STONY BROOK UNIVERSITY

APPLIED MATHEMATICS AND STATISTICS DEPARTMENT

AMS522- BAYESION METHODS IN FINANCE

PROJECT REPORT

Comparing Benchmark Models with Long-Short Term Neural Networks (LSTM) for Day-Ahead Electricity Price Forecasting

Name: Akshay Kurup

SBU Id: 114834368

Name: MVD Satya Swaroop

SBU Id: 114388425

Name: Revanth Kumar Seerangaswamy

SBU Id: 114779414

Problem Statement

The objective of this project is to develop a day-ahead electricity price forecasting model using Long-Short Term Neural Networks using the data from various markets including Nord Pool, PJM, EPEX-BE, EPEX-FR, and EPEX-DE. The model's performance will be evaluated and compared with existing models such as DNN and LEAR to determine its effectiveness.

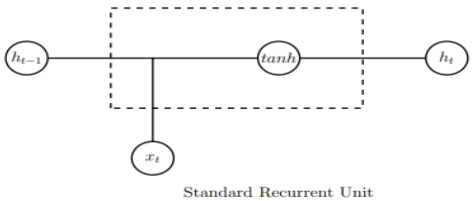
Introduction

The market for electricity is a crucial sector of the economy. Since the inclusion of renewable energy sources in today's power systems makes electricity output more variable and the corresponding electricity prices harder to estimate than ever before. For these reasons, forecasting of renewable energy sources and electricity prices is a complicated topic that is frequently covered in scholarly literature. Accurate electricity price forecasting is essential for the trading and production of energy Traditional quantitative strategy methodology uses linear regressions, the ARIMA model, and the GARCH model. During the previous regimes, these techniques were demonstrated to be successful for a while. These models lost their effectiveness as the industry as a whole underwent a regime shift. Thus, the "deep learning era"—a term that is now more commonly used—has entered the quantitative trading business. In our study, we focus on using Long Short Term Memory network, a special kind of RNN, capable of learning long-term dependencies. Our objective is to develop a LSTM forecasting model that predicts day ahead electricity prices and evaluate the performance of the model by comparing with existing models such as LEAR and DNN discussed in the paper “Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. Volume 293, 2021, 116983, ISSN 0306-2619” which are proven to be highly accurate, which will be used as a benchmark.

Methodology

LSTM

Traditionally, whenever conventional neural networks come to a problem, it may not retain any information from its previous prediction but instead begin from scratch again based on old information. Luckily, the new kind of network, Recurrent Neural Network (RNN), introduced by Rumelhart (1986) [25] could overcome this problem. In the hidden layer, while a normal Feed Forward Neural Network (FFNN) only contains simple nodes, the hidden node of RNN is a recurrent unit that allows the network to process a sequence of data. To be clear, the information of the previous timestep would be recurrently passed to the current node and combined with current information to generate the prediction. The basic structure of RNNs has 2 major steps, with the first one recurrently combining current information (X_t) with previously hidden state information ($h_t - 1$) and the second one is going through an activation function (tanh). This process repeats with a certain number of n time steps (predefined).



Standard Recurrent Unit

Fig(1)

However, traditional RNNs are prone to vanishing and exploding gradient issues, and, thus, it could be a problem for those to tackle long-term temporal dependency problems. The gradient disappearance problems could be explained by the exponential decay of the gradient of the loss function with time. Therefore, Long Short-Term memory was introduced by Hochreiter and Schmidhuber (1997) to overcome this problem through memory cells. In short, LSTM adds cell states that could retain all long-term memory for the LSTM model. The structure of the LSTM model is as below.

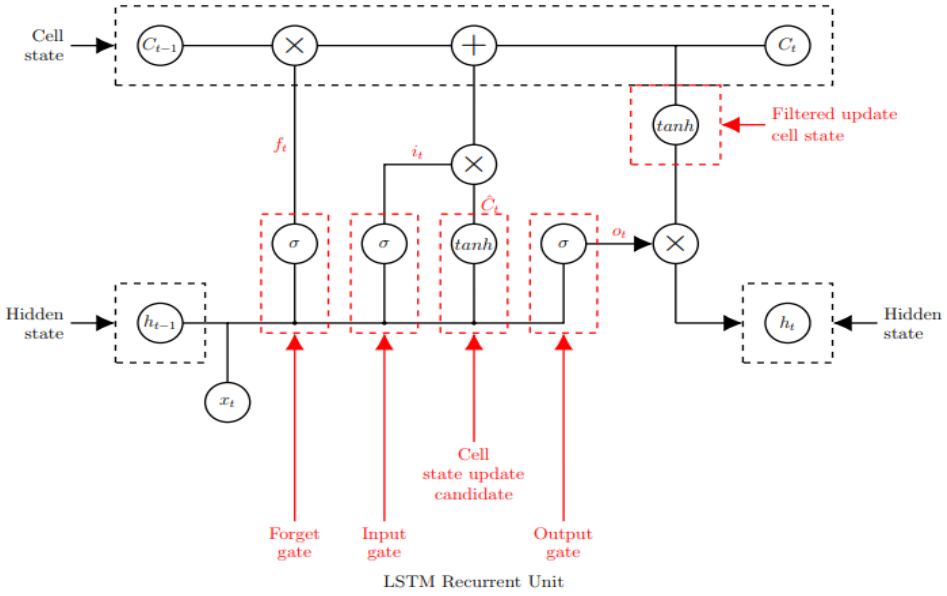


Fig (2)

Cell State

The cell state C is the key operation of the LSTM network (the horizontal line at the top of the diagram), which allows the flow of previous information to pass through LSTM unchanged. Therefore, LSTM utilizes cell states to remember information from the past. Moreover, the information in the cell state could be updated with a gating regulation consisting of 3 main gates: the forget gate, the input gate, and the output gate.

Forget Gate

The forget gate decides which information should be retained and which should be forgotten. The decision is made by a sigmoid layer. Since the value of the sigmoid function ranges between 0 and 1, a value of 1 shows that all information should be remembered while a value of 0 represents this information should be discarded.

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

Input Gate

The input gate decides which information will be used to update the cell states. Moreover, the adjacent tanh layer would create a new candidate for the cell state update. The last stage is to multiply the results of the input gate and that of the cell state candidate, which only allows important information from the input gate to be added to the current cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\widehat{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c$$

Hence, the updated cell state is. In other words, this formula represents the fact that we forgot the old information (through the value of the forget gate) and add new information we decided in the input gate.

$$C_t = f_t * C_{t-1} + i_t * \widehat{C_t}$$

Output gate

The output gate decides what information we should output. To be clear, the values of the sigmoid layer would decide appropriate output information by multiplying with that of the current cell state that has been pre-processed with the tanh layer.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o$$

$$h_t = o_t * \tanh(C_t)$$

DNN - (Deep Neural Network)

A DNN is a type of artificial neural network that consists of multiple layers of interconnected neurons. Each neuron performs a linear combination of its input followed by a non-linear activation function, and the outputs of one layer serve as the inputs to the next layer.

Given an input vector X, the output of the first layer can be represented as:

$$Z_1 = f_1(W_1 * X + b_1)$$

where W_1 is a weight matrix, b_1 is a bias vector, f_1 is an activation function,

The output of the first layer is then used as the input to the second layer:

$$Z_2 = f_2(W_2 * Z_1 + b_2)$$

where W_2 is a weight matrix, b_2 is a bias vector, f_2 is an activation function.

This process is repeated for each subsequent layer, until the final layer produces the output vector Y:

$$Y = f_l(W_L * Z_{L-1} + b_L)$$

where W_L and b_L are the weight matrix and bias vector of the final layer, respectively, and L represents the number of layers in the network.

The weights and biases are learned through a process called backpropagation, which uses gradient descent to minimize a loss function that measures the difference between the network's output and the desired output. It can be used for a variety of tasks, including image classification, speech recognition, and natural language processing.

LEAR - (Lasso Estimated Auto Regressive Model)

The LEAR is based on the so-called full ARX or fARX model, a parameter-rich autoregressive specification with exogenous variables, which in turn is inspired by the general autoregressive model, while fARX includes fundamentals and a much richer seasonal structure, it does not look too far into the past and concentrates only on the last week of data.

The data is preprocessed with the area (or inverse) hyperbolic sine variance stabilizing transformation.

$$asinh(x) = log(x + \sqrt{x^2 + 1})$$

where x is the price standardized by subtracting the in-sample median and dividing by the median absolute deviation adjusted by a factor for asymptotically normal consistency to the standard deviation. To further enhance the model, we recalibrate it daily over different calibration window lengths: 8 weeks, 12 weeks, 3 years, and 4 years. We consider short windows (8–12 weeks) in combination with long windows (3–4 years) because it has been empirically shown to lead to better results. We consider a minimum of 8 weeks, as lower windows might not have enough information to correctly estimate the parameter rich models. The LEAR model to predict the price $p_{d,h}$ on day d and hour h is defined by:

$$p_{d,h} = f(p_{d-1}, p_{d-2}, p_{d-3}, p_{d-7}, x_d^i, x_{d-1}^i, x_{d-7}^i, \theta_h) + \epsilon_{d,h}$$

Where $\theta_h = \left[\theta_{h,1}, \theta_{h,2}, \ldots\ldots, \theta_{h,247},\right]^T$ are the 247 parameters of the LEAR model for hour h. Many of these parameters are zero when estimated by LASSO.

$$\widehat{\theta_h} = argmin_{\theta_h} RSS + \lambda \left|\left|\theta_h\right|\right|_1 = argmin_{\theta_h} RSS + \lambda \sum_{i=1}^{247} \left|\theta_{h,i}\right|$$

Where $RSS = \sum_{d=8}^{N_d} (p_{d,h} - \hat{p}_{d,h})^2$ is the sum of squared residuals, $\hat{p}_{d,h}$ the price forecast N_d is the number of days in the training dataset. $\lambda \geq 0$ is the tuning hyperparameter of LASSO. To reduce the computational cost, we propose an efficient hybrid approach to perform optimal selection of λ. To select λ, we propose a hybrid approach. On a daily basis, we estimate the hyperparameter using the LARS method with the in-sample AIC. Then, using the optimal λ obtained from the LARS method, we recalibrate the LEAR using the traditional coordinate descent implementation. The reason for proposing this hybrid approach is that it provides a good trade-off between computational complexity and accuracy.

Dataset

The paper mentioned above provides us large open-access benchmark dataset on which new methods can be tested, together with the day-ahead forecasts of the proposed open-access methods. We are provided with five datasets representing five different day-ahead electricity markets, each of them comprising 6 years of data. The prices of each market have very distinct dynamics, i.e., they all have differences in terms of the frequency and existence of negative prices, zeros, and price spikes. The length of each dataset equals 2184 days, which translates to six 364-day "years" or 312 weeks.

The first and second dataset represents the Nord Pool (NP), i.e. the European power market of the Nordic countries, and Pennsylvania–New Jersey– Maryland (PJM) market in the United States respectively and covers the same time period , i.e. from 01.01.2013 to 24.12.2018.The third dataset represents the EPEX-BE market, the day-ahead electricity market in Belgium and the fourth dataset represents the EPEX-FR market, the day-ahead electricity market in France ,both the markets are operated by EPEX SPOT and spans from 09.01.2011 to 31.12.2016. The last dataset describes the EPEX-DE market, the German electricity market, which is also operated by EPEX SPOT and the dataset spans from 09.01.2012 to 31.12.2017.

The testing period is defined as the last 104 weeks, i.e., the last two years of the dataset. Each market dataset is split into train and test as shown below for the PJM market.

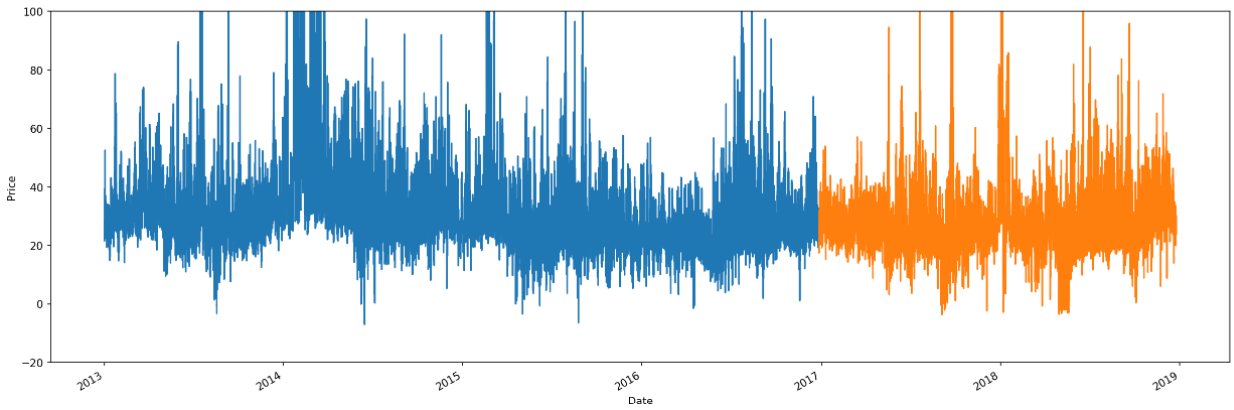


Fig (3)

Approach

To predict 1 day (24 hours data) future time steps, we utilize a look-back approach which involves concatenating the last "lookback" number of rows from the training dataset with the test dataset. This step is crucial to enable the model to make predictions for the test dataset based on the previous "lookback" number of observations in the training dataset. The concatenated dataset is then reindexed by date. In order to standardize the dataset, we employed the data transformation from epftoolbox DataScaler. This involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.

We create a function with the purpose to create data that can be used to train an LSTM model. The data parameter is expected to be a 2D NumPy array with one column, and lookback is an integer value that represents the number of previous time steps to use as input for the model. The function then iterates over the input data starting from the index lookback.

For each iteration, it appends the previous lookback number of rows from the input data as the input X and the next row as the output y. It does this until it reaches the end of the input data. Finally, the function returns two NumPy arrays, X and y, which contain the input and output data for training the LSTM model. The LSTM network expects the input data (X) to be provided with a specific array structure in the form of [short term memory, input shape]. You can transform the prepared train and test input data into the expected structure and, now ready to design and fit your LSTM network for this problem.

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. The model architecture consists of two LSTM layers, followed by a dense layer with one output unit. The first LSTM layer has 64 units and returns sequences of data to the second LSTM layer, while the second LSTM layer has 32 units and returns a single output value. The input shape for the LSTM layers is specified as (lookback, 1), where lookback represents the number of time steps in the input sequence. The linear activation function is used in the output layer, which means the model will predict a continuous value. The loss function used to evaluate the model's performance is mean squared error, and the optimizer used to update the model's weights during training is Adam, which is an adaptive learning rate optimization algorithm.

The fit method is used to train the model, i.e., involves iterating over the training data for a specified number of iterations, updating the model's weights to minimize the loss function. A batch size of 64 is used to specify the number of samples that are processed at once during each iteration. Additionally, a validation split of 0.1 is used to reserve 10% of the training data for validation, which is used to evaluate the model's performance on data that it has not seen during training. The verbose parameter is set to 1, which means that progress updates will be printed during training. Finally, the time module is used to measure the execution time of the model training process. This information can be useful for measuring the performance of the model and identifying opportunities for optimization.

Next, to predict the stock prices for a 24-hour period using the trained LSTM model, we iterate through the test data, starting from the lookback time step and ending at the predict time step. The input data for each iteration is created by selecting the current time step and the preceding lookback time steps from the test dataset. The output data is then predicted using the predict method, and then transformed back to their original scale, which reverses the scaling transformation applied to the input data during training.

When evaluating the performance of forecasting models, it is important to choose the appropriate metric to ensure accurate and meaningful results. Most widely used metrics in the field of time series forecasting are Mean Absolute Error, Root Mean Square Error and Mean Absolute Percentage Error.

$$MAE = (1/\ 24 * N_d\ ) \sum_{d\ =\ 1}^{N_d} \sum_{h=1}^{24} \left|p_{d,h} - \hat{p}_{d,h}\right|$$

$$RMSE = \sqrt{(1/24 * N_d) \sum_{d=\ 1}^{N_d} \sum_{h=1}^{24} (p_{d,h} - \hat{p}_{d,h})^2}$$

$$MAPE = (1/(24 * N_d)) \sum_{d=1}^{N_d} \sum_{h=1}^{24} \left|p_{d,h} - \hat{p}_{d,h}\right| / \left|p_{d,h}\right|$$

Where $p_{d,h}$ and $\hat{p}_{d,h}$ respectively represent the real and forecasted price on day d and hour h and N_d is the number of days in the out of sample test period in the test dataset. Since absolute errors are not always accurate to compare between the different datasets where few have negative values. Hence linear metrics do better than quadratic metrics. Also, as values are close to zero MAPE becomes very large and is overall dominated by smaller values, so it might not be as informative. We make use of Symmetric Mean Absolute Percentage Error (SMAPE) based on a statistical distribution with undefined mean and infinite variance.

$$sMAPE = \frac{1}{24 * N_d} \sum_{d=1}^{N_d} \sum_{h=1}^{24} 2 \frac{|p_{d,h} - \hat{p}_{d,h}|}{|p_{d,h}| + |\hat{p}_{d,h}|}$$

A scaled error is simply the MAE scaled by the in-sample MAE of a naive forecast. A scaled error has the nice interpretation of being lower/larger than one if it is better/worse than the average naive forecast evaluated in-sample. A metric based on this concept is the mean absolute scaled error (MASE), and in the context of one-step ahead forecasting is defined as:

$$MASE = \frac{1}{N} \sum_{k=1}^{N} \frac{|p_k - \hat{p}_k|}{\frac{1}{n-1} \sum_{i=2}^{n} |p_i^{in} - p_{i-1}^{in}|}$$

where p_i^{in} is the i^{th}price in the in-sample, i.e the training dataset, p_{i-1}^{in} is the one step ahead naive forecast of p_i^{in} , N is the number of out of sample (test) datapoints, and n the number of in-sample (training) datapoints. As MASE depends on the in-sample dataset, forecasting methods with different calibration windows will naturally have to consider different in-sample datasets. As a result, the MASE of each model will be based on a different scaling factor and comparisons between models cannot be drawn. Drawing comparisons across different time series is problematic as electricity prices are not stationary. For example, an in-sample dataset with spikes and an out-of-sample dataset without spikes will lead to a smaller MASE than if we consider the same market but with the in-sample/out-sample datasets reversed. A better metric is the relative MAE (rMAE). Similar to MASE, it normalizes the error by the MAE of a naive forecast. However, instead of considering the in-sample dataset, the naive forecast is built based on the out-of-sample dataset. For day-ahead electricity prices of hourly frequency, rMAE is defined as

$$rMAE = \frac{\frac{1}{24N_d} \sum_{d=1}^{N_d} \sum_{h=1}^{24} |p_{d,h} - \hat{p}_{d,h}|}{\frac{1}{24N_d} \sum_{d=1}^{N_d} \sum_{h=1}^{24} |p_{d,h} - p_{d,h}^{\widehat{naive}}|}$$

Let us discuss the reliability of these metrics, and highlight the pros and cons of each one.

- MAE is as reliable as rMAE. However, as the errors are not relative, comparison between datasets is not possible and rMAE is preferred.
- sMAPE is more reliable than MAPE and it agrees with MAE/rMAE. Yet, it has the problem of an undefined mean and an infinite variance. Thus, it is less reliable than rMAE.
- MAPE is not a reliable metric as it gives more importance to data points close to zero. As such, using MAPE can lead to misleading results and wrong conclusions.
- RMSE is more reliable than MAPE but it does not correctly represent the underlying risks of EPF. Hence, it should not be used alone to evaluate forecasting models.

Results

We have trained 3 LSTM models, with lookback values of 24,48,96 and compared the models with benchmark LEAR ensemble and DNN ensemble models. In order to evaluate the performance of forecasting models, at various metrics as explained earlier are used and compared across five different electricity markets.

Evaluation for Nord Pool Market	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Mean Absolute Error (MAE):	1.68338	1.73781	3.00999	3.0298	3.0712
Root Mean Square Error (RMSE):	3.31899	3.36215	5.40935	5.38649	5.48304
Mean Absolute Percentage Error (MAPE):	5.38352	5.53269	9.62428	9.66165	9.9503
symmetric Mean Absolute Percentage Error (sMAPE):	4.88031	5.0094	8.74177	8.78101	8.8699
Mean Absolute Scaled Error (MASE):	0.407134	0.420299	0.727981	0.732771	0.742785
relative Mean Absolute Error(rMAE):	0.531727	0.548921	0.950762	0.957017	0.970096

Evaluation for PJM Market	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Mean Absolute Error (MAE):	2.86217	3.01302	4.8409	4.89098	4.76671
Root Mean Square Error (RMSE):	5.04049	5.12747	7.64908	7.68574	7.57592
Mean Absolute Percentage Error (MAPE):	27.4775	30.134	34.1687	34.271	34.4963
symmetric Mean Absolute Percentage Error (sMAPE):	11.3308	11.9798	18.8925	19.1729	18.3251
Mean Absolute Scaled Error (MASE):	0.452412	0.476256	0.765182	0.773099	0.753456
relative Mean Absolute Error(rMAE):	0.586735	0.617658	0.992367	1.00263	0.977159

Evaluation for EPEX-BE Market	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Mean Absolute Error (MAE):	5.86987	6.14006	10.1599	10.0541	10.4042
Root Mean Square Error (RMSE):	15.9663	15.9738	19.9121	20.2203	20.2044
Mean Absolute Percentage Error (MAPE):	24.8922	20.7199	55.5572	52.2514	60.0286
symmetric Mean Absolute Percentage Error (sMAPE):	13.446	14.5463	25.3246	24.7441	25.5921
Mean Absolute Scaled Error (MASE):	0.577772	0.604367	0.972253	0.962131	0.995627
relative Mean Absolute Error(rMAE):	0.713092	0.745916	1.20962	1.19703	1.2387

Evaluation for EPEX-FR Market	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Mean Absolute Error (MAE):	3.67579	3.97977	10.1891	12.698	8.09986
Root Mean Square Error (RMSE):	11.8666	10.6758	20.0805	22.0557	20.7578
Mean Absolute Percentage Error (MAPE):	13.6013	14.6803	44.2287	56.9226	30.4001
symmetric Mean Absolute Percentage Error (sMAPE):	10.8125	11.5664	27.5599	33.1424	23.0269
Mean Absolute Scaled Error (MASE):	0.527279	0.542826	1.38976	1.73196	1.10479
relative Mean Absolute Error(rMAE):	0.650655	0.66984	1.71494	2.13721	1.3633

Evaluation for EPEX-DE Market	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Mean Absolute Error (MAE):	3.41346	3.60909	8.57872	3.31462	11.2848
Root Mean Square Error (RMSE):	5.92721	6.50829	13.3387	4.73194	16.7341
symmetric Mean Absolute Percentage Error (sMAPE):	14.0775	14.7444	30.2115	13.7824	36.3929
Mean Absolute Scaled Error (MASE):	0.373979	0.395412	0.938836	0.362744	1.23499
relative Mean Absolute Error(rMAE):	0.423781	0.448068	1.06412	0.411149	1.39979

In addition to evaluating the accuracy of forecasting models, it is also important to consider the time metric, which measures the time required to produce forecasts. In this comparison, we will analyze the time metric for each of the five forecasting models.

NP Market

Elapsed Time	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Time:	8-20 min	20-25 sec	2.56 min	4.48 min	8.43 min

PJM Market

Elapsed Time	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Time:	8-20 min	20-25 sec	4.97 min	10.25 min	24.99 min

EPEX-BE Market

Elapsed Time	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Time:	8-20 min	20-25 sec	2.22 min	4.52 min	9.57 min

EPEX-FR Market

Elapsed Time	DNN_ensemble	LEAR_ensemble	LSTM-24	LSTM-48	LSTM-96
Time:	8-20 min	20-25 sec	10.27 min	11.91 min	23.32 min

EPEX-DE Market

Elapsed Time	DNN ensemble	LEAR ensemble	LSTM-24	LSTM-48	LSTM-96
Time:	8-20 min	20-25 sec	2.6 min	4.68 min	9.0 min

Visualizing the performance of forecasting models is essential to understand their accuracy in predicting future prices. We've plotted the observed actual electricity price vs the forecasted price using the LSTM models across 4 days.

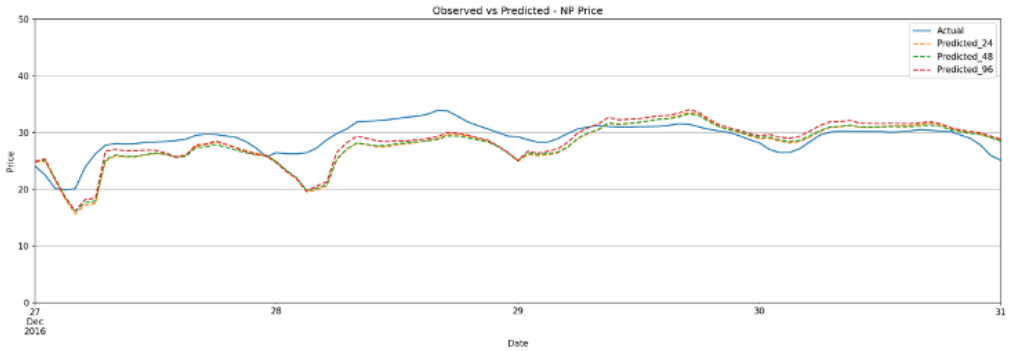


Fig (4)

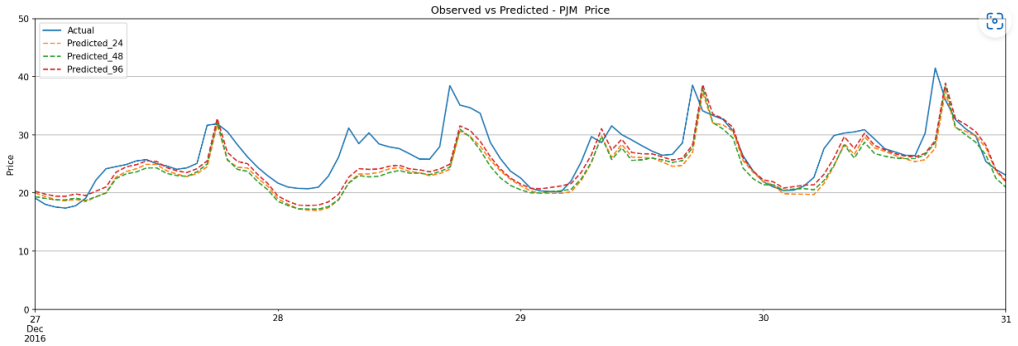


Fig (5)

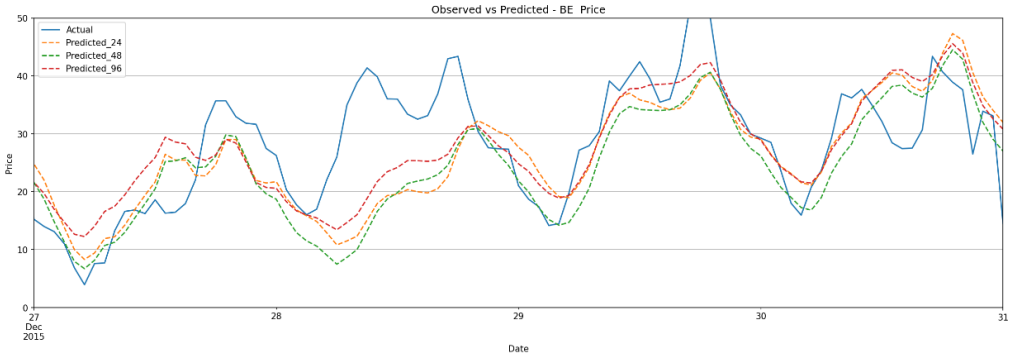


Fig (6)

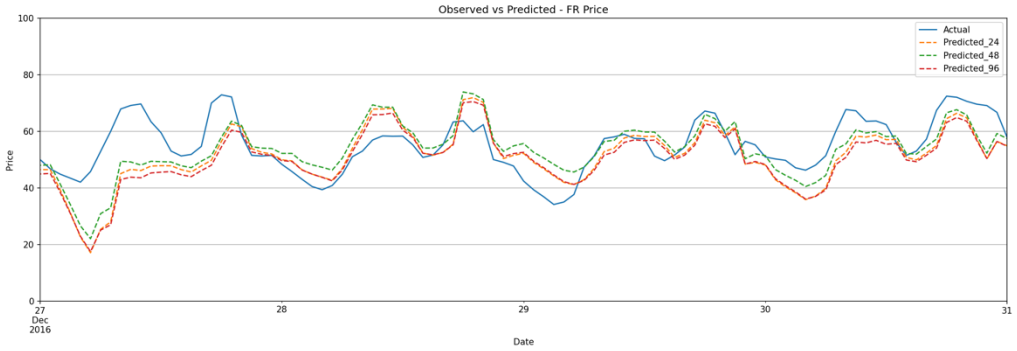


Fig (7)

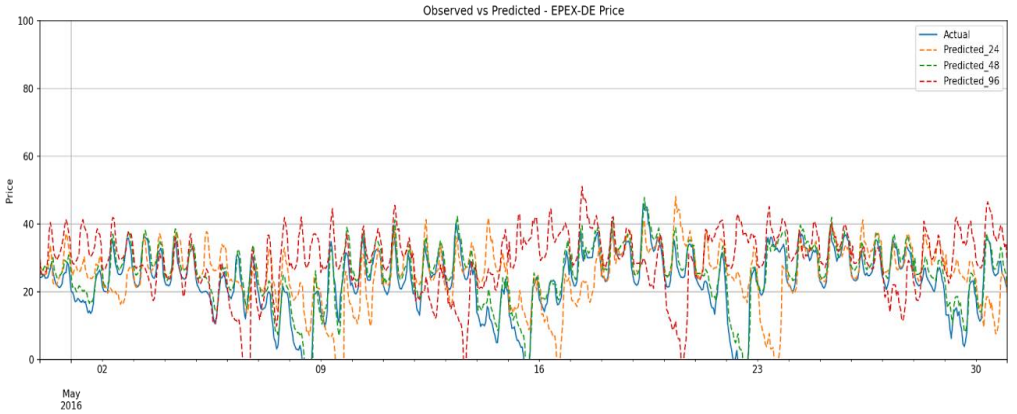


Fig (8)

Conclusion

The development and evaluation of the day-ahead electricity price forecasting model using Long-Short Term Neural Networks has shown promising results. The LSTM-24 model has demonstrated good performance among LSTM models in most markets, but the LEAR ensemble model outperformed all other models in terms of both error and time taken. The findings of this project suggest that the use of deep learning techniques such as LSTM can be effective in forecasting electricity prices. It is worth noting, however, that the performance of these models may vary depending on the particular task and dataset at hand. Therefore, additional testing with different number of input layers, neurons and experimentation may be required to identify the optimal model for a specific application.

Reference

[1] Jesus Lago, Grzegorz Marcjasz, Bart De Schutter, Rafał Weron, Forecasting Day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark, Applied Energy, Volume 293, 2021, 116983, ISSN 0306-2619

[2] Python Code Reference. URL https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

[3] LSTM Material. URL https://d2l.ai/chapter_recurrent-modern/lstm.html

[4] Epftoolbox library. URL https://github.com/jeslago/epftoolbox