Modeling and Forecasting the Nord Pool Day-Ahead Power Market through Deep-Learning

MAGNUS J. WALKER LYSFJORD

SUPERVISOR

Associate Professor Morten Goodwin
Associate Professor Lei Jiao

University of Agder, [2017]
Fakultet for Teknologi og Realfag
Institutt for Ingeniørvitenskap

Modeling and Forecasting the Nord Pool Day-Ahead Power Market through Deep-Learning

Magnus Lysfjord

Supervisors: Dr. Morten Goodwin, Dr. Lei Jiao

University of Agder

A research project submitted for the degree of

Masters of Renewable Energy

4/27/2017

Acknowledgements

I would like to thank Associate Professor Goodwin, Associate Professor Jiao, Professor Leal, MatRIC (Centre for Research Innovation and Coordination of Mathematics Teaching), Sci-Code Research Team, and Associate Professor Hogstad for their inspiration and dedication to pursue this work.

Abstract

One of the great challenges faced in the Nord Pool Market by companies and traders is the frequency of behaviour modification due to government policy such as Carbon tax, demand changes, supply changes, as well as changes faced in marginal cost. These behavioural changes have been colloquially described as price spikes or market regimes and are noted as the overall key influential factor in forecasting accuracy. This thesis undergoes a hidden markov model to categorize the hidden states within the most influential stochastic datasets provided by Nord Pool and utilizes this output as input to train the LSTM algorithm. It was found that the methodology in implementing the HMM did not allow for the LSTM to recognize the differing volatile regimes; however, a significant accuracy of 0.0082 mean absolute error was found and bench marked to other results.

Recurrent neural networks (RNNs) are suitable for time series phenomena because of their effective dynamic relationship in utilizing changing temporal information. The long-short term memory (LSTM) algorithm arose to solve the RNNs fundamental problem of vanishing gradient problem. The LSTM is thus utilized as the main algorithm in learning the Western European's largest Market for Electricity, Nord Pool.

A key issue in previous deep learning studies has been the identification of key features that are explanatory for the relationship of Nord Pool's notorious spike regimes.

Contents

List of Figures

Chapter 1

Introduction

The introduction to the trading platform, Nord Pool, that founded a competitive energy market which deregulated monopolies in Scandinavia to increase market efficiency began in the 1990s. It has drastically shaped the role of energy in today's market and has expanded into Western and Eastern Europe. Electricity is a unique commodity as it is physically non-storable while the power system must stabilize the production and consumption balance - leading to price instability especially when supply of hydropower is currently low [7]. Furthermore, electricity depends on many explanatory variables such as weather, intensity of business, daily hours, weekdays, weekends, month, year, holiday, marginal costs of supply, etc. In terms of machine learning in finding patterns in data, the advantage of such a time series is the cyclic behaviour of consumption due to the regulated times of when the consumer requires electricity [7]. The disadvantage is the non-linearity of the data reportedly due to for example, the shifts of the energy supplied, creating instances of dramatic price changes. This has led researches to intensify on their efforts of forecasting over the past decade.

The importance of price forecasting is significant on the corporate level due, as this is a primary input to the major decision making of energy companies throughout [21]. There is a serious risk faced in undercontracting and the selling or buying power in the real time market, as experienced during the California crisis of 2000 to 2001 [7]. A body such as a utility company/industrial consumer that can forecast the day ahead market, which represents 70 percent of the overall market, can drastically improve the risk and maximize profits.

A key reference in estimating the price the next day is the price of the previous day at the specific hour. Thus, dis-aggregated hourly prices representing the daily price which have been used in multivariate models have been demonstrated to signficantly outperform univariate models of just using the averaged day price (error reduction reaching 16 percent root mean square error). One would expect this due to the granularity of the data, comparing one averaged day price to hourly averaged price. Therefore, much research is now involved in finding rich multivariate models to exploit the relationship between prices and explanatory variables to said prices- forecast combination.

Chapter 2

Literature Review

The main focus on the next day electricity price as the goal criterion is due to the fact that it acts as a reference for forward, future, and the majority of derivative contracts in various electricity markets including Nord Pool Spot. As market traders can submit bids for specific individual hours of the next day, the algorithm utilized in this paper predicts the market on an hourly analysis.

2.1 Methodologies

2.1.1 AutoRegression Models

One of the most frequent methodologies used is in the domain of linear nd non-linear time-series models, such as Weron and Misiorek in 2008, Weron 2006, conducting autoregressive models. Karakatsani and Bunn in 2008 found relationships in the explanatory variables of fuel prices and demand level on price volatility.

2.1.2 Problem Statement

Price Forecasting Spikes classification utilizing the past 4 years of historical data for Hyrdopower Reservoir, Demand, and price in order to predict the next day. Different intervals of data were used, compared, and merged in a long short term memory algorithm to predict the next day.

Chapter 3

Nordic Energy Market

3.1 Introduction

Over 370 companies from 20 different countries trade on Nord Pool Spot's markets- both giant consumers and producers are involved in transactions. The Nordic region has considerable experience with deregulated electricity markets. The Nordic electricity market was formed in 1993 in conjunction with deregulation of electricity markets in the region. The derivatives and energy markets were separated in 2002 to establish Nord Pool Spot, which currently operates in Norway, Denmark, Sweden, Finland, Estonia, Latvia and Lithuania. The main goal of Nord Pool Spot is to balance the generation of electricity with the electricity demand, precisely and at an optimal price, so-called equilibrium point trading. The optimal price represents the cost of producing one kilowatt hour of power from the most expensive source needing to be employed in order to balance the system. All the employed generators are paid the same market price. Two different physical operation markets are organized in Nord Pool Spot: Elspot and Elbas. Elspot is a day-ahead energy market in which market participants submit offers to sell, or bids to buy, physical electricity for the next day. Elbas is an intra-day energy market where trades are adjusted in the day-ahead market until one hour prior to delivery time.

3.2 System Price Calculation

The elspot market is the current focus of the thesis, specifically in NO1- the Oslo region of Norway. The determination of the day-ahead system price is calculated at 12:00 pm deadline for market traders submission. The aggregation of supply and demand can then be constructed into two curves. The system price is calculated by the intersection of supply and demand representing the bids/offers of the combined

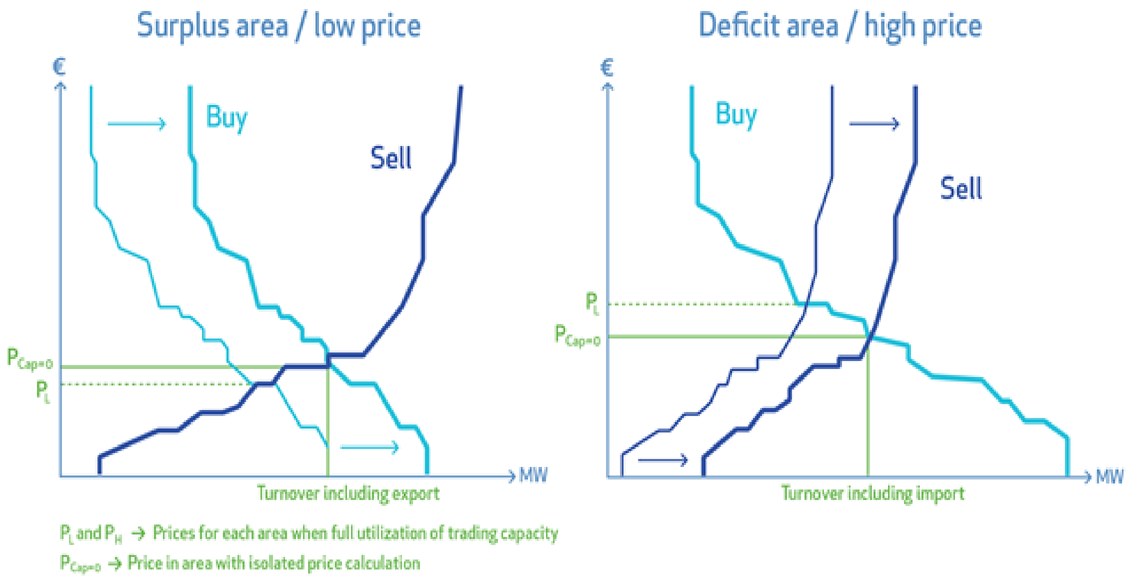


Figure 3.1: Dynamics of System Price Calculation [18]

territory. This is referred as equilibrium point trading. The following figure illustrates the dynamics of low supply or high demand necessarily affecting the system price to be determined.

The infamous volatility of the system price (which peaked in 2015 from an average price of 240kr to 2000kr) is partly a result of periods of high demand, low supply, and thus a likely switch of the merit order system. The first major supplies are provided by the cheapest sources of energy which are located at the bottom of the merit order. As depicted in the following figure, wind has the cheapest marginal cost, afterwards hydropower with the majority of supply from Norway, following is Nuclear (Sweden), later more expensive fuels are used in times of short supply which varies the price regime of the market.

3.3 Data Analysis

The datasets utilized in the following thesis are the hourly consumption of the NO1 nordic region, the volume of bids placed, as well as the historical/seasonal market price data. According to a report from NTNU, the fundamental quantities influencing the levels of electricity consumption, generation and exchange can be attributed to consumption (which is inherently dependent on temperature), and the volume of bidding (which is inherently dependent on the demand combined with resources available for hydro power producers in the Nordic region). [7]

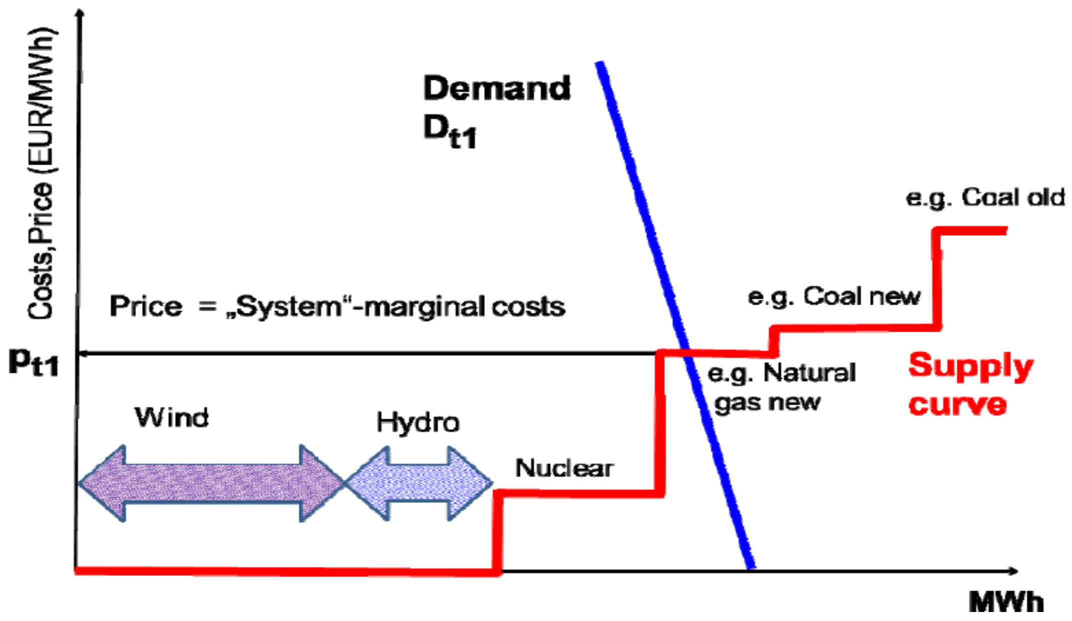


Figure 3.2: Merit Order[18]

3.4 Price and Seasonal Effects

As discussed in most of the previously reported price forecasting studies, price is largely dependent on the load or demand. On the other hand, demand itself varies depending on the season (period) of the year, day of the week and hour of the day. The hourly price can vary drastically between 240kr to 2000kr representing over 15 standard deviations in a matter of hours. As well, regular fluctuations of 200 to 400 NOK occur with mean reversion.

Season or period of the year is one of the factors that the electricity consumption depends on. During winter (the colder season of the year), for instance, there is a high demand of electricity, which in turn can cause an increase in the electricity price. In contrast, during summer (the warmer season of the year) the demand is considerably low and so does the market price. Hence, this can be considered as one of the factors that affect the system price in any electricity market. Similarly, there is a clear variation of demand for electricity between each day of the week and even between each hour of the day. Industrial loads, for instance, are usually high during weekdays while they appear to be considerably low at weekends. Residential loads, in the contrary, might be a bit higher at weekends compared to normal working days. Generally, Saturday and Sunday are days of the week having the least demand of electricity where as mostly Thursday is said to have relatively high demand. In general, this is also one factor that affects the demand or can be considered as one of

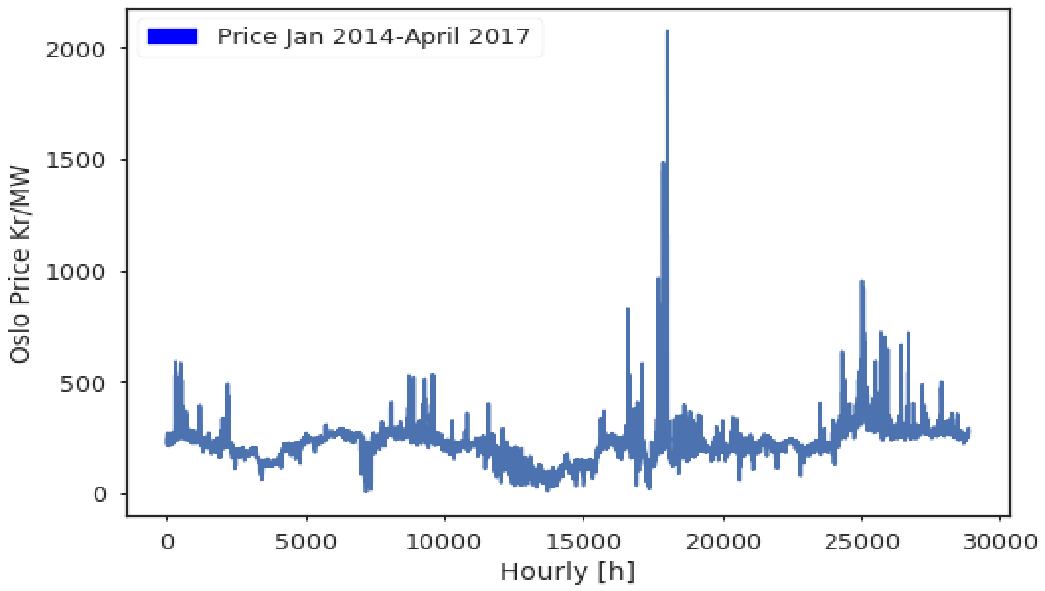


Figure 3.3: Price Over the Past 3 Years To Current

the reason for load, and hence price, variation. Likewise, demand depends greatly on the hour of the day. The hourly demand varies through out the day and so does the electricity price. In week days demand is mostly high at hours between 8:00 -11:00 in the morning and also at about 18:00 – 20:00 in the evening; however, weekends have high demand hours at around 18:00-20 in the evening. [20] Therefore, a method must be constructed to take into account both the weekly changes, daily changes, and the hourly changes in a short concise matter.The algorithm utilized will later portray the method in which this is done in a novel fashion. It should be noted that if there is enough data, yearly changes should also be taken into account [14].

3.5 HydroPower Reservoir

The reservoir level has a significant influence on the price [7], especially in the NO1 region. The volume of hydropower is dependent on rainfall or amount of water in reservoir. Increase in reservoir levels, which depends on seasonal variation, leads to a supply shift and thus reduction of price [14]. Throughout this thesis, the volume in NO1 is utilized as the indicator of hydropower reservoir supply.

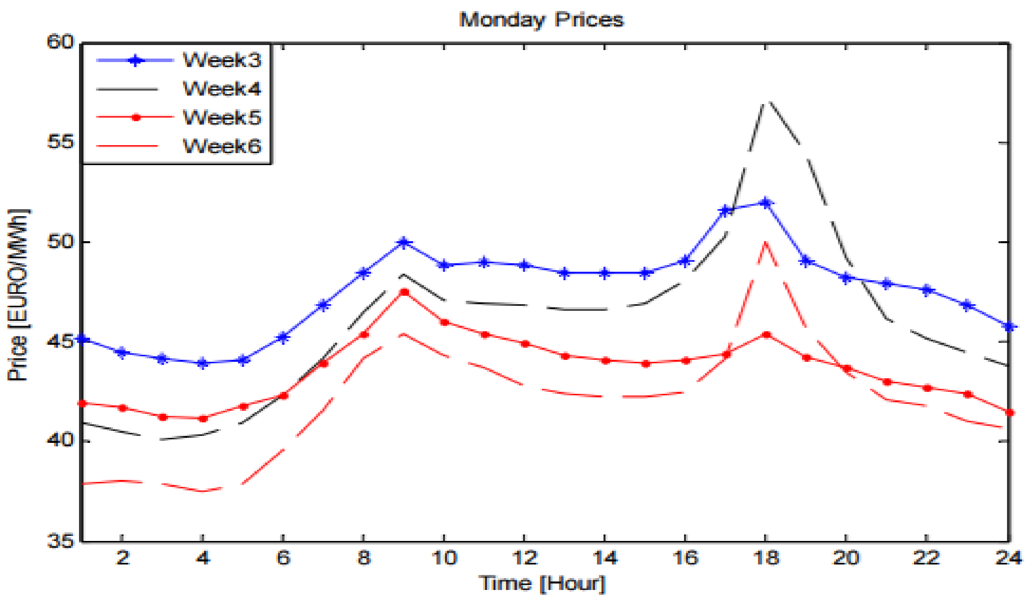


Figure 3.4: Comparison of Historical Prices given the hour [14]

3.6 Demand Usage and Volume

In Norway over 99 percent of electricity generated comes from hydropower. During periods of high demand and low shortages such as during winter, the supply for hydropower decreases as rainfall decreases due to temperature affects [3]. Thus demand is indicative of the volatility of supply. Volume is as well indicative of to what extent is consumption met. As can be seen by the dynamics of system price calculation figure during periods of over consumption or periods of too great volume/supply, the supply curve shifts. The following superposition of the volume and consumption graph over the past 3 years are similar but their minuscule differences directly affect the volatility of price as this is the dependence of the system price, i.e. the balance of supply and demand [7].

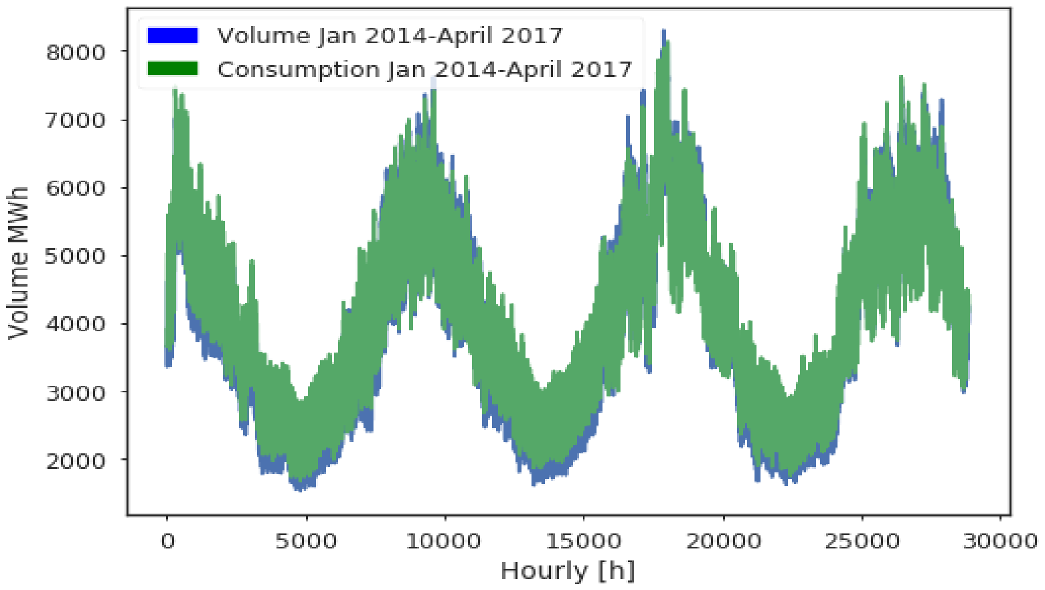


Figure 3.5: Consumption Over the Past 3 Years To Current

Chapter 4

Markov Model

4.1 Introduction

Hidden Markov Models are renown for their use in modelling time series data and have been used in speech recognition algorithms, computational molecular biology, data compression, and in general, pattern recognition [5]. To generalize, hidden markov models are a particular kind of Bayesian Network.

Hidden Markov Models (HMM) are a standard technique in time series analysis or data mining. Given a (set of) time series sample data, they are typically trained by means of a special variant of an expectation maximization (EM) algorithm[8].

4.2 What are Hidden Markov Models?

An HMM is a tool that describes the probability distributions over sequences of observations. To begin, let us describe an observation occurred denoted Yt at a time t. This observation could be a price spike, demand spike, or reservoir spike [6]. It then makes the assumption that the observation was generated by a process that has a state denoted St which is not known to the observer. As there are a multitude of reasons why a price spike can occur and the reasons are not exactly known, it can be said that the this process is hidden from the observation. The behavioural assumptions are as follows, t has equally-spaced time intervals and next, the next is the major assumption that the value of the current State, St, is independent of all states prior to t-1. The third assumption is that the outputted hidden variable is discrete. The method in which the markov model takes into account information before t-1 is thus through the transition probability of the state embedded in just the previous state. The previous statement is summarized as the joint distribution of a sequence of states and is formulated in the following way[8]: The probability of

$$P(S_{1:T}, Y_{1:T}) = P(S_1)P(Y_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(Y_t|S_t),$$

Figure 4.1: Simplified Bayesian Network Assumption [8]

the initial state at the initial time is multiplied by the probability of the observation given state at the initial time is multiplied by the join distribution of the probability of the state given the previous state at the current time multiplied by the probability of the observation given the state. [8]

Hidden Markov Models are a sublcass of Bayesian networks known as dynamic Bayesian networks. They are bayesian networks specialized in modeling time series. The assumption that time can only move forward, not backwards, simplifies the design of bayesian network where directed arcs should flow forward in time. The next aspect is to define the transition probabilites.

$$P(Y_{1:T}) = P(Y_1)P(Y_2|Y_1)\cdots P(Y_T|Y_{T-1})$$

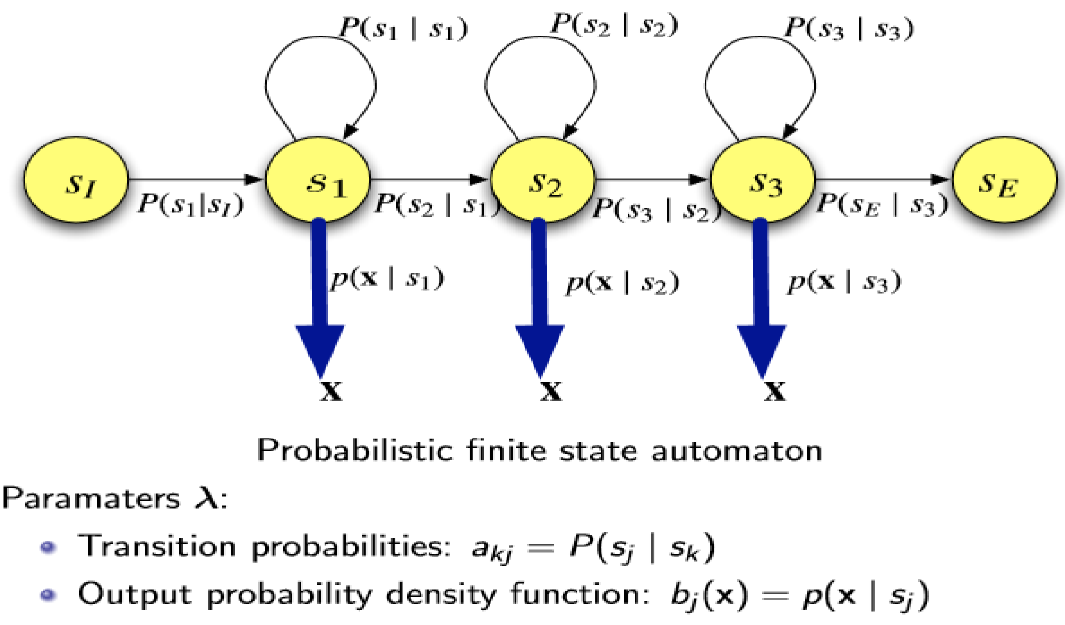
Figure 4.2

4.3 Gaussian HMM

The Gaussian Hidden Markov model can be defined as a variant of HMM where each state dependent emission probability function is calculated through a gaussian [1]. A transition matrix represents the probability of going from state at i to state at j, an emission matrix B indicating the probability of emission of symbol o from state Sj, and an initial state probability distribution (representative of first state probability).

The observation to state probability can be calculated by the following: where µj, Σj, represents the mean vector which is multiplied by the covariance matrix respectively. It is out of the scope to derive the full gaussian hmm; however, it is provided by [19] in full. The number of states is chosen based on the Bayesian Information Criterion which is defined as follows [11]: The Akaike information criterion is compared with the following equation in order to choose the optimal number of components[12]:

Using 14 components from the AIC analysis the following state transition regime is possible to be constructed through the use of the Hmmlearn framework available on github[10]. The construction of figure 5.6 is adapted from the following tutorial[4] and separates the transition regimes according to the variance and the mean according



Probabilistic finite state automaton

Paramaters λ:

- Transition probabilities: $a_{kj} = P(s_j \mid s_k)$
- Output probability density function: $b_j(\mathbf{x}) = p(\mathbf{x} \mid s_j)$

Figure 4.3: Gaussian Continuous HMM (emission matrix simplified in blue) [19]

$$b_j(\mathbf{x}) = p(\mathbf{x} \mid j) = \mathcal{N}(\mathbf{x}; \mu_j, \Sigma_j)$$

Figure 4.4: Observation probability [19]

to 14 n components. The regimes are color-coded in order to easily perceive how the behavioural regimes are divided according to their extreme or non-extreme variances. The states that are prescribed according to the variances are then used as feature inputs to the LSTM algorithm with the hypothesis that a feature engineered state will lead the algorithm to understand a more volatile regime will take place in order to better predict the spikes of such a regime. The next figure pictorially describes that given an observation of the variance and mean a state transition has a likelihood occurence of changing. It is through this methodology in which the experiment is conducted.

$$BIC = -2 \log \text{likelihood} + p \log n,$$

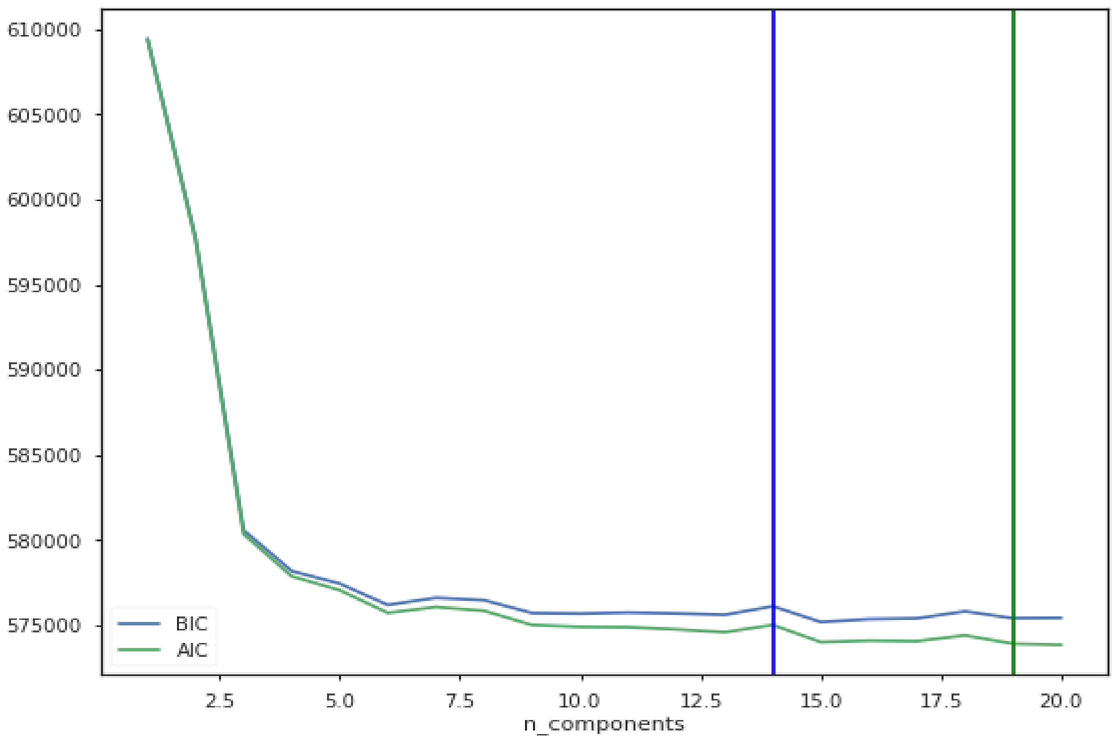


Figure 4.5: BIC vs AIC analysis

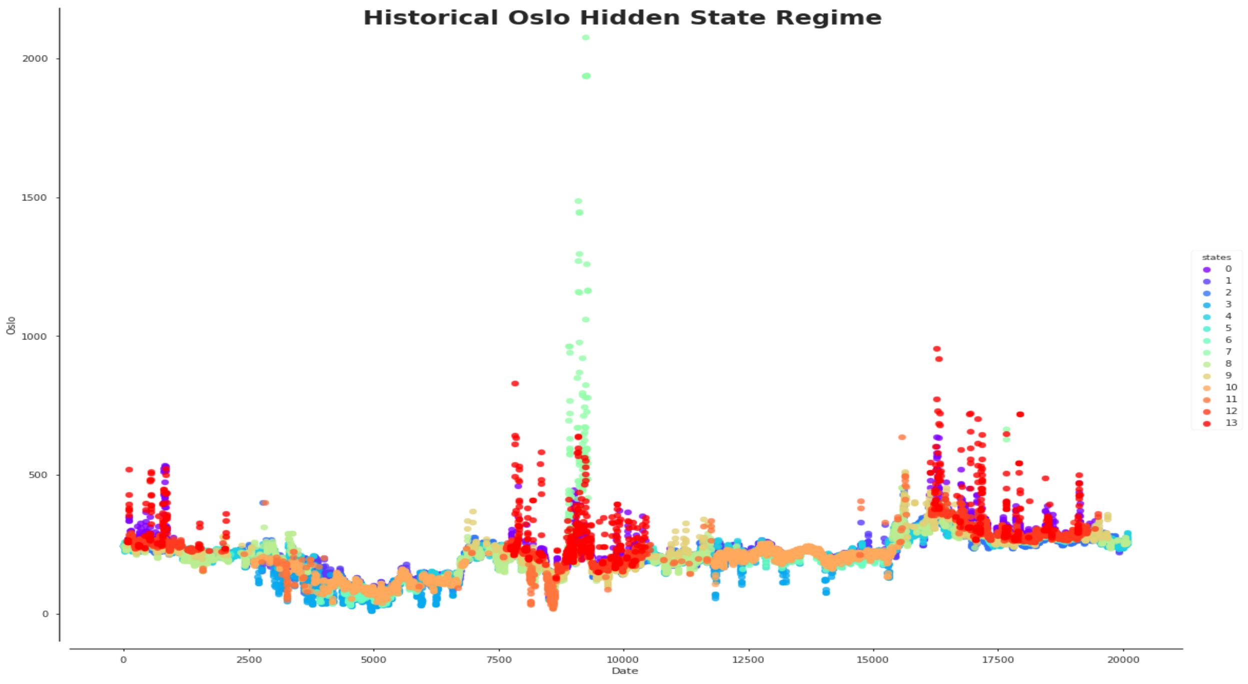


Figure 4.6: oslo historical state regime

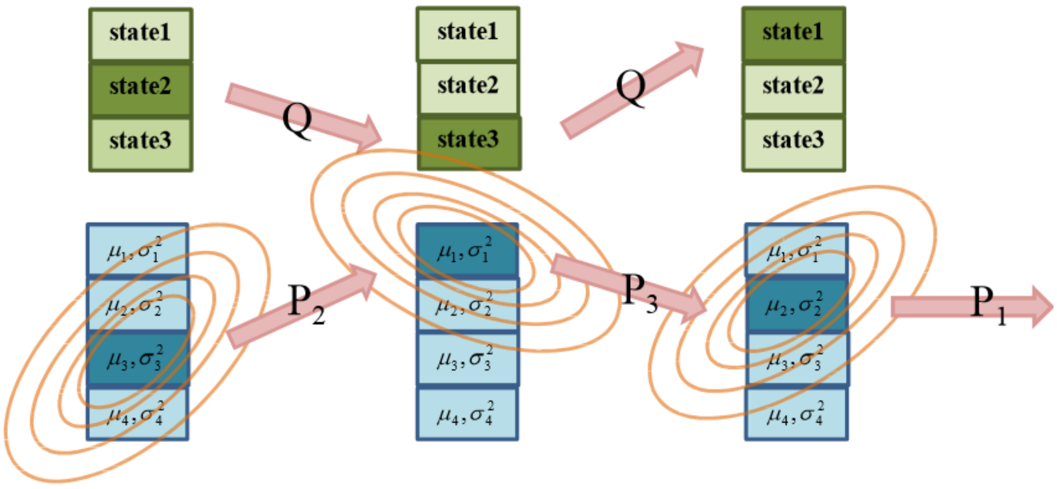


Figure 4.7: Hidden Markov Model Observation to State [13]

Chapter 5

Long Short Term Memory

The nordpool electricity market is a form of time series data. Traditionally, macroeconomic and business based methods tackled the problem of electricity market prediction- and this relied heavily on human observation of weather patterns combined with utility information [7]. The next generation of modeling time series utilized regression, which reportedly could not predict spike innacuracies. Futhermore, neural networks, deep multilayer perceptrons, have also been used to tackle spot prices. These methodologies did not work due to the temporal memory which can only be provided through a fixed sized sliding window. The neural network was a great improvement; however, it limits the sophistication of the modeling memory capabilities. Recurrent neural networks came as a resolution to this limitation by providing a gate which feeds activations from the previous time step back in as an input- this provides greater historical information to be considered in every current time step iteration. Thus temporal information is analyzed throughout the whole potential historical data that it is trained with. There were a few problems with the first models of the recurrent neural network, such as vanishing and exploding gradient – a disappearing or exponentially increasing term occurring after a multitude of iterations of taking the gradient. This has been resolved by one of the latest models introduced to the machine learning community, the long short term memory (LSTM) algorithm, that overcomes many of the problems faced by the recurrent neural network. https://arxiv.org/pdf/1603.07893.pdf The LSTM is made up of a selected threshold gate input such as the sigmoidal input, output gate, and forget gate. The novelty of this construction is that it allows the network to learn to forget previous historical inputs. The internal state of the LSTM cell is updated upon the layer being previously activated, inputs within the previous layer, as well as self-connections[4]

5.1 Keras

Keras is an open source neural network library created by Francois Chollet in 2015 as a high level framework which is used in CERN, start-up companies, google, and over 100,000 users since its creation [2]. Keras was chosen to construct the experiments due to its main support of RNN's as well as the RNN LSTM, the ability to run on tensorflow and theano, the major online community and thus documentation regarding error, the fluid pythonic language, and the fluidity in which one can merge neural networks and features together to combine the prediction analysis of datasets. The reader is provided with the exact code used for prediction in the appendix in order to further intensify the research conducted with LSTM's for the nord pool spot price. In general, if a given parameter's value is not tested throughout this paper it is because the parameters value has been suggested to be used in the keras readthedocs documentatio [2].

5.2 Network of the LSTM

LSTM is specialized in long-term dependency information. First proposed by Hochreiter Schmidhuber (1997) and was recently improved and promoted by Alex Graves[9] . In many cases, LSTM has achieved considerable success and has been widely used [9]. All RNNs have a chained form of a repeating neural network module. In the standard RNN the major difference is the simplicity of the structure, such as a tanh layer. To provide a comparison, the LSTM has 4 unique interactions that is depicted

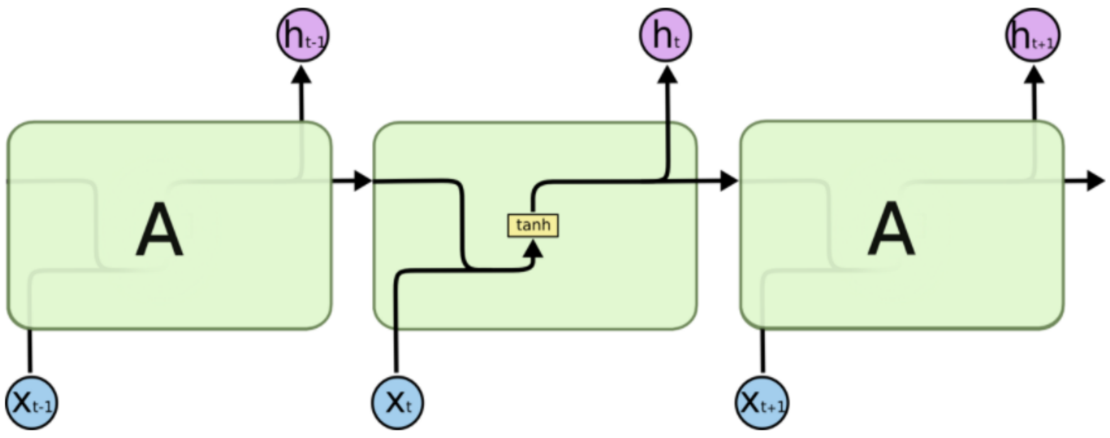


Figure 5.1: Recurrent Neural Network [17]

in the following figure: The following provides the pictorial notation for the LSTM

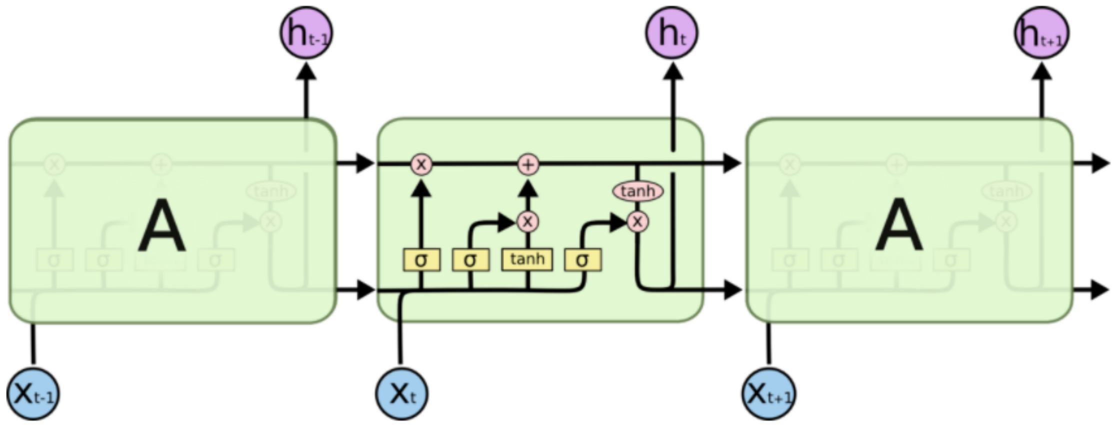


Figure 5.2: Comparison LSTM Neural Network [17]

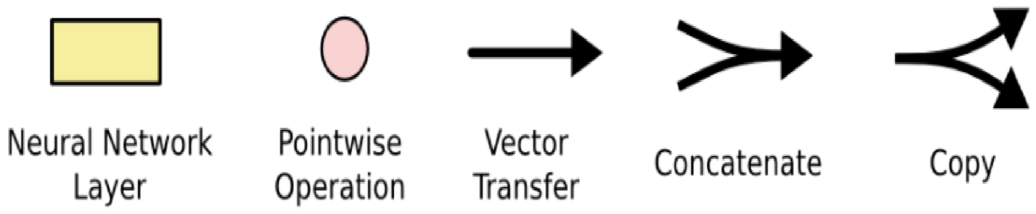


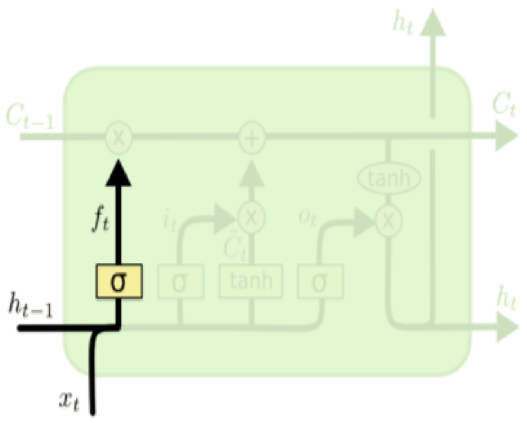
Figure 5.3: Pictorial Notation [17]

neural network We have the neural network layer in yellow which depicts the learning neural network layer. The black lines signify an entire vector, from output to input. The pink is a point-wise operation representing summation or multiplication of the/ vectors.

One can identify the key cell state that is located on the top of the figure- an analogy of this cell state is similar to that of a conveyor belt. The LSTM has the ability to forget information or increase the information to the state of the cell by a well constructed combination of gates. The gates can contain a sigmoid function (or other activation) that depending on the threshold will allow certain information to pass through. There are a total of three gates that control the cell status which is described in the next section.

5.2.1 LSTM Analysis

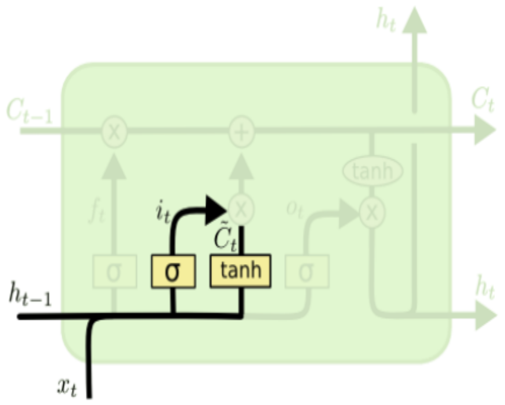
The first step in our LSTM is to decide what information we will discard from the cell state. This decision is done by a so-called door . The door reads h(t-1) and xt outputs a value between 0 and 1 to each state in the cell C(t-1) numbers. 1 means "full hold", 0 means "completely discard".



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 5.4: Forget Gate [17]

The next step can be considered the update step- determination of what kind of new information is stored in the cell state. First step, the sigmoid layer activates the "input gate" to determine the updated value. Next, the tanh layer creates a new candidate value vector C that will be added to the state.

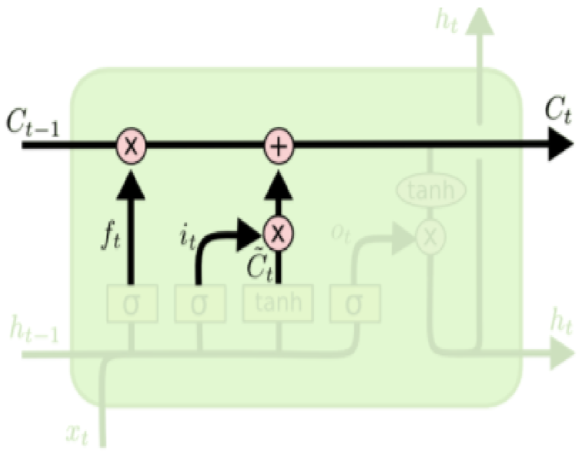


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 5.5: Update Step[17]

The previous cell state is next updated from the previous time step to the next iteration. The significance of this step is that the old state will now have the f function applied in order to determine what and how much to discard. The new candidate value is then added to this

Lastly, the output is deterimined - it is based on the cell as well as a filtered version of the cell. A sigmoid function is run and this determiens which part of the cell state to output. Afterwards, a tanh function is applied to the process of the cell state where an output of -1 and 1 occurs in order to apply a differing logic [17].



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 5.6: Update Step[17]

Chapter 6

Methodology

6.1 Batches and Epochs

An important parameter in determining the convergence of a solution is the epoch number. An epoch is defined as the entire training data exposed to the network, while the batch size is defined by the incremental gradient and the frequency in which one updates weights. The following thesis conducts 3 sets of experiments testing different network configurations with a different epoch parameter to determine the optimal methodology. In general, small batch sizes are a common modern approach to deep learning combined with a large number of epochs. The correct number of epochs can be considered to be chosen once a logarithmic curve can be found with the y-axis being the error and the x-axis the epoch.

6.2 Accuracy Validation

Mean absolute error is the main metric for validation and is calculated as follows [15]:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |(obs(i) - pred(i)|,$$

6.3 Sliding window

In order to take into account the true potential the LSTM algoroithm has to offer, the sliding window method must be used concisely. The sliding window allows for a multivariate analysis, i.e. multiple input analysis for single output. This allows one to provide a greater weight for the greatest stochastic influential factors in the inherently cyclic data sets. Considering the large variation in prices depending on the hour, the day, as well as the week a for loop is constructed to take into account the data from one week ago at an interval of 24 hours up to the current time t.

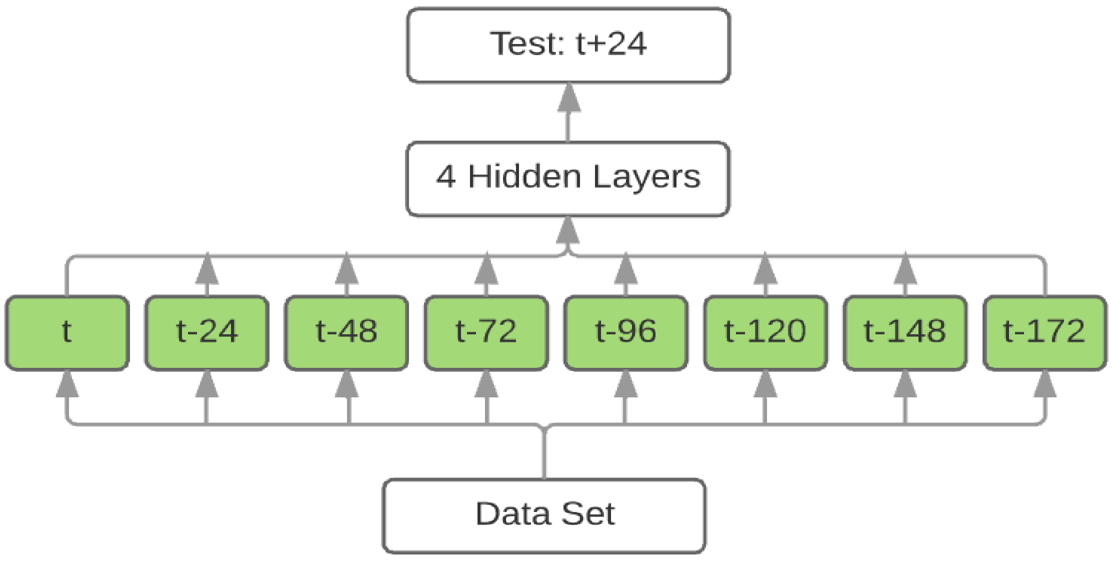


Figure 6.1: LSTM Time window input

This is conducted in the following for loop for the preprocessing section of the LSTM algorithm:

```
def train_input(dataset, look_back, ahead):
    dataX = []
    train_size = int(len(dataset) * 0.8)
    dataset = dataset[0:train_size, :]
    for i in range(len(dataset) - ahead - 1):
        a = dataset[i:1 + look_back + i:24]
        dataX.append(a)
    return numpy.array(dataX)
```

Figure 6.2: For Loop Construction

6.4 Neural Network Merging

It is possible to merge recurrent neural networks in the keras framework through the following methodology when fitting the model: The logical operations were merged as

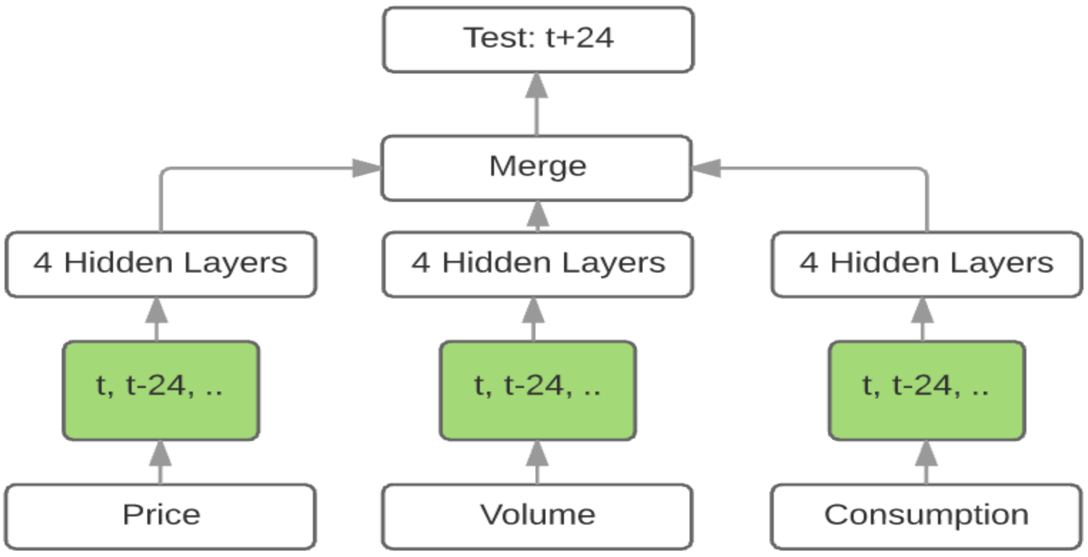


Figure 6.3: Merge

follows in order to set up the experiment: price by itself, price merged with volume, price merged with consumption, price merged with volume and consumption. The same sliding window was as well utilized for the volume and the consumption upon merging. Three models are created each with four hidden layers with 8 inputs each corresponding to the hour at the current time and the previous 7 days before at the same hour each day. The following code described how it was possible to merge the neural networks (with the full code provided in the appendix):

```
model1.add(Dense(1))
model1.compile(loss='mean_squared_error', optimizer='adam')
# model2
model2 = Sequential()
model2.add(LSTM(4, input_shape=(1, number)))
model2.add(Dense(1))
model2.compile(loss='mean_squared_error', optimizer='adam')

# model3
model3 = Sequential()
model3.add(LSTM(4, input_shape=(1, number)))
model3.add(Dense(1))
model3.compile(loss='mean_squared_error', optimizer='adam')

model = Sequential()
model.add(Merge([model1,model2, model3], mode='concat'))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'acc'])


model.fit([trainX,trainX2, trainX3], trainY, epochs=epochs, batch_size=1, verbose=2)


trainPredict = model.predict([trainX, trainX2, trainX3])
testPredict = model.predict([testX, testX2, testX3])
```

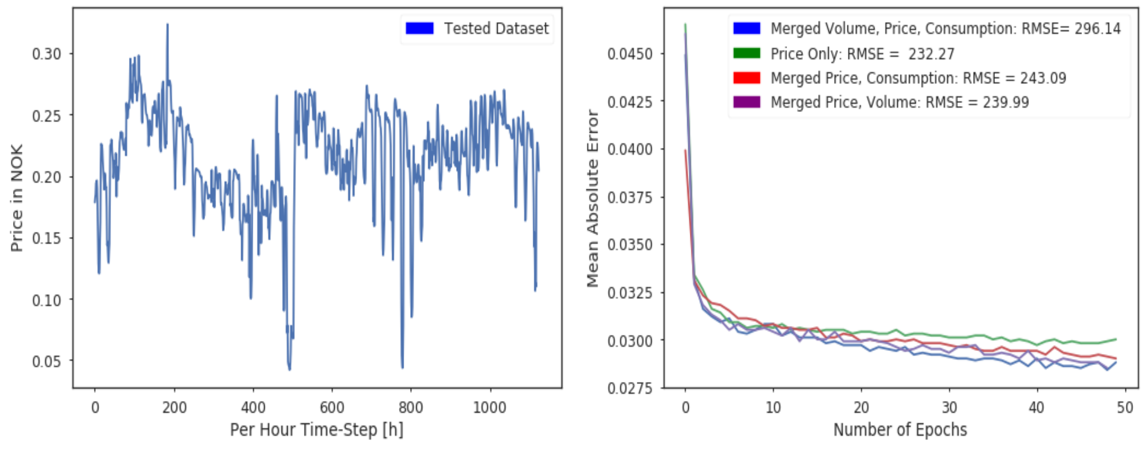
Figure 6.4: Merge Construction

Chapter 7

Results

The following results provide a comparison of utilizing the demand, volume, and consumption data sets independently, merged, and at increments of 1 year, 2 years, and 3 years to provide a comparison of the differences of accuracy in utilizing these parameters.

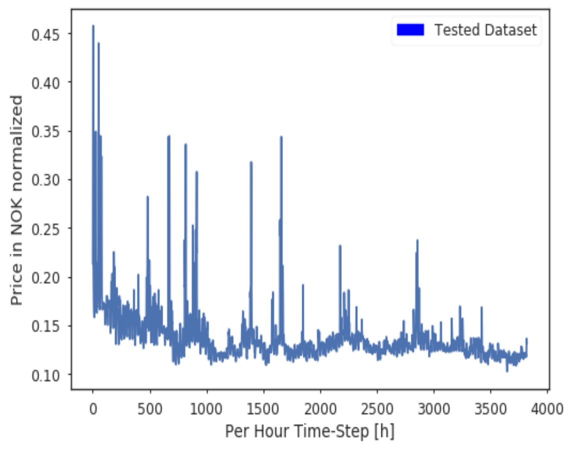
Comparison of 12 Experiments MAE			
Dataset	2014-2015	2014-2016	2014-2017
Price	0.0300	0.0094	0.0082
Price/Vol/Con	0.0288	0.0094	0.0084
Price/Vol	0.0285	0.0094	0.0085
Price/Con	0.0290	0.0095	0.0084



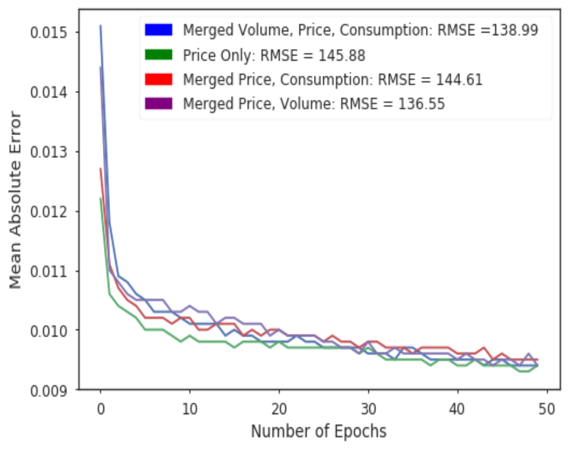
(a) Test Data Set Year 2014-2015 (b) 4 Experiments Volume, Price, Cons.

Figure 7.1: The Merged LSTM networks are compared with a 1 year analysis of data from 2014

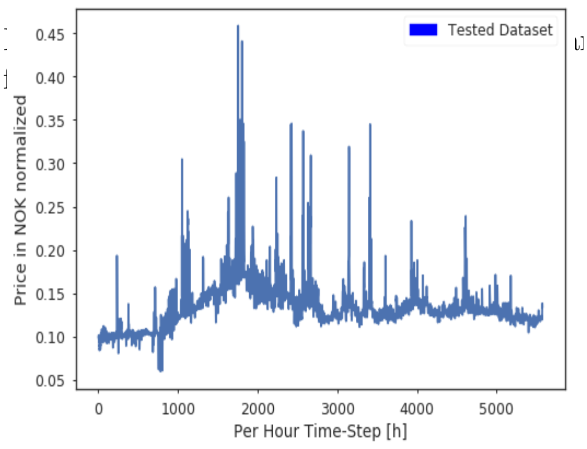
One can see that the epochs indicate a converged solution with the typical logarithmic fashion with regards to all experiments. It was found that the highest accuracy in terms of mean absolute error was in the price data set only experiment, where a 3



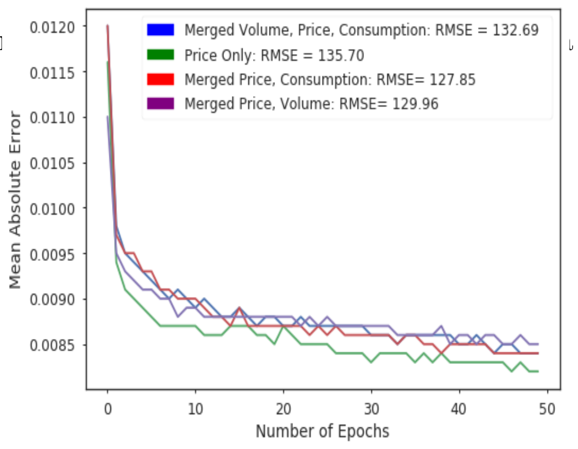
(a) Test Data Set Year 2014-2016



(b) 4 Experiments Volume, Price, Cons.



(a) Test Data Set Year 2014-2017



(b) 4 Experiments Volume, Price, Cons.

Figure 7.3: The Merged LSTM networks are compared with a 3.5 year analysis of data from 2014-2017

percent mean absolute error to 0.82 percent error in only using the price. Although, the greatest accuracy in terms of root mean square error was found to be the merged recurrent neural network of price and consumption with a value of 127.85. This was a dramatic increase in accuracy from 239.99 from the 2014-2015 data set experiment.

Comparison of 12 Experiments RMSE			
Dataset	2014-2015	2014-2016	2014-2017
Price	232.27	145.88	135.7
Price/Vol/Con	296.14	138.99	132.69
Price/Vol	239.99	136.55	129.96
Price/Con	243.09	144.61	127.85

Finally, the hidden markov model was simulated for 1 set of experiments 2014-2016 in order to test the hypothesis of implementing regime detection as a preprocessing step and further adding it as a feature could prepare the LSTM algorithm better to predict forecast such as McCandless implemented for short solar irradiance forecasting [15] as well as [16] in predicting the insulin regime and thus dose adjustments for type 1 diabetes.

HMM Regime State Feature Simulation MAE		
Dataset	2014-2016	2014-2016 with HMM
Price	0.0094	0.0096
Price/Vol/Con	0.0094	0.0097
Price/Vol	0.0094	0.0099
Price/Con	0.0095	0.0098

Unfortunately, it was found that there was little to no affect on the accuracy by adding the hidden state regime as a feature. Although the approach classified the data set according to its variance and thus classified the volatility, the LSTM did not manage to perform with greater accuracy.

As no other LSTM algorithms on Nord Pool's spot price has been conducted according to the knowledge of the author, a benchmark against research conducted in Chalmers University on the spot price through linear multiple regression (LMR) methods will be utilized [14].

Benchmark against LMR			
Dataset	Chalmer LMR	Chalmer ITSM	LSTM (UiA)
January 20	0.0362	0.0757	0.0082
March 11	0.0625	0.0317	0.0082
March 12	0.0421	0.01031	0.0082
June 30	0.0339	0.00558	0.0082

Chapter 8

Discussion

Although the gaussian hidden Markov model was not successful, it has showed promise in other papers that have sought out to detect observations through hidden states [5]. In the case of utilizing the HMM as an input for LSTM, McCandless successfully predicted solar irradation timeseries more accurately as an input to an RNN[15] as well as Mougiakakou in determining the dosage according to the insulin regime [16]. Utilizing the gaussian HMM as a feature may therefore not be the best approach for the LSTM algorithm. Another approach could be to train neural networks data specifically according to the data's regime. However, this method would lose the component of time in a time series.

However, a mean absolute error of 0.0082 was acheived in utilizing a time window of the exact previous hour of a subsequent week of data for every datastep- a multivariate analysis of 8 input steps for a one output calculation. In comparison to linear multiple regression techniques the error is three percent more accurate for the best estimation of the LMR from Chalmers which proves a promising result.

Chapter 9

Conclusion

It is absolutely vital that electricity utility company suppliers and traders are provided with accurate state of the art forecasting tools to prevent bankruptcy and increase profitability for the corporate market. An analysis of the Nord-Pool market was conducted and it was found that the LSTM algorithm can with a high accuracy predict the next day-ahead. A hidden markov behaviour regime classification was conducted and the way in which it was implemented was found to have little affect on the overall result. Another methodology must be used to further increase the LSTM's understanding of behavioural regimes in order to increase the accuracy of spike classification and prediction-which leads to the greatest losses in the Nord Pool Market. However, it is clear that a significant increase of accuracy from 3.62 percent to 0.82 percent mean absolute error is found when LSTM is used over multiple linear regression analysis.

Bibliography

[1] M. Bicego. Generalized Gaussian Distributions for Sequential Data Classication . *University of Sassari*, 2008.

[2] Francois Cholelt. Deep Learning library for Python. Convnets, recurrent neural networks, and more. Runs on TensorFlow or Theano.. 2015.

[3] T.M. Christensen. Forecasting spikes in electricity prices. *International Journal of Forecasting*, 28:400–411, 2012.

[4] Brian Christopher. Introduction to Hidden Markov Models with Ski-kit Learn . *Quantitative Analyst, Quantitative Developer-Python*, 2017.

[5] Christina Erlwein. Applications of hidden Markov models in financial modelling . *A thesis submitted for the degree of Doctor of Philosophy*, 2008.

[6] P. Exterkate. A regime-switching stochastic volatility model for forecasting electricity prices. *CREATES Research Paper*, (3), 2007.

[7] Marius Fuglerud. Bottom-Up Modeling of the Nord Pool System Price. *Norwegian University of Science and Technology. Department of Industrial Economics and Technology Management*, 2011.

[8] Zoubin Ghahramani. An Introduction to Hidden Markov Models and Bayesian Networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2015.

[9] Alex Graves. Machine Learning I Week 14: Sequence Learning Introduction. *Technische Universitat Munchen* , 2009.

[10] Hmmlearn. Hidden Markov Models in Python, with scikit-learn . 2015.

[11] Chun Yu Hong. An introduction to the use of hidden Markov models for stock return analysis . 2015.

[12] Shuhua Hu. Akaike Information Criterion . *Center for Research in Scientific Computation*, 2008.

[13] Joohyung Lee. Stock Forecasting using Hidden Markov Processes.

[14] Dawit Hailu Mazengia. Forecasting Spot Electricity Market Prices Using Time Series Models. *Chalmers University of Technology. Department of Energy Environment*, 2008.

[15] T.C. McCandless. A regime-dependent artificial neural network technique for short-range solar irradiance forecasting. *Renewable Energy*, (89):351–359, 2016.

[16] Stavroula G. Mougiakakou. A Neural Network Approach for Insulin Regime and Dose Adjustment in Type 1 Diabetes. *Diabetes technology Therapeutics*, 2(3):381–389, 2000.

[17] Chris Olah. Understanding LSTM Networks. *Google Brain, Research Scientist*, 2015.

[18] Nord Pool. .

[19] Steve Renals. Hidden Markov Models and Gaussian Mixture Models. *Automatic Speech Recognition*, ASR Lectures 45, 2013.

[20] Rafał Weron. Modeling and Forecasting Electricity Loads and Prices. *A Statistical Approach*, 2006.

[21] Rafał Weron. Forecasting spot electricity prices: A comparison of parametric and semiparametric time series models. *International Journal of Forecasting*, (24):744–763, 2008.

Initialization of Data and Merging of Neural Networks

May 18, 2017

```
In [ ]: import matplotlib as mpl
        import matplotlib.pyplot as plt
        from matplotlib.dates import YearLocator, MonthLocator
        %matplotlib inline
        import matplotlib.patches as mpatches
        import seaborn as sns
        import missingno as msno
        from tqdm import tqdm
        import math
        import warnings
        import pandas
        import math
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import LSTM
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.metrics import mean_squared_error
        from pandas import read_csv, DataFrame
        from numpy.random import seed
        from sklearn.preprocessing import scale
        from keras.models import Sequential
        from keras.constraints import maxnorm
        from keras.optimizers import SGD, RMSprop
        from keras.layers import Dense, Merge
        import numpy
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.metrics import mean_squared_error
        from pandas import read_csv, DataFrame
        from numpy.random import seed
        from sklearn.preprocessing import scale
        from keras.models import Sequential
        from keras.constraints import maxnorm
        from keras.optimizers import SGD, RMSprop, Nadam
        from keras.layers import Dense, Merge
        scaler = MinMaxScaler(feature_range=(0, 1))
        skipfooter2 = 13500
        def getData(csv, col, skiprows, skipfooter, comma = True, **kwargs):
            dataframe = pandas.read_csv(csv, usecols=[col], engine='python', skiprows= skiprows, skipfooter
            dataframe.dropna(how='all', inplace=True)
            if comma:
                dataframe = dataframe.stack().str.replace(',', '.').unstack()
            dataframe = dataframe.apply(pandas.to_numeric)
            dataset = dataframe.values
            dataset = dataset.astype('float32')
            dataset = scaler.fit_transform(dataset)
            return dataframe, dataset
            #
        priceDF, priceDS = getData('../price.csv', 10, 2, 24+skipfooter2, comma= True)
        priceDFShift, priceDSShift = getData('../price.csv', 10, 26, 0+skipfooter2, comma= True)
        consumeDF, consumeDS = getData('../consume.csv', 2, 2, 193+49+skipfooter2, comma= False)
        volumeDF, volumeDS = getData('../volume.csv', 3, 2, 312+skipfooter2, comma= True)

        def my_test(a, b):
            return np.log(a/b)

        data = pd.concat([volumeDF, consumeDF, priceDF, priceDFShift], axis=1)
        data.columns.values[3] = 'OsloF'
        lret = data.apply(lambda column: my_test(column['OsloF'], column['Oslo']), axis=1)

        data = pd.concat([data, lret], axis=1)
        data.columns.values[4] = 'lret'
        plt.plot(priceDS)

In [ ]: from keras.callbacks import ModelCheckpoint



        def train_input(dataset, look_back, ahead):
            dataX = []
            train_size = int(len(dataset) * 0.8)
            dataset = dataset[0:train_size, :]
            for i in range(len(dataset) - ahead - 1):
                a = dataset[i:1 + look_back + i:24]
                dataX.append(a)
            return numpy.array(dataX)

        def train_output(dataset, look_back, ahead):
            dataX= []
            train_size = int(len(dataset) * 0.8)
            dataset = dataset[0:train_size, :]
            for i in range(len(dataset) - ahead - 1):
                dataX.append(dataset[i + ahead, 0])
            return numpy.array(dataX)

        def test_input(dataset, look_back, ahead):
            dataX = []
            train_size = int(len(dataset) * 0.8)
        #     test_size = dataset[train_size:len(dataset), :]
            dataset = dataset[train_size:len(dataset), :]
            for i in range(len(dataset) - ahead - 1):
                a = dataset[i:1 + look_back + i:24]
                dataX.append(a)
            return numpy.array(dataX)

        def test_output(dataset, look_back, ahead):
            dataX = []
            train_size = int(len(dataset) * 0.8)
        #     test_size = dataset[train_size:len(dataset), :]
            dataset = dataset[train_size:len(dataset), :]
            for i in range(len(dataset) - ahead - 1):
                dataX.append(dataset[i + ahead, 0])
            return numpy.array(dataX)

        def LSTM_Nordpool3(days, look_back, ahead, number, epochs):
            trainX= train_input(priceDS, look_back, ahead)
```

```
    trainX2 = train_input(consumeDS, look_back, ahead)
    trainY = train_output(priceDS, look_back, ahead)
    testX = test_input(priceDS, look_back, ahead)
    testX2 = test_input(consumeDS, look_back, ahead)
    testY = test_output(priceDS, look_back, ahead)
    trainX = numpy.reshape(trainX, (trainX.shape[0], 1, number))
    trainX2 = numpy.reshape(trainX2, (trainX2.shape[0], 1, number))
    testX = numpy.reshape(testX, (testX.shape[0], 1, number))
    testX2 = numpy.reshape(testX2, (testX2.shape[0], 1, number))
    RMS = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    Nad = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
    # model1
    model1 = Sequential()
    model1.add(LSTM(4, input_shape=(1, number)))  #
    model1.add(Dense(1))
    model1.compile(loss='mean_squared_error', optimizer='adam')
    # model2
    model2 = Sequential()
    model2.add(LSTM(4, input_shape=(1, number)))  #
    model2.add(Dense(1))
    model2.compile(loss='mean_squared_error', optimizer='adam')
    # model3
    model = Sequential()
    model.add(Merge([model1, model2], mode='concat'))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'acc'])


    model.fit([trainX, trainX2], trainY, epochs=epochs, batch_size=1, verbose=2)  #


    trainPredict = model.predict([trainX, trainX2])
    testPredict = model.predict([testX, testX2])
    # #
    trainPredict = scaler.inverse_transform(trainPredict)
    trainY = scaler.inverse_transform([trainY])
    testPredict = scaler.inverse_transform(testPredict)
    testY = scaler.inverse_transform([testY])
    #

    trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
    print('Train Score: %.2f RMSE' % (trainScore))

    testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
    print('Test Score: %.2f RMSE' % (testScore))
```

```
In [ ]: def train_input(dataset, look_back, ahead):
            dataX = []
            train_size = int(len(dataset) * 0.8)
            dataset = dataset[0:train_size, :]
            for i in range(len(dataset) - ahead - 1):
                a = dataset[i:1 + look_back + i:24]
                dataX.append(a)
            return numpy.array(dataX)

        def train_output(dataset, look_back, ahead):
            dataX= []
```

```
    train_size = int(len(dataset) * 0.8)
    dataset = dataset[0:train_size, :]
    for i in range(len(dataset) - ahead - 1):
        dataX.append(dataset[i + ahead, 0])
    return numpy.array(dataX)

def test_input(dataset, look_back, ahead):
    dataX = []
    train_size = int(len(dataset) * 0.8)
#     test_size = dataset[train_size:len(dataset), :]
    dataset = dataset[train_size:len(dataset), :]
    for i in range(len(dataset) - ahead - 1):
        a = dataset[i:1 + look_back + i:24]
        dataX.append(a)
    return numpy.array(dataX)

def test_output(dataset, look_back, ahead):
    dataX = []
    train_size = int(len(dataset) * 0.8)
#     test_size = dataset[train_size:len(dataset), :]
    dataset = dataset[train_size:len(dataset), :]
    for i in range(len(dataset) - ahead - 1):
        dataX.append(dataset[i + ahead, 0])
    return numpy.array(dataX)

def LSTM_Nordpool2(days, look_back, ahead, number, epochs):
    trainX= train_input(priceDS, look_back, ahead)

    trainY = train_output(priceDS, look_back, ahead)

    testX = test_input(priceDS, look_back, ahead)

    testY = test_output(priceDS, look_back, ahead)

    trainX = numpy.reshape(trainX, (trainX.shape[0], 1, number))


    testX = numpy.reshape(testX, (testX.shape[0], 1, number))

    RMS = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    Nad = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
    model = Sequential()
    model.add(LSTM(4, input_shape=(1, number)))
    model.add(Dense(1))


    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'acc'])


    model.fit([trainX], trainY, epochs=epochs, batch_size=1, verbose=2)

    trainPredict = model.predict([trainX])
    testPredict = model.predict([testX])
    trainPredict = scaler.inverse_transform(trainPredict)
    trainY = scaler.inverse_transform([trainY])
    testPredict = scaler.inverse_transform(testPredict)
    testY = scaler.inverse_transform([testY])
```

```
        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
        print('Train Score: %.2f RMSE' % (trainScore))

        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
        print('Test Score: %.2f RMSE' % (testScore))

In [ ]: def train_input(dataset, look_back, ahead):
        dataX = []
        train_size = int(len(dataset) * 0.8)
        dataset = dataset[0:train_size, :]
        for i in range(len(dataset) - ahead - 1):
            a = dataset[i:1 + look_back + i:24]
            dataX.append(a)
        return numpy.array(dataX)

    def train_output(dataset, look_back, ahead):
        dataX= []
        train_size = int(len(dataset) * 0.8)
        dataset = dataset[0:train_size, :]
        for i in range(len(dataset) - ahead - 1):
            dataX.append(dataset[i + ahead, 0])
        return numpy.array(dataX)

    def test_input(dataset, look_back, ahead):
        dataX = []
        train_size = int(len(dataset) * 0.8)
    #     test_size = dataset[train_size:len(dataset), :]
        dataset = dataset[train_size:len(dataset), :]
        for i in range(len(dataset) - ahead - 1):
            a = dataset[i:1 + look_back + i:24]
            dataX.append(a)
        return numpy.array(dataX)

    def test_output(dataset, look_back, ahead):
        dataX = []
        train_size = int(len(dataset) * 0.8)
        dataset = dataset[train_size:len(dataset), :]
        for i in range(len(dataset) - ahead - 1):
            dataX.append(dataset[i + ahead, 0])
        return numpy.array(dataX)

    def LSTM_Nordpool4(days, look_back, ahead, number, epochs):
        trainX= train_input(priceDS, look_back, ahead)

        trainX3 = train_input(volumeDS, look_back, ahead)

        trainY = train_output(priceDS, look_back, ahead)

        testX = test_input(priceDS, look_back, ahead)

        testX3 = test_input(volumeDS, look_back, ahead)


        testY = test_output(priceDS, look_back, ahead)

        trainX = numpy.reshape(trainX, (trainX.shape[0], 1, number))
        trainX3 = numpy.reshape(trainX3, (trainX3.shape[0], 1, number))


        testX = numpy.reshape(testX, (testX.shape[0], 1, number))

        testX3 = numpy.reshape(testX3, (testX3.shape[0], 1, number))


        RMS = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
        Nad = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
        # model1
        model1 = Sequential()
        model1.add(LSTM(4, input_shape=(1, number)))
        model1.add(Dense(1))
        model1.compile(loss='mean_squared_error', optimizer='adam')


        # model3
        model3 = Sequential()
        model3.add(LSTM(4, input_shape=(1, number)))
        model3.add(Dense(1))
        model3.compile(loss='mean_squared_error', optimizer='adam')

        model = Sequential()
        model.add(Merge([model1, model3], mode='concat'))
        model.add(Dense(1))

        model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'acc'])


        model.fit([trainX, trainX3], trainY, epochs=epochs, batch_size=1, verbose=2)


        trainPredict = model.predict([trainX, trainX3])
        testPredict = model.predict([testX, testX3])
        # # invert predictions
        trainPredict = scaler.inverse_transform(trainPredict)
        trainY = scaler.inverse_transform([trainY])
        testPredict = scaler.inverse_transform(testPredict)
        testY = scaler.inverse_transform([testY])
        # calculate root mean squared error

        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
        print('Train Score: %.2f RMSE' % (trainScore))

        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
        print('Test Score: %.2f RMSE' % (testScore))
```

```
In [ ]: def train_input(dataset, look_back, ahead):
            dataX = []
            train_size = int(len(dataset) * 0.8)
            dataset = dataset[0:train_size, :]
            for i in range(len(dataset) - ahead - 1):
                a = dataset[i:1 + look_back + i:24]
                dataX.append(a)
            return numpy.array(dataX)

        def train_output(dataset, look_back, ahead):
```

```
    dataX= []
    train_size = int(len(dataset) * 0.8)
    dataset = dataset[0:train_size, :]
    for i in range(len(dataset) - ahead - 1):
        dataX.append(dataset[i + ahead, 0])
    return numpy.array(dataX)

def test_input(dataset, look_back, ahead):
    dataX = []
    train_size = int(len(dataset) * 0.8)
    dataset = dataset[train_size:len(dataset), :]
    for i in range(len(dataset) - ahead - 1):
        a = dataset[i:1 + look_back + i:24]
        dataX.append(a)
    return numpy.array(dataX)

def test_output(dataset, look_back, ahead):
    dataX = []
    train_size = int(len(dataset) * 0.8)
    dataset = dataset[train_size:len(dataset), :]
    for i in range(len(dataset) - ahead - 1):
        dataX.append(dataset[i + ahead, 0])
    return numpy.array(dataX)

def LSTM_Nordpool(days, look_back, ahead, number, epochs):
    trainX= train_input(priceDS, look_back, ahead)
    trainX2= train_input(consumeDS, look_back, ahead)

    trainX3 = train_input(volumeDS, look_back, ahead)

    trainY = train_output(priceDS, look_back, ahead)

    testX = test_input(priceDS, look_back, ahead)
    testX2 = test_input(consumeDS, look_back, ahead)

    testX3 = test_input(volumeDS, look_back, ahead)


    testY = test_output(priceDS, look_back, ahead)

    trainX = numpy.reshape(trainX, (trainX.shape[0], 1, number))
    trainX2 = numpy.reshape(trainX2, (trainX2.shape[0], 1, number))

    trainX3 = numpy.reshape(trainX3, (trainX3.shape[0], 1, number))


    testX = numpy.reshape(testX, (testX.shape[0], 1, number))
    testX2 = numpy.reshape(testX2, (testX2.shape[0], 1, number))

    testX3 = numpy.reshape(testX3, (testX3.shape[0], 1, number))


    RMS = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
    Nad = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-08, schedule_decay=0.004)
    # model1
    model1 = Sequential()
    model1.add(LSTM(4, input_shape=(1, number)))
```

```
        model1.add(Dense(1))
        model1.compile(loss='mean_squared_error', optimizer='adam')
        # model2
        model2 = Sequential()
        model2.add(LSTM(4, input_shape=(1, number)))
        model2.add(Dense(1))
        model2.compile(loss='mean_squared_error', optimizer='adam')

        # model3
        model3 = Sequential()
        model3.add(LSTM(4, input_shape=(1, number)))
        model3.add(Dense(1))
        model3.compile(loss='mean_squared_error', optimizer='adam')

        model = Sequential()
        model.add(Merge([model1,model2, model3], mode='concat'))
        model.add(Dense(1))

        model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'acc'])


        model.fit([trainX,trainX2, trainX3], trainY, epochs=epochs, batch_size=1, verbose=2)  #


        trainPredict = model.predict([trainX, trainX2, trainX3])
        testPredict = model.predict([testX, testX2, testX3])

        trainPredict = scaler.inverse_transform(trainPredict)
        trainY = scaler.inverse_transform([trainY])
        testPredict = scaler.inverse_transform(testPredict)
        testY = scaler.inverse_transform([testY])


        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
        print('Train Score: %.2f RMSE' % (trainScore))

        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
        print('Test Score: %.2f RMSE' % (testScore))
```

```
In [ ]: days = 7
        look_back = 24 * days
        ahead = look_back + 24
        number = look_back // 24 + 1
        LSTM_Nordpool(days, look_back, ahead, number, epochs = 50)
```

```
In [ ]: days = 7
        look_back = 24 * days
        ahead = look_back + 24
        number = look_back // 24 + 1
        LSTM_Nordpool2(days, look_back, ahead, number, epochs = 50)
```

```
In [ ]: days = 7
        look_back = 24 * days
        ahead = look_back + 24
        number = look_back // 24 + 1
        LSTM_Nordpool3(days, look_back, ahead, number, epochs = 50)
```

```
In [ ]: days = 7
        look_back = 24 * days
        ahead = look_back + 24
        number = look_back // 24 + 1
        LSTM_Nordpool4(days, look_back, ahead, number, epochs = 50)
```

```
In [ ]: testY = test_output(priceDS, look_back, ahead)
        plt.figure()
        plt.plot((testY))
        plt.ylabel('Price in NOK normalized')
        plt.xlabel('Per Hour Time-Step [h]')
        blue_patch = mpatches.Patch(color='blue', label='Tested Dataset')
        plt.legend(handles=[blue_patch])
```

```
In [ ]: testY = test_output(priceDS, look_back, ahead)
        plt.figure()
        plt.plot(P1)
        plt.plot(P2)
        plt.plot(P3)
        plt.plot(P4)
        plt.ylabel('Mean Absolute Error')
        plt.xlabel('Number of Epochs')
        blue_patch = mpatches.Patch(color='blue', label='Merged Volume, Price, Consumption: RMSE= 296.14 ')
        green_patch = mpatches.Patch(color='green', label='Price Only: RMSE =  232.27')
        red_patch  = mpatches.Patch(color='red', label='Merged Price, Consumption: RMSE = 243.09 ')
        purple_patch = mpatches.Patch(color='purple', label='Merged Price, Volume: RMSE = 239.99')
        plt.legend(handles=[blue_patch, green_patch, red_patch, purple_patch])
```