Sergei Danielian - 40124849
Tim Huang - 40068476
Yubin Sun - 39291517
Mathias Moelgaard – 47054759

CS175B - Neural Networks in Stock Price Prediction

Problem Statement

This project is supposed to analyze intraday market trends and predict when would be an optimal time to buy or sell. This project uses Neural Network models to train an agent to be able to profitably trade in a designated market sector. This project utilizes a Temporal Convolutional Neural Network (TCN) which takes in the *Open*, *High*, *Low*, and *Close* of every candlestick, given the current day, and analyzes at which price levels are key levels that the agent should buy or sell at. Then, when price returns to those levels, the agent decides which of those analyzed levels it should buy or sell at and for how long. The agent learns over many episodes what constitutes an important price level and the reliability that it can be traded.

As the team worked on the project, it became obvious that it may be simpler to just create a network capable of predicting the change of price in the next minute, given a set of previous minute data to work with. Instead of creating an entire framework to evaluate each transaction the network makes, we would be able to evaluate the network purely based on how far off the actual result the network predictions were. Moreover, being able to predict the movement of the market would enable the agent to find the optimal buying and selling times.

Previous Works

At first, we did some research into LSTMs and found that there were already a lot of projects using LSTMs in the field of Stock Market prediction. We also found that LSTMs are not quite as robust in this field or at least do not handle long-term memory efficiently. After more research, we found that such inefficiency is probably due to the fact that LSTMs are dependent on sequence-based data such as sentences, where the most recent data takes precedence and older data in the neural network tends to degrade, leading to a vanishing gradient problem. In Stock prediction, not only the most recent data is important, but also the big picture should be considered.

In comparison, a Temporal Convolutional Network (TCN) uses convolutions, thus making the vanishing gradient less problematic; so, we decided to use a TCN. A TCN has a longer effective memory history as well as longer memory requirements for training than a LSTM does. (Deng et al, 2019)

Though focusing on a TCN, we later reintroduced the idea of having a TCN for low-level handling of long and short term changes, and an LSTM stacked on top to handle long term changes which should in theory, take advantage of the greater speed of TCNs while mitigating the data storage problem of TCNs (Wan, Renzhuo, et al, 2019).

Through research it seemed that adding in an Auto-Regressive Integrated Moving Average (ARIMA) as a feature into the neutral network could be an interesting addition to try to improve the accuracy and increase the amount of features for the TCN to work with (Jung-Hua Wang et al, 1996). An ARIMA model is a statistical based model which uses time series data to forecast future prices. ARIMA or Auto Regressive Moving Average works through using a

number of lagged observations which can be given different weight based on how recent the observations are. ARIMA works well for removing and estimating noise of a stock.

In specifications, this project used *Keras* with *Tensorflow* as the backend; then we added libraries from *keras-tcn* which provided a TCN model to be built on.

Team Members and Division of Work

Sergei Danielian took the responsibility of gathering reliable clean data for a specified ticker symbol. He advised the team to go with ABT (Abbot Laboratories) and ATVI (Activision) stocks, with 1-minute data to train and validate our models on. This way, with 1-minute data, we will have plenty of data points to train and test our model on. Sergei also provided a guided learning algorithm that would help prevent overfitting and give the agent insight on optimal times to either buy or sell.

Mathias Moelgaard took the responsibility of creating the architecture of our network, agent's logic, normalization for pre-processing our data, as well as testing and exploring features.

Yubin Sun took the responsibility of creating the framework, *Trade-Platform* module, and provided debugging and various technical supports.

Tim Huang took the responsibility of optimizing the code, saving the models, and creating the graphs for the models to present data, as well as high level oversight of version control.

Experience in Coding the Project

The coding of this project had two phases. In the first phase, we created a framework for future work. A robust and well-considered framework is necessary here because it shapes the structure of the project. There were three major assumptions for the framework. The first one was to simulate the actions of real-life trading systems. The second was to provide maximum compatibility and flexibility for the machine learning agent. The third was to provide monitoring and error dictation. Based on such requirements, we developed a backtesting platform called *Trade-Platform* as the base of this project, and it was used throughout the project. The *Trade-Platform* module was designed to flexibly manage features such as the *clock*, which controls the pace of the machine learning agent, and *data feeding*, which feeds the machine learning agent data only at the correct time. These features ensured that errors, if any, made by the agent, can be detected early, thus easing the debug process and providing monitoring. This module was also critical in allowing the agent to Buy, Sell, or Hold its position in a trade, otherwise we would not be able to simulate a live market environment.

The next phase was developing the neural network based on the platform above. At first, the team decided to simply implement an LSTM since the LSTM performs well with time-series data. However, as mentioned in previous sections, in our research, we found that the LSTM, though being one of the first networks applied on time series data, has been outstripped by other networks such as the TCN. In addition to giving better performance than the LSTM, the TCN requires less memory and is able to retain memory. Thus, we chose the TCN for predicting stock market price movements.

As we started creating the network, we started to question if we should change our initial idea, a network that determines when to buy and sell stocks, to something that was easier to train and test on, i.e. simple prediction.

We started out our network by feeding it features of every minute's *open, high*, *low*, and *closing* prices for stocks to predict the next minute's opening price. We normalized these through

scalar operations as well as percentiles and the log change between minutes. Later, two more features, the average for the last seven and twenty-one minutes of opening prices, were added to help the network improve accuracy. When we got a better handle on the receptive field and use of dilations for TCNs, the models started showing improvements.

Results

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 12, 4)]	0	
lambda (Lambda)	(None, 12, 4)	0	input_1[0][0]
tcn (TCN)	(None, 24)	17976	input_1[0][0]
tcn_1 (TCN)	(None, 24)	17976	lambda[0][0]
add (Add)	(None, 24)	0	tcn[0][0] tcn_1[0][0]
dense (Dense)	(None, 1)	25	add[0][0]

Figure 1 - The structure of the first architecture

The first architecture we used was a set of two parallel TCN layers followed by a dense layer to get the result (Figure 1). From this model, we tested four different moments, or the number of minutes the network is allowed to look at to predict the next minute's change. The next minute's change is defined to be the percentage change from the current opening price to the next minute's opening price. The different moments we tried were 12, 15, 30, and 45 moments. We found that by using 30 moments, we were able to get predictions that were the most consistent with the actual movement of the stock, whereas 12 and 45 moments consistently underpredicted, and 15 moments would underpredict larger movements. (Figures 2-5).

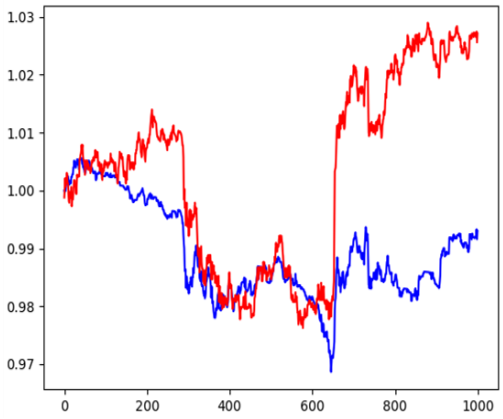

Figure 2. 12-moment predictions over 1000 minutes of test data.

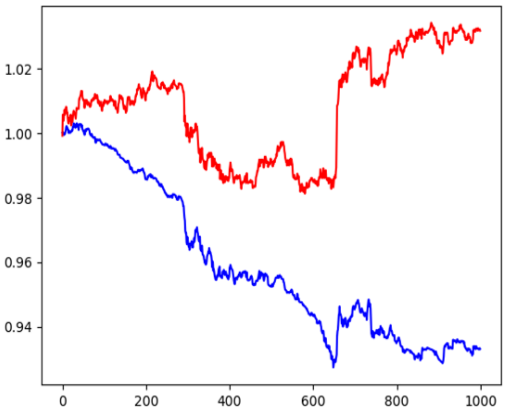

Figure 3.15-moment predictions over 1000 minutes of test data.

In all graphs in this document, X-axis is minutes passed, Y-axis is price as price divided by the price at time 0

Red is the actual price. Blue is the predicted price.

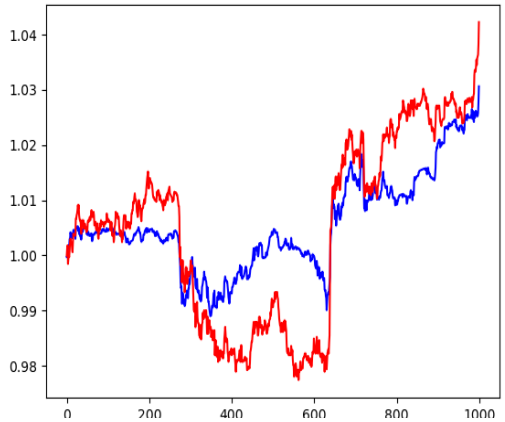

Figure 4. 30-moment predictions over 1000 minutes of test data.

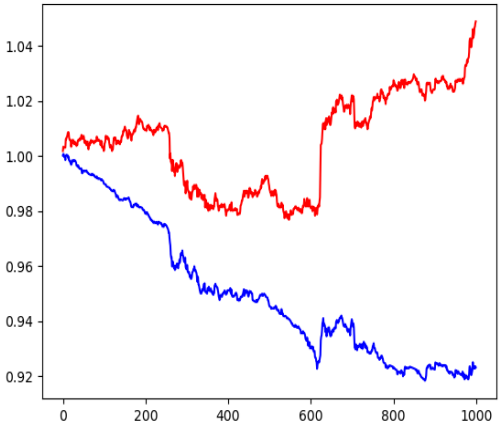

Figure 5. 45-moment predictions over 1000 minutes of test data.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 30, 4)]	0	
lambda_2 (Lambda)	(None, 30, 4)	0	input_2[0][0]
tcn_2 (TCN)	(None, 30, 60)	80700	input_2[0][0]
tcn_3 (TCN)	(None, 30, 60)	80700	lambda_2[0][0]
add_2 (Add)	(None, 30, 60)	0	tcn_2[0][0] tcn_3[0][0]
lambda_3 (Lambda)	(None, 30, 60)	0	add_2[0][0]
lstm_2 (LSTM)	(None, 5)	1320	add_2[0][0]
lstm_3 (LSTM)	(None, 5)	1320	lambda_3[0][0]
add_3 (Add)	(None, 5)	0	lstm_2[0][0] lstm_3[0][0]
dense_1 (Dense)	(None, 1)	6	add_3[0][0]

Figure 6. Architecture of a second model

Another architecture (Figure 6), that we tried was to put additional LSTM layers on top of the TCN layers that we already had, in essence attempting to combine the two types of layers to see if doing so would improve results. We found that if we test in 15 moments, the second architecture tended to overpredict (Figure 7). However, moving to 30 moments, the second architecture drastically underpredicted the movements (Figure 8).

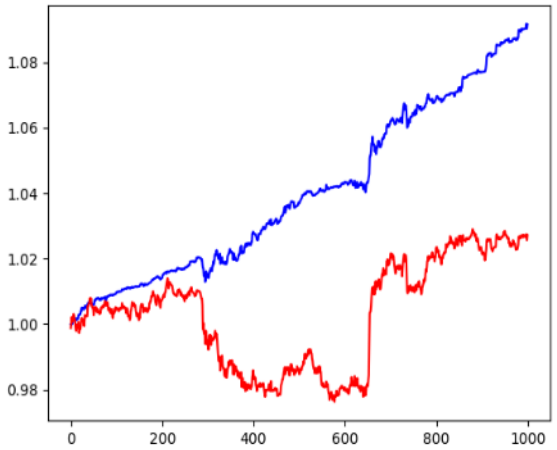

Figure 7. The second model performance on 30-moment predictions over 1000 minutes of test data.

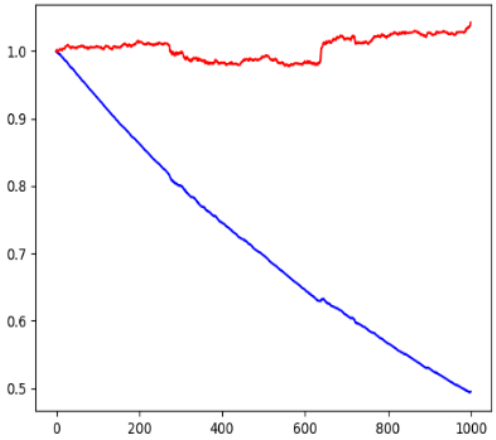

Figure 8. The second model performance on 30-moment predictions over 1000

After we trained the network on the ABT stock, we were curious as to whether it would be able to predict an entirely different stock. This led us to gather ATVI data, since Activision and Abbott Laboratories are in two completely different sectors of the market. These tests should show us how robust our models are. We found that while the first model did worse on the unknown data set, the second model did slightly better, granted the predictions were still very poor.

However, when we tested our trained ABT model on our Out of Sample Test Set for ABT, the performance of the model drastically increased. This implies that the model has learned the volatility and market movements of ABT, but cannot transfer its knowledge and weights to a different stock such as ATVI. These results and distinction are completely expected since different stocks have different price fluctuations given the volatility of that indice. Having the receptive field set to handle a moment of 16 we achieve the following on our Out of Sample Test Set for ABT, shown in Figure 9 – 12.

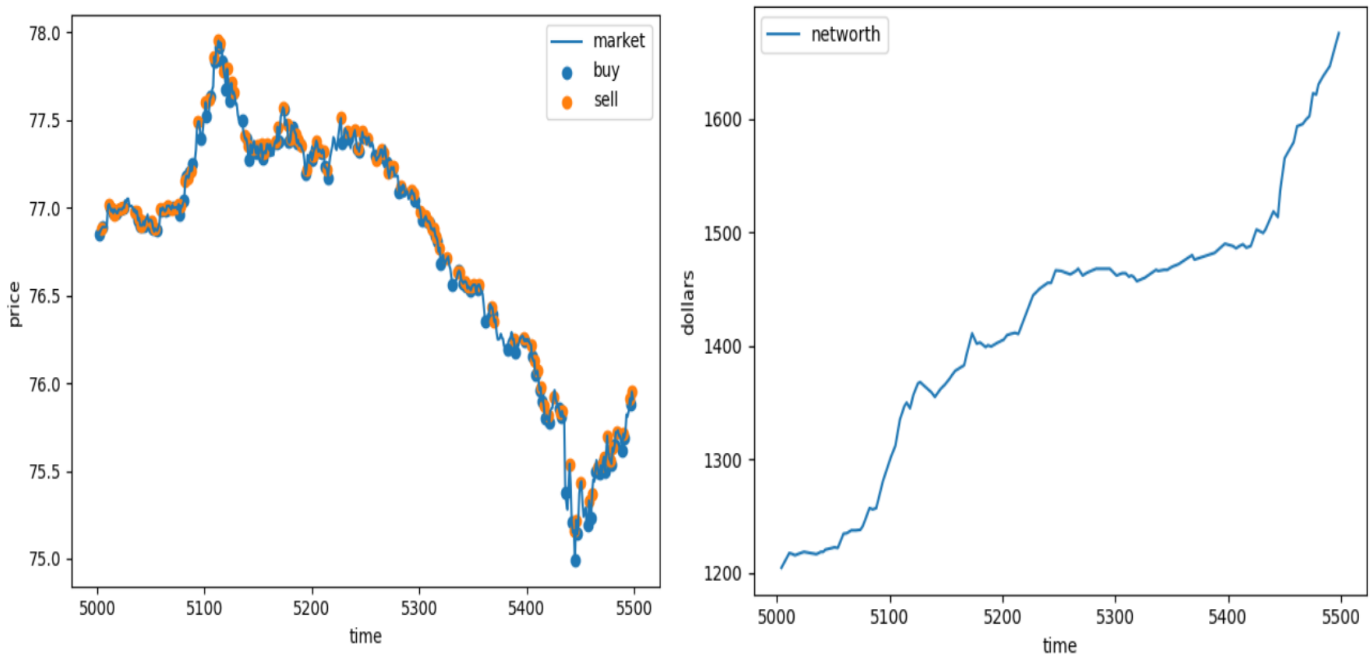

Figure 9: Model 1 performance on the market. Graph on the left depicts points in the live market where the agent either bought or sold at. Graph on the right depicts the net worth of Model 1. The model achieves a 61.4% accuracy when trying to predict if the next opening price.

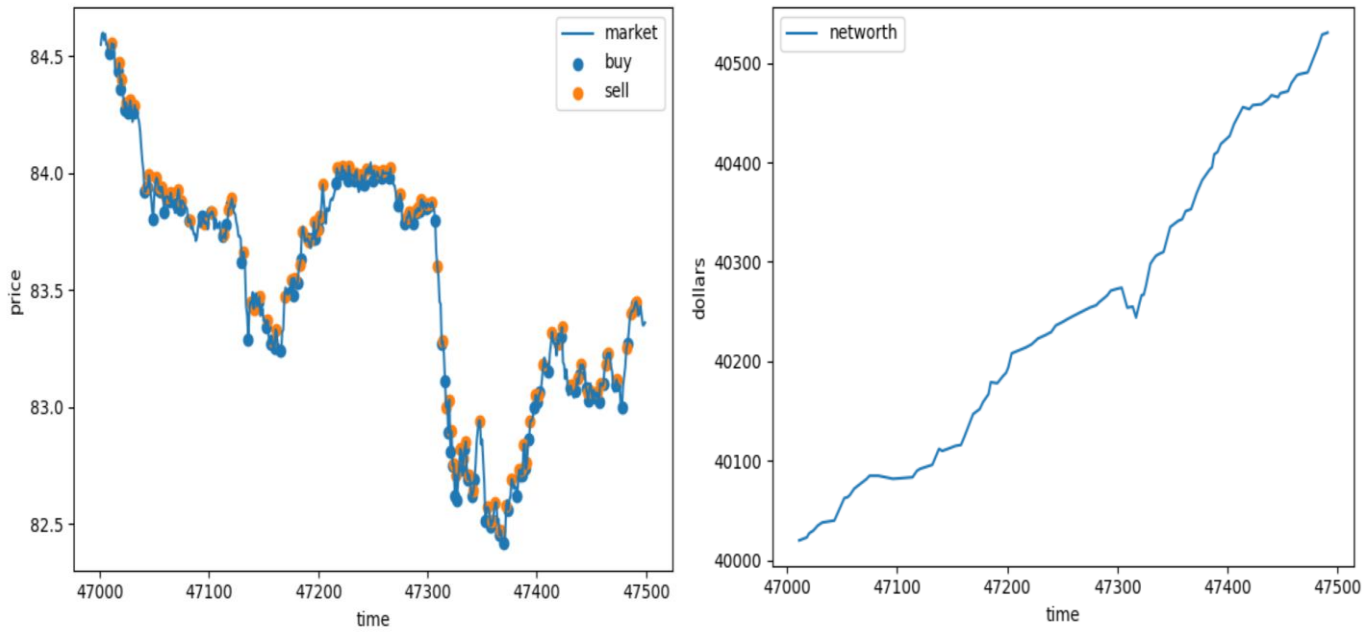

Figure 10: Model 2 performance on market. Graph on the left depicts points in the live market where the agent either bought or sold at. Graph on the right depicts the net worth of Model 2.The model achieved a 61.3% accuracy when trying to predict if the next opening price would go up or down.

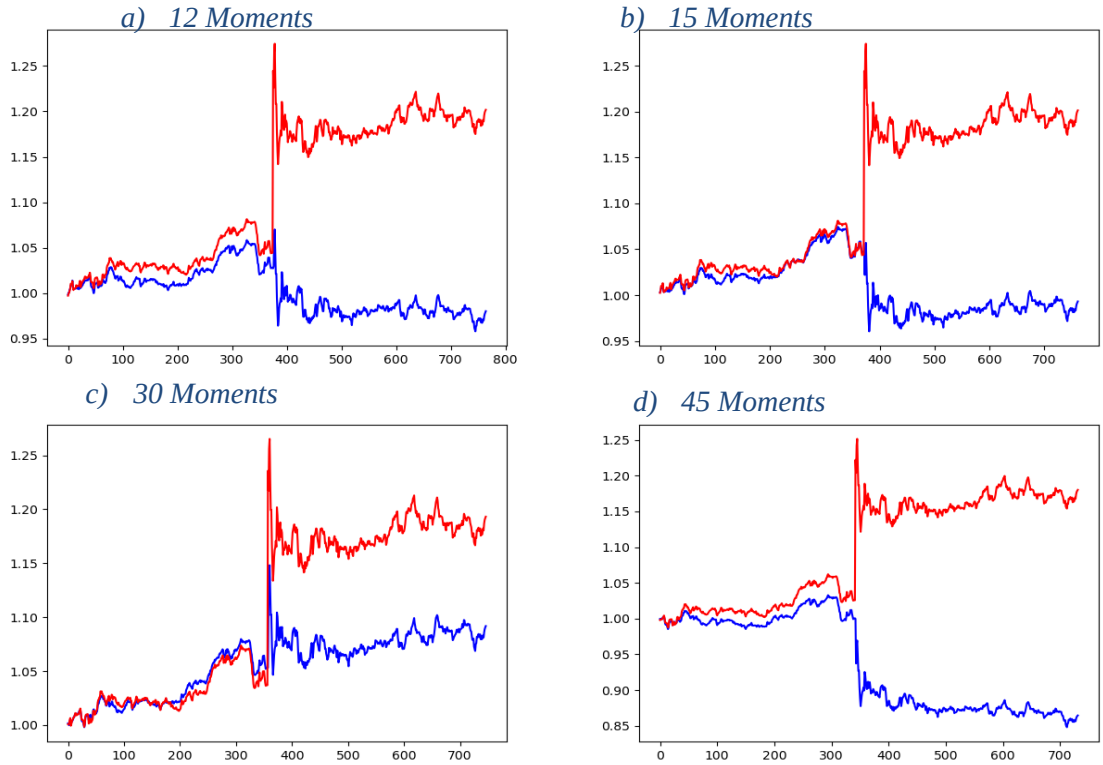

Fig 11.a), b), c), d) The first model results on different stock.

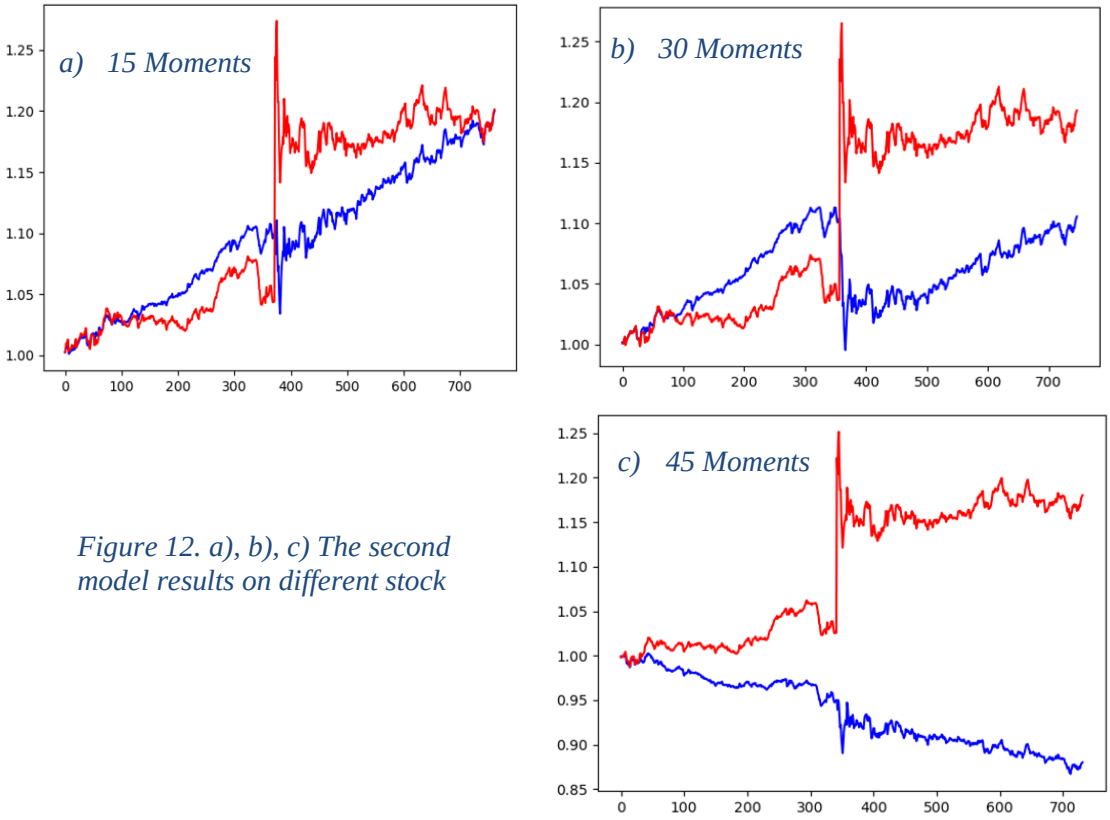

Figure 12. a), b), c) The second model results on different stock

Conclusions

Our approach of using a Temporal Convolutional Network to predict the stock market is promising as the agent performs well on the stock market, but does not perform well on other trading markets such as ATVI. The next step would be to train the model on multiple stocks across different market sectors, thus increasing the training data set size and decreasing the variance drastically. This would help the agent generalize well on future out of sample data sets and markets.

Work Cited

Downey, et al. "Predictive State Recurrent Neural Networks." *ArXiv.org*, 18 June 2017, arxiv.org/abs/1705.09353.

Yu, P., Yan, X. "Stock price prediction based on deep neural networks." *Neural Comput & Applic* 32, 1609–1628 (2020). https://doi.org/10.1007/s00521-019-04212-x

Jung-Hua Wang and Jia-Yann Leu, "Stock market trend prediction using ARIMA-based neural networks," *Proceedings of International Conference on Neural Networks (ICNN'96)*, Washington, DC, USA, 1996, pp. 2160-2165 vol.4, doi: 10.1109/ICNN.1996.549236.

Wan, Renzhuo, et al. "Multivariate Temporal Convolutional Network: A Deep Neural Networks Approach for Multivariate Time Series Forecasting." *Electronics*, vol. 8, no. 8, July 2019, p. 876., doi:10.3390/electronics8080876

Deng, Shumin & Zhang, Ningyu & Zhang, Wen & Chen, Jiaoyan & Pan, Jeff & Chen, Huajun. (2019). Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network. 10.1145/3308560.3317701.

"Keras-Tcn." *PyPI*, pypi.org/project/keras-tcn/.