An LSTM-based approach to Stock Price Prediction

Yong Seok (Tommy) Lee

Department of Computer Science, The College of Wooster

May 9, 2022

Contents

1 Abstract

Stock price prediction has always been a task of monumental difficulty due to the volatile and non-linear nature of stocks. It is because of this very nature that there exists the potential for returns or losses on a massive scale. Thus, if accurate predictions can be used to drive trading decisions, the rate of returns could be exponential. Much research and reviews have been done in the field of Deep Learning for stock market forecasting, and has shown to be accurate and generally superior to linear analysis techniques. Thus, this research aimed to implement a Long Short-Term Memory (LSTM) model, with the Adam optimizer and mean squared error as its loss function, to predict the prices of three stock market indices: S&P 500, NASDAQ Composite, and Dow Jones Industrial Average. The results have shown that, despite the simplicity of the model, the model was capable of making accurate predictions. This was inferred from the low nRMSE values and the plot that compared the predicted and actual values. Although the data set was selectively chosen to minimize volatility, good results can be derived from a simplistic implementation of LSTM. This implies that a more effective and extensive model could make accurate predictions during periods of extreme volatility (e.g. economic recessions). Given the numerous avenues in which LSTM models can be improved, this field remains of interest.

2 Introduction

Stock price prediction is a field of research that has been extensively explored due to the non-linear and complex nature of stocks. This nature is perhaps best illustrated by the recent 2020 stock market crash, where the Dow Jones Industrial Average (DJIA) plunged by approximately 26%, and stocks operating in crude petroleum, real estate, entertainment, and hospitality sectors lost more than 70% of their market capitalization. In that same period, however, firms in healthcare, food, natural gas, and software sectors had abnormally high returns with some exceeding 20% [8]. This volatile nature of stock prices opens the

prospect for both positive and negative returns on a massive scale. This implies that if one is able to consistently and accurately forecast stock prices, then the growth of their initial investments can be exponential.

In efforts to yield accurate predictions, researchers have looked towards Deep Learning models as a potential solution. Previous works have indicated that artificial intelligence models are more accurate and efficient than statistical models [4], the primary reason being that statistical techniques treat financial time series as linear systems. Other studies [2] have also supported this idea of using Deep Learning models for stock prediction, stating that traditional analytic techniques, which use linear analysis, cannot capture non-linear data movement.

This paper will be looking at an implementation of the Long Short-Term Memory (LSTM) model for stock price forecasting. This decision is derived from two main reasons. First, literature reviews have indicated that LSTM techniques are widely applied in this field [6]. Second, numerous studies have discussed the relevance of LSTM in this context. The points of relevance include LSTM models being able to retain information over long periods of time, making it suitable for time series data [10]. Thus, the option of using an LSTM model in this context is validated.

3 Background

3.1 Neural Networks

3.1.1 Neural Network Overview

A neural network is a model of computation that consists of a set of artificial nodes and a set of directed edges between them. Nodes are also referred to as neurons or units, but for the sake of consistency, this paper will adopt the use of nodes. Each node j is associated with an activation function l_j. Based on conventions adopted in foundational papers, nodes

are indexed with j and j'. The weight $w_{jj'}$, then, is the weight associated with each edge from node j to j' [6].

The value v_j of node j is calculated by applying the activation function l_j to the weighted sum of the values of its input nodes. Thus, it can be represented by the equation:

$$v_j = l_j(\sum_{n=j'} w_{jj'} * v_{j'}) \tag{1}$$

It should be noted, at this point, that a bias parameter b can be added, which allows the node to learn an offset. In this case, the equation would be:

$$v_j = l_j(\sum_{n=j'} w_{jj'} * v_{j'} + b) \tag{2}$$

The most commonly used activation functions include the sigmoid function and the rectified linear unit, the latter of which has been shown to improve the performance of many deep neural networks and has been used in recurrent neural networks [6].

3.1.2 Feedforward Networks

In regards to the order of computation, the first and simplest model to address this was the Feedfoward network. This is a class of networks that forbids cycles in the directed graph of nodes. The lack of cycles means the nodes can be arranged into layers. The overview of the network, then, would be as follows: the input x to a feedforward network is given by setting the values of the lowest layer. Each subsequent (i.e. higher) layer is then computed until the output is generated at the topmost layer $\hat{y}$ [Figure 1].

This network model has a loss function $\mathcal{L}(\hat{y}, y)$ that penalizes the distance between the output $\hat{y}$ and target y. Thus, training the model involves iteratively updating each of the weights to minimize the loss function.

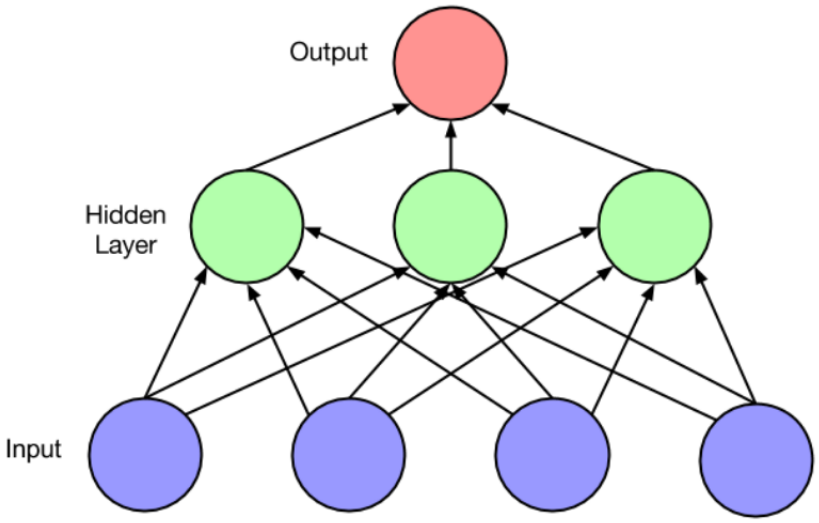

Figure 1: A diagram of a feedforward neural network [6]

3.1.3 Back-propagation

Back-propagation is an algorithm used to train neural networks. Back-propagation uses the chain rule to compute for the derivative of the loss function $\mathcal{L}$ for each parameter in the network. These weights are then adjusted by gradient descent until it reaches a local or global minimum. Given that the loss surface is non-convex, there is no guarantee that back-propagation will reach a global minimum. Regardless, back-propagation has found empirical success on many supervised learning tasks, and has been shown to be the most successful algorithm for training neural networks [6].

The back-propagation learning process can be split into two parts:

1. Forward-propagation

   (a) The input signal is propagated from the input layer to the output layer in a layer-by-layer manner. During this process, the weight and bias value of the network remain constant. The loss function then compares the output to the expected output, and return an error value.

2. Back-propagation

(a) In this process, the error value is propagated from the output layer to the input layer in a layer-by-layer manner. During each propagation, the error value is used to calculate the gradient for each node. The gradient value is then used to adjust the values of the weights and biases. The degree of adjustment is dependent on the size of the gradient. By continuously modifying the weight and bias values, the output of the model can match the expected output (i.e. the network can learn).

There are numerous ways to implement back-propagation in a neural network. Thus, discussing the ideology that guides the learning rules of back-propagation is of greater interest. The ideology is as follows: the modification to the weight value and threshold value of network shall be done along the negative gradient direction reflecting the fastest declining of function [3]. This ideology can be represented by the following equation:

$$x_{k+1} = x_k - \eta_k g_k \tag{3}$$

where x_k represents the matrix of current weight and threshold value, g_k represents the gradient of the current function, and η_k represents the learning rate.

3.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks are feedforward neural networks that have a notion of time. This is made possible by the inclusion of edges, called recurrent edges, that span adjacent time steps. While RNNs, like feedforward networks, cannot have cycles among conventional edges, recurrent edges are an exception.

At time t, nodes with recurrent edges get input from the current data point $x^{(t)}$ and from the hidden node $h^{(t-1)}$ from the network's previous state. The output $\hat{y}^{(t)}$ at each time t is calculated given the hidden node values $h^{(t)}$. Given these variables, the calculations for computation at each time step for a simple RNN [Figure 2] can be defined:

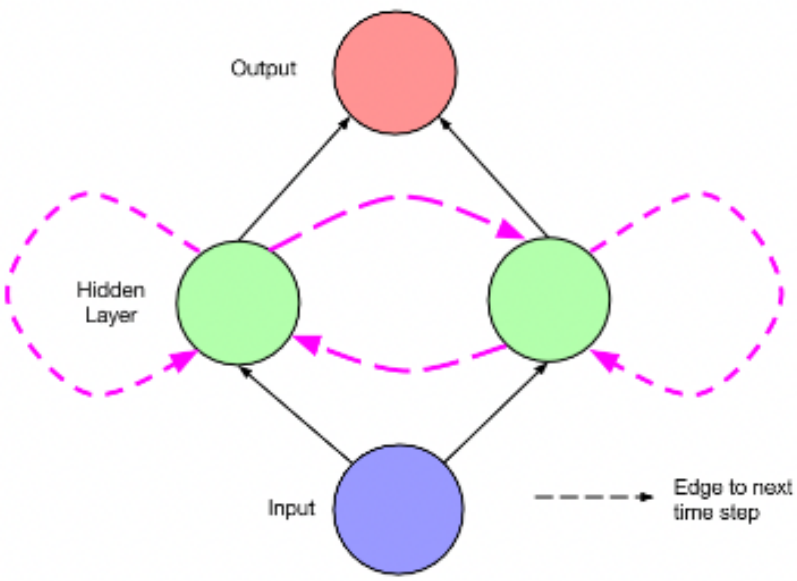

Figure 2: A diagram of a simple recurrent neural network. At each time t, activation is passed along the solid edges. Additionally, the dashed edges connect a source node at each time t to a target node at time t+1 [6]

$$\begin{aligned} h^{(t-1)} &= \sigma(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \\ \hat{y}^{(t)} &= softmax(W_{yh}h^{(t)} + b_y) \end{aligned} \tag{4}$$

where W_{hx} is the matrix of conventional weights between the hidden layer and the input and W_{hh} is the recurrent weight between the hidden layer and itself at adjacent timesteps. b_h and b_y are the bias parameters.

3.2.1 Issues in Training RNN models

Perhaps the most prominent issue that arises when training RNN models is the issue of vanishing and exploding gradients.

The vanishing gradient problem occurs during back-propagation and is exacerbated by the depth of the network [1]. As mentioned in the back-propagation section, node weights are updated in proportion to the gradient values; additionally, the weights are updated after each training iteration. However, depending on the type of activation function and network

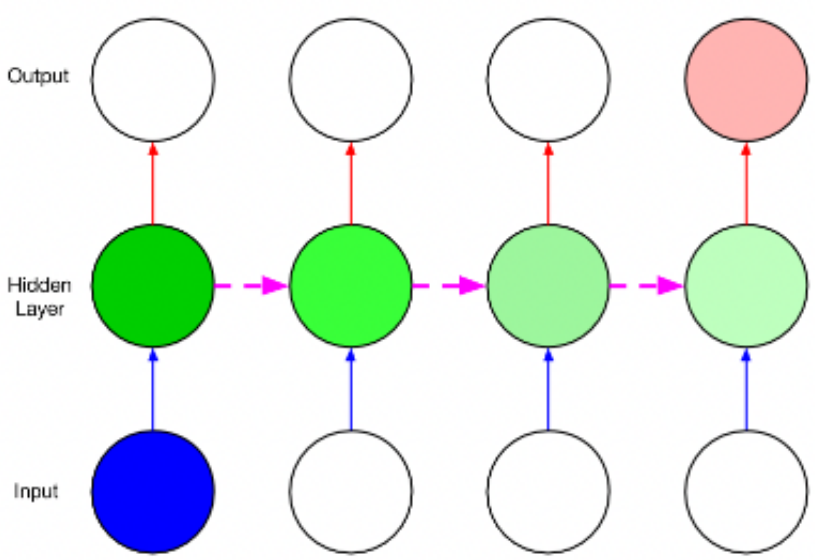

Figure 3: A Visualization of the vanishing gradient problem. The contribution of the input at the first time step, marked in green, decreases with each layer [6]

architecture, the gradient value can be diminished to the extent in which it is unable to make meaningful adjustments to the earlier layers [Figure 3]. Thus, the network is unable to learn.

The exploding gradient problem, similar to the vanishing gradient problem, is an issue that occurs during back-propagation. In this case, the gradient can grow exponentially from layer to layer [9]. This leads to two possibilities:

1. The step size is too large for updates in the lower layers to be useful

2. The step size is too small for updates in the higher layers to be useful

Given that these problems arise from the same issue (i.e. changes in gradient values), addressing one would address the other. Thus, studies often only refer to the vanishing gradient problem. To address this gradient issue, the Long Short-Term Memory model was proposed.

3.3 Long Short-Term Memory (LSTM) Model

3.3.1 LSTM Overview

The LSTM model is a modified RNN architecture that addresses the gradient issues as well as the problem of memory retention in training over long sequences [7]. There are several iterations of the LSTM architecture, but the model outlined in this study utilizes the architecture of LSTM with a forget gate. Thus, LSTM will be discussed in this context.

As previously mentioned, RNNs have feedback loops in the recurrent layer. These feedback loops are fundamental to the concept of "memory" in the model. The LSTM model, then, modifies the RNN where each ordinary node in the hidden layer is replaced by a memory cell.

3.3.2 Memory Cell Architecture

A memory cell is a composite unit, consisting of simpler nodes which are connected by specific patterns [Figure 4].

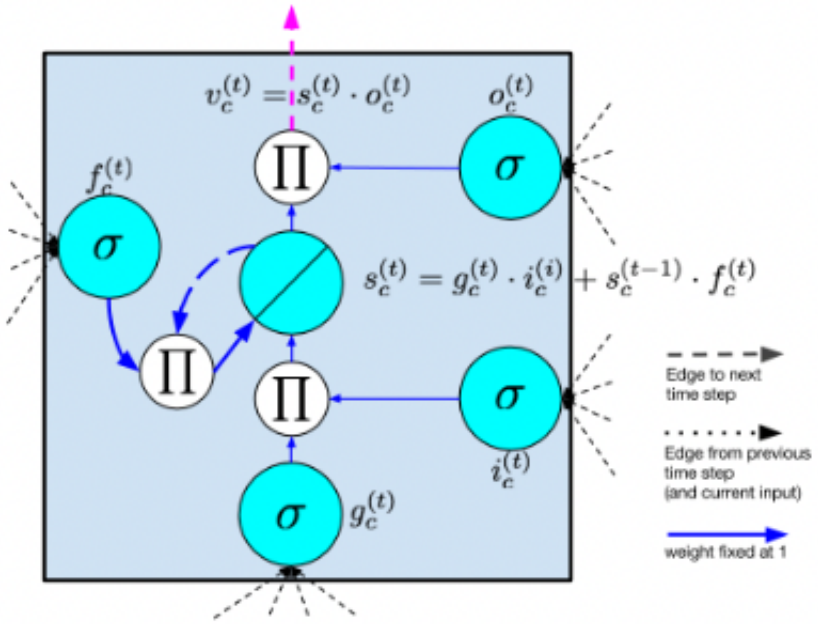

Figure 4: An LSTM memory cell with a forget gate. The π nodes are used to represent multiplicative nodes [6]

The elements of the memory cell are as follows:

- Input node: Denoted by g_c, this node takes activation from the input layer $x^{(t)}$ at time t and the hidden layer at time $h^{(t-1)}$. The input is run by an activation function, and this function varies for each implementation.

- Input gate: Gates are unique to LSTM models, and are sigmoidal units that take activation from the input layer $x^{(t)}$ at time t and the hidden layer at time $h^{(t-1)}$, much like the input node. It is called a gate because its value is used to multiply the value of another node. Thus, if its value is 0, then flow from the other node is cut off. If it is 1, then the flow is passed through. In Figure 4, the value of the input gate i_c multiplies the value of the input node

- Internal state: The internal state of the cell, represented by the node s_c, has a self-connected recurrent edge with a fixed unit weight. This edge spans adjacent time steps with constant weight; thus, error can flow across time steps without vanishing or exploding.

- Forget gate: The forget gate, denoted by f_c, provides a method for the network to flush the contents of the internal state.

- Output gate: The value, noted by v_c, represents the value of the internal state s_c multiplied by the value of the output gate o_c. Before computing the v_c, however, the internal state is run through the model's activation function.

4 Related Works

A study [10] looked at the implementation of a non-hybrid LSTM module, along with linear regression, and its effectiveness in predicting stock price movements. The model proposed here was comprised of 3 LSTM layers along with linear regression. The data set was comprised of IT companies and the mid price, (i.e. the average of the low and close price), open

price, close price, high price, and low price were considered. The study showed that the model's predictions are more accurate than standard averaging, and proposes sentimental analysis for carrying the research forward.

This proposal was explored in another study [4], which looked at implementing sentimental analysis to an LSTM model to improve accuracy of stock price forecasting. The acquired sentiments of text served as the fundamental analysis, and the forecasting of stock prices using the sentiments and historical transaction information served as the technical analysis. The sentiments were acquired from a famous Taiwan forum called PTT as well as financial, political, and international news. The news platforms used were The Epoch Times (2021), China-Times (2021), Liberty Tines Net (2021), News in United Daily Network(UDN) (United Daily Network, 2021), Money Daily in UDN (Money Daily, 2021), TVBS Media (2021), and Yahoo (2021); additionally, duplicate news articles were deleted. The extracted data was then processed using BERT to identify the sentiments of text. The sentimental data was collected by looking at information regarding specific individual stocks, and was collected using the Requests and Beautiful Soup python modules. Thus, the inputs for the LSTM model comprised of the sentiments from the news and forum articles, and the historical transaction information (i.e. include prices of opening, closing, highest, lowest, and trading volume). This study revealed that combining sentimental analysis with the forecast model could reduce Root Mean Square Error values and imply that sentiments in news and forums play a pivotal role in the stock market.

Another study [2] looked into using an LSTM model to predict stock prices in a period of volatility. The time frame of interest was in the year 2020, a year with severe fluctuations in stock market prices due to the COVID-19 pandemic. The researchers look to compare deep learning models (e.g. LSTM) to traditional data analytics and machine learning techniques (e.g. ARIMA, ARCH, SVM, etc.) and compare their performances. The LSTM model was chosen as the researchers explicitly state that the model is best suitable for time-series analysis. The model was used with the adam optimizer and sigmoid activation function to

train and test the model. The model was then evaluated using the Mean Absolute Percentage Error (MAPE). Data regarding stock indexes were extracted using the Yahoo Finance API, and the close price was used for training and testing purposes. Additionally, a window size of 60 days was used. Results showed that the model was able to predict stock prices with adequate accuracy, and the MAPE values were higher for the model than the traditional data analytics techniques.

A systematic review paper regarding the use of Deep Learning models in stock market forecasting also revealed important insights into research done in this field [5]. The most prevalent deep learning technique was LSTM, followed by hybrid models, Convolutional Neural Network, and others. A wide variety of markets were considered for these studies, including those from North America, India, China, Brazil, Korea, Europe, Taiwan, German, Morocco, and the general Cryptocurrency market. Historical stock data was collected mainly using Yahoo Finance, and hybrid models with sentimental analysis that used news data collected information from Reuters, Bloomberg, FiNet, Google News, Sina Weibo, Twitter, Tiingo, Kaggle, Epoch Times, and Nihon Keizai Shimbun newspaper. Regarding time frames, 23 of the studies used daily data. Majority of the studies used day trading as this provides a larger volume of data to train the network. Regarding the metrics used for profitability evaluation, the most common metric was accumulated profit, without factoring operating and trading costs. Only 12 assessed the profitability of their models, 2 studies considered risk management, and none of the studies implemented the model in a real environment. This review, then, highlights the importance of considering profitability and risk management. Profitability should be considered, as a high accuracy model does not inherently imply high profitability. Additionally, risk management should be considered as financial markets are noisy and chaotic, potentially leading to severe losses.

5 Methodology

The following steps have been performed to predict stock prices using the LSTM model.

1. Data Collection - The stocks used for the price prediction were the S&P 500, Dow Jones Industrial, and NASDAQ Composite indices. The data was collected from Yahoo! Finance, a media property belonging to the Yahoo! network. This platform provides the following information regarding a stock: Open, High, Low, Close, Adjusted Close, and Volume. Only the Close price was considered for analysis, however. The data was scraped using pandas, a Python library used for open source data analysis. The time frame for the stock data started from 2012/01/01 and ended at 2018/12/10. These time frames were chosen as they lie between two economic recessions: The Great Recession, which ended at June 2009, and the COVID-19 recession, which began at February 2020. A leeway of 2 to 3 years was given to ensure that the stock data retrieved was from a time of economic stability and growth. A standard 80/20 split was done, where 80% of the data was used for the training dataset and 20% of the data was used for testing dataset. A time window of 1000 trading days was used for training purposes.

2. Data Pre-Processing - The data that was retrieved from Yahoo Finance did not have missing data. This was confirmed using pandas. Additionally, analysis of the data also showed that it had no inconsistencies; the prices of the stock were tracked only on trading days (i.e. Monday to Friday). The data was normalized using the MinMaxScaler from the sci-kit learn library in the range [0,1].

3. Defining the Model - The LSTM model was built using Tensorflow, a library used for machine learning and artificial intelligence. The model has an input layer, 1 hidden layer, and an output layer, with 300 nodes per layer. Additionally, the model looks at each stock through two different epoch values, 1 and 30. Finally, the model is compiled using the Adam optimizer and mean squared error as the loss function.

4. Prediction - The model was trained on the aforementioned training dataset. The model was assessed using the normalized Root Mean Squared Error (nRMSE) value. The formula for nRMSE is as follows [11]:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$
$$nRMSE = \frac{1}{\hat{y}} RMSE \tag{5}$$

where N is the number of non-missing data points, x_i is the actual observation, $\hat{x}$ is the predicted value, and $\hat{y}$ is the normalization factor, calculated by the difference between the maximum and minimum values. The RMSE values were normalized to avoid scale dependency, which enabled comparisons of RMSE values between different stocks. Additionally, a graph that shows both the predicted and actual values were plotted.

6 Results

For each run of the model, it outputted the nRMSE value of the model as well as a plot that showed the training, testing data and the predicted values of the model. For each stock index, the model was run twice, first with epoch=1 and the second with epoch=30. Table 1 shows very low nRMSE values across the indices and epoch values. This indicates very little residual between the actual and predicted values, and imply that the predictions are accurate. This idea is supported by Figures 5, 6, and 7, which show that the predicted values nearly match that of the actual values. Additionally, the same tables and figures show that higher epoch values lead to more accurate predictions. Table 1 shows that the nRMSE values decreased substantially with higher epoch values. Figures 5, 6, and 7 show that the models which use 30 epochs have predicted values that align more closely with the actual values than those with 1 epoch.

Stock	Epoch=1	Epoch=30
S&P 500	0.036	0.014
DJIA	0.038	0.016
NASDAQ Composite	0.028	0.019

Table 1: nRMSE values for the 3 stock indices from Jan 2012 to Dec 2018

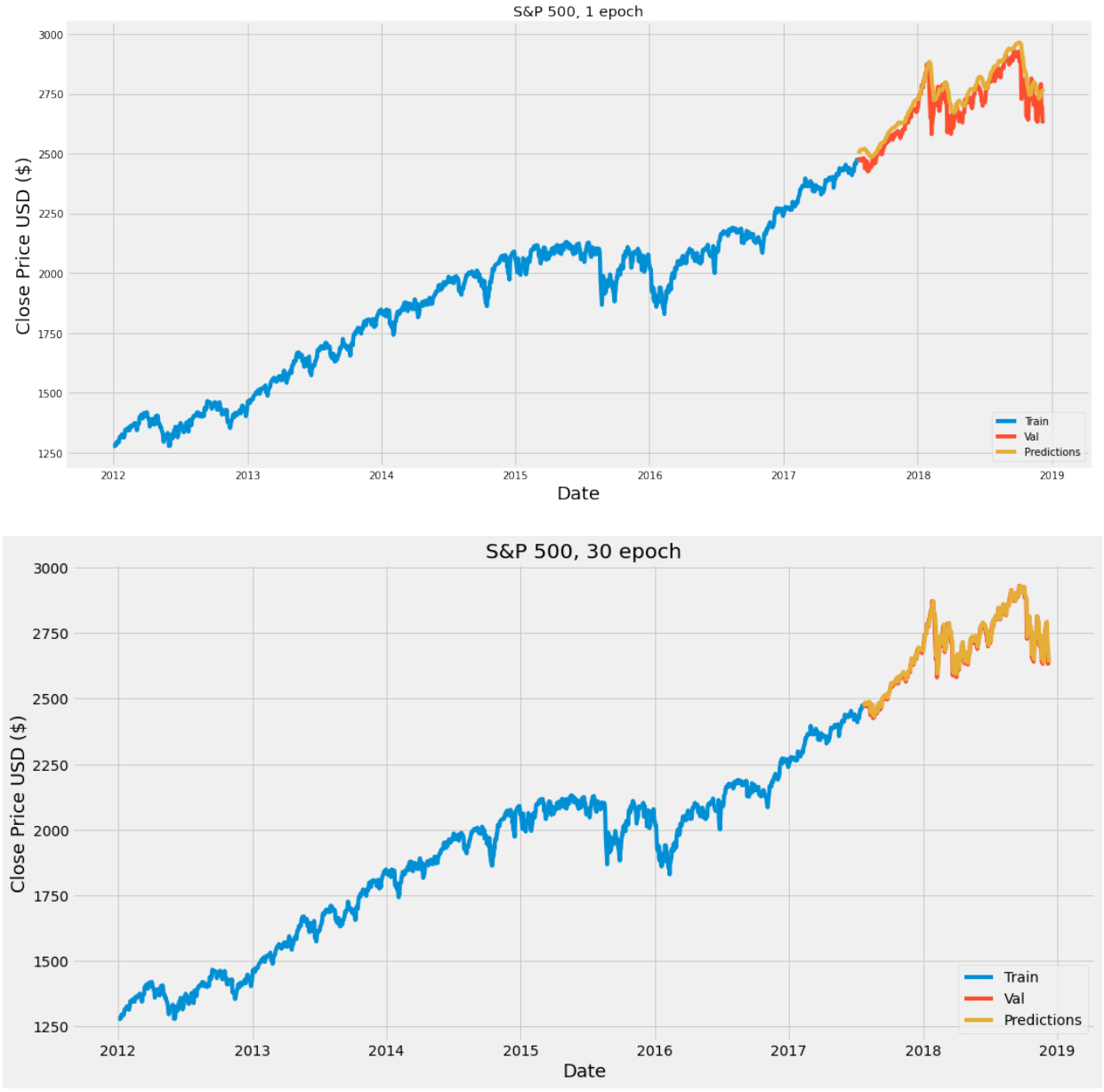

Figure 5: Predictions of the S&P 500 index prices from Jan 2012 to Dec 2018. The first figure uses 1 epoch, while the second uses 30

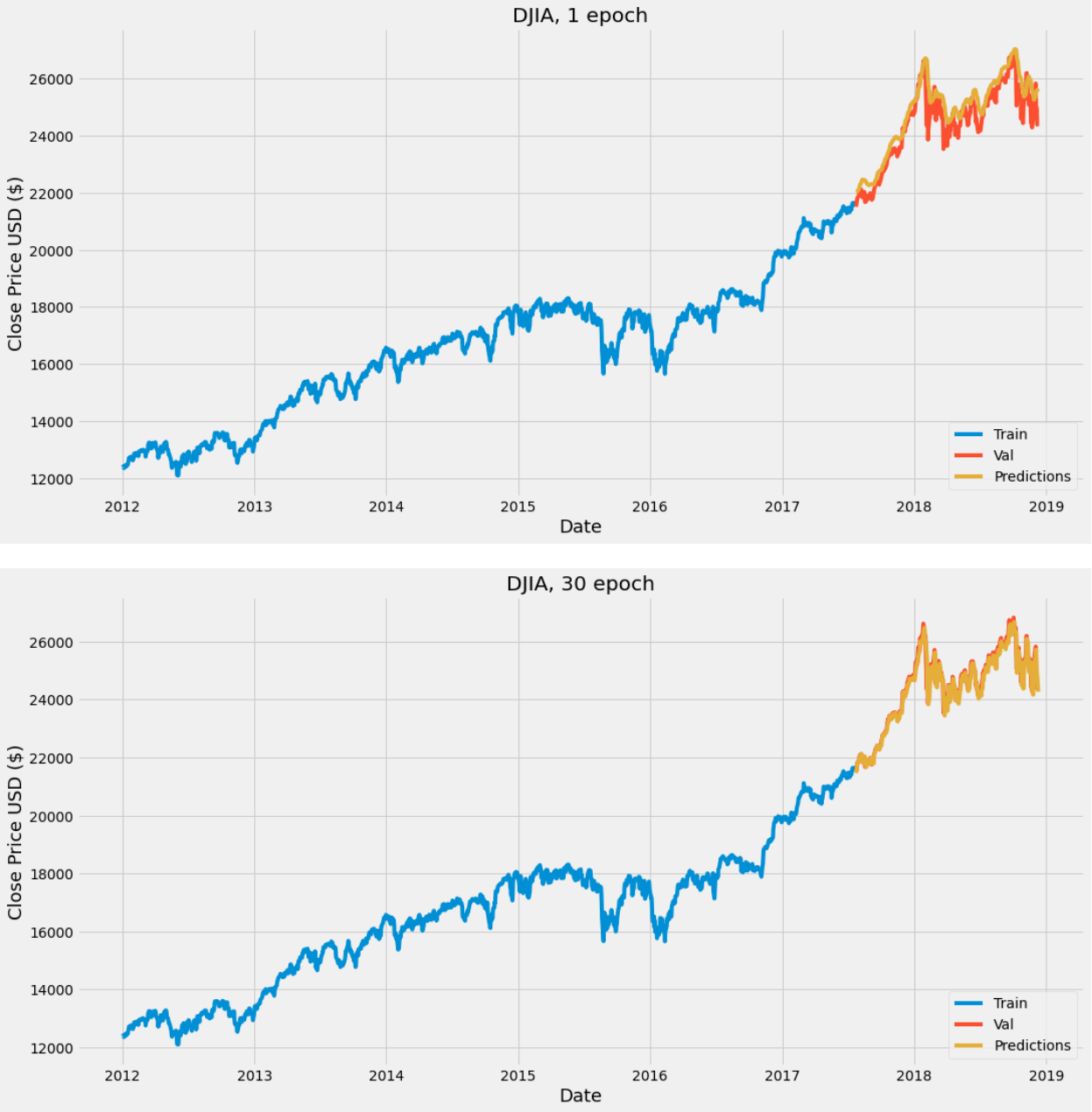

Figure 6: Predictions of the DJIA index prices from Jan 2012 to Dec 2018. The first figure uses 1 epoch, while the second uses 30

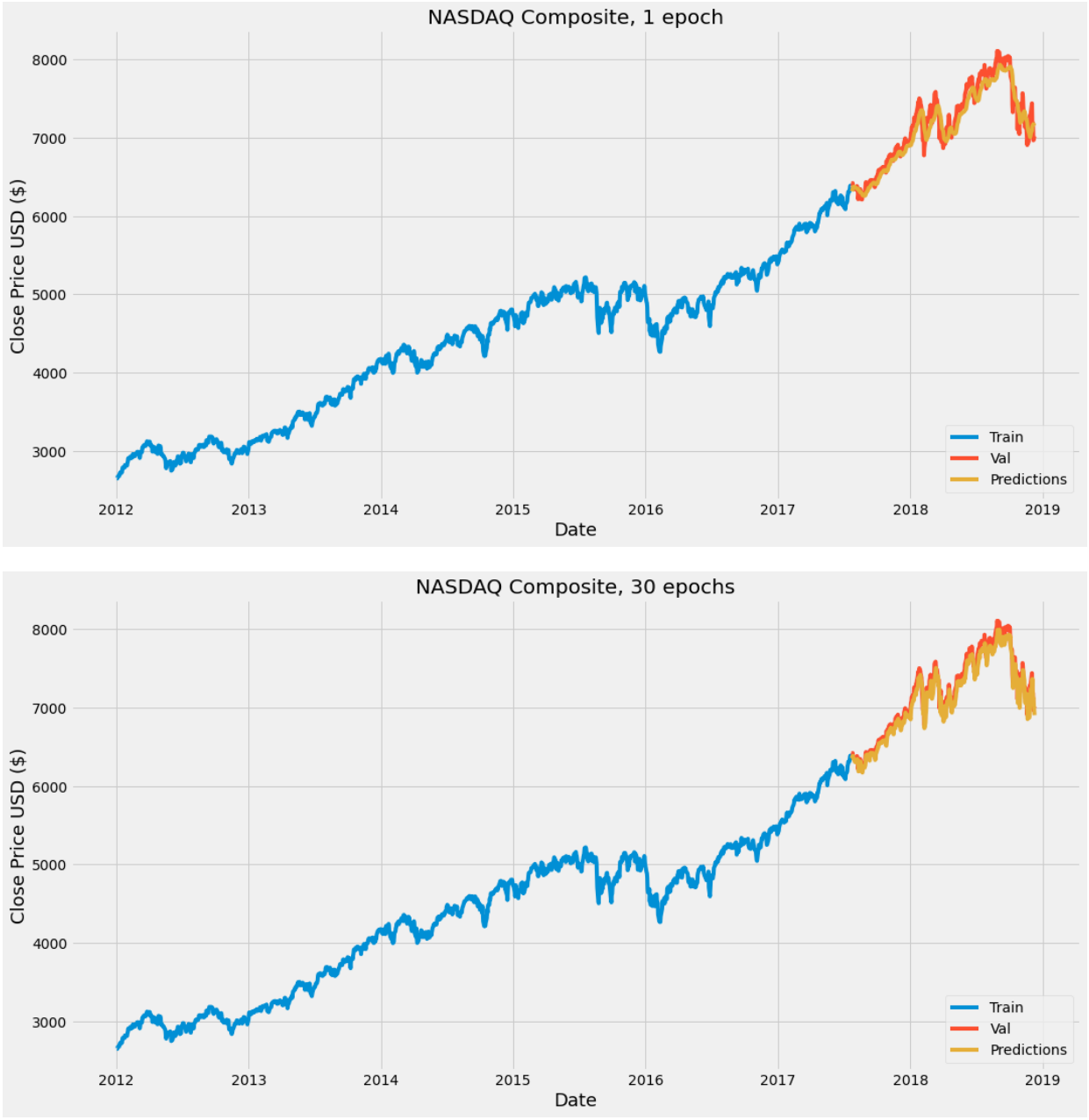

Figure 7: Predictions of the NASDAQ Composite index prices from Jan 2012 to Dec 2018. The first figure uses 1 epoch, while the second uses 30

7 Conclusion

The LSTM model was able to make accurate predictions across the three stock indices, with a significant increase in accuracy and lower nRMSE values following higher epoch values. It should be noted, however, that the stock data used in this experiment was chosen in a time frame between two economic recessions. Thus, it was a period of economic stability and growth with comparatively little volatility. Thus, it is likely that the model would not have reached the levels of accuracy shown in Figures 5, 6, and 7 if stock data during The Great Recession and the COVID-19 Recession was included.

It should be noted, however, that this model is very simple, with one hidden layer and an arbitrarily set number of nodes (300) per layer. The accuracy of the model despite its simplicity implies that a model that is more fine-tuned can make accurate predictions during periods of stock price volatility. Studies exists that show models of such capacity [2].

Some notable parameters that could greatly impact the effectiveness of the model include the activation function, number of nodes per layer, number of layers in the model, and dropout. Given sufficient time, these parameters could have been experimented with to develop a more effective model that makes accurate predictions without overfitting. Apart from the simplicity of the model, there are other threats to validity that open avenues of exploration. One shortcoming is that the only metric used to assess the model is the nRMSE value. A systematic review of studies that use deep learning for stock market forecasting [5] has shown that studies have used accuracy, profitability metrics, trading strategy, risk management, Mean Absolute Error, and Mean Squared Error to validate the performance of their models. Thus, the model can be assessed through these metrics to get a more holistic view of its performance. Additionally, the study showed that none of the studies implemented their models in a real environment. The study deemed this metric to be important, as studies have shown that highly accurate models can be unprofitable.

Generalizability of the model was also not considered. The experiment ran the model on three stock indices: S&P 500, Dow Jones Industrial Average, and NASDAQ Composite

index. These stocks all shared similar features in that they were indices, composed of many several US stocks. It could have been of interest to see how the model would perform on the sector stock indices, or even stocks of individual companies such as Apple or Tesla.

This is a field of research that provides many avenues and opportunities of research, including hybrid models paired with sentiment analysis, adoption of intelligent and adaptive trading strategies within the model, and implementation of risk management. Through extensive research and effective implementations, research in this field has the potential to shift the paradigms currently embedded in the financial stock market.

References

[1] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, September 2020. Conference Name: Big Data Mining and Analytics.

[2] Gourav Bathla, Rinkle Rani, and Himanshu Aggarwal. Stocks of year 2020: prediction of high variations in stock prices using LSTM. *Multimedia Tools and Applications*, February 2022.

[3] Jing-yuan Shi Jing Li, Ji-hang Cheng and Fei Huang. *Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement*, volume 169 of *Advances in Intelligent and Soft Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] Ching-Ru Ko and Hsien-Tsung Chang. LSTM-based sentiment analysis for stock price forecast. *PeerJ Computer Science*, 7:e408, March 2021.

[5] Audeliano Wolian Li and Guilherme Sousa Bastos. Stock Market Forecasting Using Deep Learning and Technical Analysis: A Systematic Review. *IEEE Access*, 8:185232–185242, 2020. Conference Name: IEEE Access.

[6] Zachary C Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. page 38.

[7] Navin Kumar Manaswi. RNN and LSTM. In Navin Kumar Manaswi, editor, *Deep Learning with Applications Using Python : Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras*, pages 115–126. Apress, Berkeley, CA, 2018.

[8] Mieszko Mazur, Man Dang, and Miguel Vega. COVID-19 and the march 2020 stock market crash. Evidence from S&P1500. *Finance Research Letters*, 38:101690, January 2021.

[9] George Philipp, Dawn Song, and Jaime G. Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv:1712.05577 [cs]*, April 2018. arXiv: 1712.05577.

[10] Shruthi Komarla Rammurthy and Sagar B. Patil. An LSTM-Based Approach to Predict Stock Price Movement for IT Sector Companies. *International Journal of Cognitive Informatics and Natural Intelligence*, 15(4), December 2021. Place: Hershey Publisher: Igi Global WOS:000739108700006.

[11] Maxim Shcherbakov, Adriaan Brebels, N.L. Shcherbakova, Anton Tyukov, T.A. Janovsky, and V.A. Kamaev. A survey of forecast error measures. *World Applied Sciences Journal*, 24:171–176, January 2013.