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)

In this project, we have studied data collected from the Entsoe transparency platform as well as the Norwegian Water Resoruces and Energy Directorate in an attempt to accurately forecast the day-ahead electricity price using both classical statistical approaches as well as using Machine Learning for the southern part of Norway. We have studied the power market, and the driver behind day ahead price. As the data we are inspecting is structured as a time-series, we have developed both an Autoregressive model, as well as a Recurrent Neural Network using the popular machine learning library TensorFlow. We experimented by applying different preprocessing and data generalization techniques to tune the Recurrent Neural Network, and we achieved the best MSE score by combining a 1-dimensional preprocessing layer with a deep Gated Recurrent Unit network. We have also studied the difference between a univariate RNN, with price being the only feature, as well as a multivariate RNN where load and generation from the Entsoe dataset has been included. We find no significant improvement in including additional features, though this may be a case of the limited number of features we have available. For the Recurrent Neural Network, we have mainly been studying a Sequence to Sequence approach. Finally, we have shown that the best MSE obtained was from the Autoregressive model, with an MSE = 0.00052. We conclude that though the Recurrent Neural Network show promise in forecasting day-ahead electricity price, our model does not contain enough relevant features to be able to accurately predict temporal price evolution beyond a simple mean value. Thus, we propose the Autoregressive model as a sufficient forecasting model when forecasting short term temporal evolution of day-ahead price given a feature limited or univariate dataset.

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 increased market coupling, a higher fraction of renewable energy production, and increasingly competitive power markets, 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]. Considering the aforementioned, studying what drives the power markets and the electricity price in the light of the current extreme prices motivates to this project. The extreme electricity price does not just affect the consumers of household electricity, it affects the society as a whole. To study the complex problem of electricity price forecasting, it was required for this project to tackle many different aspects such as: Studying of mechanisms of the power market, data collection trough API, data-preprocessing, time-series problem analysis, advanced time-series forecasting using both classical and neural network. For the forecasting part of this project, we settled on forecasting the electricity price for the next day at an hourly resolution (24 hours). The forecasting time-span is based on how the power markets in Norway are operating. Gaining domain knowledge on real complex problems is often imperative to success using a Machine Learning approach. However, achieving domain knowledge is no easy task, and this project also requires looking at the problem and the data from a non-statistical point of view.

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 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 briefly introduce the power market domain. Furthermore, we define

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

and structuring our dataset in the data section, 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.

THE POWER MARKET DOMAIN AND CHARACTERISTICS

There are five power regions in Norway named NO1, NO2, NO3, NO4, and NO5. Each region is a power producing region of its own, and each zone is considered its own power market where its local electricity price is set. The regions can also be referred to as a bidding zone Figure (1). The dotted lines in the bidding zone figure represents the Norwegian export and import transmission cables to neighboring countries. The transmission cables between the Norwegian zones are not drawn in the figure. 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 for each zone. Nord pool then sets the reference (system) price based on the equilibrium, sales, and purchase orders for each zone. 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, market coupling and power transmission is a major player affecting the core of each power market. Another driver affecting the price is within the markets 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].

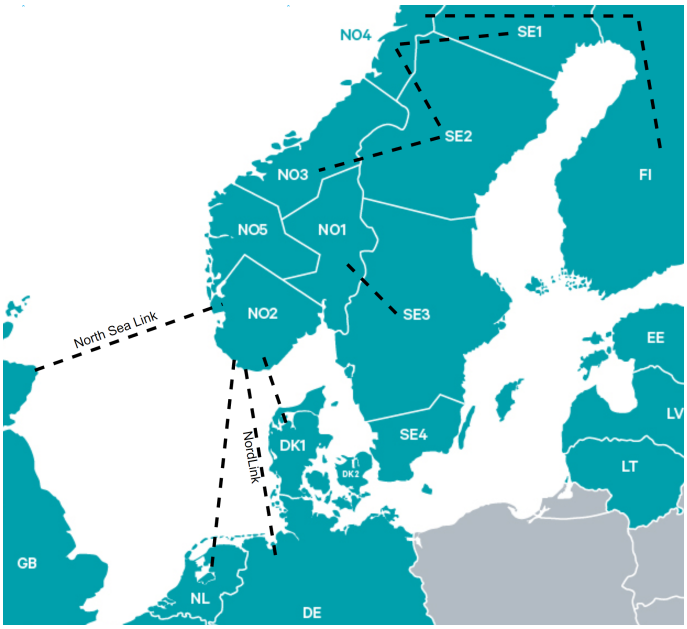

FIG. 1. Bidding zones - https://www.nordpoolgroup.com/the-power-market/Bidding-areas/

Market coupling

As previously mentioned, the different power markets are being tighter connected thorug interconnecting cables which enables flow of energy between bidding zones and countries. The dotten lines in Figure 1 indicates transmission infrastructure representing the Norwegian export and import of power. Transmission cables between the Norwegian zones are not shown in the figure. Reasontly there have been commisioned new interconnecting transmission cabels between Norway and Germany (NordLink), and Norway and England (North Sea Link). The NordLink was in trail operation from 2020-12-09 and have been a major player affecting the pricing within the NO2 market since. The North Sea Link was in operation 2021-10-01. Figure 2 is a plot of the NO price from 2016-2021. In this plot we have highlighted when the two moste recent transmission cabels come into operation.

DATA

The data used for this project is collected via the Entsoe Transparnecy Platform API. [16]. We also look at

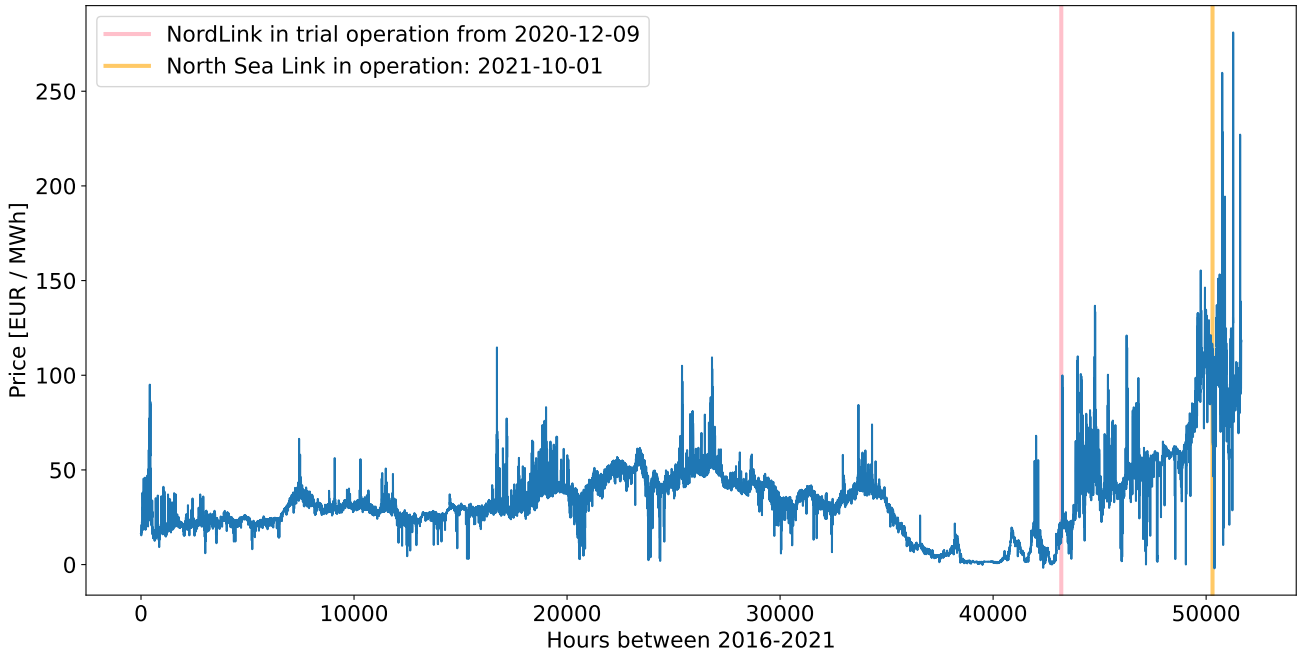

FIG. 2. Electricity price in bidding zone NO2 between 2016 and 2021. Dates are given in YYYY-MM-DD

"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 (hours) and 20 columns containing variables related to power region 2 and 5 in Norway. However, we ended up studying the NO2 zone as mentioned in the introduction. Figure 2 shows the nature of our target data and its complexity.

Feature investigation

Figure 3 shows the data and features we focused on and downloaded through the entsoe and NVE API. This include the actual day ahead price, actual and forecasted data of electricity load (demand) and electricity generation, and the aggregated "fyllingsgrad" from the dams in zone NO2. "fyllingsgrad" represents electricity from hydro power that potentially can be produced relative the the amount of water residing within the dam. The delta features were created extra as we wanted to see if forcasted minus actual value would be beneficial. None of the features indicate a significant correlation with respect to the NO2 day ahead price.

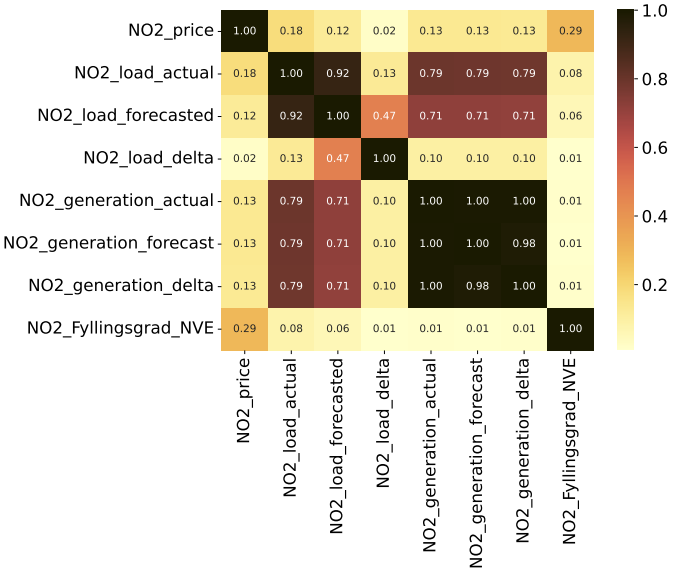

FIG. 3. Correlation between data

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 (Batch Size, Time Steps, 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 as features. Where a value of 1 would refer to a univariate dataset. This is in line with some literature, which 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:

$$x : T \to \mathbb{R}^k \tag{1}$$

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 describes how several variables changes over time. These variables are: actual power generation, generation forecast, actual load(demand), load forecast, the delta between forecast and actual and "fyllingsgrad". 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 analyzing 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].

Figures (4) and (5) display the decomposed characteristics for both the Day Ahead price from the Entsoe dataset as well as the "fyllingsgrad" from the NVE dataset. From the decomposed day ahead price plot in Figure (4), the trend clearly correlates with the currently experienced increase in electricity prices, much more so than previous years. When looking at the increase in the residual graph, there is also a clear increase in values as well as in its volatile behavior. The residual is what noise is left after the rest of the data is decomposed, which shows the complexity of the data being modeled, especially for the last year.

In comparison, the "fyllingsgrad" plot shows a clear periodicity in all the decomposed characteristics except the residuals, which follows the physical constraints of the data. Moreover, the residuals differ about 0, such that their contribution is dominated by the clear seasonal component and trend. Also expected due to the mechanics of hydroelectric power production. However, comparing both the decomposed Figures, the increased trend in Figure (4) follows from the decreasing trend in Figure (5).

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

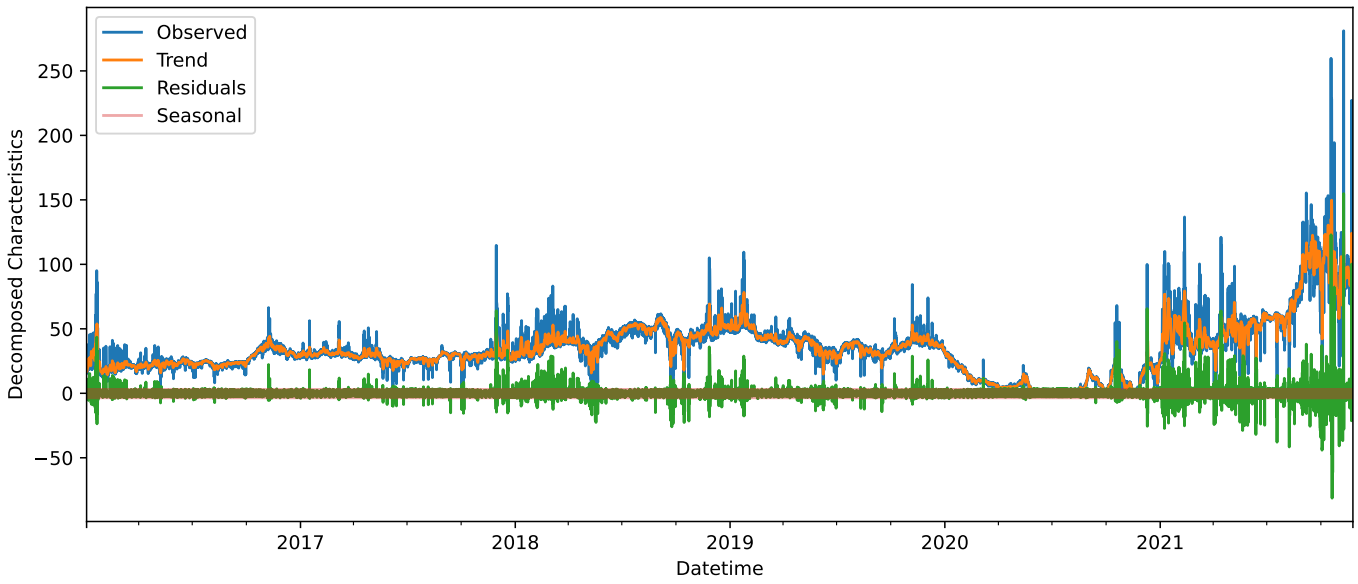

FIG. 4. Decomposed characteristics of day ahead price NO2

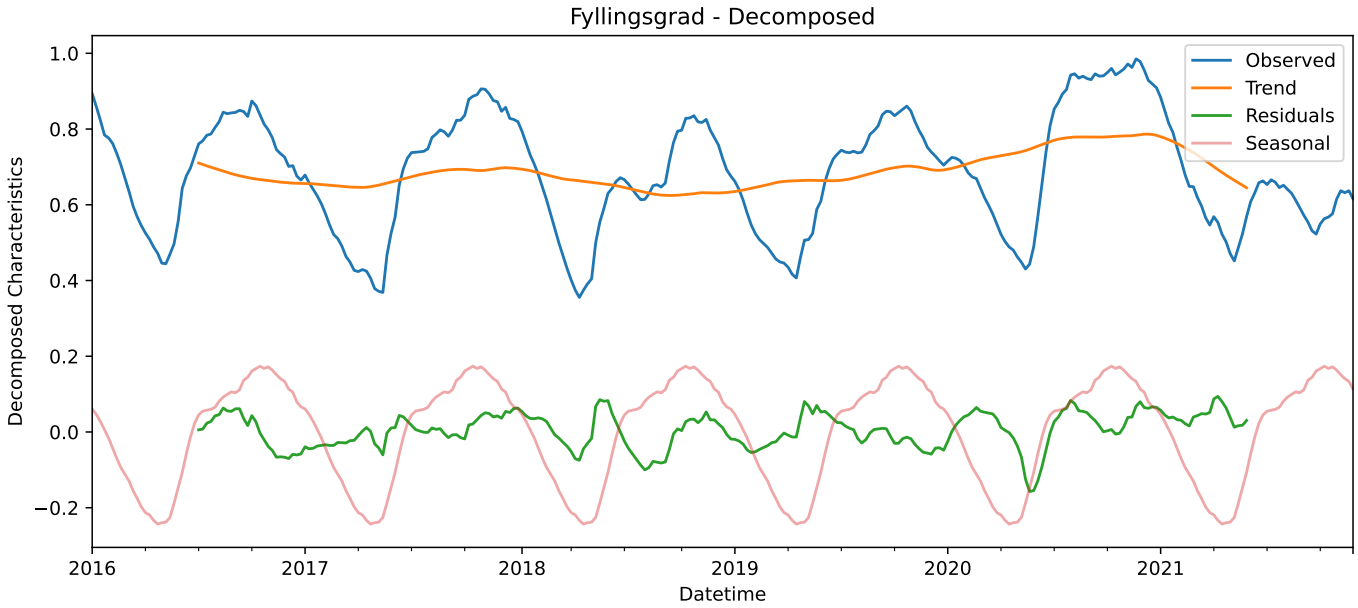

FIG. 5. Decomposed characteristics of NVE "fyllingsgrad"

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]. We can then proceed 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

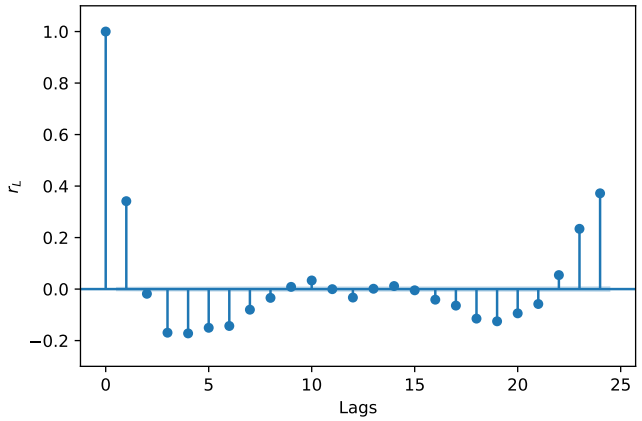

FIG. 6. Auto correlation of day ahead price

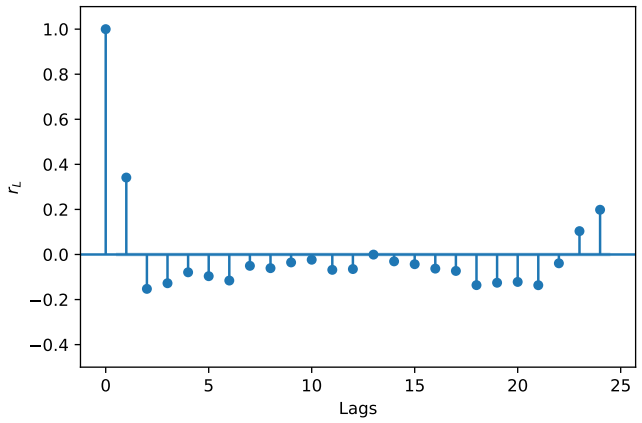

FIG. 7. Partial auto correlation of day ahead price

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.

Figure (6) shows that the close neighboring hours are highly auto correlated, i.e. a given hour could be replaced by its closest residing observation. Thus, we may assume that the day-ahead price observations are influenced by neighboring hours, which is a condition in which a Autoregressive model may benefit from. However, the auto correlation decreasingly persists up until the 12th, where there seem to be little to no significant correlation. Albeit the closest hours do influence each other, the day-ahead price 12 hours later is not significantly correlated. Moreover, regardless of hour chosen, the day-ahead price 24 hours later can be assumed correlated. In practice, this relates to how the day-ahead price tend to somewhat repeat itself each day. The same observations can be made by studying Figure (7).

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:

$$X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{2}$$

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 linked in the Appendix under code/AR_model.ipynb.

To estimate the coefficient related to the lag for each time step, we use the Yule-Walker method which consists of the set of equations

$$\gamma_m = \sum_{k-1}^{p} \phi_k \gamma_{m-k} + \sigma_\epsilon^2 \delta_{m,0} \tag{3}$$

where γ_m is the autocovariance function of X_t, m is the number of lags from 0 to p, σ_ϵ^2 is the standard deviation of the Errors and $\delta_{m,0}$ is the Kronecker delta function [11]. Furthermore, we can rewrite this equation as

$$\gamma = \Gamma^{-1}\Phi$$

where $\gamma \in \mathbb{R}^p$, $\Gamma \in \mathbb{R}^{pxp}$ and $\Phi \in \mathbb{R}^p$. which we can solve by applying the Ordinary Least Squares method. By optimizing the equation for Φ using Ordinary Least Squares, we obtain the expression

$$\hat{\Phi} = \left(\Gamma^T\Gamma\right)^{-1}\Gamma^T\gamma$$

Where we can use the obtained estimates for our ϕ's to find the optimal solution for the σ_ϵ^2 in Equation (3), which enables us to fit the AR model.

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 Equation (2) 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 the Yule-Walker algorithm as described above from the Statsmodel module [19].

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]

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^T \nabla_{\boldsymbol{o}^{(\tau)}} L \tag{4}$$

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

$$\boldsymbol{W}^T J(\boldsymbol{a})(\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^T(\nabla_{\boldsymbol{o}^{(t)}} L) \tag{5}$$

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 in-

formation 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.

Iterative Forecasting

Though the main objective of this Project have been to utilize a Sequence to Sequence Recurrent Neural Network to simultaneously predict day-ahead price 24 hours into the future, we also wanted to study another approach to forecast for several time-steps ahead. As such, we have studied iterative forecasting as an alternative way to forecast day-ahead price for the following day. The main idea behind the iterative scheme is to predict one time step into future, then reusing the just made prediction by including it in the test dataset sample to iteratively predict forwards in time [9].

However, as iterative forecasting depend on an ever increasing amount of forecasted values to make predictions, earlier predictions are likely to attain higher accuracy than later predictions. This is due to how later predictions base themselves on previous predictions, inheriting both the errors in the data as well as the prediction error [9].

Moreover, as the iterative approach only requires one target value to make an prediction, it is possible to implement a sliding window algorithm which structures the data such that it can be fed through the Feed Forward Neural Network which we developed for Project 2.

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

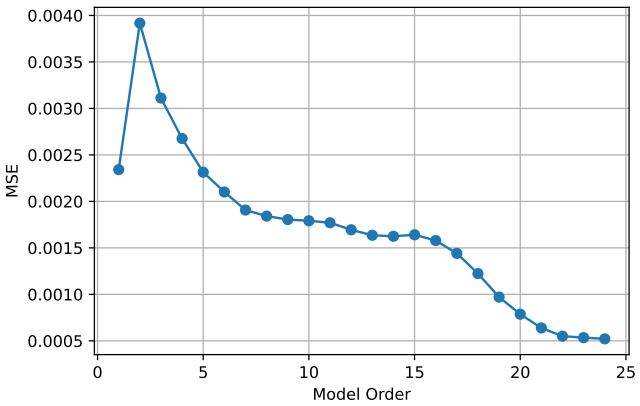

FIG. 8. AR grid search on order p

TABLE II. Performance metrics computed for a 24th order AR model

	Score
R2	0.95913
MSE	0.00052
RMSE	0.02284

readable for those with visual deficiency [6]. Moreover, some results have been moved by latex to the bottom of the paper.

Auto Regressive Model

Figure (8) show a Grid Search where the MSE is shown as a function of model order. Moreover, Figure (9) displays the actual observed day ahead price with the forecasted values made individually for all 24 hour intervals from the Autoregressive model plotted above. Also included in Figure (9) are visualization of the establishment of two interconnecting cables to the European power market.

Table (II) presents the R2, MSE and RMSE score metrics for a Autoregressive model of order 24.

Recurrent Neural Networks

Behaviour of uni -and multivariate models

These results are part of a 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 fyllingsgrad data. All results for the current subsection are made using a Sequence to Sequence model.

Figure (10) shows the Mean Squared Error as a function of number of Neurons for a single layered, univariate model. Whereas the lowermost plot in Figure (10) shows the Mean Squared error for the same models, but this time using a multivariate input of price, forecast load and forecast generation.

Figures (11) 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. The uppermost Figure is for the SimpleRNN cell whereas the lowermost Figure 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 (12) displays a Grid Search performed over different architectures where one or two Dense Feed Forward layers preceded the GRU layers in the RNN.

Figure (13) shows a Grid Search performed over different output space dimensionality and kernel sizes, with a fixed stride of 3.

Finally, Figure (14) presents a Grid Search performed over different combinations of dropout and recurrent dropout.

Optimizing the models

Figures (15 - 20) and (21 - 26) shows the result of a Grid Search over each of the different Recurrent Neural Network cells, for both a univariate and multivariate case with using architecture following the remarks made in the coming discussion. Figures (15 - 20) display the training and validation loss as function of epochs. While Figures (21 - 26) show hourly averaged values covering the difference between the true target and predicted value, mean MSE and variance.

Iterative predicitions

The following Figures (27) and (28) presents the result of a 24 hour iterative prediction made with an LSTM - Recurrent Neural Network as well as our own Feed Forward Neural Network. The predictions are made for the last sample in the training data. The LSTM has a deep architecture with two layers, both containing 25 neurons, whereas our own Neural Network is initialized

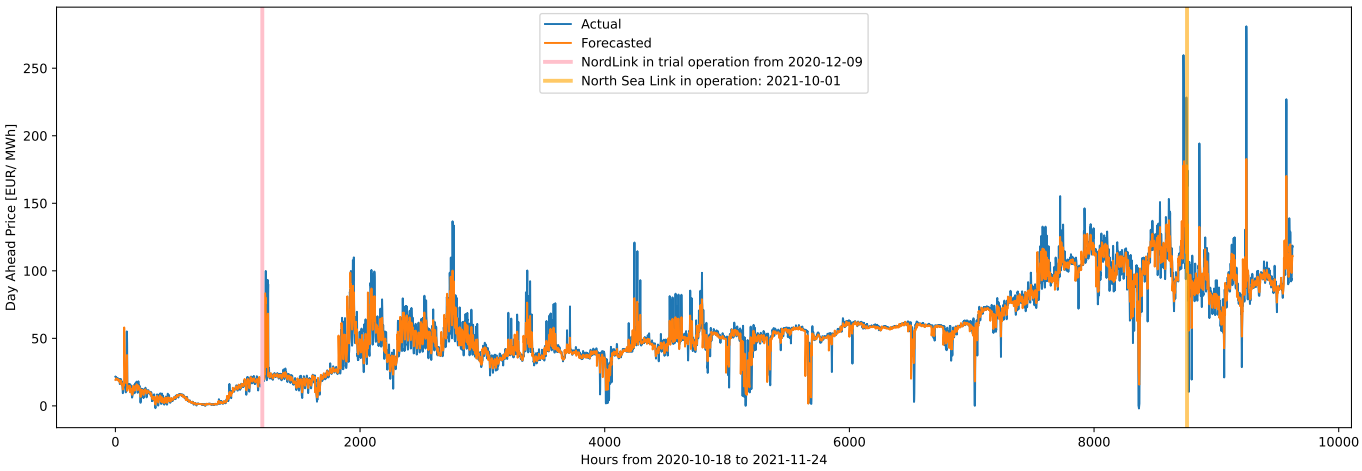

FIG. 9. AR-model(24), 24 hour forecast)

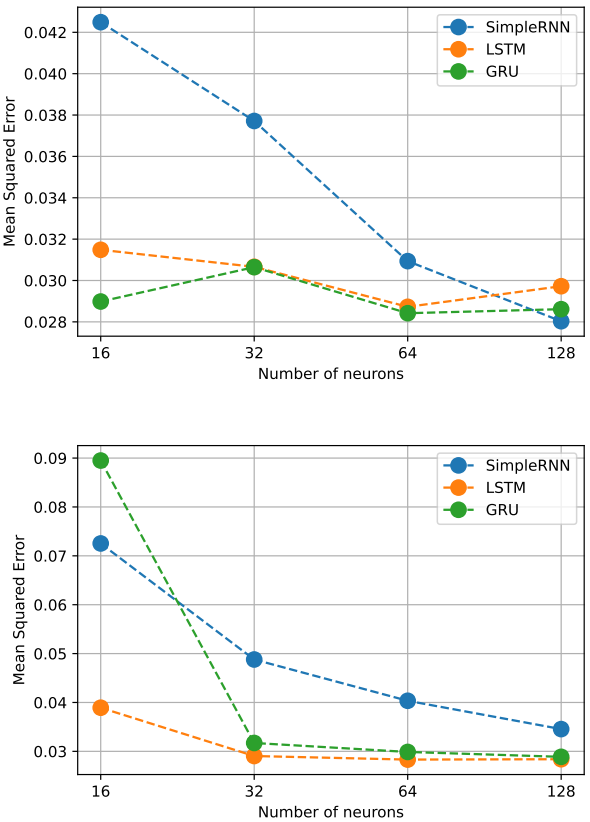

FIG. 10. Mean Squared Error as function of number of neurons for a single layer model. The uppermost figure shows a univariate case, while the plot at the bottom shows for a multivariate case

with a single hidden layer containing 64 neurons, as well as the leaky ReLU activation function.

Tables (III) and (IV) show the attained MSE value following the predictions made in Figures (27) and (28).

TABLE III. Performance metrics Iterative LSTM - one single sample forecast

	Score
MSE	0.002

TABLE IV. Performance metrics Iterative Own Neural Net project2 - one single sample forecast

	Score
MSE	0.00829

DISCUSSION

The day ahead pricing is much based on other forecasting models. When the power-producing operators contribute to the bidding process, they want to sell their power and buy from other participants. Therefore, buyers and sellers heavily rely on different market estimates to play in their respective bidding zone. Therefore, we expected the forecasted features for load and generation to contribute significantly when forecasting the pricing. However, the load and generation forecast did not show a high level of correlation in figure 3 with respect to price. Additionally, load and generation forecast did neither contribute significantly to the performance of our Recurrent Neural Networks.

The Autoregressive model

Grid Search over model order

Figure (8) shows that the first order attains a lower MSE than the second order model. This result can be related to Figure (6) which we previously described while presenting and analyzing our data. We noted based on Figure (6) that the neighboring hour was highly correlated to the

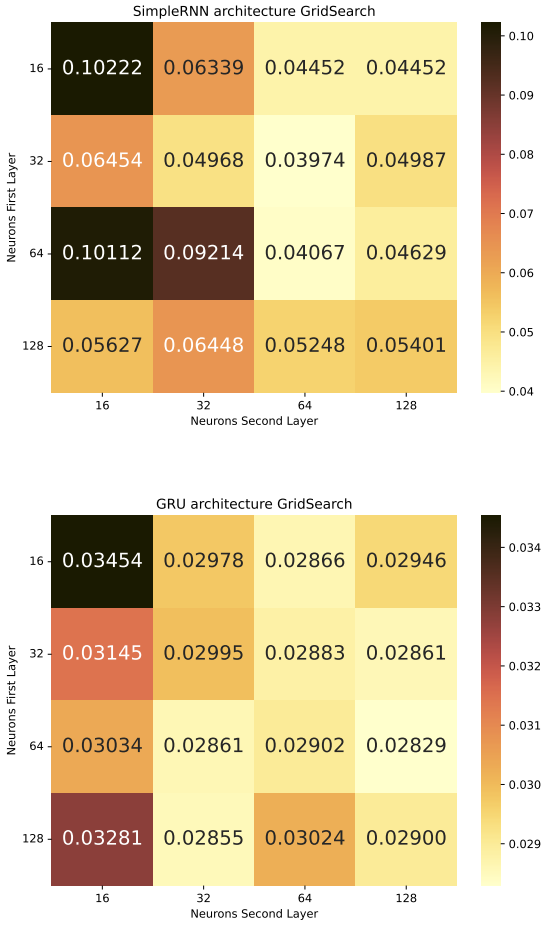

FIG. 11. Grid Search over different combination of layer size for a multilayered multivariate SimpleRNN and GRU network using the forecasted data

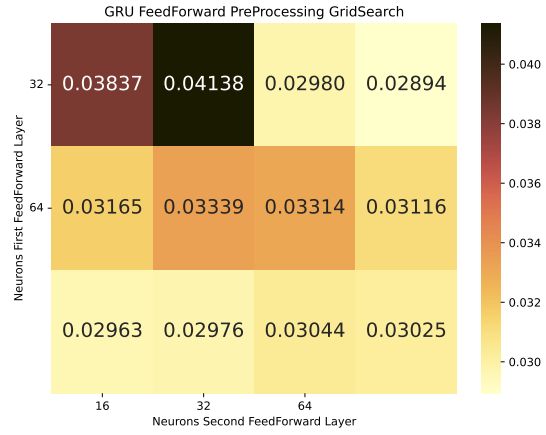

FIG. 12. Grid search over different architectures utilizing Dense layers preceding the GRU layers

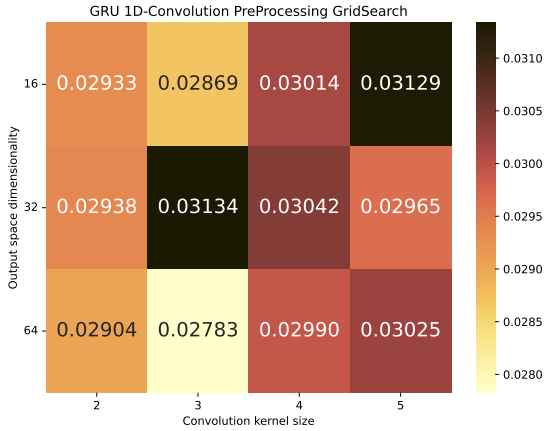

FIG. 13. Grid Search over different architectures utilizing a 1 Dimensional convolutional layer preceding the GRU layers

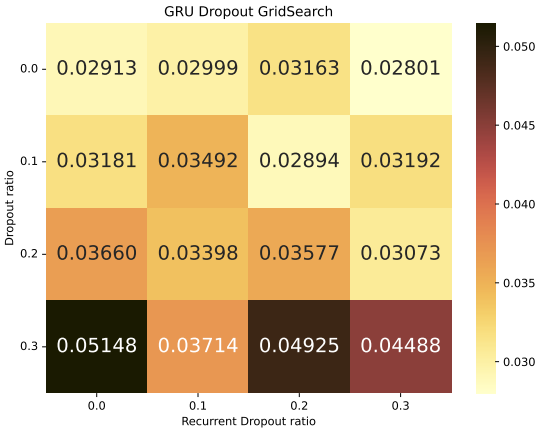

FIG. 14. Grid Search over different ratios of dropout and recurrent dropout.

current hour, more so than the hour following afterwards. Reflected in Figure (8) is thus the high correlation between neighboring hours, and the subsequent drop in correlation for the following hour. However, though the second hour attains almost a correlation value of 0, the following hours regain their correlation value, which is reflected in the AR model by further lowering the MSE value starting at its third order.

The flat area observed in the middle of Figure (8) can be seen in conjunction with the lag interval (8 - 15) in Figure (6), where the autocorrelation show that there are close to none correlation in the specified interval. As such, this implies that including these lagged values in the model only results in added noise, which has no direct improvements on the MSE. However, the correlation values beyond the specified interval regains their significance, thereby furthering the MSE reduction.

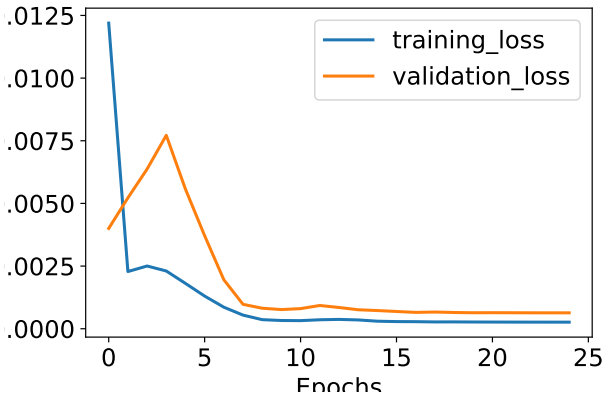

FIG. 15. Univariate train evaluation RNN

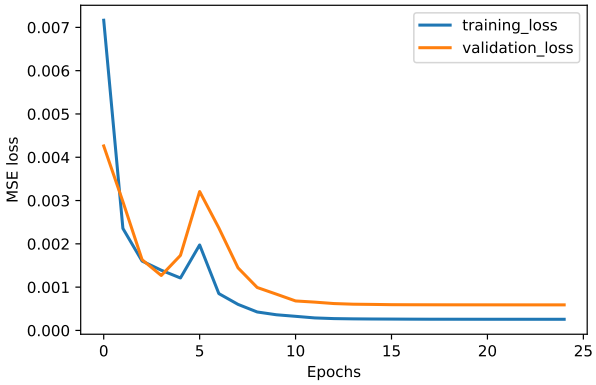

FIG. 16. Univariate train evaluation conv GRU

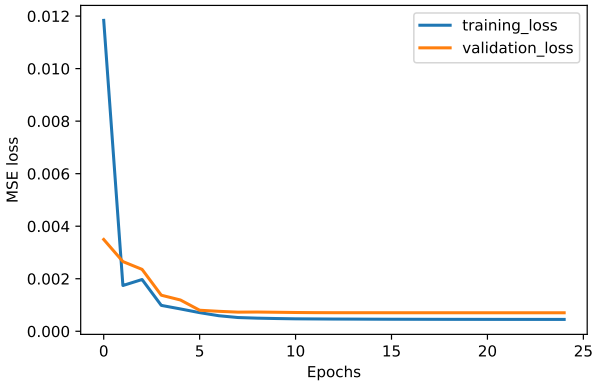

FIG. 17. Univariate train evaluation LSTM

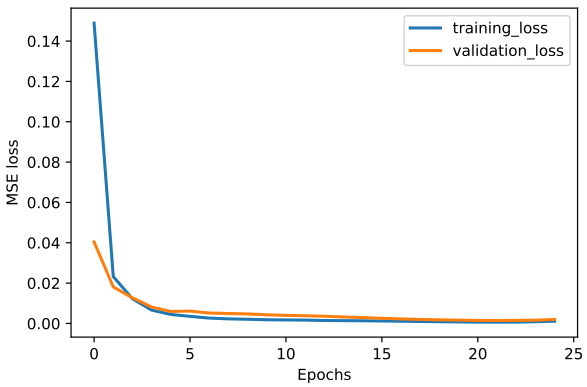

FIG. 18. multivariate train evaluation RNN

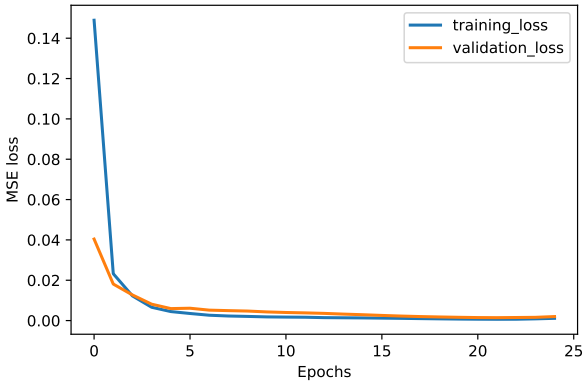

FIG. 19. multivariate train evaluation conv GRU

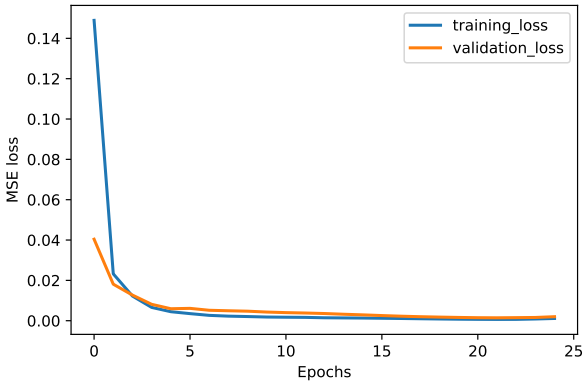

FIG. 20. multivariate train evaluation LSTM

Figure (9) shows the result of all predictions made by the Autoregressive model combined into one plot over the entire test interval corresponding to 10% of our dataset. We see that the autoregressive model follows the general structure of the observations, though it struggles to properly forecast extreme price events. This may be a consequence of the nature of Ordinary Least Squares. Since Ordinary Least Squares tries to minimize with respect to the Mean Squared Error, the resulting coefficients will give results close to the mean. This will naturally lead to the ϕ coefficients not being optimized for predicting outliers attaining extreme values from the mean.

Finally, by inspecting Table (II), we note that the MSE value from all predictions is $= 0.00052$, which is highly better than any of the resulting MSE values obtained by the Recurrent Neural Networks which will be studied and

discussed in greater detail in the following section. We note however that though the MSE itself seem satisfactory as a forecasting skill score, there is a weakness in not being able to predict somewhat unexpected values. This may be due to how the OLS in general optimizes based on similar values, and that freak events are far and few between. Especially since the test data itself can be considered a freak event in terms of the day-ahead price, as presented in Figure (4).

Recurrent Neural Networks

Response to architectural changes

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 in conjunction with the Machine Learning library TensorFlow. In this section, we will limit ourselves to the study of our Sequence to Sequence 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]. Furthermore, for the multivariate case, we will only study the model behavior with the forecasted values in the dataset.

By inspecting Figure (10), 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 for the SimpleRNN, 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. However, it can be argued that both the GRU and LSTM models respond somewhat less uniform to the change in model complexity, especially for the univariate case. 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 for the univariate case. This may be reasonable as this is for a univariate model, where the amount of data is sufficient for an SimpleRNN cell to fit the data appropriately, whereas the more complex cell structures of LSTM and GRU internally may overcomplicate the data.

Furthermore, by moving downwards in Figure (10), some similarities may be drawn between the overall behavior. However, both the SimpleRNN and GRU models are initially having a harder time fitting to the data than the LSTM cell model. As such, by introducing two 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 cell network, where it seems that the 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 the univariate data for already at 32 neurons. Thus, the two plots in Figure (10) 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.

Figure (11) 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 the uppermost Figure in (11), it can be seen that the MSE responds somewhat similarly to the multi-layered model as for the single layered SimpleRNN model. Namely that the MSE attains lower values as the complexity of both layers are increased, though that the MSE is relatively high for less complex architectures. This trend is similar for the lowermost Figure in (11), which displays the same Grid Search but for a GRU model instead. It can argued that the lowest MSE is attained with the most complex model, as there seem to be a relationship between MSE value and second layer complexity, with higher second layer complexity being better. However, the best result is attained for 128 and 32 neurons as seen in the Figure, though this may be a stochastic result which arises from the weight initialization of the modes. We note that the results are not entirely consistent between model compilations (not shown), though the relationship between the MSE and model complexity as shown in Figures (10) and (11) is consistent between model runs.

Effect of data alteration for long sequences

It has been noted in previous sections that there are two central drawbacks for Recurrent Neural Networks, namely the gradient problem and the short-term memory problem, especially for long sequences of data [9]. As a measure to decrease the effects of these limitations, we have tried several preprocessing layers as well as studied the effect of Dropout. For preprocessing layers, we have studied the effect of preprocessing the data using one or two Dense FeedForward layers before the Recurrent Network layers, with the intent that the FeedForward layers may project our data into a more suitable feature space before being fed to the Recurrent Neural Network. The result of preprocessing the data with one or two FeedForward Dense layers can be seen in Figure (12). For clarity, all results in this section has been produced using a Sequence-to-Sequence multivariate GRU following the optimal architecture as argued for by combining the results of Figure (10) and (11), i.e. two Recurrent layers both with 128 neurons each.

By inspecting Figure (12), there seem to be no clear pattern in how the MSE is affected by different FeedForward architectures. Though some architectural combinations do achieve a somewhat equal MSE to the model with no preprocessing layers at all, the results do seem to be consistently worse across all combinations of Dense layer architecture.

Figure (13) shows a Grid Search approach which studies the implementing of a 1-dimensional convolutional layers which precedes the two Recurrent layers. The 1-dimensional layer downsamples the input sequence, as well as preserving the notable structures in the data through convolution [9]. This process helps the coming Recurrent layers to detect larger patterns in the data, by making the sequences to detect shorter. As can be seen in Figure (13), the effect of the 1-dimensional convolutional layer seem to be somewhat tied to the size of the convolutional kernel, more so than the number of output features of the layer. Though the distribution shown in Figure (13) is relatively stochastic. With a kernel size of three, the model attains lower MSE values than what was reported in figure (11). As such, a 168-hour input width may prove longer than what the short-term memory of a Recurrent Neural Network is able to retain, as the MSE attains a better score with downsampled data for some combinations of output space dimensionality and kernel size. However, other combinations do deter the overall MSE score, such that there is no clear benefit of applying the convolutional layer unless the parameters are tuned to fit the current data.

Finally, the effect of Dropout and Recurrent Dropout can be seen in Figure (14). The central idea behind dropout is to generalize the training data by deactivating a ratio of the current neurons, i.e. setting their internal value to 0 essentially stopping all incoming signals. By inspecting Figure (14), a single MSE value, that being the rightmost uppermost corner value related to Dropout = 0 and Recurrent Dropout = 0.3 do achieve a lower MSE value than what was reported in Figure (11). However, the trend is that Dropout heightens the MSE value, with regular Dropout having a more severe negative impact to the MSE than Recurrent Dropout. This may be due to the training sample used being relative sparse, only counting a total of $\tilde{1}500$ individual samples. As such, the dropout removes important information from the dataset, which may not be covered elsewhere, instead of generalizing the dataset.

Univariate and Multivariate performance

The figures (15 - 20) shows the training evaluation performance of univariate and multivariate models. For the univariate models the RNN and GRU had similar pattern with the validation loss spiking at epoch 4-5. We have implemented a learning rate scheduler that reduces the learning rate on plateau which helps in correcting for the overfit. Furthermore, univariate LSTM, multivariate RNN, multivariate GRU, and multivariate LSTM had similar training performance, and they all converge at an early epoch. None of the aforementioned training evaluation plots shows signs of overfit.

The figures (21 - 26) shows how the model models perform relative to each specific hour. The blue line visualizes the MSE specific to each hour (MSE of the forecast for an hour n against the target for an hour n). The black line indicates the true mean for the target and the true variance of the target for each hour. The red lines show the corresponding mean and variance of the forecasts from each model. We see that the mean value reflects the load or demand signature of a daily household. We start our predictions at hour 13:00. This means that 0 on the x-axis corresponds to 13:00. At x-axis equal to 3 corresponds to 16:00. At this time, people start to come home from work and use more electricity. When many consumers consume electricity simultaneously, the electricity price increases, thus following an increase in the mean value. For the performance of the univariate models, we observe from the mean value of the forecasted hour (red lines) that the models cannot fit closely to the true mean value of the price for each specific hour. We see a small variation in the prediction over all the 24 hours. Looking at the variance of the forecasted hours, we observe a somewhat similar pattern as for the mean. The models cannot learn the variance specific for each hour considering all samples. Looking at the blue line representing MSE, the values show a significant increase in residual error following the mean price signature hours. The models have larger residual error between 16:00-20:00 and between 05:00 and 09:00. Looking at the multivariate performance, they share similar performance as for the univariate.

Iterative predictions

Figures (27) and (28) show how an LSTM network and our own Neural Network iteratively predicts 24 hours into the future. As noted in the introduction, Feed Forward Neural Networks are not optimized for fitting time-series data, as it relates a specific point in a time-series to a specific target. However, as seen in Figure (28), our own Neural Network is able to somewhat predict the shape of the actual observations. Though there is a bias towards greater values, especially when comparing to the LSTM forecast.

Moreover, though our Neural Network seem to be able to predict that the price will rise two times as seen in the target observation, it is not able to accurately predict the initial dip in value. Moreover, the prediction seem to stay high towards the end of forecast where sum of errors by previous forecasts may further increase the predicted values. This also is a result of the positive bias of the outputs from our Neural Network prediction.

In comparison, the LSTM network predicts the observations to a higher approximation. This can be seen by comparing the MSE values in Table (III) and (IV). Interestingly, the MSE obtained by both the Neural Network as well as the LSTM network is lower than the MSE values obtained while studying the Sequence to Sequence Recurrent Neural Network, though we note that the just presented MSE values are computed from a single sample, not the entire test data.

CONCLUSIONS

Throughout this project we have studied different approaches to accurately forecast day-ahead electricity price based primarily on features from the Entsoe dataset. We did this by implementing a Autoregressive model which is based on a Ordinary Least Squares approach, a Sequence to Sequence Recurrent Neural Network as well as structuring the data such that our Feed Forward Neural Network developed in Project 2 could attempt at an iterative prediction scheme. For the development of the Recurrent Neural Network, we utilized the Machine Learning library TensorFlow.

While exploring our data, we noted that there is an apparent relationship between the trends of the day-ahead price and the NVE data. Thus, part of the increased electricity price currently experienced in Norway, may be due to the shallow levels of water currently in the reservoirs. Additionally, though our data did not include specific information on inter-market trade and electricity export, it is of our belief that some of the missing explainability on the increased day ahead price can be found by studying the European power import/export market. Which Norway newly have established interconnecting cables to.

When Studying the Recurrent Neural Network, we highlighted some of its inherent problems, namely the gradient problem and the short-term memory problem. We started off by opting to utilize a Sequence to Sequence network, such that we could maximize the gradient flow through the network. To further study how the Recurrent Neural Network behaved, grid searches over different architectural pattern as well as different preprocessing and data generalizations techniques were performed. It was seen that the Recurrent Neural Network responded favorably to an increased architectural complexity, though the different pre processing schemes tended to significantly increase or add noise to the MSE. However, we noticed that some combinations of output dimensionality and kernel size decreased the MSE for a network with the GRU cell.

However, the Sequence to Sequence Recurrent neural network tended to just predict a mean value for all time steps, as discussed previously when inspecting Figure (22). Though this resulted in the lowest MSE value, forecasting a daily average would not prove sufficient for an end user. We suspect that the predictions made would better predict the target data if more features where included. For example, a similar approach conducted by Wei Li et.al [12] included $+tilde50$ features, which is far more compared to the three features we included.

In comparison, the Autoregressive model, which is based on the Ordinary Least Squares algorithm, tended to accurately predict the day-ahead price. Attaining a far lower MSE value than the Recurrent Neural Networks. This may be due to the day-ahead price being a auto correlated sequence, which the Autoregressive model benefits from. As such, we report that given a feature sparse situation, day-ahead price itself as a univariate time series is not enough to accurately predict using a Recurrent Neural Network. The RNN is itself too complex a model, such that the information residing in the day-ahead time series is better extracted with a more classical statistical model such as the Autoregressive model.

We also briefly studied iterative predicting, where we structured our data in such a way that we could feed it to our Feed Forward Neural Network. Though our Neural Network itself is not an appropriate model to study time series, the prediction reached a reasonable MSE when comparing to the benchmark LSTM predictions. Though we for this project mostly studied Sequence to Sequence RNNs, the iterative approach seemed to give reasonable results, with the potential to outperform the Sequence to Sequence model. Sadly, due to time constraints, we were not able to further investigate iterative prediction.

In conclusion, the Autoregressive model seemed to outperform our Recurrent Neural Network when predicting day-ahead price given our limited dataset. However, due to time constraints, we were only able to properly study Sequence to Sequence RNN. As we see promise in other RNN schemes such as the iterative prediction scheme, we recommend future work to study how different RNN schemes such as Sequence to Vector or Encoder Decoder networks would perform.

Source Code

Link to github repository containing all developed code for this project: https://github.com/AndreasBordvik/FYS-STK4155-Prj3_report

[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.

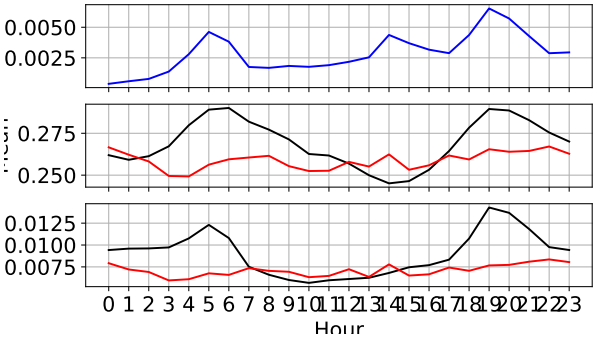

FIG. 21. Univariate Simple RNN performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

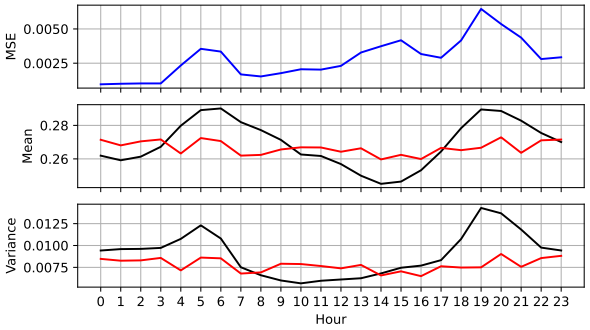

FIG. 22. Univariate ConvGRU performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

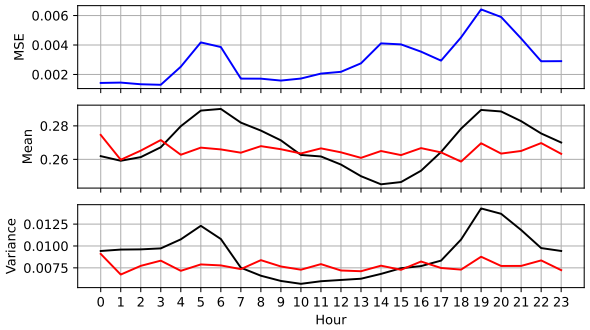

FIG. 23. Univariate LSTM performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

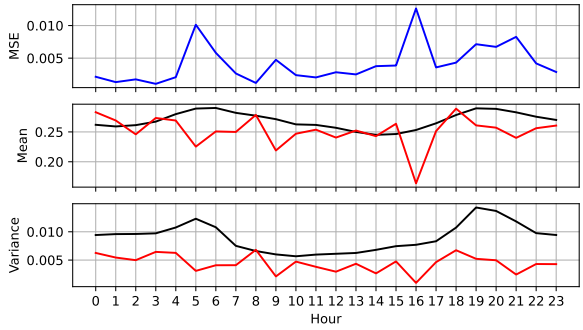

FIG. 24. Multivariate Simple RNN performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

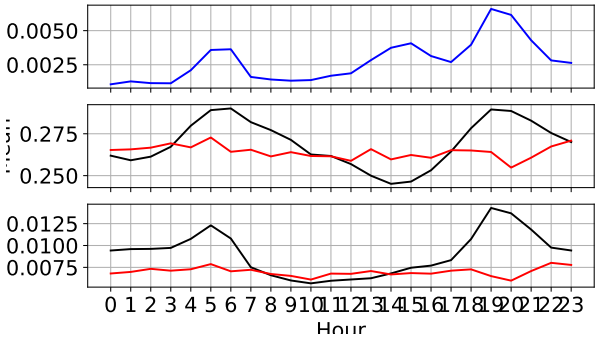

FIG. 25. Multivariate ConvGRU performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

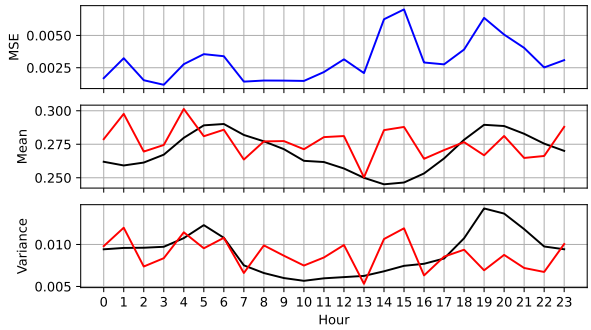

FIG. 26. Multivariate LSTM performance - Blue color indicate difference in MSE between target and predicted for specific hours. Black indicate mean and variance of target, and red indicate mean and variance of all predictions relative to its hour

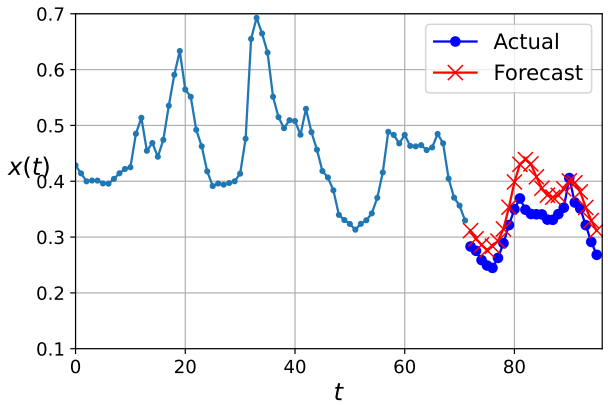

FIG. 27. Iterative LSTM approach forecasting price for the next 24 hours reusing previous forcasted values

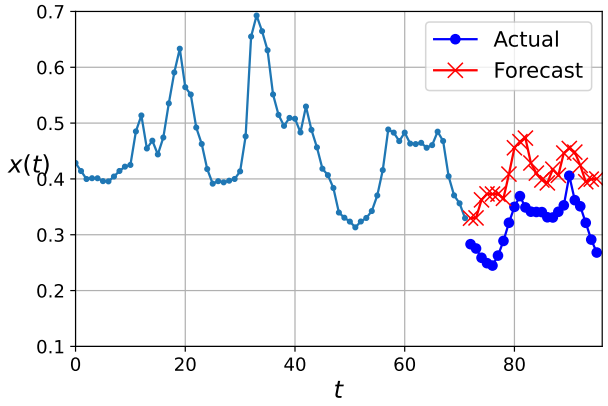

FIG. 28. Iterative Own developed Neural Net from project 2 - forecasting the next 24 hours reusing previous forcasted values