Evaluating the Potential of Deep Learning in Financial Time Series Forecasting: A Comparative Approach

Friday 24th November, 2023

Avula Mohana Durga Dinesh Reddy[1], Prabhath Chellingi[1], Prajwaldeep Kamble[2], and Ojjas Tyagi[1]

[1]Department of Computer Science and Engineering, IIT Hyderabad, India
[2]Department of Mathematics and Computing, IIT Hyderabad, India
{cs20btech11005, cs20btech11038, ma20btech11013, cs20btech11060}@iith.ac.in

Abstract

Financial time series forecasting can be dated back to old times when it stood as the major field of a country's economic stability. It plays a vital role in modern finance, providing insights and predictions for ***crucial investment decisions****,* ***risk management****, and* ***market analysis*** *from an individual to the nation's level. Many traditional mathematical methods are used, but the introduction of Machine learning models like* ***CNNs, LSTM, RNNs,*** *and* ***Transformers*** *made a new revolution in prediction. They are more accurate and specific with the* ***feasibility of choosing several parameters****. The models are not just trained on the statistical data but also on visual information of the graphs. Here, we are doing a* ***comparative study of these models*** *on different datasets to show their results and specific capabilities so that they can be chosen in future works or research for various jobs.*

1. Introduction

Our goal in this project is to compare multiple deep-learning algorithms for time-series forecasting in the context of financial engineering. We aim to develop and evaluate the performance of three distinct deep learning architectures: Long Short-Term Memory (LSTM) [5], Transformer-based models [14], and Mixer-MLP [11] networks. These architectures will be applied to financial time series data to assess their suitability for predicting future price movements, a critical task in trading algorithms and investment strategies.

There are many advantages to time series forecasting with ML. One of those is the accuracy and flexibility of parameters. The best thing about using machine learning models is that no specific algorithms or formulas are needed, decreasing half of our effort. It reduces the effort in traditional ways of specifically segregating and studying data. Similar to Finance data, we can extend this to other data like Energy consumption [7], raw material consumption, predicting demand and supply chain ratios. It stands out of the time, increasing the speed of the computation and prediction, helping us to know about the future markets.

2. Brief Intro to time-series models

In this section, we provide an overview of the deep learning architectures and methodologies we plan to employ in our project:

Time-Series Forecasting. [6] Time series forecasting in financial engineering involves using historical time-dependent data to predict future values or trends in financial markets. The data typically includes observations of financial metrics such as stock prices, exchange rates and other economic indicators recorded over regular intervals.

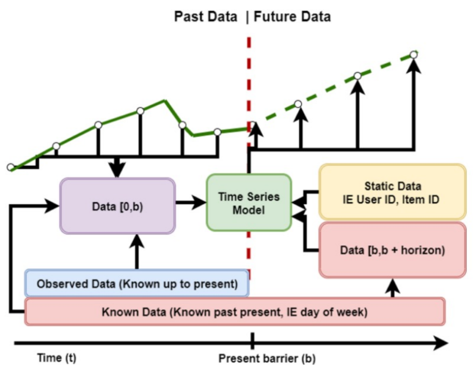


Figure 1. A picture of complete data flow in time forecasting $**$.

It plays a crucial role in decision-making for investors, traders, and financial institutions. As a result, many researchers are looking into new methods for better performance [10].

LSTM (Long Short-Term Memory). LSTM is a popular recurrent neural network (RNN) variant known for its ability to capture long-range dependencies in sequential data. We will implement LSTM-based models to analyze their effectiveness in modelling financial time series and forecasting future prices.

LSTM can be said to have originated in 1997 with the paper by S. Hochreiter and J. Schmidhuber [8].

The latest significant contribution to LSTM was made in 2014 by Hung Junyoung, Gulcehre Caglar, Cho KyungHyun, and Bengio Yoshua [9]. Here are some of the latest papers on LSTM [13] [5]

Mixer-MLP (Mixed Data and Label Perceptron). Mixer-MLP is a novel architecture designed for handling sequences and non-sequential data. Considering its unique structure and advantages, we will investigate its potential for financial time series prediction tasks.

The significant difference in Mixer-MLP compared to other algorithms and models is that Mixer-MLP doesn't use convolutions or self-attention layers and is entirely made up of Multi-Layer Perceptrons.

Transformers The transformer was originally introduced in order to solve the language translation in NLP problem which was earlier addressed using recurrence based models such as RNN and LSTM's where the entire "context vector" was generated based which was the final output given by the LSTM/RNN

In a simpler sense we can say that the use case of an transformer is to let the model itself decide it's recurrence on previous tokens rather than manually limiting the model to using some recurrence relation. We majorly used this paper for our transformer architecture [12] .

3. Experimental setup

To evaluate the performance of the aforementioned deep learning models, we will conduct a series of experiments using historical financial data. Key steps of our experimental setup include:

- Data Collection: Gathering historical financial data, including stock prices, trading volumes, and relevant economic indicators.
- Data Preprocessing: Cleaning and transforming the data to ensure it is suitable for training deep learning models.
- Model Training: Training LSTM, Transformer, and Mixer-MLP models on historical data to learn patterns and predict future prices.
- Evaluation Metrics: Employing metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to assess the accuracy of predictions.

** https://vitalflux.com/different-typesof-time-series-forecasting-models/

3.1. Datasets

For the datasets, we are currently using yfinance library in Python. Specifically, we are using the Apple dataset from the yfinance library.

3.2. Evaluation Criteria

We are exploring different strategies for evaluating our models. The standard k-fold cross-validation techniques cannot be applied due to the inherent temporal component of the datasets. Therefore, we decided to use backtesting, a time-based cross-validation method, where the model is trained and tested on batches of temporally sequential data. Some of the strategies that come under backtesting are:

- **Overlapped strategy** It is similar to the simple strategy, but the data used in the testing split of the previous chunk is reused in the training split of the next chunk.

4. Architectures

4.1. LSTM

4.1.1 About LSTM

- LSTM [3] is a special kind of RNN, capable of learning long-term dependencies. LSTM is comparatively better than RNN when it comes to remembering long term dependencies. It does so by using a memory unit called cell state, and three gates: input, output and forget gate.

- LSTM [16] is used in many applications like speech recognition, text generation, image captioning, language translation and Time Series Forecasting. Here we will use LSTM [2] for Time Series Forecasting of Stock Prices.

4.1.2 Architeture

The architecture of the LSTM is shown in the Figure 2. It comprises of various blocks which functions specific memory operations in a loop.

∗ https://data.nasdaq.com/
† https://data.worldbank.org/
‡ https://www.imf.org/en/Data
§ https://github.com/datasets/finance-vix
⅄ https://www.oecd.org/

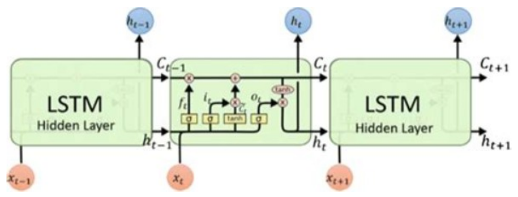


Figure 2. LSTM Architecture

- **Memory Unit**: Cell State, it runs through the entire network with minor linear interactions. It is like a conveyor belt, It can be thought of as the information that is passed along from cell to cell.

- **Input Gate**: it is responsible for the information that is allowed to enter the cell state. It is a sigmoid layer which decides what information should be let in.

- **Forget Gate**: it is responsible for the information that is allowed to leave the cell state.

- **Output Gate**: it is responsible for the information that is allowed to output from the cell state.

All the above gates process values between 0 and 1. So we had to scale the values between 0 and 1. We used *MinMaxScaler* for this purpose. The reason for using *MinMaxScaler* is that it is sensitive to outliers. So if there are any outliers in the data it will preserve that also information also. we can use the $inverse\_transform()$ function to get the original values back.

4.1.3 Experimental Results

We used LSTM to predict the stock prices of Apple we downloaded the stock data from yahoo finance. We used the data from 2001 to 2022 Out of which we used the last 500 days for testing and rest for training. We only used the Adj Close column for training and testing. We used 30 days of data to predict the next day's stock price. We used Adam optimizer and mean squared error as loss function. We used 2 LSTM layers with 50 units each and a dense layer with 50 units, the input shape is (30, 1) and the output shape is (1,). The code we developed for the LSTM can be found in the footnote †.

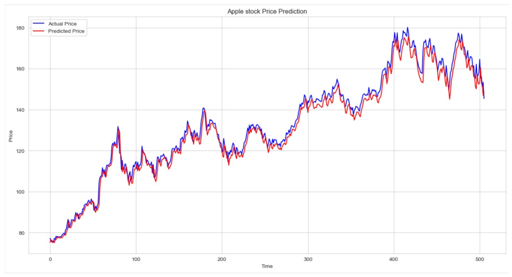


Figure 3. Some of the results of the LSTM model trained on yahoo finance dataset compared with the actual values

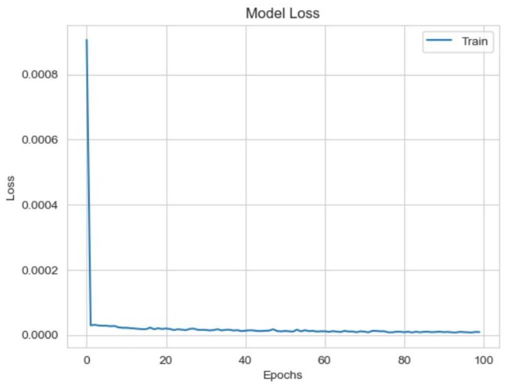


Figure 4. The training loss the LSTM model on yahoo finance dataset vs epochs

We got the following results:

- RMSE [1] is 2.3418861509782993
- Accuracy is 4.636934578937033
- Training loss is 8.918677849578671e-06

4.2. MIXER MLP

4.2.1 Mixer MLP For Vision

In vision architecture, we have two components in the Mixer block: ' channel-mixing' and 'token-mixing'.

- *Channel Mixing*: In channel mixing, we have a fully connected block over all the input channels.

- *Token Mixing*: A token/patch is a subsection of the input that is disjoint from all other tokens. And in token mixing, we have a fully connected block in the token

† Link to code for the LSTM: https://github.com/Prabhath003/DL-Project-Time-series-forecasting/tree/main/LSTM

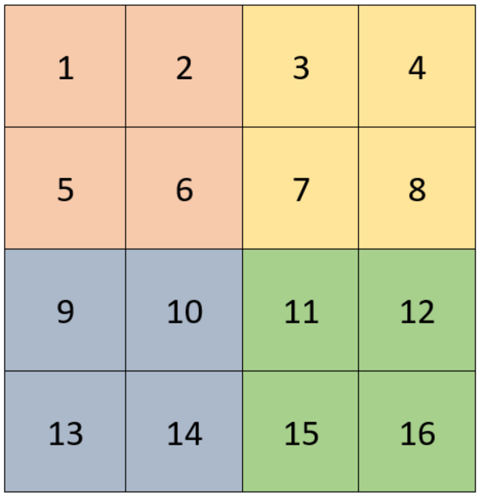


Figure 5. Example image.

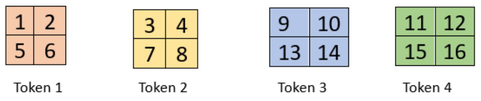


Figure 6. Tokenized image.

Below is an image 7 from the paper on Mixer MLP, which shows the architecture of Mixer MLP.

4.2.2 Mixer MLP for time-series

Contrary to vision models our goal for time series is a bit different, so we take a little bit different approach when tokenizing. In this case, suppose we have 16 time units of data with some n features. Then for each part we tokenize in the fashion as shown in the image 8

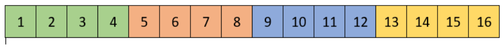


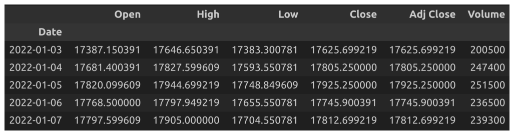
Figure 8. Tokenized image.

And the rest of the mixer architecture is almost the same with some modifications to accommodate the time series model.

4.2.3 Input Formatting

As the data we are using is not of images, so we formatted the data-set into two different forms such that we can send them in place of the images. We took some inspiration from CNN [15] models for time-series forecasting to surpass this step.

Original data-set format: Input data is indexed with dates with 6 parameters- Open, High, Low, Close, Adj Close, Volume as shown in Figure 9.



	Open	High	Low	Close	Adj Close	Volume
Date						
2022-01-03	17387.150391	17646.650391	17383.300781	17625.699219	17625.699219	200500
2022-01-04	17681.400391	17827.599609	17593.550781	17805.250000	17805.250000	247400
2022-01-05	17820.099609	17944.699219	17748.849609	17925.250000	17925.250000	251500
2022-01-06	17768.500000	17797.949219	17655.550781	17745.900391	17745.900391	236500
2022-01-07	17797.599609	17905.000000	17704.550781	17812.699219	17812.699219	239300

Figure 9. Original Data-set

The used different formats to send as input to MLP-Mixer such that we can produce better and accurate results:

- First way, we tried to make 1x30x6 input where we took 30 days and all 6 parameters as shown in Figure 10 to predict the next 31^{th} day values 1x6 parameters or a 1x1 single parameter. Here we used the patches of size 2x2, where will have a 45 patches, whose embeds are sent to N mixer blocks and goes through the following architecture. Here we tried tuning the number of parameters(6) and the number of training days(30).

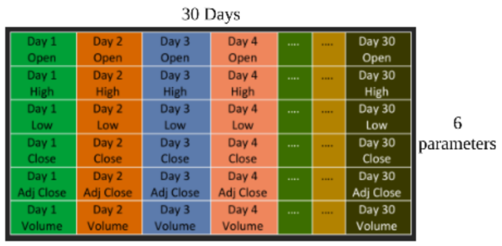


Figure 10. Pixels of Images created for the Input to the mixer MLP from the tabular data (Format 1)

- Second way, we tried to make 6x8x8, where we took 8x8=64 days of data to predict a single parameter of the 65^th day as shown in Figure 11. Here we used the patches of size 4x4, hence 4 patches, which follow the further architecture. Here we even tried tuning the channel size and number of days.

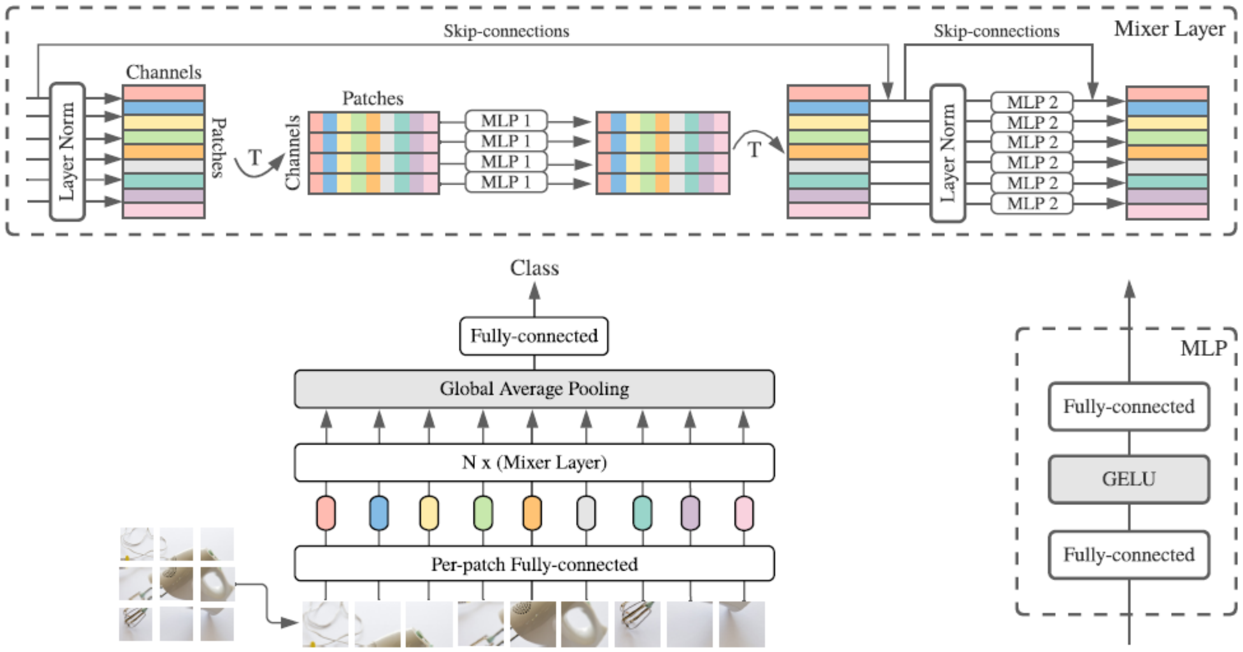


Figure 7. Mixer MLP architecture

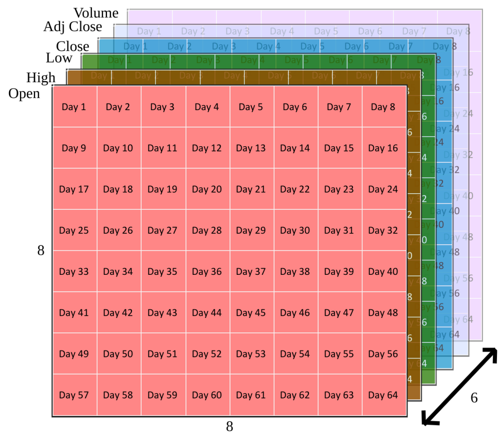


Figure 11. Pixels of Image created with parameters as channels for the Input to the mixer MLP from the tabular data (Format 2)

- We are up to trying other different ways to format the input and even modifying the architecture to increase the accuracy.

We have tried various configurations to produce the best results.

4.2.4 Experimental Results

We maintained the same hidden structure of the network, changing the input layer to handle the different input formatting. In the network, we used:

- path size 4
- 16 sequential mixer blocks

We trained our network Apple dataset from yfinance, which is calculated on 500 days and with values of Closing, Low, High, and Opening prices along with Volume. We focused on predicting the Closing price of a future date from the previous days. As mentioned above, we preprocessed the data to convert it into different formats. We trained the network separately for each input format. We used the common hyper-parameters for training:

- learning rate is 0.01
- batch size is 8 (here, a batch means taking eight from the pool of formatted data)
- ran for 2000 iterations
- After 2000 iterations, we used an early stop mechanism with a threshold of 0.01% for the last 100 iterations on training loss

You can follow the code from ∗.

We also trained with and without preprocessing data on a min-max-scaler between 0 to 1.

We first trained with Format 1(not our primary focus) input type. Here, we tried to predict the Future Closing

∗ Link to code for the MLP Mixer: https://github.com/Prabhath003/DL-Project-Time-series-forecasting/tree/main/MLP_MIXER price from the set of past Closing prices and no other pricing parameters were used. In Figure 12, the prediction is compared with actual values.

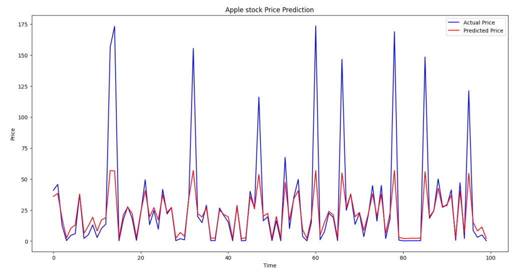


Figure 12. Some of the results of the MLPmixer model trained on yfinance dataset with format 1 type inputs compared with the actual value

4.2.4.1 Using only close without minmax

Here, we trained with the Format 2 input type. We tried to predict the Future Closing price of 65^th day from the past Closing prices of 64 days formatted into a 1x8x8 matrix, and no other pricing parameters were used. Figure 13 compares the prediction with actual values.

4.2.4.2 Using only close with minmax

Here, we trained with the Format 2 input type. We tried to predict the Future Closing price of 65^th day from the past Closing prices of 64 days formatted into a 1x8x8 matrix, and no other pricing parameters were used. We used min-max-scalar in data preprocessing to converge between 0 and 1 and rescaled up to get the predicted value, which is really helpful in training by decreasing the long jumps. Figure 14 compares the prediction with actual values.

4.2.4.3 Using close, low, high and volume without minmax

Here, we trained with the Format 2 input type. We tried to predict the Future Closing price of 65^th day from the past Closing, low, and high prices along with the Volume of 64 days formatted into a 4x8x8 matrix, and no other pricing parameters were used. Here, we did this without using min-max-scalar. Figure 15 compares the prediction with actual values.

4.2.4.4 Using close, low, high and volume with minmax

Here, we trained with the Format 2 input type. We tried to predict the Future Closing price of 65^th day from the past Closing, low, and high prices along with the Volume of 64 days formatted into a 4x8x8 matrix, and no other pricing parameters were used. We used min-max-scalar in data preprocessing to converge between 0 and 1 and rescaled up to get the predicted value. Figure 16 compares the prediction with actual values.

4.3. TRANSFORMERS

Main idea basic parts of transformer architecture:

4.3.1 Origins in Natural Language Processing(NLP)

The transformer was originally introduced in order to solve the language translation in NLP problem which was earlier addressed using recurrence based models such as RNN and LSTM's where the entire "context vector" was generated based which was the final output given by the LSTM/RNN .

This should theoretically also give priority to the earlier inputs contribution to the final result but due to the vanishing gradient problem this wasn't the case ,since the same weights are used for the various layers of the LSTM/RNN to accommodate variable sized inputs.

From this came the motivation to introduce transformers,where we instead decide to use an attention based mechanism using the same basic formulation discussed in class given below.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V \tag{1}$$

where Q,K and V are the query,key and value matrices respectively.

In a simpler sense we can say that the use case of an transformer is to let the model itself decide it's recurrence on previous tokens rather than manually limiting the model to using some recurrence relation.

4.3.2 Basic model

Figure 17 is an example of the basic model architecture.
It differs from an LSTM by only using attention based mechanisms and totally removing the long short-term memory recurrence pattern.

4.3.2.1 Attention and self attention

The output for each token is decided by an first an self attention mechanism which can generate outputs for each token in parallel and then an attention mechanism across the encoder and decoder which uses the outputs of the self attention encoder as the values while the output value of the decoder is used as the query,before this final output is fed to an FFN to get the next output token.

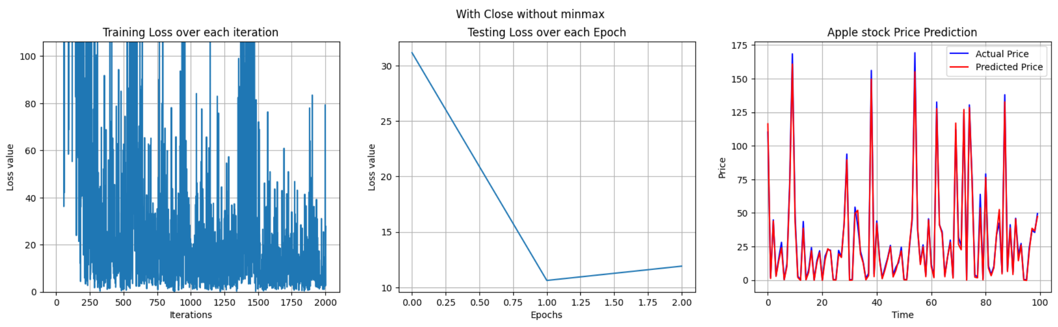


Figure 13. Prediction, Training loss and Test loss With only Close without minmax

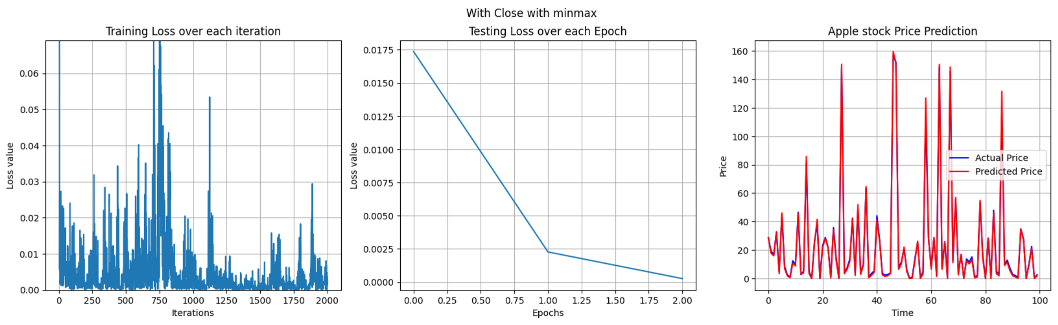


Figure 14. Prediction, Training loss and Test loss With only Close with minmax

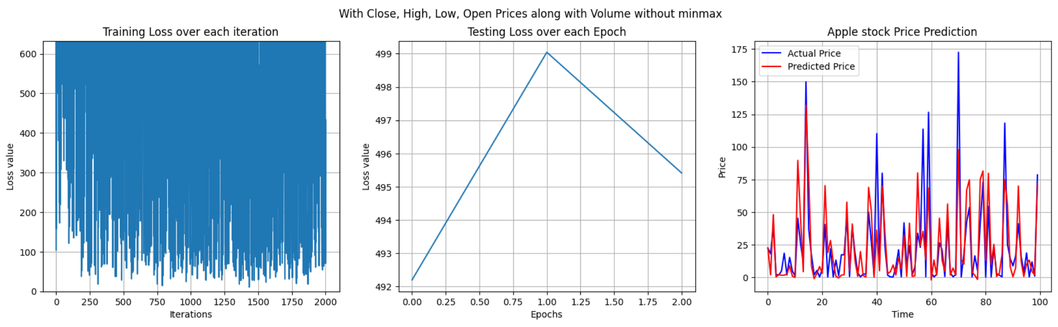


Figure 15. Prediction, Training loss and Test loss With Close,High,Low,Open prices without minmax

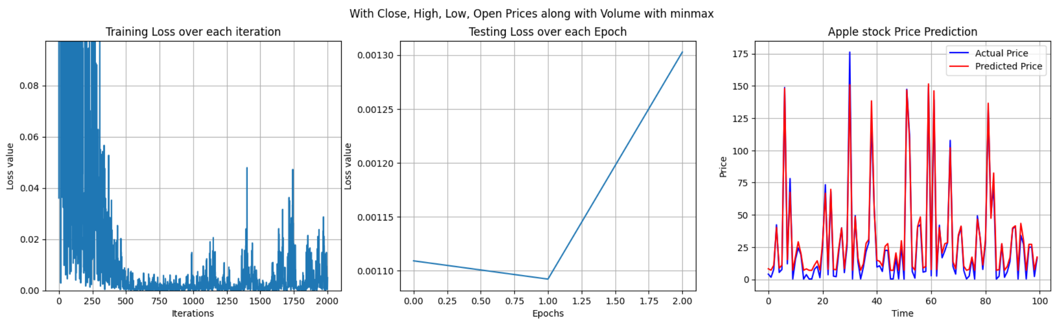


Figure 16. Prediction, Training loss and Test loss With Close,High,Low,Open prices with minmax

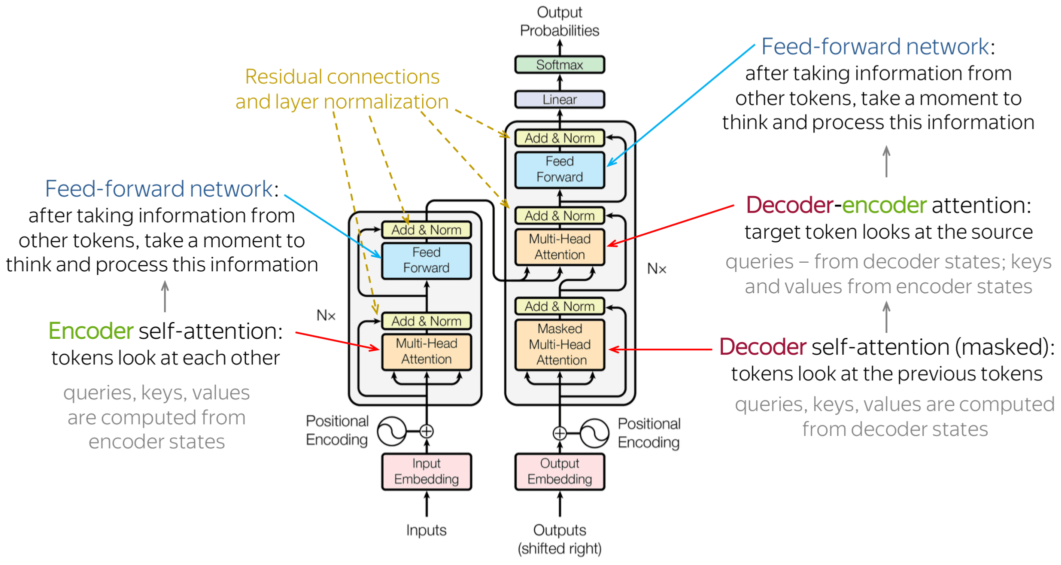

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Nx
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states

Figure 17. Transformer architecture for NLP

It is important to note that the weights used for self-attention within encoder and decoder and attention across the encoder and decoder are different from each other.

4.3.2.2 Positional Encoding

The architecture raises a question that if all the keys are processed for each query using the attention mechanism then how is it that the order information of various tokens is taken into account for ?

This problem is resolved by using something known as positional encoding where periodic sinusoidal waves of the same size as the encoded vector are added to it. This allows us to utilize information about positional inputs.

4.3.3 Superior parallel computation

Since the attention mechanism works using the same values of key and value ,multiple queries/tokens can be processed in parallel without any kind of sequential order imposed implicitly in RNN and LSTM's ,which in practical scenario's allows us to utilize GPU's much more efficiently and get better training times.

4.3.4 Transformers for time series

We initially experimented with decoder/encoder only transformers but upon further literature analysis decided to use the influenza model [12] as our main model reference in order to implement a full stack encoder-decoder model inspired from the paper's architecture for uni-variate time series forecasting.

Our model consists of firstly a linear layer to map the uni-variate data to an vector embedding of size 256(d model) which is an chosen hyperparameter followed by running a stack of 4 encoder layers followed by 4 decoder layers ,where during training the inputs to the decoder are taken as the right-shifted output and masking is employed to prevent cheating (ie. last input of encoder + first n-1 predicted outputs where n= size of output window).

Currently we are using an one step ahead prediction scheme due to which the necessity of sampling away from teacher forcing wasn't required ,but could be easily implemented as described by the paper [4].

4.3.5 Experimental results

We used the Adam optimizer with a learning rate of 0.0001 and the MSE loss to train our model consisting of 11,576,577 parameters for 100 epochs.

We ran experiments on two stocks apple and ibm where we aimed to predict the outputs of the next day using the data from the past 30 days ,these were our results.

for the Apple dataset the best testing loss we got was 0.00660 mean while for the IBM dataset the best testing loss we got was 0.000823.
We can see that the transformer performed much better on the IBM data set compared to the Apple data set,one reason for this might be that since at the beginning the apple stock prices were low for some years compared to their all time peak.

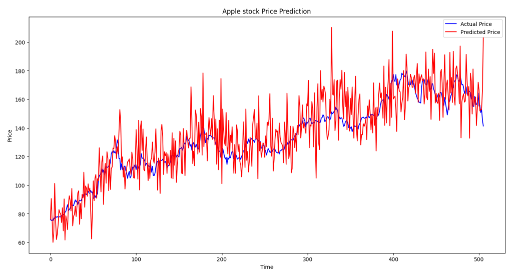


Figure 18. Apple stock predictions

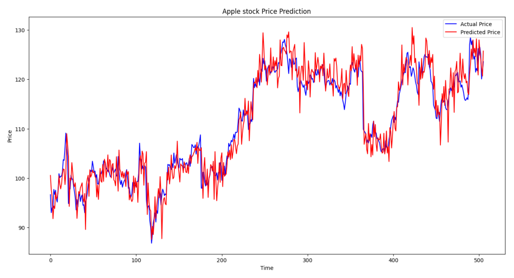


Figure 19. IBM stock predictions

Due to the small magnitudes after scaling and positional encoding the model wasn't able to learn too much useless information out of this part leading to poor performance ,meanwhile IBM stocks which didn't see a such a pricing boom were able to be learnt much more effectively. You can get the code for Transformers from &

5. Comprehensive results

Model	Training time	Number of Parameters	Best MSE	Inference Time
LSTM	3min	32k	3.1 $\times 10^{-4}$	3.2 $\times 10^{-4} s$ per prediction
MixerMLP	30min	33M	1.3 $\times 10^{-3}$	6.4 $\times 10^{-3} s$ per prediction
Transformers	7min 27sec	11.6M	6.6 $\times 10^{-3}$	3.17 $\times 10^{-4} s$ per prediction

Table 1. Comparing all models

& Link to code for the Transformers: https://github.com/Prabhath003/DL-Project-Time-series-forecasting/tree/main/TRANSFORMER

6. Conclusion

From our observations we can see that LSTM seems to be performing the best among all 3 in terms of MSE loss and time. But when we compare MixerMLP and Transformers MixerMLP seems to be slightly better in terms of MSE loss but Transformers outperforms it in terms of training and inference times.

References

[1] How to compute an accuracy measure based on rmse? is my large dataset normally distributed? https://stats.stackexchange.com/questions/22029/how-to-compute-an-accuracy-measure-based-on-rmse-is-my-large-dataset-normally-d. Accessed: 2023-10-23. 3

[2] Long short-term memory. https://en.wikipedia.org/wiki/Long_short-term_memory. Accessed: 2023-10-23. 2

[3] Understanding of lstm networks. https://www.geeksforgeeks.org/understanding-of-lstm-networks/. Accessed: 2023-10-23. 2

[4] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015. 8

[5] Lindemann Benjamin, Müller Timo, Vietz Hannes, Jazdi Nasser, and Weyrich Michael. A survey on long short-term memory networks for time series prediction, 2021. 1, 2

[6] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. *Machine Learning Strategies for Time Series Forecasting*, volume 138. 01 2013. 1

[7] Jui-Sheng Chou and Duc-Son Tran. Forecasting energy consumption time series using machine learning techniques based on usage patterns of residential householders. *Energy*, 165:709–726, 2018. 1

[8] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, 1997. 2

[9] Chung Junyoung, Gulcehre Caglar, Cho KyungHyun, and Bengio Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014. 2

[10] I.Sapankevych USA Nicholas and Ravi Sankar. Time Series Prediction Using Support Vector Machines: A Survey, 2009. 2

[11] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP Architecture for Vision, 2021. 1

[12] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case, 2020. 2, 8

[13] Hua Yuxiu, Zhao Zhifeng, Li Rongpeng, Chen Xianfu, Liu Zhiming, and Zhang Honggang. Deep Learning with Long Short-Term Memory for Time Series Prediction, 2018. 2

[14] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting?, 2022. 1

[15] Zhen Zeng, Rachneet Kaur, Suchetha Siddagangappa, Saba Rahimi, Tucker Balch, and Manuela Veloso. Financial time series forecasting using cnn and transformer, 2023. 4

[16] Zhi Bin Zou and Zihao Qu. Using lstm in stock prediction and quantitative trading. 2020. 2