Parallelized Recurrent Neural Network for Stock Price Prediction

Matthew J. Tate
Dept. of Computer Science
University of Texas at Dallas
Richardson, Texas
mjt170000@utdallas.edu

***Abstract*—Stock price prediction has been the topic of research for many years as researchers look for ways of optimizing pattern-matching to potentially beat the market in performance. This paper explores the potential of parallelizing the backpropagation aspect of recurrent neural networks to more efficiently process stock datasets. The RNN model developed and fine-tuned for this paper was able to achieve RMSE as low as 114.06 for the TSLA stock and as low as 11.34 for the Coca-Cola stock.**

***Index Terms*—recurrent neural network, backpropagation, stock price prediction**

I. Introduction

The financial benefit of successful stock price prediction has been a large motivator for exploring the ability to model stock price movement. This task has posed a steep challenge considering the sheer number of factors that influence stock prices each day. One of the approaches to this challenge is to treat it as a time-series prediction. That is, to use deep learning to model the pattern of movements in the stock price. This leads to the idea of recurrent neural networks (RNN), which are designed to handle sequential data. For example, an RNN could train on a corpus of sentences and then generate a new sentence in a sequence. In this case, we can train recurrent neural networks on sequential stock prices in an attempt to predict the next stock in the sequence [3].

II. Background

Since stock price prediction is a kind of time-series prediction, one of the main approaches is to use sequential models. Because of recent research in the field of natural language processing, we have many different kinds of deep learning models to choose from. Selvin et al. explore different models such as LSTMs and CNNs for stock price prediction, showing the applicability of deep learning models [1]. Khare et al. suggest that simply combining feedforward neural networks with basic recurrent neural networks can even outperform the more complicated LSTMs [3].

In addition to exploring different types of recurrent neural networks, there is also research in parallelizing these deep learning models despite their sequential structure. Martin and Cundy show in their paper that it is possible to parallelize recurrent neural networks by using their parallel scan algorithm [2]. By processing the data in parallelized blocks, the overall time for computation is reduced for large datasets.

There is also research into simplifying the process of backpropagation through time in recurrent neural networks. This type of backpropagation is generally computationally expensive since it requires propagating the gradients from the current timestep back to the first timestep. One approach to simplifying this is called truncated backpropagation through time. This involves limiting the number of timesteps that must be propagated through when computing the gradients for the connected hidden layers. Tang and Glass demonstrate an efficient way to accomplish this in the application of speech recognition [4]. This paper explores the potential of limiting the backpropagation through time to one single timestep for each input.

III. Problem Definition

The main goal of this paper was to develop and tune a recurrent neural network model from scratch in a parallelized fashion that can predict the price movement of stocks. A good recurrent neural network model should be able to take a sequence of consecutive training data as input and then continue to generate consecutive predictions afterward. To achieve this, recurrent neural network models break the inputs into separate timesteps, with results from each timestep feeding into the immediate next timestep.

The challenge in this case is parallelizing the model despite each timestep relying on the results of the previous timestep. As mentioned in the previous section, truncated backpropagation through time can be used to reduce the reliance of each timestep on previous timesteps during the backpropagation phase of the model training [4]. This paper will detail how truncated backpropagation through time can be used to allow for the parallelization of the model training process for faster computation.

IV. Model Architecture

This paper uses a basic Recurrent Neural Network architecture for stock prediction. The model consists of a hidden layer with an activation function as well as an output layer. Both tanh and sigmoid were explored as possible activation functions for this model. The hidden layer's size h is defined by the user during model creation and takes the input features, the result of the previous timestep's hidden layer activation function, and a bias as input. The input features of size n

are multiplied with an n x h weight matrix Wxh. The result of the previous timestep's hidden layer activation function is multiplied with a weight matrix Whh. The results of the current timesteps's hidden layer are then multiplied with an h x 1 matrix Why and a bias is added to produce the model's prediction. Note that squared loss was used as the loss function for this model. The square loss was also multiplied by 1/2 in order to simplify the gradients. This forward propagation process is illustrated by the equations below:

$$h_t = X_t W_{xh} + a_{t-1} W_{hh} + b_h \tag{1}$$

$$a_t = tanh(h_t) \tag{2}$$

$$\hat{y}_t = a_t W_{hy} + b_y \tag{3}$$

$$L = \frac{1}{2}(y_t - \hat{y}_t)^2 \tag{4}$$

One of the greater challenges faced in this paper was incorporating parallelization into a model framework that is inherently sequential in structure. While the traditional sequential process of forward propagation through timesteps was preserved, map-reduce was used to parallelize the backpropagation process. Traditional recurrent neural networks perform backpropagation through a technique called backpropagation-through-time, which is also inherently sequential. To get around this, the gradients were truncated so that they were calculated on a strictly per-timestep basis, allowing for the use of parallelization. For each epoch, the results of forward propagation were parallelized into a PySpark RDD, distributing each timestep's results onto separate worker nodes. The map phase involved computing the gradients for each timestep in parallel, improving time efficiency for datasets with large input sizes. The reduce phase involved combining the gradients of each timestep into one gradient per weight, followed by the weight updates. The gradients for backpropagation are illustrated below:

$$\frac{\partial L}{\partial W_{hy}} = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial W_{hy}} \tag{5}$$

$$= -1 \cdot (y_t - \hat{y}_t) \cdot a_t \tag{6}$$

$$\frac{\partial L}{\partial b_y} = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial b_y} \tag{7}$$

$$= -1 \cdot (y_t - \hat{y}_t) \tag{8}$$

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial a_t} \cdot \frac{\partial a_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{xh}} \tag{9}$$

$$= -1 \cdot (y_t - \hat{y}_t) \cdot W_{hy} \cdot (1 - (tanh(h_t))^2) \cdot X_t \tag{10}$$

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial a_t} \cdot \frac{\partial a_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_{hh}} \tag{11}$$

$$= -1 \cdot (y_t - \hat{y}_t) \cdot W_{hy} \cdot (1 - (tanh(h_t))^2) \cdot a_{t-1} \tag{12}$$

$$\frac{\partial L}{\partial b_h} = \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial a_t} \cdot \frac{\partial a_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial b_h} \tag{13}$$

$$= -1 \cdot (y_t - \hat{y}_t) \cdot W_{hy} \cdot (1 - (tanh(h_t))^2) \tag{14}$$

V. Methodology

In terms of using this RNN architecture for stock prediction, the first step was to choose the right input features. Stock datasets typically organize the data in order of consecutive days that the market is open. Each of these days will have an Open and Close price for the stock ticker as well as other information such as High, Low, and Volume. The main approach for this paper was to utilize a sliding window of previous Open prices as input features for predicting the current day's Open price. The best size of the sliding window was unknown, so it was treated as a hyperparameter to be tuned on validation data. Some examples of this hyperparameter tuning are later discussed in the results section.

Since most stock datasets contain relatively high values, the inputs to the model of course had to be scaled down in order to prevent overflow. Scikit-learn's MinMaxScaler function was utilized to scale all of the stock open prices between 0 and 1. This same scaler was also used to revert the predictions of the model back to their original scale for the purpose of better visualization.

Another consideration when utilizing the model for stock prediction was how the results of the model would be evaluated/compared. Since the model performs regression when estimating the open price for a stock on a given day, the RMSE (Root Mean Squared Error) was utilized as an evaluator.

Another hyperparameter of the model to tune was the learning rate. This value was more complicated to tune since it depended on the specific stock ticker being evaluated. If the value was too high, then the gradient descent process would diverge, rather than converge towards zero loss. If the value was too low, then it would take an unreasonable amount of time/epochs for the model to converge towards zero loss. For example, a learning rate of 0.0001 seemed to work better with the Tesla stock, whereas a learning rate of 0.00001 seemed to work better with the Coca-Cola stock. The tuning results of this hyperparameter are also shown in more detail in the results section.

VI. Results

The recurrent neural network model architecture described in this paper was tested on two separate stock datasets: Tesla and Coca-Cola. These two stock datasets were incorporated to evaluate and tune the model on stocks with completely different behaviors. The Tesla stock has become a rather volatile stock since the beginning of 2020, as shown in Fig. 1. On the other hand, Coca-Cola is considered to be a safe and stable stock. Fig. 2 shows the 500 most recent Open prices for the Coca-Cola stock, which have mostly remained within a small range. By testing the RNN model on both of these types of stock datasets, we can get a better idea of how strongly the model can capture patterns in stock price movements.

In terms of tuning the learning rate, it had to be done separately between the two stocks due to different scales of values and input sizes. Table. I depicts the results of testing different learning rates on the Tesla stock. The highest learning rate of 0.001 was tested first. This value was clearly too

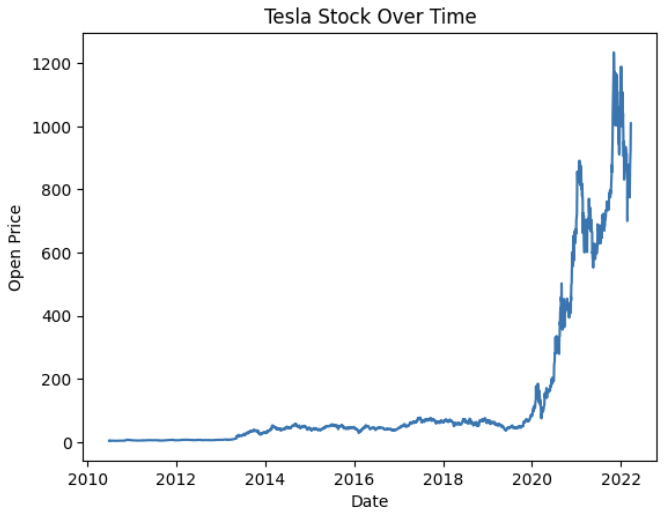

Fig. 1. Plots the open prices of the TSLA ticker over time, starting from 2010 and ending in 2022.

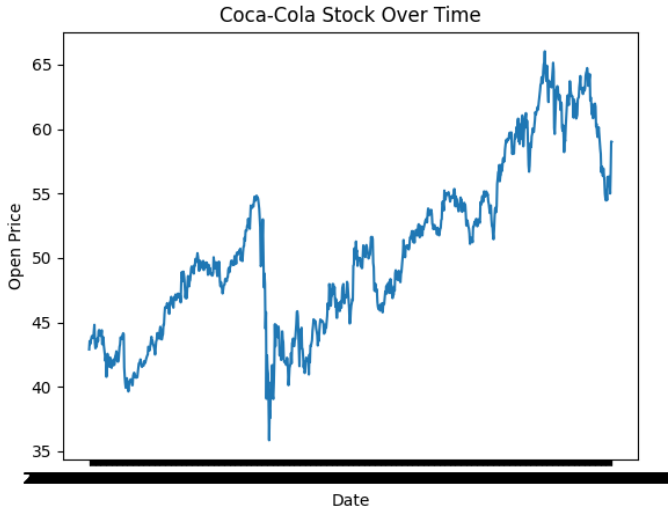

Fig. 2. Plots the 500 most recent open prices of the Coca-Cola stock ticker over time.

high as the loss diverged instead of converging towards zero, resulting in an unusable model. Between 0.0001 and 0.00001, 0.0001 resulted in a much more desirable model. This may be due to the smaller learning rate not having enough epochs to converge.

The Tesla stock was also used to compare the two activation functions, tanh, and sigmoid. Table. II shows the results of the two activation functions when fixing the hidden layer size to 128 and the learning rate to 0.0001. The model using the tanh activation function performed much better than the model using the sigmoid function. What was interesting is that both models were able to converge to a loss of near zero during the training phase, but performed vastly differently on the test set. This suggests that models using the sigmoid activation function tend to overfit when used on time-series data.

One of the most important hyperparameters to test was the feature size. It is difficult to know how much of an effect previous opening stock prices have on the next stock prices as there are many other factors such as real-world events that can influence stock price movements. Compared to the previous two hyperparameters, the results for different values in terms of RMSE were not that different. As shown in Table. III, using a sliding window of 50 previous Open prices had the best effect on accuracy.

The hidden layer size hyperparameter was tuned on the Coca-Cola stock dataset instead of the TSLA stock dataset since it seemed to produce more varying results in terms of RMSE. As shown in Table. IV, the model performed best when 128 hidden units were used. This value also seemed to be best when performing different runs of the model on the TSLA stock.

TABLE I
RESULTS OF TUNING LEARNING RATE ON TSLA STOCK

Hidden Layer	Activation	Learning Rate	RMSE
128	tanh	0.001	NaN
128	tanh	0.0001	114.06
128	tanh	0.00001	2115.57

TABLE II
RESULTS OF TUNING ACTIVATION FUNCTION ON TSLA STOCK

Hidden Layer	Learning Rate	Activation	RMSE
128	0.0001	tanh	114.06
128	0.0001	sigmoid	406.75

TABLE III
RESULTS OF TUNING FEATURE SIZE ON TSLA STOCK

Hidden Layer	Learning Rate	Activation	Feature Size	RMSE
128	0.0001	tanh	30	149.24
128	0.0001	tanh	50	114.06
128	0.0001	tanh	100	138.19

TABLE IV
RESULTS OF TUNING HIDDEN LAYER SIZE ON COCA-COLA STOCK

Learning Rate	Activation	Hidden Layer	RMSE
0.00005	tanh	64	13.61
0.00005	tanh	128	11.34
0.00005	tanh	256	13.49

VII. DISCUSSION

There were many challenges encountered when implementing the recurrent neural network model using PySpark. Initially, rather than have the results of the hidden layer's activation function feed into the next timestep's hidden layer, the idea was to feed the results of the hidden layer itself into the next timestep. When testing the model on small-sized training data, this did not present an issue. However, training

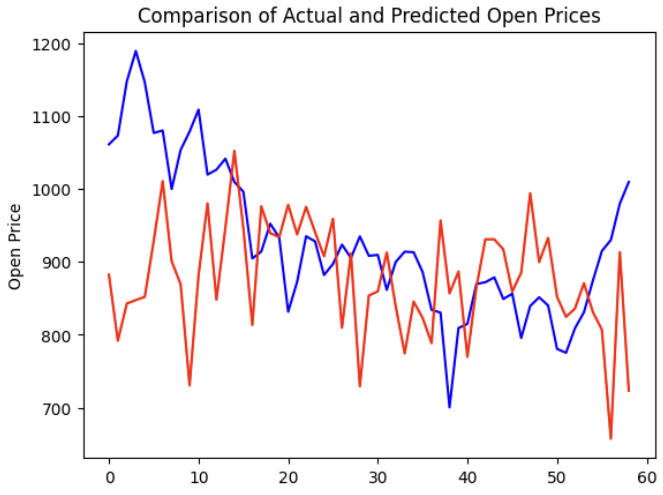

Fig. 3. Plots the predicted Open prices against the actual Open prices for TSLA.

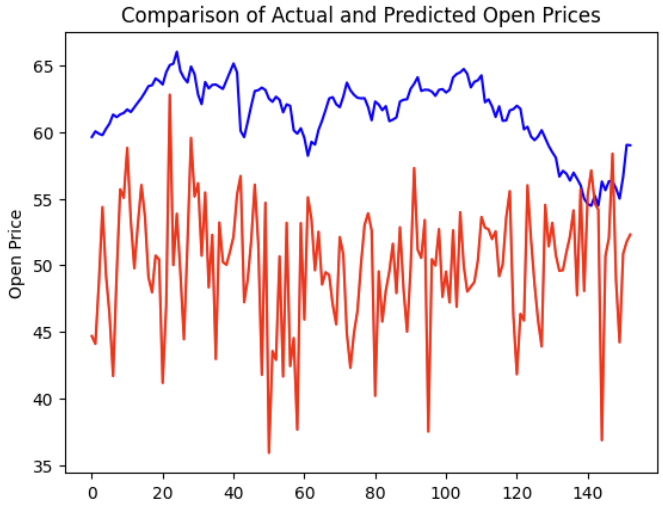

Fig. 4. Plots the predicted Open prices against the actual Open prices for Coca-Cola.

the model on any actual stock dataset resulted in overflow after just the first epoch. It became evident that the results of the previous timestep's hidden layer would need to be rescaled before being incorporated into the current timestep, which led to the idea of using the results of the activation function instead. This is because after each timestep, the results of the activation layer will be no larger than 1, and no less than -1 (or 0 in the case of the sigmoid activation function).

Another challenge was handling the starting values for the previous timestep's hidden layer/activation function for the very first iteration of both training and testing the model. At first, these values were initialized to zero, since there is no prior information to incorporate into the model. While this seemed to work well for the training phase, it made less sense for the testing phase. It was then decided to ensure that the first Open price of the test set would be the day after the last Open price of the training set, with the results of the final activation function of the training set being passed to the test set as its initial value. This restriction however made it impossible to utilize any type of cross-validation when tuning hyperparameters since the test set must always occur right after the training set in the original dataset.

Handling the train-test splits for the stock datasets also posed a challenge for the Tesla stock, since the stock behaves vastly differently depending on the time frame. As shown in Fig. 1, the stock behaves in a rather stable manner between 2010 and 2020. Since the Coca-Cola stock already covers this type of stock price movement, it was decided to only use the last 500-1000 instances of the Tesla stock.

In terms of evaluating the RNN's ability to model the Tesla stock, the RMSE was rather high even in the best case. Fig. 3 shows a graph of both the actual open prices and predicted open prices after training the model with 128 hidden layer nodes, 1000 epochs, tanh activation function, and a learning rate of 0.0001. While the model was able to reach a squared loss of near zero during the training phase, it does not generalize very well on a volatile stock like TSLA.

Although the RMSE was much smaller, the model still did not perform well on the test set for the Coca-Cola stock. As shown in 4, the model predictions consistently underestimated the actual Open prices of the Coca-Cola stock. One interesting observation however is that the vertical movements in the predictions seemed to somewhat match the movements in the actual stock prices, even if they were underestimated by 10 dollars are more. The model may have done a better job of modeling the prices of the Coca-Cola stock due to the stability of the actual stock prices over time.

VIII. Conclusion

While this paper demonstrates that parallelization of recurrent neural networks through the use of truncated backpropagation is possible, the accuracy of the predictions produced by the model is far from good. As shown in 3 and 4, there is a significant difference between the actual and predicted stock prices. It is difficult to tell whether this inaccuracy is due to the lack of depth in the recurrent neural network's layers or the approximation given by the truncated backpropagation.

In terms of computation, it seems that the use of parallelization is not worth it when the input size is relatively small. As the number of hidden units increased, the model took much longer to complete the same number of epochs. There may also be more time-efficient methods of performing map-reduce than using basic PySpark RDDs.

One of the main limitations of the paper was the use of truncated backpropagation, which leads to an approximation of the gradients, and in turn more loss during the training phase. For future work, it would be interesting to explore other methods of parallelizing recurrent neural networks that make less extreme assumptions in the model framework. It would also be useful to test these methods of parallelization on more complicated RNN models such as GRUs, LSTMs, and transforms in the context of time-series prediction. Another

limitation of this paper was the use of only one hidden layer in the recurrent neural network model. In deep learning, neural networks often have issues in terms of generalizing the data when there are not enough layers to capture the patterns in the data.

REFERENCES

[1] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 2017, pp. 1643-1647, doi: 10.1109/ICACCI.2017.8126078.

[2] E. Martin and C. Cundy, 'Parallelizing Linear Recurrent Neural Nets Over Sequence Length', arXiv [cs.NE]. 2018.

[3] K. Khare, O. Darekar, P. Gupta and V. Z. Attar, "Short term stock price prediction using deep learning," 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2017, pp. 482-486, doi: 10.1109/RTEICT.2017.8256643.

[4] H. Tang and J. Glass, "On Training Recurrent Networks with Truncated Backpropagation Through time in Speech Recognition," 2018 IEEE Spoken Language Technology Workshop (SLT), Athens, Greece, 2018, pp. 48-55, doi: 10.1109/SLT.2018.8639517.