Recurrent Neural Networks for Time Series Prediction

1st Muhammad Muhibullah
mam200014@utdallas.edu

2nd Leonardo Duenes Gomez
lxd180007@utdallas.edu

3rd Jehu Didimos
jdd170004@utdallas.edu

4th Priyesh Patel
php180002@utdallas.edu

***Abstract*—Recurrent Neural Networks (RNN) have become essential for time series prediction. This research paper details the implementation of a recurrent neural network for stock price prediction. More specifically, using a recurrent neural network-based algorithm to predict stock prices for a given stock within a predefined time. The predictions are then validated against the same stock within another time range. Given the nature of the market, indicators like trading volume and moving average are treated as attributes, making this a linear regression problem. The model utilizes gradient descent to minimize the cost between the predicted and actual price; backward propagation ensures that indicators with the most influence on price are prioritized. The results indicate that stock prices can be predicted within a significantly low margin of error given an optimally tuned RNN. An optimal RNN considers the vanishing and exploding gradient effects with respect to backward and forward propagation. Additionally, it utilizes the ideal number of epochs to maximize accuracy and limit training time.**

I. Introduction

Predicting the future is an idea so commonly pondered that it may well be innate. Machine learning and neural networks have enabled the developed algorithms to predict the future to a degree previously unimaginable. Using neural networks to model and predict time series has become increasingly popular in recent years. The most notable reason is the limitation of the linear models that have been used to the same end for decades. Linear models cannot capture nonlinear relationships in data, meaning their effectiveness heavily depends on the nature of the data being used and other similar factors. Because of the inherent nonlinear characteristics in the data, no single linear model is able to approximate all types of nonlinear data structures equally well [1]. Naturally, the inherent nonlinear nature of neural networks perfectly complements the nonlinearity of real-world data and proves highly effective in discovering the underlying relationship in complex data sets.

Nevertheless, recurrent neural networks (RNNs) have their drawbacks. RNNs can be challenging to train and may suffer from a local minimum; this issue is further complicated because even the most refined backpropagation algorithm may yield fewer optimal parameters. The second issue is that if the data has a significant linear component, the added nonlinear complexity in RNNs might produce worse results as compared to the linear models [2]. These issues are mitigated by the nonlinear nature of stock prices, thus simplifying the approach and shifting the focus to the efficacy of the backpropagation algorithm. Given its application's scope and nature, it is possible to build a sufficiently accurate time series prediction model using a recurrent neural network and optimized backpropagation algorithm.

II. Conceptual Study

The application of neural networks in financial market prediction is extensive and ever-growing. Nijole, Aleksandras, and Algirdas explored the effects of different epochs and the number of neurons on time series prediction results [3]. Their results indicate that RNNs require a sufficiently large number of epochs to achieve a stable learning rate; in their case, 164 epochs. Additionally, they found that increasing the number of epochs did not benefit the learning rate. Increasing the number of neurons yields better results; however, the significant increase in training time limits the benefit of doing so. Basic neural networks sometimes struggle to predict stock prices accurately, so it is necessary to explore different architectures and web architectures.

A. RNN

A Recurrent Neural Network (RNN) is an advanced form of neural network that has internal memory that makes RNN capable of processing long sequences [4]. This means that RNNs are capable of learning long-term relationships in a given dataset. Given inputs in the form x1, x2, . . . , xt , an RNN is defined in figure 1.

$$h_t = \sigma(Wx_t + Uh_{t-1} + b)$$

Fig. 1. Equation of an RNN [2]

In the equation, ht represents the hidden state vector at time t, and represents the sigmoid activation function; W, U, and b represent the hidden state parameters. A major weakness of RNN is apparent when we consider backpropagation. When calculating the output, all neurons in the present layer and all past layers are involved; therefore, all the participating weights must be updated to minimize the error. To propagate backward, the current neuron's weight is multiplied by the recurring weight, leading to a fast decreasing gradient as the initial weights are already close to 0. If the gradient is too low for a particular layer, that layer will not be trained correctly; therefore, the next layer in the RNN will take in untrained inputs. Figure one illustrates an RNN where xt represents the input, ht represents the hidden layer and yt represents the output. The right side of figure one expands the RNN to show the flow of the gradients and the backpropagation algorithm at work. After the RNN outputs a prediction, we compute the prediction error and feed it into the backpropagation algorithm to find the gradient. This way, the gradient descent algorithm can be used to ensure no loss in the learning rate, as is the case when the gradient becomes too small.

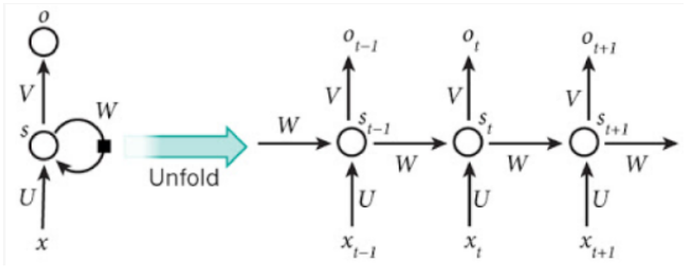


Fig. 2. Simple RNN architecture [4]

B. LSTM

Long Short-Term Memory (LSTM) is a sub architecture of RNN that has shown significant performance with sequential data, particularly learning long term dependencies and relationships. LSTM is designed specifically to solve the vanishing gradient problem that plagues standard RNNs. In LSTM each hidden state is represented by a cell with its own memory, called cell state, and gates to modify said memory. The cell can learn new memories through the input gate, forget old memories though the forget gate, and decide which memories to present as outputs through the output gate. Each gate is a single layer neural network within its own weights and sigmoid activation function; the outputs of these gates indicate how much to read, write, or forget. Figure three gives a functional breakdown of each part of the cell.

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

Fig. 3. Equations that govern LSTM-Based RNNs [2]

The left side of the equations represents the outputs of the forget gate, input gate, candidate state, cell state, output gate, and the final cell output in that order [2]. As with the standard RNN, W, U, and bs represent the network parameters learned during training. LSTM can prevent the vanishing gradient effect because, during backpropagation, the cell state is updated using only the addition operation; therefore, a constant error is propagated for each cell.

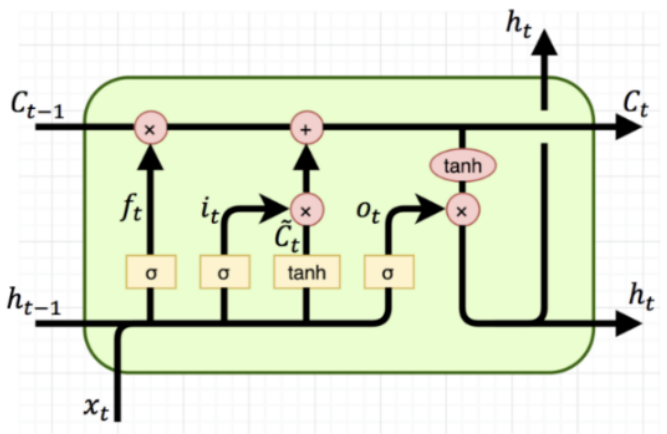


Fig. 4. A single cell LSTM architecture [4]

Figure four depicts a single LSTM cell at time series t. The cell has inputs ht-1 and xt, which are fed into theta, a single-layer NN with a sigmoid activation function [2]. Cell state ct represents the long-term dependencies learned by the LSTM; the derivative of ct helps prevent the vanishing gradient effect. Each cell in the LSTM has output ht; furthermore, on the k-th time step, the network will output a prediction vector h(k). The backpropagation algorithm is used to compute the gradient for each time series, and the complete error gradient is given by the sum of these subgradients. To prevent the vanishing gradient effect in the LSTM, the network must ensure that at least one of the subgradients does not converge to zero.

III. Potential Issues

RNNs can suffer from issues ranging from complications in data processing due to the model choice to vanishing and exploding gradients. RNN is inherently nonlinear; therefore, it benefits from studying similarly nonlinear data. Stock market data introduces an interesting dynamic. While stock prices and the parameters that affect them are nonlinear, it is possible to conceptualize price fluctuations linearly. By doing this, the RNN can effectively understand the nonlinear parameters affecting the stock price while simplifying the learning process. Paired with a gradient descent algorithm, we can mitigate another significant issue, the local minima. Setting up the RNN to take advantage of the data and avoid complications resulting from the model data prepossessing is essential.

The vanishing and exploding gradient effects are opposite but equally detrimental issues. As mentioned before, the vanishing gradient effect results from sigmoid transformation. The initial weights of the RNN are usually very small but not zero. As the error is propagated backward, it is multiplied by the weight of each neuron in the network. Therefore, with the weight already being small, the propagated error will decrease significantly with each backward pass. The result is that areas of the network will not have a large enough gradient to train. The outputs of these layers, which act as the input for the following layer, will be untrained. A network experiencing the vanishing gradient effect cannot learn anything from the data it is processing because it will be untrained at every level.

The opposite is true for exploding gradients. As the network updates, large error gradients accumulate and cause very large updates in the network weights. The updates are exponential and will eventually overflow, leading to NaN values for future weights. A neural network with exploding gradients is unstable and, like with the vanishing effect, cannot learn from its data.

The equation in figure 5 illustrates the vanishing and exploding gradient effects. The sequence becomes longer as the distance between t and k increases; the y value determines whether the gradient will vanish or it will explode. If y is greater than 1, the gradient will explode; otherwise, the gradient will vanish.

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| = \left\| \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq (\gamma_{\mathbf{w}} \gamma_{\mathbf{h}})^{t-k}$$

Fig. 5. Vanishing / Exploding Gradient Equation [5]

IV. Data processing

With stock data being sequential, meaning that the next outcome depends on previous outcomes, we decided to implement an RNN. RNNs contain memory, allowing them to remeber past events and use that knowledge to make educated predictions on future events. Since RNN is a model dependent on the data being in a sequence,, we had to arrange the training data to simulate predicting the price. To do so, we accumulated 2,000 price points of Netflix stock throughout the past years. We then specified a specific time slice to indicate how far back to look to predict the next point. In our case, we used ten days for our time slice. This meant after every nine data points into our x, it would store away the 10th point into y. So the predictions are based only on the previous nine points and comparing compared to the stored values.

V. Limitations

When approaching the project, the initial plan was to implement an LSTM network rather than a generic model that could also take in sequences of other features, such as RSI, EMA, and volume. However, due to time constraints and workloads from other classes, the ability to fully grasp and implement the more complex nature of an LSTM network was hindered, and we had to settle for a general RNN that would only take in one feature.

VI. Results

When it came to tuning the main parameter we altered was the epoch. The time slice was another parameter that could've been tuned, however ts influence was not deemed significant enough to be worth experimenting with. With an epoch of 40 our RNN was able to predict stock prices for upcoming days with an error of only 0.188 and a loss of 0.0071 the graphs below display the relation between epoch and loss as well as the results for the model's prediction:

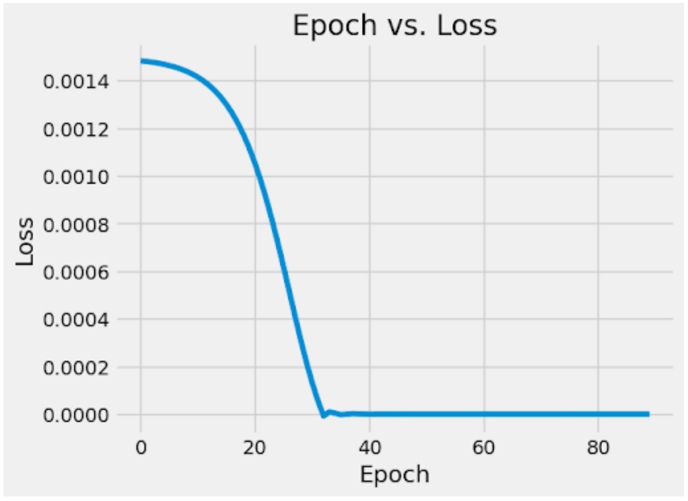


Fig. 6.

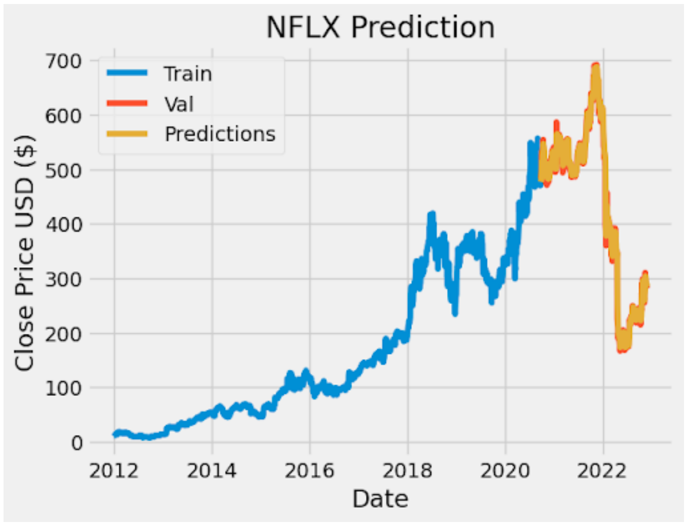
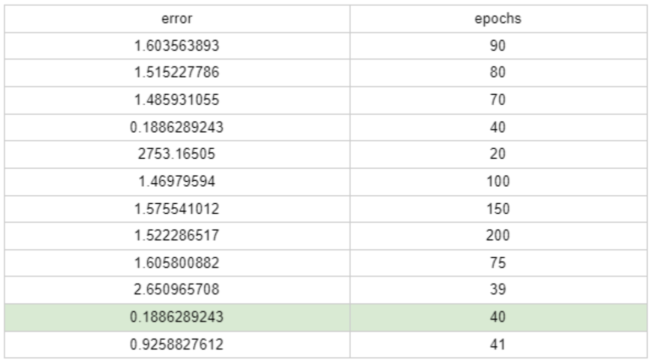


Fig. 7.

The table below compares the error to the number of epochs. Our research found that when using an RNN, there is an optimal number of epochs for maximizing prediction accuracy. Increasing the epoch count beyond this point would yield diminishing results because of the increase in training time.

In this case, the optimal number was found to be 40 epochs. After around 15-20 epochs, there were drastic decreases in the loss and error but any change after 40 had a detrimental

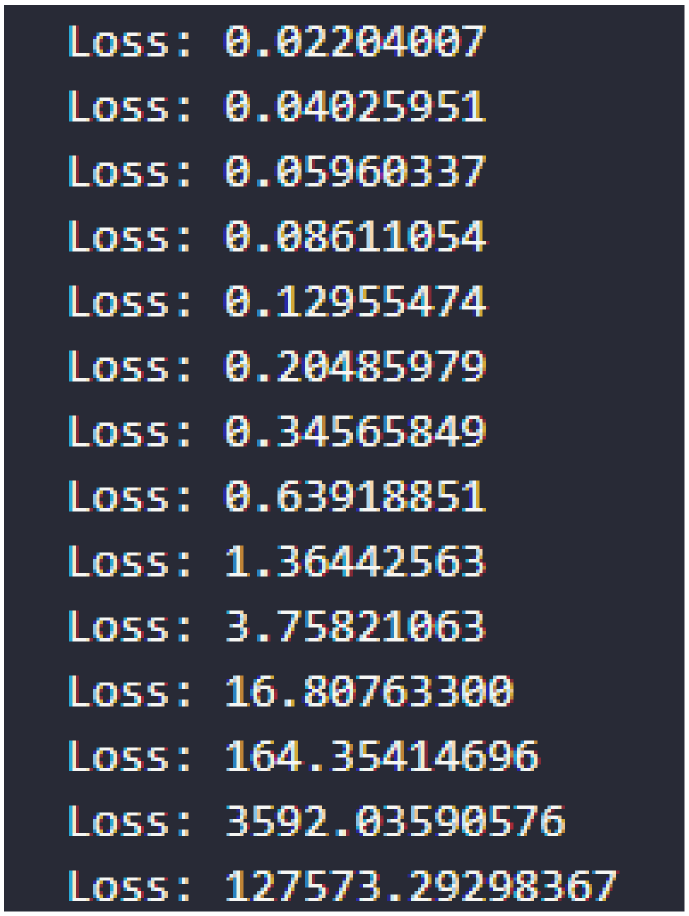


error	epochs
1.603563893	90
1.515227786	80
1.485931055	70
0.1886289243	40
2753.16505	20
1.46979594	100
1.575541012	150
1.522286517	200
1.605800882	75
2.650965708	39
0.1886289243	40
0.9258827612	41

Fig. 8.

effect. Decreasing the epochs below 20 greatly affected the error, while increasing it above 40 caused a notable increase in the error.

Figure nine illustrates the negative effect of the exploding gradient issue on the experiment.



```
Loss: 0.02204007
Loss: 0.04025951
Loss: 0.05960337
Loss: 0.08611054
Loss: 0.12955474
Loss: 0.20485979
Loss: 0.34565849
Loss: 0.63918851
Loss: 1.36442563
Loss: 3.75821063
Loss: 16.80763300
Loss: 164.35414696
Loss: 3592.03590576
Loss: 127573.29298367
```

Fig. 9.

During development, we also ran into the issue of exploding gradients. A potential drawback for RNNs is that sometimes when analyzing long data sequences, the gradient's derivative may be extremely large. An exploding gradient occurs when the gradient approaches infinity, and the adjustments to the network's weights become too large. The vanishing gradient effect is the opposite of this, where the gradient approaches zero, and the network's weights are no longer significantly updated. In one of our implementations of the RNN, we would experience an exploding gradient after around 45 epochs. As indicated by figure eight, the gradients increased exponentially and affected the weights of the RNN wildly. The result is an unstable network that cannot learn from the training data, and as the gradient continues to grow, the RNN will no longer be able to update the weights. After re-evaluating our network, accompanied by some research, the fix was to alter our gradient function from using sklearn's mean squared() function to a simple error formula:

$$error = \frac{true - predicted}{n}$$

Fig. 10.

VII. Conclusion

The goal of the research laid out in this paper is to develop a recurrent neural network capable of predicting stock prices within a predefined space of time. To that end, the optimal RNN would need to consider three major issues: exploding gradient, vanishing gradient, and epoch count. The problem in our implementation was the exploding gradient effect caused by an accumulation of significant error gradients resulting in large updates to the weights in the network. We solved this by altering the gradient function to reduce the size of the weight output by the Sklearn mean squared function. The optimal number of epochs was found through simple trial and error; the result was an epoch count of 40. Though an attempt was made to implement a long short term model, the RNN proved to be highly efficient and accurate in predicting stock market prices for Netflix stock. Therefore, given the scope and nature of its application, it is possible to build a sufficiently accurate time series prediction model using a recurrent neural network and optimized backpropagation algorithm.

VIII. Future Works

As mentioned previously, the exploding gradient was an issue we ran suffered with our implementation. In the future, implementing an LSTM rather than a basic RNN would decrease the likelihood of exploding or vanishing gradients and mitigate the need for reconstructing the neural networks and architecture. LSTM is specifically designed to prevent these issues; however, the added complexity prevented us from exploring its implementation in our experiment. Another potential improvement for the model is the addition of more features, such as the RSI, EMA, and volume. These are some data markers used by day traders to make predictions for stock prices that may yield better results when considered.

References

[1] G. P. Zhang, "Neural Networks for Time-Series Forecasting," Handbook of Natural Computing, pp. 461–477, 2012, doi: 10.1007/978-3-540-92910-914.

[2] H. Goel, I. Melnyk, and A. Banerjee, "R2N2: Residual Recurrent Neural Networks for Multivariate Time Series Forecasting," arxiv.org, Sep. 2017, doi: 10.48550/arXiv.1709.03159.

[3] Q. Gao, "Stock market forecasting using recurrent neural network," mospace.umsystem.edu, 2016. https://mospace.umsystem.edu/xmlui/handle/10355/56058

[4] P. Dey et al., "Comparative Analysis of Recurrent Neural Networks in Stock Price Prediction for Different Frequency Domains," Algorithms, vol. 14, no. 8, p. 251, Aug. 2021, doi: 10.3390/a14080251.

[5] Jefkine, "Vanishing And Exploding Gradient Problems," DeepGrid, May 21, 2018. https://www.jefkine.com/general/2018/05/21/2018-05-21-vanishing-and-exploding-gradient-problems/