LSTM Stock Price Prediction

Timothy Robbins

Bellevue University

Professor Catie Williams

October 10, 2020

Abstract

In recent years, researchers have explored the application of deep learning algorithms to financial time series prediction using recurrent neural networks (RNNs). Traditional RNN's and other prediction models have struggled with the ability to predict long-term time-series data, such as the stock market. Long Short-Term Memory neural networks (LSTMs), a special type of RNN, overcome this problem by incorporating sequence dependency and memory modules into the model and are thus able to process non-linear, non-stationary data. In this study, I will be using Python's Keras library to build a multi-layer LSTM model to predict Amazon's closing stock price.

The remainder of this study is organized as follows: The background around recurrent neural networks is discussed, along with an overview of LSTMs. I then discuss the methodology, including dataset selection, data preprocessing, and the modeling process. Finally, the study is concluded with the results and conclusions.

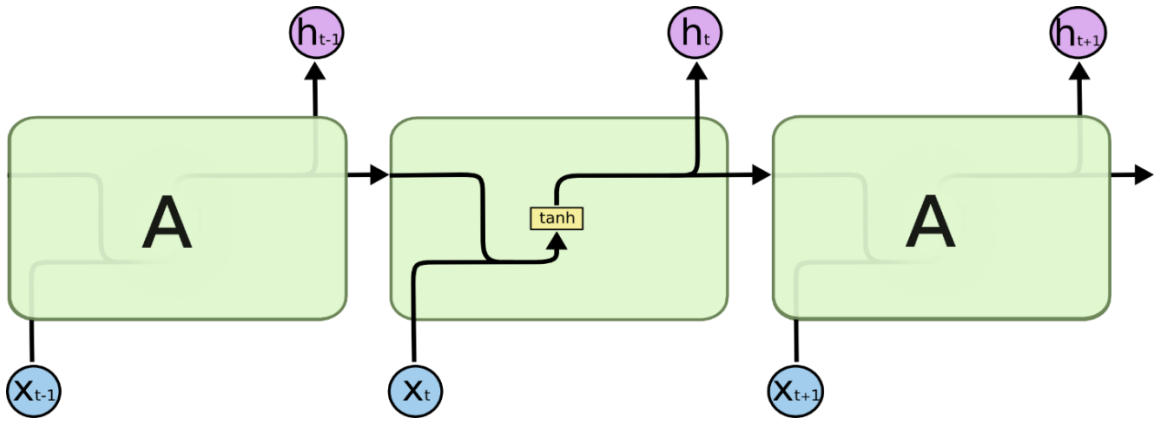
Keywords: stock prediction, stocks, recurrent neural networks, long-short term memory, deep neural networks, python, time series, keras.

Background

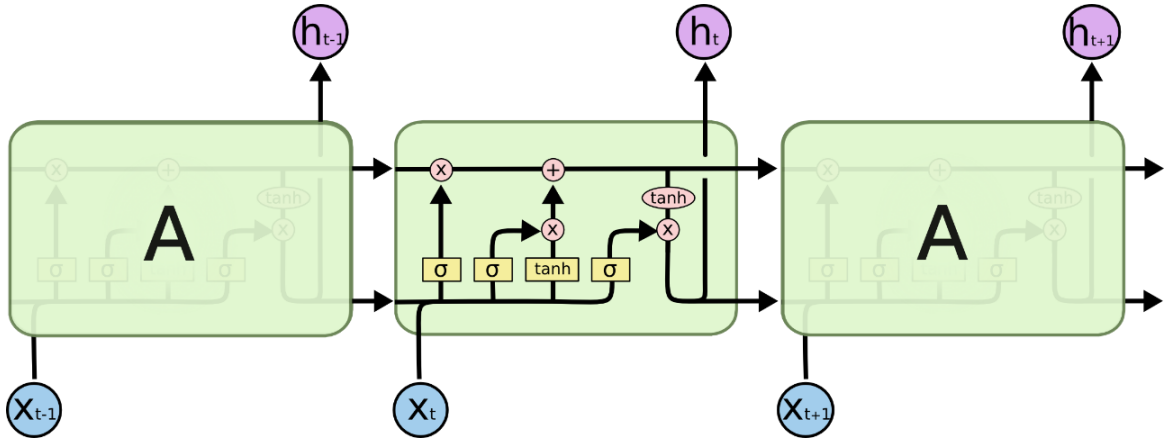
Applications of deep learning to highly dynamic and versatile environments, such as the stock market, have been on the rise in the last 10 years. It is estimated that over 1.4 billion shares of stock are traded each day world-wide. Stock prediction enables investors to make more informed decisions about which stocks they should invest in by providing a reliable estimate of stock attributes, such as closing price, volume, market trends, etc. Accurate prediction of the stock market is considered an extremely challenging task because of the noisy environment and high volatility associated with the external factors. Factors such as company news and performance, investor sentiment, politics, and economy contribute to the fluctuation in stock prices and increased investment risk. Due to such complexity, a simple linear projection model would likely not be an ideal fit for the multivariate nature of the stock market. Deep neural networks, particularly RNNs, when combined with the ability to comprehend long-term dependencies, are an ideal fit for this scenario.

Traditionally, time series prediction models such as ARIMA and GARCH have been used to predict financial time series by assuming a specific model. However, complex real-time series data, such as the stock market, contains noise that cannot be properly reflected in the model and thus has limited applicability. On the other hand, deep learning models have been found to exhibit major advantages in processing non-linear, non-stationary data since it can capture non-linear features among the feature vectors. Such algorithms have been successfully used in many areas of artificial intelligence, such as image recognition, natural language processing and driverless cars.

Long Short-Term Memory networks (LSTM's) are a special type of RNN that are capable of learning long-term dependencies. LSTM's were explicitly designed to avoid the long-term dependency problem, as they can remember information for long periods of time. All RNN's have the form of a chain of repeating modules of neural network, each representing a time series. In standard RNNs this repeating module usually has a very simple structure with only one tanh function, as shown below:



LSTMs also have this chain-like structure, but instead of a single neural network layer, there are 4 layers of both tanh and sigmoid functions, that interact in a very special way:



The core idea behind LSTMs is the cell state which runs across the top of the entire chain with only some minor linear interactions. The cell state carries information througout the processing of the sequence and allows information from the earlier time steps to make its way to later steps, reducing the effects of short-term memory. The LSTM adds or remove information to the cell state, which is carefully regulated by structures called gates. Gates are a way to optionally let information through. The gates contain sigmoid activations which, in short, trasform values to be between 0 and 1. This is important to note since any number multiplied by 0 is 0, meaning the values disappear or are "forgotten." On the other hand, any number multiplied by 1 maintains the same value and is thus "remembered." Basically, the network

learns which data is important and is thus kept or not important and thus forgotten. An LSTM has 3 such gates that protect and control the cell state: a forget gate, input gate, and output gate. In short, the forget gate decides what is important to keep from prior steps. The input gate decides what information is important to add from the current step and the output gate determines what the next hidden layer should be. Together, these layers and gates allow us to make a prediction that solves both the long-term dependency and vanishing gradient problems.

Methodology

Dataset

The dataset contains 5 years of historical trading data for Amazon stock (AMZN) retrieved from Investors Exchange Trading (IEX). There are 6 key variables. Date ("date") is the date of the trade. The stock's opening price ("open") is the price at which a security first trades when the exchange opens on a given trading day. The closing price ("close") is the final price at which a security is traded on a given trading day. The "high" is the highest price at which a stock trades over the course of a trading day. The "low" is the lowest price at which a stock trades over the course of a trading day. Finally, the "volume" is the total quantity of shares traded over the course of a trading day.

The data that a trader is most interested in is usually the close price as this reflects the gain or loss at the end of each trading day. Amazon's closing price on each day for the last 5-years is shown in the graph below:

As you can see, the stock price increased steadily between days 0 and 550 and then started increasing more rapidly between days 550 to 750. The stock remained relatively flat from days 750 through 1150, before increasing at a much more rapid pace.

Data Preprocessing

Several preprocessing steps were necessary to get the data ready for input into the LSTM model. Preprocessing was done in Python using numpy and pandas. First, a new data frame was created to store only the "close" price of the stock. Since the LSTM model only accepts arrays, the data frame was then converted to a numpy array. A standard practice when working with RNN's is to normalize the data with a Sigmoid function in the output layer. This was done using a Min/Max Scaler in the range 0 to 1.

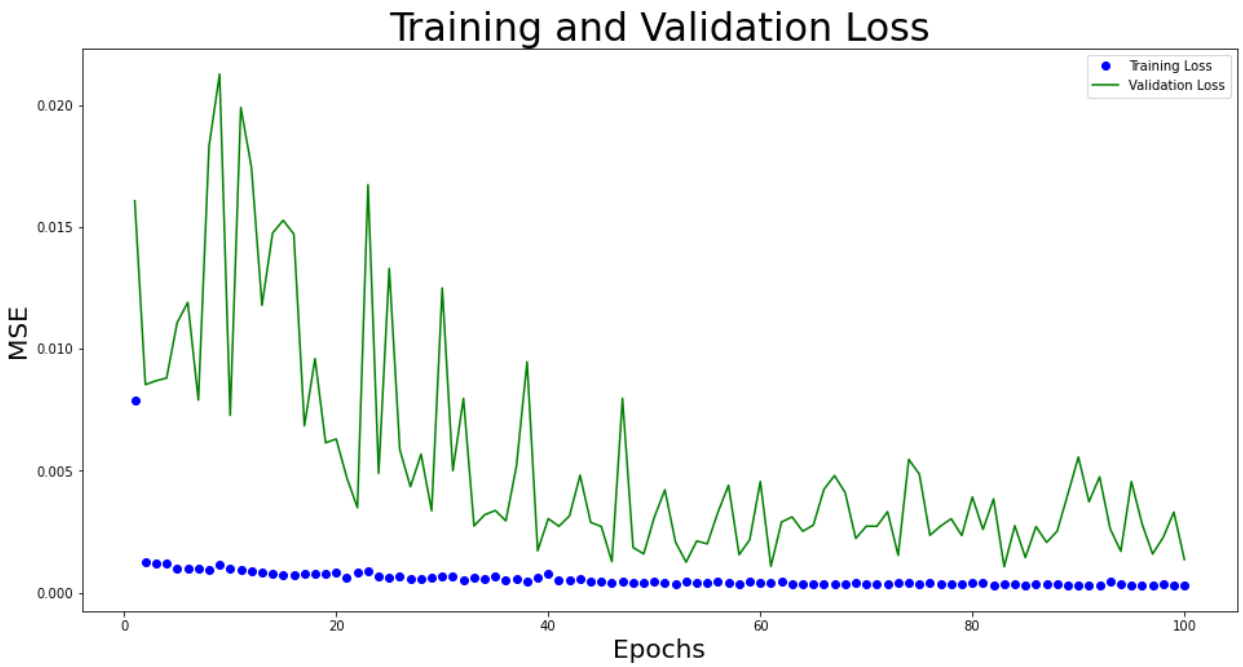
A special data structure was needed to cover 60-time stamps to predict the price on the 61st day. The number of past time stamps was set to 60 based on some experimentation. x_train, a nested list, was created to store a sequence of 60 time-stamp prices. y_train was created to contain a list of stock prices which is the next day's stock price, corresponding to each list in x_train. Essentially, each row in x_train contains 60 prices that are used to predict the corresponding next-day stock price in y_train. Finally, it was necessary to reshape the data to be 3-dimensional in the form: number of samples, number of time steps, and number of features, as the LSTM model is expecting a 3-dimensional array. The number of samples represents the number of LSTM neurons in the layer. 100 neurons will give the model high dimensionality, which will enable it to capture the upwards and downwards trends. The number of time stamps (60) is the number of days used to predict the next day price. The number of indicators was set to 1, as we have only one feature which represents the "close" price.

Modeling

After preprocessing the data set, I began building the LSTM model using Python's Kera's library. There are no definitive rules on how many nodes or hidden layers one should choose, so it took some experimentation to determine. Using Keras, I was able to stack multiple layers on top of each other by

initializing the model as sequential. Generally, two layers have shown to be enough to detect more complex features. More layers can be better but are harder to train. Since the stock market is a very complex, I chose to add 3 LSTM layers to the model. In addition, I accompanied each LSTM layer with a drop-out layer to avoid overfitting. Overfitting is reduced by randomly ignoring selected neurons during training and thus reduces the sensitivity to the specific weights of individual neurons. 20% is often used as a compromise between model accuracy and overfitting. For this reason, I added 3 drop-out layers at 20% following each of the LSTM layers. The final layer to add was the activation layer. The output dimension was set to 1 since we are predicting only the next day's price each time. See the Appendix for the model parameters used.

Next, I compiled the LSTM by choosing an optimizer and loss function. An optimizer is an algorithm that determines how to adjust the weights on the nodes in a neural network based on the difference between the predicted and actual values. For the optimizer, I chose Adam, which is often referred to as a safe choice for RNNs. Loss functions are used to compute the quantity that a model should seek to minimize during training, for which I used the mean of squared errors between actual values and predictions. I then fit the model to the training data. After much experimentation, I found the best combination of parameters to be 100 units, a batch size of 20, ran for 100 epochs. The batch size that yielded the lowest root mean squared error (RMSE) was used. Eighty percent of the data was used for training and the remaining twenty percent used for validation. The training and validation losses for 100 epochs is plotted below:

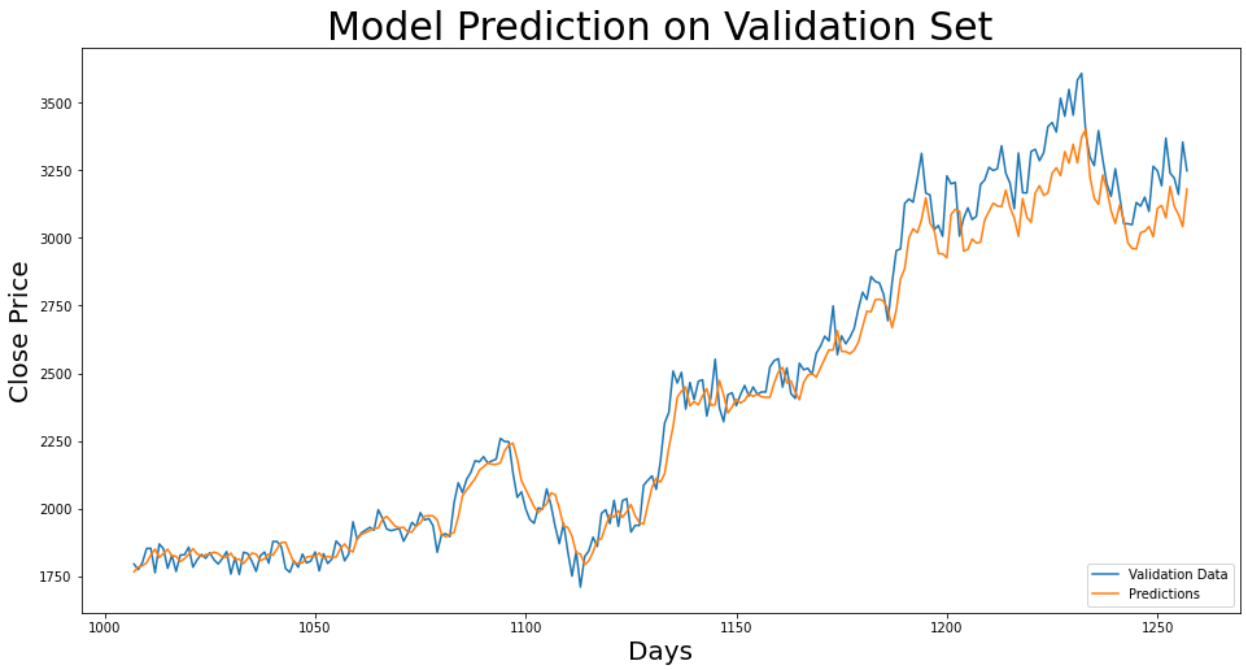


Note that the overall losses decreased with more epochs, with the minimum validation loss occurring after 82 epochs.

After importing the test data, it was necessary to concatenate the training and test data sets for prediction, as we are using the previous 60 days' stock prices to predict the next-day price. Thus, the input for prediction started with the index 60 days prior to the first date in the test set. Similar to the data preprocessing steps performed on the training set, I followed the same procedure by reshaping the inputs to have one column, scaling the data between 0 and 1, and creating the special test data structure to cover the 60 time stamps.

Results

After transforming the test data into the format required for the LSTM model, it was fed into the model to make a prediction. The plot below shows how the prediction did against the validation data.



As you can see above, the predictions closely followed the validation data, reflecting the overall upwards and downwards trends of Amazon's closing prices. The predicted prices tended to deviate further from the actuals as the time frame increased since the predictions are not adjusted to actuals – they are based on the prior predictions. For this reason, I will be predicting only one-day into the future with the test data. I also noted that the predicted spikes tend to be less pronounced and more rounded then the actual data. In addition, there tends to be lag between the validation and predicted data, with the predicted data being several days behind the validation data. This may indicate that the model may not react quickly to non-linear changes but tends to react well to smooth changes.

With the final test set, the AMZN price predicted for the next business day was $3,134.55, while the actual came out to be $3,248.18, reflecting an accuracy of approximately 96.5%.

Conclusion

This study has found that a relatively simple deep learning algorithm can be used to make reliable stock predictions. The importance of long-term dependency issues was discussed, and it was explained how such challenges can be resolved using an LSTM model. Historical closing prices of Amazon stock were used as the key factor in predicting the stock price. However, simply considering the impact of historical data on price trends may be too singular and may not be able to forecast the price fully and accurately on a given day. Future studies could potentially be enhanced by adding stock-related news and key technical indicators, such as 50-day moving average, relative strength index, and mean reversion.

Appendix

Table 1 – LSTM Model Parameters

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 60, 100)	40800
lstm_1 (LSTM)	(None, 60, 100)	80400
dropout (Dropout)	(None, 60, 100)	0
lstm_2 (LSTM)	(None, 60, 100)	80400
dropout_1 (Dropout)	(None, 60, 100)	0
lstm_3 (LSTM)	(None, 100)	80400
dropout_2 (Dropout)	(None, 100)	0
dense (Dense)	(None, 1)	101

Total params: 282,101
Trainable params: 282,101
Non-trainable params: 0

References

Anderson, R. (2019, December 23). *Stock Price Prediction Using Python & Machine Learning*. Retrieved from Medium: https://medium.com/@randerson112358/stock-price-prediction-using-python-machine-learning

Banushev, B. (2019, January 14). *Using the latest advancements in AI to predict stock market movements*. Retrieved from Python Awesome: https://pythonawesome.com/using-the-latest-advancements-in-ai-to-predict-stock-market-movements/

Ganegedara, T. (2020, January 1). *Stock Market Predictions with LSTM in Python*. Retrieved from Data Camp: https://www.datacamp.com/community/tutorials/lstm-python-stock-market

Koehrsen, W. (2018, January 19). *Stock Prediction in Python*. Retrieved from Towards Data Science: https://towardsdatascience.com/stock-prediction-in-python

Kohorst, L. (2018, November 9). *Predicting Stock Prices with Python*. Retrieved from Towards Data Science: https://towardsdatascience.com/predicting-stock-prices-with-python

Migdon, H. (2020, September 2). *How To Use the Alpha Vantage API with Python to Create a Stock Market Prediction App*. Retrieved from Last Call - The RapidAPI Blog: https://rapidapi.com/blog/stock-market-prediction-python-api/

Müller, F. (2020, March 24). *Stock market prediction: a time series forecasting problem*. Retrieved from Relataly: https://www.relataly.com/stock-market-prediction-using-a-recurrent-neural-network

Nayak, A. (2019, March 18). *Predicting Stock Price with LSTM*. Retrieved from Towards Data Science: https://towardsdatascience.com/predicting-stock-price-with-lstm

Pochetti, F. (2014, September 20). *Stock Market Prediction in Python Intro*. Retrieved from Francesco Pochetti: http://francescopochetti.com/stock-market-prediction-part-introduction

Shamdasani, S. (2017, December 15). *Build a Stock Prediction Algorithm*. Retrieved from Enlight: https://enlight.nyc/projects/stock-market-prediction

Shyam R, V. P. (2020). Stock Prediction Overview and a Simple LSTM based Prediction Model. *International Research Journal of Engineering and Technology (IRJET)*.