Stock Market Prediction with LSTM Neural Networks

Jostein Barry-Straume
jostein@vt.edu
Virginia Polytechnic Institute and State University
Blacksburg, VA

ABSTRACT

In this paper, I present an evaluation of hyperparameter impact on validation accuracy for an optimized Long-Short-Term-Memory (LSTM) Neural Network (NN). LSTMs are widely used as the go-to architecture for time series forecasting. I used Google Colab to train and test 84 models to find the optimal hyperparameters with which to apply a LSTM towards the Stock Market Dataset available on Kaggle. I was able to realize a validation accuracy of 99.14% on a given Exchange Trade Fund. I found that neural density appears to have the least impact on validation accuracy, whereas the time spent training and the number of epochs had the largest impact.

KEYWORDS

long-short-term-memory, neural networks, stock market, hyperparameters, forecasting

ACM Reference Format:
Jostein Barry-Straume. 2021. Stock Market Prediction with LSTM Neural Networks. In *Advanced Machine Learning Final Project: Stock Market Prediction with LSTM Neural Networks, December 11, 2020, Blacksburg, VA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 What is the problem?

In this case study, the application of Long-Short-Term-Memory (LSTM) Neural Networks (NN) is implemented such that several machine learning techniques are experimented with to tackle the following problem. Specifically, I experiment with diagnostically benchmarking and evaluating eighty-four different Long-Short-Term-Memory (LSTM) neural networks, to forecast time series data such that the future direction of a given stock or exchange-traded fund (ETF) can be predicted reliably.

Accomplishing this optimization problem necessitates subsetting the data into training and test sets. Note that because the movement of a stock or ETF is implicitly dependent on time, the data is not shuffled. However, the training and test datasets are split in a way to create a forecasting horizon window with which model evaluation can be performed.

Optimization of the hyperparameters of the LSTM NN is realized through validation accuracy. Although model checkpoint and early stopping functionality is readily available, it was deemed unnecessary for the given scenario (training and testing on only 1 ETF). In the future, when leveraging all stocks and ETFs for developing a production ready model, these two methods will be utilized. Training and testing on all stocks and ETFs was determined to be too time consuming, even with using these two methods. With that being said, the loss function is monitored and minimized during training. By training with only 1 stock or ETF at a time, we are afforded the opportunity to save all models for rigorous evaluation of their respective validation accuracies.

1.2 Why is this an important problem?

Part of my long-term career goals involves working as a financial quantitative analyst. Working directly with financial data would go a long way towards having a project portfolio that aligns with what firms are looking for. Personally, I also just have an interest in investing. However, portfolios less than \$20,000 – 25,000 are often restricted to a given amount of “day trades.” Violating this limit can invoke temporary pauses on trading with said account. To get around this, I am targeting strategies that involve at least a 24-hour holding period.

2 RELATED WORK

2.1 Literature Review

B.V. Vishwas’ and Ashish Patel’s book "Hands-on Time Series Analysis with Python" provides a comprehensive overview of explaining, wrangling with, and solving time series problems [8]. The book begins with covering the basics of time series characteristics, then steadily delves deeper. Data preparation of time series is touched upon. Traditional time series techniques, such as implementing ARIMA are explained. As the reader works their way toward the end of the book, more "bleeding edge" technologies are covered. Namely, and directly in relevance to this report, is Chapter 7. This chapter deals with "bleeding edge" techniques for multivariate time series; namely LSTM neural networks. Overall, the book Vishwa and Patel put forth is invaluable for a general book to have on hand in case brushing up on time series is called for.

"Financial time series forecasting model based on CEEMDAN and LSTM" by Cao and et al. presents a new hybrid time series forecasting model [1]. This hybrid model is developed by combining empirical mode decomposition (EMD) and complete ensemble empirical mode decomposition with adaptive noise (CEEMDAN) with a LSTM neural network [1]. By doing so, Cao's and et al.'s proposed model produce results that are more accurate than similar models [1]. Moreover, one-step ahead forecasting efficiency of financial time series is improved upon by leveraging EMD and CEEMDAN [1].

"NSE Stock Market Prediction Using Deep-Learning Models" by Hiransha et al. is a comparative evaluation of four types of deep learning neural networks and their respective performance in predicting the stock price of a company based on historical closing prices [2]. The four types of neural networks examined in this paper are: Multilayer Perceptron (MLP), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Convolutional Neural Network (CNN) [2]. Hiransha et al. found that CNNs outperformed the other three architectures, but that all four of the examined non-linear models beat out the linear ARIMA model [2].

"Stock price prediction using DEEP learning algorithm and its comparison with machine learning algorithms" by Nikou et al. evaluates the prediction power of machine learning models by predicting the closing price of iShares MSCI United Kingdom ETF over the span of three years of historical data [4]. Nikou et al. show that deep learning outperforms Support Vector Regression (SVR), but that SVR beats neural networks and random forest methods [4].

"CREST: Cross-Reference to Exchange-based Stock Trend Prediction using Long Short-Term Memory" by Thakkar and Chaudhari leverages LSTM NNs to predict trends of a given company that is listed on multiple stock exchanges [7]. In particular, a cross-reference to exchange-based stock trend (CREST) prediction method is proposed using long short-term memory (LSTM) [7]. Thakkar and Chaudhari train and test CREST on three given cross-listed companies. Their model evaluation was based on "using root-mean-square error and directional accuracy along with precision, recall, and F-measure for the results of all three companies" [7].

"LSTM Model Optimization on Stock Price Forecasting" by Wang et al. delves deep into the application of LSTMs towards the stock market [9]. Wang et al. evaluate the prediction results on stock data from both a BP neural network and a LSTM neural network [9]. Wang et al. show that LSTM outperforms BP NNs, and that their prediction accuracy rates reach upwards to 65% [9]. They also seek to solve the saw-tooth phenomenon of the gradient descent algorithm [9]. Moreover, Wang et al "defined a parameter combination library and use the skill of dropout to get the more ideal prediction results" [9].

2.2 Techniques Chosen and Why

I chose LSTM because it works well with time series data, both conceptually and in actuality. Moreover, there is an abundance of resources available online for LSTM models versus say trying to implement a genetic algorithm. With the limited time frame I have had to work on this project this semester, it seemed to be the wiser choice to go with a tried and true technical strategy for deliverable sake.

3 BACKGROUND

I will be using the Stock Market dataset available on Kaggle [5]. The total size of the dataset folder is 3GB. Historical pricing information for all NASDAQ stocks and ETFs contained in this dataset stops on April 1st, 2020. Figure 1 shows a sample of 10 ETFs' historical adjusted closing prices from approximately 2005 to 2018. Figure 2 illustrates what the data looks like after importing the Kaggle dataset as a Pandas dataframe. Figure 3 illustrates the technical indicators created as additional features for which the LSTM NN can utilize and learning components.

I calculated various technical indicators to better inform the LSTM model of the stock's "momentum." Said technical indicators are calculated as following:

- 7 day moving average
- 21 day moving average
- Moving average convergence divergence
- Bollinger bands
- Exponential moving average
- 1 day Momentum
- 14 day Relative Strength Index (RSI)

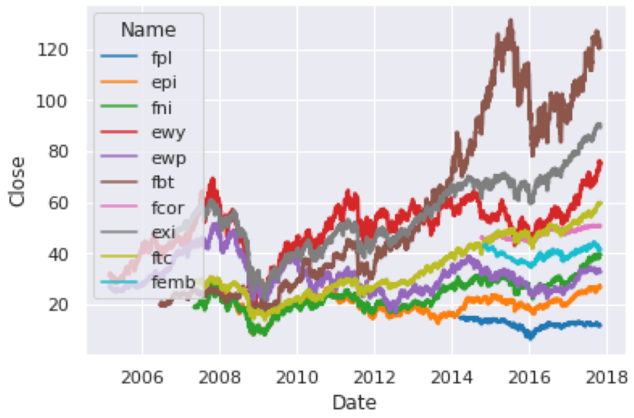
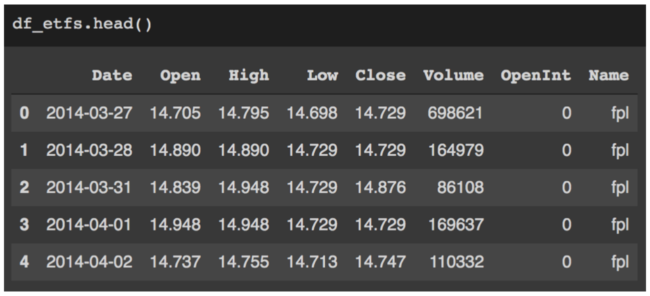


Figure 1: Adjusting Closing Historical Prices of ETFs

4 METHODOLOGY

Task: I believe this term project would be classified as a supervised learning problem. In other words, I am predicting results within a continuous output, which maps input variables to a continuous function. In this situation, the input variables are the four basic daily information columns given as a stock's open, close, high, and low values for the day. From here, more nuanced technical



```
df_etfs.head()
```

	Date	Open	High	Low	Close	Volume	OpenInt	Name
0	2014-03-27	14.705	14.795	14.698	14.729	698621	0	fpl
1	2014-03-28	14.890	14.890	14.729	14.729	164979	0	fpl
2	2014-03-31	14.839	14.948	14.729	14.876	86108	0	fpl
3	2014-04-01	14.948	14.948	14.729	14.729	169637	0	fpl
4	2014-04-02	14.737	14.755	14.713	14.747	110332	0	fpl

Figure 2: Snapshot of Kaggle Dataset

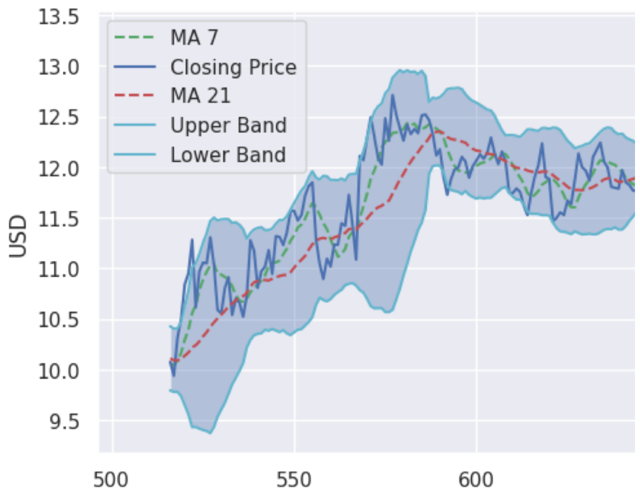


Figure 3: Technical Indicators for an ETF

indicators are calculated as further input variables. Moreover, the output variable would be the stock's closing price where time t = i + 1. I am modelling this behavior by creating a binary label that captures if the stock held or increased versus decreased compared to the previous day.

Performance measure: The performance measure for this project is validation accuracy. I am looking to see how accurate a model can correctly predict which way a stock or ETF will move. Although this is a very basic performance measure, it is one of the most necessary to determine if I have a well performing model. In the future, I hope to implement a couple other performance measures into my model evaluation, which I will talk about in the later sections of this presentation.

Learning component: The learning component of this problem would be the technical indicators of a given stock or ETF. Specifically, the model will learn and implicitly adjust the weights for the stock's technical indicators based on a lookback window of time. For example, one ETF had 916 days of financial history. The first 800 days were used as a lookback window. Then the last 116 days were used to validate the performance of the learning components.

The Talos API offers a way to augment the traditional Keras workflow such that a semi-automated process of finding the best model can take place:

"Talos radically changes the ordinary Keras workflow by fully automating hyperparameter tuning and model evaluation. Talos exposes Keras functionality entirely and there is no new syntax or templates to learn" [6].

"Talos is made for data scientists and data engineers that want to remain in complete control of their Keras models, but are tired of mindless parameter hopping and confusing optimization solutions that add complexity instead of reducing it. Within minutes, without learning any new syntax, Talos allows you to configure, perform, and evaluate hyperparameter optimization experiments that yield state-of-the-art results across a wide range of prediction tasks. Talos provides the simplest and yet most powerful available method for hyperparameter optimization with Keras" [6].

With this in mind, I prepared and tested a normal Keras model with the general neural network architecture that I wanted. Once I validated that the model was working properly, I set the search boundary in the parameters dictionary [6]. Afterwards, to configured and ran the experiment by invoking the Scan() functionality Talos. Subsequently, Reporting() and Predict() functionalities enabled me to analyze and evaluate the result Talos. Once the best model was identified, the Deploy() functionality allowed for the creation of production ready model files. This is extremely useful when it is determined that enough back-testing has been done, so that production-level model can be deployed without constantly being forced to retrain a model. Figure 4 illustrates this general workflow.

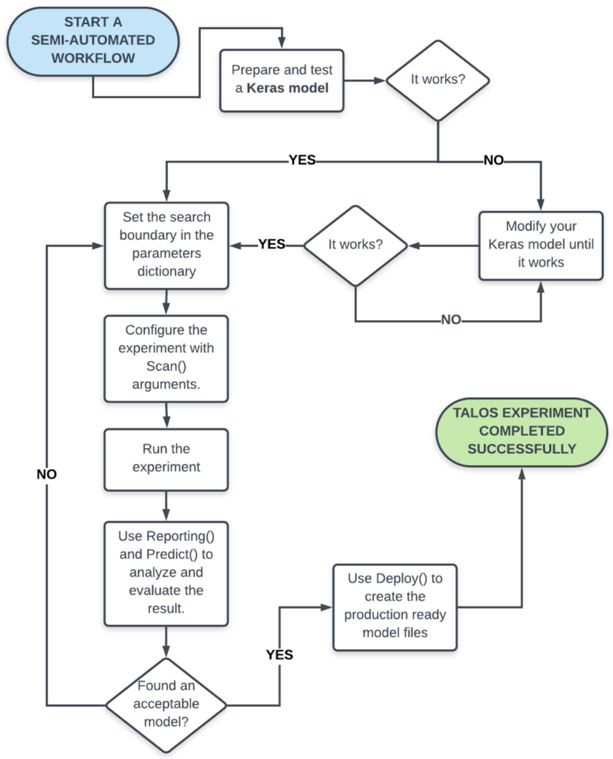

START A SEMI-AUTOMATED WORKFLOW
Prepare and test a **Keras model**
It works?
YES
NO
Modify your Keras model until it works
It works?
YES
NO
Set the search boundary in the parameters dictionary
Configure the experiment with Scan() arguments.
Run the experiment
Use Reporting() and Predict() to analyze and evaluate the result.
Found an acceptable model?
NO
YES
Use Deploy() to create the production ready model files
TALOS EXPERIMENT COMPLETED SUCCESSFULLY

Figure 4: Talos API Workflow

5 RESULTS AND EVALUATION

Validation Accuracy has strong, positive correlations with the following:

- Number of neurons
- Epochs
- Duration of training
- Round of epoch training

These correlation relationships are illustrated in figure 5.

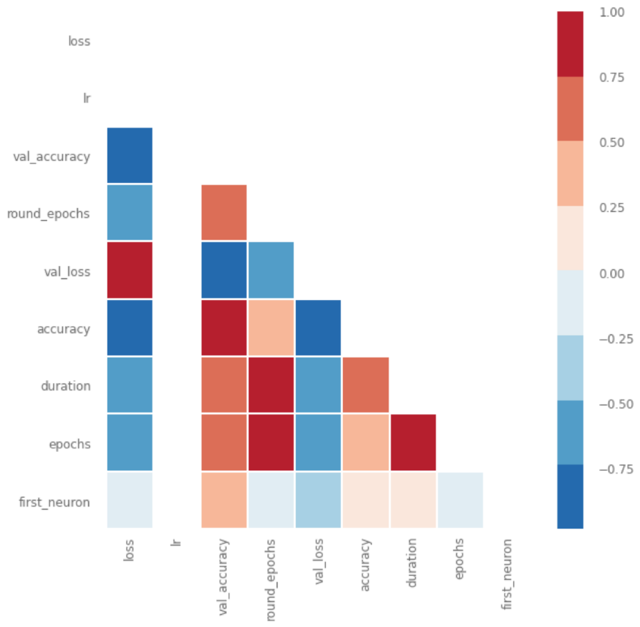


Figure 5: Correlation Heatmap between Performance Metrics and Hyperparameters

With regard to the distribution of validation accuracy among optimizers per each epoch level, overall distribution trend is logarithmic. Please refer to figure 6 for an illustration of this trend. In other words, as epochs increase (x-axis), validation accuracy increases. Moreover, the Adam optimizer has the largest variance, whereas the RMSprop optimizer has the smallest variance.

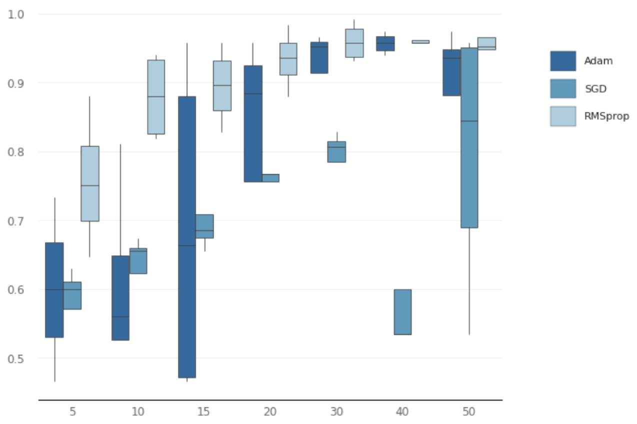


Figure 6: Box Plot of Validation Accuracy by Optimizer per Epoch

By breaking down validation accuracy for each neuron density by epochs per each optimizer, it can be shown in figure 7 that RMSprop performs consistently better in each neuron density. Additionally, SGD seems to struggle at highest neural density; perhaps overfitting? Future work is needed to implement dropout layers and investigate possible feature leakage with regard to learning components.

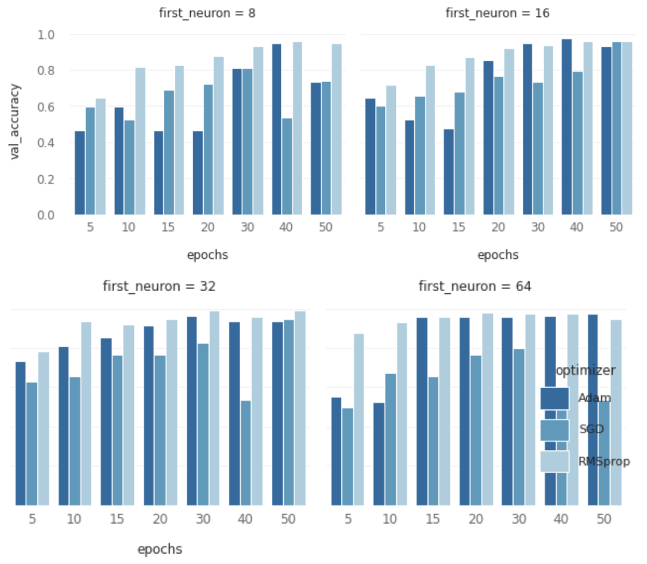


Figure 7: Four Dimensional Bar Grid of Validation Accuracy

6 CONCLUSION

6.1 What is the Best Technique?

After 84 different models were trained, the best model was discovered at round 56. The best model has a validation accuracy of 99.14%. Table 1 shows the best performing hyperparameters. In summary, the best model utilized the highest number of epochs, the lowest gradient step / learning rate, and the RMSprop optimizer. Of note is the fact that the model performed the best using the second highest level of neural density.

Table 1: Hyperparameters of Best Performing Model

Round	ValAcc	LR
56	99.14%	0.0001

Epochs	Optimizer	Neural Density
50	RMSprop	32

6.2 Is the Problem Solved?

The problem of predicting stock market movements is by no means solved. Expecting to solve stock market predictions via a class project is overly ambitious. Especially when many brokers and mathematicians put forth the market following the random walk theory. In other words, the market behaves randomly, thus there is no underlying trend, and consequently any future movements cannot be reasonably predicted [3]. While I still have reservations about subscribing to this theory, I look forward to continuing to tinker with this project. There a plethora of issues to be addressed and that stand in the way of developing a production level model that can stand up to a real world environment.

6.3 Future Research

The existence of overfitting needs to be established, and to what degree it is occurring. Moreover, for time constraint purposes, in the end only one ETF was trained and tested on. The Kaggle dataset is modestly large. Early on in development, one of my approaches attempted to create a rolling lookback window, which created test observation points at given interval. The plan was to apply this to each stock and ETF to create a superset of training and test data. Doing so would mean the model would be unlikely to over-fit to any given stock or ETF's data. The complexity of this approach proved to be difficult, and I was not comfortable in leveraging an existing function from another online resource that does this without fully understanding what was going on under the hood.

I am dubious of the extremely high validation accuracy. Further investigation is needed to determine if there is any sort of feature leakage. In other words, is the model somehow cheating by accidentally using a feature it should not? I presume the rolling weighted averages just give a really good idea for the general momentum of the adjusting closing price, but more due diligence is called for before making a determination.

Any future work with the Talos API necessitates reaching out to the repository's owner, and potentially submitting a pull request. Currently, Talos' Deploy() functionality does not work 100% as intended. In particular, if a three-dimensional model is saved, Talos will write the sample X and Y data as empty. For some reason, this interferes with the Restore() functionality and as a result any attempt to restore a three-dimensional Keras model will not work. Another Github user has already opened an issue for this, so perhaps I will chime in and provide additional insight as to why this is happening.

REFERENCES

[1] Zhi Li Cao, Jian and Jian Li. 2019. Financial Time Series Forecasting Model Based on Ceemdan and Lstm. *Physica a: Statistical Mechanics and Its Applications, vol. 519, 2019, pp. 127–139.* https://doi.org/10.1016/j.physa.2018.11.061
[2] Gopalakrishnan E.A Vijay Krishna Menon M, Hiransha and Soman K.P. 2018. Nse Stock Market Prediction Using Deep-Learning Models. *Procedia Computer Science 2018, 132, 1351–1362.* https://doi.org/10.1016/j.procs.2018.05.050
[3] Burton G. Malkiel. 1973. A Random Walk Down Wall Street (6th ed.).
[4] Gholamreza Mansourfar Nikou, Mahla and Jamshid Bagherzadeh. 2019. Stock Price Prediction Using Deep Learning Algorithm and Its Comparison with Machine Learning Algorithms. In *Intelligent Systems in Accounting, Finance and Management.* https://virginiatech.on.worldcat.org/oclc/8534080303
[5] Oleh Onyshchak. [n.d.]. Stock Market Dataset. In *Kaggle.* https://www.kaggle.com/jacksoncrow/stock-market-dataset
[6] Autonomio Talos. 2019. Talos API. http://github.com/autonomio/talos
[7] Ankit Thakkar and Kinjal Chaudhari. 2019. Crest: Cross-Reference to Exchange-Based Stock Trend Prediction Using Long Short-Term Memory. In *Procedia Computer Science.* https://virginiatech.on.worldcat.org/oclc/8579922574
[8] A. Vishwas, B. V.; Patel. 2020. Hands-On Time Series Analysis with Python : From Basics to Bleeding Edge Techniques. https://virginiatech.on.worldcat.org/oclc/1191051978
[9] Yuying Liu Meiqing Wang Rong Liu Wang, Yifeng. 2018. LSTM Model Optimization on Stock Price Forecasting. In *2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (Dcabes).* https://virginiatech.on.worldcat.org/oclc/7949863345