Using Vanilla Recurrent Neural Networks for Time-Series Stock Prediction

Jeffrey Muhammad
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, U.S.
jfm170000@utdallas.edu

Utsav Malik
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, U.S.
uxm170000@utdallas.edu

Soham Mukherjee
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, U.S.
sxm180113@utdallas.edu

Shanmugathevan Kanagaraj
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, U.S.
sxk180011@utdallas.edu

***Abstract*—In this paper, we describe how we implement Vanilla Recurrent Neural Networks to predict values of a time series. We implement the Recurrent Neural Network in Python and use a dataset of the open, high, and low values of a stock over thousands of days to predict the stock value given arbitrary parameters over time. In this paper, we discuss the mathematical theory behind Recurrent Neural Networks and their related algorithms, as well as how to use them in the context of predicting stock prices. We also discuss the fundamentals of stock analysis and predicting stock prices using market data. Next, we look at results from our Recurrent Neural Network implementation in Python and compare our predicted stock prices with historical market data. Finally, we analyze these results and discuss the reasoning behind shortcomings or prediction error in our implementation and Recurrent Neural Networks in general. (*Abstract*)**

Keywords—Recurrent Neural Network (RNN), Neural Network (NN), Long Short-Term Memory (LSTM), Backpropagation through time(BPTT), Root-Mean-Square-Error(RMSE)

I. Introduction and Background Work

Recurrent Neural Networks, or RNNs for short, are a class of artificial neural networks which utilizes sequential data. It utilizes previous outputs as inputs. It has also played a big role in deep learning. Because it is sequential [1], its biggest application by far lies in time series forecasting, which is done step-by-step, while maintaining an internal state that is either plotted or stored through the propagation of data through the neural network. In this paper, we utilize a neural network to predict stock prices thanks to the use of a Kaggle dataset [2], and output its results based on different parameters as well as a conclusion based on our results.

The RNN itself was first conceived in 1986 as a sort of amnesiac model. What that means is that in each subsequent layer of RNN, it tries to remember as much as possible from the previous layer whose output turned into input for the current layer. The RNNs utilize an algorithm known as backpropagation through time to determine the next output that a layer of an RNN will give [3].

Backpropagation, which is used in Backpropagation through time, is about going reverse through the neural network to find the partial derivatives of the layer's errors [4]. By going reverse through the neural network, you can subtract the partial derivatives of the error in relation to the weights from the weights. The neural network can "learn" by adjusting the weights up or down, which is based on the gradient descent using the derivatives.

Backpropagation through Time uses backpropagation for an unrolled Recurrent Neural Network. An unrolled Recurrent Neural Network is more like a sequence of neural networks, where the current component of the network is dependent on the prior component of the network. This is run through at every time step and goes backwards after the forward pass to pick up the partial derivatives and add (or subtract) them from the current weights.

Because of RNN's sequential nature, this makes it nearly perfect for time-series prediction, a common predicament as analyzing patterns over time is helpful for the prediction/classification of otherwise conventionally modeled data. Time-series prediction is commonly used in an abundance of areas – from predicting weather forecasting to pattern recognition, the possibilities of this field are extraordinary. One of its most interesting uses, however, is the prediction of stock prices.

We are students of the Introduction to Machine Learning class, who have worked on three projects concerning machine learning, which include linear regression, building and propagating data through a neural network, and calculating Jaccard distance of tweet clusters utilizing unsupervised learning. In this project, we will be discussing the use of a vanilla

RNN to predict stocks via feedforward networks that utilize backpropagation.

II. Problem Introduction (Introduction to Stock Analysis)

Introduction to stock analysis. People generally will resort to the use of stock analysis to beat the market. Fundamental analysis and technical analysis are two of the main types of stock analysis methods used in the industry today.

Fundamental analysis will focus of the "fundamentals" behind a certain stock, such as looking into the company itself and gleaming information such as "recent earnings, past earnings growth rates, projected earnings, multiples like PE ratio, debts and other financial and accounting measures"[5], in order to create a true price of the stock and depending on this value this stock can either be undervalued or overvalued, this method is usually used to predict the movement of a stock in the long term.

Technical analysis focuses on looking at the movement and behaviors of a certain stock, looking for "patterns in a stock price's historical movements, look for repeats of these patterns and make investment decisions based on whether these patterns emerge" [5]. Technical analysis is what will be focused on throughout this project and will be what is implemented in the project.

In the stock predication model created in this project we will utilize the close values of previous days in order to predict what the value will be for the closing price of a certain stock. To understand how stocks work in general we will first need to first understand what the open price, the high price, low price, and close price values represent. These are values used to describe a particular period.

- The open price will be the price that the stock during a particular period opens at or starts at.
- The high price will be the value that represents the highest price the stock was at during this certain time frame.
- Like the high price, the low price represents the lowest price that the stock was at during the specific time frame.
- The closing price is what the final price of that stock will be at the end of the specified time period.

In this stock prediction model, we have two options to predict the close price. We can either use a one-to-one model, where we use a set window of previous values of close to predict close price at the time beginning immediately after that point, or we can use a many-to-one model, where we will use all three of these factors, open price, high price and low price, in order to predict the closing price of a certain stock for a certain period of time. We have decided to begin with using only the close price in a one-to-one model to keep implementation more straightforward.

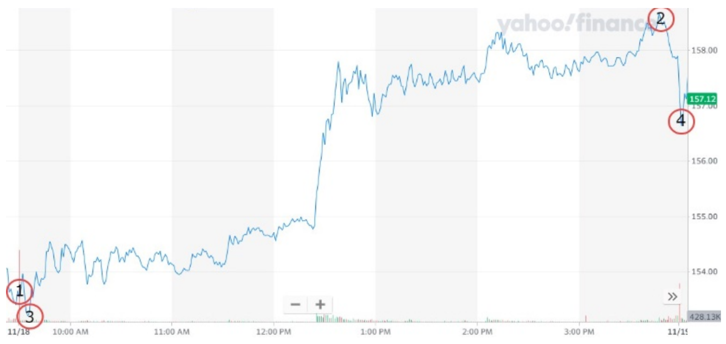

Fig. 1. Apple.Inc Daily Stock Chart via finance.yahoo.com

Fig. 1. shows the open price, the high price, low price, and close price values on a graph with a time period of one day. At the start of the day, whatever values the stock starts at will be the open price, this is seen in position 1 in Fig. 1. Whenever the stock price hits its highest value during the period, being one day, will be high price, as seen in position 2 in Fig. 1. Vice versa, the low price will be recorded by finding the lowest value during the one day, as seen in position 3 in Fig. 1. And finally, similar to how the open price is found, by seeing the price that the stock opens at, the closing price will be found by looking at what price the stock ended the day at, as seen in position 4 in Fig. 1. To train our model we will be giving our model the closing price for the past certain number of days and using these values we will predicate the closing price. In that sense, our model is a one-to-one model, as we take in this one attribute to produce one given output.

III. Theoretical and Conceptual Analysis

Our problem primarily revolves around utilizing stock data to train a recurrent neural network which will tell us whether a stock is going to go up or down and depending on that tell us whether to buy or sell given the pattern of the stock.

Here, we will detail how the process works.

Firstly, we need to preprocess the data to prepare the data to be used in our algorithm. We only want to use attributes that are relevant from the dataset to predict the result through the algorithm. When preparing the dataset, we can also consider on what type of architecture we would like to use for recurrent neural networks. In our case, we used a one-to-one type of network, which we based on our data set. We used numerous data entries of closing price to train a model within a given day window (30 at the minimum) what the stock will be on the 31st. The data must be preprocessed in such way to maintain that type of model relation. In order to do, we dropped all relevant columns. Since date is an important attribute, we made sure that our data was also sequential to prevent arbitrary spikes and otherwise. After preprocessing the data, we started on the actual training.

To train the network, we will need to do a forward pass to find errors of the training data. First, we will need to establish our hyperparameters. For the sake of this program, we used the maximum iterations (epochs), learning rate, stock window (the days before the prediction), dimensions of the hidden layer, and dimension of the output (which is 1 in this case). For the sake of our backpropagation, we also had a backpropagation truncation value to limit timesteps, and maximum/minimum clip values to prevent our gradients from exploding.

Finally, and obviously most importantly, we initialized the three-layer architecture we would be using for this model. It consisted of an input layer with one variable, a hidden layer, and an output layer which produced one singular output.

$$Loss = ((actual - predicted \wedge 2)/2 \tag{1}$$

In order to keep track of our previous loss values for each iteration, we created a memory table to memorize the previous losses while computing the current. In order to compute the loss, we utilized a cost function, given equation (1), and summed it up after accessing each timestamp of the given data.

The reason for using a cost function call is two-fold. First, we could output the loss on the training data to keep track of whether it was properly descending. Second, we noticed during the gradient descent that sometimes, the values would stagnate around a particular position. For that reason, we calculated whether the values have stagnated by subtracting the previous loss from the current loss and taking its absolute value to check whether it was underneath said threshold. This was the creation of our convergence function, which if the model was finished training and had surpassed the threshold of 5 iterations, it would then terminate gradient descent. To branch off this, the convergence function was another condition to continuing the process of training, as we did not want to conduct redundant training.

Then, it came to training our model. At each iteration, we created a new previous activation array to keep track of the neurons in the hidden layers. Each iteration consisted of a forward pass with corresponding backpropagation. We will then consider the data from prior states of the network by adding the multiplication done recently to do the multiplication of the weights in the layer of RNN. Then the sigmoid activation function will use this value, and a multiplication is done of the sum of values with the weights between the output and hidden layers to determine whether the neuron is activated, and the input can go through. We also keep in mind that there is no need to pass the value through the activation layer at the output layer. Finally, we keep the data related to the state of the current layer and all the data related to prior stamps somewhere. Let us delve further into these algorithms.

A forward pass is done to modify the weights with our training data as we feed it through the network. During forward passes, we require that we assume the input is both discrete and the hidden layer contains an activation function [6]. Since our input is already numeric, we didn't have to modify it further, and as for our activation function, we utilized the sigmoid function. We would iterate through the timestamps of the given training data and take the dot products to properly weight the graph. Once our input reached the hidden layer state, we'd have to activate it to get to the output layer; once again, we used the sigmoid on the cumulative sum of our two previous layers to get the weighted output. Remember that in recurrent neural networks, memory is the key component that sets it apart from the rest, and much like our loss function we combined the first two layers' outputs to get the third. We use an array to keep track of these activations as we iterate through the various data's timestamps, before finally returning the weighted layers as well as the absolute and sigmoid-applied memory values. Though they may seem unnecessary at first, in fact they will be integral to our backpropagation step.

With backpropagation-through-time the aim is to reduce error and update the weights; accordingly, it is referred to as backpropagation-through-time even though it is in an originally sequential model, due to conducting the backpropagation process on different timestamps of the input data as opposed to the input data itself [4]. It starts from the output layer and goes to the input, quite the opposite of feed-forward. Remember that the output of each layer during the feed-forward process is the next layer's input. In a way, it's similar to composite functions, as the output function o(x) takes the hidden layer function's h(x) which comes from the input layer i(x). The network can be represented as o(h(i(x))), because that is what it is. The memory that we've stored from earlier, both absolute and sigmoid memory, now comes into play as the absolute memory represents the memory between the input and hidden, whereas the sigmoid memory represents the memory between the hidden and output.

As we iterated back through this now-unrolled network we use the resulting derivative of our previous trials to change the weights iteratively through backpropagation. Of course, we don't want to overextend our resources, so for that reason, we utilized a truncation to limit the calculation of all the timesteps, rather getting an estimate for future calculation [7]. Along with unfurling the network k number of times as we went backwards, we also had clipping values. Our gradients tended to explode if not left unchecked and we used these clipping values on any parts of the layer's derivatives that would exceed or go under the max clip and min clip respectively. We would then update with the now-reduced weights as we continued our gradient descent. Once either condition – convergence, or iterations reached – had been met, we tried to predict our training data through the now-trained network before returning the now-weighted graph for use in our next step. Backpropagation is then conducted after the forward pass. We utilize a specific type of backpropagation called Backpropagation through time (BPTT) [7]. We do this specific method as it is best for the sequential data, and once again, we want to retain as much memory as possible from the previous steps. We calculate the gradients and try to minimize the error through backpropagation.

We then will update the weights with the weight gradients. Since the gradients themselves may tend to become extreme without proper boundaries, we include clipping values that will "clip" the gradients in the event of "exploding" gradients. After doing the clipping, we will update the weights accordingly, between input, hidden, and output layers respectively.

IV. Results & Analysis

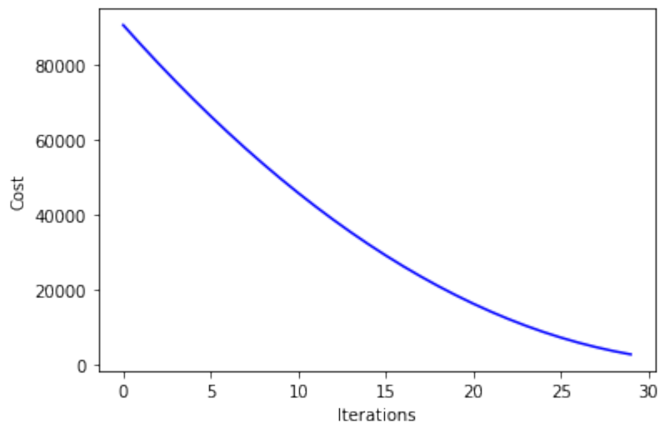

Fig. 2. Cost over Iterations

Trials	Attributes				
	Max Iterations	*Backprop Truncation*	*Clip Absolute Value*	*Train RMSE*	*Test RMSE*
Trial 1	30	5	5	62.69	35.97
Trial 2	50	7	7	32.09	26.79
Trial 3	50	10	10	31.36	16.31
Trial 4	50	3	3	57.87	30.4
Trial 5	20	10	15	40.6	45.03

Fig. 3. Sample Log Table

After running our code through the necessary number of epochs, it is time to get our predictions. We do this through a forward pass through the network. We plot the predicted values against the actual, and we also run it against the validation data to make sure we didn't overfit. After that happens, we use the root-mean-square-error, or RMSE, score to check how much error between the predicted/actual data.

Fig. 2. shows the cost of training the initial training data subset over an increasing amount of iterations. As expected of gradient descent, the cost goes down as the iterations increase until a minimum can be reached. It is interesting to note that the curve doesn't reach 30 iterations but stops a little bit before due to convergence. The formula was set up to end convergence once the values begin to stagnate, and we can see that in the cost.

Fig. 3. shows a sample trial log. The attributes listed are variable hyperparameters that were changed during our trials; what remained constant were the activation (sigmoid), learning rate (0.0001), hidden dimensions (130), the window (30 days), and the training/testing split (86/14). We chose these conditions because we found them most optimal when trying to run the network. We utilized 250 records as well for the optimal value.

The number of iterations we chose to change as the number of iterations has a significant effect on the cost of training the network, the more iterations, the better. The backpropagation truncation serves to limit the timesteps as computing all of them in the long run can prove to be computationally expensive. We also changed our clip values as we utilize the clips to make sure that our gradients do not explode.

The predictions were calculated by feeding forth our prediction data through the network and calculating the corresponding metrics. The main metric that we used to calculate the error of the model was the root-mean-square-error, or RMSE. We used this to measure the differences between our actual data and our predicted data. We also printed out the normalized RMSE, which ranged from 0.02 to less than 0.3 dependent on the variable hyperparameters. We utilized a normalized RMSE to get a better metric to put it in percentage, and like its non-normalized counterpart, a lower value indicates a better fit [9].

Overall, the RMSE seemed to be lowest for trials where the max iterations were 50, the backpropagation truncation amount was 10, the absolute value of the clip was 10, and the training/testing RMSE was 31.36/16.31 respectively. This success is attributed to the ideal factors that the max iterations are increasing, along with the backpropagation truncation and the absolute value of the clips. This gives us a more accurate model of short-term stock prediction, as the loss increases significantly due to increasing the number of iterations, and as a result can converge more accurately.

V. Conclusion & Future Work

Since the introduction of recurrent neural networks, there have been improvements made to address the vanishing and exploding gradient problems, as well as improve the overall accuracy of recurrent models. Two other models, LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) recurrent neural networks are very successful where the problems of vanilla models appear. GRU networks, as their name (gated) implies, possess two types of gates between calculations, update gates and reset gates [10]. These gates update and reset vectors based on the current weights and biases in order to determine whether the current hidden states should be adjusted or completely reset. This reduces the gradient issue. LSTM recurrent neural networks have forget gates, similar to reset gates in GRU networks that reset hidden states, but the main advantage LSTM networks possess is memory that can span over a longer period [10]. This gives greater accuracy when modeling data over longer time periods. GRU and LSTM models are more recent and more accurate models compared to vanilla networks [10].

Some future applications of this model would be to adapt it to longer data and add some more layers for further validation. In addition to this, trying out subset sampling may be more helpful if we wish to work with smaller data in the future. Along with this, trying out different activations would also be an interesting step to take to see if the model would be a better fit. Along with this, having multivariate input would be an important step as well, given the one attribute we've used in the input data. As for now, though, with a newcomer who may be interested in stocks, or someone who understands the basic principles of stocks but may be untrained to identify patterns within the graph, this model will be a good fit for them as it utilizes the most recent data of 30 days to help them make an informed decision on starting out in this field.

The overall conclusion is that while vanilla RNNs may not be as ideal as a more specialized model like an LSTM, they do well in short-term predictions with this model. The normalized RMSEs of both the training and the testing data were low which indicates a stronger model for this algorithm.

References

[1] *IBM Cloud Education. "What Are Recurrent Neural Networks?" IBM, https://www.ibm.com/cloud/learn/recurrent-neural-networks.*

[2] *Cody. "Stock Exchange Data." Kaggle, 7 June 2021, https://www.kaggle.com/mattiuzc/stock-exchange-data.*

[3] *"Who Invented Backpropagation?" People.idsia.ch, https://people.idsia.ch/~juergen/who-invented-backpropagation.html.*

[4] *"Backpropagation¶," Backpropagation - ML Glossary documentation. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html.*

[5] About the author: Chris Davis is a NerdWallet investing writer. He has more than 10 years of agency, "Stock analysis: An introduction," NerdWallet. [Online]. Available: https://www.nerdwallet.com/article/investing/stock-analysis-for-beginners. [Accessed: 30-Nov-2021].

[6] *J. Nabi, "Recurrent neural networks (rnns)," Medium, 21-Jul-2019. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.*

[7] *J. Brownlee, "How to prepare sequence prediction for truncated BPTT in Keras," Machine Learning Mastery, 14-Aug-2019. [Online]. Available: https://machinelearningmastery.com/truncated-backpropagation-through-time-in-keras/.*

[8] *Backpropagation through Time - Hit. http://ir.hit.edu.cn/~jguo/docs/notes/bptt.pdf.*

[9] *"Understanding Normalized Mean Squared Error in Power Amplifier Linearization," IEEE Xplore Full-text PDF: [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8494799.*

[10] *N. Adaloglou, "Recurrent neural networks: Building GRU cells vs LSTM cells in Pytorch," AI Summer, 17-Sep-2020. [Online]. Available: https://theaisummer.com/gru/. [Accessed: 30-Nov-2021].*