Stock Market Price Prediction using LSTM RNN model

Muhammad Waqas   
Department of Software Engineering  
The University of Texas at DallasRichardson, Texas 75080, USA  
Muhammad.Waqas@utdallas.edu

Bijoy Prakash  
Department of Software Engineering  
The University of Texas at DallasRichardson, Texas 75080, USA   
Bijoy.Prakash@UTDallas.edu

Moniruzzaman Choudhury  
Department of Software Engineering  
The University of Texas at DallasRichardson, Texas 75080, USA  
Moniruzzaman.Choudhury@UTDallas.edu   
Nithya Shanmugam  
Department of Software Engineering  
The University of Texas at DallasRichardson, Texas 75080, USA  
Nithya.Shanmugam@UTDallas.edu

*Abstract*— The stock market prices generate sequential data in batches that is an ideal candidate for applying Machine Learning and Big Data management. There are several factors that affect the stock market prices which make it challenging for machine learning algorithms to train a model and these include irrational and rational trends, market talks, physiological versus physical factors and in the modern time even tweets from strong personalities can swing the prices with no pattern that a model can analyze and learn.

In this project we attempt to develop a RNN (Recurrent Neural Network) with memory popularly known as LSTM (Long Short-Term Memory). We aim to train this model with historical data for 1 ticker symbol from the last 5 years. With this trained model we will try to predict the live price and trend of the ticker on a stock market trading day.

Keywords—Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN), Machine Learning, Alpaca API.

# Introduction and background work

RNN model is a type of neural network that is used in predicting outcome in continuous data and can comprehend provided data better than any other machine learning algorithm [1]. LSTM model is a type of RNN model which can store details related to data for longer time frames and later utilizes that information to process, predict and classify it [2].

Investment corporations have been researching and investing to explore how AI or machine learning models can be utilized to improve the understanding of the stock market movement and accurately predict stock prices. The historical market data that is readily available and distributed by various APIs can be fed to Machine Learning models that can be tuned to give out its predictions based on the trends it analyzed in the historical data.

Historically the analysis and studying of market trends are done by humans in investment companies which provide users with research and possible stock trends, and it is a thriving market. Even though the consensus by every stock expert is that the prices cannot be predicted correctly because of the underlying market sentiment and various other factors that can affect the prices, it cannot be denied that the historical data available has a wealth of information to determine trends at least on an average day where external influences are low. There are investment and banking companies hiring machine learning analysts to build predictive model to prepare software that allows algo trading or predicting prices for users who subscribe to their services.

In this project, we first started working on getting real time streaming data for stock prices using Alpaca API [6]. We used the historical data provided from Alpaca to get data from stock market for development during off market hours and weekends.

Once we got the necessary ticker data from the API following is the plan, we formulated to get the whole system working. Data received as JSON from the stock market on a web socket is sent to a Kafka topic. This JSON is consumed by a Spark streaming application that reads data written to the above Kafka topic. The received data is sent as batches to a reading function that would process it and update the test data set that would be used by our custom LSTM model. Then we use the data to train the model and give out a predicted stock price for the next time step (5 minutes). Once this is done, we would replace the trained model with the new model and wait for the next data coming to the streaming application.

To achieve this, we had to build our custom LSTM model. To ensure we had a baseline to refer to, we did some analysis using the keras LSTM library and ran some numbers to ensure we have data to compare to. Next step was to build the model. We used the NumPy library for basic matrix calculations and array management. After numerous attempts we were able to build a basic LSTM model with 1 hidden layer that was giving out results with a fair accuracy.

While running the model on the test data we realized that the speeds observed were unrealistic for real world computation. With further analysis we realized that using GPU would give us better results. We used the numba library to make changes to the model code to make it suitable to be run on GPU. With the new LSTM module, we observed that the performance improved drastically. We decided to keep both versions of the code as the initial LSTM class is more readable and organized in comparison. Also, the GPU version needs more work to make it compatible with numba to observe more performance improvement.

# Theoretical and conceptual study

Human beings have the capability to retain memory and facts from the past that help us connect the dots for information we observe in the future. Machine Learning using Neural Networks does a great job in prediction of data that has no sequential dependency. But when there is dependency and data in the past retains value and significance for predicting data in the future, the neural network models lose their edge as they aren’t very good at retaining past information.

To address this issue RNNs were introduced. They are basically same as an ANN with the difference that the hidden layer information from the previous time step is fed into the hidden layer of the next time step along with the incoming data. This ensures that the future data gets some information from the past even though it fades away as time progresses. Figure 1 shows this information in a compressed loop form.

Diagram

Description automatically generated  
Figure 1: Loops inside RNN [5].

Figure 2 shows the same network but unrolled. We can see the flow of the data from hidden layer of previous timestep to the hidden layer of the next timestep.

A picture containing text, clock

Description automatically generated

Figure 2: Unrolled RNN [5].

RNNs find application in several fields of computer science such as speech recognition, image captioning, language modelling, the infamous autocomplete on mobile phones and various other applications. As RNN training progresses in time, the data from the past starts fading as data from the newer timesteps will have more influence. With this design RNNs usually retain memory of the very recent past and not the far past. This can become a problem when the model must make predictions based on data that was way back in the past. To overcome this problem a variant of RNN was introduced called LSTM.

A LSTM block includes gates and states that help with the process of retaining memory for information that the model would need to make predictions in the future. There are multiple variations of LSTM that were suggested and implemented. We chose to implement the one where we have 3 gates. Forget, Output and Input gate. Each gate has a very specific task to control the flow of information. The gates specifically help in controlling how much of the information it needs to remember. The forget gate would take the data and the previous hidden state. The gate would decide how much of the hidden state data needs to be forgotten based on the new input data that came in. This is achieved by passing the data through the sigmoid function that scales the data between 0 and 1 where 0 is to forget and 1 is to remember.

The input data and the hidden data are sent through a tanh function. This data is then sent through a sigmoid. The data from the sigmoid decides how much of the input data would be sent to the memory state of the block and how much would be ignored.

The output gate will work in the same way with the output data. It calculates the output based on the input, previous state data and the allocated weights. This data is passed through another sigmoid to filter out data that is deemed unnecessary by the model.

The dot product of tanh of the current memory state and the filtered output from the output gate gives us the output state that is fed to the next time step along with the current memory state that is calculated after removing data using the forget gate and adding relevant data using the input gate. Figure 3 depicts these gates in a pictorial fashion in one LSTM block.

Diagram, schematic

Description automatically generated

Figure 3: Long Short-Term Memory Unit diagram [7]

Input sequence and output sequence are utilized by the LSTM network to compute a mapping between them by utilizing the following equations iteratively from t=1 to T, to calculate the network unit activations:

Text

Description automatically generated

Figure 4: Weight definitions of LSTM gates and states [7]

The term Wn and Un in these equations depict the weight matrices and bn depicts the bias vector. We can see the formula for the activation function used for the input, the three gates for remembering or forgetting information and the two states (internal memory state and the output state) maintained in the block and sent to the next time step. The weight matrix for every gate is different and unique.

Following are the steps involved in the basic setup of a LSTM model:

1. Initialize the LSTM model. This would include all the weight matrices and bias values for the gates and the input activation function. Assign them random weights. Also add lists for storing previous states so that we can refer to them from the next time step.
2. Decide the number of epochs and batch size of the dataset. The dataset would be divided into batches and then sequenced through the model. This whole process would be repeated epoch number of times to train the model.
3. The training involves forward and backward propagation for each set of batch sequence. The forward propagation would be done for the complete batch sequence and then the model would do a backward propagation to go and redistribute the weights to minimize the error.
4. During backward propagation compute the difference of the target vs the predicted output. We use the L2 loss function to compute the loss. This error delta is passed back to the weights using the derivatives. Update the parameters using gradient descent and recalculate the weights and continue to the next epoch.

With the above LSTM model implemented the overall code architecture is depicted in Figure 5

Diagram

Description automatically generated

Figure 5: Code Architecture diagram

With the code architecture finalized let’s detail the actual system architecture. We have 2 phases in the project. The first is the training phase where we train the model using historical data. The next is the actual phase that is run during normal stock trading hours.

Figure 6 depicts the backend architecture for the system that is used to train the model using historical data during off market hours. The stock data is accessed via Alpaca API and sent to the model that trains itself to learn the patterns in the data. We used 5 years of historical ticker data for Apple (AAPL). The trained model is pickled using the python pickle library to be used later during prediction.

Diagram

Description automatically generated

Figure 6: Backend Architecture diagram

The application architecture involves more modules. Figure 7 illustrates the flow. The Alpaca API helps to get the market data every 5 minutes to our module. This data is written to a Kafka topic which is beind read by a Spark streaming module. This module has a trained model loaded from the backend traning module. Once the new data is received, we update the dataset to add the new entry. The model is retrained using the new entry that we got from the market. The retrained module is pickled and saved to be used for the next time step. Then we use it to predict the price for the next time step. Once we get the prediction we send it to a Kafka topic which is indexed by logstash and visualized in Kibana where we can get an idea of how the model is faring.

Diagram

Description automatically generated

Figure 7: System Application Architecture diagram.

# Results and analysis

To get an idea of the working of LSTM and how it handles and predicts data along with the sequencing of datasets we decided to work on familiarizing ourselves with the usage of LSTM library as well. This was done on a mix of datasets that included data from 7 days, 1 month and 5 years price data for Apple and few other combinations. The smaller datasets were used to aid with quick development work and the actual large dataset was used for training the model.

**Training Phase**:

The training phase involved creating a basic trained model that could be used during real time stock prediction. We decided to use 5 years of historical data of Apple stock prices that were taken 5 minutes apart. As a result, we had a huge dataset.

We tried training the model using this dataset and soon realized that its impractical. It was not able to complete even 1 epoch in 24 hours. This led us to explore more options and redesign our code to work with the GPU.

We used the numba module to redesign our LSTM model code. Using this we saw a very high peak in performance in comparison to the CPU code. We were able to complete 50 epochs in 24 hours and decided to use that as our base trained model.

**Dataset preparation and hyperparameter selection for prediction**:

Next step was to prepare the dataset for actual prediction for the price. We were planning to predict the stock price every 5 minutes. This placed the requirement of completing the intermediate training well within 5 minutes. We reduced the dataset size to 7 days of data and an epoch of 5. While running the GPU version of the code we were able to get results with 180 seconds (3 minutes) most of the time. We ran the CPU version in parallel with an epoch of 2 and got mixed results where sometimes the prediction overshot the 5 minutes mark.

**Testing Phase**:

We decided to run a test phase before trying out actual data on a trading day. We split the 5-year dataset and took a subset of it and fed it to our software. The results were promising as we could see that the predicted price was very close to the actual price. The trends at least were very similar.

A picture containing text, map, sky

Description automatically generated

Figure 8: Test run results

We can clearly see that the model did a fair job in predicting the trends. This is the result of the dataset that was run overnight, and data sent from the dummy dataset every 5 minutes

**Production Phase**:

The hyperparameters were already tuned during the test run to suit to whether it was using a CPU vs a GPU. We kept 2 instances of the software and ran it separately. One on a system with no GPU and another that has a GPU and used the modified LSTM instance for GPU support. The application was started around 9 am due to some hiccups with initializing the Alpaca APIs. Once everything was clear we let it run till market closed at 3 pm CST.

The Alpaca API was reliable, and we got data every 5 minutes and sent to the model for prediction. The model predicted the price and sent the predicted price and the actual price to Kibana for visualization.

Here are the results from the CPU version:

Chart, line chart

Description automatically generated

Figure 9: Prediction result visualization – CPU version

We can clearly see that the model did a below average job at prediction. The trends are not clear, and accuracy is below average. The breaks are result of the times where the model prediction overshot the 5-minute mark and skipped prediction for the next time step. All this data loss along with the low number of epochs clearly affected the overall accuracy of the system.

Here is the result from the GPU version

Chart, line chart, histogram

Description automatically generated

Figure 10: Prediction result visualization – GPU version

We can see a stark difference here. The trends are clearer in comparison. The model started off on the wrong foot with a big difference in predicted value. This is because the trained model was built 1 week before. It did not have the insight 1 week where the price had gone down. But we can see that with time the model caught up and started giving fairly accurate predictions. The trends are clear. However, we can still see at many places that the prediction lags the actual price when there is a big drop or peak in the actual price. The model does a fair job predicting the trends but certainly cannot be relied upon to do actual trading of the stock as stock markets are volatile and we can see sudden dips and peaks that the model clearly misses.

# Conclusion and future work

Implementing a LSTM clearly helped in understanding the internals of neural networks. To take a step-by-step approach we implemented a simple neural network and then a RNN and then moved on to implement a LSTM. Understanding the relevance of LSTM in sequential data processing allowed us to visualize how it can be applied to our use case of trying to predict the stock market prices.

Along with this we understood the importance of processors and GPUs and the effect it can have when we deal with deep learning along with large datasets. This clearly gave us an insight of how to approach a similar problem in the future that involves working with large datasets and deep learning.

The model we implemented did a fair job predicting. But there is a large room for improvement. With better hyperparameters and longer training on more historical data we can certainly expect better results.

As for future work we can propose the following:

1. Use a larger dataset for preparing the model. We used 5 years historical data. Dataset with 20 years would probably give the model more insights.
2. Optimize the LSTM model further. Use a better loss function and add optimization functions.
3. Change the model implementation to use GPU resources completely. Current implementation just uses it partially. A complete remodel would surely give better results.
4. Current trained model used 50 epochs. We can use a much higher number to train it further.
5. Currently training during live prediction uses only 7 days dataset and 5 epochs. We can optimize to increase these numbers and get better predictions. On the other hand, we can use 15 minutes time step instead of 5 minutes to add more epochs.

##### References

1. N. Donges, “A guide to RNN: Understanding recurrent neural networks and LSTM Networks,” Built In, 29-Jul-2021. [Online]. Available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm. [Accessed: 21-Nov-2021].
2. *Deep learning: Introduction to long short term memory*. GeeksforGeeks. (2021, September 29). Retrieved November 22, 2021, from https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/.
3. H. Sak, A. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling.” Jan-2014.
4. N. Adaloglou, “Recurrent neural networks: Building a custom LSTM cell,” *AI Summer*, 10-Sep-2020. [Online]. Available: https://theaisummer.com/understanding-lstm/. [Accessed: 27-Nov-2021].
5. “Understanding LSTM networks,” *Understanding LSTM Networks -- colah's blog*, 27-Aug-2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 27-Nov-2021].
6. <https://alpaca.markets/docs/api-documentation/api-v2/market-data/>
7. <https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9>
8. [arXiv:1909.09586](https://arxiv.org/pdf/1909.09586.pdf) [cs.NE] Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks