem that would lead to worse test forecasting accuracy than if newer temperature predictions were available, however as discussed in section 7.2, the errors of day-ahead temperature forecasts are so low that there is no significant difference in forecasting accuracy between using temperature forecasts or actual temperatures and thus this is not a problem in practice.

A manual search was used to find good values for the hyperparameters in the network. In general the model is not very picky with what the values of the hyperparameters are, many different configurations give very similar results.

The Adam algorithm was used to fit the weights in the neural network. Kingma and Ba [10] suggests that good default values for the parameters in the Adam algorithm are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$, however a learning rate of $\alpha = 0.001$ was found to give a non-smooth learning curve, so it was lowered to 0.0001. Stochastic gradient descent gave equally good results as the Adam algorithm, but training times were more than twice as long. There was no noticeable difference between using the mean absolute error or the more commonly used mean squared error as the error function, so the latter was chosen. Bengio [18, p.9] suggests that 32 is a good default value for the batch size, however this was found to give a non-smooth learning curve even when the learning rate was reduced significantly further. Increasing the batch size all the way to 256 gave a smoother learning curve while giving equally good results.

During the hyperparameter optimization the model was trained until 100 epochs had gone by without any improved performance on the valdation set. In the testing phase this was changed to initially training the model for 2000 epochs.

It was found that adding more than one hidden layer to the model did not improve model performance and that a single hidden layer with 40 neurons is sufficient. All weights in the neural network were initialized before training by drawing random values from a standard normal distribution. Sigmoid, ReLU and hyperbolic tangent activation functions were all tested in the hidden layer and all three gave equally good predictions. ReLU was chosen since it is slightly faster than the other two. A linear activation function was used in the output layer because the target values in the standardized power load time series can theoretically take any real value and thus the range of the activation function in the output layer must be the set of real values.

Regularization techniques such as L1/L2 regularization or dropout that punish model complexity were not used as the model is not sufficiently complex to warrant their use.

4.1.2 Direct model

The basic idea of the direct model is that instead of training one multilayer perceptron with 24 outputs that predicts the power load for each hour the next day we can instead train 24 separate models, each of which is trained to predict the power load for a different hour.

The inputs to the model are the exact same as for the MIMO model, except that we do not include the temperature predictions for future time steps. E.g. the model that predicts P_{t+M+n} uses the temperature predictions $\hat{T}_{t+1}, ..., \hat{T}_{t+M+n}$ as input, but does not use $\hat{T}_{t+M+n+1}, ..., \hat{T}_{t+M+23}$. All other aspects of the models like the neural network architecture, the optimization technique and the hyperparameters used are the exact same as in the MIMO model.

4.1.3 SMSO model

The single model single output (SMSO) MLP is a variation of the MIMO model where instead of returning 24 predictions, one for each hour of the next day, the hour that we want to predict is sent as an argument to the model and only the prediction for that hour is returned. The inputs to the MISO model are the exact same as those to the MIMO model except that 24 extra binary variables are added that specify which hour of the day the target value is for. Otherwise the model is identical to the MIMO model, with the exception that the number of training epochs was reduced from 2000 to 200 because now that there is one training example for each hourly measurement in the power time series, instead of one for each day, the size of the training set is 24 times larger and thus since the number of iterations per epoch is increased 24-fold fewer epochs are needed for the model to be fitted. Some experimentation was done to see if changing the other hyperparameters would lead to better performance, but no noticeable improvements were observed, so the other hyperparameters were kept the same.

4.2 Recurrent neural networks

4.2.1 LSTM encoder-decoder

Like with the MLP models let us denote the current hour by t and the most recent power load measurement available by P_{t-K} and assume that we want to predict the power loads for each hour of the next day: $P_{t+M}, P_{t+M+1}, ..., P_{t+M+23}$, where $1 \leq M \leq 23$.

An illustration of the LSTM encoder-decoder model is shown in figure 14. The model consists of two recurrent neural networks, one called the encoder and one called the decoder. The job of the encoder is to take the last 24 power load measurements and the temperature measurements from the same period and encode them in a vector that is used as the first hidden state in the decoder. The decoder first predicts $\hat{P}_{t-K+1}$ using the inputs from the encoder as well as a vector of exogenous information $\mathbf{x}_{t-K+1}$, then in the second step the decoder predicts

$\hat{P}_{t-K+2}$ using $\mathbf{x}_{t-K+2}$ as well as the hidden state from the previous step. This process is repeated recursively up to $\hat{P}_{t+M+23}$. The exogenous vector $\mathbf{x}$ has 32 dimensions. 24 of the variables are binary variables that specify the hour of the day, while seven others are binary variables that specify the day of the week. The final variable specifies either the weighted average temperature for that hour or the predicted weighted average temperature depending on whether the time step is in the past or in the future.

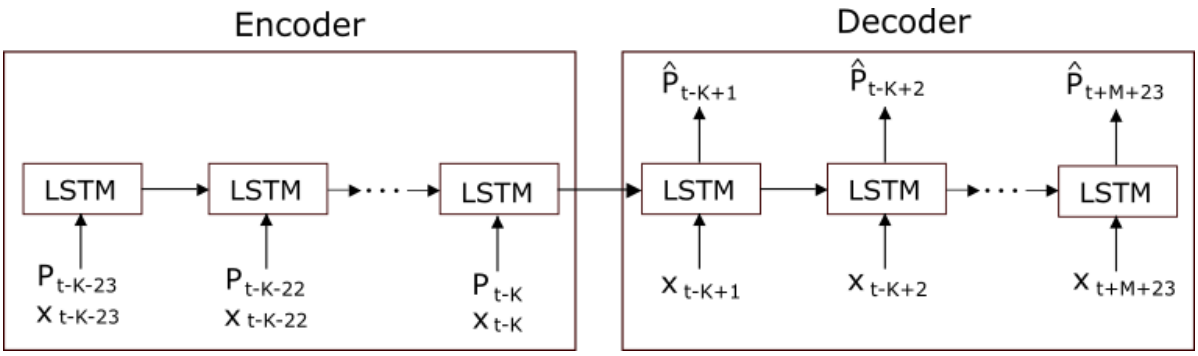

Figure 14: Illustration of an unrolled encoder-decoder LSTM model.

A possible variation of this model would be to feed P_{t+a-1} as an input to the decoder when predicting $\hat{P}_{t+a}$ during training and to replace P_{t+a-1} with $\hat{P}_{t+a-1}$ during testing. The problem with this approach is that since the power load usually does not change enormously from one hour to the next the decoder might be overly prone to learning the naive mapping $\hat{P}_{t+a} = P_{t+a-1}$, which could lead to poor forecasting accuracy. In fact, preventing the model from learning the naive mapping was the main motivation behind using an encoder-decoder model instead of a vanilla LSTM model.

The model was trained for 1000 epochs using the Adam algorithm with a learning rate of $\alpha = 0.0005$ and a batch size of 256. The two recurrent neural networks are trained simultaneously as one model, meaning that during training the errors are backpropagated all the way from the last LSTM unit of the decoder to the first LSTM unit of the encoder. Like with the MLP models 40 hidden units were used, since there doesn't appear to be any benefit in adding more units than that.

4.2.2 Elman encoder-decoder

The Elman encoder-decoder RNN model is the exact same as the LSTM encoder-decoder model except that the LSTM units were replaced with Elman hidden layers. It was tested primarily to see how large of a performance increase we get from using LSTM layers over Elman layers.

4.3 Direct linear regression model

The direct linear regression model is similar to the direct MLP model described in section 4.1.2. 24 seperate linear regression models are trained to predict each of the day-ahead power loads. The formulas for the models are

$$\begin{aligned} P_{t+M+n} = \alpha &+ \sum_{i=0}^{23} \beta_i P_{t-K-i} + \sum_{i=1}^{7} D_{i,t} \\ &+ \sum_{i=0}^{K+24} \gamma_i T_{t-K-23+i} + \sum_{i=1}^{M+n} \gamma_{i+K+24} \hat{T}_{t+i} \\ &+ \sum_{i=0}^{K+24} \delta_i T^2_{t-K-23+i} + \sum_{i=1}^{M+n} \delta_{i+K+24} \hat{T}^2_{t+i} \\ &+ \sum_{i=0}^{K+24} \epsilon_i T^3_{t-K-23+i} + \sum_{i=1}^{M+n} \epsilon_{i+K+24} \hat{T}^3_{t+i}, \quad 0 \leq n \leq 23. \end{aligned} \tag{59}$$

See section 4.1.1 for explanations of what the different covariates stand for. Note that all the inputs are standardized before they are fed into the model. The inputs of each singleton model are the same as for the direct MLP model except that in addition to feeding the temperature measurements (and forecasts) into the model it is also fed the square and the cube of the temperatures measurements (and forecasts). The reason for this is that we want the model to be able to represent non-linear relationships between power and temperature.

Using 24 separate singleton models, one for each hour of the day, is a crucial part of the model. Many linear regression STLF models are based on a single model that uses the hour of day as a covariate. The problem with that approach is that because the relationship between the other covariates and the power load is highly dependent on the hour of day we then need to add a ton of interaction terms of various orders to the model to get good forecasting accuracy. With this in mind it might be tempting to take the idea further and have 168 separate models, one for each hour of the week, however this degrades the forecasting accuracy because the training set becomes too small. Besides, it was found empirically that simply treating the day of the week as an additive term in the model in equation 59 works quite well and that adding interaction terms between the day of the week and other variables like e.g. the temperature is unnecessary.

Note that the model in equation 59 can be also be written in matrix notation as

$$P_{t+M+n} = \mathbf{H}_t \boldsymbol{\beta}, \quad 0 \leq n \leq 23, \tag{60}$$

where

$$\begin{aligned}\mathbf{H}_t = &[1, P_{t-K}, ..., P_{t-K-23}, D_{1,t}, ..., D_{7,t},\\ &T_{t-K-23}, ..., T_t, \hat{T}_{t+1}, ..., \hat{T}_{t+M+n},\\ &T^2_{t-K-23}, ..., T^2_t, \hat{T}^2_{t+1}, ..., \hat{T}^2_{t+M+n},\\ &T^3_{t-K-23}, ..., T^3_t, \hat{T}^3_{t+1}, ..., \hat{T}^3_{t+M+n}]\end{aligned} \tag{61}$$

and

$$\begin{aligned}\boldsymbol{\beta} = &[\alpha, \beta_0, ..., \beta_{23}, \gamma_1, ..., \gamma_{K+M+24+n},\\ &\delta_1, ..., \delta_{K+M+24+n}, \epsilon_1, ..., \epsilon_{K+M+24+n}, 1, ..., 1]^T.\end{aligned} \tag{62}$$

This second notation will be useful in section 4.5.2.

4.4 Direct support vector regression model

The direct support vector regression (SVR) model is similar to the direct MLP model described in section 4.1.2 and the direct linear regression model described above. One support vector regression model is trained to predict each of the day-ahead power loads $P_{t+M}, ..., P_{t+M+23}$. The inputs to each of the models are the exact same as for the direct MLP model.

A two-step grid search was used to determine the optimal values of the hyperparameters C, γ and ϵ. In the first step the parameter values for C that were tried were $\{0.01, 0.1, 1, 10, 100\}$, while for γ and ϵ they were $\{0.0001, 0.001, 0.01, 0.1, 1\}$, which results in a total of $5^3 = 125$ combinations. After it was found that the best out of these combinations was 10^i with $i = 0$ for C and 10^i with $i = -2$ for γ and ϵ a further $3^3 - 1 = 26$ combinations were investigated by trying all possible combinations of $\{3 \cdot 10^{i-1}, 1 \cdot 10^i, 3 \cdot 10^i\}$ for all three parameters. The optimal values of the hyperparameters were thus determined to be $C = 3, \gamma = 0.01$ and $\epsilon = 0.01$.

4.5 Kalman filters

4.5.1 NTE's current model

NTE's current model is based on Kalman filtering. It is proprietary and was made by Powel AS in 1997. Because the model is proprietary I did not have access to the source code of the model, nor do I know the exact details of the model, I only had access to the model's predictions from 2011 to 2017. However, as mentioned in section 2.3, I do know that the model uses the exact same weather data as I have access to to make its predictions.

4.5.2 Direct Kalman filter model

For comparison purposes I also created my own Kalman filter model. The model is essentially the same as the direct linear regression model except that the regression coefficients are treated as the internal state of a state space model and are determined by recursively using Kalman filtering instead of choosing the coefficients that minimize the mean squared error on a training set. The model uses a direct strategy and consists of 24 separate Kalman filters, each of which predicts the power load on a certain hour of the day. The state space model used is

$$\begin{cases} P_{t+M+n} = \mathbf{H}_{t+M+n}\boldsymbol{\beta}^* + b_{t+M+n} \\ \boldsymbol{\beta}_{t+M+n} = \boldsymbol{\beta}_{t+M+n-24} + \mathbf{a}_{t+M+n}, \quad 0 \leq n \leq 23, \end{cases} \tag{63}$$

where P_t denotes the power load at hour t, $a_t \sim N(0, \mathbf{\Sigma})$ and $b_t \sim N(0, \omega)$ are random variables, while $\boldsymbol{H}_t$ is the same as in equation 61 and $\boldsymbol{\beta}_t$ has the same form as in equation 62. $\boldsymbol{\beta}^*$ denotes that we use the most recent state vector estimate for the correct hour that is available, which may vary depending on i and M. By comparing equation 45 with 63 you may note that the state-transition matrix $\mathbf{A}$ is absent, or more precisely it has been turned into an identity matrix. This is because the estimate of the coefficients $\boldsymbol{\beta}$ is only updated during the filtering step of the Kalman filter algorithm, not during the forecasting step.

4.6 Ensemble average model

The predictions of the ensemble average model is simply the average of the predictions from the three best performing models, namely the MIMO MLP model, the SMSO MLP model and the direct Kalman filter model.

4.7 Naive model

The naive model simply predicts that the power consumption during a certain hour on a certain day is equal to the power consumption during the same hour on the previous day of the same type where we know the actual power consumption, where the two types of days we distinguish between are workdays and weekends.

As an example: If we want to forecast the power load for each hour on a Tuesday based on the data we have available at 5 pm on the day before (Monday), then the predictions from 1 am up to 5 am on Tuesday will simply be the power loads from 1 am to 5 am on Monday, while the predictions from 6 am to 12 pm will be the power loads during the same hours on the preceding Friday.

The purpose of the naive model is to provide a baseline of what a useful model is. In a nutshell; a model that does not outperform the naive model is completely useless.

4.8 Dealing with public holidays

As can be seen in figure 4 in section 2.2, Norwegian public holidays tend to have load patterns that are similar to those of weekends even if they fall on a regular workday. Thus a simple trick to deal with public holidays is to treat them as Sundays. E.g. if December 25th falls on a Tuesday then instead of setting the Tuesday binary variable to one we set the Sunday variable to one. This trick was used in all the models.

4.9 Dealing with daylight savings time

In Norway the clock is turned forward one hour each year on the last Sunday in March from 2 am to 3 am and back one hour on the last Sunday in October at 2 am to 1 am. Thus there is one power load observation less than usual on the former day and one more than usual on the latter day. The way this is dealt with varies a bit from model to model. The MIMO model for instance always outputs 24 predictions, but on the last Sunday in March the 2 am prediction is discarded, while on the last Sunday in October the 2 am prediction is duplicated. Similarly, for the direct models, the 2 am model is used twice on the last Sunday in March and zero times on the last Sunday in October. The daily power load pattern is primarily dependent on consumer behavior, not on natural cycles (apart from temperature variations), thus for the direct models 9 am in the Summer is predicted using the same singleton model that predicts 9 am in the Winter, even though one is in Summer time and the other is not.

5 Test method

To test the models we will look at the special case where the current time is 10 am and we want to predict the power load for each hour of the next day. E.g. if today is Tuesday then based on the information available at 10 am today we want to predict the power load on Wednesday for each hour from 1 am to midnight. This is the special case that is of most relevance to NTE and it is the case that allows us to compare the performance of the different presented models directly to those of NTE's current model. Note however that accurate power loads for the last couple of hours are typically not available and a typical scenario is that the last power load measurement available is from five hour previously at 5 am. Thus even though relative to the current time we are predicting the power loads 15-38 hours ahead using 15-38 hours ahead temperature forecasts, if we look at it relative to the last available power load measurement we are actually predicting 20-43 hours ahead. In other words, using the terminology of section 4.1.1 we are looking at the special case where $K = 5$ and $M = 15$.

Although this is the special case that we will look at, the models from section 4 were designed and written in such a way that they can easily be modified to work in other similar situations.

5.1 Test strategies

When evaluating the performance of a supervised learning model it is important that the forecasting accuracy of the model is evaluated on different data than it was trained on. This is primarily to ensure that the model has not overfitted the training data. Overfitting means that the model has not only captured the underlying patterns in the training data, but also the noise, resulting in a model that generalizes poorly to new data.

The simplest way to do this is to split the data set into two non-overlapping subsets, a training set and a test set, and only evaluate the performance of the model based on its performance on the test set. A more sophisticated approach is to use k-fold cross validation, where the original data set is randomly split into k non-overlapping subsets of roughly equal size, called folds. One of the folds is designated as the holdout fold and the model is trained on the $k-1$ other folds before the forecasting accuracy is evaluated on the holdout fold. This process is repeated k times using each fold as the holdout fold exactly once. The k results can then be averaged or otherwise combined to produce a single estimate of the forecasting accuracy. An advantage of this method is that all observations in the data set are used for validation exactly once. A disadvantage is that this validation process is more time consuming than using a simple training/test set split.

Two problems arise when cross-validation is used on problems where the data set is a time series. The first is that, because subsequent measurements in the time series are usually correlated with each other, randomly splitting the data set can lead to artificially good forecasting

accuracy. As an example consider the power load data set from section 2. In the real world, when we forecast the power load on a day, e.g. on the 12th of May, we will know the power load on the 10th of May, but we will not know the power load on the 13th of May and on most of 11th of May. If however we use k-fold cross validation with random splits then it is quite likely that the power load on those two days will be a part of the training set. Because the power load on subsequent days is highly correlated the model will then have been trained on two days that are very similar to the 12th of May that it wouldn't have been trained on in the field, which may result in artificially good forecasting accuracy. This problem can be avoided by using folds of continous data, e.g. by splitting the data set into one fold for each year, as shown in figure 15.

The second problem is that we are subjecting the evaluation to look-ahead-bias by training the model on future data. An alternative approach that avoids this second problem is to only train the model on past years. That is, we first train the model on 2011 and test it on 2012, then we train it on 2011 and 2012 and test it on 2013 and so on. This is called an expanding window approach and is illustrated in figure 16. Another name for this approach is forward chaining. One thing to note about an expanding window approach is that the amount of data that the model is trained on increases each year, unlike with k-fold cross-validation where it stays the same. The downside of this is that the forecasting accuracy of different years cannot be compared directly because the quality of the forecast can usually be expected to increase as we train on more data, as long as we don't train on data that is so far back in the past that changes in consumer patterns has made it irrelevant. The upside is that we get an estimate of the impact that the training set size has on the forecasting accuracy.

Hyndman and Athanasopoulos [19, ch.2.5] recommend using an expanding window approach for time series prediction problems where the window expands for each new observation. That is, if k previous observations are needed to make a reliable forecast, then the model is first trained on the first k observations in the time series and tested on observation $k+1$. Then observation $k+1$ is added to the training set and the model is retrained and tested on observation $k+2$ and so on. They refer to this as time series cross-validation or alternatively as "evaluation on a rolling forecasting origin". The advantage of this approach is that the next prediction is always made using all available past information. The disadvantage is that for some models retraining or updating the model for each new observation that comes along can be very time intensive.

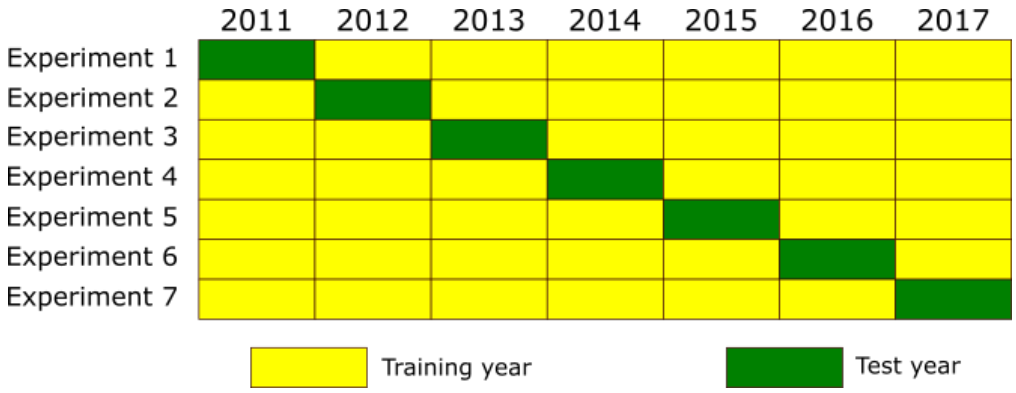

Figure 15: Illustration of k-fold cross validation with continous folds

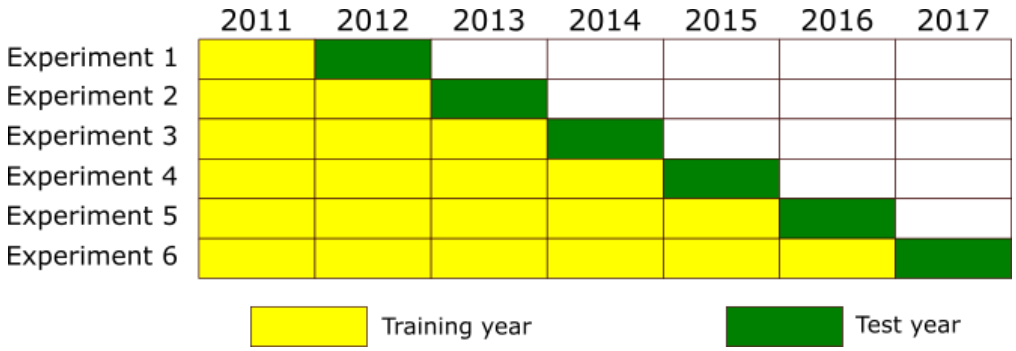

Figure 16: Illustration of an expanding window approach.

5.2 Tests performed

5.2.1 Expanding window test run

The primary approach that was chosen for testing the performance of the different models was the one suggested by Hyndman and Athanasopoulos, where the models are continously updated as new data becomes available, since this is the approach that most closely mimics a real-world situation, however the precise way this was done varies a bit from model to model. The Kalman filter models were tested using an expanding window with a window size of a single day where each day the models predict the power loads for the next day and then when the actual power loads for that day become available the forecasting errors are calculated and the internal states of the Kalman filters are updated accordingly. The support vector regression and linear regression models were trained using an expanding window approach with a window size of seven days, meaning that they were completely retrained each seven days using all past available data and then tested on the next seven days and so on.

The MIMO and direct MLP models were first trained for 2000 epochs on the data from 2011 and tested on the data from 2012. Each day the previous day's data is added to the training set

and the models were updated by training for a further five epochs on the new training set, a technique which will henceforth be referred to as rehearsing. Then the models predicted the power loads for the next day. By the end of 2012 each of the two models had been trained for roughly $2000 + 5 \cdot 363 = 3815$ epochs. At the end of 2012 the models were then completely retrained from scratch for 2000 epochs using all data from both 2011 and 2012 and the cycle was repeated for 2013. The encoder-decoder and SMSO neural network models were tested using the exact approach except that the encoder-decoder models were first trained for 1000 epochs instead of 2000 each year, while the SMSO MLP model was first trained for 200 epochs and only rehearsed for one epoch each day.

For a discussion of why these update strategies were chosen see section 7.3.

5.2.2 Cross validation test run

In the cross validation test run the test approach from figure 15 was used. That is, first the models were trained on all years except 2011 and tested on 2011, then they were trained on all years except 2012 and tested on 2012 and so on. Note however that the models were continuously updated as new information became available in the same manner as in section 5.2.1. E.g after the SVR and linear regression models had been trained on data from all years except 2011 and tested on the first week of 2011, the first week of 2011 was then added to the training set and the models were retrained before being tested on the second week of 2011 and so on.

Note that, as discussed in subsection 5.1, when using this test approach the test results are subject to look-ahead bias since we are training the model on future data. Thus the cross validation test results should be taken with a grain of salt and be viewed as less reliable than the expanding window test results.

5.2.3 Varying the number of training years

To investigate how the accuracy of a forecast varies depending on the number of training years the MIMO MLP, SMSO MLP and direct Kalman filter models were additionally tested by training on varying numbers of previous years. E.g for 2014 each of these models were tested three times, once by training on 2013 and testing on 2014, once by training on both 2012 and 2013 and testing on 2014 and once by training on 2011, 2012 and 2013 and testing on 2014. Once again note that the models were continuously updated throughout the year as new information became available in the same manner as described in section 5.2.1.

5.2.4 Testing temperature input sensitivity

Finally I tested how the accuracy of the MIMO MLP model changes when different temperature inputs are used. The testing approach used was once again an expanding window approach

with continuous updating using the rehearsal technique. The reason why only the MIMO MLP model was tested is that testing the forecasting accuracy for all years using a large number of different temperature inputs would be quite time consuming for most of the models.

6 Results

6.1 Expanding window test results

6.1.1 Errors broken down by year

Figure 17 shows the mean absolute errors of all the models from section 4 for each year except the naive model, whose errors are shown in table 2. The precise error numbers are shown in table 11 in the appendix, while table 12 in the appendix shows the mean absolute percentage errors.

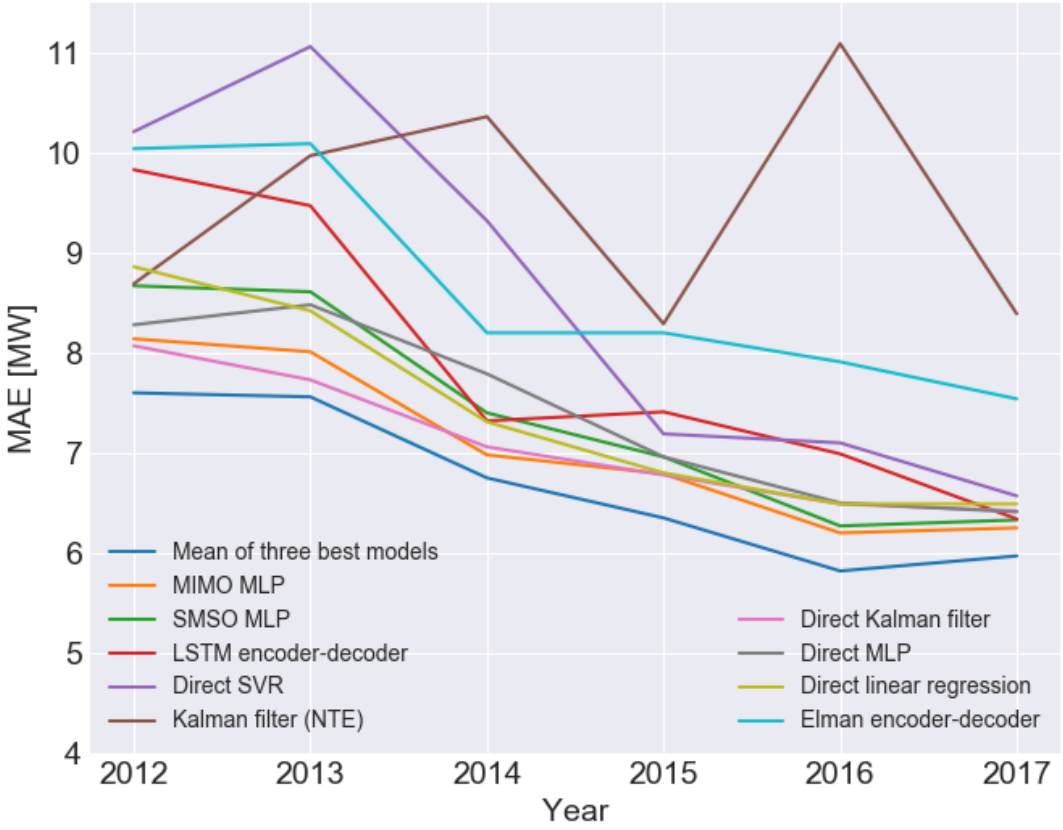

Figure 17: Mean absolute errors (MAE) in MW for each of the different models from section 4, broken down by year.

Model	2012	2013	2014	2015	2016	2017
Naive model	17.46	18.54	16.78	16.37	17.66	18.26

Table 2: MAEs in MW for the naive model.

6.1.2 Errors broken down by the hour of day

Figure 18 shows how the errors of seven of the models tested varied throughout the day in 2017, the year where the models were trained on the largest amount previous of data. The shape of the daily error curves for the different models look very similar for the other years, except for NTE's Kalman filter model, whose curve varies a lot from year to year. The four other models were left out to avoid cluttering the graph too much. The direct MLP and linear regression models have a daily error curve similar to most of the other models, while the curve of the Elman encoder-decoder is a bit different as it becomes very high at the end of the day. The error of the naive model increases drastically from 5 am to 6 am, but before and after this dramatic increase the error is relatively constant throughout the day.

Note that the error at 01:00 is for a 20 hours ahead forecast, relative to the last available power load measurement, whereas the error at 24:00 is for a 43 hours ahead forecast. Furthermore note that the power load measurement at e.g. 02:00 is actually the average power load between 01:00 and 02:00 and thus the error at 02:00 in the figure is actually the average error between 01:00 and 02:00.

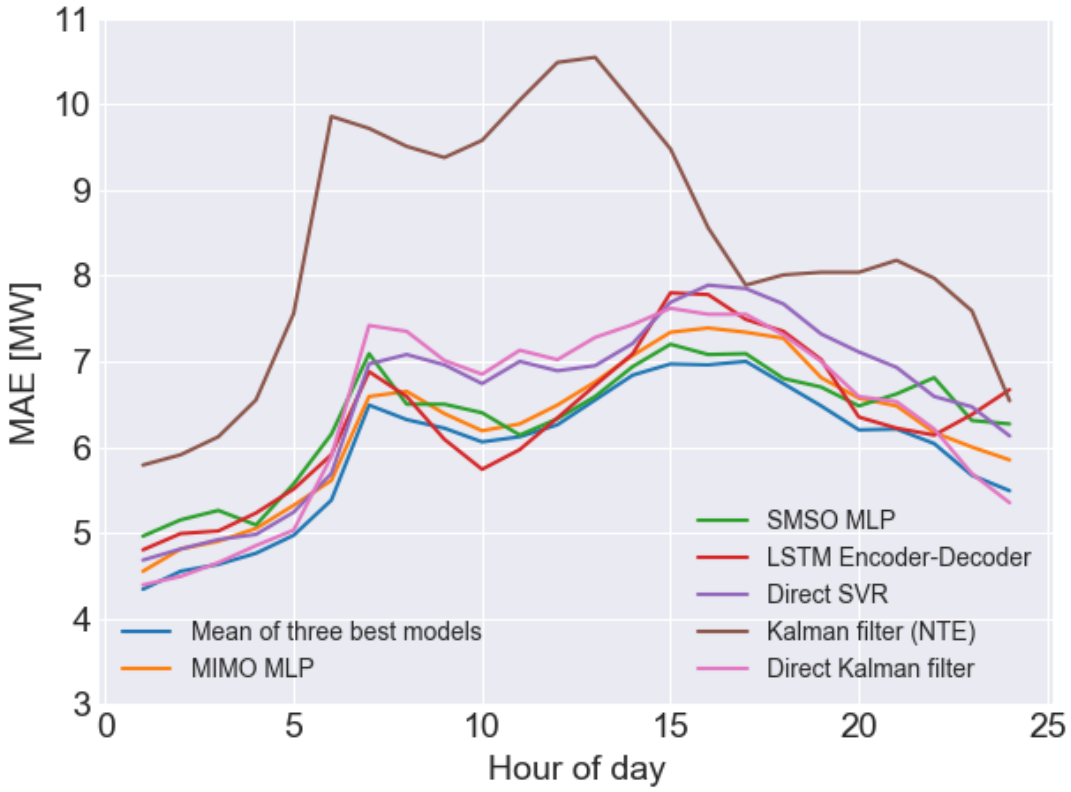

Figure 18: Mean absolute error at different hours of the day for 2017 for seven of the tested models.

6.1.3 Errors broken down by weekday

Table 3 compares the errors of the same seven models as in subsection 6.1.2 for different days of the week. The table also shows the average error on public holidays. Again, only the forecasts for 2017 are considered, although the corresponding errors for the other years look similar (although apart from 2016 they are a bit higher in general).

Model	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Holidays
Mean of three best models	6.15	5.79	6.60	6.10	5.59	5.49	5.39	8.13
MIMO MLP	6.28	6.24	7.06	6.45	5.77	6.10	6.10	8.62
SMSO MLP	6.44	6.33	7.20	6.45	6.23	5.56	6.13	8.69
LSTM (encoder-decoder)	7.06	6.93	7.28	7.13	5.42	6.54	6.14	10.44
Direct SVR	7.06	6.71	7.17	6.80	6.83	5.99	5.47	8.74
Kalman filter (NTE)	8.52	9.18	8.54	9.37	7.70	7.44	8.01	9.50
Direct Kalman filter	6.38	6.00	7.14	6.70	6.33	6.21	6.21	9.33
Naive model	23.93	23.15	14.62	14.17	15.32	19.69	19.65	24.66

Table 3: Mean absolute errors for different weekdays in 2017. The measurement unit is MW.

6.1.4 Training and update times

Table 4 shows the training times of each model. The times are for an Intel Core i5-6600K Skylake 3.5GHz processor, which is a mid-range desktop processor from 2015.

Model	2012	2013	2014	2015	2016	2017
Direct linear regression	00:28	00:30	00:31	00:32	00:34	00:34
Direct SVR	00:32	00:32	00:28	00:31	00:38	00:44
MIMO MLP	00:20	00:32	00:42	01:02	01:15	01:19
Direct Kalman filter	00:32	00:36	00:56	01:11	01:26	01:27
SMSO MLP	00:49	01:14	01:51	02:33	03:21	03:41
Elman encoder-decoder	01:31	02:05	03:11	03:51	04:59	05:17
Ensemble average	01:53	02:22	03:29	04:46	06:02	06:47
LSTM encoder-decoder	04:08	05:56	09:16	11:10	14:32	15:56
Direct MLP	11:16	12:29	14:19	20:28	23:15	28:45

Table 4: Training times in minutes and seconds for each of the different models.

Table 5 shows the daily update times for the neural network models and the direct Kalman filter model. Note that the table only shows the time needed to update the models and does not

include the time needed to load a trained model from memory, which might add a couple of extra seconds of overhead, depending on the implementation, when a model is used in a real world situation.

Model	2012	2013	2014	2015	2016	2017
Direct Kalman filter	0.08	0.10	0.13	0.15	0.18	0.16
MIMO MLP	0.04	0.08	0.09	0.16	0.18	0.18
SMSO MLP	0.17	0.27	0.43	0.61	0.80	0.73
Ensemble average	0.29	0.45	0.65	0.92	1.16	1.07
Elman encoder-decoder	0.35	0.56	0.81	1.05	1.31	1.47
Direct MLP	1.30	1.93	2.50	2.97	3.32	3.81
LSTM encoder-decoder	1.19	1.91	2.64	3.36	4.24	4.64

Table 5: Daily update times in seconds.

6.1.5 Ensemble average residuals

Figure 19 shows the residuals of the ensemble average model for 2017. The predictions of the ensemble average model is the mean of the predictions from the three best performing individual models, namely the MIMO MLP, SMSO MLP and direct Kalman filter models.

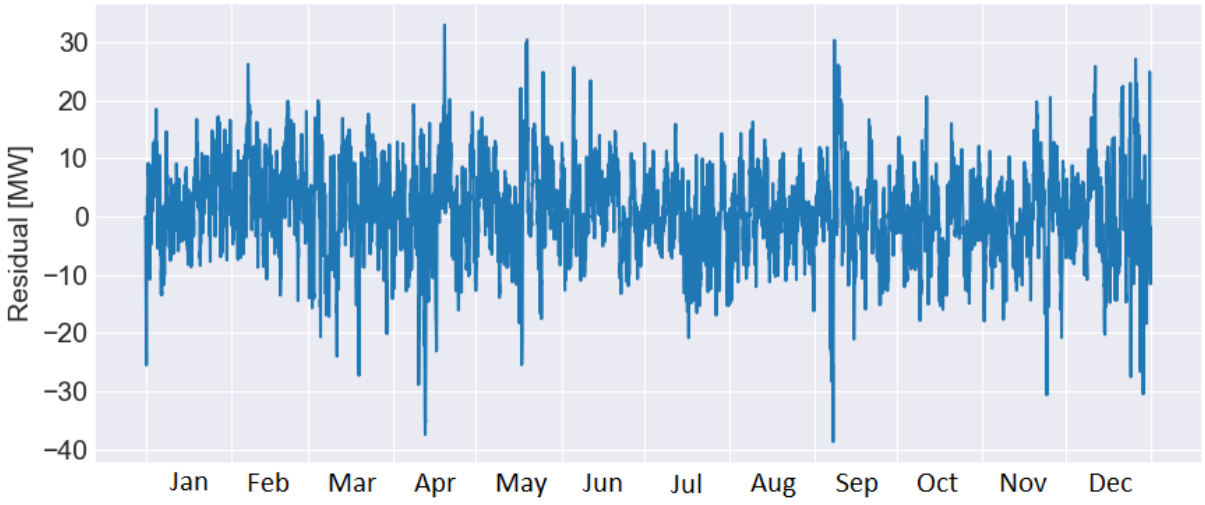

Figure 19: Hourly forecasting residuals of the ensemble average model for 2017.

Figure 20 shows the predictions of the ensemble average model plotted against the actual power loads for the same period as figure 2, while figure 21 shows the predictions of the model around Easter 2017, which is typically one of the hardest periods of the year to predict accurately.

Finally, figure 22 shows the predictions of the model for a challenging period of 2017 where the temperature changed rapidly several times over a short period. The temperatures for this period are plotted in figure 23.

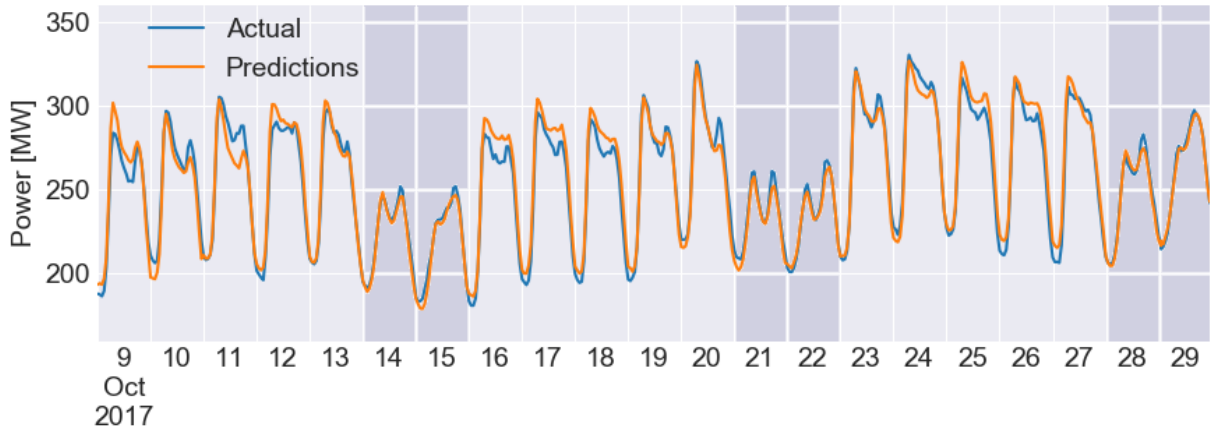

Figure 20: Predictions from the ensemble average model plotted against the actual power loads for the period from Monday October 9th to Sunday October 29th 2017. Weekends are shaded with a darker background than workdays. The MAE during this period was 5.40 MW, which is slightly better than the yearly average, which was 5.97 MW.

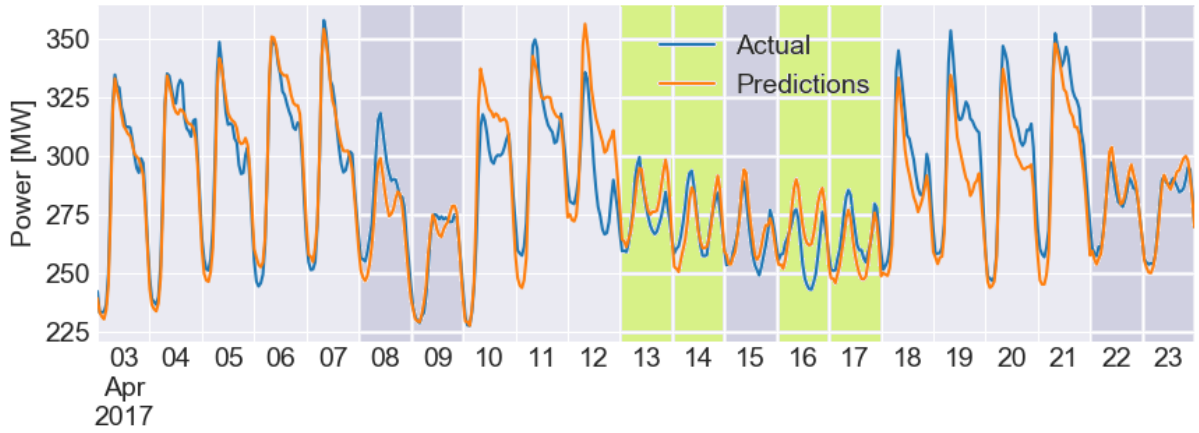

Figure 21: Predictions from the ensemble average model plotted against the actual power loads for the period from Monday April 3rd to Sunday April 23rd 2017. Weekends are shaded with a darker background than workdays and the Easter holidays are shaded in green. The MAE during this period was 8.04 MW. This was the model's worst performing three week period in 2017.

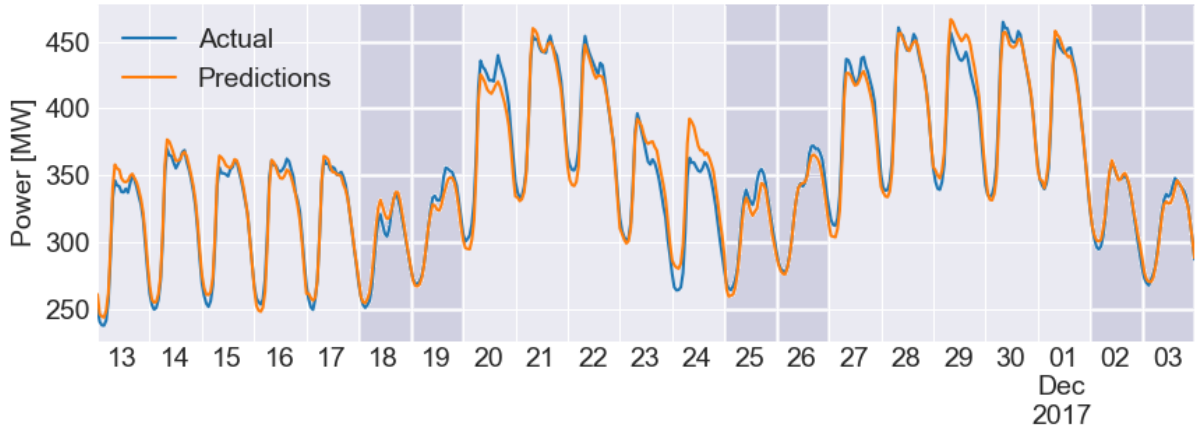

Figure 22: Power load predictions from the ensemble average model versus actual power loads for the period from Monday November 13th to Sunday December 3rd 2017. Weekends are shaded with a darker background than workdays. The MAE during this period was 6.26 MW, which was slightly worse than the yearly average.

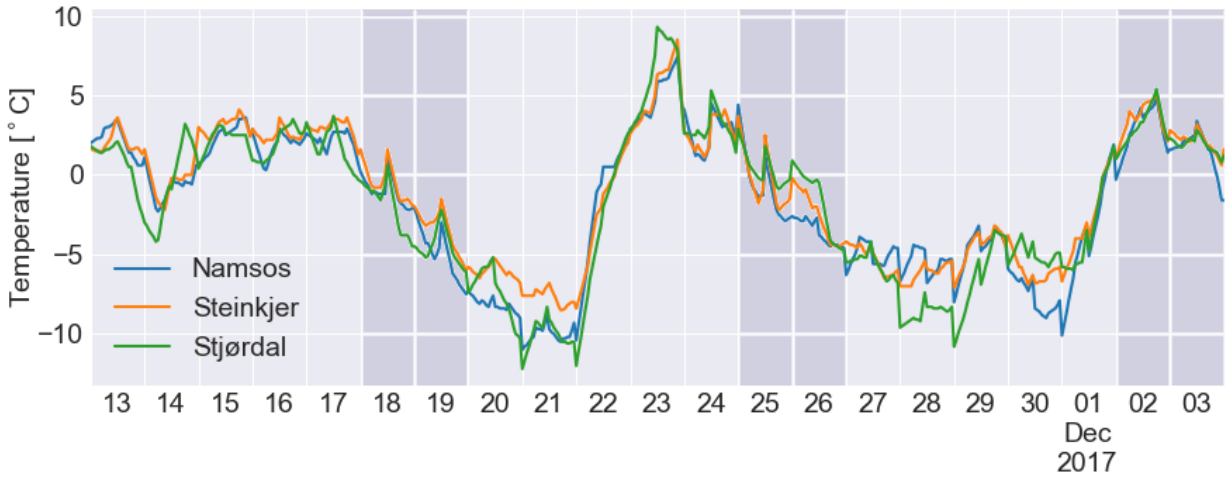

Figure 23: Temperature measurements from the same period as figure 22.

6.2 Cross validation test results

6.2.1 Errors broken down by year

Figure 24 shows the mean absolute errors of the different models for each year when using a cross validation test approach where the models are trained on an equal amount of data for each test year. The precise errors numbers are shown in table 13 in the appendix.

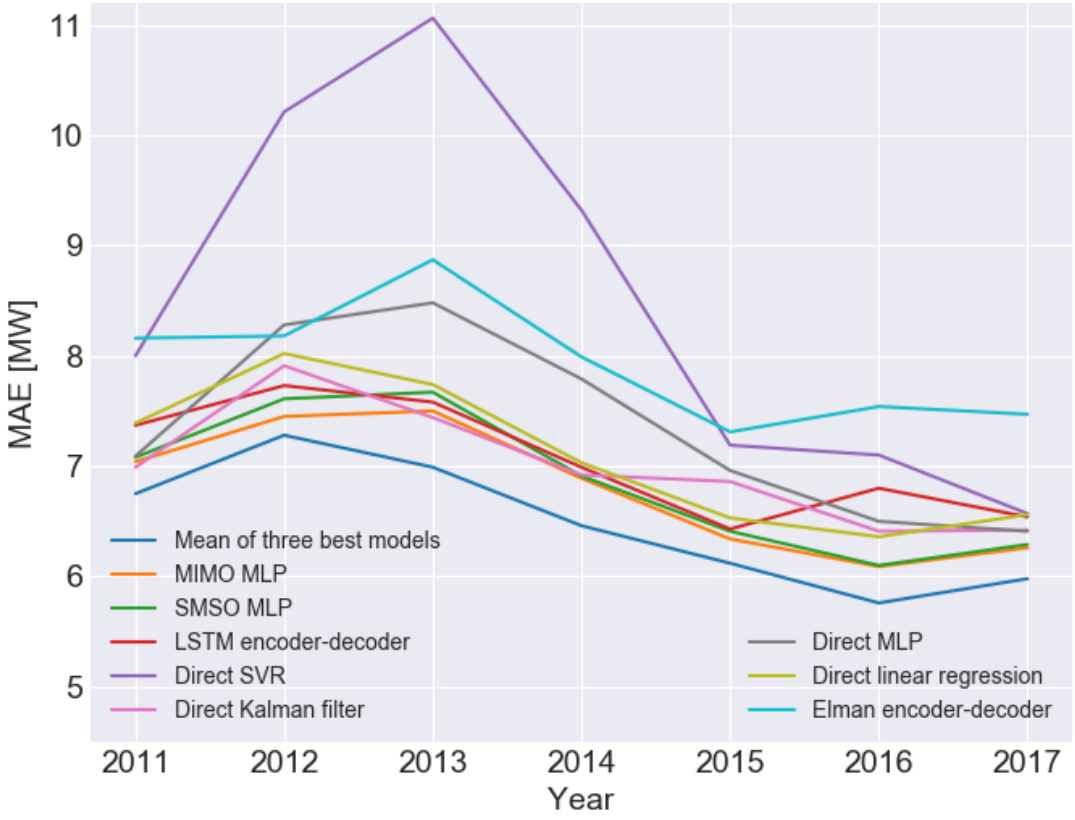

Figure 24: Mean absolute errors (MAE) in MW for the different models for each year when a cross validation test approach was used.

6.2.2 Errors broken down by the hour of day

Figure 18 shows how the errors of seven of the models change at different hours of the day when a cross validation test approach is used. Unlike in section 6.1.2 the predictions from all the years were used to produce the plot, not just those for 2017.

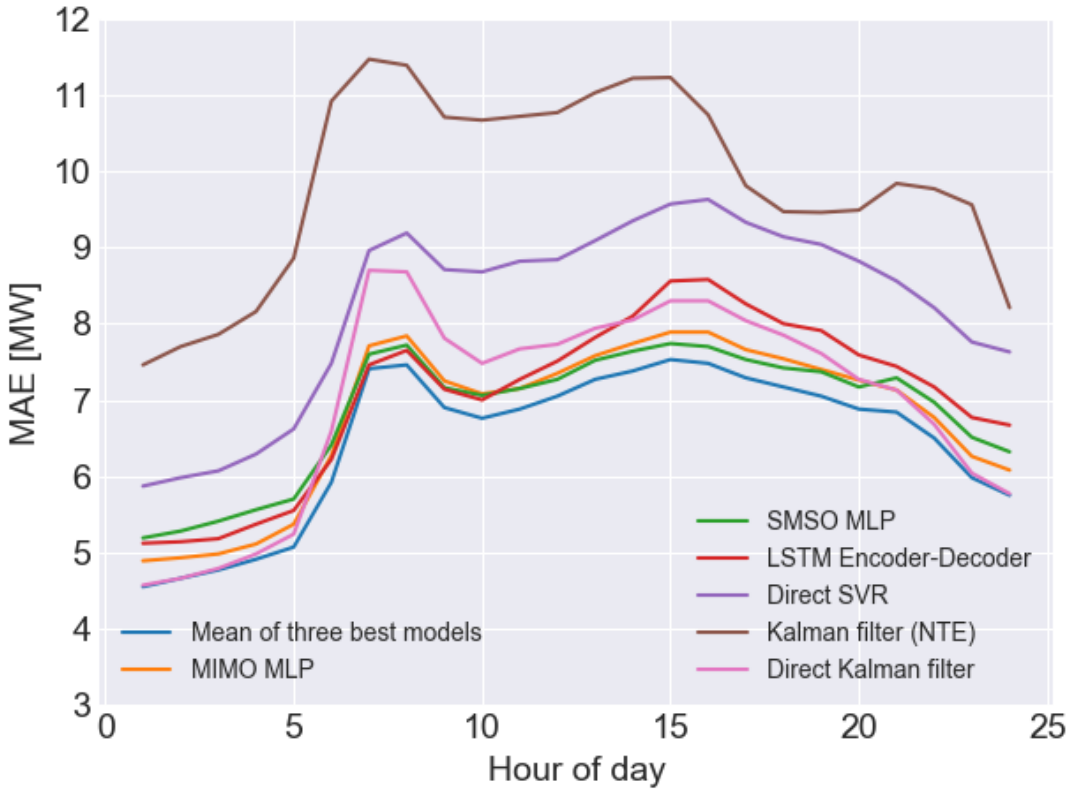

Figure 25: Mean absolute error at different hours of the day calculated from all predictions from 2011 to 2017.

6.2.3 Errors broken down by weekday

Table 6 shows the errors from the cross validation test run broken down by weekday. Once again note that unlike in section 6.1.3 the predictions from all the years were used to produce the table.

Model	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Holidays
Mean of three best models	7.20	6.64	6.71	6.59	6.20	6.23	5.77	8.70
MIMO MLP	7.41	6.93	7.18	6.97	6.44	6.50	6.16	8.95
SMSO MLP	7.44	7.09	7.13	6.97	6.64	6.51	6.26	8.86
LSTM encoder-decoder	7.98	7.08	6.90	7.13	6.92	7.15	6.27	9.40
Direct SVR	9.00	8.30	8.03	8.18	7.95	8.12	8.08	10.00
Kalman filter (NTE)	10.03	9.97	10.20	10.19	9.78	9.29	9.53	11.04
Direct Kalman filter	7.83	6.93	6.90	6.95	6.79	7.20	6.37	9.47

Table 6: Mean average errors for different weekdays, as well as for public holidays. The measurement unit is MW.

6.3 Varying the number of training years

Tables 7 shows the forecasting accuracy of the ensemble average model when the model is trained on different years using an expanding windows strategy where the model is constantly updated throughout the year with different numbers of initial training years. E.g. the cell in column 2016 and row three years back shows the mean absolute error in MW when the model was initially trained on 2013, 2014 and 2015 before it was tested on 2016. Too see the individual errors of each of three models that make up the ensemble average model see tables 14, 15 and 16 in the appendix.

Training years	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**
One year back	7.61	7.86	7.13	6.79	6.27	6.34
Two years back	X	7.62	6.93	6.52	6.21	6.13
Three years back	X	X	6.69	6.42	5.98	6.05
Four years back	X	X	X	6.31	5.89	5.90
Five years back	X	X	X	X	5.89	6.08
Six years back	X	X	X	X	X	6.01

Table 7: MAE of ensemble average model in MW when tested on different years using a continuously updating forecasting strategy with different numbers of initial training years.

6.4 Temperature input sensitivity

Figures 8 and 9 show how the average forecasting error of the MIMO MLP model changes when different temperature inputs are used. In all cases the MIMO MLP model was tested using the same approach as the one used in section 6.1.

Temperature time series used	**MAE [MW]**
Weighted average	7.06
All three time series	7.27
Steinkjer only	7.41
Namsos only	7.63
Stjørdal only	7.89

Table 8: Mean absolute error of MIMO MLP model when different temperature time series are used as input. The error is the average of the errors from each year from 2012 to 2017.

Temperature inputs	**MAE [MW]**
$T_{t-K-23}, ..., T_t, T_{t+1}, ..., T_{t+M+23}$	7.03
$T_{t-K-23}, ..., T_t, \hat{T}_{t+1}, ..., \hat{T}_{t+M+23}$	7.06
$\hat{T}_{t+1}, ..., \hat{T}_{t+M+23}$	7.85
$T_{t-K-23}, ..., T_t$	11.22
No temperature information	12.29

Table 9: Mean absolute error of MIMO MLP model when using different temperature inputs. All the temperature measurements are from the weighted average temperature time series, not from a single location. The error is the average of the errors from each year from 2012 to 2017.

7 Discussion

7.1 Discussion of test results

By comparing figure 17 in section 6.1.1, which shows the forecasting accuracy of the different models for each year when using an expanding window test approach, with table 2, which shows the forecasting accuracy of the naive model, we observe that all the models tested outperform the naive model by a large margin. Overall the multiple-input multiple output multilayer perceptron (MIMO MLP) model and the direct Kalman filter model have the lowest errors of any of the individual models tested, but the two other MLP models and the direct linear regression model follow very closely behind.

When looking at tables 4 and 5 in section 6.1.4 we observe that the training and update times of the direct MLP model are roughly 20 times longer than for the MIMO MLP model since the former essentially consists of 24 neural networks of roughly the same size as the latter. Since the direct MLP model is both a lot slower than the MIMO MLP model and gives slightly higher errors the MIMO MLP model is clearly the preferable one.

Similarly, although the direct linear regression and direct Kalman filter models give very similar results, since they are essentially the exact same model except that they use different methods to estimate the model parameters, the Kalman filter model can be updated very easily and quickly each day, whereas the linear regression model has to be retrained from scratch to be updated. Thus the direct Kalman filter model is clearly the preferable one.

NTE's Kalman filter model performed decently on 2012 compared to the other models, since it is the only model that has been trained on data from before 2011, but overall the accuracy of the model is bad compared to most of the others. The model has a tendency to behave erratically at times and the accuracy of the model fluctuates wildly from year to year. In particular it was observed that the model struggles to adapt during periods where the temperature changes rapidly. The strong performance of the direct Kalman filter model suggests that this is due to a poorly implemented model rather than a flaw of Kalman filtering itself. Part of the reason for the comparatively bad performance might be that NTE's model is from 1997, when power consumption patterns were quite different from now. However considering that the covariates that are relevant for predicting the power consumption are unlikely to change much over time, a well-implemented Kalman filter model should have been able to adapt to the changes in consumer patterns.

Compared to the other models the direct support vector regression (SVR) model performs well on the last three years, but poorly on the first three. Upon closer inspection of the errors it was observed that the poor performance on the first three years is actually a result of the model catastrophically failing during one week in 2012, two in 2013 and a further two in 2014. Other-

wise the model performance is similar to the best models. These five weeks were ones that experienced quick shifts in temperature and/or temperatures that were lower or higher than what the model had seen before. Whether this problem is due to poorly chosen hyperparameters or something else is unclear.

As expected the LSTM encoder-decoder model appears to significantly outperform the Elman encoder-decoder, however the performance of these two models is overall disappointing compared to the others, especially when you consider the fact that these two models are both harder to implement and slower to train than the others. During the writing this thesis quite a lot of time was spent trying out different neural network models that use a recursive strategy where a single neural network is trained to predict the power load one hour ahead and then this model is used recursively to produce multiple hours ahead forecasts up to 43 hours ahead, like the encoder-decoder models do. The motivation behind this was that a recursive model, especially a recurrent neural network, might achieve better forecasting accuracy than the MIMO, SMSO and direct MLP models by exploiting the sequential nature of the data, which the others only utilize implicitly. For instance, a MLP model that produces one-hour ahead forecasts was implemented and used to recursively produce forecasts up to 43 hours ahead. Although that model occasionally gave day-ahead MAEs as low as 6.5 on 2017, it tended to give very different results each time it was trained, even when using the exact same hyperparameters, making the model very unstable. The problem appears to be that, as mentioned in section 3.2.1, models that utilize a recursive strategy tend to suffer from accumulating forecasting errors. As an example, lets say that we train the recursive MLP model twice and calculate the three hour ahead forecast on the same day each time and that the forecasting error is a tiny bit higher the second time than the first. What will then happen is that when that forecast is fed back into the neural network and used to recursively produce load forecasts further and further into the future this tiny extra error will blow up into a larger error. Vanilla Elman and LSTM RNNs had the same issue. Only by using a combination of an encoder-decoder architecture that encourages the model to produce good multi-step ahead forecasts, not just good one-step ahead forecasts, and LSTM cells, that allow valuable information to propagate over many time steps without degradation, was I able to get results that were consistently close to those of the best models.

The best results were obtained by the ensemble average model which is simply the mean of the predictions from the MIMO MLP model, the direct Kalman filter model and the SMSO MLP model. This ensemble average model took less than seven minutes to train on six years of data and the time needed to update the model each day is only around a second when ignoring overhead, which makes the model very convenient to use in a real world situation.

Figure 19 in section 6.1.5 shows the residuals of the ensemble average model for 2017. Although the mean absolute errors of the model are relatively constant throughout the year, because power consumption is roughly twice as high in the winter as in the summer, the percent-

age errors are actually considerably lower in the winter than in the summer. The reason that absolute errors were used consistently throughout this thesis to measure forecasting accuracy instead of percentage errors is that we did not want to punish absolute errors in the summer harder than absolute errors in the winter.

Section 6.1.5 also contains some close-up comparisons of the predicted and actual power loads. From these plots we observe that qualitatively the predictions look very good most of the time. We observe that the model did a good job predicting the power consumption during the Easter weekend of 2017. As mentioned in section 4.8 the way public holidays are dealt with by the model is to treat them as if they were Sundays and qualitatively this has been observed to work well most of the time. In 2017 the model actually had higher errors on the three days preceding the Easter weekend than on the Easter weekend itself. The reason is that the power consumption on these three days tends to be slightly lower than on regular workdays because the days are located in the Easter vacation, when schools are closed. Similarly, the model tends to be a bit inaccurate on Christmas eve and new year's eve, especially if they fall on a workday, since the power consumption on these two days tends to be a bit lower than on regular workdays, but still higher than on weekends. A solution to this problem is to treat these days as special cases where the power load is first predicted normally and then downjusted a little bit afterwards. Unfortunately because there are very few instances of each of these days in the data set it is difficult to estimate precisely how much the power load should be downjusted, so it is probably inevitable that the error on these days will be a bit higher than normal.

Another minor issue is that the model sometimes underpredicts the power consumption on workdays that occur two days after a public holiday, as seen on the Wednesday after the Easter weekend in 2017 in figure 21. The reason is that the model uses the power load on Monday to predict the power load on Wednesday. Since Monday was a public holiday the power consumption on that day was a lot lower than usual for a Monday. The model sees this and predicts that this means that the power consumption on Wednesday will probably be a bit lower than usual as well, which turns out to be false. A quick fix for this problem would be to tell the model that this Wednesday is actually a Tuesday, so that it believes that the power loads from two days ago are from a Sunday.

From figure 18 in section 6.1.2 and figure 24 in section 6.2.2 we observe that the average error at different hours of the day is very similar for most of the models tested. The error is at its lowest for the first few hours of the day, which is logical since these are the predictions that are made the least far into the future. Then the error spikes at 7 am, most likely because, as can be seen in figure 3 in section 2.1, this is the time of the day where the power load quickly rises on regular weekdays as the day begins. A second spike occurs later around 4 pm when the workday ends.

As seen in tables 3 and 6 the errors do not vary much between different days of the week for any of the models tested except the naive model. The errors on public holidays on the other

hand are a bit higher than on other days, however, once again, considering that public holidays are exceptions to the general rules and represent a small portion of the data set slightly higher errors on these days should probably be considered inevitable and acceptable.

From figure 17 in section 6.1.1 we observe that in the expanding window test run the forecasting accuracy of all models except NTE's Kalman filter model tended to gradually improve each subsequent year. It is tempting to draw the conclusion that this is solely due to the models being trained on more and more data each year, however by looking at figure 24 in section 6.2.1 we observe that when a cross validation test approach is used, where the models are trained on an equal amount of data for all test years, the tendency for the forecasting accuracy to gradually improve from 2012 to 2017 is still present. Furthermore from table 7 in section 6.3 we observe that when the ensemble average model was only trained on the preceding year the error was still highest in 2012 and 2013 and lowest in 2015 and 2016. However the same table also show that the forecasting accuracy of the ensemble average model does indeed improve significantly when the model is trained on multiple previous years of data. For 2013, 2014 and 2015 the best results were obtained by training the model on the maximum number of previous years available, while for 2016 there was a tie between four and five previous years. For 2017 training only on the previous four years of data actually gave a slightly lower error than using the previous five or six yeas.

Considering that there hasn't been any large changes in consumer patterns over the period that the data set stretches the conclusion is thus that the tendency for the forecasting accuracy to gradually drop from 2012 to 2017 in figure 17 is partially due to the accuracy of the models improving as they are trained on more and more data and partially due to randomness. It appears that 2016 and 2017 happened to be relatively easy years to predict, whereas 2012 and 2013 happened to be relatively hard years predict.

It is hard to tell what the optimal number of training years is. Although some further testing could be done on the data set to try to give an answer to this question, in a real world situation the optimal number is in all likelihood going to depend on how much consumer behavior changed throughout the last couple of years, which is likely to fluctuate over time. A good solution in a real world situation might thus possibly be to train the same model several times using training sets that go back a varying number of years, test each of the trained models on recent data and use the version of the model that achieved the lowest errors.

As an aside, towards the end of writing this thesis I discovered that the MIMO MLP model that I came up with in this thesis is actually very similar to a model called the third generation ANNSTLF [20] (artificial neural network short-term load forecaster) that has been used by many electric utilities in the United States since the 90s. The main difference is that the ANNSTLF model does not use just a single MIMO MLP, but instead uses a weighted sum of the predictions from two MIMO MLPs, one that predicts the power loads for the next day and one that predicts

the difference between the power loads for today and tomorrow.

7.2 Temperature input sensitivity

From table 8 in section 6.4 we observe that feeding the MIMO MLP model a weighted average of the temperature measurements from three different locations in Nord-Trøndelag gives a lower forecasting error than feeding it only the temperatures from a single location, which is not surprising. A bit more surprising is the observation that feeding the model all three temperature time series also gives worse model performance, which holds true even if the size of the neural network is increased to account for the larger number of model inputs. The reason is likely that, because the temperature measurements from the three different locations are highly correlated, feeding the model all three time series does not add much extra useful information to the model. Thus doing so only makes it harder for the optimization algorithm to discern what the useful information in the input is.

Interestingly enough, using only the temperatures from Stjørdal, the location whose temperature forecasts are weighted the heaviest in the weighted average temperature time series, produces higher errors than using only the temperatures from Steinkjer or Namsos. Part of the reason might be found in figure 6 in section 2.4, which shows that the residuals of Stjørdal's temperature predictions are not quite normally distributed, but have a noticeable positive bias, which might negatively affect the forecasting accuracy. In this thesis I have not looked deeply into the weights used in the formula for the weighted average temperatures since the default weights seamed to work quite well, however table 8 suggests that weighting the predictions from Stjørdal the highest might not be a good idea and that either the weight for Stjørdal should be lowered or less biased temperature forecasts for Stjørdal should be obtained somehow.

Another interesting observation from section 6.4 is that, as seen in table 9, there is no significant change in forecasting accuracy between using actual or forecasted weighted average temperatures. This is not that surprising when you consider that, as seen in figure 6, the residuals of the weighted average temperature predictions are normally distributed with a mean of 0.24°C and standard deviation of 1.31°C, meaning that they are quite accurate. Still, this is actually a very useful result. It is easy to find historical temperature measurements for an area, but historical day-ahead temperature forecasts can be a lot harder to find, so it is nice to know that actual temperatures can be used as a substitute for temperature forecasts when training a model without effecting the accuracy of the forecast much.

7.3 Updating models

Table 10 shows the forecasting accuracy of the MIMO MLP model when using expanding window test approaches with daily, weekly, monthly or yearly window sizes. It also shows the ac-

curacy when using the approach that was used in section 6.1, where the model is trained once at the beginning of each year using all past data and then continuously updated throughout the year. From the table we observe that as the window size gets smaller the error tends to decrease. Using the approach from section 6.1 gives test results that are roughly as good as using an expanding window approach with weekly window sizes and slightly worse than an expanding window approach with daily window sizes, although the difference between these three approaches is so small that it is unlikely to be statistically significant. The reason that the testing strategy from section 6.1 was used for the neural network models instead of retraining the model from scratch each day is time usage. From table 4 we observe that the average training time of the direct MLP model is around 20 minutes. Thus from basic arithmetic, testing the direct MLP model by retraining it from scratch every single day from 2012 to 2017 would take around 700-800 hours, which isn't practical. Furthermore in a real world situation having to spend 20 minutes each day updating a model is not ideal. Similarly the linear regression and support vector regression models were only retrained once a week instead of every day to make training times a bit more manageable.

Strategy	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**Avg**
Expanding window (daily)	7.92	7.88	7.01	6.62	6.16	6.27	6.98
Method from section 5.2.1	8.14	8.01	6.98	6.79	6.20	6.25	7.06
Expanding window (weekly)	8.07	8.02	7.04	6.71	6.18	6.33	7.06
Expanding window (monthly)	8.42	8.10	7.17	6.95	6.33	6.30	7.21
Expanding window (yearly)	8.82	8.35	7.09	6.98	6.47	6.35	7.34

Table 10: Mean absolute error of the MIMO model when using an expanding window approach with different window sizes.

There is a trade-off present when using an expanding window approach where the model is regularly retrained from scratch. We want the model to be updated as often as possible to incorporate the newest available data to get the best possible forecast, but since retraining a model from scratch is time consuming we don't want to retrain the model more often than is necessary. The rehearsal technique used in this thesis to continuously update a neural network model as new data becomes available is a simple way to obtain forecasting accuracy's that are very close to those obtained when retraining the model daily without having to spend an inordinate amount of time updating the model each day. As can be seen from the tables in section 6.1.4, for 2017 updating the MIMO model by retraining it from scratch takes more than a minute, whereas updating the model using the rehearsal technique only takes around 0.2 seconds on a mid-range processor.

8 Conclusion

In this thesis a number of different models, some based on neural networks and some based on other methods, were applied to the problem of forecasting the power consumption in Nord-Trøndelag for each hour of the next day. The best performing neural network model that was tested was a multilayer perceptron (MLP) that uses a multiple-input multiple-output (MIMO) strategy, while the best performing model that wasn't a neural network was a Kalman filter model that consists of 24 time-varying linear regression models, one for each hour of the day, each of which uses Kalman filtering for parameter estimation. Even though these two models are quite different, in terms of performance they are very similar, both in terms of accuracy, training time and daily update times.

The lowest forecasting errors were obtained by using an ensemble average model whose predictions are the mean of the predictions from the two above-mentioned models and a variant of the MIMO MLP model that was referred to as the single model single output (SMSO) MLP. The ensemble average model takes a couple of minutes to train on a modern computer, but once it has been trained updating it each day as new data becomes available can be done in a matter of seconds. It was also found that the model can likely be continuously updated in this manner for a year without retraining without a significant drop in forecasting accuracy. In terms of accuracy the model is a big improvement compared to NTE's current model and unlike that model it is able to quickly adapt to changing temperatures. The only real flaw of the model is that it experiences slightly higher errors on and around public holidays and other special days than on regular days, but this is likely to be the case for any model.

The ensemble average model can likely be fine-tuned to improve model performance a little bit further, e.g. by fine-tuning how heavily it weights the temperature measurements from the three seperate measurement locations, however considering how strong the model performance already is, we have likely reached a point of diminishing returns where we are unlikely to see any big reductions in the forecasting errors. Overall the model should thus be considered to be ready for implementation.

References

[1] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *International Journal of Forecasting*, vol. 32, pp. 914–938, 2016.

[2] A. Srivastava, A. S. Pandey, and D. Singh, "Short-term load forecasting methods: A review," *International Conference on Emerging Trends in Electrical, Electronics and Sustainable Energy Systems*, 2016.

[3] T. K. Chheepa and T. Manglani, "A critical review on employed techniques for short term load forecasting," *International Research Journal of Engineering and Technology*, vol. 4, 2017.

[4] F. M. Bianchi, M. C. K. Enrico Maiorino, A. Rizzi, and R. Jenseni, "An overview and comparative analysis of recurrent neural networks for short term load forecasting," 2017.

[5] S. B. Taieb, G. Bontempi, A. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition," 2011.

[6] D. C. Montgomery, E. A. Peck, and G. Vining, *Introduction to Linear Regression Analysis*. John Wiley Sons, Inc., 4 ed., 2006.

[7] C. Bishop, *Pattern Recognition And Machine Learning*. Springer, 2006.

[8] A. Courville, I. Goodfellow, and Y. Bengio, *Deep Learning*. The MIT Press, 2016.

[9] J. McGonagle, G. Shaikouski, A. Hsu, J. Khim, and C. Williams, "Backpropagation." https://brilliant.org/wiki/backpropagation/. Accessed: 2018-06-01.

[10] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015.

[11] C. Olah, "Understanding LSTM networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed: 2018-06-18.

[12] K. Hornik, "Long short-term memory," *Neural Computation*, vol. 9, pp. 251–257, 1997.

[13] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," 2013.

[14] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2003.

[15] MathWorks, "Understanding support vector machine regression," 2018.

[16] W. Wei, *Time Series Analysis*. Pearson, 2 ed., 2006.

[17] R. Wang, "Marginal and conditional distributions of multivariate normal distribution." http://fourier.eng.hmc.edu/e161/lectures/gaussianprocess/node7.html. Accessed: 2018-05-21.

[18] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012.

[19] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2 ed., 2018.

[20] A. Khotanzad, R. Af'khami-Rohan, and D. Maratukulam, "ANNSTLF—A neural network short-term load forecaster - Generation three," *IEEE Transactions on Power Systems*, vol. 13, no. 8, 1998.

A Appendix

A.1 Detailed expanding window forecast errors

Model	2012	2013	2014	2015	2016	2017	Avg
Mean of three best models	7.60	7.56	6.75	6.35	5.82	5.97	6.67
MIMO MLP	8.14	8.01	6.98	6.79	6.20	6.25	7.06
Direct Kalman Filter	8.07	7.73	7.06	6.78	6.49	6.42	7.09
Direct Linear Regression	8.66	8.42	7.31	6.80	6.49	6.49	7.36
SMSO MLP	8.67	8.61	7.40	6.96	6.27	6.33	7.37
Direct MLP	8.28	8.48	7.79	6.96	6.50	6.41	7.40
LSTM Encoder-Decoder	9.83	9.47	7.32	7.41	6.99	6.34	7.89
Direct SVR	10.21	11.06	9.32	7.19	7.10	6.57	8.58
Elman Encoder-Decoder	10.04	10.09	8.20	8.20	7.91	7.54	8.66
Kalman Filter (NTE)	8.69	9.97	10.36	8.29	11.09	8.39	9.47
Naive model	17.46	18.54	16.78	16.37	17.66	18.26	17.51

Table 11: Mean absolute errors (MAE) in MW for all models for the years from 2012 to 2017 when using the expanding window test approach described in section 5.2.1.

Model	2012	2013	2014	2015	2016	2017	Avg
Mean of three best models	3.06%	3.09%	2.82%	2.47%	2.22%	2.30%	2.66%
MIMO MLP	3.25%	3.28%	2.94%	2.64%	2.37%	2.41%	2.81%
Direct Kalman Filter	3.32%	3.18%	2.94%	2.65%	2.55%	2.51%	2.86%
SMSO MLP	3.48%	3.51%	3.12%	2.73%	2.40%	2.42%	2.94%
Direct Linear Regression	3.50%	3.46%	3.11%	2.65%	2.50%	2.51%	2.96%
Direct MLP	3.29%	3.46%	3.34%	2.73%	2.49%	2.47%	2.96%
LSTM Encoder-Decoder	3.80%	3.76%	3.08%	2.84%	2.56%	2.40%	3.07%
Elman Encoder-Decoder	3.94%	4.09%	3.52%	3.15%	2.96%	2.92%	3.43%
Direct SVR	3.88%	4.46%	4.39%	2.85%	2.64%	2.53%	3.46%
Kalman Filter (NTE)	3.45%	3.96%	4.17%	3.17%	4.33%	3.21%	3.71%
Naive model	6.74%	7.41%	6.79%	6.16%	6.52%	6.67%	6.71%

Table 12: Mean absolute percentage errors for the same test run as in figure 11.

A.2 Detailed cross validation forecast errors

Model	2011	2012	2013	2014	2015	2016	2017	Avg
Mean of three best models	6.75	7.18	6.99	6.46	6.12	5.76	5.98	6.40
MIMO MLP	7.04	7.45	7.50	6.89	6.34	6.09	6.26	6.80
SMSO MLP	7.08	7.61	7.67	6.91	6.41	6.10	6.29	6.87
Direct MLP	7.09	8.28	8.48	7.79	6.96	6.50	6.41	6.87
Direct Kalman Filter	6.99	7.91	7.44	6.92	6.86	6.41	6.42	6.92
LSTM Encoder-Decoder	7.37	7.73	7.58	6.99	6.43	6.80	6.54	7.06
Direct Linear Regression	7.39	8.02	7.74	7.03	6.53	6.36	6.56	7.09
Elman Encoder-Decoder	8.16	8.18	8.87	7.99	7.31	7.54	7.47	7.93
Direct SVR	8.00	10.21	11.06	9.32	7.19	7.10	6.57	8.49

Table 13: Mean absolute errors (MAE) in MW for all models for the years from 2011 to 2017 when using a cross validation approach for testing where the models are trained on an equal amount of data for each year.

A.3 Varying the number of training years - Results for each submodel

Training years	2012	2013	2014	2015	2016	2017
One year back	8.19	8.60	7.84	7.90	6.76	6.93
Two years back	X	8.08	7.49	7.17	6.59	6.50
Three years back	X	X	7.05	6.84	6.24	6.43
Four years back	X	X	X	6.75	6.21	6.08
Five years back	X	X	X	X	6.19	6.44
Six years back	X	X	X	X	X	6.31

Table 14: MAE of MIMO MLP model in MW.

Training years	2012	2013	2014	2015	2016	2017
One year back	8.56	9.17	8.11	7.81	7.12	7.05
Two years back	X	8.81	7.75	7.33	6.95	6.74
Three years back	X	X	7.26	7.17	6.51	6.43
Four years back	X	X	X	6.94	6.47	6.32
Five years back	X	X	X	X	6.53	6.58
Six years back	X	X	X	X	X	6.37

Table 15: MAE of SMSO MLP model in MW.

Training years	2012	2013	2014	2015	2016	2017
One year back	8.07	8.04	7.13	6.83	7.31	6.59
Two years back	X	7.73	7.14	6.85	7.19	6.48
Three years back	X	X	7.06	6.87	6.79	6.55
Four years back	X	X	X	6.78	6.56	6.43
Five years back	X	X	X	X	6.49	6.40
Six years back	X	X	X	X	X	6.42

Table 16: MAE of direct Kalman filter model in MW.