Time series forecasting using LSTMs
Project 3 - FYS-STK4155

David Andreas Bordvik*
Department of Informatics, University of Oslo

Gard Pavels Høivang† and Are Frode Kvanum‡
Department of Geoscience, University of Oslo
(Dated: December 17, 2021)

Background: This part would describe the context needed to understand what the paper is about.

Purpose: This part would state the purpose of the present paper.

Method: This part describe the methods used in the paper.

Results: This part would summarize the results.

Conclusions: This part would state the conclusions of the paper.

INTRODUCTION

The Norwegian and European power markets are changing. Several trends and demands in todays society result in increased electricity usage and power consumption. Additionally, a higher integrated fraction of renewable energy production coupled with tighter integration of power transmission infrastructure between countries affects several aspects within the power market, such as; electricity price, electricity production, electricity transmission, and electricity demand. A tighter connection of power infrastructure between countries also connects the power markets between countries to a greater extend, giving room for a growing number of market participants and competition. In the light of a changing and increasingly competitive power market, the need for forecasting the electricity price has become a fundamental process for energy companies. The ability to make qualitative decisions through forecasting the electricity price has never been more relevant. [17]. Besides the aforementioned, studying the electricity price from a consumer standpoint considering the ongoing extreme increase in electricity price in the Norwegian market is the main motivation for this project. In this project, we focus on investigating the electricity price in the context of time-series analysis and forecasting the electricity price for the next day at an hourly resolution (24 hours).

There are five power regions in Norway named NO1, NO2, NO3, NO4, and NO5. Each region is a power market of its own, having its electricity price, and the regions can also be referred to as a bidding zone Figure (1). The dotted lines in the bidding zone figure represents export and import transmission cables. The bidding name reflects the trading part, where participants buy and sell electricity. Each bidding zone has a day-ahead auction where electricity is traded for delivery the next day. In the day ahead market (the spot market), Nordpool finds the equilibrium between expected power demand and power production between aggregated quantities among all traders. Nord pool then sets the reference (system) price based on the equilibrium, sales, and purchase orders. The hourly electricity prices are then finalized for the next day, named day-ahead price or spot price. Nord pool closes the day-ahead market at 12:00 each day, and the hourly prices for the next day are published thereafter [18]. Even though the different zones are individual markets, they are still related to each other in many ways. All zones are interconnected via transmission cables, thus enabling power trading across bidding zones. All power markets and systems must be in balance at all times to ensure a stable and reliable power source. Therefore, power transmission is a major player affecting each power market's core, including the price. Another driver affecting the price is the cost of producing electricity. An increasingly integrating power production from renewable sources gives rise to new challenges since some renewable power sources are volatile in nature, e.g., wind power and solar. Having volatile power-producing contributors gives rise to added uncertainties to the electricity price. Therefore, highly accurate forecasting of the electricity price is regarded as very challenging considering the characteristics of the electricity price and the power markets [5, 12, 15]. Due to time constraints, this project only focuses on forecasting the electricity price for the southern part of Norway (bidding zone NO2).

In the two previous projects, we studied how spatial and time-independent data could be used to fit an arbitrary function with a given error following the Universal Approximation Theorem [14] using a fully connected Neural Network. However, in Project 3, our data is sequential and time dependant. As such, a fully connected Neural Network is not designed to remember and make use of

* davidabo@mail.uio.no
† gardph@mail.uio.no
‡ afkvanum@mail.uio.no

FIG. 1. Bidding zones - https://www.nordpoolgroup.com/the-power-market/Bidding-areas/

historical content while feeding forward its current inputs. Therefore, this project will study Recurrent Neural Network and state-of-the-art methods for time-series data.

In the coming sections, we will start off by defining and structuring our dataset, set by the constraints imposed by the different models. Moreover, we briefly cover time-series data and point out the main differences between our data and the Terrain dataset used in Projects 1 and 2. Furthermore, we will develop the framework for both classical statistical models, as well as a sliding window algorithm that can structure our data to be used by both the Feed Forward Neural Network we developed in Project 2 as well as the Recurrent Neural Networks we will study in this Project. After developing our Theory and presenting our Models, we will study their behavior in light of our dataset. Our discussion will cover topics such as central problems regarding RNN performance, such as the unstable gradients and its short term memory as well as comparing the different models with each other. Moreover, we will try to optimize our developed models such that we are able to accurately forecast the electricity price for the next 24 hours. Finally, we will give our final thoughts as we summarize our findings.

DATA

The data used for this project is collected via the Entsoe Transparnecy Platform API. [16]. We also look at "fyllingsgrad" over Norwegian dams, which represents potential power generation. The data is publicaly provided via The Norwegian Energy Regulatory Authority (NVE) [1]. We constructed our own data based on the ones downloaded from the two different APIs. The constucted dataset used for this project can be found in GitHub linked in the appendix, in the data folder. The file used is named MAIN_DATASET.csv. The file contains 51649 rows and 20 columns containing variables related to power region 2 and 5 in Norway.

Pre-processing

The dataset has been preprocessed such that it does not contain any NaN values, and the missing values are filled with the previous hours. The data from NVE, "fyllingsrad" initially has weekly time-steps, while the entsoe data has an hourly frequency. Since the nature of the data from NVE, "fyllingsgrad", turned out the be very low-frequency data, we interpolated hourly between weeks to match the hourly time-steps in the entsoe data. Lastly, we shifted the entire dataset to match the closing and opening of the bidding process for the day-ahead market. Thus, our first record starts at 12:00, not 00:00. The electricity price varies within each hour, but the price for the next 24 hours is set at 12:00 the day before. As such, one time-steps for the electricity price is actually 1 day (24 hours) not 1 single hour. It is essential that the model learns the patterns leading up to when Nord pool sets the system price before the market closes and the hourly prices for the next 24 hours are settled. If the model had been trained on intervals between two market days, it would not be synchronized with the power market and our task of forecasts the next day.

When feeding data into the RNN, which will be described in greater detail in the coming sections, the data has to be formatted to take the form of the 3D array $(\text{Batch Size}, \text{Time Steps}, \text{dimensionality})$. Specifically, Batch Size refers to the number of samples, Time Steps reflect the historical sequence. Finally, dimensionality refers to the number of time series used, such that dimensionality $= 1$ would refer to a univariate time series. Some literature refers to the last dimension as the number of features. We also normalized the data using a minmax scaling, where the data is transformed to fit the range between 0 and 1.

THEORY AND METHODS

Data Windowing

Working with time-series data often requires formulating the time-series data into a supervised learning problem. This can be done by organizing the data into chunks of historical and future data, where the future data is regarded as the target data. This allows for looking back at a sequence of historical data to predict the

future. We have applied a sliding window-based algorithm to structure the data into appropriate sample sizes that fit the constraints imposed by converting a time-series data stream to a supervised learning problem fitted for Recurrent Neural Networks. Compared to the previous projects, our data does not contain a relationship between some input data and some target variables; instead, we want to recognize the relationship throughout a sequence of values. Before sectioning the data into windows, we note that our data is evenly spaced which is a requirement for a time-series problem; in our case, we have an hourly time series. Table I visualize an example of how the different chucks of historical and future data can be collected from the original univariate price data stream. **t** represents the time step in an hour, **p** represents the price at the given time step, and n is the last record in the price data stream. The example in table I considers a historical sequence of 4 time-steps indicated by the grey color and a forecast horizon of 2 time-steps indicated as red. Each window including the historical sequence and future data is considered a sample in the constructed windowed dataset. The six leftmoste colums in table I indicates the sliding process with a stride equal to 1. The two right moste columns ilustrates in more detail the lookback and horizon steps including indexing.

t	**p**	**t**	**p**	**t**	**p**	**t**	**p**
0	p_0	0	p_0	0	p_0	0	p_0
1	p_1	1	p_1	1	p_1	1	p_1
2	p_2	2	p_2	2	p_2	$\cdots$	$\cdots$
3	p_3	3	p_3	3	p_3	$\cdots$	$\cdots$
4	p_4	4	p_4	4	p_4	$n-4$	p_{n-4}
5	p_5	5	p_5	5	p_5	$n-3$	p_{n-3}
6	p_6	6	p_6	6	p_6	$n-2$	p_{n-2}
7	p_7	7	p_7	7	p_7	$n-1$	p_{n-1}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	n	p_n
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$n+1$	p_{n+1}
n	p_n	n	p_n	n	p_n	$n+2$	p_{n+2}

TABLE I. Visualization of the sliding window process with stide =1, lookback sequence = 4 and horizon = 2

For our windowed dataset, the window algorithm starts off by sectioning the data into 24-hour chunks, having a stride equal to 24. This allows some flexibility when specifying the input width in terms of a varying number of days. Secondly, a window of length input width and forecast horizon containing 24-hours large cells with a final cell the size of the forecast horizon strides through the chunks with a stride of 1 day (in other words it moves one chunk at a time). This generates a consecutive sample from our data, which is then restructured together with the other generated consecutive samples to match the required 3D array format imposed by the Recurrent Neural Network.

Since the chunks containing days are repeated in all data windows within the range of the sliding window, we are able to reuse data points in new isolated sequences as the window strides though the chunks. This allows us to effectively increase the number of samples, as contiguous samples contain overlapping data, though differ with regards to the first day used in the input width and data used as forecast horizon. This data window structure allows us to forecast an arbitrarily amount of time ahead, based on the data from a specified number of days ago.

Finally, during train test split, a single data window sample is split in such a way that the input width defines our X (input signal), and the forecast horizon specifies labels y. If the labels y have a length of 1, the RNN is a sequence to vector model predicting a single time step ahead. However, a label vector y with length greater than 1 would not itself specify whether the Recurrent Neural Network is a vector or sequence outputting model, as that would have to be specified in the model declaration. However, a length greater than 1 would indeed indicate that the model is forecasting several time steps ahead.

Time-series and statistical analysis

The nature of time series

As already mentioned, in contrast to projects 1 and 2, this project deals with data in a sequential matter, called time series. What defines this type of format is that the data points decrease, increase or change in chronological order throughout time[4]. A formal definition could be formulated as a mapping from a time-domain to real numbers:

$$x : T \to \mathbb{R}^k \qquad (1)$$

where $T \subseteq \mathbb{R}$ and $k \in \mathbb{N}$ [3]

In the scope of this project, our time series consist of discrete values that describe how price, actual power generation, generation forecast, actual load(demand), load forecast, the delta between forecast and actual, and "fyllingsgrad" changes over time. Dealing with data in such a format might yield additional complexity when trying to analyze the raw data at hand. A widely used yet simple approach for analyzing and forecasting time series includes only the variable destined for forecasting. When analyzing time series on this form, we do not attempt to explain the observed data by the relationship with other possible variables. The only relevant data used for analyzing and forecasting is the past of the variable itself. Data in this format is called a univariate time series, as a contrast to a multivariate time series, which includes several variables connected through a shared period of time. Methods for forecasting the latter will be discussed thoroughly in later sections, but the following section will focus on univariate time series leading up to the Auto-

Regressive model, as this works as the most fundamental building block for time series forecasting [11]

Decomposing the Characteristics

As previously mentioned, time series consist of data that describes how different variables change through time. As a result of this aggregation of data, specific characteristics of interest can be decomposed from the original time series. These characteristics can highlight certain aspects of the data in question and help in better the understanding and analysis of the data. To be able to decompose these characteristics, we have made use of the methods implemented in the python module Statsmodels [19].

This will help us in visualizing three different components from the time series, in addition to the original data. The first component produced from this method is the trend. This is produced by implementing the method seasonal_decompose. from the Statsmodels module. This method estimates the trend by applying a convolutional filter [19].

A trend is a general direction in which something is changing [3]. This trend is, of course, highly dependent on the scope of the time period in question, and the direction of change can differ for different parts of the time series [4] Another component that can be highly descriptive for visualizing the time series is the seasonal component. These are patterns in the data with a repeating cyclic variation. After the removal of these components, we are left with the residuals. This is the irregular variation, not explained by the above components[4].

The Auto-Regressive Model

Stationarity

With the different characteristics of the time series decomposed, we move forward with implementing a model with the goal to be able to forecast the price of electricity a given amount of steps ahead in time. As already pointed out, this will be done solely based on the history of the price itself. A general challenge with univariate modeling may arise if the above-mentioned trend is present in the data. If this is the case, the data series is *non-stationary*, and might prove difficult to forecast. As almost every time series will have varying data to some degree, we introduce the Augmented Dickey-Fuller(ADF) test as a statistical test for stationary checking. This is done by implementing the "adfuller" method from Statsmodel [19]. We state a H_0 that the unit root is present in the time series.The ADF will return a p-value, where a value smaller than 0.05 will result in the null hypothesis being rejected. Failing to reject this hypothesis will imply that the time series has a non-stationary structure [4], and will need further processing before being used for forecasting.

In the case of the latter, a typical method for further preprocessing is the use of differencing. Using this method creates a new time series, consisting of the differences between each value and the following [11]. This new data series is then used to compute a new statistical feature, namely autocorrelation. As forecasting with a univariate time series naturally only includes the variable itself, finding patterns on how the past influences the future is crucial. Plotting the autocorrelation gives us a clear visual representation of how the variable is correlated with itself at different time lags. This proves, in other words, a valuable method for determining the linear relationship between time step t and $t-1$ [4]. For the scope of this project, this will give us a measure of how much the price of electricity at a given hour is correlated with the hours ahead. As with regular correlation, the autocorrelation can be both positive and negative [11]. However, the use of *lags* is a concept in autocorrelation that differs from the regular. This notion deals with the number of time steps in which the correlation between each point should be calculated. Another vital metric closely connected with autocorrelation is partial autocorrelation. This statistical interpretation of the time series also deals with the correlation between lags, the difference being the latter having removed the effect of in-between time lags[4]. They both are plotted against the specific time lag with a confidence interval, yielding a tool for visually inspecting significant time lags.

Model Definition

The before mentioned autocorrelation is the basis for the model we then implement for univariate forecasting. At the base of this model is its nature of predicting future variables as a function of the lagged values. This model is formulated as:

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t$$

Where X_t is the current time step, ϕ is a coefficient for a specific lag, and ϵ is estimated random noise, for all p time lags [11]. Our implementation of this algorithm can be found in the GitHub repository. As previously stated, in this project, our goal is to predict prices for electricity 24 hours ahead. After conducting a grid search, we found that the optimal order of our AR model for solving this problem is 24. This comes as no surprise, as electric pricing has a 24 hours cyclic variation. In this case, where our objective is to forecast multiple time steps, we implemented an iterative approach. As demonstrated by the above equation, the number of previous time steps included in the prediction is dictated by order of the

AR model. Our first prediction is, therefore a result of our ϕ coefficients being multiplied and summed with actual observed data from the original time series. For a 1 hour prediction, the expected prediction should as a result of this, be fairly accurate. However, when moving further along with the next predicitions, a increasingly larger part of the values from the previous time steps will be predicted values, rather than actual observed. This creates a situation where one runs the risk of aggregating errors throughout the predictions, which is a known weak spot for iterative algorithms of this manner.

As above equation demonstrates, the AR model of order p needs a total number of p specific ϕ coefficients. The optimal value for these coefficients is calculated by fitting the model to the data. To achieve this, we have implemented another function from the Statsmodel module [19]. This method is partly based on a central algorithm from project 1, namely Ordinary Least Squares[11].

The Auto-Regressive Model

Stationarity

With the different characteristics of the time series decomposed, we move forward with implementing a model with the goal to be able to forecast the price of electricity a given amount of steps ahead in time. As already pointed out, this will be done solely based on the history of the price itself. A general challenge with univariate modeling may arise if the above-mentioned trend is present in the data. If this is the case, the data series is *non-stationary*, and might prove difficult to forecast. As almost every time series will have varying data to some degree, we introduce the Augmented Dickey-Fuller(ADF) test as a statistical test for stationary checking. This is done by implementing the "adfuller" method from Statsmodel [19]. We state a H_0 that the unit root is present in the time series.The ADF will return a p-value, where a value smaller than 0.05 will result in the null hypothesis being rejected. Failing to reject this hypothesis will imply that the time series has a non-stationary structure [4], and will need further processing before being used for forecasting.

In the case of the latter, a common method for further preprocessing is the use of differencing. Using this method creates a new time series, consisting of the differences between each value and the next [11]. This new data series is then used to compute a new statistical feature, namely autocorrelation. As forecasting with a univariate time series naturally only includes the variable itself, finding patterns on how the past influences the future is crucial. Plotting the autocorrelation gives us a clear visual representation of how the variable is correlated with itself at different time lags. This proves, in other words, a valuable method for determining the linear relationship between time step t and $t-1$ [4]. For the scope of this project, this will give us a measure of how much the price of electricity at a given hour is correlated with the hours ahead. As with regular correlation, the autocorrelation can be both positive and negative [11]. However, the use of *lags* is a concept in autocorrelation that differs from the regular. This notion deals with the number of time steps in which the correlation between each point should be calculated.

Model Definition

The before mentioned autocorrelation is the basis for the model we then implement for univariate forecasting. At the base of this model is its nature of predicting future variables as a function of the lagged values. This model is formulated as:

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t$$

Where X_t is the current time step, ϕ is a coefficient for a specific lag, and ϵ is estimated random noise, for all p time lags [11]. As equationXXX shows, the AR model of order p needs a total number of p specific ϕ coefficients. The optimal value for these coefficients is calculated by fitting the model to the data. To achieve this, we have implemented another function from the Statsmodel module [19]. This method is partly based on a central algorithm from project 1, namely Ordinary Least Squares.

RECURRENT NEURAL NETWORKS THEORY AND METHODS

Sequence processing using Recurrent Neural Networks

As our data is sequenced over time, we need a Neural Network that can share its weights across time steps. In a Recurrent Neural Network, this is done by applying the same update rule to the previous output when producing a new output [7]. Thus, since a Recurrent Neural Network shares its weights across several time steps, it is suitable to use for temporal data then a standard FeedForward Neural Network as touched upon in the introduction.

Mathematically, we can define this system as

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

Where $\boldsymbol{s}^{(t)}$ is the state of the system, $\boldsymbol{x}^{(t)}$ is an external signal which drives the system and $\boldsymbol{\theta}$ parametrizes f, though we note that this system itself does not contain any output.

Recurrent Neural Networks support multiple different input and output sequences [9]. A tuple of inputs or outputs over different time steps is considered a sequence,

whereas a input or output from a singular time step is considered a vector. RNNs support different constellations of sequence/vector input and output, such that it is possible to input a sequence and output a sequence where some time steps are ignored. This is a favorable property given our data and task of producing a full day forecast some time steps after the end of our input sequence.

Backpropagation through time

Though computing the gradient through a Recurrent Neural Network is analogous to how it was computed in a Feed Forward Neural Network, the algorithm has to be applied throughout all the time steps of the Recurrent Neural Network recursively. The recursion starts at the final time step, where at the final time step τ the gradient of the loss L can be computed as follows, using the notation of [7]

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^T \nabla_{\boldsymbol{o}^{(\tau)}} L \tag{2}$$

where $\boldsymbol{V}$ is the weights between the hidden states and output values $\boldsymbol{o}$. Going backwards trough time ($t < \tau$), the gradient of the propagated loss can be computed using the following equation

$$\boldsymbol{W}^T J(\boldsymbol{a})(\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^T (\nabla_{\boldsymbol{o}^{(t)}} L) \tag{3}$$

where $\boldsymbol{W}$ is the weights associated with the hidden-to-hidden state connection and $J(\boldsymbol{a})$ is the jacobian of the activation function.

Given a long time sequence, a Recurrent Neural Network can be both memory and cpu intensive. Moreover, for long sequences of data, the unrolled RNN will be a considerably deep Neural Network. As we discussed in Project 2, a deep Neural Network is prone to the unstable gradient problem, where the gradient can either completely vanish or explode through the bounds of the datatype used to contain the value. In addition to the RNN being prone to unstable gradients given a long sequence, the memory of the first inputs will deter if the RNN is processing a long sequence [9].

The unstable gradient problem

Our main motivation behind introducing different activation functions such as the ReLU and Leaky ReLU to replace the Sigmoid activation function in Project 2 was to avoid the vanishing gradient which arose from the bounded output from the Sigmoid. However, a non-saturated activation function such as the ReLU family of functions runs the risk of updating the set of weights between two layers in such a way that the computed output is slightly increased. For a Feed Forward Neural Network, given how additional sets of weights are computed during gradient descent, this weight update pattern is not given to reoccur. Though for a Recurrent Neural Network, the same sets of weights are updated at every time step, such that a weight update pattern that ends up in a slight increase of the output would be reapplied until the output explodes [9]. As such, when implementing a Recurrent Neural Network in the coming Section, we will use a saturating activation function such as the Hyperbolic Tangent $\tanh = \frac{\sinh}{\cosh}$, which conveniently is supplied as the default activation function for RNNs in TensorFlow [2].

The Short-Term memory problem

The second problem which arises in the context of training a Recurrent Neural Network is its inevitable memory loss of the earliest states as a result of how some information is lost for each time step [9]. By following the approach of [10], implementing Long Short-Term Memory (LSTM) cells can alleviate the memory loss. The idea behind LSTM cells is to store some information in the long-term, while at the same time forget unnecessary information whilst reading the rest which result in the short term output memory. In this way, for each time step some information is kept, while at the same time some long term memories are dropped if they meet certain requirements after each time step [9]. As such, LSTM cell can be able to recognize, preserve, utilize and abolish trends in the data as needed, acheiving a better mean squared error than for the simple RNN cell. Another gated RNN which might alleviate the short term memory problem of Recurrent Neural Networks is the Gated Recurrent Unit, which simplifies the LSTM cell by combining the forget and output gate into one single gate [7]. additionally, the GRU cell merges both the long term memory state and the hidden state into a single vector. Though a GRU cell would outperform an LSTM cell in terms of computational efficiency [9], GRU as well as other gated RNNs derived from the LSTM cell performs approximately the same as LSTM [8]. A long sequence can also be shortened by applying a 1-dimensional convolutional layer as a pre-processing layer, before passing the shortened signal on to an RNN. Given our current application, the convolutional layer will effectively downsample a given input sequence, given the size of the convolutional kernel [9]. The convolutional layer will both detect structures in the sequence as well as shortening the sequence. This in turn can benefit the RNN in terms of its memory, as it will be able to remember longer sequences than in the absence of a preprocessing convolutional layer. Another preprocessing layer which might improve the performance of a RNN is a FeedForward layer. The FeedForward layer will project the input into a feature space with temporal dynamics, which may yield improved performance. [13]

Throughout this project, we will study both the effect of the aforementioned architectural differences, namely the difference between a sequence to vector and sequence to sequence RNN when predicting several time steps ahead.

Furthermore, we will also study the effect of the different preprocessing techniques outlined above in response to both the gradient and memory loss problem. All in context of the ENTSOE dataset.

Setting up our RNN

What follows is the basic structure that our developed RNN models will attain, following the constraints and conditions outlined in the previous sections. For clarity, all RNNs will be developed using the Keras frontend of TensorFlow [2]. For a complete implementation of the models, we refer to the GitHub linked in the Appendix.

We have developed models that are able to forecast a single, as well as several time steps ahead in time both via a vector and sequence approach. The main difference between a sequence to vector and sequence to sequence in terms of the RNN architecture is how the final recurrent layer returns its output. For a sequence to vector model, the only the final output node is considered, whereas for a sequence to sequence model we set the *return_sequences* flag to **True** as that would return all computed output nodes distributed through time. Finally, regardless of model architecture, a regular Dense feed forward layer distributes the output of the Recurrent Layer according to the specified forecast Horizon. Note that a Dense layer in TensorFlow supports sequences as input, meaning that it can handle the output from a recurrent layer regardless of vector or sequence form [9].

A preprocessing layers such as a 1-dimensional convolution layer or a FeedForward layer can be implemented akin to the layers discussed above, taking the place as the input layer for their specific model.

As stated previously, the hyperbolic tangent function is sufficient as activation function for an RNN due to its saturating property. As such, our RNNs will default to the hyperbolic tangent as activation function. Moreover, an adaptive learning rate optimizer during gradient descent is chosen. For our models, we will default to the ADAM optimizer.

RESULTS

We note that all results have been created with the Source Code found in the Appendix. The results are reproducible, for further explanation for how to run the code and reproduce the results, refer to the README in the GitHub. All color maps used have been developed to fairly represent all data, with an emphasis of being readable for those with visual deficiency [6].

Recurrent Neural Networks

Behaviour of uni -and multivariate models

These results are part of a preliminary study to determine how the different Recurrent Neural Network Cells respond to a change in number of neurons. For all results, each model is created equal according to the choice of parameters, except for the Cell structure. In the univariate case, only price is used. Moreover, in the multivariate case, forecasted values and actual values are assigned to specific models. For clarity, no results show the inclusion of the NVE fyllingsgraddata. All results for the current subsection are made using a Sequence to Sequence model.

Figure (16) shows the Mean Squared Error as a function of number of Neurons for a single layered, univariate model. Whereas Figure (17) shows the Mean Squared error for the same models, but this time using a multivariate input of price, forecast load and forecast generation.

Figures (18) and (19) display a Grid Search approach where Mean Squared Error is dependant on the number of neurons in each layer of a two layered Deep Recurrent Neural Network. Figure (18) is for the SimpleRNN cell whereas Figure (19) is for the GRU cell.

Data alteration for long sequences

In the context of our data, we refer to a long sequence as a input width of 168 hours, i.e. one week of data. We set up an RNN with with two GRU cells each with 128 neurons, as an immediate response to the MSE values obtained in previous results. We then performed different experiments by tweaking individual aspects of the RNN while keeping the other constant. All results for this subsection are also made using a Sequence to Sequence model, as for the results in the subsection above.

Figure (20) displays a Grid Search performed over different architectures where one or two Dense Feed Forward layers preceded the GRU layers in the RNN.

Figure (21) shows a Grid Search performed over different output space dimensionality and kernel sizes, with a fixed stride of 3.

Finally, Figure (22) presents a Grid Search performed over different combinations of dropout and recurrent dropout.

DISCUSSION

Recurrent Neural Networks

As stated in the Introduction, we have developed a Recurrent Neural Network to study time-series data. For this, we have utilized the frontend Keras for the Machine Learning library TensorFlow. In this section, we will limit ourselves to the study of our Sequence to Sequence

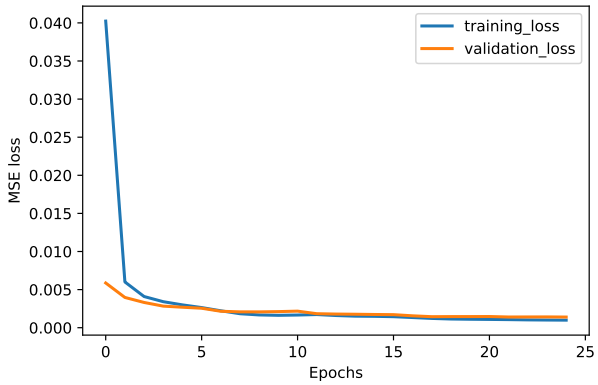

FIG. 2. Univariate train evaluation RNN

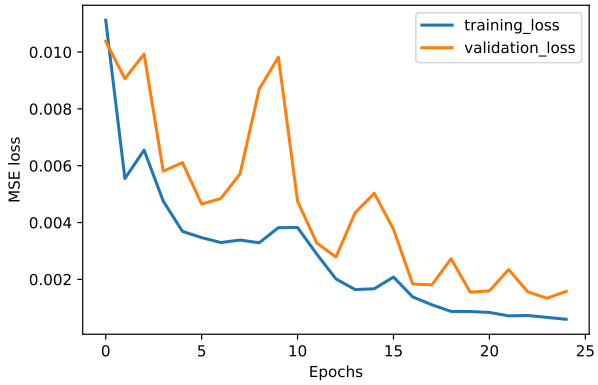

FIG. 3. Univariate train evaluation conv GRU

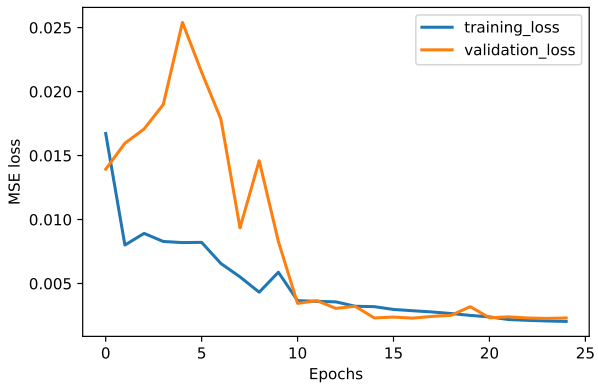

FIG. 4. Univariate train evaluation LSTM

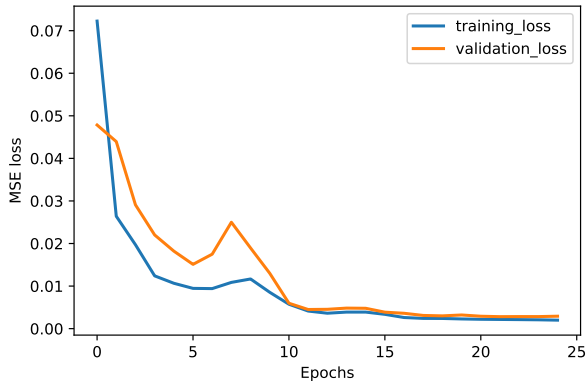

FIG. 5. multivariate train evaluation RNN

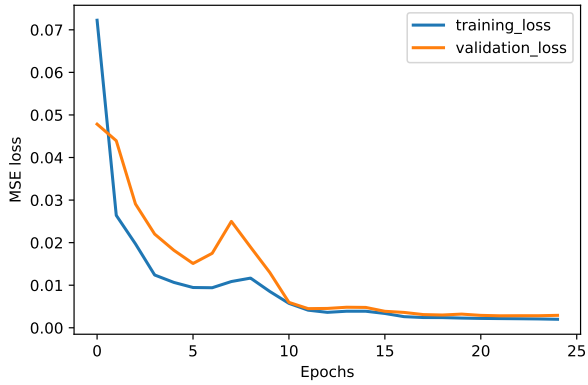

FIG. 6. multivariate train evaluation conv GRU

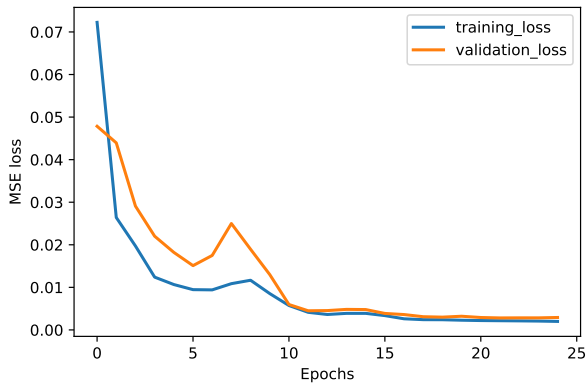

FIG. 7. multivariate train evaluation LSTM

model approach, which predicts 24 hours into the future at every time step instead of at just the final time step. This increases the number of error gradients propagated backwards through our model [9].

By inspecting Figure (16), it can be seen how the MSE varies as a function of number of neurons for a single layered, univariate model with price as its only feature. By following the curve, as the number of neurons in the layer increases, the lower the MSE becomes. As the MSE is a negatively oriented score, this is a favorable response. Moreover, though all three cells performs somewhat equal, the SimpleRNN cell attains the lowest MSE score with the highest number of neurons, i.e. 128. This may be a reasonable as this is for a univariate model, where the amount of data is sufficient for an SimpleRNN cell to fit the data appropriately.

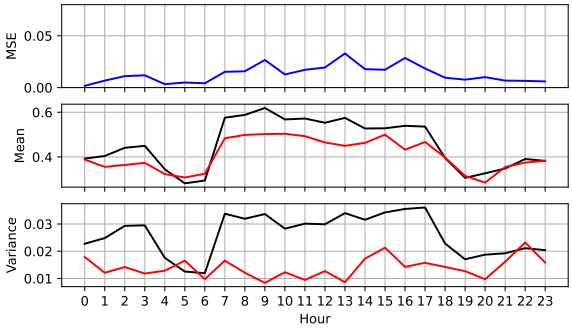

FIG. 8. Univariate Simple RNN performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

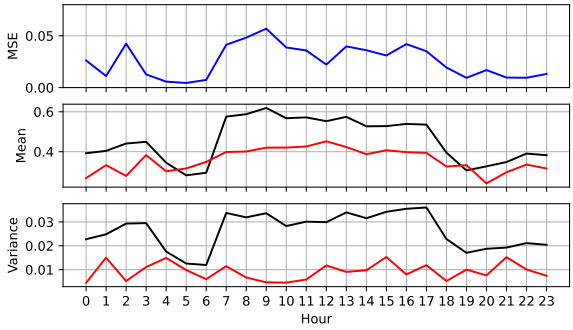

FIG. 11. Multivariate Simple RNN performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

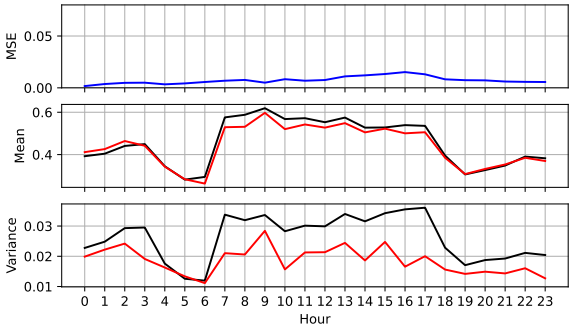

FIG. 9. Univariate ConvGRU performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

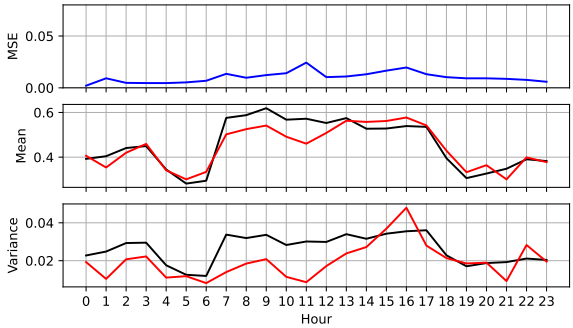

FIG. 12. Multivariate ConvGRU performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

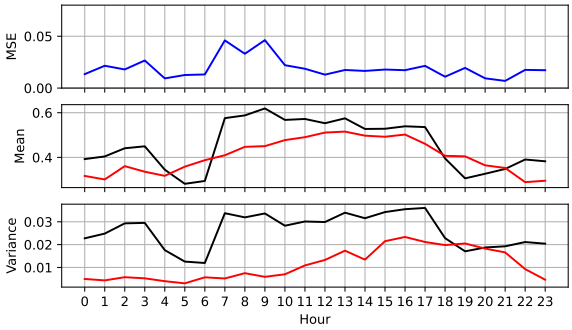

FIG. 10. Univariate LSTM performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

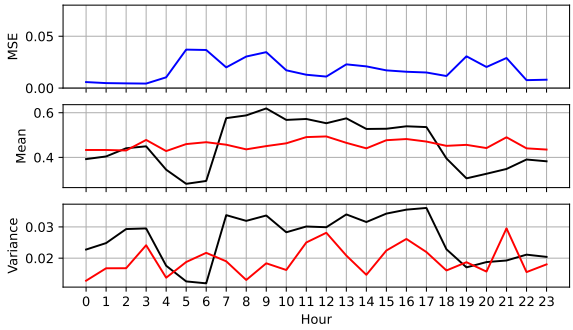

FIG. 13. Multivariate LSTM performance - Blue color indicate MSE for spesific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

Furthermore, by moving over to Figure (17), some similarities may be drawn as the SimpleRNN is having a harder time fitting to the data than the LSTM and GRU cell models. As such, by introducing to more features to be used when predicting the price, the SimpleRNN may be overloaded with data such that it is having a harder time extracting the relevant structures for further predictions. However, this may not be the case for the LSTM and GRU cell networks, where it seems that their internal gates are able to extract and remember the relevant structures even with a lower neuron count. Interestingly, the LSTM network performs better with the multivariate data than

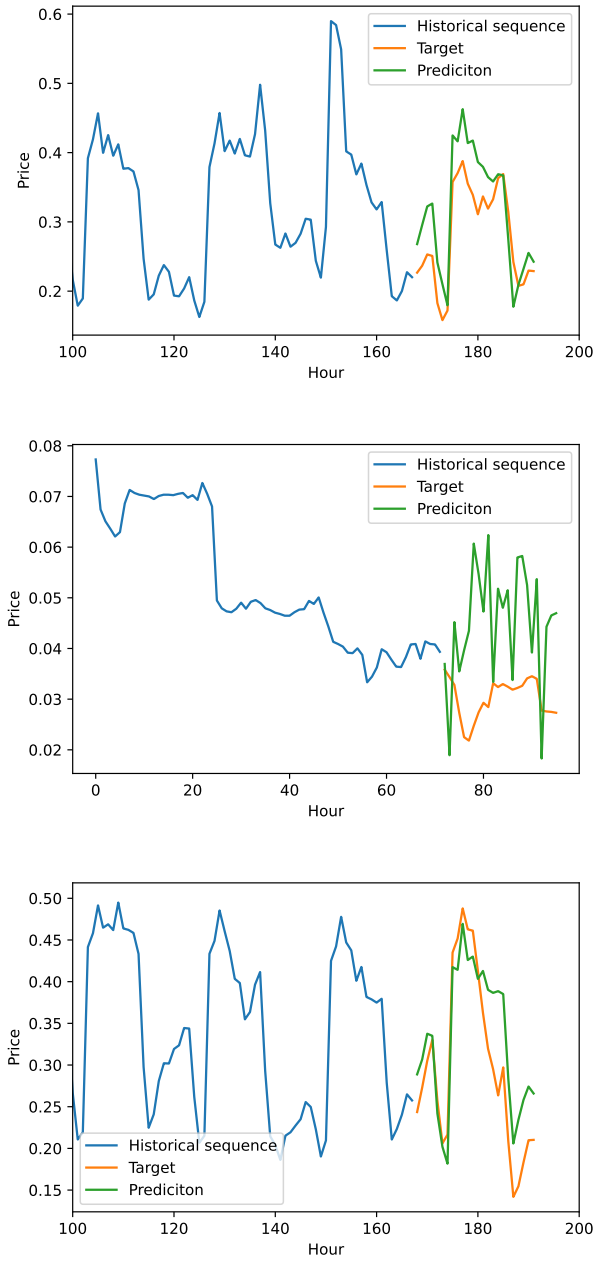

FIG. 14. Three sample predictions from the best univariate model GRU with 1d convolution

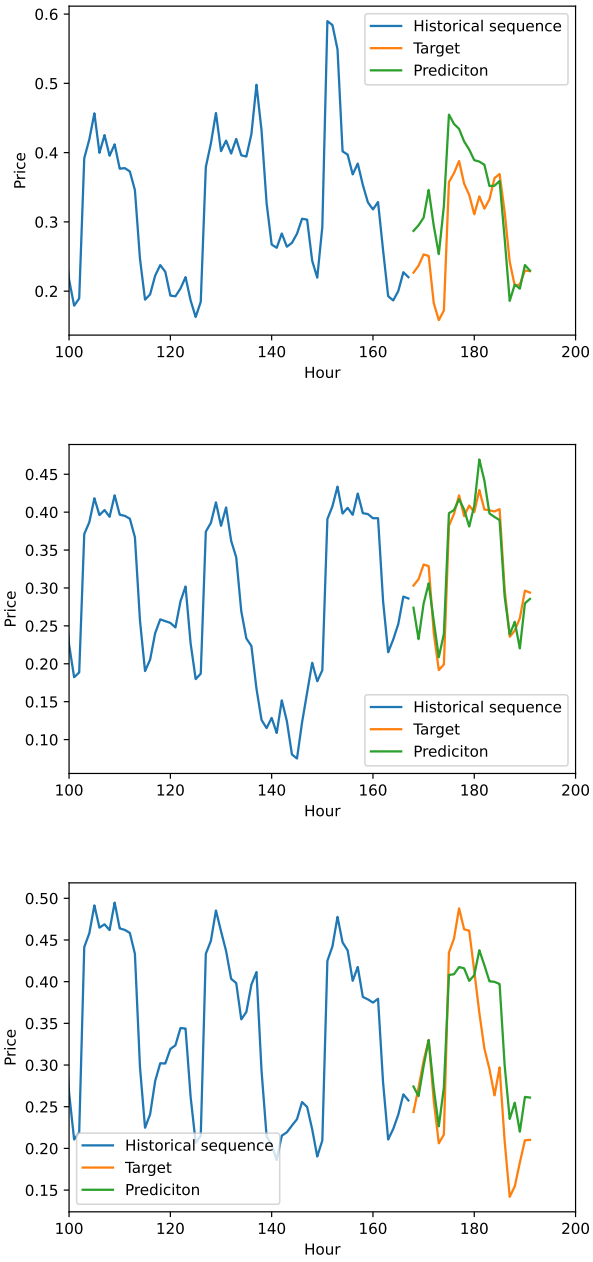

FIG. 15. Three sample predictions from the best multivariate model GRU with 1d convolution

the univariate data for both 16 and 32 neurons. Thus, Figures (16) and (17) may indicate that the SimpleRNN is sufficient for univariate data, while for the multivariate case it is more suitable to use an LSTM or GRU. This may be a consequence of the short-term memory problem of SimpleRNN cells, as the multivariate feature space is three times as large as the univariate feature space. Thus, having a long-term memory component as found in the LSTM and GRU cell is necessary to not discard relevant information during training.

Figures (18) and (19) are Grid Search approaches to determine how the Mean Squared Error responds to a change in architecture for a two layered dense multivariate model. Inspecting Figure (18), it can be seen that the MSE responds somewhat similarly to the multilayered model as for the single layered SimpleRNN model. Namely that the MSE attains lower values as the complexity of both layers are increased. This trend is similar for Figure (19), which displays the same Grid Search but for a GRU model instead. It can easily be seen that the lowest MSE is attained with the most complex model.

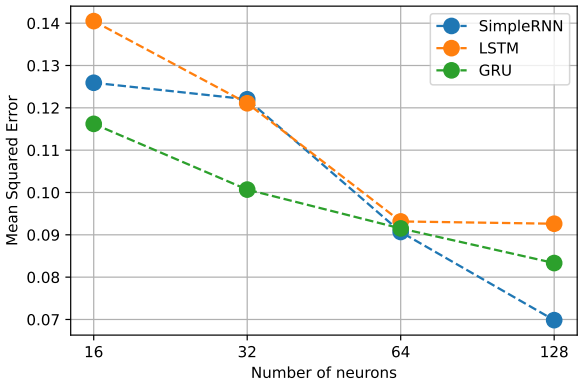

FIG. 16. Mean Squared Error as function of number of neurons for a single layer, univariate model using price as its feature

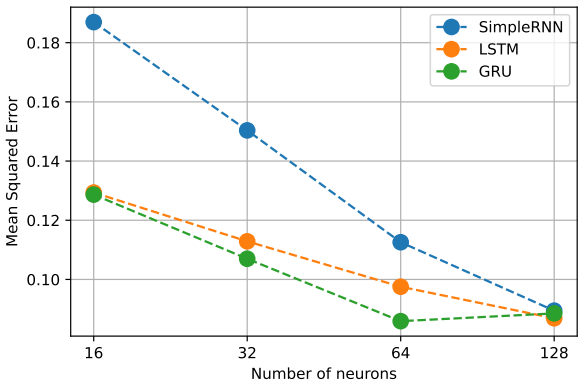

FIG. 17. Mean Squared Error as a function of number of neurons for a single layer, multivariate model using forecast values including price as its features

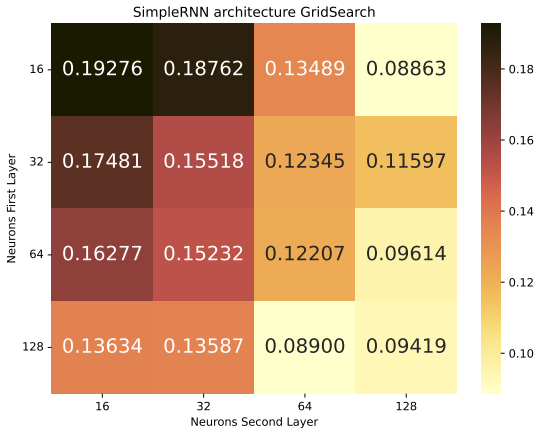

FIG. 18. Grid Search over different combination of layer size for a multilayered multivariate SimpleRNN network using the forecasted data

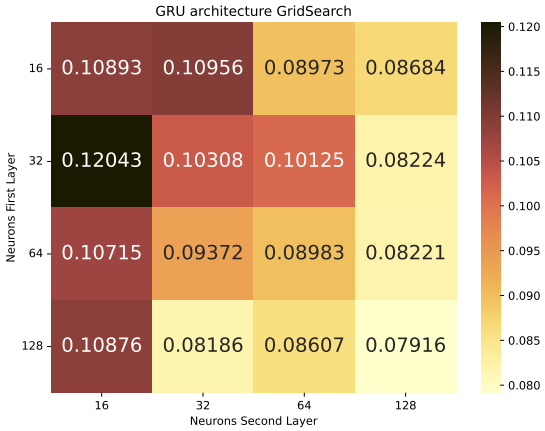

FIG. 19. Grid Search over different combination of layer size for a multilayered multivariate GRU network using the forecasted data

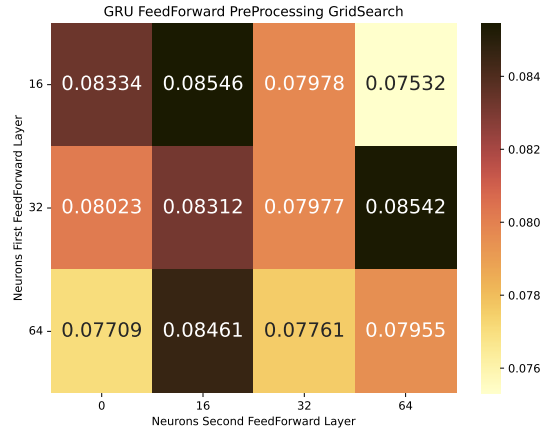

FIG. 20. Grid search over different architectures utilizing Dense layers preceding the GRU layers

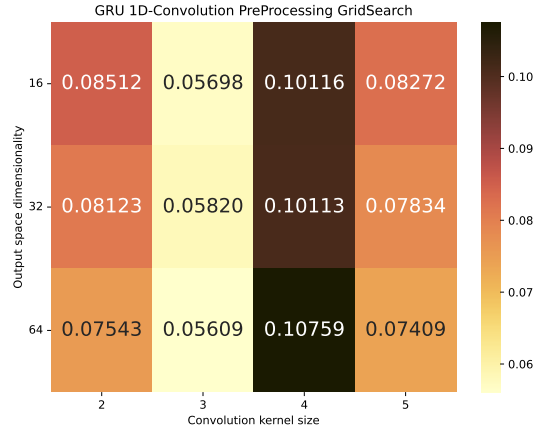

FIG. 21. Grid Search over different architectures utilizing a 1 Dimensional convolutional layer preceding the GRU layers

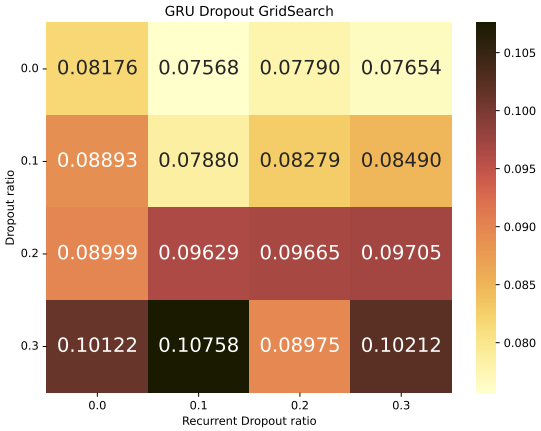

FIG. 22. Grid Search over different ratios of dropout and recurrent dropout.

Response to architectural changes

Effect of data alteration for long sequences

Recurrent Neural Network

CONCLUSIONS

Source Code

Link to github repository containing all developed code for this project: **CORRECT URL HERE**

[1] Magasinstatistikk - NVE.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Ben Auffarth. *Machine Learning for Time-Series with Python*. Packt Publishing, 2021.

[4] ASHISH PATEL B V Vishwas. *Hands-on Time Series Analysis with Python*. Apress, 2020.

[5] D.W. Bunn. Forecasting loads and prices in competitive power markets. *Proceedings of the IEEE*, 88(2):163–169, February 2000. Conference Name: Proceedings of the IEEE.

[6] Fabio Crameri. Scientific colour maps, 2021.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[9] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly UK Ltd., October 2019.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Joos Korstanje. *Advanced Forecasting with Python*. Apress, 2021.

[12] Wei Li and Denis Mike Becker. Day-ahead electricity price prediction applying hybrid models of LSTM-based deep learning methods and feature selection algorithms under consideration of market coupling. *Energy*, 237:121543, December 2021.

[13] Pushparaja Murugan. Learning the sequential temporal information with recurrent neural networks. *CoRR*, abs/1807.02857, 2018.

[14] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Last update: Thu Dec 26 15:26:33 2019.

[15] F.J. Nogales, J. Contreras, A.J. Conejo, and R. Espinola. Forecasting next-day electricity prices by time series models. *IEEE Transactions on Power Systems*, 17(2):342–348, May 2002. Conference Name: IEEE Transactions on Power Systems.

[16] https://www.entsoe.eu/data/transparency-platform/.

[17] Jakub Nowotarski and Rafał Weron. Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews*, 81:1548–1568, January 2018.

[18] Nord Pool. Bidding areas. https://www.nordpoolgroup.com/the-power-market/Bidding-areas/.

[19] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.