Financial Market Timely Sequential Forecasting: A Comparative Study On Hybrid CNN-LSTMs

Wenhan Yang[1]

Abstract

Accurately predicting financial market trends is critical for investors and related institutions. With the increasing availability of time-series data, deep learning models have become a popular approach for price prediction. In this study, we explore the effectiveness of one traditional model - Long Short-Term Memory (LSTM) - and two hybrid timely sequential models - CNN-LSTM and Convolutional LSTM (ConvLSTM) - in learning and extracting temporal patterns of the input features, as well as for predicting future prices of given ticker(s). The models are evaluated on multiple datasets to determine their predictive accuracy and efficiency. We use experimental results to demonstrate that the proposed hybrid timely sequential models outperform traditional models and other deep learning models in terms of predicting the returns of given ticker, in the univariate settings. Specifically, our incorporated ConvLSTM-based model achieves the lowest test error compared to other models. Additionally, we perform Grid Search CV and sensitivity analysis to determine the optimal hyper-parameters of the proposed models. Overall, we highlights the promising potential of hybrid timely sequential models for one-dimensional return price prediction, providing a more accurate and efficient solution for predicting the returns.

1. Introduction

Predicting forthcomings based on time-series inputs is a common but crucial task across various domains. Inspiring applications range from weather forecasting in terms of local precipitation intensity[10] (Shi et al. 2015) or hourly air temperature[4] (Hou et al. 2022), civil engineering in terms of travel demand prediction[13] (Wang et al.) and urban expansion[1] (Boulila et al. 2021), healthcare in terms of predicting biological age from physical activity patterns[8] (Rahman et al. 2019) and even video sequence representations of the future and past's encoding through unsupervised learning scheme[11] (Srivastava et al. 2016). The flexibility of sequential model and augmenting structures provides immense adaptability in tackling a wide range of problems. With the ability to analyze various types of data, these models can be customized and fine-tuned to suit specific task requirements.

The financial market is a complex system that generates large amounts of data with a high level of noise and volatility. In traditional time-series forecasting, the primary focus is on capturing the temporal patterns in the data. The main challenge in the financial market is that the data is not only dependent on time but also exhibits spatial dependencies. For example, the price of a Exchange Traded Fund(ETF) or stock may be influenced by the performance of other ETFs or stocks in the same industry, the behavior of the market as a whole, or even global economic trends. Therefore, it is essential to take into account these spatial dependencies to build a more accurate and robust financial market prediction model.

Traditional models, such as linear regression and autoregressive integrated moving average (ARIMA), have struggled to accurately capture the underlying patterns and dynamics of the financial market. Nevertheless, Deep learning models, such as LSTM and CNN, have shown remarkable performance in capturing both temporal and spatial dependencies for predicting short term forthcomings[6]. LSTM is a recurrent neural network architecture that can capture long-term dependencies in sequential data by maintaining a memory state over time. On the other hand, CNN is a feedforward neural network that can learn spatial dependencies in the data by performing convolutional operations. When combined, LSTM and CNN can learn spatiotemporal features simultaneously, making them a suitable approach for time-series prediction tasks.

In addition to spatial dependencies, there is also a distinction between local and global dependencies. Local dependencies are short-term relationships between variables that occur

[1]Fu Foundation of Engineering and Columbia Business School, Columbia University, New York, NY, USA. Correspondence to: Wenhan Yang <wy2403@columbia.edu>.

Proceedings of the CBS GSB Artificial Intelligence, 2023.

within a limited time frame, while global dependencies are long-term relationships that occur over an extended period. LSTM and CNN models can capture both local and global dependencies, which makes them particularly useful for financial market prediction. For example, LSTM can learn the patterns of daily fluctuations, while CNN can capture the broader market trends that occur over several months or years. Overall, deep learning models with LSTM and CNN can help overcome the limitations of traditional time-series forecasting by capturing both temporal and spatial dependencies and provide more accurate and robust predictions.

2. Preliminary

2.1. Problem Statement

Our study aims to develop an accurate and robust predictive framework comparatively that incorporates a range of deep learning-based regression models. To accomplish this, we utilize historical stock data spanning over a period of nine years to construct and assess our proposed models. For our investigation, we have selected a practical prediction horizon of one day. Our hypothesis is that the deep learning models will be able to extract meaningful features from the past stock ticker values and produce accurate forecasts of future values. In this study, we experiment our proposal to include three different deep learning-based regression models. While two of the models are augmentation of CNN and LSTM, the remaining model is built using purely LSTM network architecture as the baseline.

2.2. Convolutions

The operation between the input x and kernel ω

$$s(t) = (x * \omega)(t) = \int_{-\infty}^{\infty} x(\alpha)\omega(t-\alpha)d\alpha = \int_{-\infty}^{\infty} x(t-\alpha)\omega(\alpha)d\alpha$$

that outputs a feature map s is called convolution. It helps improve the neural network by (i) sparse interactions, (ii) parameter sharing, and (iii) equivariant representations. By enforcing multiple filters of given kernel size, the output feature maps can consist of differentiated and specialized spatial representations of inputs.

2.3. Long-short Term Memory

A key feature of LSTM is its memory cell, which is capable of storing information over long periods of time. It is consits of consists of three gates ($\in (0,1)$) at specific time t: the input gate i_t, the forget gate f_t, and the output gate o_t, controlling new information at t in flow into the cell, old information until $t-1$ outflow from cell, and the flow of cell's internal state to the next time step, respectively. Here, c_t is a accumulator of state information until t, i.e. the long-term memory. In contrast, the short-term memory/state of time step t is h_t, and for each t, the input to next cell is the combination of long-term and short-term memory, with amount of each implicitly controlled by gates. In the following formulation, "$\circ$" means Hadamard product:

$$\begin{aligned}
i_t &= \sigma(W_{x_i}x_t + W_{h_i}ht - 1 + W_{c_i} \circ c_{t-1} + b_i)\\
f_t &= \sigma(W_{x_f}x_t + W_{h_f}ht - 1 + W_{c_f} \circ c_{t-1} + b_f)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{x_c}x_t + W_{h_c}h_{t-1} + b_c)\\
o_t &= \sigma(W_{x_o}x_t + W_{h_o}ht - 1 + W_{c_o} \circ c_t + b_o)\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}$$

LSTM has proven capturing long-term dependencies in modeling complex sequential data[3]. They can also process multiple inputs and can be easily stacked to create deeper architectures, which we refer to as FC-LSTM.

2.4. ConvLSTM

ConvLSTM[10] improves from vanilla LSTM in handling spatiotemporal data by encoding spatial information in its input-to-state and state-to-state transitions, using 3D tensors($\mathcal{X}, \mathcal{C}, \mathcal{H}$,gates) whose last two dimensions are spatial dimensions (rows and columns). This enables the model to capture faster motions with a larger transitional kernel and slower motions with a smaller kernel. In the following formulation, "$*$" means convolutions and "$\circ$" means Hadamard product:

$$\begin{aligned}
i_t &= \sigma(W_{x_i} * \mathcal{X}_t + W_{h_i} * \mathcal{H}t - 1 + W_{c_i} \circ \mathcal{C}_{t-1} + b_i)\\
f_t &= \sigma(W_{x_f} * \mathcal{X}_t + W_{h_f} * \mathcal{H}t - 1 + W_{c_f} \circ \mathcal{C}_{t-1} + b_f)\\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{x_c} * \mathcal{X}_t + W_{h_c} * \mathcal{H}_{t-1} + b_c)\\
o_t &= \sigma(W_{x_o} * \mathcal{X}_t + W_{h_o} * \mathcal{X}t - 1 + W_{\mathcal{C}_o} \circ \mathcal{C}_t + b_o)\\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}$$

To visualize the convolution structure within recurrent units, we can zoom in using the picture proposed by Shi et al.:

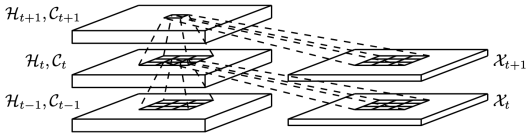


Figure 1. Inner Structure of ConvLSTM[10]

Noticeably, traditional FC-LSTM can be seen as a special case of ConvLSTM, where all features are represented on a single cell and the last two dimensions of its tensors are 1, making it a more flexible and adaptable architecture.

One intricacy is that padding is necessary before performing the convolution operation to ensure that the states match the same number of rows and columns as the

inputs. To achieve this, the hidden states are padded on the boundary points, which can be interpreted as incorporating the state of the external environment in the computation.

2.5. Recurrent Dropout

Recurrent dropout is a regularization technique that is specifically designed for recurrent neural networks, such as LSTMs, to prevent overfitting during training. It randomly drops out some of the units in the recurrent layer, forcing the network to learn more robust representations. In contrast, normal dropout is applied after the recurrent layer, which can lead to inconsistent representations and issues with preserving temporal dependencies. Recurrent dropout has been shown to be effective in improving the performance of RNNs[2][7] without memory loss[9].

3. Methodology

In order to get comparable results, all the following LSTM-based structures are constructed with three recurrent layers.

3.1. FC-LSTM

The FC-LSTM model was chosen as the baseline model due to its effectiveness in modeling sequential data and its ability to handle long-term dependencies. Moreover, the stacking of three LSTM layers to form a block structure allows the model to capture both short-term and long-term temporal patterns in the data. This is achieved by the multiple layers of non-linear transformations that can learn complex representations of the input data. Additionally, the use of fully connected layers in the LSTM cell allows for the incorporation of spatial information, which is particularly important in tasks involving spatiotemporal data such as market price prediction.

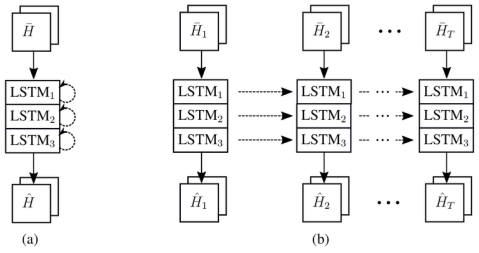


Figure 2. Illustration of a "stacked" LSTM network of depth 3 with recurrent connections[5]

Inputs of the FC-LSTM model is a 3D tensor $(\texttt{batch\_size}, \texttt{window\_size}, \texttt{input\_features})$, since uni-variate this becomes $(\texttt{batch\_size}, \texttt{window\_size}, 1)$. Output of LSTM layers are then passed into a flattening layers for connecting to a dense layer to predict single output.

3.2. CNN-LSTM

CNN-LSTM literally stacks the CNN on top of LSTM so that the strengths of convolutional neural networks and LSTM networks are augmented. The CNN component is designed to extract relevant features from the input data, while the LSTM component is used to learn the temporal dependencies from the representation of data. In this study, a 3-layer CNN followed by a 3-layer LSTM network was used to create a CNN-LSTM architecture.

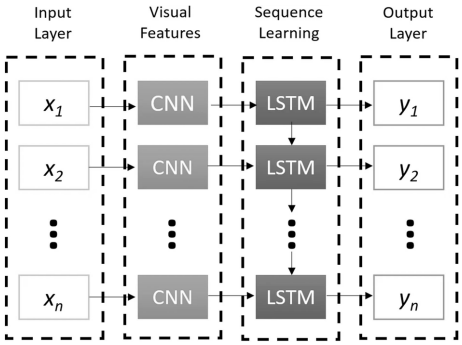


Figure 3. Illustration using CNN and LSTM blocks consecutively in a network [12]

Since the CNN is construed in a time-distributed manner with `TimeDistributed(Conv1D)`, the input to the model should be a 4D tensor $(\texttt{batch\_size}, \texttt{num\_channels}, \texttt{window\_size}, \texttt{input\_features})$, so in the uni-variate single stock settings, the shape becomes $(\texttt{batch\_size}, 1, \texttt{window\_size}, 1)$. To ensure maximized feature leaning, avoiding information loss while ensuring dimensional reduction, only one max-pooling layer after CNN block is applied. Both recurrent dropouts and normal dropout layers are applied within and after each recurrent layer. Output of the final LSTM layer was passed through a dense layer with a single output, representing the predicted value.

3.3. ConvLSTM

As shown in the structure of *Figure 1.*, ConvLSTM mainly differs from CNN-LSTM in terms of feature learning scheme. ConvLSTM incorporates convolutional operations within the LSTM cell, while CNN-LSTM uses a time distributed CNN layer before the LSTM layer. Therefore, spatial features and representations are "refreshed", "reminded" and "updated" through each recurrent unit in ConvLSTM. It is designed to handle spatiotemporal data and is expected to model spatial dependencies in the input data better.

Similarly, the input is also a 4D tensor in $(\texttt{batch\_size}, \texttt{num\_channels}, \texttt{window\_size}, \texttt{input\_features})$ shape. Both recurrent dropouts and normal dropout layers are

applied within and after each `ConvLSTM1D` layer. Output of the final LSTM layer was passed through a dense layer to predict a single output.

4. Experiments

4.1. Datasets

The dataset is dynamically imported from the Tiingo website using Tiingo API key, of stock ticker AAPL Index, from 2010-01-01 to 2016-12-31 (train), 2017-01-01 to 2019-12-31 (validation), 2020-05-01 to 2023-05-01 (test). Note the data from 2020-01-01 to 2020-04-30 was intentionally omitted, due to the overlapping term with the Covid-19 abnormal period as the pandemic had a significant impact on various factors such as consumer behavior, economic activities, and social interactions. Thus, the broad level train-validation-test split is of 6-3-3 years. Uni-variate models are constructed based on "low" values of stock return. Intuitively, this is a valid approach because setting stop-loss orders or limit orders based on predicted low prices can help investors protect their investments and manage their risk. These orders can be used to automatically sell shares if the stock price drops below a certain level, which can help investors avoid significant losses. In this way, predicting the low price of a stock can be a useful benchmark for setting risk management strategies in portfolio selection.

4.2. Pre-processing and Post-processing

Regarding this experiment, we are only using "low" price to fit and cross-compare uni-variate models. Function

$$\texttt{window\_data}(\texttt{df}, \texttt{window\_size})$$

is a *generator* function that slices time-series data into given window size (*lookback*) with fixed step size of 1, while computing within-window percentage change of low price inputs and targets are first calculated, i.e., subtracting then dividing the 1st day stock price from each 2nd to t+1 day price, so that the dynamics of data can be best captured within time frame and help to reduce the impact of trend and seasonality, thus stationary. Same procedure is also applied to target y (*delay*). Input 4D tensors' dimensions are specified in Appendix A.1, A.2. Now the $\texttt{t} : \texttt{t} + \texttt{window\_size}$ is used to predict $\texttt{t} + \texttt{window\_size} + 1$, the next-day-low-percentage-change-from-window-start. After getting the trained models, the function

$$\texttt{plot\_stock\_prediction}(\texttt{window}, \texttt{model}, \texttt{X\_test})$$

use given model to predict day-by-day percentage change in low price from window start, and we times then plus starting low value to get the recovery of values in normal scale of dollars. Finally, both true values and predicted values are visualized in the same canvas. MSE, MAE in raw, dollars and R^2 are reported.

4.3. Experimental Details

Depending on different window size, 2 weeks (14 days) or 1 month (30 days), separate trials are performed to discover the relationship between window size and the ability of regression model to predict next-day-percentage-return. Experiments were run on MacOS with 1.4 GHz Intel Core i5 processor.

Loss Function $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$ is used during optimization with Adam optimizer because of its robustness to outliers. This is gains more agility in the situation when MSE is small in scale and insensitive to small change.

Early Stopping Due to that neural networks could potentially overfit epochs, the number of epochs need to be carefully selected. Regarding computational feasibility, Early Stopping with a patience of 5 is used for both Grid Search CV in hyper-parameter tuning and training the fine tuned model to reduce polynomial time by power 1.

Padding Mentioned in section 2.4, padding is needed for ConvLSTM to achieve state matching. Therefore, argument `padding` is set to "`same`" in `ConVLSTM1D` layers.

Batch Size The batch size is fixed at the value 32, which is a relatively small size comparing to the size of data input($\approx$ 2000), aiming for better generalization performance.

Hyper-parameter Tuning Using with 3-fold Grid Search CV, the "best" set of parameters are selected in a enumerated manner (note: value in "()" is for 30 days window, * indicates baseline model):

Params	**Values**	**Selected Values**
Recurrent units	32, 64, 128	128 (128)
Dropout rate	0.3, 0.5	0.3 (0.3)
Recurrent dropout	0.3, 0.5	0.3 (0.3)

Table 1. Tuning for FC-LSTM

Params	**Values**	**Selected Values**
Recurrent units	32, 64, 128	128 (64)
Dropout rate	0.3, 0.5	0.3 (0.3)
Recurrent dropout	0.3, 0.5	0.5 (0.3)
CNN unites	[32,64,128], [64,64,128]	[32,64,128] ([64,64,128])
Kernel size	3, 5	3 (3)

Table 2. Tuning for CNN-LSTM

Params	Values	Selected Values
Dropout rate	0.3, 0.5	0.3 (0.5)
Recurrent dropout	0.3, 0.5	0.5 (0.3)
CNN units	[128,64,32], [32,64,128], [64,64,64]	[128,64,32] ([64,64, 64])
Kernel size	3, 5	5 (5)

Table 3. Tuning for ConvLSTM

Note the choice of CNN units indicates the test and error of whether "funnel", "broadcasted", or "uniform" structure would works the best on generalizing input features. For FC-LSTM and CNN-LSTM's pure LSTM block, we gently assume uniformly stacking LSTM units would be an optimal choice without loss of generality. It also follows naturally that the selection of "broardcasted" structure in the CNN block would be convincing, as this allows hierarchical learning that learning non-trivial intricacy out of financial data for better future inference.

4.4. Results and Analysis

Model	**MAE(raw)**	**MAE($)**	R^2
FC-LSTM*	0.0241	4.94	0.874
CNN-LSTM	0.0160	3.04	0.914
ConvLSTM	0.0157	3.11	0.917

Table 4. Result table: window = 14; variation depends on trials

Model	**MAE(raw)**	**MAE($)**	R^2
FC-LSTM*	0.0279	6.41	0.909
CNN-LSTM	0.0218	4.89	0.938
ConvLSTM	0.0188	4.00	0.943

Table 5. Result table: window = 30; variation depends on trials

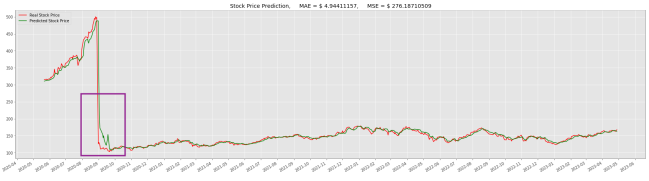
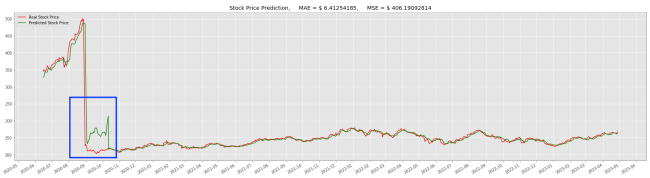


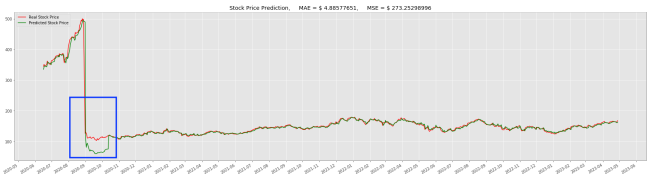
Figure 4. FC-LSTM gradient decrease, window = 14

Uni-variate Forecasting The result tables indicate the trend of improvement in both MAE and R^2, suggesting better predictability and explainability as the model architecture complicates. ConvLSTM achieved lowest MAE and highest R^2 in either trials. From the visualizations in *Figure 4.*, we can see that LSTM didn't get good assimilation at the "cliffs" and "foot" of the cliffs as the structure have relatively weaker ability in generalizing as the long-term dependencies overweight the detection of local patterns such that the sudden hit-bottom-and-stabilize is smoothed and modeled as gradient decrease. But with CNN, the spatial-temporal dependencies are captured, making the last two models perform better than the baseline.

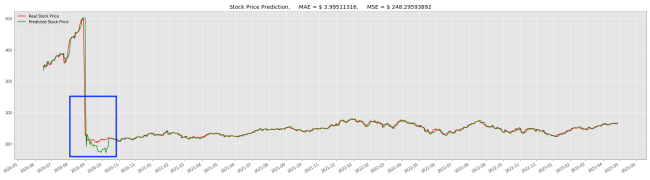
The two hybrid models, given different structures, landed on differentiable results. CNN-LSTM learns representations first, then recurs this information through context cell for prediction, while ConvLSTM regenerates feature maps in each iteration along side memory recursion. ConvLSTM allows repetitive reminding of important input features and new representations such that undetectable patterns are not "dilluted" throughout deep networks, making it outperforms CNN-LSTM. This effect is amplified when input window = 30 - the gap between MAE = 0.0218 and 0.0188 is more obvious than 0.0160 and 0.0157 - due to that ConvLSTM's ability to capture spatial-temporal dependencies is more apparent for longer sequence input. Additionally, longer input sequences can also exacerbate the issue of vanishing gradients in CNN-LSTM models, which can limit their ability to capture long-term dependencies and contribute to their relatively poorer performance compared to ConvLSTM models.



(a) FC-LSTM



(b) CNN-LSTM



(c) ConvLSTM

Figure 5. Cross comparing test set prediction results, window = 30

Sensitivity Analysis We investigated the impact of window size on model performance. As shown in *Table 5.*, we observed that for longer input sequences (e.g., 30), initially

led to a degradation in model performance, proven by increased level of MAE. However, R^2 has been improved with the input length because longer inputs contain more repeating short-term patterns, meaning that model can potentially capture more complex temporal dependencies and patterns in the data. This relationship is not always straightforward – there may be a point of diminishing returns where longer input sequences do not necessarily lead to improved prediction accuracy.

In the visualization shown by *Figure 5.*, all models have poorer approximation of the "foot" around 2020-09 to 2020-10, comparing to the modeling using 14-day window frame. The LSTM is being optimistic around the "foot", as previous window period showed averagely high prices, and mimicking the growing before 2020-08; however, the with CNN modules added, CNN-LSTM and ConvLSTM are quite pessimistic as the convolutions condenses features that reports abnormality of patterns within window. Yet, due to the broad period of input, such generalization cannot be made accurately. Nevertheless, regarding reporting alerts, being pessimistic may be more helpful because it may help investors to anticipate potential downside risks and protect their investments.

Computation Efficiency The training period of each model is timed and displayed in *Table 6*. The ConvLSTM model provided the best accuracy among all the models we tested, but at the cost of increased time complexity from overparameterized network and increasing training time. Noticeably, CNN-LSTM model can be trained quickly with only 1/3 cost of time, and it is stably efficient across different length of inputs. Besides, the training and cross-validation time for ConvLSTM and LSTM was significantly longer than that of the CNN-LSTM model. Therefore, the choice of model depends on the specific requirements of the problem at hand. If accuracy is the main focus and the computational resources are available, the ConvLSTM model is the best choice. However, if time complexity is a limiting factor, then the CNN-LSTM model is a better option.

Model	**Parameters**	**Training Time (s)**
FC-LSTM*	331,521 (333,569)	94.8 (300.8)
CNN-LSTM	622,529 (513,281)	35.6 (33.8)
ConvLSTM	638,785 (413,569)	161.2 (184.1)

Table 6. Model training times; variation depends on trials

5. Future Works

There are several directions for future work that can further improve the proposed model. Firstly, the model can be extended to handle multi-channeled input to achieve a "one-model-fit-many" approach. Specifically, the model can be trained to simultaneously intake a wide range of stock tickers' data to generate predictions for each of them, and capturing the spatial-temporal representation among different stocks. To handle this, a computationally effective environment can be used to process the multiple channels of data.

In addition, while within-window percentage change in low price is commonly used as a feature in finance prediction tasks, there are other features that can be used to potentially improve the model's performance. Future work can explore incorporating multiple other financial features such as volume, technical indicators, and market sentiment indicators into the model's input. By doing so, the model can potentially capture more complex relationships and dependencies among different factors in the financial market, leading to better prediction performance.

6. Conclusion

While our study has shown the potential of hybrid timely sequential forecasting models for financial market prediction, it is important to consider the practical trade-offs of implementing these models in a financial setting. One major consideration is the computational cost, which can be significant for models that involve multiple layers and large datasets. The promising side is that the potential gain in accuracy and the resulting reduction in MAE in dollars may outweigh the computational cost. Additionally, it is important to note that these models rely heavily on the quality and relevance of the input data, as well as the selection of appropriate hyper-parameters. Further research could investigate the optimal window sizes and other hyper-parameters for these models, as well as the potential benefits of incorporating additional features beyond percentage change in adjusted close. Overall, CNN-LSTM-based hybrid timely sequential forecasting models offer a promising avenue for financial market prediction, but their practical implementation must be carefully considered and evaluated.

References

[1] Wadii Boulila et al. "A Novel CNN-LSTM-based Approach to Predict Urban Expansion". In: *CoRR* abs/2103.01695 (2021). arXiv: 2103.01695. URL: https://arxiv.org/abs/2103.01695.

[2] Yarin Gal and Zoubin Ghahramani. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*. 2016. arXiv: 1512.05287 [stat.ML].

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[4] Jingwei Hou et al. "Prediction of hourly air temperature based on CNN–LSTM". In: *Geomatics, Natural Hazards and Risk* 13.1 (2022), pp. 1962–1986. DOI: 10.1080/19475705.2022.2102942. eprint: https://doi.org/10.1080/19475705.2022.2102942. URL: https://doi.org/10.1080/19475705.2022.2102942.

[5] Zhenyu Liu, Mason del Rosario, and Zhi Ding. "A Markovian Model-Driven Deep Learning Framework for Massive MIMO CSI Feedback". In: *IEEE Transactions on Wireless Communications* 21.2 (2022), pp. 1214–1228. DOI: 10.1109/TWC.2021.3103120.

[6] Sidra Mehtab and Jaydip Sen. "Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models". In: *2020 International Conference on Decision Aid Sciences and Application (DASA)*. IEEE, Nov. 2020. DOI: 10.1109/dasa51403.2020.9317207. URL: https://doi.org/10.1109%5C%2Fdasa51403.2020.9317207.

[7] Taesup Moon et al. "RNNDROP: A novel dropout for RNNS in ASR". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 2015, pp. 65–70. DOI: 10.1109/ASRU.2015.7404775.

[8] Syed Ashiqur Rahman and Donald A. Adjeroh. "Deep Learning using Convolutional LSTM estimates Biological Age from Physical Activity". In: *sci* Rep 9, 11425 (2019). DOI: https://doi.org/10.1038/s41598-019-46850-0.

[9] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. "Recurrent Dropout without Memory Loss". In: *CoRR* abs/1603.05118 (2016). arXiv: 1603.05118. URL: http://arxiv.org/abs/1603.05118.

[10] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *CoRR* abs/1506.04214 (2015). arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

[11] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. "Unsupervised Learning of Video Representations using LSTMs". In: *CoRR* abs/1502.04681 (2015). arXiv: 1502.04681. URL: http://arxiv.org/abs/1502.04681.

[12] Abdulkadir Tasdelen and Baha Sen. "A hybrid CNN-LSTM model for pre-miRNA classification". In: *Sci Rep* 11, 14125 (2021). URL: https://doi.org/10.1038/s41598-021-93656-0.

[13] Dongjie Wang, Yan Yang, and Shangming Ning. "DeepSTCL: A Deep Spatio-temporal ConvLSTM for Travel Demand Prediction". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489530.

Appendix

Appendix A. Reproducibility

The code for this project was written in Python and utilized popular libraries such as Tensorflow, Keras, Pandas, and Scikit-learn. The code is publicly available on Google Colab Folder (LionMail access required) to ensure reproducibility of the results. Tiingo API key is required for fetching the data, which is also included in the folder.

- Folder content:
  - *stock_market_AAPL_window=14.ipynb*: code and last session's results with 14-day window
  - *stock_market_AAPL_window=30.ipynb*: code and last session's results with 30-day window
- Guide to run on Colab: As obtaining Tiingo API, save it to a file "tiingo" and upload it to "session storage" on the left hand side of notebook for each runtime. Then, click on code block "start" button or select Runtime→ Run all from menu bar to run the code.
- Flexibility: this research paper demonstrate the feasibility in methodologies. It is possible for the model to be reconstruct on any given stock ticker data, or time-series data, in similar means. To demonstrate on other stocks, change all the "AAPL" in the second code block to desired ticker code.

Appendix A.1 Model, Window = 14 Days

0. Imports

[ ]:
```
# read in data from tiingo
from pandas_datareader import data as pdr
# basic dataframe and array manipulation
import numpy as np
import pandas as pd
from datetime import datetime
# data visualization
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")
# keras sequential model pipeline
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, MaxPooling1D, MaxPooling2D, Flatten,␣
 ↪Dense, Dropout
from tensorflow.keras.layers import LSTM, ConvLSTM1D, TimeDistributed
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import r2_score
from tensorflow.keras.utils import plot_model
# for GridSearch CV
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
# for timing training
import timeit
# plotting results: formating x axis date
import matplotlib.dates as mdates
```

[ ]:
```
# get spy data from tiingo
with open('credentials/tiingo','r') as f:
    TIINGO_API_KEY = f.read().strip()
# get train data, 6 year
start=datetime(2010, 1, 1)  # start day
end=datetime(2016, 12, 31)  # end day
train_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
# get test data, 3 year, skipping covid boom period (2020 spring)
start=datetime(2017, 1, 1)  # start day
end=datetime(2019, 12, 31)  # end day
val_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
# get test data, 3 year, skipping covid boom period (2020 spring)
start=datetime(2020, 5, 1)  # start day
end=datetime(2023, 5, 1)  # end day
test_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
```

Appendix A.1 Model, Window = 14 Days

1. Data Preprocessing

[ ]:
```
window_size = 14
def window_data(df, window_size):
    X = []
    y = []
    for i in range(1 , len(df) - window_size -1 , 1):
        first = df.iloc[i,2]
        temp = []
        temp2 = []
        for j in range(window_size):
            temp.append((df.iloc[i + j, 2] - first) / first)
        temp2.append((df.iloc[i + window_size, 2] - first) / first)
        X.append(np.array(temp).reshape(window_size, 1))
        y.append(np.array(temp2).reshape(1, 1))
    return X, y

X_train, y_train = window_data(train_data, window_size)
X_val, y_val = window_data(val_data, window_size)
X_test, y_test = window_data(test_data, window_size)

X_train = np.array(X_train)
X_val = np.array(X_val)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_val = np.array(y_val)
y_test = np.array(y_test)

X_train = X_train.reshape(X_train.shape[0],1,window_size,1)
X_val = X_val.reshape(X_val.shape[0],1,window_size,1)
X_test = X_test.reshape(X_test.shape[0],1,window_size,1)
```

[ ]:
```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(1746, 1, 14, 1)
(738, 1, 14, 1)
(739, 1, 14, 1)
```

2. Model construction

2.a Baseline model: FC-LSTM

2.a.(i) Hyperparameter Tuning

[ ]:
```
def create_fc_lstm(dropout_rate=0.5, recurrent_dropout=0.5, lstm_units = 64):
    model = tf.keras.Sequential()
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
 ↪return_sequences=True,
                   input_shape=(window_size, 1)))
    model.add(Dropout(dropout_rate))
```

Appendix A.1 Model, Window = 14 Days

```
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,activation='relu',␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,activation='relu',␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'lstm_units': [32, 64, 128],
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
tf.keras.backend.clear_session()
model = KerasRegressor(build_fn=create_fc_lstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train.reshape((len(X_train), X_train.shape[2], X_train.
↪shape[3])),y_train,
                       validation_data=(X_val.reshape((len(X_val), X_val.shape[2],␣
↪X_val.shape[3])),y_val))
fc_lstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24: DeprecationWarning:
KerasRegressor is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras)
instead. See https://www.adriangb.com/scikeras/stable/migration.html for help migrating.

Best params: {'dropout_rate': 0.3, 'lstm_units': 128, 'recurrent_dropout': 0.3}
```

2.a.(ii) Training the model

[ ]:
```
tf.keras.backend.clear_session()
fc_lstm = create_fc_lstm(dropout_rate=fc_lstm_best_params['dropout_rate'],
                         recurrent_dropout=fc_lstm_best_params['recurrent_dropout'],
                         lstm_units = fc_lstm_best_params['lstm_units'])
fc_lstm._name = "FC-LSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
fc_lstm.summary()
```

```
Model: "FC-LSTM"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

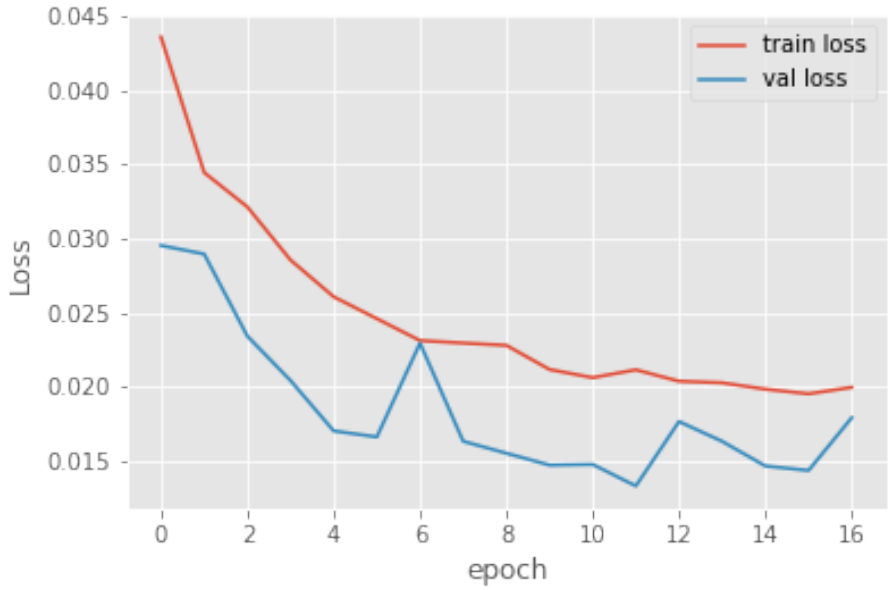
Appendix A.1 Model, Window = 14 Days

```
lstm (LSTM)                 (None, 14, 128)           66560

dropout (Dropout)           (None, 14, 128)           0

lstm_1 (LSTM)               (None, 14, 128)           131584

dropout_1 (Dropout)         (None, 14, 128)           0

lstm_2 (LSTM)               (None, 14, 128)           131584

dropout_2 (Dropout)         (None, 14, 128)           0

flatten (Flatten)           (None, 1792)              0

dense (Dense)               (None, 1)                 1793

=================================================================
Total params: 331,521
Trainable params: 331,521
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
fc_lstm_history = fc_lstm.fit(X_train.reshape((len(X_train), X_train.shape[2], X_train.
 ↪shape[3])),y_train,
                              validation_data=(X_val.reshape((len(X_val), X_val.
 ↪shape[2], X_val.shape[3])),y_val),
                                                epochs=100,batch_size=32, verbose=0,␣
 ↪callbacks=[early_stopping])
fc_lstm_elapsed = timeit.default_timer() - start_time
plt.plot(fc_lstm_history.history['loss'], label='train loss')
plt.plot(fc_lstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x15e0713c8>
```

Appendix A.1 Model, Window = 14 Days



2.b CNN-LSTM

2.b.(i) Hyperparameter tuning

[ ]:
```
# Use Grid Search to Tune drop out
def create_cnn_lstm(dropout_rate=0.5, recurrent_dropout=0.5, cnn_units=[32, 64, 128],
                    lstm_units = 64, kernel_size=5):
    model = tf.keras.Sequential()
    # CNN layers
    model.add(TimeDistributed(Conv1D(cnn_units[0], kernel_size=kernel_size,
                                     activation='relu', input_shape=(None,␣
↪window_size, 1))))
    model.add(TimeDistributed(Conv1D(cnn_units[1], kernel_size=kernel_size,␣
↪activation='relu')))
    model.add(TimeDistributed(Conv1D(cnn_units[2], kernel_size=kernel_size,␣
↪activation='relu')))
    model.add(TimeDistributed(MaxPooling1D(2)))
    model.add(TimeDistributed(Flatten()))
    # cnn_lstm.add(Dense(5, kernel_regularizer=L2(0.01)))
    # LSTM layers
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=False))
    model.add(Dropout(dropout_rate))
```

Appendix A.1 Model, Window = 14 Days

```
    # FC layers, output
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'cnn_units': [[32, 64, 128], [64, 64, 128]],
    'lstm_units': [32, 64, 128],
    'kernel_size': [3, 5]
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
tf.keras.backend.clear_session()
model = KerasRegressor(build_fn=create_cnn_lstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train, y_train, validation_data=(X_val, y_val))
cnn_lstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
<ipython-input-12-be7f2ccfb87d>:35: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor(build_fn=create_cnn_lstm, epochs=100, batch_size=32,

Best params: {'cnn_units': [32, 64, 128], 'dropout_rate': 0.3, 'kernel_size': 3,
'lstm_units': 128, 'recurrent_dropout': 0.5}
```

2.b.(ii) Training the model

[ ]:
```
tf.keras.backend.clear_session()
cnn_lstm = create_cnn_lstm(dropout_rate=cnn_lstm_best_params['dropout_rate'],
                           recurrent_dropout=cnn_lstm_best_params['recurrent_dropout'],
                           cnn_units=cnn_lstm_best_params['cnn_units'],
                           lstm_units=cnn_lstm_best_params['lstm_units'],
                           kernel_size=cnn_lstm_best_params['kernel_size'])
cnn_lstm._name = "CNN-LSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
cnn_lstm.build(input_shape=(len(X_train),None, window_size, 1))
cnn_lstm.summary()
```

```
Model: "CNN-LSTM"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (1746, None, 12, 32)     128
 ibuted)
```

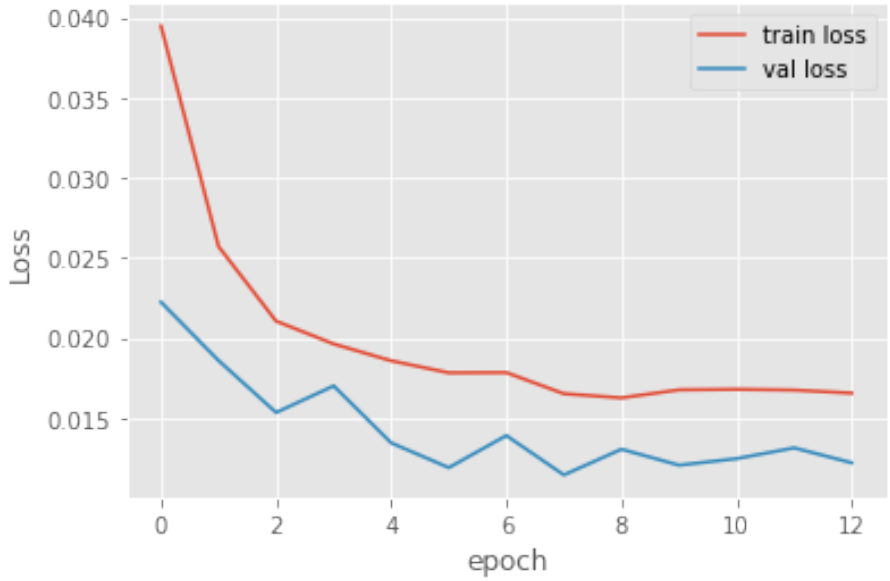
Appendix A.1 Model, Window = 14 Days

```
time_distributed_1 (TimeDis  (1746, None, 10, 64)     6208
tributed)

time_distributed_2 (TimeDis  (1746, None, 8, 128)     24704
tributed)

time_distributed_3 (TimeDis  (1746, None, 4, 128)     0
tributed)

time_distributed_4 (TimeDis  (1746, None, 512)        0
tributed)

lstm (LSTM)                  (1746, None, 128)        328192

dropout (Dropout)            (1746, None, 128)        0

lstm_1 (LSTM)                (1746, None, 128)        131584

dropout_1 (Dropout)          (1746, None, 128)        0

lstm_2 (LSTM)                (1746, 128)              131584

dropout_2 (Dropout)          (1746, 128)              0

dense (Dense)                (1746, 1)                129

=================================================================
Total params: 622,529
Trainable params: 622,529
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
cnn_lstm_history = cnn_lstm.fit(X_train, y_train, validation_data=(X_val,y_val),
                    epochs=100,batch_size=32, verbose=0, callbacks=[early_stopping])
cnn_lstm_elapsed = timeit.default_timer() - start_time
plt.plot(cnn_lstm_history.history['loss'], label='train loss')
plt.plot(cnn_lstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x159512780>
```

Appendix A.1 Model, Window = 14 Days



2.c ConVLSTM

2.c.(i) Hyperparameter tuning

[ ]:
```
def create_convlstm(dropout_rate=0.5, recurrent_dropout=0.5, units=[64,64,128],␣
↪kernel_size=5):
    model = tf.keras.Sequential()
    model.add(ConvLSTM1D(filters=units[0], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", input_shape=(None, window_size, 1),␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[1], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[2], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=False))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'units': [[128, 64, 32],[32, 64, 128],[64, 64, 64]],
```

Appendix A.1 Model, Window = 14 Days

```
    'kernel_size': [3, 5]
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
model = KerasRegressor(build_fn=create_convlstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train, y_train, validation_data=(X_val, y_val))
convlstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
<ipython-input-15-e965e00549af>:25: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor(build_fn=create_convlstm, epochs=100, batch_size=32,

Best params: {'dropout_rate': 0.3, 'kernel_size': 5, 'recurrent_dropout': 0.5, 'units':
[128, 64, 32]}
```

2.c.(ii) Training the model

[ ]:
```
tf.keras.backend.clear_session()
convlstm = create_convlstm(dropout_rate=convlstm_best_params['dropout_rate'],
                           recurrent_dropout=convlstm_best_params['recurrent_dropout'],
                           units=convlstm_best_params['units'],
                           kernel_size=convlstm_best_params['kernel_size'])
convlstm._name = "ConVLSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
convlstm.build(input_shape=(len(X_train),None, 30, 1))
convlstm.summary()
```

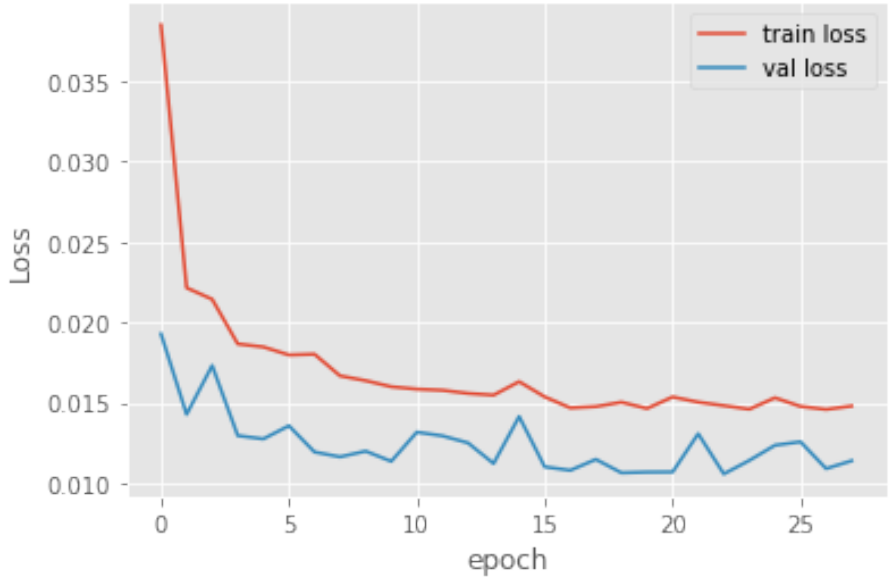
```
Model: "ConVLSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm1d (ConvLSTM1D)    (None, None, 14, 128)     330752

 dropout (Dropout)           (None, None, 14, 128)     0

 conv_lstm1d_1 (ConvLSTM1D)  (None, None, 14, 64)      246016

 dropout_1 (Dropout)         (None, None, 14, 64)      0

 conv_lstm1d_2 (ConvLSTM1D)  (None, 14, 32)            61568

 dropout_2 (Dropout)         (None, 14, 32)            0

 flatten (Flatten)           (None, 448)               0
```

Appendix A.1 Model, Window = 14 Days

```
 dense (Dense)               (None, 1)                 449

=================================================================
Total params: 638,785
Trainable params: 638,785
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
convlstm_history = convlstm.fit(X_train, y_train,␣
 ↪validation_data=(X_val,y_val),epochs=100,
                                batch_size=32, verbose=0, callbacks=[early_stopping])
convlstm_elapsed = timeit.default_timer() - start_time
plt.plot(convlstm_history.history['loss'], label='train loss')
plt.plot(convlstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x161f34ac8>
```



3. Testing Phase

[ ]:
```
def plot_stock_prediction(window,model,X_test):
    test_result = model.evaluate(X_test, y_test)
    predicted = model.predict(X_test)
    true_Y = test_data['low'].iloc[(window+1):-1]
    true_Y_prev = test_data['low'].iloc[1:-1-window]
    df_test_time = true_Y.index.date
```

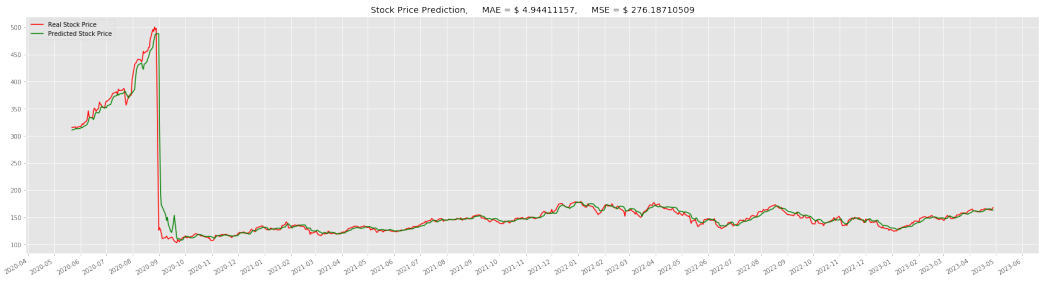
Appendix A.1 Model, Window = 14 Days

```
    pred_Y = (np.array(predicted[:,0])*true_Y_prev)+true_Y_prev
    print("R2 score of ",model._name,": ",r2_score(y_test.reshape(-1,1), predicted))
    # plot test data
    %matplotlib inline
    fig, ax = plt.subplots(figsize=(30, 8))
    monthly_locator = mdates.MonthLocator()
    half_year_locator = mdates.MonthLocator(interval=1)
    year_month_formatter = mdates.DateFormatter("%Y-%m")
    ax.xaxis.set_major_locator(half_year_locator)
    ax.xaxis.set_minor_locator(monthly_locator)
    ax.xaxis.set_major_formatter(year_month_formatter)
    ax.plot(np.array(df_test_time), true_Y,color = 'red', label = 'Real Stock Price')
    ax.plot(np.array(df_test_time), pred_Y,color = 'green', label = 'Predicted Stock␣
↪Price')
    ax.title.set_text(f'Stock Price Prediction, \
    MAE = \$ {round(float(mean_absolute_error(pred_Y,true_Y)),8)}, \
    MSE = \$ {round(float(mean_squared_error(true_Y,pred_Y)),8)}')
    ax.legend(loc='upper left')
    fig.autofmt_xdate()
```

3.a FC-LSTM

[ ]:
```
plot_stock_prediction(window_size,fc_lstm,X_test.reshape((len(X_test), X_test.
↪shape[2], X_test.shape[3])))
```

```
24/24 [==============================] - 1s 40ms/step - loss: 0.0241 - mse: 0.0020 - mae:
0.0241
24/24 [==============================] - 1s 34ms/step
R2 score of  FC-LSTM :  0.8740441414805302
```



[ ]:
```
print("Model training time:",fc_lstm_elapsed)
```

```
Model training time: 94.75706969799921
```

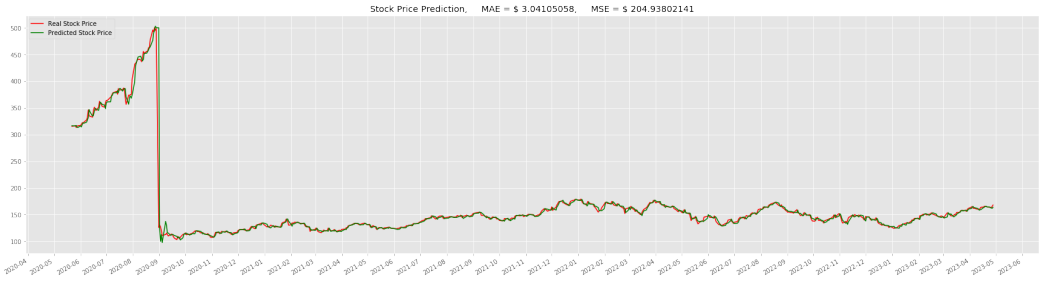
3.b CNN-LSTM

[ ]:
```
plot_stock_prediction(window_size,cnn_lstm,X_test)
```

```
24/24 [==============================] - 0s 11ms/step - loss: 0.0160 - mse: 0.0014 - mae:
0.0160
```

Appendix A.1 Model, Window = 14 Days

```
24/24 [==============================] - 0s 10ms/step
R2 score of  CNN-LSTM :  0.9139324437237274
```



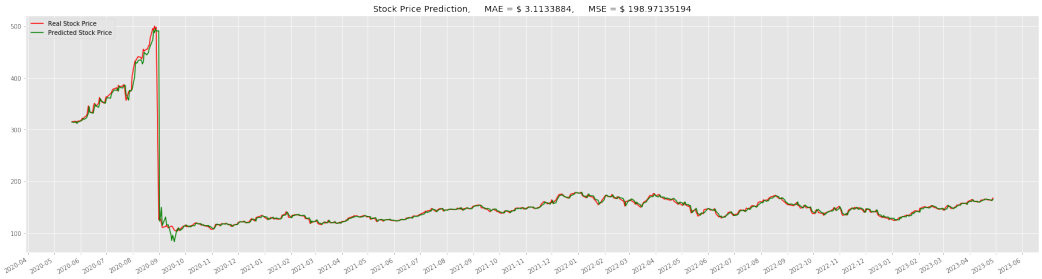

```
[ ]: print("Model training time:",cnn_lstm_elapsed)
```

```
Model training time: 35.57562987100005
```

3.c ConVLSTM

```
[ ]: plot_stock_prediction(window_size,convlstm,X_test)
```

```
24/24 [==============================] - 1s 31ms/step - loss: 0.0157 - mse: 0.0013 - mae:
0.0157
24/24 [==============================] - 1s 31ms/step
R2 score of  ConVLSTM :  0.9177376534570053
```




```
[ ]: print("Model training time:",convlstm_elapsed)
```

```
Model training time: 161.20439217799958
```

Appendix A.2 Model, Window = 30 Days

0. Imports

[ ]:
```
# read in data from tiingo
from pandas_datareader import data as pdr
# basic dataframe and array manipulation
import numpy as np
import pandas as pd
from datetime import datetime
# data visualization
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")
# keras sequential model pipeline
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, MaxPooling1D, MaxPooling2D, Flatten,␣
 ↪Dense, Dropout
from tensorflow.keras.layers import LSTM, ConvLSTM1D, TimeDistributed
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import r2_score
from tensorflow.keras.utils import plot_model
# for GridSearch CV
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
# for timing training
import timeit
# plotting results: formating x axis date
import matplotlib.dates as mdates
```

[ ]:
```
# get spy data from tiingo
with open('credentials/tiingo','r') as f:
    TIINGO_API_KEY = f.read().strip()
# get train data, 6 year
start=datetime(2010, 1, 1)  # start day
end=datetime(2016, 12, 31)  # end day
train_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
# get test data, 3 year, skipping covid boom period (2020 spring)
start=datetime(2017, 1, 1)  # start day
end=datetime(2019, 12, 31)  # end day
val_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
# get test data, 3 year, skipping covid boom period (2020 spring)
start=datetime(2020, 5, 1)  # start day
end=datetime(2023, 5, 1)  # end day
test_data = pdr.DataReader(["AAPL"], 'tiingo', start, end, api_key = TIINGO_API_KEY).
 ↪loc["AAPL"]
```

```
/anaconda3/lib/python3.7/site-packages/pandas_datareader/tiingo.py:234: FutureWarning: In
a future version of pandas all arguments of concat except for the argument 'objs' will be
keyword-only
```

Appendix A.2 Model, Window = 30 Days

```
    return pd.concat(dfs, self._concat_axis)
```

1. Data Preprocessing

[ ]:
```
window_size = 30
def window_data(df, window_size):
    X = []
    y = []
    for i in range(1 , len(df) - window_size -1 , 1):
        first = df.iloc[i,2]
        temp = []
        temp2 = []
        for j in range(window_size):
            temp.append((df.iloc[i + j, 2] - first) / first)
        temp2.append((df.iloc[i + window_size, 2] - first) / first)
        X.append(np.array(temp).reshape(window_size, 1))
        y.append(np.array(temp2).reshape(1, 1))
    return X, y

X_train, y_train = window_data(train_data, window_size)
X_val, y_val = window_data(val_data, window_size)
X_test, y_test = window_data(test_data, window_size)

X_train = np.array(X_train)
X_val = np.array(X_val)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_val = np.array(y_val)
y_test = np.array(y_test)

X_train = X_train.reshape(X_train.shape[0],1,window_size,1)
X_val = X_val.reshape(X_val.shape[0],1,window_size,1)
X_test = X_test.reshape(X_test.shape[0],1,window_size,1)
```

[ ]:
```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(1730, 1, 30, 1)
(722, 1, 30, 1)
(723, 1, 30, 1)
```

2. Model construction

2.a Baseline model: FC-LSTM

2.a.(i) Hyperparameter Tuning

[ ]:
```
def create_fc_lstm(dropout_rate=0.5, recurrent_dropout=0.5, lstm_units = 64):
    model = tf.keras.Sequential()
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
 ↪return_sequences=True,
                   input_shape=(window_size, 1)))
```

Appendix A.2 Model, Window = 30 Days

```
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,activation='relu',␣
 ↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,activation='relu',␣
 ↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'lstm_units': [32, 64, 128],
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
tf.keras.backend.clear_session()
model = KerasRegressor(build_fn=create_fc_lstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train.reshape((len(X_train), X_train.shape[2], X_train.
 ↪shape[3])),y_train,
                       validation_data=(X_val.reshape((len(X_val), X_val.shape[2],␣
 ↪X_val.shape[3])),y_val))
fc_lstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
<ipython-input-8-3d4139973af7>:23: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor(build_fn=create_fc_lstm, epochs=100, batch_size=32,

Best params: {'dropout_rate': 0.3, 'lstm_units': 128, 'recurrent_dropout': 0.3}
```

2.a.(ii) Training the model

[ ]:
```
# tf.keras.backend.clear_session()
# fc_lstm = create_fc_lstm(dropout_rate=0.3,
#                          recurrent_dropout=0.3,
#                          lstm_units = 128)
# fc_lstm._name = "FC-LSTM"
# early_stopping = EarlyStopping(monitor='val_loss',patience=5)
# fc_lstm.summary()
tf.keras.backend.clear_session()
fc_lstm = create_fc_lstm(dropout_rate=fc_lstm_best_params['dropout_rate'],
```

Appendix A.2 Model, Window = 30 Days

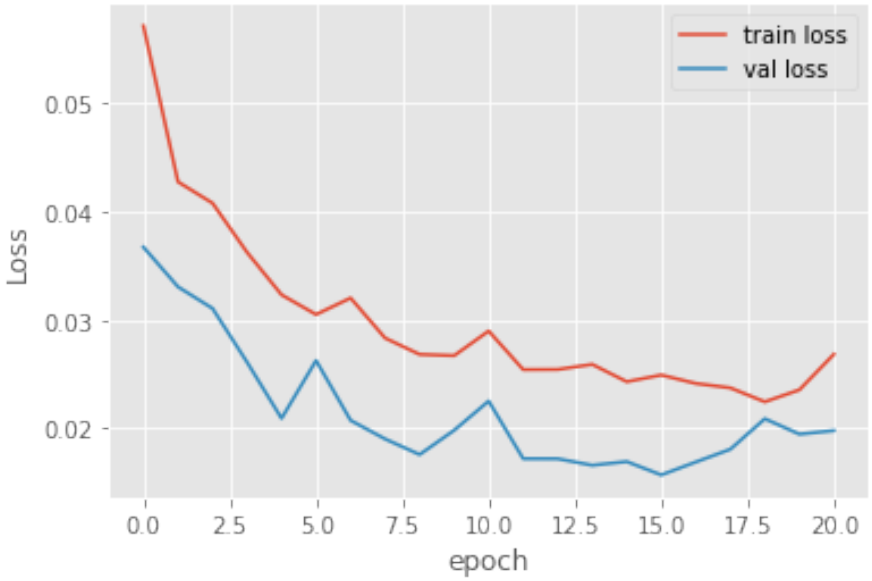
```
                          recurrent_dropout=fc_lstm_best_params['recurrent_dropout'],
                          lstm_units = fc_lstm_best_params['lstm_units'])
fc_lstm._name = "FC-LSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
fc_lstm.summary()
```

```
Model: "FC-LSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 128)           66560

 dropout (Dropout)           (None, 30, 128)           0

 lstm_1 (LSTM)               (None, 30, 128)           131584

 dropout_1 (Dropout)         (None, 30, 128)           0

 lstm_2 (LSTM)               (None, 30, 128)           131584

 dropout_2 (Dropout)         (None, 30, 128)           0

 flatten (Flatten)           (None, 3840)              0

 dense (Dense)               (None, 1)                 3841

=================================================================
Total params: 333,569
Trainable params: 333,569
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
fc_lstm_history = fc_lstm.fit(X_train.reshape((len(X_train), X_train.shape[2], X_train.
 ↪shape[3])),y_train,
                              validation_data=(X_val.reshape((len(X_val), X_val.
 ↪shape[2], X_val.shape[3])),y_val),
                                                epochs=100,batch_size=32, verbose=0,␣
 ↪callbacks=[early_stopping])
fc_lstm_elapsed = timeit.default_timer() - start_time
plt.plot(fc_lstm_history.history['loss'], label='train loss')
plt.plot(fc_lstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x1058ca550>
```

Appendix A.2 Model, Window = 30 Days



2.b CNN-LSTM

2.b.(i) Hyperparameter tuning

[ ]:
```
# Use Grid Search to Tune drop out
def create_cnn_lstm(dropout_rate=0.5, recurrent_dropout=0.5, cnn_units=[32, 64, 128],
                    lstm_units = 64, kernel_size=5):
    model = tf.keras.Sequential()
    # CNN layers
    model.add(TimeDistributed(Conv1D(cnn_units[0], kernel_size=kernel_size,
                                     activation='relu', input_shape=(None,␣
↪window_size, 1))))
    model.add(TimeDistributed(Conv1D(cnn_units[1], kernel_size=kernel_size,␣
↪activation='relu')))
    model.add(TimeDistributed(Conv1D(cnn_units[2], kernel_size=kernel_size,␣
↪activation='relu')))
    model.add(TimeDistributed(MaxPooling1D(2)))
    model.add(TimeDistributed(Flatten()))
    # cnn_lstm.add(Dense(5, kernel_regularizer=L2(0.01)))
    # LSTM layers
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, recurrent_dropout=recurrent_dropout,␣
↪return_sequences=False))
    model.add(Dropout(dropout_rate))
```

Appendix A.2 Model, Window = 30 Days

```
    # FC layers, output
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'cnn_units': [[32, 64, 128], [64, 64, 128]],
    'lstm_units': [32, 64, 128],
    'kernel_size': [3, 5]
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
tf.keras.backend.clear_session()
model = KerasRegressor(build_fn=create_cnn_lstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train, y_train, validation_data=(X_val, y_val))
cnn_lstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
<ipython-input-5-be7f2ccfb87d>:35: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor(build_fn=create_cnn_lstm, epochs=100, batch_size=32,

Best params: {'cnn_units': [64, 64, 128], 'dropout_rate': 0.3, 'kernel_size': 3,
'lstm_units': 64, 'recurrent_dropout': 0.3}
```

2.b.(ii) Training the model

[ ]:
```
tf.keras.backend.clear_session()
cnn_lstm = create_cnn_lstm(dropout_rate=cnn_lstm_best_params['dropout_rate'],
                           recurrent_dropout=cnn_lstm_best_params['recurrent_dropout'],
                           cnn_units=cnn_lstm_best_params['cnn_units'],
                           lstm_units=cnn_lstm_best_params['lstm_units'],
                           kernel_size=cnn_lstm_best_params['kernel_size'])
cnn_lstm._name = "CNN-LSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
cnn_lstm.build(input_shape=(len(X_train),None, window_size, 1))
cnn_lstm.summary()
```

```
Model: "CNN-LSTM"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (1730, None, 28, 64)     256
 ibuted)
```

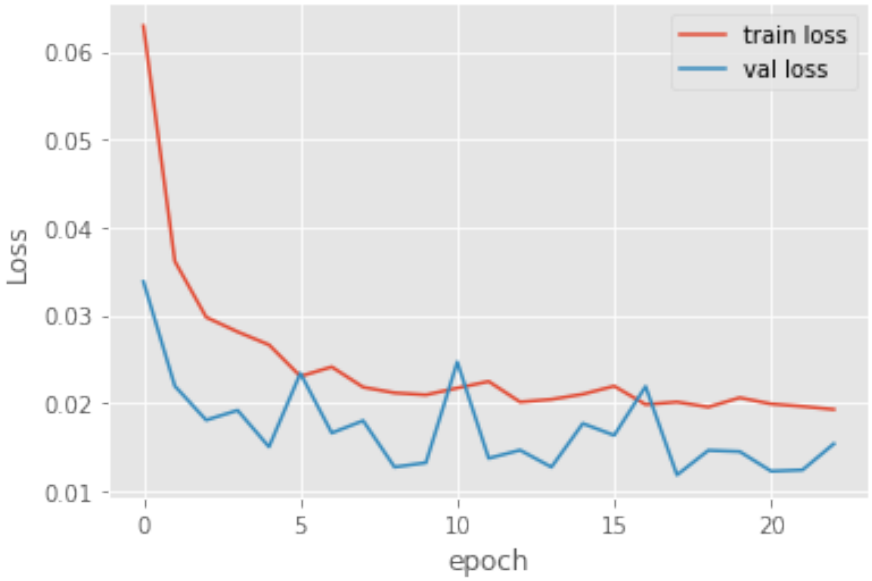
Appendix A.2 Model, Window = 30 Days

```
time_distributed_1 (TimeDis  (1730, None, 26, 64)     12352
tributed)

time_distributed_2 (TimeDis  (1730, None, 24, 128)    24704
tributed)

time_distributed_3 (TimeDis  (1730, None, 12, 128)    0
tributed)

time_distributed_4 (TimeDis  (1730, None, 1536)       0
tributed)

lstm (LSTM)                  (1730, None, 64)         409856

dropout (Dropout)            (1730, None, 64)         0

lstm_1 (LSTM)                (1730, None, 64)         33024

dropout_1 (Dropout)          (1730, None, 64)         0

lstm_2 (LSTM)                (1730, 64)               33024

dropout_2 (Dropout)          (1730, 64)               0

dense (Dense)                (1730, 1)                65

=================================================================
Total params: 513,281
Trainable params: 513,281
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
cnn_lstm_history = cnn_lstm.fit(X_train, y_train, validation_data=(X_val,y_val),
                    epochs=100,batch_size=32, verbose=0, callbacks=[early_stopping])
cnn_lstm_elapsed = timeit.default_timer() - start_time
plt.plot(cnn_lstm_history.history['loss'], label='train loss')
plt.plot(cnn_lstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x153822c50>
```

Appendix A.2 Model, Window = 30 Days



2.c ConVLSTM

2.c.(i) Hyperparameter tuning

[ ]:
```
def create_convlstm(dropout_rate=0.5, recurrent_dropout=0.5, units=[64,64,128],␣
↪kernel_size=5):
    model = tf.keras.Sequential()
    model.add(ConvLSTM1D(filters=units[0], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", input_shape=(None, window_size, 1),␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[1], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[2], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=False))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
```

[ ]:
```
def create_convlstm(dropout_rate=0.5, recurrent_dropout=0.5, units=[64,64,128],␣
↪kernel_size=5):
    model = tf.keras.Sequential()
```

Appendix A.2 Model, Window = 30 Days

```
    model.add(ConvLSTM1D(filters=units[0], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", input_shape=(None, window_size, 1),␣
↪return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[1], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(ConvLSTM1D(filters=units[2], recurrent_dropout=recurrent_dropout,␣
↪kernel_size=kernel_size,
                         padding="same", return_sequences=False))
    model.add(Dropout(dropout_rate))
    model.add(Flatten())
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mae', metrics=['mse', 'mae'])
    return model
# GridSearch CV
param_grid = {
    'dropout_rate': [0.3, 0.5],
    'recurrent_dropout' :[0.3, 0.5],
    'units': [[128, 64, 32],[32, 64, 128],[64, 64, 64]],
    'kernel_size': [3, 5]
}
# create KerasRegressor and GridSearchCV objects
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
model = KerasRegressor(build_fn=create_convlstm, epochs=100, batch_size=32,
                       verbose=0, callbacks=[early_stopping])
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=0)

# perform Grid Search CV
grid_result = grid.fit(X_train, y_train, validation_data=(X_val, y_val))
convlstm_best_params = grid_result.best_params_

# print results
print(f"Best params: {grid_result.best_params_}")
```

```
<ipython-input-14-e965e00549af>:25: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor(build_fn=create_convlstm, epochs=100, batch_size=32,

Best params: {'dropout_rate': 0.5, 'kernel_size': 5, 'recurrent_dropout': 0.3, 'units':
[64, 64, 64]}
```

2.c.(ii) Training the model

[ ]:
```
tf.keras.backend.clear_session()
convlstm = create_convlstm(dropout_rate=0.5,
                           recurrent_dropout=0.3,
                           units=[64,64,64],
                           kernel_size=5)
```

Appendix A.2 Model, Window = 30 Days

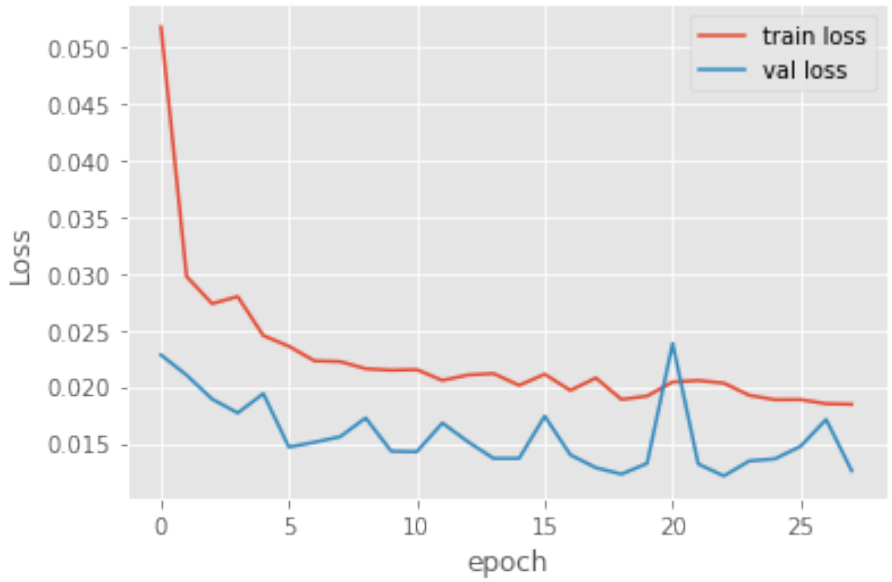
```
convlstm._name = "ConVLSTM"
early_stopping = EarlyStopping(monitor='val_loss',patience=5)
convlstm.build(input_shape=(len(X_train),None, 30, 1))
convlstm.summary()
# tf.keras.backend.clear_session()
# convlstm = create_convlstm(dropout_rate=convlstm_best_params['dropout_rate'],
#                            ␣
 ↪recurrent_dropout=convlstm_best_params['recurrent_dropout'],
#                             units=convlstm_best_params['units'],
#                             kernel_size=convlstm_best_params['kernel_size'])
# convlstm._name = "ConVLSTM"
# early_stopping = EarlyStopping(monitor='val_loss',patience=5)
# convlstm.build(input_shape=(len(X_train),None, 30, 1))
# convlstm.summary()
```

```
Model: "ConVLSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm1d (ConvLSTM1D)    (None, None, 30, 64)      83456

 dropout (Dropout)           (None, None, 30, 64)      0

 conv_lstm1d_1 (ConvLSTM1D)  (None, None, 30, 64)      164096

 dropout_1 (Dropout)         (None, None, 30, 64)      0

 conv_lstm1d_2 (ConvLSTM1D)  (None, 30, 64)            164096

 dropout_2 (Dropout)         (None, 30, 64)            0

 flatten (Flatten)           (None, 1920)              0

 dense (Dense)               (None, 1)                 1921

=================================================================
Total params: 413,569
Trainable params: 413,569
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
start_time = timeit.default_timer()
convlstm_history = convlstm.fit(X_train, y_train,␣
 ↪validation_data=(X_val,y_val),epochs=100,
                                batch_size=32, verbose=0, callbacks=[early_stopping])
convlstm_elapsed = timeit.default_timer() - start_time
plt.plot(convlstm_history.history['loss'], label='train loss')
plt.plot(convlstm_history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

Appendix A.2 Model, Window = 30 Days

[ ]: <matplotlib.legend.Legend at 0x1571ec4a8>



3. Testing Phase

[ ]:
```
def plot_stock_prediction(window,model,X_test):
    test_result = model.evaluate(X_test, y_test)
    predicted = model.predict(X_test)
    true_Y = test_data['low'].iloc[(window+1):-1]
    true_Y_prev = test_data['low'].iloc[1:-1-window]
    df_test_time = true_Y.index.date
    pred_Y = (np.array(predicted[:,0])*true_Y_prev)+true_Y_prev
    print("R2 score of ",model._name,": ",r2_score(y_test.reshape(-1,1), predicted))
    # plot test data
    %matplotlib inline
    fig, ax = plt.subplots(figsize=(30, 8))
    monthly_locator = mdates.MonthLocator()
    half_year_locator = mdates.MonthLocator(interval=1)
    year_month_formatter = mdates.DateFormatter("%Y-%m")
    ax.xaxis.set_major_locator(half_year_locator)
    ax.xaxis.set_minor_locator(monthly_locator)
    ax.xaxis.set_major_formatter(year_month_formatter)
    ax.plot(np.array(df_test_time), true_Y,color = 'red', label = 'Real Stock Price')
    ax.plot(np.array(df_test_time), pred_Y,color = 'green', label = 'Predicted Stock␣
↪Price')
    ax.title.set_text(f'Stock Price Prediction, \
    MAE = \$ {round(float(mean_absolute_error(pred_Y,true_Y)),8)}, \
    MSE = \$ {round(float(mean_squared_error(true_Y,pred_Y)),8)}')
    ax.legend(loc='upper left')
```

Appendix B. Capstone Project Report

Table of Contents

1. Background....3
2. Objective....3
3. Dataset....4
4. Literature Review....5
5. Data Transformation....6
   - 5.1.1. Importance....6
   - 5.1.2. Stationary....6
   - 5.1.3. Data transformation methods and results....6
6. Prediction models....8
   - 6.1. Baseline Model: Last Hour Price Prediction....8
     - 6.1.1. Model Description....8
     - 6.1.2. Results....8
   - 6.2. ARIMA & SARIMA....9
     - 6.2.1. Model Description....9
     - 6.2.2. Building, tuning, and the results of the models....10
     - 6.2.3. Results....10
     - 6.2.4. Potential Problems....12
   - 6.3. Vector Autoregression (VAR)....12
     - 6.3.1. Model Description....12
     - 6.3.2. Building, tuning, and the results of the models....12
     - 6.3.3. Results....12
     - 6.3.4. Potential Problems....13
   - 6.4. Baseline Deep Learning Model....13
     - 6.4.1. Model Description....13
     - 6.4.2. Building, tuning, and the results of the models....13
     - 6.4.3. Results....13
   - 6.5. Gated Recurrent Neural Networks and Long-Short Term Memory (RNN and LSTM)....14
     - 6.5.1. Model Description....14
     - 6.5.2. Building, tuning, and the results of the models....15
     - 6.5.3. Results....16
   - 6.6. Informer (A Transformer based model)....17
     - 6.6.1. Model Description....17
     - 6.6.2. Building, tuning, and the results of the models....19
     - 6.6.3. Results (Appendix 4)....19
     - 6.6.4. Potential Problems....20
7. Project Timeline....20

Appendix B. Capstone Project Report

8. Appendix 21

1. Background

Motivation

In this revolutionized age, every industry is thriving towards profit margin maximization by utilizing advanced algorithms. Energy sector is also developing strategies to gain a larger profit margin and remain competitive in the field. One of the most important means to achieve such a position is to reduce the production cost. Efficient strategic planning then becomes a crucial factor to manage production cost in both short term and long term.

2. Objective

Objective

This project aims to enhance short term production cost efficiency by providing **Day-ahead Market (DAM) electricity price prediction using Machine Learning tools**. With the ability to forecast the DAM price, stakeholders can efficiently make strategic decisions by submitting a profitable bidding (for example, if the price is expected to be higher in a specific operating day, producer can submit high price to make as much profit as possible), and optimizing production costs in their daily operations (for example generation scheduling, and man-power planning) corresponding to the predicted price. Research estimated that a 1% improvement of the short-term forecasting can result in about $0.5 million savings per year for a utility company with 1 GW peak load.

Electricity flow

Electricity is generated by producers, passed to the distribution network, and then delivered to users. Producers have various fuel types to generate electricity, including natural gas, coal, hydroelectric, wind, solar, nuclear, etc.

In the electricity market, special characteristic is adopted due to physical equipment constraints. Electricity in power grid networks must be always balanced. Thus, electricity produced must be used simultaneously. Additionally, industrial battery storage is still developing and thus not economic. Therefore, supply must meet demand exactly in the power grid and this must be handled in real time. In this situation, ISO/TRO (independent system operator or regional transmission organization) has been established, specifically to manage the demand and supply in each region. Currently, there are a total of 9 ISO organizations operating in North America.

Electricity pricing

DAM price, as well as Real-time Market (RTM) price, are one of the aspects ISO/TRO has to manage to ensure agreed electricity price and supply from both producers and users. For DAM, users and producers submit the bidding amount and price of the electricity they plan to

Appendix B. Capstone Project Report

purchase/supply in each hour of tomorrow. DAM is agreed one day before the operating/delivering day. On the other hand, RTM deals with the change in production and consumption throughout the actual operating/delivery day. RTM is agreed 15 minutes before delivery time. Trades can take place immediately when individual purchases and sale bids meet.

3. Dataset

To make time-series predictions on electricity price, we will be firstly generating a baseline model with only basic features, and then add features that might be relevant to electricity price in Texas areas. The datasets we will be using for this project therefore include *basic datasets* and *additional feature*.

Basic Dataset

To generate a baseline model, we use Historical DAM Load Zone and Hub Price, collected from the Electric Reliability Council of Texas (ERCOT), which manages the flow of electric power to 26 million Texas customers, representing about 90% of the state's electricity load. This is a yearly recorded dataset with information on historical DAM Settlement Point Prices (SPPs) for each of the Hubs and Load Zones. We aggregated datasets from 2010 to 2023 for baseline model generation. Features of this dataset include Delivery Date, Hour Ending, Repeated Hour Flag, Settlement Point, and Settlement Point Price.

Additional Feature

Additional feature used is Electricity loading, for which the concept is similar to demand. A prediction in demand will be helpful for predicting price using economic theories. Data is recorded hourly since 2010.

Appendix B. Capstone Project Report

4. Literature Review

Our research is mainly inspired by two literatures in the field:

- Forecasting electricity prices for a day-ahead pool-based electric energy market by Antonio J. Conejoa et al.
- Day-Ahead Price Forecasting in ERCOT Market Using Neural Network Approaches by Jian Xu et al.
- Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting by Zhou, Zhang, et al.

In the first paper mentioned above, Conejoa et al. have highlighted the importance of forecasting electricity prices for price-taker producers in their research. In their paper, the authors explain how accurate price forecasts can help price-taker producers make optimal decisions on self-scheduling and bidding in electricity markets. They further emphasize that price forecasts are vital for improving the efficiency of electricity markets and enhancing the investment incentives for market participants. The authors of the paper proposed three different frameworks for forecasting electricity prices. The first framework, called the "Baseline Error Testing," involves comparing the hourly errors of any forecasting technique with the market-clearing prices of a similar day to determine if the estimates are accurate. The second framework uses time series analysis, such as ARIMA, dynamic regression, and transfer function models, which are considered the most effective empirical models for analyzing day-ahead market (DAM) electric prices. However, the authors also noted that **time series data have several unpleasant characteristics, such as high frequency, non-constant mean and variance, daily and weekly seasonality, calendar effects on weekends and holidays, high volatility, and presence of outliers.** To address these issues, they suggested using the differencing method to make the mean constant, taking logarithms to alleviate non-constant variance, and incorporating seasonality models to capture daily and weekly seasonality. The third framework involves the use of neural network models.

The second paper explored different deep learning models to predict electricity prices. The authors found that **recurrent neural networks performed the best** among all the neural network models. They also noted that **removing the spike prices as noise from the training dataset could improve the accuracy of forecasting** prices that are under $100/MWh, but more inputs need to be included to forecast price spikes accurately. This is an area that the authors suggest for further research. Overall, the paper provides insight into the use of neural network approaches for day-ahead price forecasting and identifies some areas for improvement and future exploration.

The third paper in this report focuses on **the Informer model**, an improved version of the transformer model, and explores its theoretical background and practical applications in time

Appendix B. Capstone Project Report

series forecasting. The paper provides a detailed examination of the model's architecture, which features an encoder-decoder structure with a multi-level temporal attention mechanism. The attention mechanism consists of an encoder stack, an embedding module, a multi-scale feature representation module, a decoder stack, and a forecasting module. Notably, the authors discuss how the **ProbSparse Attention** mechanism can improve the training speed by selectively focusing on relevant information and ignoring irrelevant information, making it particularly effective for **dealing with sparse or skewed data**. The paper presents experiments involving the building of **univariate and multivariate models** using different deep learning architectures, which were conducted on financial, weather, and real estate price data. The results demonstrate that the **Informer model outperforms** all other models in terms of test mean squared error (MSE). Based on these findings and the demonstrated superiority of the Informer model, we have decided to incorporate it into our dashboard for time series forecasting. We are confident that this powerful model will provide accurate and reliable predictions, enabling us to make informed decisions and drive better outcomes.

The above literature reviews provided us a future direction for DAM electricity price estimation—neural network method, e.g. LSTM & RNN. However, under the advice of GHD representatives, we will be focused on building well-performed time series analysis in the short term, in order to enhance our understanding of this whole economical event as well as the dataset.

5. Data Transformation

5.1.1. Importance

Models such as ARIMA/SARIMA estimate relationships between certain period values and past period values to predict future period values with the assumption that the same relationship holds, in other words, data is stationary. In deep learning models, being capable of learning nonlinearities, stationary is less necessary yet proved to help increase the performance. (Smoothing and stationarity enforcement framework for deep learning time-series forecasting)

5.1.2. Stationary

Criteria: Constant mean, Constant variance, No seasonality/predictable repeating pattern
Data Observation: Mean and variance not constant, Seasonality observed on daily basis

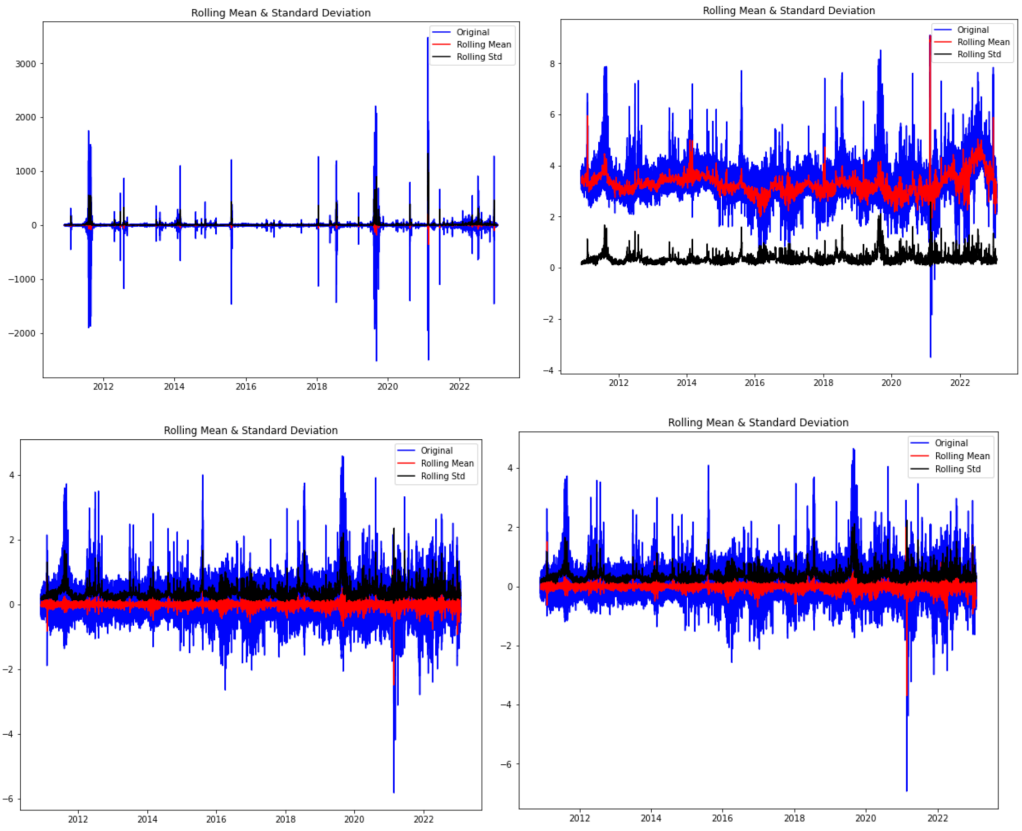
5.1.3. Data transformation methods and results

Several data transformation methods have been applied. Criteria to confirm stationary are visual observation and Augmented Dickey-Fuller Test. From the results, data with time-shift of log-scaled data is selected to proceed with model development since it is observed to be most

Appendix B. Capstone Project Report

stationary. Note that different models have compatibility with different transformed data and in model development, transformation method can be alternated.

	Data Transformation	Visual	ADF test	P-value
1	Time shift	Spike for mean and stdev observed	-46.30	< 0.05
2	Log scale	Not stationary	-12.05	< 0.05
3	Log scale - Moving Average	Spike for mean and stdev observed	-42.66	< 0.05
4	Exponential Decay	Spike for mean and stdev observed	-32.48	< 0.05
5	Time shift log scale	Few stdev fluctuation observed	-61.40	< 0.05



Appendix B. Capstone Project Report

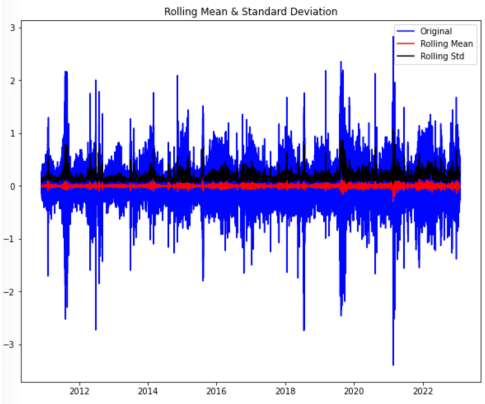


Figure 1 Data Transformation Results

(1-top left, 2-top right, 3-middle left, 4-middle right, 5 bottom left)

6. Prediction models

6.1. Baseline Model: Last Hour Price Prediction

6.1.1. Model Description

The baseline model is a straightforward approach that relies on using the price of the current hour to predict the price of the next hour.
At its core, the model assumes a linear relationship between the prices of consecutive hours and that this relationship remains consistent over time. With this assumption, the model generates a forecast for the price of the next hour based on the current hour's price.
While it may seem simple, this approach can provide valuable insights into the behavior of an asset over time. It's often used as a starting point for more sophisticated forecasting methods and can help establish a baseline for evaluating the effectiveness of those methods.

6.1.2. Results

- MAE on scaled data as 0.062.
- MAE in dollars is $16.63.

Appendix B. Capstone Project Report

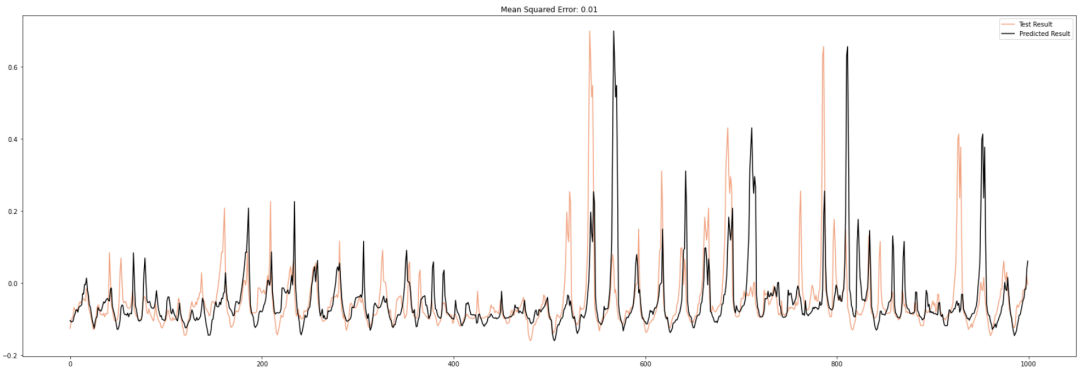


Figure 2 Baseline Model Prediction Result

6.2. ARIMA & SARIMA

6.2.1. Model Description

ARIMA stands for AutoRegressive Integrated Moving Average. The model consists of three parts as we can see from the name itself.

- The AutoRegressive part utilizes the dependent relationship between an observation and a number of lagged observations, in other words, it represents a linear regression that predicts future values based on its own past values.
- The Moving Average part regresses time series observations on their past residual errors.
- The Integrated part takes the difference of an observation from an observation at the previous time step in order to make the time series stationary. A stationary series has mean, variance, and autocorrelation that do not change over time, which is particularly essential in models such as ARIMA that assumes the time series data is stationary. A nonstationary series can make it difficult to identify trends and patterns in the data and make accurate predictions.

6.2.2. Building and tuning

1. Data preprocessing
   - Log-scaled first-different data transformation
2. Train/Test sets
   - Train data: 2022-10-22 to 2023-01-22 (3 months)
   - Test data: 2023-01-22 to 2023-01-29 (7 days)
3. Hyperparameter tuning
   - To build the model, we utilized statsmodel Python library and grid search method for hyper-parameter optimization, including:

Appendix B. Capstone Project Report

ARIMA			SARIMA		
	Description	**Used**		**Description**	**Used**
p	Trend autoregression order	4	**p**	Trend autoregression order	1
d	Trend difference order	1	**d**	Trend difference order	1
q	Trend moving average order	2	**q**	Trend moving average order	3
			P	Seasonal autoregressive order	1
			D	Seasonal difference order	0
			Q	Seasonal moving average order	1
			m	Number of time steps for single seasonal period	24

6.2.3. Results

After visualizing the prediction result of our ARIMA model, in Appendix 1, we found that it captures the initial spike of the test data, but then exhibits a smooth, straight line. Even though the MAE of the model is relatively low, the model did not capture the seasonality nature of the time series model. Seasonality in time series refers to the recurring patterns of the data that occur at a fixed interval, which can be hourly, daily, weekly, monthly, or yearly.

We then chose to build another model, SARIMA, in Appendix 1, which is an extension to the ARIMA model that incorporates a further autoregressive model, moving average model, and a differencing component to account for the seasonality nature. We can specify the seasonality time period in the SARIMA model, and in our case, it should be hourly as the DAM prices data is in hours

- The best MAE for the ARIMA model is 0.1456.
- The best MAE for the SARIMA model is 0.1285.

Appendix B. Capstone Project Report

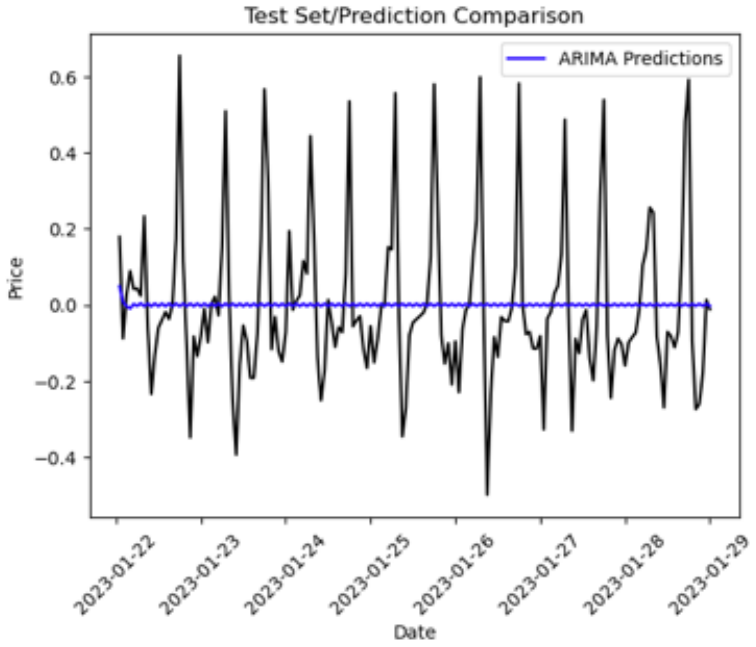


Figure 3 ARIMA Model Prediction Result

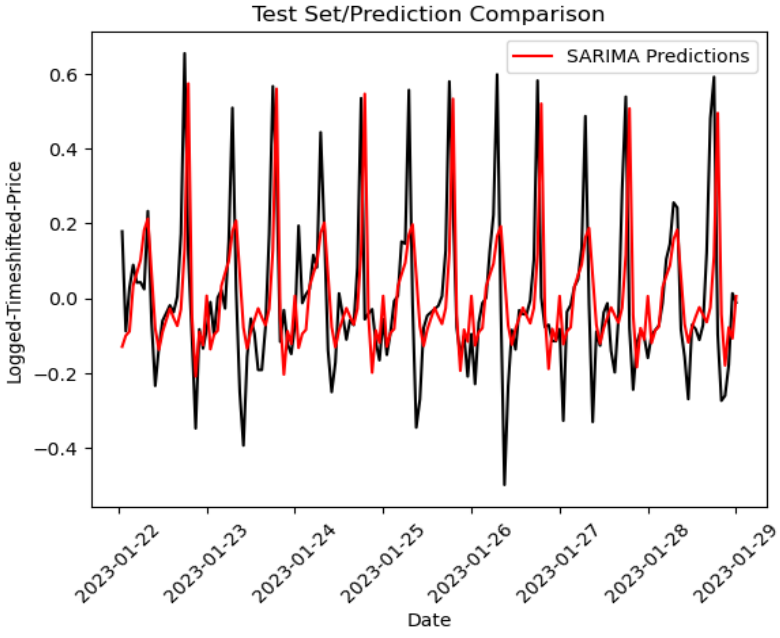


Figure 4 SARIMA Model Prediction Result

Appendix B. Capstone Project Report

6.2.4. Potential Problems

To incorporate other features in addition to price data to our SARIMA model, we can use SARIMAX, which is an extension to the SARIMAX model that can take in exogenous variables. However, there is a caveat regarding the exogenous variables we can use in the SARIMAX model. The data for the exogenous variables should be available for the entire period of interest that we are forecasting, which makes it a limiting factor to some of the potential features such as CPI and precipitation as these data will not be known for the future.

6.3. Vector Autoregression (VAR)

6.3.1. Model Description

Vector Autoregression (VAR) model describes the evolution of a set of k variables over time. The current observation of each variable depends on its own lagged values and the lagged values of each of the other variables.

6.3.2. Building and tuning

1. Data preprocessing
   - Join multiple datasets on the “time” variable. We need to compile the datasets into same time granularity if they are not.
   - Check if data is stationary using ‘adfuller’ package. If p-value is less than or equal to 5%, then the data is stationary. Otherwise, we need to differentiate the data. The dataset we currently have is stationary and no further step need to be done.
   - Check correlations between each feature. We need to make sure whether a feature causes another and vice versa. We do this step by using the ‘granger causality tests’ package, if p-value is less than or equal to 5%, then there is the causal relationship. Otherwise, we need to drop the lag. The dataset we currently have has significant causal relationships and no further step need to be done.
2. Train/test sets
   - Train data: 2022-06-01 to 2022-11-30
   - Test data: 2022-12-01 to 2022-12-31
3. Hyperparameter tuning
   - The hyperparameter of this model is the number of lags. The best number of lags to use is 490.

6.3.3. Results

- The best MAE for the VAR model is 144.38

Appendix B. Capstone Project Report

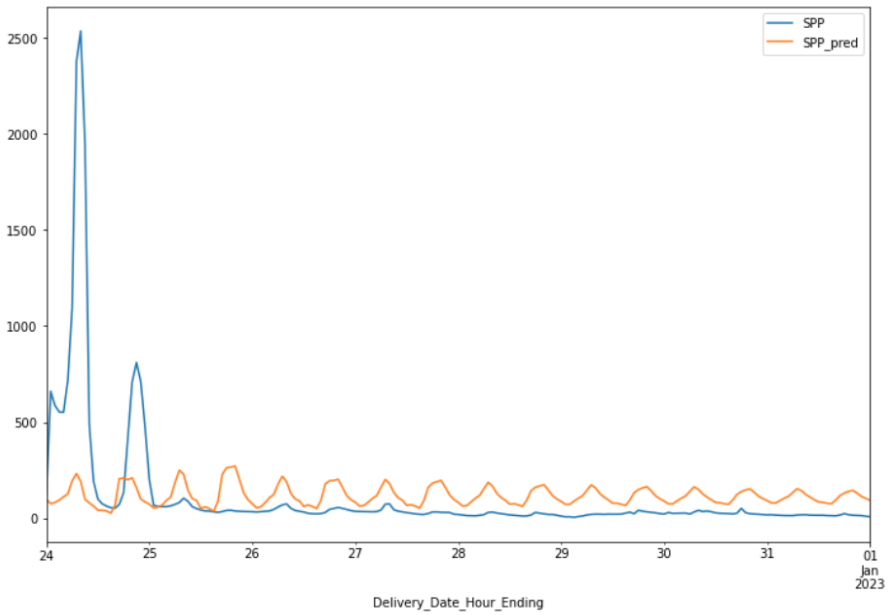


Figure 3 VAR Model Prediction Result

6.3.4. Potential Problems

Model efficiency is not worth the amount of time used to run. We do not recommend this model.

6.4. Baseline Deep Learning Model

6.4.1. Model Description

Prior to delving into intricate and computationally demanding models like RNN, it would be worthwhile to consider employing a small, densely connected network. This approach serves as a crucial litmus test to verify whether any additional complexity introduced to the model genuinely results in tangible benefits.

6.4.2. Building and tuning

1. Data preprocessing

We have utilized a dataset encompassing the previous twelve years' worth of information, consisting of 106608 rows that represent hourly intervals. As a pre-processing step, we have applied a normalization technique to the time series data, which involves adjusting it to have a mean of zero and a standard deviation of one.

2. Train/Validation/Test sets
   - The initial 80,000 rows of the dataset are allocated for training purposes.
   - Subsequently, rows 80,001 to 90,000 are reserved for validation purposes.
   - All the remaining rows beyond 90,000 are designated as the test set.

Appendix B. Capstone Project Report

3. Hyperparameter tuning
    - This model entails a very simple sequential architecture with three layers, Input layer, hidden layer with 32 neurons, employing the Rectified Linear Unit (ReLU) activation function, and an output layer.

6.4.3. Results

The model yields a relatively bad result compared to our baseline model

- MAE on scaled data as 0.12
- MAE in dollars as $32.84.

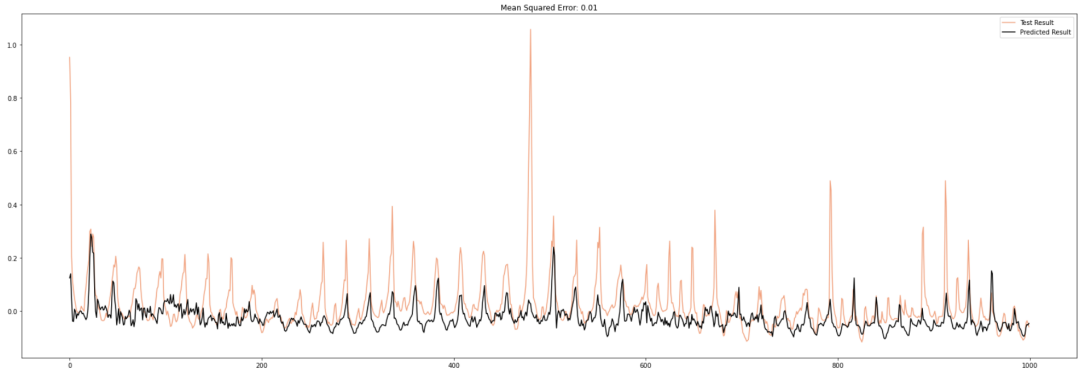
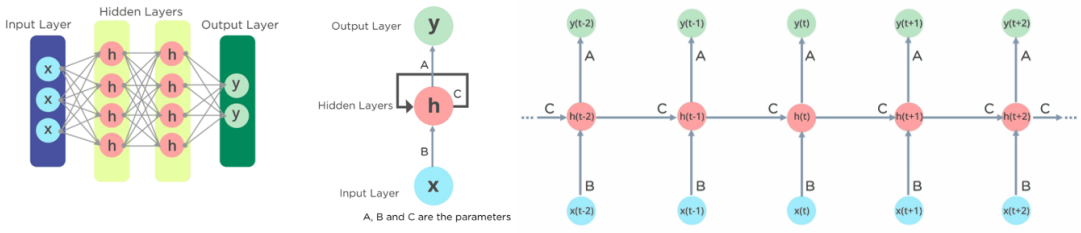


Figure 5 Baseline Deep Learning Model Prediction Result

6.5. Gated Recurrent Neural Networks and Long-Short Term Memory (RNN and LSTM)

6.5.1. Model Description

While **simple neural networks** (baseline deep learning model) is compatible with non-linearities, it still cannot handle sequential data. **Recurrent neural networks (RNN)** method has been introduced to tackle such problems. RNN computation has an ability to take into account the historical data. RNN calculation structure composed of several simple neural networks for each time step. And each time step takes the result from previous and current step values to make a prediction.



Simple Neural Network (ANN) **Recurrent Neural Network (RNN)**

Figure 6 ANN and RNN comparison

Appendix B. Capstone Project Report

Simple RNN also has drawbacks. Since the estimation of parameters uses loss function of all time steps and is solved by back propagation (weights update using multiplication of gradient Loss change with respect to weights and bias), it eventually leads to 'Vanishing/Exploding Gradients' problem. Multiplicative gradients can be exponentially decreasing (gradient very close to 0, too small steps)/increasing (gradient very large, ends up jumping around) with respect to the number of layers. This obstacle has been solved by the improved version of RNN, introduced as **'Long Short-Term Memory' or LSTM**.

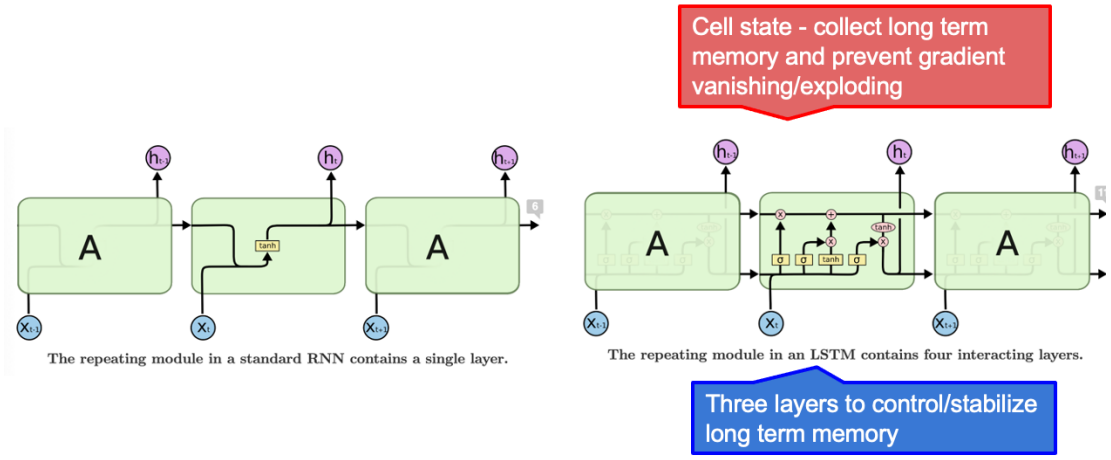


Figure 7 Simple RNN and LSTM comparison

Additional functions introduced in LSTM are mainly to collect long term memory and prevent gradient vanishing/exploding. LSTM has been widely proved efficient, thus, we will implement this certain model.

6.5.2. Building and tuning

1. Data preprocessing and Train/Validation/Test set

Same as what we did before, as part of our data preparation, we have utilized a dataset containing 106,608 rows, representing hourly intervals over a twelve-year period. Prior to further analysis, we have applied a normalization technique to the time series data, ensuring that the mean value is set to zero and the standard deviation is set to one. We allocate the first 80,000 rows for training, the following 10,000 for validation (rows 80,001 to 90,000), and the remaining rows beyond 90,000 for testing.

2. Hyperparameter tuning

The following variables can be designated and adjusted to enhance the accuracy of prediction models.

- "lookback" refers to the number of timesteps that the input data should span. For instance, choosing 168 timesteps allows for a seven-day period to be analyzed.

Appendix B. Capstone Project Report

- "delay" pertains to the number of timesteps in the future that the target should be predicted. Selecting 24 timesteps, for example, enables a day to be predicted.
- "min_index" and "max_index" are indices that determine the timesteps used for training, validation, and testing. This helps to ensure that the dataset is properly segmented.
- "shuffle" denotes whether the data samples should be randomly shuffled or presented in chronological order. In this instance, shuffling was not applied, but it can be if desired.
- "batch_size" indicates the number of data samples per batch.
- "step" represents the time period between each data sample, with a value of 1 meaning that one data point is taken each hour.
- "Val_steps" and "Test_steps" signify how many steps should be drawn from the validation and test generators, respectively, in order to encompass the entire dataset.

hyperparameters that we can fine-tune are:

- Number of neurons in each layer.
- Activation function.
- Optimizer.

3. Models

We build two RNN based models: GRU(Gated Recurrent Neural Networks) model and LSTM (Long Short Term Memory) model.

- The GRU layers operate on a similar principle to LSTM layers, albeit with a simplified design that renders them computationally less expensive (although they may not be as expressive as LSTM layers). The GRU model follows a sequential architecture and comprises a single GRU layer with 32 neurons, along with a Dense layer containing a sole output. During training, we utilize the RMSprop optimizer and adopt Mean Absolute Error (MAE) as the loss function.
- The LSTM model is a sequential model consisting of an LSTM layer with 32 neurons and a Dense layer with one output. The model is trained using the RMSprop optimizer and Mean Absolute Error (MAE) is utilized as the loss function. The training procedure includes 20 epochs with 200 steps per epoch, and validation is performed on a separate validation generator with 200 validation steps.

6.5.3. Results

- GRU
  - MAE: 0.13011572008860783
  - MAE in dollars: 34.8053559325
- LSTM
  - MAE: 0.18296050840141323
  - MAE in dollars: 48.9410934525

Appendix B. Capstone Project Report

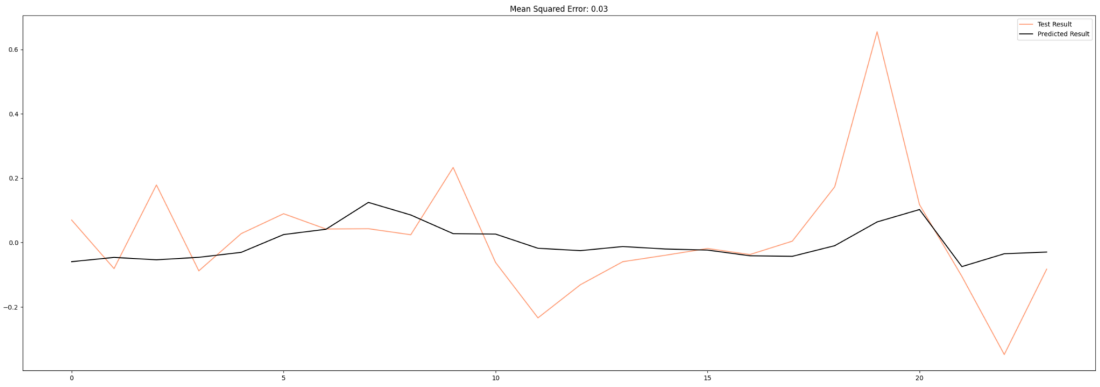


Figure 8 GRU Model Prediction Result

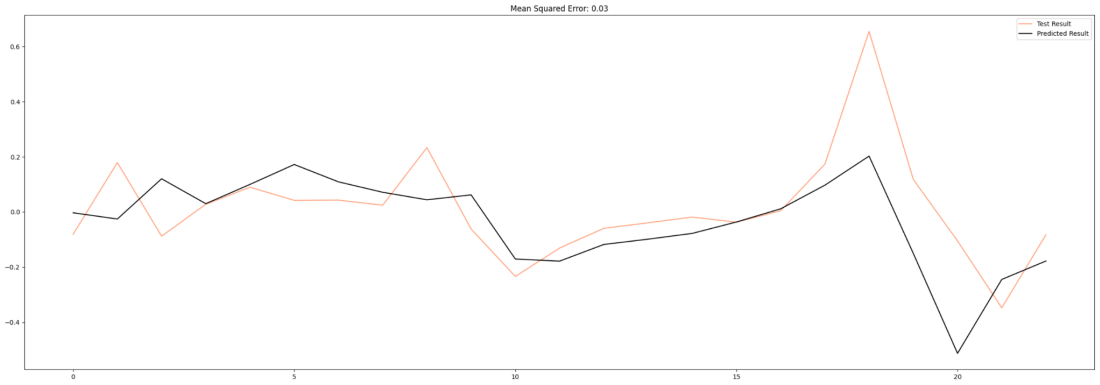


Figure 9 LSTM Model Prediction Result

6.6. Informer (A Transformer based model)

6.6.1. Model Description

Compared to RNN, the Transformer model already achieves capturing long-range dependency by allowing long-sequence input all-in-once. However, traditional Transformer models still haven't resolve time complexity(*O(L^2)*) with self-attention, high memory bottleneck(*O(J L^2)*, J layers) and encode-decoder architecture limitation in speed (as slow as RNN). Other Possible alternatives, endeavored to tackle the above problems, but could never find a great balance of solving all of these at the same time. For instance, Sparse Transformer solved time complexity but exchanged for efficiency; Reformer also resolved time complexity but put limits on length of sequence input; Informer reduced time complexity the most(*O(L)*), but with the risk of degradation to *O(L^2)* if on real-world long-sequence input.

Here, the proposed Informer model by Zhou, Zhang, et al. strived to find a great balance of time, memory and architecture. It proposed a ProbSparse mechanism which supports random sampling U query paris that summed most of information without ranking query sparsity

Appendix B. Capstone Project Report

measurements and selecting Top-U query paris. Using ProbSparse self-attention mechanism, it incorporates multi-head attention blocks in the encoder and simple, one-forward generative-style decoder that improves time and memory complexity(*O(LlogL)*). The attention mechanism cascades the layer inputs to reduce space complexity *O((2 − ε)L log L)*.

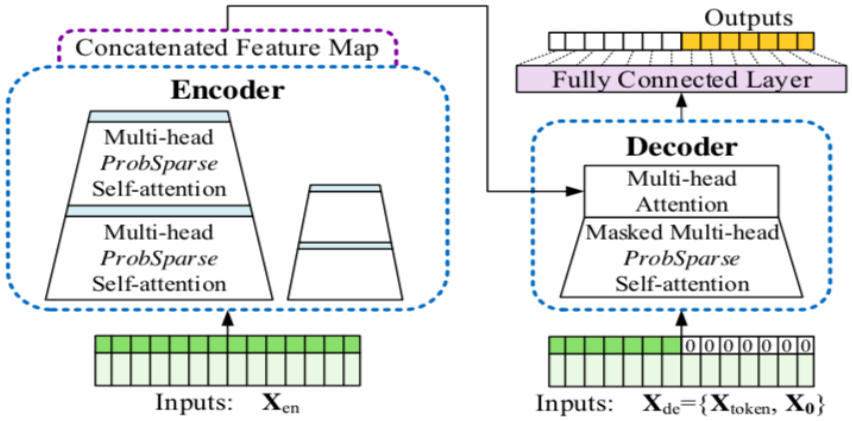


Figure 10 Informer Model Overview
(Left: encoder + long-sequential input; Right: decoder -> outputs generatively)

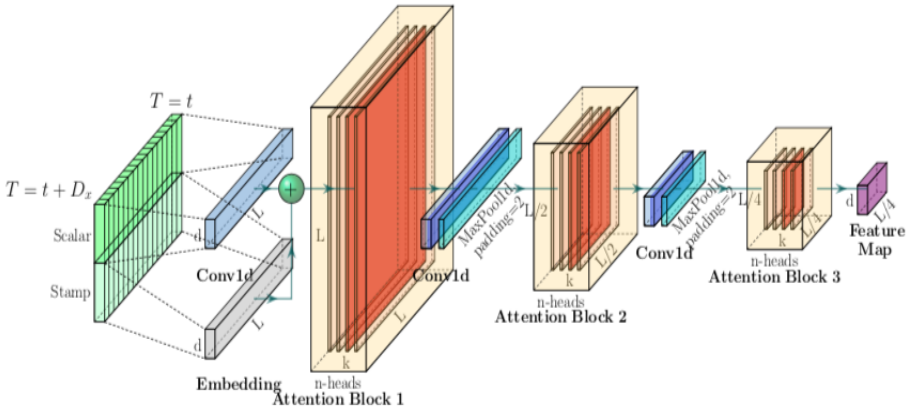


Figure 11 The Single Stack in Informer's Encoder

Appendix B. Capstone Project Report

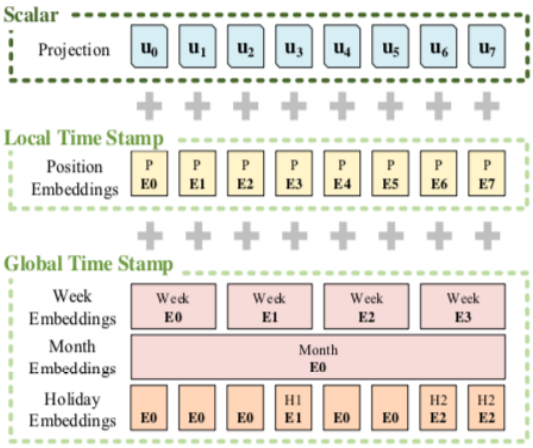


Figure 12 Uniform Input Representation

6.6.2. Building and tuning

1. Data preprocessing
   - HB_BUSAVG, Log-scaled first-different data transformation (log return)
   - Feature?
   - Input context embedding (fixed position + learnable time stamp embedding)
2. Train/Test set
   - Train data: 2022-10-22 to 2023-01-22 (3 months)
   - Test data: 2023-01-22 to 2023-01-29 (7 days)
3. Hyperparameter tuning
   - Learning rate(chosen to be 1e^-4, divide by 2 every epoch, total 8 epochs)
   - Batch size: 32
   - Train epochs: 20
   - Model structure:
     - Head number of multi-head attention: 8
     - Dimension of multi-head attention's output: 512
     - Length of encoder's input sequence: 168
     - Length of decoder's start token L_token: 84
     - Layer of encoder: 2
     - Layer of decoder: 1
     - Regularization-dropout: 0.05

6.6.3. Results

- MAE: 0.10165277

Figure 13 Transformer Model Prediction Result

Appendix B. Capstone Project Report

6.7. Model Summary and recommendation

Summary of models performance is shown below. Transformer is the most efficient model in terms of both time consumption and accuracy.

Model	MAE (unconverted data)	MAE (converted data)	Training time approximation	Visual observation
SARIMA	0.097	2.31	MacBook Pro: 20 mins (only for training 3 months of data)	good
VAR	-	144.38	MacBook Air: 6 hours	bad
LSTM	in progress	in progress	MacBook Air: 2 hours (initial trial)	
Informer (Transformer-based)	0.097	2.67	MacBook Pro: 30 mins (1 year data) Nvidia GPU: 5 mins	good

7. Dashboard

We built an interactive dashboard using Streamlit connecting to our trained Informer model…..

8. Conclusion

Given the mechanism of electricity market and transactions, stakeholders can make strategic decisions regarding electricity price bidding with the future price predictions. In this project, we managed to study time series analysis and develop machine learning models that could help with the price predictions. Based on the literature review, talks with the GHD representatives, and our own knowledge in this domain, we developed and tested on 6 aforementioned models, including both the baseline models and the ones with exogenous features. Considering both the model performance and the running efficiency, we recommend the Informer (Transformer-based model) to be the most practical model in the case we study. The results are presented through a dashboard using Streamlit.

For further improvements, it would be beneficial to consider more exogenous features that might be impactful to electricity price (currently we only consider hourly loading), for example, weather (especially extreme climate), consumer behaviors, regulations and big events (e.g. elections).

Appendix B. Capstone Project Report

Techniques that could potentially improve model performance and working efficiency are also worth studying.

9. Project Timeline

Report	Prof/TA meeting	Peer feedback	GHD meeting		Plan	Actual

Wk	SUN	MON	TUE	WED	THU	FRI	SAT
1	22-Jan	23-Jan	24-Jan	25-Jan	26-Jan	27-Jan	28-Jan
	Kick off meeting - Problem definition. Identify data sources						
	Kick-off meeting. Problem definition. Identify data sources						
2	29-Jan	30-Jan	31-Jan	1-Feb	2-Feb	3-Feb	4-Feb
	Collect data. Clean Data. Begin building appropriate feature sets						
	Background description, Data identification, Acedamic paper research						
3	5-Feb	6-Feb	7-Feb	8-Feb	9-Feb	10-Feb	11-Feb
	Collect data. Clean Data. Begin building appropriate feature sets						
	ARIMA, SARIMA first models						
4	12-Feb	13-Feb	14-Feb	15-Feb	16-Feb	17-Feb	18-Feb
	Feature exploration. Examine distributions. Visualize data. Figure out dependent variables. Feature engineering. Identify appropriate techniques.						
	Data transforrmation and update SARIMA model						
5	19-Feb	20-Feb	21-Feb	22-Feb	23-Feb	24-Feb	25-Feb
	Feature exploration. Examine distributions. Visualize data. Figure out dependent variables. Feature engineering. Identify appropriate techniques.						
6	26-Feb	27-Feb	28-Feb	1-Mar	2-Mar	3-Mar	4-Mar
	Modeling. Run models. Iteratively re-engineer features, adjust hyper parameters, evaluate results						
	5-Mar	6-Mar	7-Mar	8-Mar	9-Mar	10-Mar	11-Mar
	MIDTRERM						
	12-Mar	13-Mar	14-Mar	15-Mar	16-Mar	17-Mar	18-Mar
	SPRING BREAK						
7	19-Mar	20-Mar	21-Mar	22-Mar	23-Mar	24-Mar	25-Mar

Appendix B. Capstone Project Report

8	26-Mar	27-Mar	28-Mar	29-Mar	30-Mar	31-Mar	1-Apr
9	2-Apr	3-Apr	4-Apr	5-Apr	6-Apr	7-Apr	8-Apr
	Finish modeling. Work on visualizations of the results. Build a prototype app if required by the project.						
	DAM and Load data (additional feature) - VAR, RNN, LSTM, Transformer models results						
10	9-Apr	10-Apr	11-Apr	12-Apr	13-Apr	14-Apr	15-Apr
	Finish modeling. Work on visualizations of the results. Build a prototype app if required by the project.						
11	16-Apr	17-Apr	18-Apr	19-Apr	20-Apr	21-Apr	22-Apr
	Complete work. Make final presentation to client. Deliver report/Presentations/dashboards/apps to client.						
	Dashboard						
12	23-Apr	24-Apr	25-Apr	26-Apr	27-Apr	28-Apr	29-Apr
	Complete work. Make final presentation to client. Deliver report/Presentations/dashboards/apps to client.						
13	30-Apr	1-May	2-May	3-May	4-May	5-May	6-May
	Final Report deliverable to GHD						
14	7-May	8-May	9-May	10-May	11-May	12-May	13-May

Work Cited

Conejo, Antonio J., et al. “Forecasting Electricity Prices for a Day-Ahead Pool-Based Electric Energy Market.” International Journal of Forecasting, vol. 21, no. 3, 2005, pp. 435–462., https://doi.org/10.1016/j.ijforecast.2004.12.005.

Xu, Jian, and Ross Baldick. “Day-Ahead Price Forecasting in ERCOT Market Using Neural Network Approaches.” Proceedings of the Tenth ACM International Conference on Future Energy Systems, 2019, https://doi.org/10.1145/3307772.3331024.

Appendix B. Capstone Project Report

Zhou, Zhang, et al. “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting.” https://doi.org/10.48550/arXiv.2012.07436.

10. Appendix

Appendix 1: Kick-off presentation by GHD
Appendix 2: Feb 9th Progress presentation by MSBA
Appendix 3: Feb 24th Progress presentation by MSBA
Appendix 4: Mar 23rd Progress presentation by MSBA
Appendix 5: Apr 6th Progress presentation by MSBA
Appendix 6: DAM data
Appendix 7: Electricity loading data