Recurrent Neural Networks (RNN) for stock market prediction

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].

Huge investment corporations have been researching to explore what AI or machine learning models can be utilized to improve the understanding of stock market to help in accurately predicting stock prices. Machine learning algorithms can be fed all the previous stock price data to give out its predictions.

Workers in investment corporations’ study and analyze the data to predict stock prices. These workers stay up to date with stock market trends, examine the history of companies involved in stocks, watch news, and keep track of other data that assists them in making predictions about stock prices.

The general hypothesis is that stock prices cannot be predicted correctly because they are irregular but then we need to ask ourselves as to why are investment and banking companies hiring machine learning analysts to build predictive model to predict stock market prices. In our minds we mostly think that people in stock market walk on trading floors and are on call 24/7 to stay up to date with stock market prices. This will most likely change in the near future as we will see machine learning specialists using their computers to predict future stock market prices.

In this project, we first started working on getting real time streaming data for stock prices using Alpaca API. We also worked on finding an offline option to get data from stock market during Friday and weekends. We got the data from the stock market on a web socket, sent the data to Kafka producer, Kafka consumer then wrote it to Spark, and the Spark stream then displayed it to the console. Hence, we were able to get live data from the stock exchange via Kafka in Spark stream DF and cleaned the data to make it look organized. Our idea was for our application to subscribe to Alpaca API Web socket for trade data for a ticker, Kafka producer to read the data and send the JSON to a topic, Kafka consumer to read the data from the topic, Kafka consumer to send it to Spark Streaming API, and Streaming API to feed the data to the trained LSTM model and get the estimate on future stock prices.

Before even actually implementing the LSTM model, we first tried using the LSTM library. We used a CSV file to train that model and another CSV file to test that model. We then compared the real stock price with predicted stock price by creating a plot graph. After this step, we then actually tried implementing an LSTM model by coding from scratch. In the beginning our accuracy was not good but later on we used only 1 hidden layer in our LSTM model, and we made some changes in the model which improved our accuracy by a huge margin. We worked on creating two versions of LSTM models, 1 model runs on CPU and other one runs on GPU to get faster results.

In this paper we will discuss our results that we got from using our LSTM model and will analyze those results.

# Theoretical and conceptual study

We as human beings do not forget everything we have learned and start analyzing and understanding things from scratch. Generally, normal neural networks are not able to learn from past events which is their vital shortcoming. They are not able to process details from one step to another. This issue is addressed by Recurrent Neural Networks (RNN) because they have loops inside them which allows information to stay within them [5].

Diagram

Description automatically generated  
Figure 1: Loops inside Recurrent Neural Network [5].

In figure 1, *A* (piece of neural network) gets *Xt* as the input and outputs *Ht* as the outcome. The loop that is shown in this figure enables the details to move from one step to another. We can think of recurrent neural network as something that has several duplications of an identical network and each duplicated network is transferring a message to a successor [5].

A picture containing text, clock

Description automatically generated

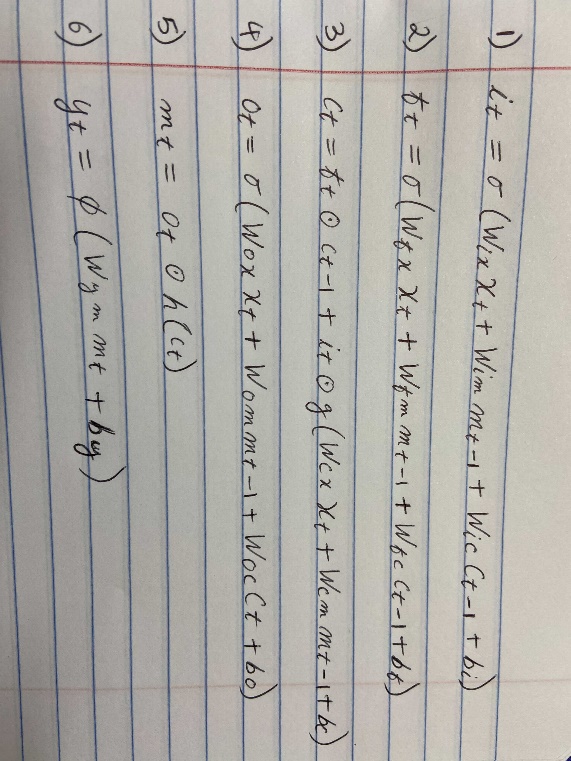
Figure 2: Unrolled Recurrent Neural Network [5].

Based on figure 2, we can come to a conclusion that RNN are related to lists and sequences. This chain-like nature of RNN is the natural architecture of neural networks. RNNs have been successfully applied in several subfields of computer science such as speech recognition, image captioning, language modelling etc. A very special type of RNN known as LSTM has also been successfully used for many tasks which is a much better version of RNN [5].

RNN are capable of making predictions based on past events that are recent. They can learn by using past details. However, if the gap between the past event and the current step is large then RNNs are not able to connect the information. So, in this scenario, LSTM helps in resolving this shortcoming. LSTMs are a special type of RNN because they help in preventing long-term dependency problem [5].

Memory blocks are the special units of LSTM inside the recurrent hidden layer. Memory cells are a part of memory blocks which have the ability to self-connect and store network’s temporal state. They also contain gates which are a special multiplicative units to control the flow of information. Output gate and input gate were included in each memory block inside the original architecture. The flow from input activations to the memory cell is controlled by the input gate. The flow from cell activations to the rest of the network is controlled by the output gate. Memory block was later on expanded further by adding the forget gate to it which resolved the weakness that LSTM models had with operating continuous input stream data that is not divided into arrays. Memory cell’s internal state is scaled by the forget gate before including it as input to the cell through the connection of the cell that is self-recurrent, thus adaptively forgetting or rebooting the cell’s memory. Furthermore, to learn accurate timings of the outputs, peephole connections was included inside the modern LSTM architecture from its internal cells to the gates in the same cell [3].

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:



The term W in these equations depict the weight matrices, for example, from the input gate to the input, *Wix* is the matrix of weights. For peephole connections, *Wic*, *Wfc*, *Woc* are the diagonal weight matrices. Term *b* denotes bias vector, sigma depicts the logistic sigmoid function, and *c*, *f*, *i*, and *o* are cell activation vectors, forget gate, input gate, and output gate. All of these have the same size as the cell output activation vector *m*. Symbol dot is the element-wise product of the vectors, *h* and *g* are the cell output activation functions and cell input. Normally, *theta* and *tanh* are utilized as softmax and network output activation function [3].

Equation 1 is the input gate. Weight matrices used in this equation embody the memory of the cell. The input *Xt* used in this equation is in the current input timestep, whereas *c* and *m* are indexed with the preceding timestep. Every matrix *W* in this equation can be considered as a matrix multiplication or a linear layer. With this equation we can take several linear combinations of *x*, *m*, *c* and match *m* and *c* with the dimensionality of input *x*. In PyTorch deep learning framework, the hidden state parameters are conceptualized as the dimensionalities of *c* and *m*. Hidden states used to be previously referred as neurons but that term is now deprecated. Furthermore, the term *b* also called bias vector, is a fragment of the linear layer and is a trainable vector. The value that we get as the outcome is in the dimensionality of the hidden and cell vector. Thus, we get a non-linear activation function to initiate non-linearities which permits the learning of more complex representations after three linear layers from different inputs. Sigmoid function is mostly used in this scenario [4].

Equation 2 is the forget gate and it is very similar to equation 1. One thing to note here is that the weight matrices in this equation are different. This implies that we would receive a different set of linear combinations which will give different results. We may want to apply different steps while modelling even though the equations might be similar [4].

Equation 3 is the new cell vector. This equation is another linear combination of the hidden and input vectors which makes it different from the previous equations. Purpose of this equation is to keep track of previous states. In this equation we filter the information of the new cell by implementing an element-wise product with the input gate vector *i*. After this the forget gate vector becomes operational. Element-wise vector multiplication is first performed with the previous cell vector rather than just appending the filtered input details [4].

Equation 4 can be called as the almost new output. New computed cell state will be utilized in this equation as compared to equations 1 and 2. This equations helps in computing the output vector of a single cell in a specific timestep [4].

Equation 5 can be called as the new context. This equation is used to compute the new hidden state. The aim of this equation is to combine the cell vector *ct* with the computed output vector *ot*. Equation 4 is just creating an output but in this equation, we are also injecting the vector called cell [4].

A picture containing text, clock, sign

Description automatically generated

Figure 3: Long Short-Term Memory Unit diagram [4].

In figure 3, it is shown that a cell receives another cell and hidden state as inputs from the preceding timestep. And the current timestep sends the input vector to a cell. A hidden state and the new cell state are outputted by each cell inside LSTM. These outputs are utilized in processing the subsequent timestep. The output of the cell is its hidden state.

Following are the three architectural diagrams of our LSTM project:

Diagram

Description automatically generated

Figure 4: Backend Architecture diagram.

Diagram

Description automatically generated

Figure 5: Code Architecture diagram.

Diagram

Description automatically generated

Figure 6: 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].