Transforming Stock Market Predictions

Name: Aayush Agrawal
SUNet ID: aayush2k
Department of Computer Science
Stanford University
aayush2k@stanford.edu

1 Introduction

Stock market prediction is a problem that has been worked on over the last 60 years. As far back as 1959, Maury Osborne published a paper [1] that connected the theory of Brownian Motion to common-stock prices. In fact, the first mention of neural networks and stock market prediction dates all the way back to 1988 [2]. As machine learning models improve, researchers have continually evaluated their accuracy on modelling stock prices. In addition to the obvious financial incentive, the importance of the problem also stems from the academic urge to accurately model a phenomenon that seems so inherently random.

Ever since the proposal of the Transformer in the 2017 paper "Attention is All You Need" [3], the model has been applied successfully in a number of tasks. I wanted to investigate its performance in terms of stock market prediction. Thus, I implemented a transformer specialized for stock time-series and compared its performance to two models: ARIMA and Bi-LSTM (the former being the baseline). I implemented the transformer from scratch, including building out the multi-head attention component. The input to the algorithm is a time-series sequence with 4 price features (open, high, low, close) and the volume traded per day. We then use the aforementioned models to output a predicted a normalized closing returns for the next day.

I arbitrarily decided to focus on modelling the IBM stock closing returns. The reason for predicting the closing returns for the following day was because of the intended application of my project. Closing returns are the best input to a hypothetical trading bot that would be making investment decisions. I decided to focus this project on the ML model prediction; however, the natural next step is to create a bot that makes buy/short decisions based on the model predictions.

2 Related Work

The baseline for financial time-series forecasting tasks are auto-regressive models such as ARIMA (AutoRegressive Integrated Moving Average). In their 2014 paper [4], Ariyo et al. implemented an ARIMA model that focused on short-term stock price prediction of Nokia stock. This approach is still quite robust, as ARIMA models are still used today for time-series forecasting - particularly short-term forecasts which are its main strength. On the other hand, ARIMA's main drawback is that it performs quite poorly on long-term forecasts; additionally, it is computationally expensive.

Many of the drawbacks of linear models like ARIMA were addressed with the advent of deep learning models like LSTM (Long Short-term Memory). In their 2017 paper [5], Selvin et al. compared the performance of the LSTM model to that of the ARIMA baseline and found that the former outperformed the latter on the stock price prediction task. They argued that LSTMs were able to better capture the non-linear hidden dynamics of the stock market. The drawbacks of LSTM models are that they can take a long time to run (as they are built on Recurrent Neural Networks), require more data than ARIMA to train, and have many hyperparameters to tune. In more recent years, the BiLSTM model has been used more as it is able to capture more information than the vanilla LSTM model [6]. In fact, the current state-of-the-art stock price prediction models involve

LSTMs.

Building on the drawbacks of LSTMs (mainly the slow training times), the transformer was proposed as a model in 2017 [3]. Particularly, the model discards RNNs and uses a combination of convolutions and self-attention in order to both model local features and process global interactions. As a result, training speeds are increased significantly while still capturing complex, non-linear relationships. In their 2020 paper [7], Ding et al. used a hierarchical multi-scale Gaussian transformer to predict stock movement and were able to show that the model outperformed an LSTM-based approach. This was mainly because they were able to train their model with much more data due to the fast training times. I was most enamored by this approach, and thus decided to emulate a simpler version of it.

3 Dataset and Features

I used a standard stock market dataset from Kaggle [8]. The dataset keeps track of every stock in the NASDAQ, NYSE, and SP500, and records the Date, Low, Open, High, and Closing Prices, as well as Volume Traded in time series since 2005. All of the following features were used as input to my models. As closing returns were not provided, I calculated them myself by comparing the day's closing price to the previous day's closing return. The task was not time-consuming as I only needed to compute the returns for IBM stock. I normalized the returns (from 0 to 1) so that the outputs of my model were more intuitive, as well as keeping the option of using logistic regression open.

I separated the data into my training (11750 examples), validation (1469 examples), and test sets (1468 examples). Figure 1 visualizes the exact data split.

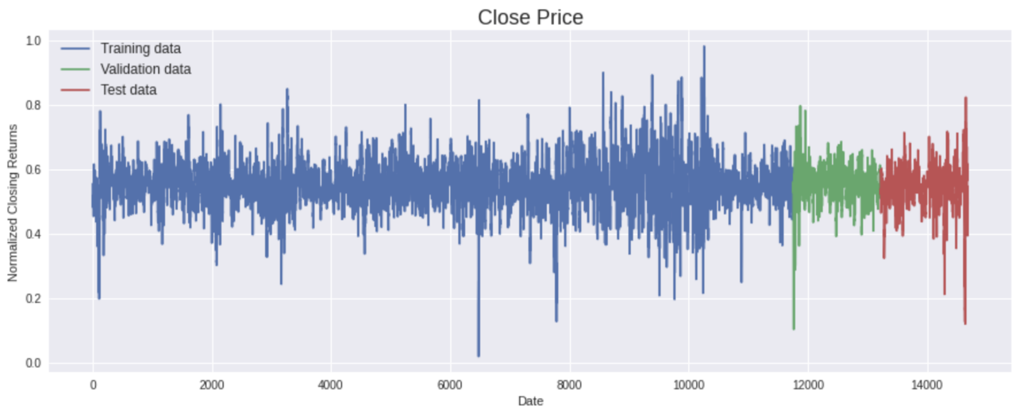

Figure 1: Data Set Split

Finally, I decided to create a time embedding vector as described in the 2019 paper Time2Vec [9]. Kazemi et al. propose a "model-agnostic vector representation for time" with both periodic and non-periodic elements. Instead of a positional encoding layer, which is normally used in transformers in NLP applications, my transformer begins with a "time encoding" layer that creates time vectors from the dates and uses the vectors as features capturing time. I used the same approach as highlighted in the paper to create these time embeddings.

4 Methods

4.1 ARIMA

An ARIMA (AutoRegressive Integrated Moving Average) model is a class of statistical models that specializes in forecasting time-series data. The model utilizes the dependent relationship between a data point and a number of lagged data points. Additionally, the model relies on taking the difference between observations at sequential time steps as it makes sure that the time series stationary. The "moving average" aspect of the model involves the dependency between a data point and a residual error from a moving average model applied to lagged data points. Every ARIMA model has three important parameters including p (number of lag data points), d (number of times raw observations are differenced), q (size of moving average window). The parameters determine what type of linear regression model is constructed.

4.2 BiLSTM

I wanted to use a model that would realistically be deployed for a stock market returns prediction task as my baseline, so I decided to implement a bidirectional LSTM (long short-term memory). A biLSTM is a RNN model which, as the name suggests, combines two LSTMs - the first of which takes the input in a forward direction, while the other does the same in a backwards direction. Unlike a standard LSTM, the information flows in both directions of the input sequence, which effectively increases the context available to the algorithm. [10]

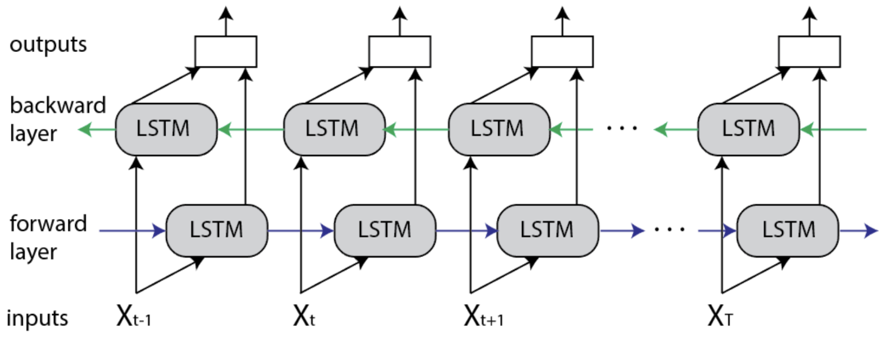

Figure 2: BiLSTM Structure

As suggested by its definition, it is necessary to first understand an LSTM. It is a special type of RNN that was created in order to deal with the issue of vanishing or exploding gradients in the vanilla-RNN. It is able to capture long-range dependencies in input, a trait core to its design. A unit of LSTM consists of a cell state, input gate, output gate, and forget gate. The key intuition behind its structure is that the cell state functions as a conveyor belt for information - making it easy for information to flow unchanged (which is why it is good at long-range dependencies). The LSTM can then manipulate what information flows through the LSTM using the three aforementioned gates. [10]

4.3 Transformer

The primary purpose of my project was to test the efficacy of the transformer in modelling stock-time series. Traditionally, the transformer has been used in NLP tasks. The model is composed of five types of layers. First, a positional encoding of the input is retrieved. Second, the embeddings are passed through a series of encoder blocks. These encoder blocks start with a positional encoding followed by a series of convolutions, a self-attention mechanism, and a feed-forward layer, all with skip connections and layer normalization. Next, the output of the encoder blocks is fed through a context-query attention. The output is then passed through three model encoder blocks, which outputs the final predicted closing returns. [3]

In this specific case, I want the positional encoding of the transformer to encode the notion of time. As I mentioned before, I created time embeddings in the "encoding layer". Thus, I was able to use time vectors in combination with the price and volume features as the input for my transformer. The other core part of the architecture is the self-attention mechanism. It consists of a Single-Head Attention and Multi-Head Attention layer. The self-attention mechanism is able to connect all time-series steps with each other at once, leading to the creation of long-term dependency understandings. The multi-head attention layer concatenates attention weights and applies a non-linear transformation; this allows the model to focus on multiple time-series steps at once. Finally, all these processes are parallelized within the Transformer architecture, allowing an acceleration of the learning process.

5 Experiments

5.1 Hyperparameters

I systematically explored the hyperparameter settings for each of the models that were used; for the sake of time, I only ran 5 epochs of training on the models for the hyperparameter tuning before testing on the validation set. When generating the final test errors, I trained each of the models for 30 epochs. For the ARIMA model, I conducted an exhaustive grid search for values 0, 1, 2, 3 for the three aforementioned core hyperparameters (p, d, q). After conducting the tuning on the validation set, I ended using $p = 1$, $d = 1$, $q = 2$. For the Bi-LSTM model, I chose a sequence length of 128 after conducting a quick ablation study comparing sequence lengths of 256 and 512 (all three values are reasonable norms for sequence length). The transformer model had significantly more hyperparameters, so I only tuned a subset of the parameters. I used a batch size of 32, sequence length of 128, $d_k = d_v = ff_{dim} = 256$, and 12 heads in multi-head attention.

5.2 Evaluation Method

In order to evaluate the models, I used MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error) scores. They are two of the most common metrics for regression tasks, and seemed appropriate to use. MAE is the average magnitude of error produced by the model, while the MAPE is how far the model's predictions are off from their corresponding outputs on average.

5.3 Results

Model	MAE	MAPE
ARIMA	0.0292	6.2077
BiLSTM	0.0264	4.5289
Transformer	**0.0208**	**4.4646**

The table above summarize the results of my models on the task of predicting normalized returns of IBM stock. We see that as the MAE and MAPE scores of the transformer were the lowest, the transformer was able to outperform my baseline model ARIMA, in addition to the Bi-LSTM model. In Figures 3 and 4, we see the graph visualization of the BiLSTM and transformer models making daily predictions, respectively. Even qualitatively examining the graphs, we can see with the naked eye that the transformer model is able to fit the real-world data much more closely.

Model	Correct Prediction %
ARIMA	45
BiLSTM	56
Transformer	**63**

The table above summarizes more qualitative results that examines how often the models were able to correctly guess whether the closing returns of the following day would be positive or negative (thus, absolute value of the prediction is irrelevant). This is very significant it pre-supposes my follow-up project: a bot that buys or shorts depending on the output of the ideal ML model.

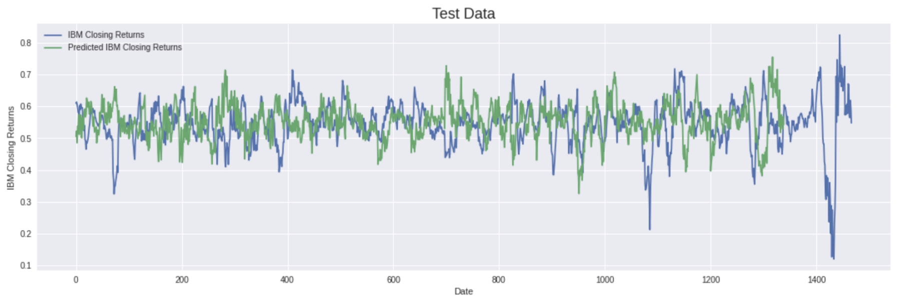

Figure 3: BiLSTM Model Results

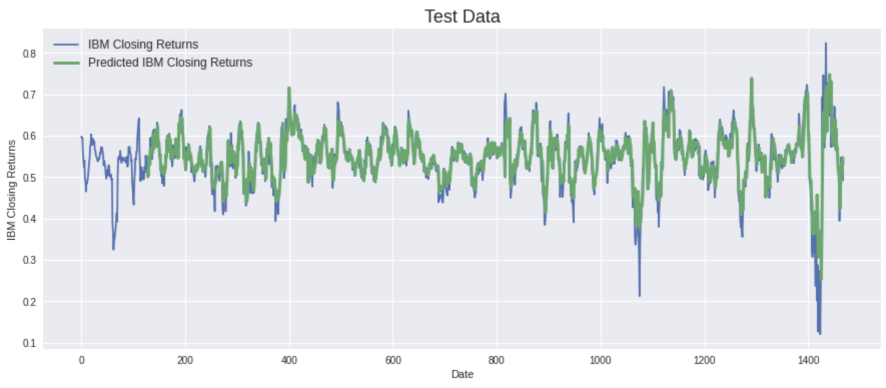

Figure 4: Transformer Model Results

5.4 Analysis

The results of my experiments roughly followed my hypotheses based on research. While ARIMA functions well as a baseline, it was not able to capture the complexity of the stock prices due to it being a linear model. The BiLSTM model fared much better; however, the predicted line was much more erratic as can be seen in the figure 3. Comparing the training error to that of the test error, it was obvious to see that the BiLSTM model I had implemented was overfitting the data. Additionally, the BiLSTM model took the longest to train. It is important to note that the current state-of-the-art models involve BiLSTMs packaged with other models; something I was not able to implement myself. Finally, the transformer seemed to fare the best at the prescribed task. It overfit the data to a much less extent, which can be attributed to the use of dropout. Additionally, the model also was able to capture short as well as long term patterns much better - as can be seen in 4.

One very interesting observation was that all of the models were terrible at making predictions around large peaks or troughs in returns. In economic terms, these would be classified as large rallies or crashes. In the test set, the obvious market crash was the one that occurred in March of 2020. As can also be seen qualitatively in the graphs, the models are unwilling to predict returns of very high magnitude. The transformer was able to predict some values with very large magnitudes, albeit with a large prediction delta on days with extreme daily change rate. This phenomenon does make intuitive sense, as it would be hard to predict outlier events without any other external data such as tweets (which are a very common method of trying to predict the stock market [11]).

6 Conclusion

In this project, I explored three disparate models and their performance on a stock market regression task; specifically, I tried to predict the closing returns of IBM with just time-series stock price and volume data. I used ARIMA as the baseline model, as it is the standard model used for time-series forecasting. I implemented both a BiLSTM and novel transformer that utilized time embeddings to process time-series data. Surprisingly, the transformer was able to perform better than the BiLSTM, which did better than the baseline.

As I have referenced to throughout the report, my ideal next step would be to create a stock market trading bot that would make buy/sell decisions based on the outputs of the model. Additionally, if I had more team members or more computational resources, I would have preferred to have conducted a more thorough hyperparameter search for my models. Also, using external data such as tweets or Facebook posts as input to the models would have helped to reduce overall loss; I suspect this would help with predicting more outlier events. Regardless, I learned a great deal about time-series forecasting in this project - and don't intend to stop my exploration of stock market prediction.

7 Contributions

This final report was worked on solely by Aayush Agrawal, with much-appreciated help provided by class CA Kamil Ali.

References

[1] Maury FM Osborne. Brownian motion in the stock market. *Operations research*, 7(2):145–173, 1959.

[2] Halbert White. Economic prediction using neural networks: The case of ibm daily stock returns. In *ICNN*, volume 2, pages 451–458, 1988.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Adebiyi A Ariyo, Adewumi O Adewumi, and Charles K Ayo. Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 106–112. IEEE, 2014.

[5] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017.

[6] Wenjie Lu, Jiazheng Li, Jingyang Wang, and Lele Qin. A cnn-bilstm-am method for stock price prediction. *Neural Computing and Applications*, 33(10):4741–4753, 2021.

[7] Qianggang Ding, Sifan Wu, Hao Sun, Jiadong Guo, and Jian Guo. Hierarchical multi-scale gaussian transformer for stock movement prediction. In *IJCAI*, pages 4640–4646, 2020.

[8] Paul Mooney. Stock market data (nasdaq, nyse, sp500), May 2022.

[9] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[10] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time, 2019.

[11] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[12] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[13] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[16] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.