An Exploration of Long Short-Term Memory Networks for Stock Forecasting

Eliot Liucci, Eric Folsom, Crystal O'Connell, Nicholas Clausen

December 2024

1 Introduction

Neural networks are statistical models that are flexible and able to self-improve over time to complete any task that can be quantitatively represented. The challenge of time series forecasting, specifically in the stock market, is infamous for being a challenging task due to the unpredictable nature of the market. Day traders are able to predict market behavior by recognizing patterns from past behavior to determine where the market may go. Long Short-Term Memory networks (referred to as LSTMs for the remainder of this paper) are a special kind of recurrent neural network tuned for dealing with long-term dependencies in sequential data (Hochreiter and Schmidhuber, 1997). These models are able to account for long sequences of data by having a cell state that acts as long-term memory and various gates to ensure short-term memory doesn't impact future predictions too much. The goal of this project is to be able to use LSTMs to create accurate forecasts by accounting for past patterns.

2 Data Preparation

The process starts by finding a source of mass amounts of free data. This is to ensure that the models have sufficient data to converge to the optimal hyper-parameters. The `yfinance` package in Python allowed for us to load 10 year historical data inline. The dataset for each stock was initially loaded as a CSV file, containing the columns Date, Close/Last, Volume, Open, High, and Low. The 'Close/Last' column, representing the daily closing prices, was cleaned by stripping dollar signs and ensuring the data was in chronological order. To separate the seasonal component of the time series, the data was decomposed using the seasonal decomposition function from the `statsmodels` package, specifying a periodicity of 365 days. This decomposition allowed us to separate trend, annual, and residual components. The dataset was then split into a training set (90%) and a testing set (10%).

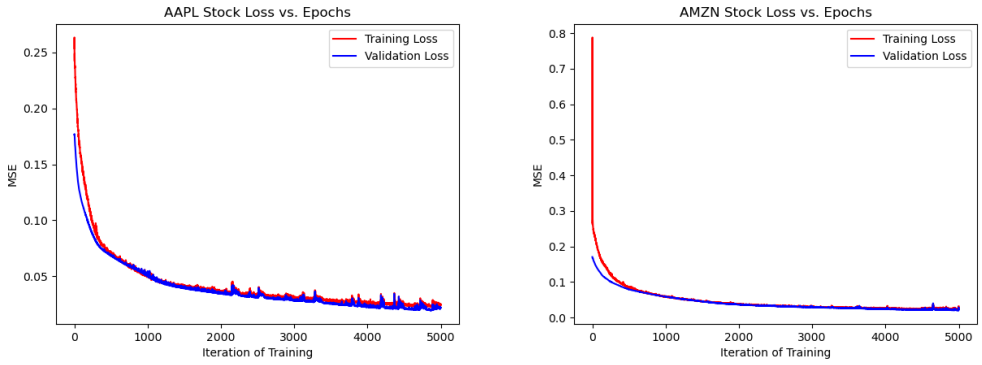
For outlier detection and to improve model robustness, the data was scaled using the `RobustScaler` from the `sklearn` package (Pedregosa et al., 2011). This transformation centers all the values to have median 0 and inter-quartile range of 1. This transformation was applied to both the training and testing sets to ensure efficient model training by minimizing the impact of extreme values.

3 Model Training

Before training models in `tensorflow` (Abadi and Agarwal, 2015), the hyper-parameters within the model were tuned using Bayesian Optimization (Snoek et al., 2012). The hyper-parameters to

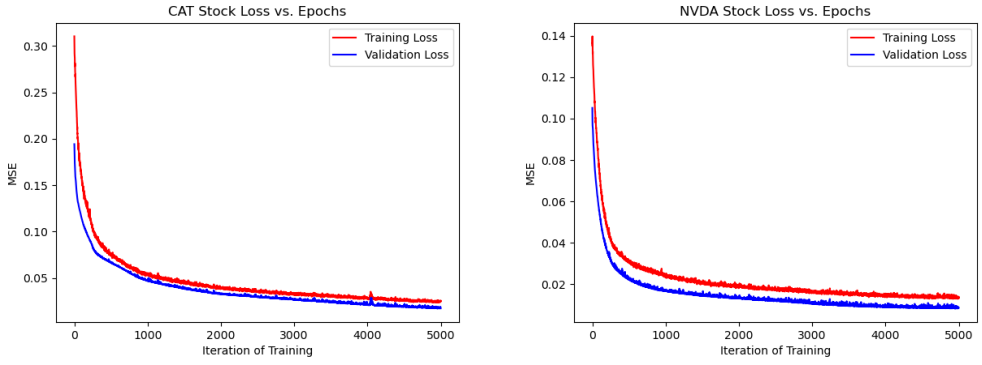
be tuned are the number of nodes in each layer (ranging from 8 to 64) and the dropout rate for each layer (ranging from 0.01 to 0.15). For all 4 stocks, the initial model contained 3 LSTM layers and a singular dense layer with 1 node corresponding to the predicted closing price for the next day. The tuning process involved initializing random values, training the model for 1,000 iterations, then recording the validation set MSE. Hyper-parameters are then tweaked and MSE values are compared. The Bayesian approach is optimized to find the minimum of the MSE gradient faster than using a random search.

The final models for all 4 stocks had 8 LSTM nodes in the first layer, 8 LSTM nodes in the second layer, and 64 LSTM nodes in the third layer. Note that each model had different dropout rates for each layer. This finding is interesting as all models had similar architecture despite being very different stocks in practice. Once these model architectures were determined, the models were trained for 7,500 iterations. The mean squared error on the training and validation sets against number of iterations is shown for each stock in Figure 1, indicating that the models converged nicely without over-fitting.



(a) AAPL

(b) AMZN



(c) CAT

(d) NVDA

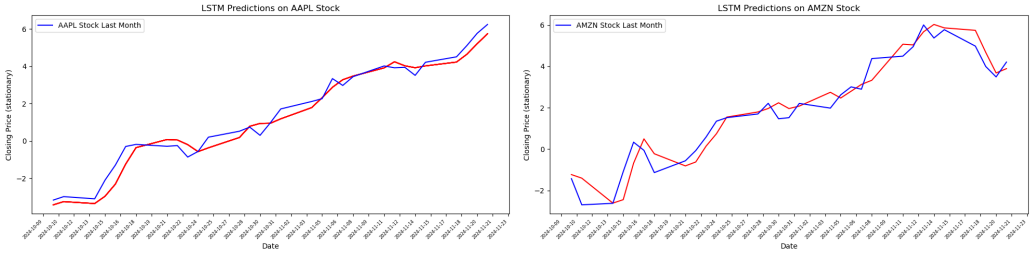
Figure 1: A series of plots showing the loss (mean squared error) against the number of iterations each model was trained for. The loss on the training set is shown in red and the loss on the validation set is shown in blue.

4 Forecasting

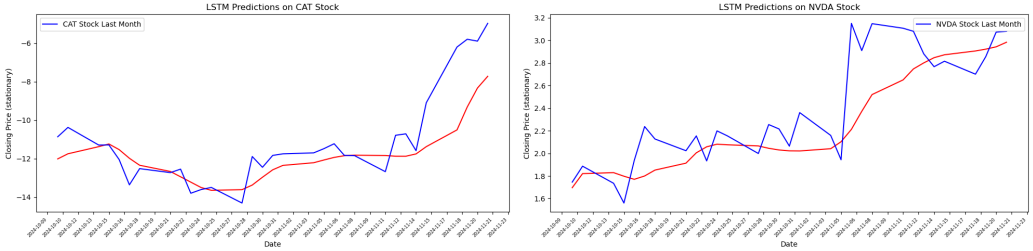
Once the models were trained, the last batch (31 days of closing prices) from the test set were fed into the models to predict the first day's closing price of the test set. Then, the last 30 days from the training set and the first predicted day of the test set were used to predict the second day's closing price. This process repeated until we had 14 days of predictions past the training data set. These values were then compared to the observed closing prices from the test set.

4.1 Predicting 1 Month Out

To get a true sense of forecast accuracy, the last 31 trading days were withheld from the training and testing sets initially. Each model was used to forecast these dates and then was compared to the actual closing prices over the last 31 trading days (Figure 2). From these graphics, it can be noted that the models capture the major trends in the stocks. The Nvidia stock was not captured quite as well as the Amazon stock, though this could be due to needing more training time.



(a) AAPL (b) AMZN

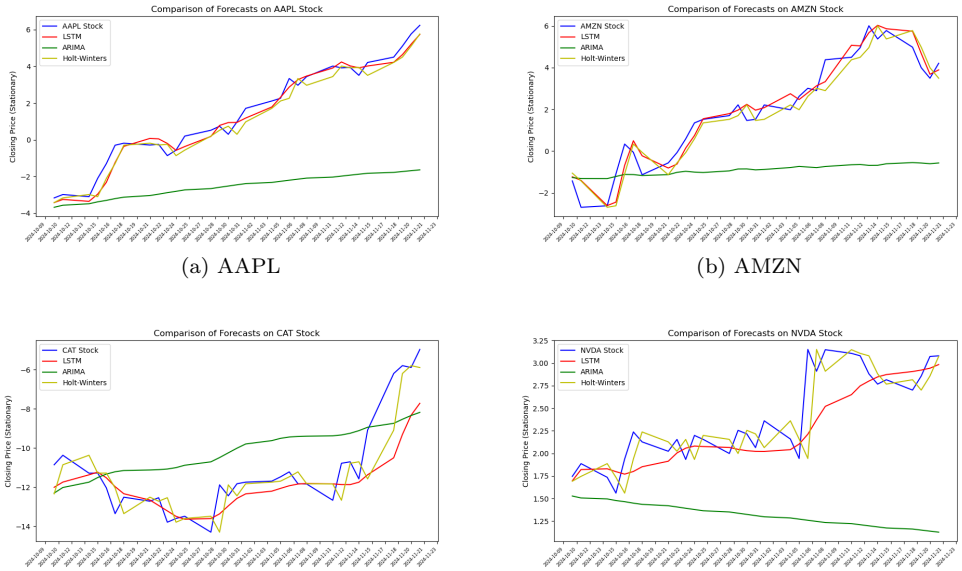


(c) CAT (d) NVDA

Figure 2: A series of plots showing LSTM predictions for a month out of sample. The last month of each stock was retained before the train/test split was created. Then, each LSTM was used to forecast 31 days out of the test set and compared to the retained month.

4.2 Comparing LSTM to Holt-Winters and ARIMA

It is difficult to assess the model performance of LSTMs without a baseline. To remedy this, the same process was done using an Autoregressive Integrated Moving Average (ARIMA) model and a Holt-Winters Exponential Smoothing model. The forecasts between the 3 methods are compared in Figure 3. The ARIMA model does not perform well, although the Holt-Winters model is able to outperform the LSTM.



(a) AAPL (b) AMZN

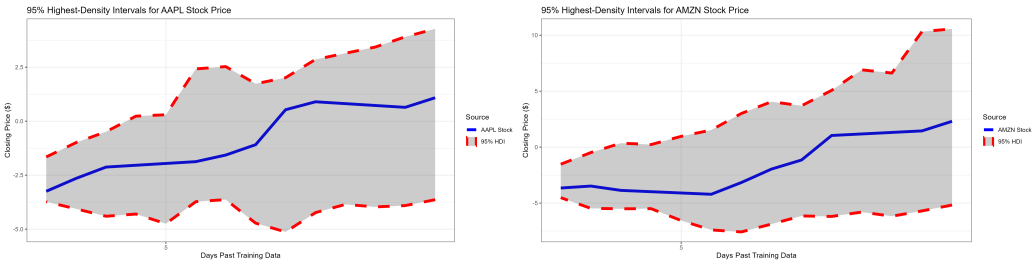
(c) CAT (d) NVDA

Figure 3: A series of plots comparing the LSTM forecasts (red) to a Holt-Winters Exponential Smoothing model (yellow) and an Autoregressive Integrated Moving Average model (green).

4.3 Credible Intervals

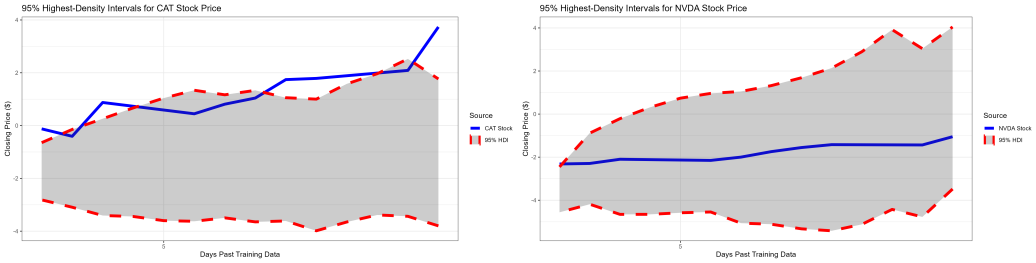
Credible intervals were obtained for LSTM forecasts by training many models of the same architecture and forecasting each time. To account for variability in the stock market, we used a bootstrap on the training data each time. We applied the Moving Block Bootstrap (MBB) method from the `tsbootstrap` package to generate resampled datasets. A block length of 31 days was chosen to preserve the temporal structure of the data. This method was used to create 20 bootstrapped samples, each forming a separate training and testing set. These resampled datasets provided a good base for model validation and forecasting.

The forecasts from the bootstrapped samples were aggregated to construct prediction intervals. The observed testing data was plotted against the predictions using `ggplot2` for visualization (Figure 4). This comparison allowed us to assess the model's accuracy and the associated uncertainty. Prediction intervals were stored for further analysis. The visualizations depicted the forecasts and the variability, providing insights into the forecast's reliability.



(a) AAPL

(b) AMZN



(c) CAT

(d) NVDA

Figure 4: A series of plots showing 95% Highest Density intervals when predicting into the original test set when the LSTMs were trained on bootstrapped training data.

5 Conclusion

Overall, it seems as though LSTMs can be used very effectively for stock forecasting. Point predictions do have the potential to be inaccurate, so having a credible interval can be helpful. For example, the credible interval for the Apple stock in Plot (a) of Figure 4 would indicate that there it is very likely that the stock would go up over the 30 training days.

Other methods of time series forecasting with LSTMs is to use multivariate time series as the input for the model. This process would involve taking the last 31 days of information from the multivariate time series of the covariates and pairing it with the 32nd day of the response time series. Then, the model could be trained on the pair of covariate matrix and response value. This method was explored but forecasts were not able to be generated as they would involve having the covariate information for the days that were to be forecasted.

References

Abadi, M. and Agarwal, A. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Lahiri, S. N. (2013). *Resampling methods for dependent data*. Springer Science & Business Media.

pandas development team, T. (2020). pandas-dev/pandas: Pandas.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peters, T. (2004). The Zen of Python. PEP 20.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.