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 tensorflow 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

We tried running LSTM library model on different datasets with several parameter combinations. There are three levels of data being tracked in the graph. The blue line represents the close price of Apple stock based on the historical data. The red segment predicts the future price of the stock centered on the historical close price, while the green line indicates the actual state of the stock price on the ground.

From the graph, we can safely conclude that the real stock price went up while our model also predicted that the price of the stock would go up. This clearly shows how powerful LSTMs are for analyzing time series and sequential data.

For layer activation functions, we use *relu* and *sigmoid* algorithm while *adam*, *adamax,* and *sgd* for Keras API optimizers.

After we saw the results, we analyzed and concluded that graph numbers 3, 7, and 9 configurations are more realistic.

Graphical user interface, chart, line chart

Description automatically generated

Graphical user interface, chart

Description automatically generated

Figure 7: Graph 1 (dataset: apple\_data.csv).

Parameters we used in graph 1:

• **Loop\_back = 1**

• **Batch size = 20**

• **Activation = ‘relu’**

• **Epochs = 5**

• **Optimizer = adam**

• **Loss = ‘mse’**

Chart

Description automatically generated

Chart

Description automatically generated

Figure 8: Graph 2 (dataset: apple\_5min\_data.csv).

Parameters we used in graph 2:

**• Loop\_back = 1**

**• Batch size = 20**

**• Activation = ‘relu’**

**• Epochs = 5**

**• Optimizer = adam**

**• Loss = ‘mse’**

Graphical user interface, chart, line chart

Description automatically generated

A picture containing chart

Description automatically generated

Figure 9: Graph 3 (dataset: apple\_5min\_data.csv).

Parameters we used in graph 3:

**• Loop\_back = 10**

**• Batch size = 32**

**• Activation = ‘relu’**

**• Epochs = 20**

**• Optimizer = adam**

**• Loss = ‘mse’**

Chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 10: Graph 4 (dataset: apple\_5min\_data.csv).

Parameters we used in graph 4:

**• Loop\_back = 50**

**• Batch size = 24**

**• Activation = ‘relu’**

**• Epochs = 50**

**• Optimizer = adam**

**• Loss = ‘mse’**

Graphical user interface, application, Word

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Figure 11: Graph 5 (dataset: apple\_5min\_data.csv)

Parameters we used in graph 5:

**• Loop\_back = 100**

**• Batch size = 24**

**• Activation = ‘relu’**

**• Epochs = 100**

**• Optimizer = adam**

**• Loss = ‘mse’**

Graphical user interface

Description automatically generated with medium confidence

A picture containing chart

Description automatically generated

Figure 12: Graph 6 (dataset: apple\_5min\_data.csv)

Parameters we used in graph 6:

**• Loop\_back = 10**

**• Batch size = 32**

**• Activation = ‘sigmoid**

**• Epochs = 20**

**• Optimizer = adam**

**• Loss = ‘mse’**

Chart, line chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Figure 13: Graph 7 (dataset: apple\_5min\_data.csv)

Parameters we used in graph 7:

**• Loop\_back = 10**

**• Batch size = 32**

**• Activation = ‘relu’**

**• Epochs = 20**

**• Optimizer = adamax’**

**• Loss = ‘mse’**

Chart

Description automatically generated

Chart

Description automatically generated

Figure 14: Graph 8 (dataset: apple\_5min\_data.csv)

Parameters we used in graph 8:

**• Loop\_back = 10**

**• Batch size = 32**

**• Activation = ‘relu’**

**• Epochs = 20**

**• Optimizer = sgd**

**• Loss = ‘mse’**

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 15: Graph 9 (dataset: apple\_5min\_data.csv)

Parameters we used in graph 9:

**• Loop\_back = 15**

**• Batch size = 32**

**• Activation = ‘relu’**

**• Epochs = 25**

**• Optimizer = adam**

**• Loss = ‘mse’**

# Conclusion and future work

In this paper, we discussed what RNN and LSTM are and how LSTM works from a conceptual standpoint. We shared architectural images of our LSM project. We also displayed our results that we got by running LSTM library model. While working on this project we realized how powerful LSTMs actually are and if used properly, they can help tremendously in accurately predicting real world data.

**Future Work**: We are currently using only 1 hidden layer which is giving us fair accuracy. In the future we can work on increasing the hidden layers which can help in improving the accuracy.

##### 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>