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 15, 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. [**?**]. Besides the aforementioned, studying the electricity price from a consumer standpoint considering the recent experienced extreme electricity price 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 **??**. 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 [**?**]. 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 [**?** **?** **?**]. Due to time constraints, this project only focuses on forecasting the electricity price for the bidding zone named 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 [**?**] 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 foreward its current inputs. Therefore, this project will make us of Recurrent Neural Network and state-of-the-art methods for time-series data. A Recurrent Neural Network shares its

* 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/

weights across several time steps, which makes it suitble to use for temporal data [**?**].

DATA

The data used for this project is collected via the Entsoe Transparnecy Platform API. [**?**]. 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) [**?**]. 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 (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, such that dimensionality = 1 would refer to a univariate time series. Some literature refers to the last dimension as the number of features.

Sliding window

test k

t	p		t	p		t	p		t	p	
t_0	p_0		t_0	p_0		t_0	p_0		t_0	p_0	
t_1	p_1	X_0	t_1	p_1		t_1	p_1		t_0	p_1	
t_2	p_2		t_2	p_2	X_1	t_2	p_2		...	...	
t_3	p_3		t_3	p_3		t_3	p_3	X_2	...	...	
t_4	p_4	H_0	t_4	p_4		t_4	p_4		$n-4$	p_{n-4}	
t_5	p_5		t_5	p_5	H_1	t_5	p_5		$n-3$	p_{n-3}	
t_6	p_6		t_6	p_6		t_6	p_6	H_2	$n-2$	p_{n-2}	X_n
t_7	p_7		t_7	p_7		t_7	p_7		$n-1$	p_{n-1}	
...	...		...	...		...	...		n	p_n	
...	...		...	...		...	...		$n+1$	p_{n+1}	H_n
t_n	p_n		t_n	p_n		t_n	p_n		$n+2$	p_{n+2}	

Data Windowing

We have applied a window based algorithm to structure the data into appropriate sample sizes that fit the constraints imposed by the Recurrent Neural Network. In comparison to the previous projects, our data does not contain a relationship between some input data and some target variable, 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, in our case we have an hourly time series.

Our window algorithm starts off by sectioning the data into 24-hour chunks. 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 CLASSICAL METHODS

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[**?**]. 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 $\in \mathbb{N}$ [**?**]

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, delta between forecast and actual, and "fyllingsgrad" changes over time. Dealing with data on such a format might yield an 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 [**?**]

Decomposing the Characteristics

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

This will help us in visualizing 3 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$_d ecompose. from the Statsmodels module. This method estima$

A trend is the general direction in which something is changing [**?**]. 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 [**?**]

Another component which can be highly descriptive for visualizing the time series is the seasonal component. This component is the .....

Stationarity and Hypotesity test

Autocorrelation

Auto regressive method

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 [**?**]. 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 [**?**]. 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 [**?**]

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

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 [**?**].

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 [**?**]. 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 [**?**].

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 [**?**]. By following the approach of [**?**], 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 [**?**]. 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 [**?**]. 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 [**?**], GRU as well as other gated RNNs derived from the LSTM cell performs approximately the same as LSTM [**?**]. 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

[**?**]. 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. [**?**]

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 [**?**]. 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 [**?**].

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

DISCUSSION

CONCLUSIONS

Source Code

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