Machine Learning Engineer Nanodegree

Capstone Project:

Stock price prediction using Long-Short Term Memory (LSTM) Networks

Jacob Anthony

(jacobscottanthony@gmail.com)

March 2nd 2020

I. Definition

Project Overview

The domain of this proposal is finance and more specifically stock price prediction. Price prediction is a hard problem and one that is certainly not solved. Applying machine learning techniques to this problem is well studied (Krolner, et al., 2010). The increase in interest in this area of study has coincided with a simultaneous reduction in computing cost due to Moore's law (Wikipedia, 2020), an increase in Artificial Intelligence research (Wikipedia, 2020) and an increase in value of global markets (Federal Reserve Bank of St. Louis, 2020). Financial markets are ripe for the application of Artificial Intelligence techniques given they are typically data-rich, are vast and varied, are difficult to model through quantitative means and offer a substantial financial reward to players able to successfully develop them.

Stock price prediction is an important problem for any marker participant who wishes to make informed decisions in the market. Thus, there are many reasons to predict the future price of an individual security or an entire segment of the market.

Most price predictions assume a relationship between past prices $(t_0 \ \rightarrow \ t_{-n})$ or their correlates and future prices $(t_{+1} \rightarrow t_{+n})$ in the form of:

$$\$_{t+1} = f(\$_t \rightarrow \$_{t-n})$$

In the simplest case, one can start with the assumption that future price of an asset is a function of previous prices. In more complex cases one can include many other leading and lagging correlates of price and include these in a model.

This project explores methods for predicting the closing price of Ecolab Inc which trades on the New York Stock Exchange under the ticker NYSE: ECL. The project started with developing a baseline statistical model, then a univariate, multistep LSTM model and progresses to building a multivariate, multistep LSTM model. The performance of each model is compared to both a benchmark model and the baseline model.

Problem Statement

Stock price prediction is a crucial skill for any investor who wishes to make informed decisions in the market. Predictions of some kind are a part of (virtually) all trading decisions or strategies. Supervised Machine Learning techniques that use artificial neural network units such as the Long Short-Term Memory (LSTM) cell are well suited to these kinds of time-series prediction problems and can help to generate predictions of the market which in turn help to improve decision making for traders.

A recurrent neural net (RNN) using LSTM cells will be implemented to predict the next 30 days of Closing prices for NYSE: ECL using the previous 30 days of data.

Metrics

This problem is a supervised regression problem where we are predicting the future price of a stock over a defined horizon and then comparing this prediction with the actual stock price for that period. As such we could have chosen a number of common regression metrics. For this project, all models have been evaluated using:

- Mean Absolute Error, MAE
  - The MAE gives an understanding of the precision of the model in the same scale as the underlying data without looking at the direction of the error. Low MAE indicates high model skill.
  - The scikit-learn (Pedregosa, 2011) implementation of MAE has been used throughout the project.
    - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error
- Mean Squared Error, RMSE
  - The RMSE gives an understanding of the precision of the model in the same scale as the underlying data. Due to its squaring of the error, it gives a higher weight to large errors. This metric is typically preferred where even occasional large errors cannot be tolerated. Low RMSE indicates high model skill.
  - The scikit-learn (Pedregosa, 2011) implementation of RMSE has been used throughout the project.
    - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error
- Pearson Correlation, PCC
  - The PCC gives an understanding of whether the model is well correlated with the underlying data.
  - A PCC of 1.0 would indicate the model perfectly models the underlying data.
  - The pandas (McKinney, 2011) implementation of PCC has been used throughout the project.
    - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

In some instances, these metrics are applied to a single prediction step and in other instances they have been applied over a series of predictions. This has been made explicit throughout the report.

II. Analysis

Data Exploration

This project started with purchasing an Open, High, Low, Close, Volume (OHLCV) dataset for the S&P500 going back to the year 2000. The data has a one-minute resolution. OHLCV is commonly used to describe a trading session, typically a day, and describes the range of prices the stock was traded at during the session as well as the volume that was traded during the session. For the dataset purchased, a session lasts one minute. This means there are approximately 1.9M OHLCV time-series datapoints for a stock that has been in the index for the entire 20-year period.

The dataset covers all current (and past) stocks in the S&P 500 index. This index is comprised of companies that are typically established, large cap organizations that have a history of strong performance and are a leader in their respective industries. Thus, their price typically doesn't swing wildly from day to day.

The OHLC portion of the data is commonly represented as a candlestick, as adjacent with the y-scale representing the price.

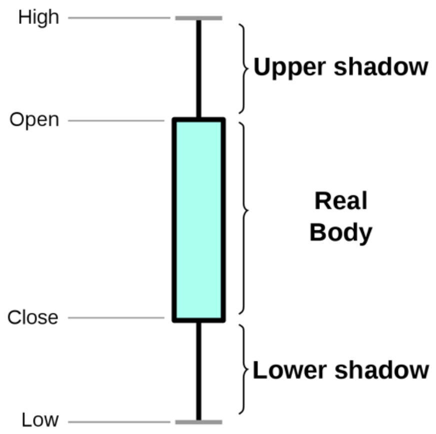
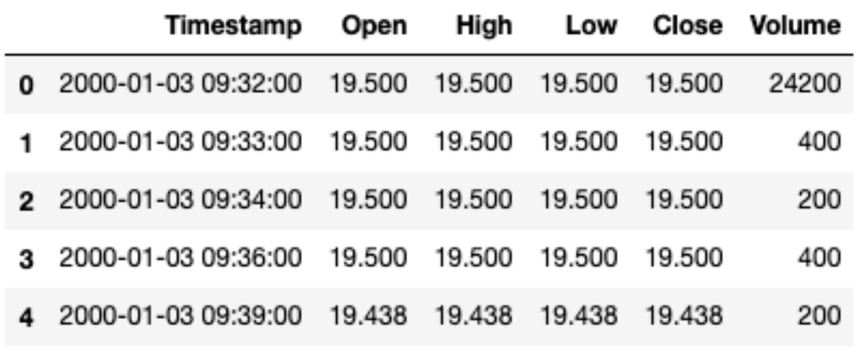


Figure 1 - Example OHLC Candlestick

Ecolab Inc.

Ecolab Inc was chosen as the target stock simply because I used to work for them. No consideration was given to it possessing any specific desirable (or undesirable) statistical properties.

The first five rows of the raw data are presented below in Figure 2.



	Timestamp	Open	High	Low	Close	Volume
0	2000-01-03 09:32:00	19.500	19.500	19.500	19.500	24200
1	2000-01-03 09:33:00	19.500	19.500	19.500	19.500	400
2	2000-01-03 09:34:00	19.500	19.500	19.500	19.500	200
3	2000-01-03 09:36:00	19.500	19.500	19.500	19.500	400
4	2000-01-03 09:39:00	19.438	19.438	19.438	19.438	200

Figure 2 - First five rows of the raw data from ECL Inc.

As can be seen in Figure 3, the price of Ecolab has risen steadily from 2000 through the end of 2019 from ~US$20 / share to ~$200 / share.

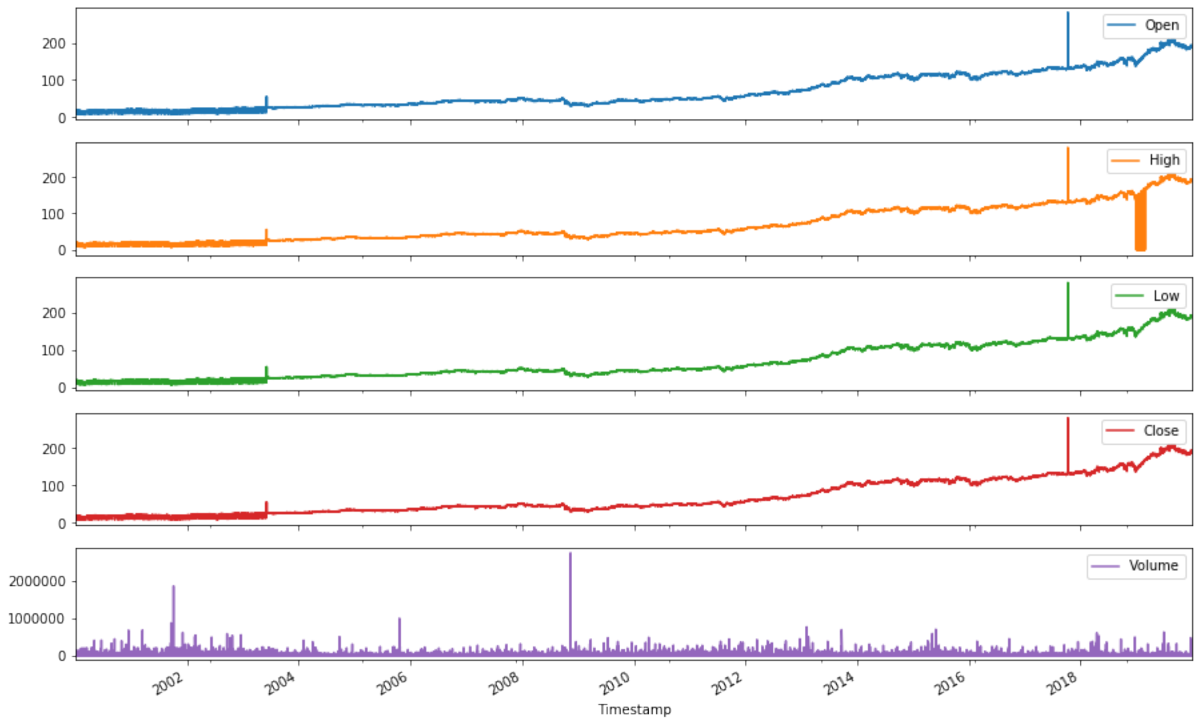


Figure 3 - ECL OHLCV from 2000 through 2019

From Figure 3 it's clear there are a number of artefacts (for eg. High Price < Low Price) in the data that need to be addressed before we can consider the data ready for use in any modelling. These are all expanded on in the Data Pre-Processing section.

Table 1 - Descriptive Statistics for ECL OHLCV at minute resolution

	Open	**High**	**Low**	**Close**	**Volume**
count	1778998	1778998	1778998	1778998	1778998
mean	71.44	71.45	71.42	71.44	3102
std	47.59	47.60	47.59	47.59	7345
min	7.06	0.00	7.06	7.08	0
25%	34.47	34.47	34.45	34.47	700
50%	48.97	48.99	48.96	48.97	1564
75%	112.43	112.45	112.41	112.43	3300
max	281.76	281.76	281.76	281.76	2749482

The minute resolution data described in Table 1 shows:

- There are ~1.8M datapoints.
- Open, High, Low, Close all share a close clustering of mean, std, min and max which suggests that all prices move in unison.
- Volume potentially suffers from some outliers given the max is 850 times the mean.

- There are a few artefacts in there that will need to be remedied during pre-processing.

Table 2 - Descriptive Statistics for ECL OHLCV at day resolution

	Open	High	Low	Close	Volume
count	5031	5031	5031	5031	5031
mean	67.78	68.70	66.86	67.81	1086004
std	48.11	47.97	48.21	48.11	883396
min	7.12	7.63	7.06	7.20	118500
25%	32.09	32.33	31.78	32.02	597400
50%	46.76	47.22	46.17	46.72	862664
75%	109.78	110.51	109.13	109.95	1311234
max	209.22	209.87	206.43	208.55	23743910

The daily resolution data in Table 2 shows:

- There are 5031 trading days in the period from 2000 to 2019.
- A greater spread on the mean, std, min and max of Open, High, Low, Close. This is expected given each trading session is expanded to a whole day.

Technical Indicators

In order to build on the dataset above, I have also added a collection of Technical Indicators to the OHLCV dataset. These additional features are all a function of the OHLCV data and describe various qualities of the data such as their moving average. It is believed these additional features will improve model skill when added to the dataset. The indicators added are listed below in Table 3.

Table 3 - Technical Indicators used

Technical Indicator	Function of
Simple Moving Average	Close
Exponential Moving Average	Close
Relative Strength Index	Close
Rate of Change	Close
Moving Average Convergence / Divergence	Close
Williams Oscillator	High, Low, Close
Average True Range	High, Low, Close

Definitions of how each of these are calculated can be found in the documentation of the Technical Analysis in Python library (Padial, 2018).

Exploratory Visualization

Below two charts are provided that show how the Exponential Moving Average (Period 10 and Period 50) and Moving Average Convergence/ Divergence (MACD) are implemented in the processed data.

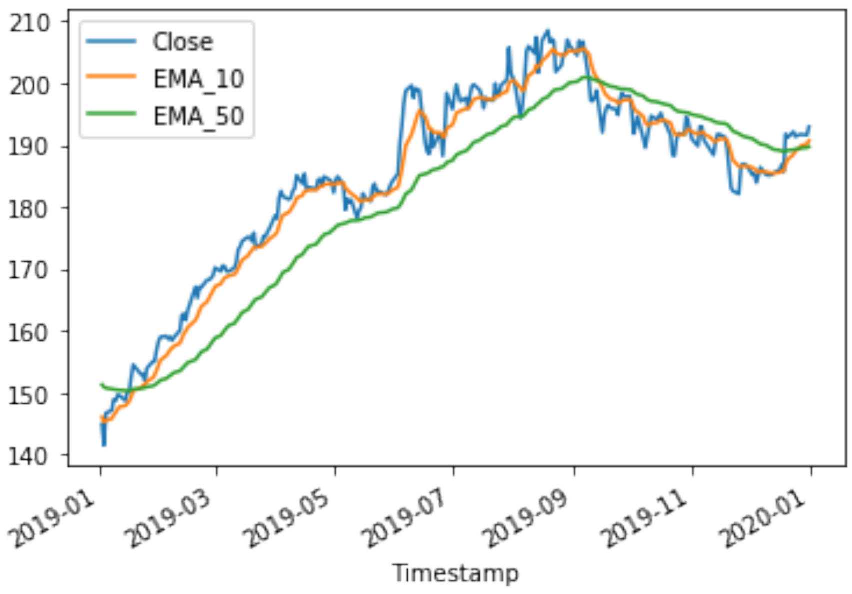


Figure 4 - Example of EMA Technical Indicator

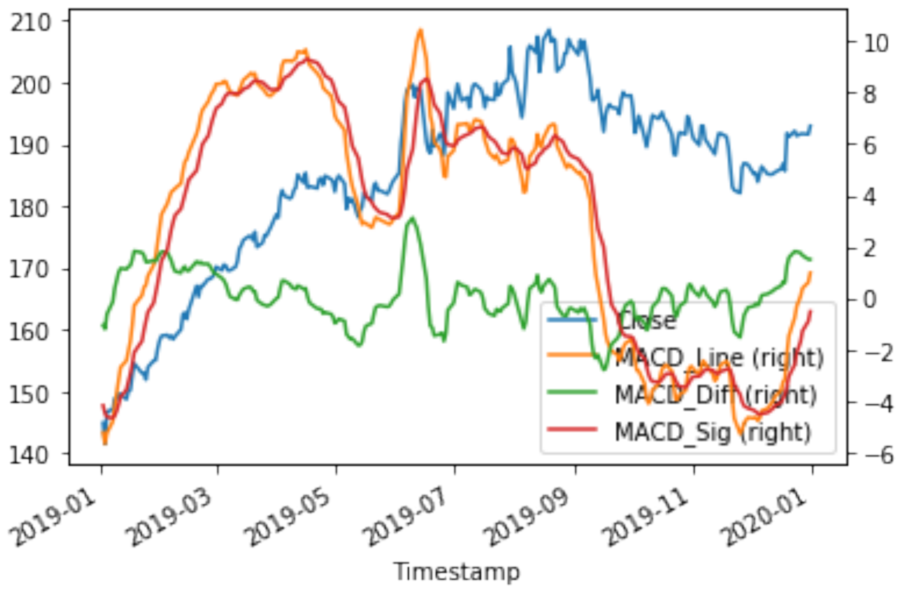


Figure 5 - Example of MACD Technical Indicator

After adding the indicators to the dataset, a Principal Components Analysis (PCA) was conducted to understand which variables explain the variance in the Close price. For the ECL dataset, this is presented in Figure 6.

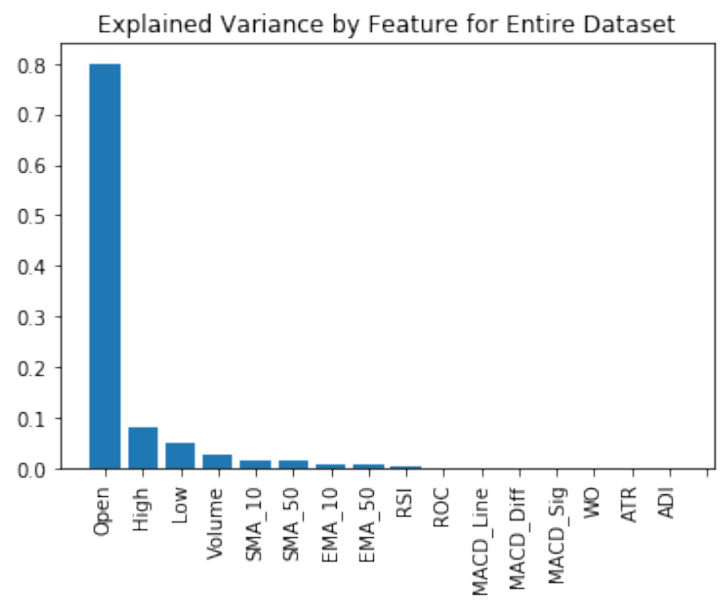


Figure 6 - Explained Variance by feature for ECL

Perhaps unsurprisingly, the OHLCV and moving averages (Rolling and Exponential) explain > 99% of the variance in Close price. It's crucial to note that this was conducted on the entire dataset before it was restructured into a supervised learning dataset. I suspect there may be years where correlations wax and wane. Therefore, this PCA result will not be used to reduce input dimensionality.

Algorithms and Techniques

There are two algorithms implemented in this project. One is used as a baseline and another which aimed to outperform the baseline (and benchmark). The choice of model in both cases was preceded with research into what is commonly used or considered state of the art.

Baseline Model

For the Baseline model, I have used an implementation of an Autoregressive Integrated Moving Average (ARIMA) model.

An excellent presentation of how the ARIMA model processes data can be found here (Brownlee, 2017). For my project the implementation makes use of all three components of the ARIMA model, typically referred to as the p, d, q parameters. The initial model order has been determined from autocorrelation and partial-autocorrelation plots.

The ARIMA model is one of the most widely used models for univariate time series forecasting. The model can handle data with trends but it not skilful if the model includes seasonality. It was selected primarily because it is common, secondarily because it's suitable for the problem and thirdly because it's simple to implement.

Deep Learning Model

For the deep learning model, I have created implementations of Long-Short Term Memory (LSTM) recurrent neural nets (RNN). I have used a stacked LSTM architecture that uses multiple layers of LSTMs followed by multiple dense layers and finally a dense output layer.

The LSTM cell is used exclusively within RNNs. Unlike (typical) feedforward neural nets, RNNs also have connections between the neurons that point backward. This means that at every state the neuron receives the vector at timestep, t and the previous time step, t-1. This property gives each neuron a form of memory.

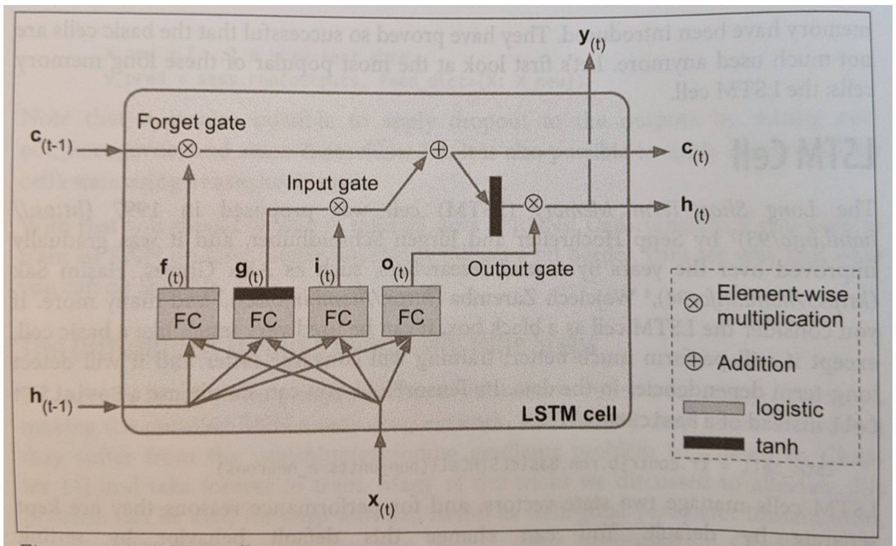


Figure 14-13. LSTM cell

Figure 7 – (Figure 14-13) from (Geron, 2019) showing an LSTM Cell

The following description relies heavily on the example presented in Chapter 14 of (Geron, 2019).

Within the LSTM there are a number of gates. The key idea is that the cell can learn what to store in the long-term state $C_{(t)}$, what to store in the short-term state $h_{(t)}$, what to throw away and what to read from it. The basic flow is as follows:

- The input vector $x_{(t)}$ and the previous short-term state $h_{(t-1)}$ are fed into the four fully connected layers (depicted as FC)
- The main layer (depicted with tanh activation) outputs $g_{(t)}$ which goes directly to $y_{(t)}$ and $h_{(t)}$
- The other three layers are gate controllers:
    - The forget gate (controlled by $f_{(t)}$) controls which parts of the long-term state should be forgotten
    - The input gate (controlled by $i_{(t)}$) controls which part of $g_{(t)}$ should be added to the long-term state
    - The output gate (controlled by $o_{(t)}$) controls which parts of the long-term state should be read and output at the current timestep to $h_{(t)}$ and $y_{(t)}$

In summary, the LSTM can learn to recognize important input (input gate), store it in the long-term state, learn to preserve it for as long as it is needed (forget gate) and learn to extract it whenever it is needed.

For time series forecasting, LSTMs are considered state of the art (Spyros Makridakis, 2018). Figure 8 shows LSTMs are in the top 3 model types for one-step forecasts.

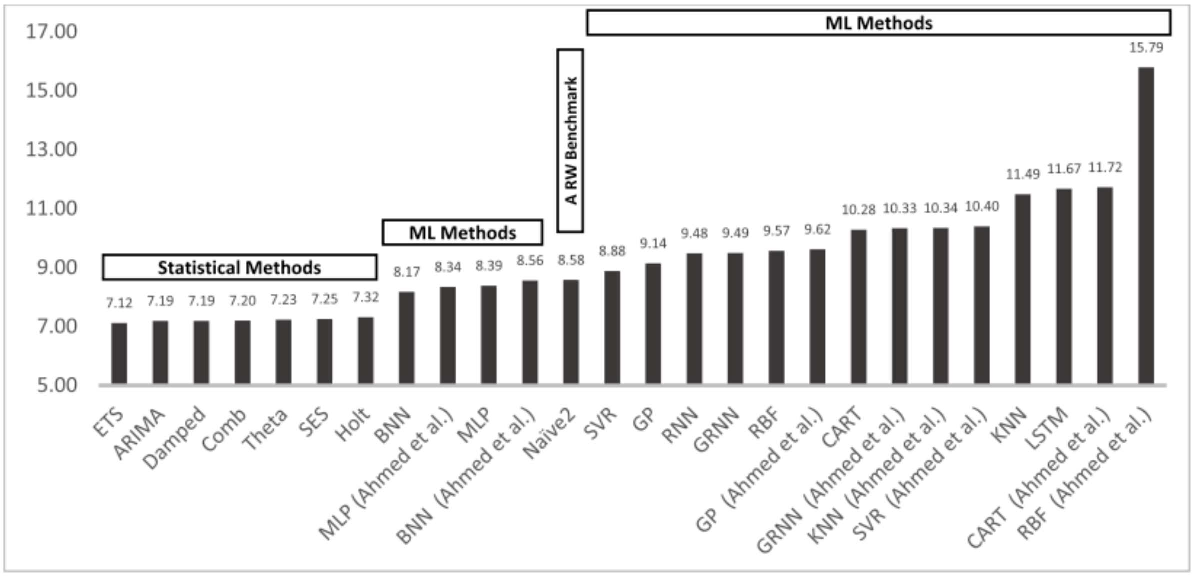


Figure 8 - Bar chart comparing model performance in (Spyros Makridakis, 2018)

I have chosen LSTMs as they are the current state-of-the-art and for their applicability to time series forecasting. In order to arrive at a starting architecture, I have relied on the examples in Chapter 9 of (Brownlee, 2020). I have then iterated on these architectures seeking improved model performance for my task.

Benchmark

I am proposing one benchmark model to compare the results achieved by this project.

"Stock Prices Prediction using Deep Learning Models" (Liu, et al., 2015)

- https://arxiv.org/pdf/1909.12227

This team trained an LSTM network to achieve a MAPE prediction accuracy of ~ 0.01 or 1% using daily OHLCV data for the S&P500 index.

This baseline shares similar input data structure, uses similar error measures and a similar model architecture.

III. Methodology

Data Preprocessing

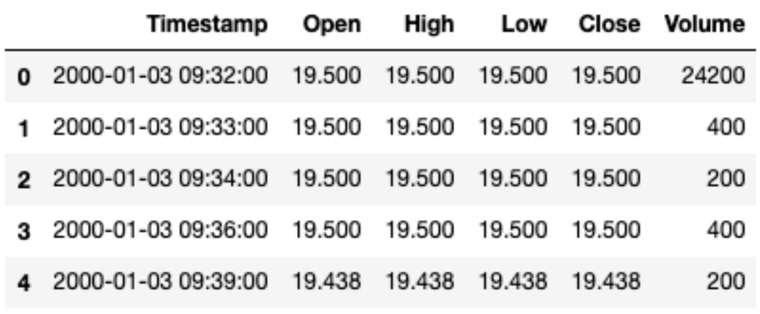
OHLCV Data

After exploring the data by charting each year I found a number of artefacts that needed to be addressed. These were:

1. Missing Timesteps
2. Unstable Price until 2003
3. Odd Trading Day in 2017
4. High Price < Low Price
5. Create Hour and Day Resolution from Minute Resolution

Missing Timesteps

As Figure 9 shows, all Timestamps with Volume = 0 had been removed from the dataset.



	Timestamp	Open	High	Low	Close	Volume
0	2000-01-03 09:32:00	19.500	19.500	19.500	19.500	24200
1	2000-01-03 09:33:00	19.500	19.500	19.500	19.500	400
2	2000-01-03 09:34:00	19.500	19.500	19.500	19.500	200
3	2000-01-03 09:36:00	19.500	19.500	19.500	19.500	400
4	2000-01-03 09:39:00	19.438	19.438	19.438	19.438	200

Figure 9 - First five rows of data showing missing timestamps

The company I purchased the data from explained this was a deliberate choice. However, for this work, having those timestamps was important as they may have some explanatory power. Ie. if the Volume = 0, the price cannot move. Therefore, I have followed these steps:

- Create a dataframe with a DateTimeIndex that mimics the trading calendar for the NYSE using the *pandas_market_calendars* library (Sheftel, 2019).
- Using the pandas merge_asof function, merged the trading calendar dataframe with the existing OHLCV dataframe with direction='nearest'.
- I then created a list from all the timestamps present in the OHLCV dataframe.
- Using numpy where and isin functions I iterated over the trading calendar dataframe and replaced any timestamps that weren't in the OHLCV dataframe with Volume = 0.

Unstable Price until 2003

Originally, I reached out to the data vendor asking for an explanation why the price was jumping between ~$10 – ~$20 in exact $10 increments from minute to minute.

They were not able to provide an explanation. However, after looking at how much data I had, I decided to cut the dataset down to 10 years (2010 – 2019). Thus, this artefact could be ignored.

Odd Trading Day in 2017

Looking at the yearly charts, I could see a few days where the trading price spiked +$100 for a single minute. After checking other online data sources, I determined this was an artefact and not a real data point. Therefore, I located the index where the odd values occurred and replaced this timestamp with an interpolated datapoint from the nearest neighbours.

High Price < Low Price

From the charts I could see there were some timestamps where the High Price was lower than the Low Price for a given minute. Obviously, this isn't logically possible. Therefore, I created a simple lambda formula to iterate over all the rows and replace any High values with the Low value for that timestamp.

Creating Hour and Day

Given the dataset is still at a minute resolution, I used the pandas resample function to create the Hourly and Daily datasets from the minute resolution data.

Technical Indicators

After the pre-processing was completed, I then added the Technical Indicators listed in Table 3. This was completed using the pandas Technical Analysis library (Johnson, 2019). Once you have a neatly formatted OHLCV dataframe, you can add a whole range of indicators with a single line of code. I chose a range of papers based on the work of (Krolner, et al., 2010).

Implementation

The implementations progressed iteration by iteration from the baseline model to a univariate, multistep model to a multivariate, multistep model. The iterations follow a logical series of steps which corresponded with increasing complexity of the model itself and its instantiation into code.

1. ARIMA Baseline

Adapting an example in Chapter 13 of (Brownlee, 2020), I worked up an implementation of an auto-ARIMA. Once I had the ARIMA trained, I then used walk-forward validation to walk the model through the test set with a step size of 30. The prediction was then plotted against the actual values and the errors were calculated.

The full implementation is named: *ECL ARIMA Baseline.ipynb*

N.B. An implementation of the Facebook Prophet model was also created as an alternative baseline model. This was not taken forward but can be found in the Archive folder.

2. **Univariate, Multistep**

Using an example in Chapter 9 of (Brownlee, 2020), I worked up an implementation of a stacked LSTM for the problem I had specified. To begin with, I focussed on working up an entire notebook that would run in a reasonable time on my laptop and provide all the outputs I wanted rather than optimizing the LSTM. I deliberately left the optimization for a separate notebook.

The full implementation is named: *ECL Univariate, Multistep LSTM.ipynb*

3. **Multivariate, Multistep without Scaling**

With an example from Machine Learning Mastery (Brownlee, 2018), I worked up an implementation based on OHLC data for my problem. This allowed me to extend the previous example without scaling as all these variables are of the same magnitude. As before, I started with initializing a very simple LSTM and demonstrated a prediction. Once this was working, I put the model into a walk forward validation on the test set. After the walk forward, I created the same charts and error calculations as the univariate example.

The full implementation is named: *LSTM Question_Resolved with Volume.ipynb*

N.B. There were a number of dead ends and failed attempts to get this working as desired. These can all be found in an unformatted state in the Archive folder.

4. **Multivariate, Multistep with Scaling**

Extending the previous example, I first added only the Volume into the dataset. This increased the input variables to five (OHLCV) and a single output variable. As Volume is a vastly different magnitude to the OHLC data, I had to scale the dataset. I initially had a lot of trouble working out at which point of the data preparation this should be applied. The confusion revolved around reshaping the predictions and X_test data back into the right shape to invert the scaling. After making a post on r/MLQuestions (ref), I took their advice and initialized two separate scalers for X and y which made reshaping after the predictions much simpler.

The full implementation is named: *ECL Multivariate, Multistep LSTM with Scaling.ipynb*

5. **Grid Searching using Talos**

After arriving at working implementations of the Baseline, Univariate and Multivariate iterations, I returned back to optimizing the LSTM architecture to vastly improve the model performance. In order to make this process simple, I have used the Talos library (Anon., 2020). This library allows you to specify an architecture and make (almost) any Keras hyperparameter a search target. I setup multiple different Talos grid searches looking to optimize validation loss in order to arrive at the architecture used.

This was implemented and run on a Google Cloud VM. The full implementation can be found in the Talos Optimization folder.

6. **Final Implementations**

After I arrived at an architecture I was satisfied with, I copied it back into the working implementations and ran the Notebooks to validate performance.

Refinement

In the first phase of the project, the focus was to arrive at robust code that would train a model, test it and evaluate its performance. Once I had that code, I then switched gears to look at improving performance.

The initial LSTM, referred to as "Vanilla LSTM" in the code has a single layer of 50 LSTM nodes followed by a Dense layer with 30 neurons. This baseline was taken from (Brownlee, 2020). The model summary is included below as Figure 10.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 50)                10400
_________________________________________________________________
dense_1 (Dense)              (None, 30)                1530
=================================================================
Total params: 11,930
Trainable params: 11,930
Non-trainable params: 0
```

Figure 10 - Model.Summary() for the Vanilla LSTM

During the Talos grid searching, I explored five key directions:

- 3 stacked LSTM layers
  - 3 of the iterations surveyed were able to produce a validation loss < 100. Their typical features included larger LSTM neuron counts (200 – 400), larger batch sizes and no dropout.
- 3 stacked LSTM layers with Batch Normalization
  - 10 of 36 iterations surveyed were able to produce a validation loss < 100. Their features included lower LSTM neuron count (100 – 200) and some dropout.
- 3 stacked LSTM layers, 1 Dense layer
  - None of the 16 iterations surveyed were able to produce a validation loss < 100.

- 3 stacked LSTM layer, 3 Dense layers
    - 2 of the 9 iterations surveyed were able to produce a validation loss of < 100. Their features included low LSTM neuron count, high dense neuron count, large batch sizes and no dropout for either LSTM or Dense layers.
- 6 stacked LSTM layers
    - 3 of the 18 iterations surveyed were able to produce a validation loss < 100. Their typical features included lower neuron count (100 – 200), large batch sizes and no dropout.

After approximately 100 iterations in multiple directions, I decided to move forward with the 3 stacked LSTM layers and 3 Dense layer architecture. This produced the best overall result. The model summary is included below as Figure 11.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 30, 100)           40800
_________________________________________________________________
lstm_3 (LSTM)                (None, 30, 100)           80400
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_2 (Dense)              (None, 400)               40400
_________________________________________________________________
dense_3 (Dense)              (None, 400)               160400
_________________________________________________________________
dense_4 (Dense)              (None, 400)               160400
_________________________________________________________________
dense_5 (Dense)              (None, 30)                12030
=================================================================
Total params: 574,830
Trainable params: 574,830
Non-trainable params: 0
```

Figure 11 - Model.Summary() for the Final Model Architecture

Although it is no longer shown in the Talos notebooks, I also experimented with:

- Number of epochs
- EarlyStopping
- Other optimizers and their learning rate parameters
- Other LSTM activations

None of these parameters were consistently improving performance, so I reverted to using the default iterations provided within the Keras library.

IV. Results

Model Evaluation and Validation

All models have been compared using the same suite of error calculations. This starts with comparing the MAE and RMSE for each step of the walk forward validation. Then I dive deeper into the first and last step of the model performance to further probe their performance.

Baseline Model

Table 4 summarizes the baseline model performance for the entire walk forward.

Table 4 - Baseline model performance summary

	Mean Absolute Error	**RMSE**
mean	3.86	28.51

Figure 12 shows the model performance by step (iteration) in the walk forward validation.

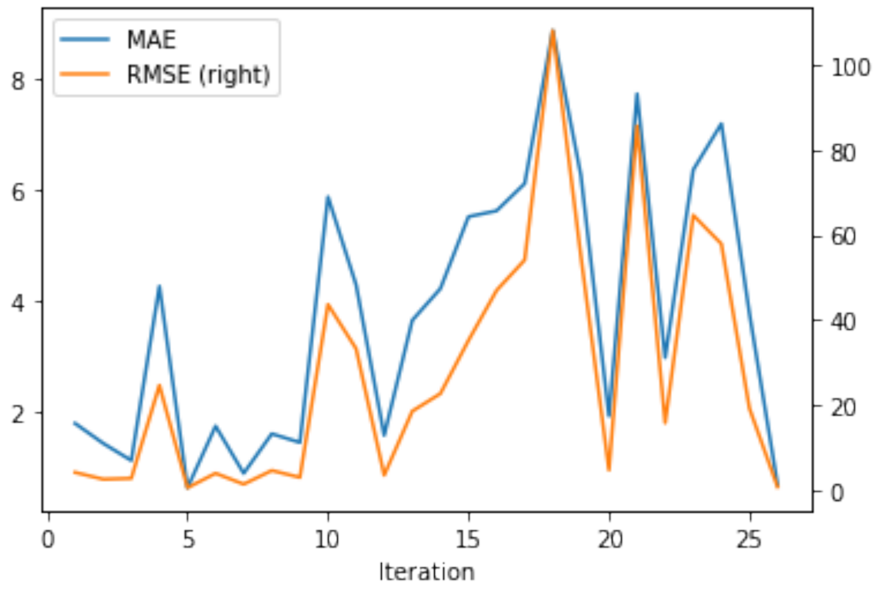


Figure 12 - Baseline Model Performance by Iteration

Figure 13 shows the model performance by prediction horizon.

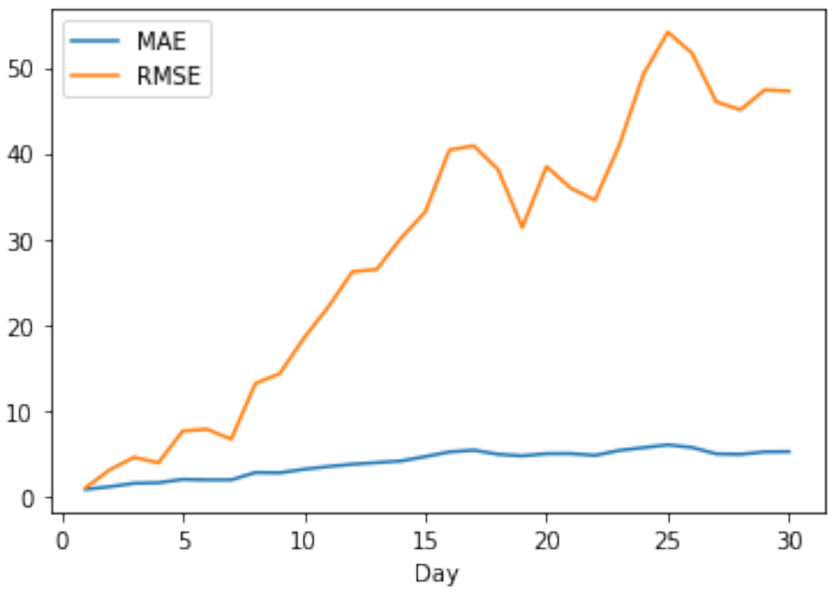


Figure 13 - Baseline model performance by prediction horizon

Figure 14 shows the model predictions versus expected prices for the entire walk forward.

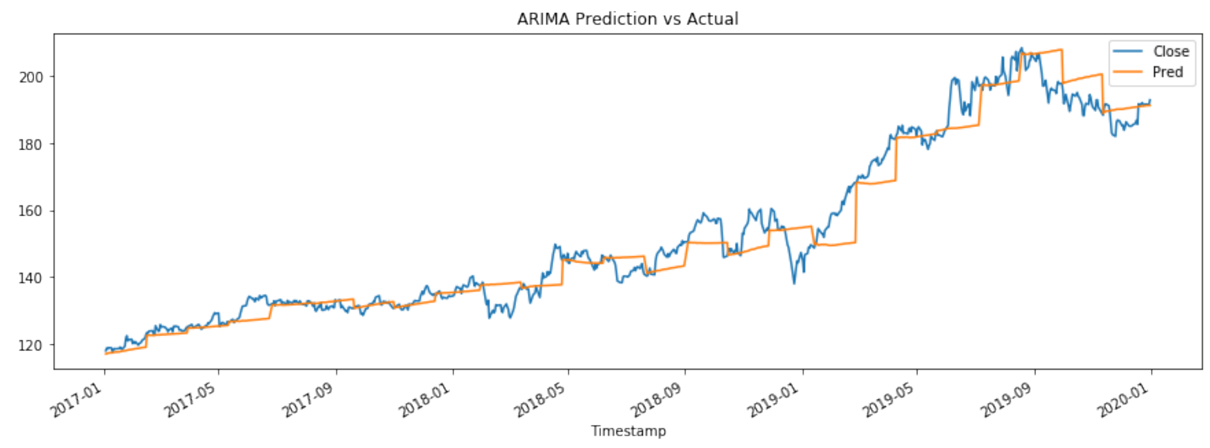


Figure 14 - Baseline Model Predictions vs Actual

Univariate, Multistep Model

Overall Model

Table 5 summarizes the univariate model performance for the entire walk forward.

Table 5 – Univariate model performance overview

	Mean Absolute Error	RMSE
mean	15.63	1559.93
std	21.56	5862.41
min	6.18	65.35
max	113.76	29963.14

Figure 15 shows the model performance by step (iteration) in the walk forward.

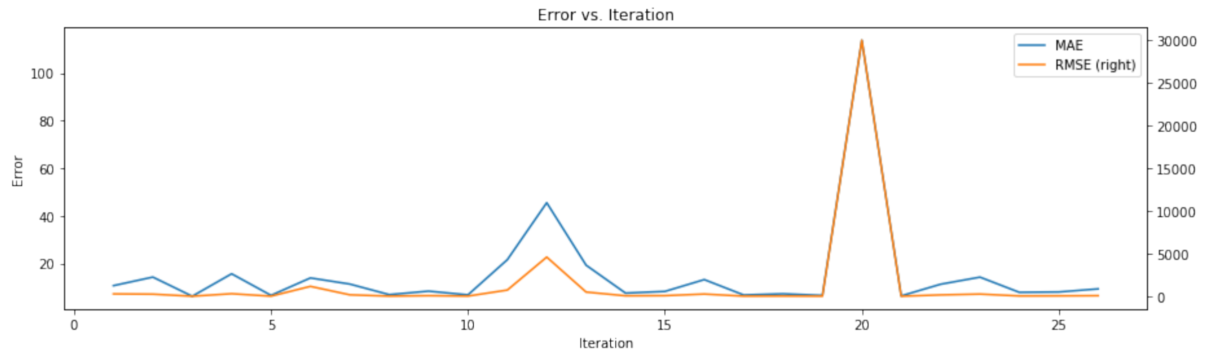


Figure 15 - Error by Iteration for Univariate Model

First Iteration (Least Training Data)

Figure 16 shows the composite prediction versus the actual closing price for the first model.

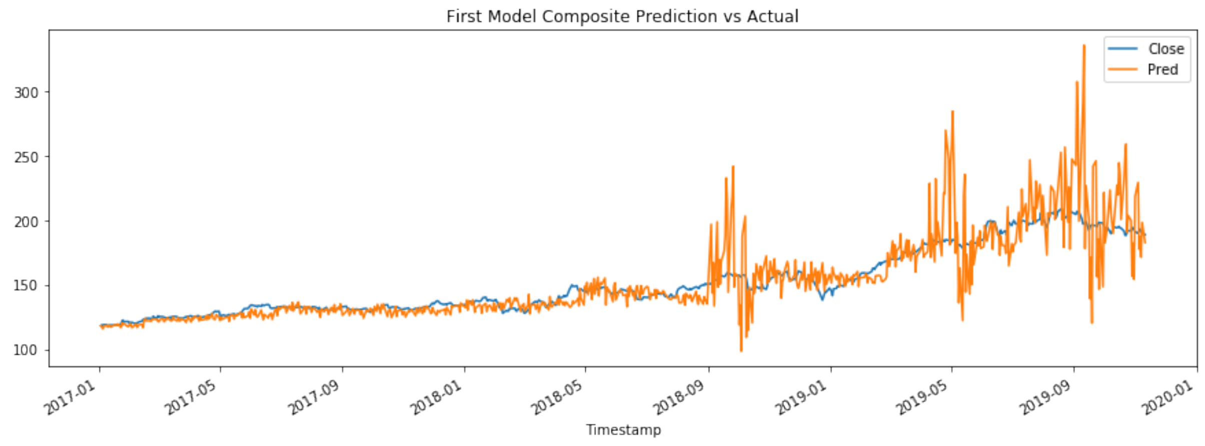


Figure 16 - Univariate, first iteration composite prediction vs actual

Table 6 summarizes the performance of the composite line in Figure 16.

Table 6 - Univariate, first iteration composite errors

Mean Absolute Error	9.54
Mean Squared Error	324.70
Pearson Correlation	0.85

Figure 17 shows the error versus the prediction horizon for the first model.

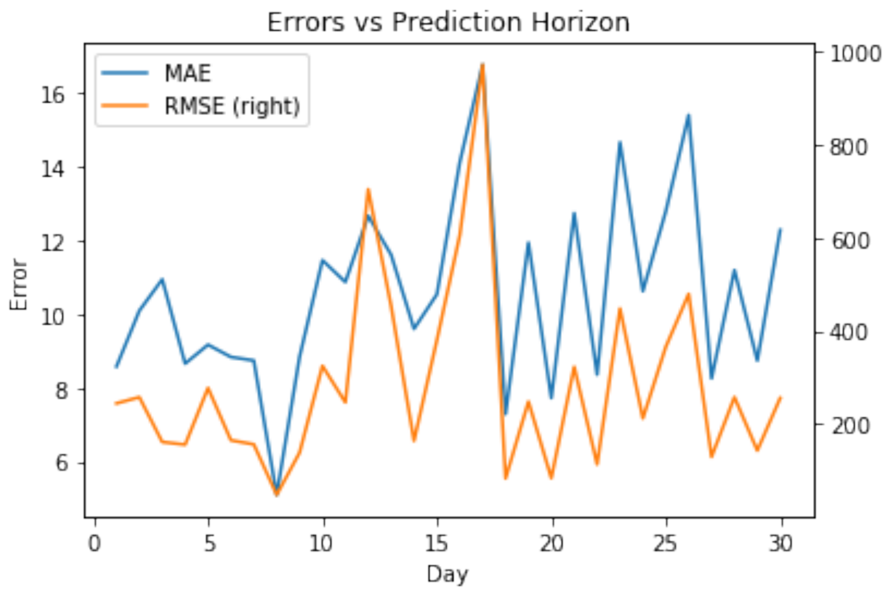


Figure 17 - Univariate, first iteration MAE by Prediction Horizon

Last Iteration (Most Training Data)

Figure 18 shows the composite prediction versus the actual closing price for the first model.

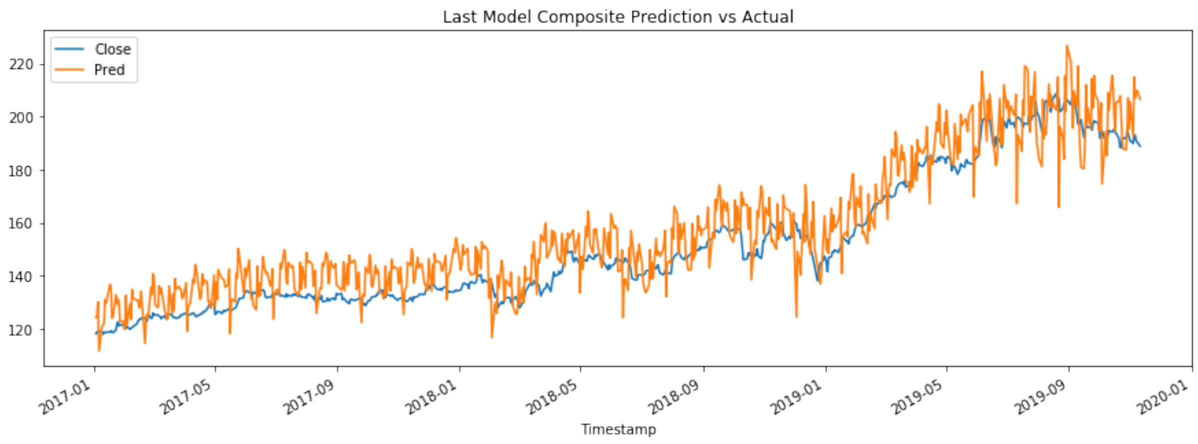


Figure 18 - Univariate, last iteration composite prediction vs actual

Table 7 summarizes the performance of the composite line in Figure 18.

Table 7 - Univariate, last iteration composite errors

Mean Absolute Error	8.60
Mean Squared Error	109.57
Pearson Correlation	0.94

Figure 19 shows the error versus the prediction horizon for the first model.

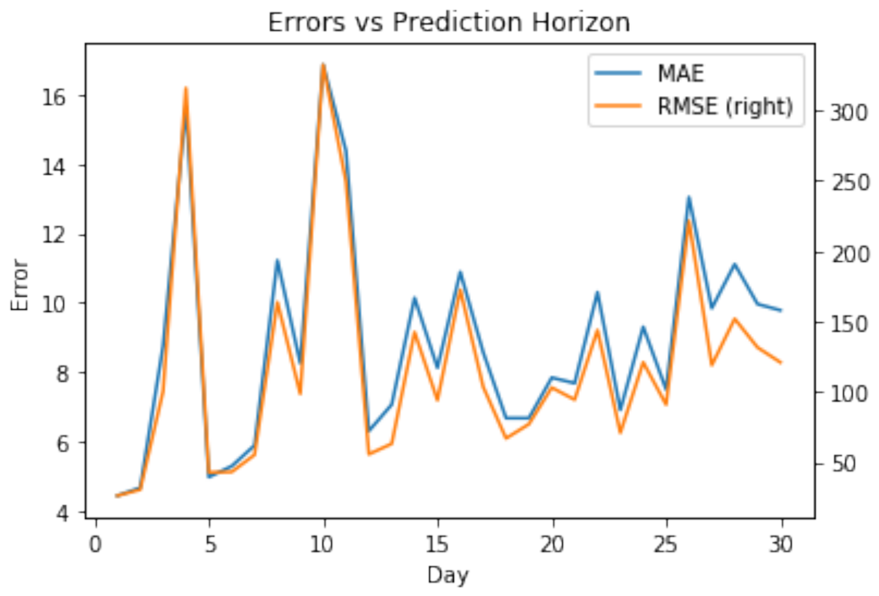


Figure 19 - Univariate, last iteration MAE by Prediction Horizon

Summary

- The first iteration becomes unstable when the price begins to steadily rise in early 2019. This is possibly due to the model not seeing a behaviour like this in the training data. We can see in the last iteration, the model has learned something about this behaviour as it is much more stable.
- As expected, the model has improved between the first and last step of the walk forward. This is seen in the improved MAE, RMSE and Pearson scores.
- Unexpectedly, the model has a less pronounced drop off in performance for the increased time horizon when comparing it to the baseline model.

Multivariate, Multistep

Overall Model

Table 8 summarizes the multivariate model performance for the entire walk forward.

Table 8 - Multivariate model performance overview

	Mean Absolute Error	**RMSE**
Mean	7.831	150.00
Std	3.07	176.28
Min	5.19	45.02
Max	16.98	804.39

Figure 20 shows the model performance by step (iteration) in the walk forward.

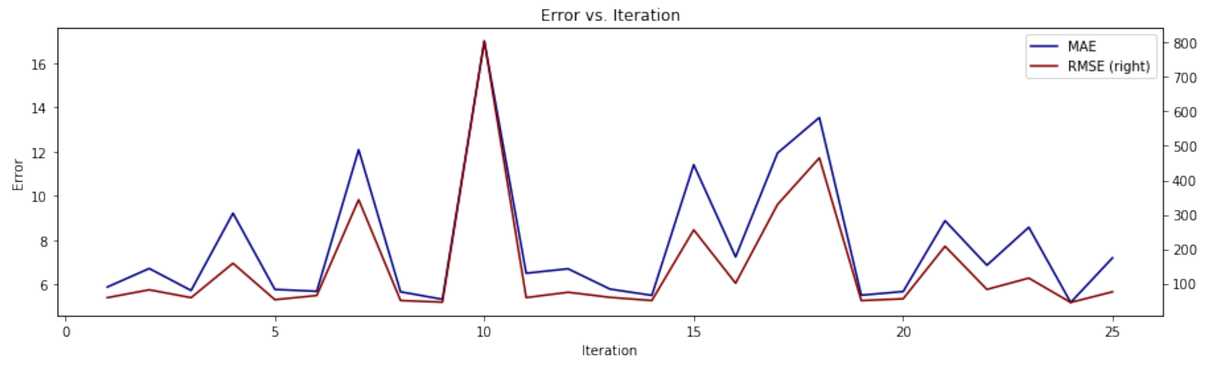


Figure 20 – Error by iteration for the multivariate model

First Iteration (Least Training Data)

Figure 21 shows the composite prediction versus the actual closing price for the first model.

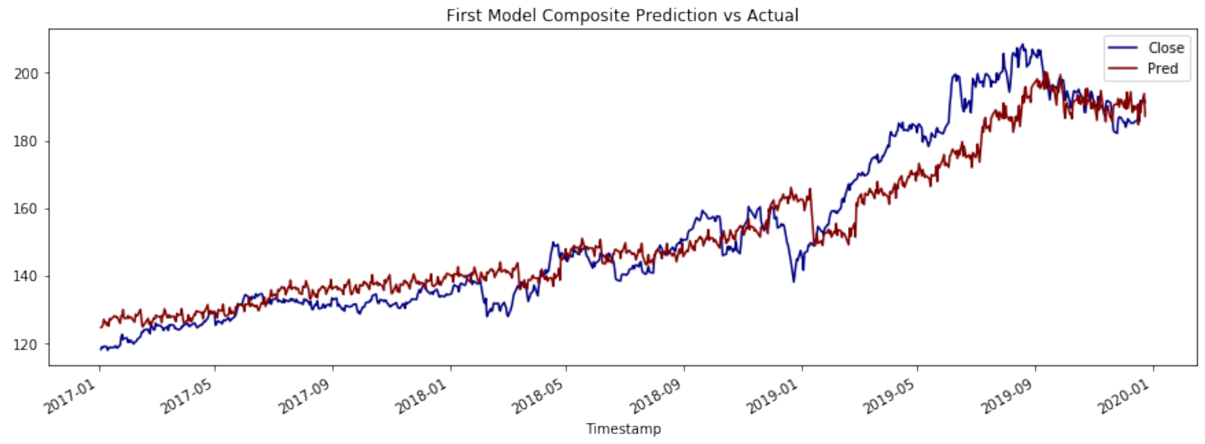


Figure 21 – Multivariate model, first iteration composite prediction vs actual

Table 9 summarizes the performance of the composite line in Figure 21.

Table 9 – Multivariate first iteration composite line errors

Mean Absolute Error	6.12
Mean Squared Error	63.07
Pearson Correlation	0.96

Figure 22 shows the errors versus the prediction horizon for the composite line.

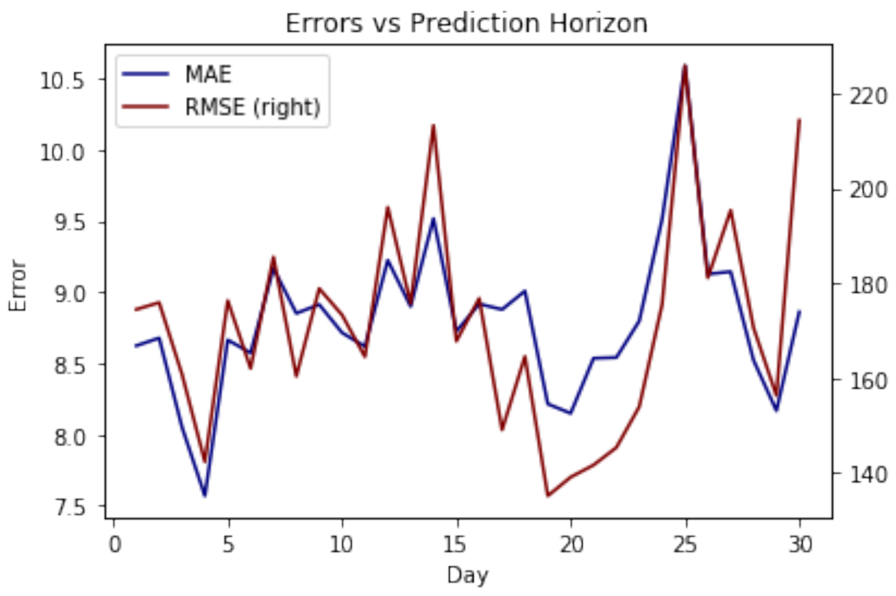


Figure 22 - Multivariate, first iteration errors by prediction horizon

Last Iteration (Most Training Data)

Figure 23 shows the composite prediction versus the actual closing price for the first model.

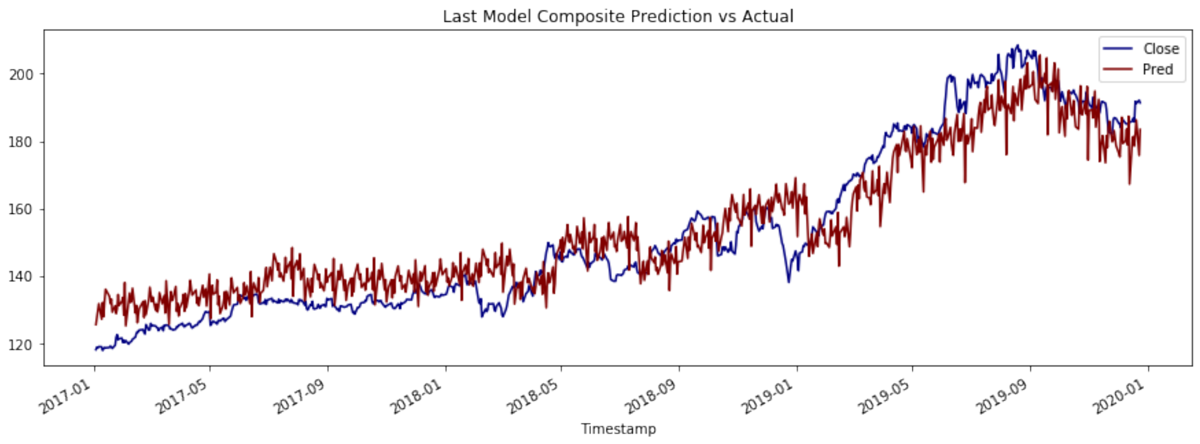


Figure 23 – Multivariate model, last iteration composite prediction vs actual

Table 10 summarizes the performance of the composite line in Figure 21.

Table 10 - Multivariate model last iteration composite line errors

Mean Absolute Error	7.29
Mean Squared Error	76.64
Pearson Correlation	0.95

Figure 24 shows the errors versus the prediction horizon for the composite line.

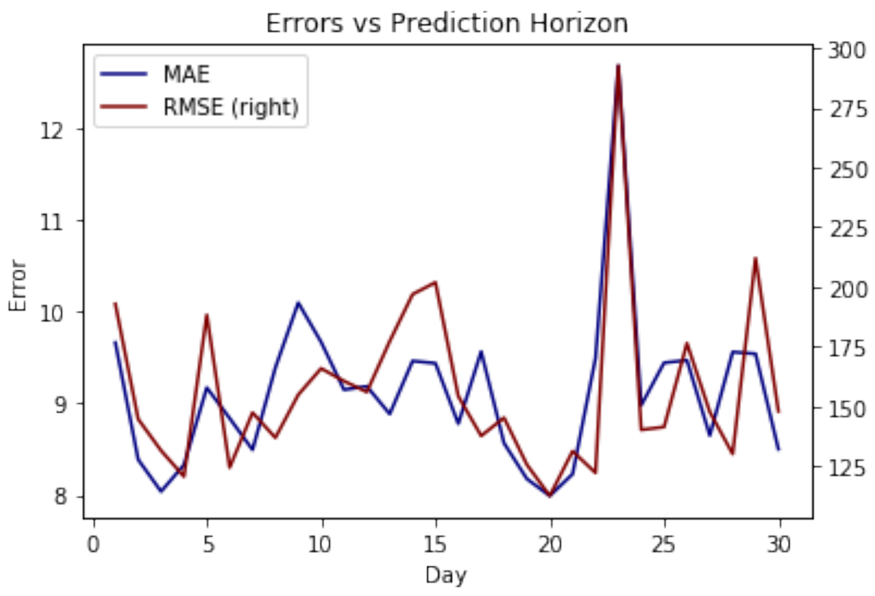


Figure 24 - Multivariate, last iteration errors by prediction horizon

Summary

- Overall, the multivariate model performs better than the univariate model. This is seen in the MAE, RMSE and Pearson scores. It is also evident it the standard deviations which are much lower. The maximum errors are also much lower than the univariate model.
- Unexpectedly, the first iteration performs better than the last iteration for this model. This may be due to overfitting on the test set.
- Unexpectedly, I also observe no clear increasing trend in error with increasing prediction horizon.

Justification

In order to make a direct comparison with the benchmark model we need to convert the MAE to Mean Absolute Percentage Error (MAPE). We can do this under worst-case conditions of $120 which is the price at the start of the testing set.

Table 11 - Comparison of Baseline, Univariate, Multivariate and Benchmark models

Model	MAE	RMSE	PCC	MAPE for ECL $120	MAPE for ECL $200
Baseline ARIMA	3.86	28.51	0.97	0.03	0.02
Univariate, multistep	8.60	109.57	0.94	0.07	0.04

Multivariate, multistep	6.12	6307	0.96	0.05	0.03
Benchmark Model	-	-	0.95	0.01	0.01

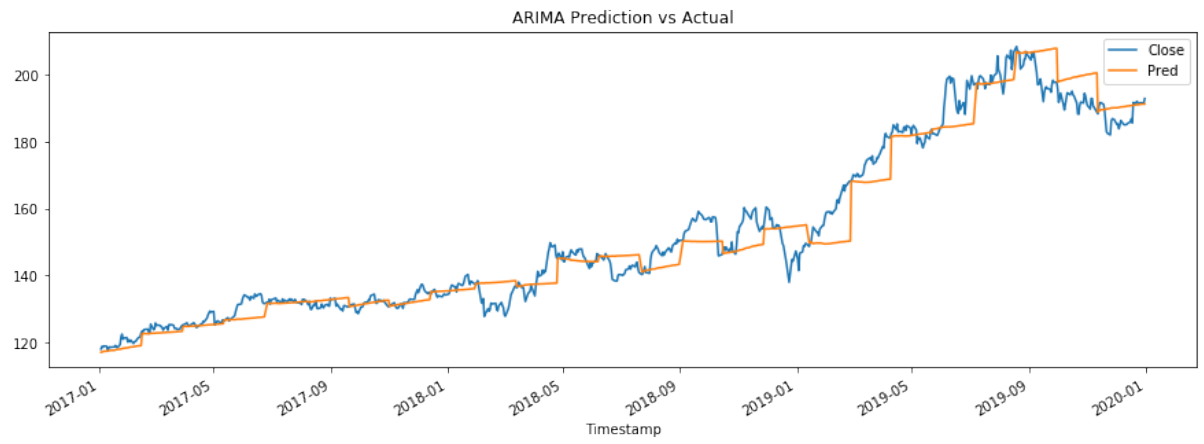
The results are weaker than the benchmark, but this isn't surprising as the tasks are different. The benchmark score is on a next step prediction whereas I'm predicting 30 future timesteps. This opens the model up to much larger errors. I'm pleased the results are in the same magnitude as the benchmark study. This suggest the results have some robustness. The excellent performance of the benchmark ARIMA model is surprising. The benchmark model has been able to outperform both deep learning models implemented.

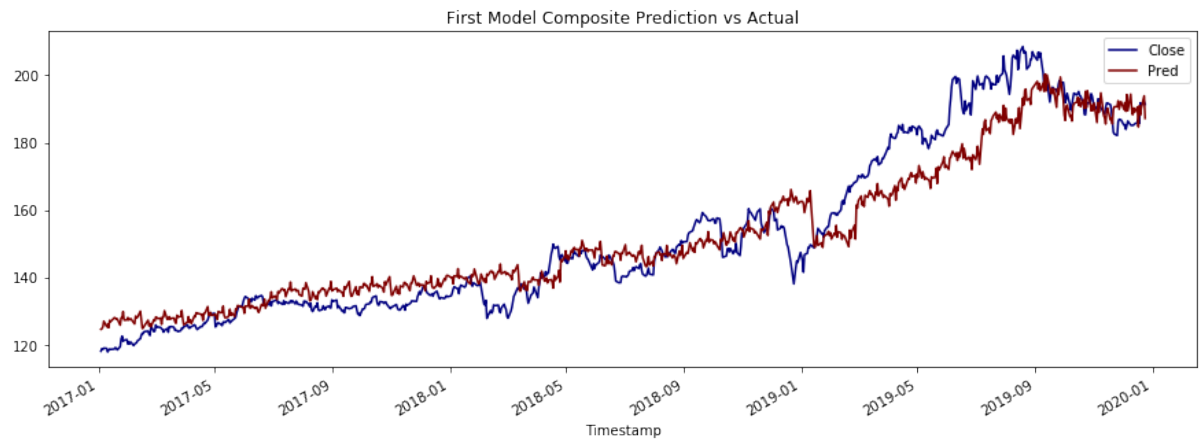
If I were a trader, I think I'd stick with a simple ARIMA model until I could at least match the performance with a more complex deep learning solution.

V. Conclusion

Free-Form Visualization

The best way I can summarize this project is the following two charts. They represent the Baseline model's performance on the task and the deep learning model's performance on the same task.





I see a number of interesting things when I directly compare these charts:

- The baseline model clearly relies on a near linear relationship with previous steps. The deep learning model is able to develop a more complex relationship between past measurements. This is evident in comparing any one of the 30 timestep prediction. I had expected this behaviour, but it is great to see this borne out in the implementation.
- Both models struggle to react to or generalize a behaviour for the steadily rising price in 2019. A rise at this rate is not present in the training set and I'm surprised that the LSTM is unable to infer this from one of the technical indicators.
- Both models are making both positive and negative errors.

Reflection

The end to end solution implemented in this project is as follows:

- Rearrange the time series dataset into a supervised learning dataset
- Split of the dataset into features, X, and targets, y.
    - If scaling of the dataset is required, it is applied at this step.
- Split the dataset further into Test and Training sets.
- The dataset is further reshaped into a 3D structure (samples, timesteps, features) which is expected by the LSTM layers.
- Specify a deep learning model.
- Train the model using the training dataset until the training reaches an acceptable loss value (or other stopping criteria).
- Validating the performance of the model using the Testing set.
- Use the trained and validated model on the "out of model' samples.
- Periodically retrain the model by adding any new measurements to the training set.

I have found three things challenging in arriving at this solution:

1. Understanding how and why we need to transform the time series dataset into a supervised learning problem.
2. How and why we need to reshape the data into the 3D arrays for the training the LSTM network.
3. Coding the loop to walk the model forward over the training set and collect the data it generates as it moves.

In saying these were challenging, I've also found them immensely interesting. Whilst I think I would still struggle to explain the mathematics that happen within an LSTM cell, I'm very pleased to have been able to implement them successfully for the problem I framed.

Overall, the model performs the task that I set out to achieve but not to a performance standard I find satisfying. I was hoping to be able to do better than the benchmark model.

Improvement

I can think of many ways to improve this work. They include:

- Changing the problem framing. For eg. predict the next 7 days using the previous 30 days.
- Implement the same architecture on a different time horizon (hours instead of days). This may have produced a very different result.
- With more time, I'd love to have implemented a PCA on the supervised dataset. I believe I'd then have seen more explanatory power of the technical indicators.
- As with almost every deep learning project, there is always room for further hyperparameter and model architecture optimization.
- Conducting the analysis on a number of additional of different stocks.

- Exploring other LSTM cell types such as CNN-LSTMs, bidirectional LSTMs and CONV-LSTMs.

VI. Bibliography

Anon., 2020. *Talos.* [Online]
Available at: https://github.com/autonomio/talos

Brownlee, J., 2017. *A Gentle Introduction to the Box-Jenkins Method for Time Series Forecasting.* [Online]
Available at: https://machinelearningmastery.com/gentle-introduction-box-jenkins-method-time-series-forecasting/

Brownlee, J., 2018. *How to Develop Multivariate Multi-Step Time Series Forecasting Models for Air Pollution.* [Online]
Available at: https://machinelearningmastery.com/how-to-develop-machine-learning-models-for-multivariate-multi-step-air-pollution-time-series-forecasting/

Brownlee, J., 2020. *Deep Learning for Time Series Forecasting.* s.l.:s.n.

Brownlee, J., 2020. *Long Short-Term Memory Networks with Python.* s.l.:s.n.

Federal Reserve Bank of St. Louis, 2020. *Wilshire 5000 Total Market Full Cap Index.* [Online]
Available at: https://fred.stlouisfed.org/series/WILL5000INDFC

Geron, A., 2019. *Hands-On Machine Learning with Scikit-Learn & Tensorflow.* s.l.:s.n.

Johnson, K., 2019. *Pandas Technical Analysis Libray.* [Online]
Available at: https://pypi.org/project/pandas-ta/

Krolner, B., Vanstone, B. & Finnie, G., 2010. *Financial Time Series Forecasting with Machine Learning Techniques: A Survey.* Bruges, Belgium, ESANN, pp. 25-30.

Liu, J., Chao, F., Lin, Y. & Lin, C., 2015. Stock Prices Prediction using Deep Learning Models.

McKinney, W., 2011. pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing.*

Padial, D. L., 2018. *Technical Analysis Library in Python.* [Online]
Available at: https://technical-analysis-library-in-python.readthedocs.io/en/latest/

Pedregosa, F., 2011. Scikit-learn: Machine Learning in Python. *JMLR 12,* pp. 2825-2830.

Sheftel, R., 2019. *Pandas Market Calendars.* [Online]
Available at: https://pypi.org/project/pandas-market-calendars/

Spyros Makridakis, E. S. V. A., 2018. Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLoS ONE.*

Wikipedia, 2020. *History of aritifical intelligence.* [Online]
Available at:

https://en.wikipedia.org/wiki/History_of_artificial_intelligence#Deep_learning,_big_data_and_artificial_general_intelligence:_2011%E2%80%93present

Wikipedia, 2020. *Moore's law.* [Online]
Available at: https://en.wikipedia.org/wiki/Moore%27s_law