ISO Day-Ahead Forecasting Using CNN-LSTM

Ali Camlibel

GitHub: https://github.com/acamlibe/ISO-Price-Forecasting-CNN-LSTM-

Abstract

Independent System Operators (ISO) and Regional Transmission Organizations (RTO) are independent entities that ensure the reliability of the grid across the US. One way they do this is by facilitating the wholesale purchase and sale of power (MWh). Market participants bid to buy and sell power in real-time and in a day-ahead market. For the day-ahead market, bids and offers are entered in a day prior to the trading day. External factors can drastically affect load, and in turn, alter the price. To counter load change, intraday, real-time, trades must occur to keep the grid in balance. Often times, these contingency plans end up being more costly. Due to this price volatility, it is crucial for market participants to accurately forecast prices and hedge against this risk. This report aims to explore various deep neural network (DNN) architectures and proposes a configuration of convolutional neural network (CNN) and long short-term memory (LSTM) layers to forecast prices in different time windows.

INTRODUCTION & MOTIVATION

Forecasting energy prices in the day-ahead market within ISO/RTOs is a topic of great interest to many market participant. Many different types of forecasting models have been used in the past, ranging from statistical models such autoregressive integrated moving averages (ARIMA) and support-vector machines (SVM) to artificial neural network based models such as recurrent neural networks and convolutional neural networks. ARIMA models can struggle against changing environmental conditions over a long period of time. For example, the increased generation of renewable generation. As solar generation replaces conventional coal and gas plants, the price of energy drops. Research has been done on this topic exploring various neural network architectures. The papers mentioned in this report specifically look at how artificial neural networks perform in this domain.

DATA COLLECTION

Hourly price data was collected through the New-England ISO (NE-ISO) API (https://webservices.iso-ne.com/docs/v1.1/) from 2017 through 2024. Only a single locational marginal pricing (LMP) node - MASHPEE.

The hourly weather data was sourced from the National Centers of Environmental Information's (NCEI) quality controlled datasets (https://www.ncei.noaa.gov/access/crn/qcdatasets.html). The weather features considered in this project are dry-bulb temperature, dew point, relative humidity, and cloud cover.

RELATED RESEARCH

Deep learning for day-ahead electricity price forecasting (Chi Zhang, 2020) compared SVM and LSTM models for price forecasting. With the LSTM model proving better performance with an average mean absolute percentage error (MAPE) of 21.04%. Price and weather data was incorporated, but additionally added prior load data (MWh), and gas prices ($/MBTU). The basic principle of the model was to use a recurrent neural network with an LSTM cell unit to more effectively predict sequence data. The paper explored a similar time-horizon of predicting 24 hours ahead using 24 hours of prior data.

Figure 1 shows the hyper-parameters considered in this research's architecture.

Input sequence size	**24**
Output sequence size	**24**
Training batch size	**168**
Validation batch size	**168**
Hidden units	20, **30**, 40, 50, 80
Number of hidden layers	**3**, 4, 5, 6, 7
Epochs	**20**, 50, 100
Optimiser	**RMSPropOptimiser**

Power Market Price Forecasting via Deep Learning (Yongli Zhu, 2018) explored a LSTM architecture to predict prices. This paper also explored different forecasting windows such as 3, 6, 12, and 24 hours prior to predict the next hour. However, this model only explored 1 hour ahead pricing, and used a very small neural network consisting of 1 hidden layer with 4 LSTM cell units.

FEATURE ENGINEERING

The hourly price data often follows a daily and yearly seasonality. Price tends to be higher during the summer and winter months due to cooling and heating requirements in households and businesses. Looking at the price curve for a given day, it generally follows a pattern where price is lowest in the middle of night and highest during the afternoon. The daily seasonality factors are due to residential, commercial, and industrial power demand.

Seasonality Figures

Figure 2 shows yearly seasonality in the price data.

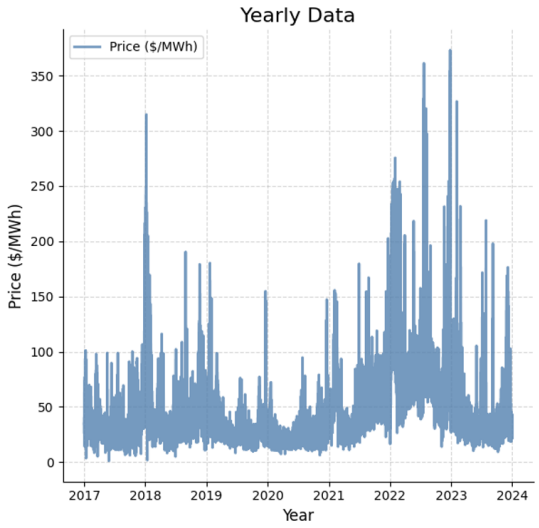
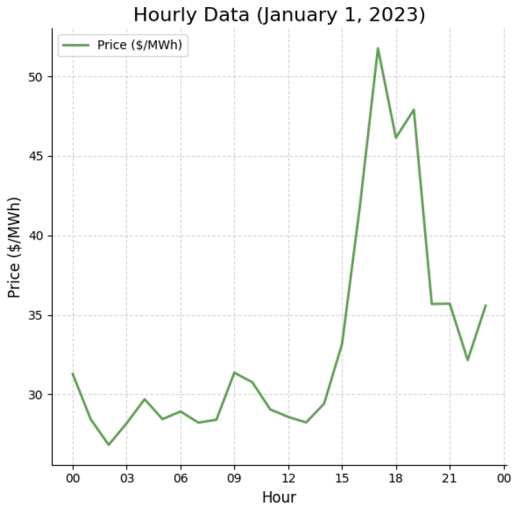


Figure 3 shows daily seasonality in the price data.



To capture this seasonality, feature engineering was performed to calculate sine and cosine values representing the instantaneous point the hour occurred.

```
secs = dates.map(pd.Timestamp.timestamp)

day = 24*60*60
year = (365.2425)*day

day_sin = np.sin(secs * (2 * np.pi / day))
day_cos = np.cos(secs * (2 * np.pi / day))
year_sin = np.sin(secs * (2 * np.pi / year))
year_cos = np.cos(secs * (2 * np.pi / year))
```

Code 1. Python code to calculate seasonality.

Holidays were also considered. For each hour a binary field was set. This was done using the Python package *holidays* (https://pypi.org/project/holidays/).

DATA PREPROCESSING

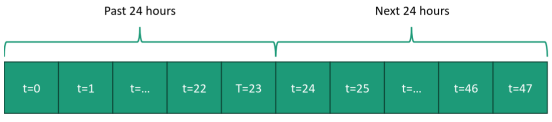
For certain periods, prices could go up to $500/MWh when on average it is $20-$100. Weather can also show dramatic peaks and dips due to inclement weather. Due to the non-stationary nature of price and weather, data was normalized before splitting and training.
The min-max scaling technique was used, defined by the formula below.

$$x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{1}$$

The data contained 64,076 hourly observations. The first 60,000 observations were used for the training set, and the rest were used for testing. 10% of the training set was used for model validation.

WINDOWING DATA

The goal of windowing data is to turn a time-series problem into a supervised learning one. The data is shaped in such a way that each "window" is associated with one or more outputs. For example, the prior 24 time-steps of each step can be associated with 1 time-step into the future.
Figure 4 shows windowing 24 time-steps prior associated with 24 time-steps into the future.



The two window periods explored in this report are using 24 hours prior to predict the next hour and 24 hours prior to predict the next 24 hours. The code below ensures that there is enough of a window to get the start and end of the training and testing datasets by subtracting the number of past observations from the length and adding the number of future steps to the current index.
The function to window the data is defined below.

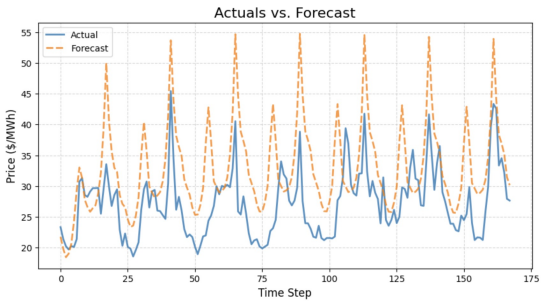
```
def get_window_data(np_arr, n_past, n_future):
    X, y = [], []
    for i in range(n_past, len(np_arr) -
    n_future + 1):
        X.append(np_arr[i - n_past:i])
        y.append(np_arr[i:i + n_future, 0])

    return np.array(X), np.array(y)
```

Code 2. Python code to window data.

SARIMA ARCHITECTURE

The first model was a simple univariate ARIMA model. Training the model was very intensive on CPU and took an extensive amount of time to train on the 7 years of price data. Only price was considered for this model. The main parameters of the SARIMA model (p, d, q) (P,D,Q) were set as (2, 0, 1) (2, 0, 1), using 2 lagged variables, no differencing, and 1 moving average term. The model ended up performing poorly, but proves that price does have temporal dependencies that can be learned.
Figure 5 shows the results of the SARIMA model for one week.



DENSE LAYER ARCHITECTURE

The first attempt at using neural networks was to build a simple feed-forward neural network with Dense layers.
Figure 6 shows the dense model architecture.

Model: "Dense Model"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 24, 10)	21
flatten (Flatten)	(None, 240)	0
dense (Dense)	(None, 64)	15,424
output (Dense)	(None, 1)	65

Total params: 15,510 (60.59 KB)
Trainable params: 15,489 (60.50 KB)
Non-trainable params: 21 (88.00 B)

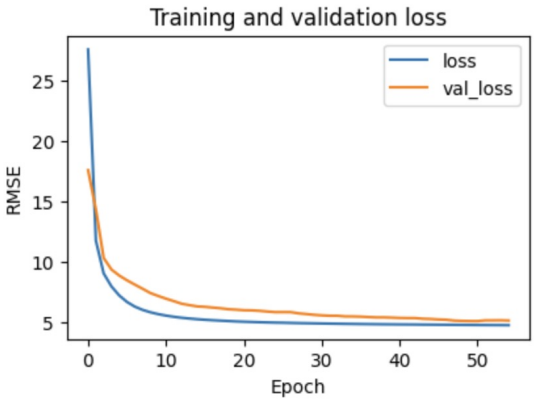
Hyperparameters:

- Neurons: 64
- Activation: ReLU
- Epochs: 100 /w EarlyStopping (patience=4)
- Optimizer: RMSProp
- Loss Function: RMSE
- Batch Size: 168

Two separate models were trained with this architecture. One for 24 hours prior to predict the next hour, and the other using 24 hours prior to predict the next 24 hours.

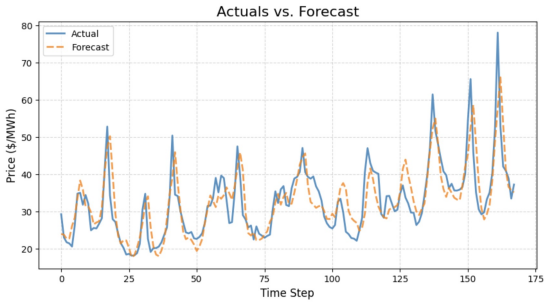
24 hours prior to predict 1 hour ahead

Figure 7 shows the loss curve during training (24 hours prior, 1 hour ahead).



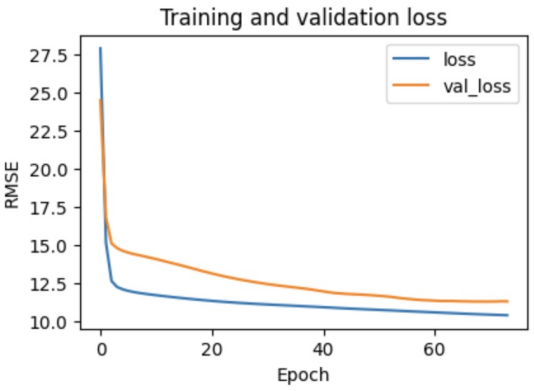
Over the test dataset, an average MAPE of **9.78%**.

Figure 8 shows the results of forecasting 1 week.



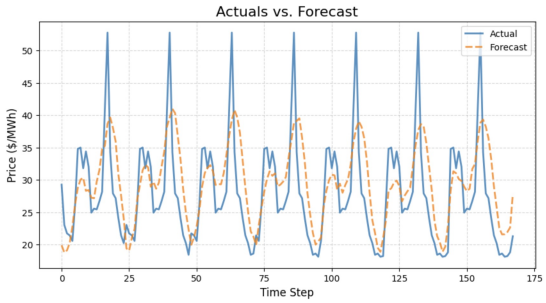
24 hours prior to predict 24 hours ahead

Figure 9 shows the loss curve during training (24 hours prior, 24 hours ahead).



Over the test dataset, an average MAPE of **22.73%**.

Figure 10 shows the results of forecasting 1 week.



This model performs poorly while predicting 24 hours ahead as it struggles to learn the peaks of each day, but it still seems like it's learning something. The supervised learning approach with the windowing function seems to work even on a Dense architecture.

CNN ARCHITECTURE

A CNN architecture which uses a 1-d convolutional layer and max pooling was also considered. The idea behind using CNN for forecasting would be to better learn the inner relationships between the weather and seasonality inputs when it comes to price.
The input and output layers are similar to the dense architecture. Flatten is used after the max pooling layer to reduce the data to a single dimension, so that the dense layer can learn easier.
Figure 11 shows the CNN architecture.

Model: "cnn"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 24, 10)	21
conv1d_1 (Conv1D)	(None, 24, 64)	1,984
maxpool1d_1 (MaxPooling1D)	(None, 12, 64)	0
flatten_1 (Flatten)	(None, 768)	0
dense_1 (Dense)	(None, 32)	24,608
dense_out (Dense)	(None, 1)	33

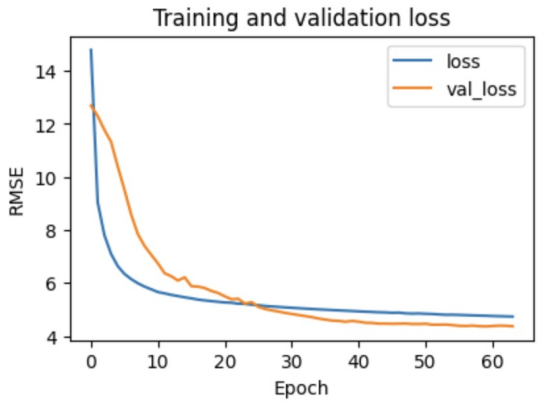
Total params: 26,646 (104.09 KB)
Trainable params: 26,625 (104.00 KB)
Non-trainable params: 21 (88.00 B)

Hyperparameters:

- Filters: 128
- Activation: ReLU
- Epochs: 100 /w EarlyStopping (patience=4)
- Optimizer: RMSProp
- Loss Function: RMSE
- Batch Size: 168
- Kernel Size: 3x3
- Strides: 2
- Padding: same

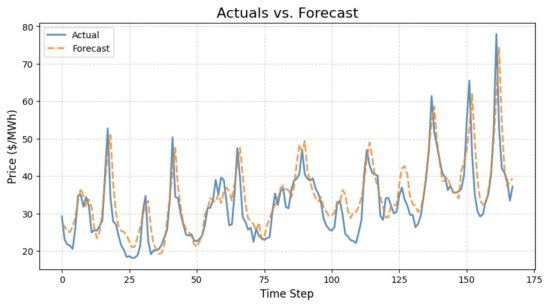
24 hours prior to predict 1 hour ahead

Figure 12 shows the loss curve of training.



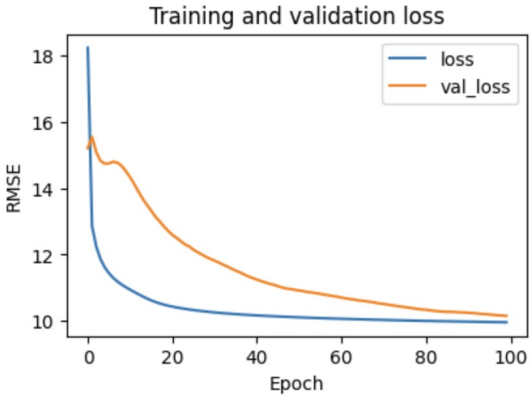
Over the test dataset, an average MAPE of **11.13%**.

Figure 13 shows the results of forecasting a week.



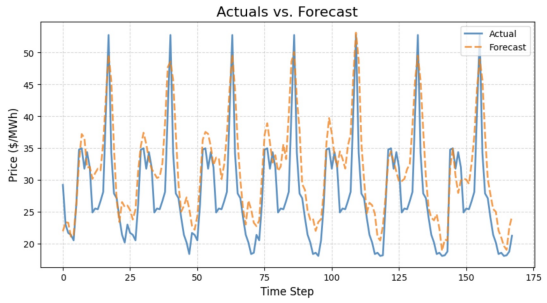
24 hours prior to predict 24 hours ahead

Figure 14 shows the loss curve of training.



Over the test dataset, an average MAPE of **22.88%**.

Figure 15 shows the results of forecasting a week.



Both CNN models seem to be similar to the Dense output, yet the 24 hour-ahead CNN model seems to better capture the peaks of each day compared to the respective dense model.

LSTM ARCHITECTURE

The research in this report was primarily based on using recurrent neural networks (RNN) with LSTM cells to better forecast as there are temporal/sequential dependencies between the price and weather data.

Figure 16 shows the LSTM architecture.

Model: "lstm"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 24, 10)	21
lstm_1 (LSTM)	(None, 24, 64)	19,200
lstm_2 (LSTM)	(None, 24, 64)	33,024
lstm_3 (LSTM)	(None, 32)	12,416
dense_out (Dense)	(None, 1)	33

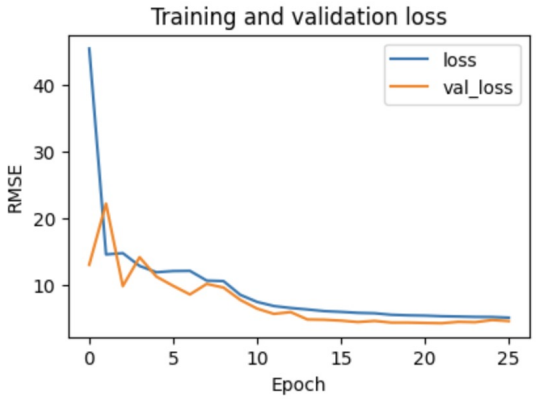
Total params: 64,694 (252.71 KB)
Trainable params: 64,673 (252.63 KB)
Non-trainable params: 21 (88.00 B)

Hyperparameters:

- Activation: ReLU
- Epochs: 100 /w EarlyStopping (patience=4)
- Optimizer: RMSProp
- LSTM cell units: 30
- Loss Function: RMSE
- Batch Size: 168

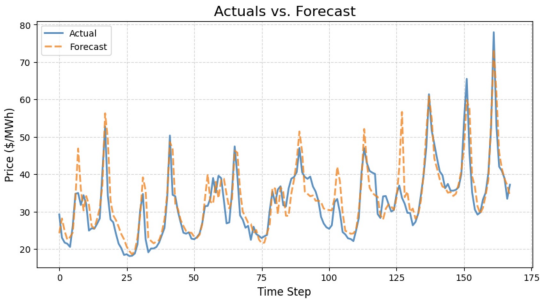
24 hours prior to predict 1 hour ahead

Figure 17 shows the loss curve.



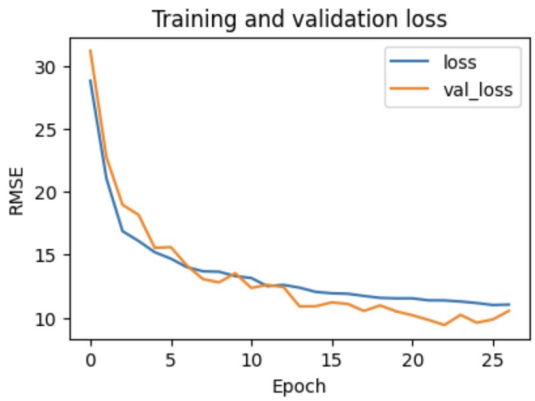
Over the test dataset, an average MAPE of **10.66%**.

Figure 18 shows the forecasting results for 1 week.



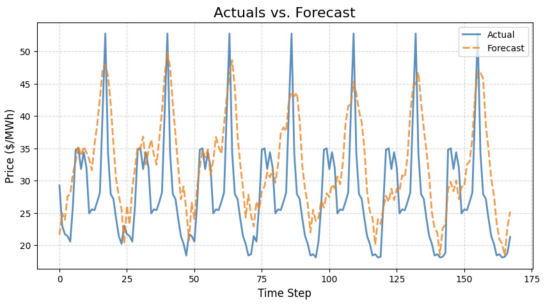
24 hours prior to predict 24 hours ahead

Figure 19 shows the loss curve.



Over the test dataset, an average MAPE of **23.17%**.

Figure 20 shows the forecasting results for 1 week.



The results of the LSTM model do not learn well for the 24 hour-ahead model. It seems to capture the daily peaks, but not the troughs. It's strange that it wouldn't perform as well as at least the dense model. Still performed similar to the previous two models though.

CNN-LSTM ARCHITECTURE

The proposed method in this report is a combination of convolutional layers and LSTM layers. The idea is that the convolutional layers would learn the relationships between the independent variables and then feed the output of those 1-d convolutional layers into LSTM layers to learn from the sequence of the data.
Figure 21 shows the architecture of the CNN-LSTM model.

Model: "cnn-lstm"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 24, 10)	21
conv1d_4 (Conv1D)	(None, 24, 128)	3,968
maxpool1d_1 (MaxPooling1D)	(None, 12, 128)	0
lstm_1 (LSTM)	(None, 12, 60)	45,360
lstm_2 (LSTM)	(None, 60)	29,040
dense_out (Dense)	(None, 24)	1,464

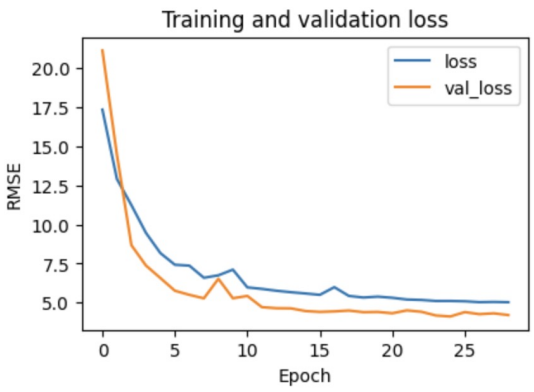
Total params: 79,853 (311.93 KB)
Trainable params: 79,832 (311.84 KB)
Non-trainable params: 21 (88.00 B)

Hyperparameters:

- Activation: ReLU
- LSTM cell units: 60
- Epochs: 100 /w EarlyStopping (patience=4)
- Optimizer: RMSProp
- Loss Function: RMSE
- Bach Size: 168

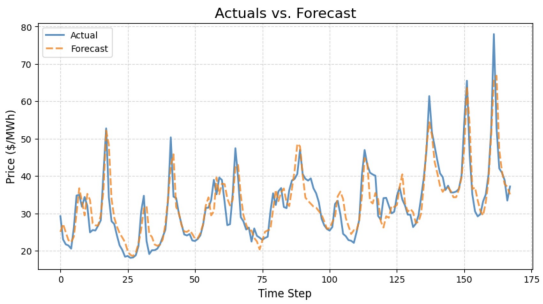
24 hours prior to predict 1 hour ahead

Figure 22 shows the loss curve.



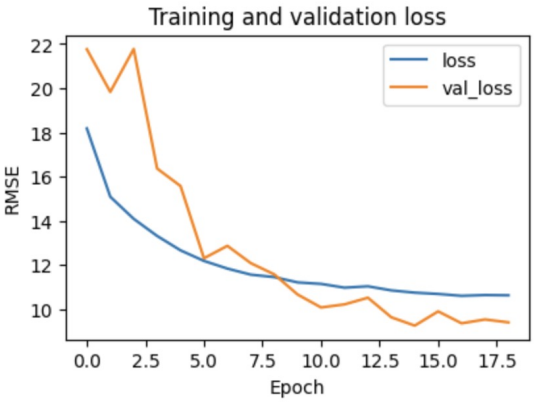
Over the test dataset, an average MAPE of **9.23%**.

Figure 23 shows the forecasting results for 1 week.



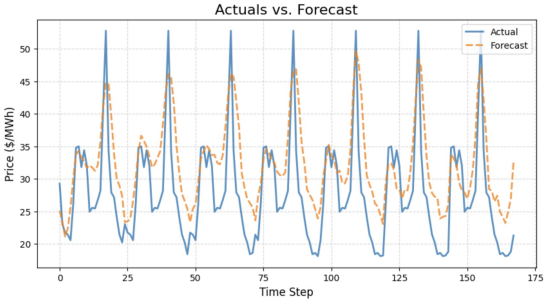
24 hours prior to predict 24 hours ahead

Figure 24 shows the loss curve.



Over the test dataset, an average MAPE of **21.07%**.

Figure 25 shows the forecasting results for 1 week.



HYPER-PARAMETER OPTIMIZATION

The models' hyperparameters were mostly tuned by hand. The number of epochs, batch size, number of neurons, number of units per layer, optimizer, and loss function were all adjusted carefully to get the best results given the lack of data.

Genetic Algorithm

Rather than using Keras' fit method against a model and providing hyperparameters, a Genetic Algorithm (GA) was attempted to generate the model weights. The algorithm starts at an initial population (set of weights) and evolves it over generations. The fitness function is used to determine the next population. The goal is still to minimize the loss function, just in a different way.

Sample code using pyGAD for GA.

```
import pygad.kerasga

keras_ga = pygad.kerasga.KerasGA(model=
    lstm_model,num_solutions=10)

from tensorflow.keras.losses import
    MeanAbsoluteError

def fitness_func(ga_instance, solution, sol_idx)
    :
    global X_train, y_train, keras_ga,
    lstm_model

    predictions = pygad.kerasga.predict(model=
    lstm_model,solution=solution,data=X_train)

    mae = tensorflow.keras.losses.
    MeanAbsoluteError()
    abs_error = mae(data_outputs, predictions).
    numpy() + 0.00000001
    solution_fitness = 1.0/abs_error

    return solution_fitness

def on_generation(ga_instance):
    print(f"Generation = {ga_instance.
    generations_completed}")
    print(f"Fitness    = {ga_instance.
    best_solution()[1]}")

num_generations = 250
num_parents_mating = 5
initial_population = keras_ga.population_weights

ga_instance = pygad.GA(num_generations=
    num_generations,num_parents_mating=
    num_parents_mating,initial_population=
    initial_population,fitness_func=fitness_func
    ,on_generation=on_generation)

ga_instance.run()
```

Code 3. pyGAD with Keras.

ALL RESULTS TABLE

Table 1, shows results for the 24 hour prior, 1 hour ahead models.

Model	Val RMSE	Test MAPE
Dense	5.9	9.78%
CNN	5.4	11.13%
LSTM	5.7	10.66%
CNN-LSTM	4.8	9.23%

Table 2, shows results for the 24 hour prior, 24 hours ahead models.

Model	Val RMSE	Test MAPE
Dense	12.5	22.23%
CNN	10.8	22.88%
LSTM	11.2	23.17%
CNN-LSTM	9.8	21.07%

These models performed decently well. It seems like the combination of CNN-LSTM has potential in the realm of day-ahead price forecasting.

FEEDBACK RESPONSE

The main feedback for this project was that there needed to be more experimentation. I believe that I could have gotten the model better given more time, but given what I had, I think it turned out okay. One aspect that was added after the initial plan was the experimentation with GA. It didn't quite work as well as I had hoped, but it was still a great learning experience.

CONCLUSION

Overall, I would say this project taught me many different concepts when it comes to time-series forecasting. Concepts such as stationary vs. non-stationary, windowing techniques, and different techniques for forecasting a time-series problem. For the 24 hours ahead CNN-LSTM model, I achieved a MAPE of 21%. My goal was 15%, but I am still happy with the results. This is a very hard problem with not many resources online about it. It's a problem that many academic journals have been written about, but in practice, it is still a very difficult problem that even energy utilities and other market participants struggle with. I hope to improve upon the CNN-LSTM architecture and incorporate more data to further improve the results.

Uncomment the command `\tableofcontents` to display it.

Remember that numberless sections will not appear in the ToC, however, you can place them manually with the command `\addcontentsline{toc}{section}{section name}`.

REFERENCES

Y. Zhu, R. Dai, G. Liu, Z. Wang and S. Lu, "Power Market Price Forecasting via Deep Learning," IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, 2018, pp. 4935-4939, doi: 10.1109/IECON.2018.8591581.

Y. -Y. Hong, J. V. Taylar and A. C. Fajardo, "Locational Marginal Price Forecasting Using Deep Learning Network Optimized by Mapping-Based Genetic Algorithm," in IEEE Access, vol. 8, pp. 91975-91988, 2020, doi: 10.1109/ACCESS.2020.2994444.

Zhang, C., Li, R., Shi, H. and Li, F. (2020), Deep learning for day-ahead electricity price forecasting. IET Smart Grid, 3: 462- 469. https://doi.org/10.1049/iet-stg.2019.0258