Forecasting Stock Lows with Deep Learning

Aaron Wan
Department of Computer Science
Stanford University
aaronwan@stanford.edu
Project Code: https://github.com/aaronlwan/deep-learning-stock-lows

Abstract

Forecasting stock prices plays an important role in our economy. Studies have shown that deep learning models have the potential to generate precise stock price predictions. However, the vast majority of these studies have focused on forecasting a stock's closing price. Since stock prices typically reverse at key price levels, forecasting a stock's low price may be an easier task. In this project, I applied three types of deep learning models–two RNNs (single-layer and two-layer), a CNN, and a CNN-RNN hybrid–to forecast the lows of the SPY ETF using price and volume data from previous time-steps. I also experimented with several input and prediction time-frames (e.g. using previous 15 days to predict the following day vs using previous 5 days to predict the following day) to determine the optimal prediction time-frame for precise price predictions. The best performing model was the CNN-RNN hybrid which, when applied to the test data, predicted the low price of the SPY ETF each hour with a relative error of 0.137%. The size of the input (e.g. 15 days vs 10 days) had little impact on model performance. Input and prediction time-frames where the input and output lie in the same interval produced better results. The RNNs were the poorest performing models, but the results of the RNNs in predicting SPY's low prices still outperformed the results produced in comparable research aiming to forecast SPY's closing prices, indicating that low price prediction may be an easier task for deep learning models. Future research aiming to forecast stock prices with deep learning should consider predicting low price as opposed to the close price.

1 Introduction

The ability to predict stock prices plays an important role in our economy. Due to the prevalence of algorithmic trading, changes in stock prices are now largely driven by decisions made by these algorithms. Thus, it is possible to develop neural networks to predict stock prices because these algorithms act based on patterns derived from previous price action. Research has shown that deep learning models have produced promising results when applied to forecast stock prices [3].

Yet, the majority of applications of deep learning for stock price forecasts have focused on predicting the closing price of a stock. There is a lack of research applying deep learning models to forecast a stock's low price. Importantly, stock prices typically reverse at key price levels identified by traders and algorithms, so predicting the low may be easier than predicting the close.

Thus, in this project, I aim to apply deep learning models to predict a stock's low price. The input variables will be the open price, close price, high price, low price, and volume data occurring at specified intervals over a specified window that precedes time-frame of our prediction. The output variable will be the lowest price a stock will reach over a specified window.

2 Related Work

Price Trends: Some papers, such as [7] focus on predicting the price trend of a stock (up or down). I have chosen to predict the low price as opposed to the trend of the low price because a price prediction is more informative, and additionally, in theory, a model trained to predict stock prices should also be able to capture price trends.

Basic Models: [3] experimented with MLPs, RNNs, LSTMs, and CNNs to predict the S&P 500's daily closing price using the previous 14 day's price and volume data. They found that a single-layer RNN produced the lowest MAE loss on test data with a loss of 0.0148. Since this is the model that produced the best results, I will use a single-layer RNN as a baseline model to determine whether forecasting the low price is an easier task than forecasting the close.

Similarly, [4] investigated MLPs, RNNs, LSTMs, and CNNs but instead applied them to forecast the daily closing prices of five different companies on the National Stock Exchange and the New York Stock exchange. They found a CNN to have the lowest MAPE between the prediction and actual prices. Since I cannot access stock data on the specific companies they experimented with, I cannot compare my results to their results to draw any conclusions about predicting the low price. However, I will experiment with a CNN in this project since their paper demonstrated promising results with the CNN.

Complex Models: Many researchers have developed more complex models for stock price forecasts. In this project, I took inspiration from the CNN-LSTM hybrid model proposed by [6] for closing price forecast, as this hybrid model was able to produce more robust results than the basic models.

Input Features: Researchers have experimented with various input features, such as [8] who incorporated stock indicators such as Moving Averages for stock price prediction or [5] who incorporated news sentiment analysis. This project will focus on using price and volume data the input features because acquiring these features is less time-intensive.

3 Dataset and Features

3.1 Dataset Description

I experimented with two types of data for price prediction: daily data and intraday data. I used the yfinance API to acquire stock data on the daily interval ranging from Jan 2002 to Apr 2022, and I used FirstRate Data to acquire data on intraday intervals (5min and 1hr) from Jan 2005 to Apr 2022. I focused on the SPY ETF because 1) it is an ETF representing 500 large-cap stocks, so it provides a good representation of the entire stock market, 2) it was the ETF I was able to acquire the largest and most complete dataset for, and 3) it allows us to compare our results to [3] since they also focused on forecasting the S&P 500.

3.2 Preprocessing

3.2.1 Standardizing Data

I tried three standardization methods: min-max scaling the price and volume data between 0 and 1 using the min and max within each sample window, min-max scaling the price and volume data between 0 and 1 using the min and max of the entire training window, and standardizing the price and volume data to have a mean of 0 and a standard deviation of 1 using the mean and standard deviation of the training window. Following some testing, min-max scaling the price and volume data between 0 and 1 over the entire training window led to the best results, so this was the standardization method I used to preprocess the data.

3.2.2 Input and Prediction Time-frame

A key challenge with forecasting stock prices is that the optimal time-frame for both the model input and prediction are unknown. In this project, I experimented with the following sets of time-frames displayed in the table below. To interpret the table, "Hour to Day", "36" means that price and volume data from each of the previous 36 hours were used to predict the low price of the following day.

Day to Day	Hour to Day	Hour to Hour	5 Min to Hour
1	12	5	12
10	24	10	24
15	36	15	36

The day to day input and prediction time-frame contained 7,300 samples. The hour to day time-frame contained 4,500 samples. The hour to hour and 5 min to hour time-frames contained 66,000 samples.

4 Methods

4.1 Deep Learning Models

I applied three types of deep learning models: two Recurrent Neural Networks, a Convolutional Neural Network, and a hybrid model consisting of a CNN and RNN. The models were implemented using Keras [2] and Tensorflow [1].

4.1.1 Single-Layer RNN

A single-layer RNN for many-to-one prediction was the first model I experimented with since [3] found a single-layer RNN to be most effective for closing price prediction. The input of the model was shaped as a t x 5 matrix. The number of rows, t, represents the number of time-steps, and 5 represents the number of features (open, high, low, close, volume). The RNN will process information across the time-steps of the input to inform the prediction. The architecture of the model is shown in section 7.1.1 in the Appendices.

4.1.2 Two-Layer RNN

A two-layer RNN for many-to-one prediction was used. With a deeper model, the network should be able to learn more information across the time-series and have lower prediction error. The architecture is in section 7.1.2 in the Appendices.

4.1.3 CNN

A model with two 2D convolutional layers and 3 fully connected layers was used. The input data was a t x 5 x 1 matrix, with an additional dimension being added in order to provide valid input to the 2D CNN layers. For each layer, a 3x3 kernel with stride of 1 and "valid" padding with 5 filters were used to process the relationships between features across the time-series as well as within each time-step. The model architecture is in section 7.1.3 in the Appendices.

4.1.4 CNN-RNN Hybrid

Taking inspiration from the CNN-LSTM hybrid model implemented by [6], a hybrid model involving a CNN (three 2D convolutional layers) and a RNN (two recurrent layers) was used. I chose to implement a RNN as opposed to a LSTM due to the results of [3] indicating that a RNN performs better at stock price forecasts than a LSTM. The input data was a t x 5 x 1 matrix. All three convolutional layers used a 3 x 2 kernel with "same" padding. The first, second, and third layers used 3, 2, and 1 filters, respectively. The output of the convolutional layers was reshaped into a t x 5 matrix for input into the proceeding two RNN layers. The idea behind this model is that the convolutional layers are used to process the patterns in the input price and volume data, and the recurrent layers are used to process information from the resulting time-series output of convolutional layers. The model architecture is in section 7.1.4 in the Appendices.

4.2 Evaluation Metric

To evaluate the model performance, the output of the model representing the standardized low price was converted to the unstandardized price prediction, and then the relative error between the unstandardized prediction and actual low price was computed.

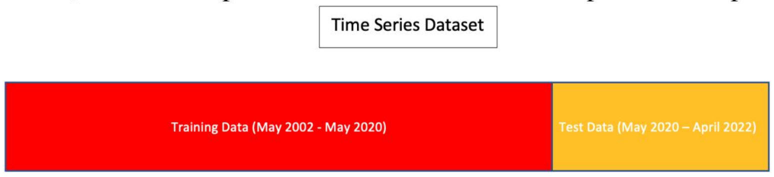
4.3 Loss Functions

I experimented with three loss functions: Mean Absolute Error, Mean Squared Error, and Mean Relative Error. After some initial experimentation, based on the evaluation metric, MRE lead to the worst results, and MAE and MSE led to the similar results. I decided to stick with MAE because it is easier to interpret.

4.4 Training

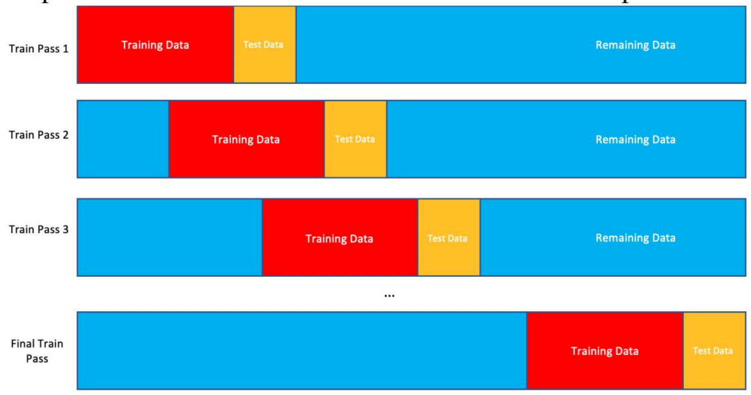
4.4.1 Fixed Training Window

Initially, I used a fixed training window for model training. The time-series data was split such that the training data precedes the test data. After some initial tests, a train/test split of 80/20 was deemed most optimal. A depiction of such a split is shown below.


Time Series Dataset
Training Data (May 2002 - May 2020)
Test Data (May 2020 – April 2022)

4.4.2 Rolling Training Window

With the fixed training window strategy, the models suffered from high variance. In hopes of reducing the variance, I applied a rolling training window strategy, training the model by continuously rolling a training window across the time-series data, each time testing on a new period of unseen data. Since the broad trend of stock prices is that they increase over time, a rolling training window makes the model less likely to overfit to the lower prices in the training data. After testing out various sliding window sizes and step sizes, a sliding window that creates 5 total training passes and that steps halfway between the overall range of the previous window produced the best results. A train/test split of 80/20 was also found to work the best. A depiction of this training procedure is show below.



5 Results/Discussion

5.1 Additional Hyperparameters

Following some experimentation, a learning rate of 0.001 and a batch size of 32 was determined to be optimal for model performance. The models were trained with early stopping with a patience of 15 epochs. Additionally, an Adam optimizer with the default Keras parameters produced the best model performance.

5.2 Model Loss Results

5.2.1 Fixed Training Window

For the fixed training window, all the models were trained on the the day-to-day time-frame However, due to the relatively poorer performance of the RNNs, they were not trained on the remaining prediction and input time-frames. The losses of the models on the various prediction and input time-frames are shown below.

Day to Day Prediction

Input Size	Single-Layer RNN		Two-Layer RNN		CNN		CNN-RNN	
	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss
15	0.006	0.011	0.00491	0.0102	0.0028	0.0069	0.0027	0.0068
10	0.0059	0.0109	0.00489	0.01	0.0029	0.007	0.0028	0.0067
5	0.0054	0.0103	0.00447	0.00993	0.0028	0.0069	0.0027	0.0067

Hour to Day Prediction

Input Size	CNN		CNN-RNN	
	Train Loss	Test Loss	Train Loss	Test Loss
36	0.009	0.0134	0.0088	0.0132
24	0.0082	0.0122	0.0081	0.0121
12	0.0081	0.012	0.0081	0.0119

Hour to Hour Prediction

Input Size	CNN		CNN-RNN	
	Train Loss	Test Loss	Train Loss	Test Loss
15	0.0011	0.0019	0.00089	0.0019
10	0.001	0.0019	0.00088	0.0019
5	0.001	0.0019	0.00087	0.0019

5-min to Hour Prediction

Input Size	CNN		CNN-RNN	
	Train Loss	Test Loss	Train Loss	Test Loss
36	0.0021	0.0031	0.0019	0.003
24	0.0019	0.003	0.0019	0.0029
12	0.0019	0.003	0.0018	0.003

As seen in the tables, all the models suffered from high variance. The single-layer RNN produced the highest losses. The two-layer RNN outperformed the single-layer RNN, but still produced worse results than the CNN and CNN-RNN. The CNN and CNN-RNN produced similar losses for all time-frames except hour to hour, where they had similar test losses, but the CNN-RNN had higher variance.

The tables also illustrate that prediction and input time-frames where the input and output time-series variables lie in the same interval (day to day and hour to hour) produced better results than the time-frames where the input and output variables were on different intervals (hour to day and 5min to hour).

The size of the input was only relevant for the single-layer RNN, for the two-layer RNN, and when applying the CNN and CNN-RNN for hour to day prediction. For these scenarios, decreasing the size of the input led to a lower loss.

The losses of the models when the output variable was on the hourly interval were lower than the losses of the models when the output variable was on the daily interval. This difference makes sense because a stock's price will have a larger range on larger intervals.

5.2.2 Rolling Training Window

When using the rolling training window strategy, I focused on training the CNN-RNN because it produced the lowest overall losses on the training and test data, and it had a higher variance than the CNN. I used the hour to hour time-frame because this was the time-frame that resulted in the lowest losses. The losses of the model on the final window, as well as the second and third to last windows as validation are displayed below.

Hour to Hour Prediction

Input Interval	CNN-RNN					
	Window 3		Window 4		Window 5	
	Train Loss	Test Loss	Train Loss	Test. Loss	Train Loss	Test Loss
15	0.0021	0.0031	0.0023	0.003	0.0022	0.0029
10	0.0022	0.003	0.0022	0.0031	0.0021	0.0028
5	0.0023	0.0032	0.0021	0.0029	0.0021	0.0029

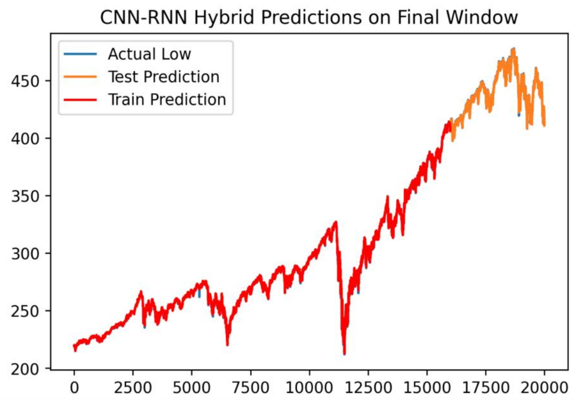
The rolling training window significantly reduced the variance of the CNN-RNN. The model's losses on the third and fourth window validated the ultimate losses on the final window. The size of the input had little impact on the losses. The losses here are larger than the CNN-RNN's losses on the hour to hour time-frame with the fixed training window. However, size of the rolling training window is smaller than the size of the fixed training window, so the scale by which the prices are normalized is different. In order to compare the results, we must look to the evaluation results in the following section

5.2.3 Model Evaluation Results

To evaluate the models, the models' predictions on the test data were converted to the unstandardized low price prediction, and then the relative error between the unstandardized predicted low price and the actual low price was computed. The evaluation results for the applying the models on the day to day and hour to hour time-frames are shown below, since these were the time-frames with the relatively stronger results from computing the losses. For the rolling training window, the evaluation results on the test set for the final window are displayed.

Relative Error Between Prediction and Actual Prices							
Input Size	Day to Day				Hour to Hour (Fixed Training Window)		Hour to Hour (Sliding Training Window)
	Single-Layer RNN	Two-Layer RNN	CNN	CNN-RNN	CNN	CNN-RNN	CNN-RNN
15	1.020%	1.0001%	0.684%	0.683%	0.190%	0.19%	0.138%
10	1.001%	0.997%	0.683%	0.685%	0.200%	0.19%	0.137%
5	0.998%	0.991%	0.684%	0.683%	0.190%	0.18%	0.138%

The evaluation results confirm that applying the sliding window training strategy improved the performance of the CNN-RNN on the hour to hour time-frame. A visual comparison between the CNN-RNN's unstandardized predictions and the actual prices is shown below. On y-axis is the low price, and the x-axis represents the each hour from the start of the window.



Visually, the CNN-RNN appears to capture the price trends very well, as the plot of the actual low prices is almost entirely covered by the model's predictions. Even in the places where the model does not generate a precise prediction, the model's predictions still appear to capture the price trends.

6 Conclusion/Future Work

Using the open price, high price, low price, close price, and volume from each of the previous 10 hours, the CNN-RNN hybrid model was able to predict the low price of the SPY ETF in the following hour with a relative error of 0.137%. This was the lowest relative error achieved across all the models on any of the time-frames tested. Using a rolling training window as opposed to a fixed training window significantly reduced the variance of the CNN-RNN and led to a better performing model.

Under the fixed training window, the CNN produced similar results to the CNN-RNN, but the CNN-RNN produced relatively lower training error than the CNN when applied to hour to hour prediction. Since the goal of the sliding window training strategy was to reduce variance, and the CNN-RNN had a higher variance, I focused on the CNN-RNN for the sliding window training procedure.

The single-layer RNN and two-layer RNN were the poorest performing models. This may be due to these two models being relatively shallower models than the CNN and CNN-RNN. However, the results produced by these two models still outperformed the results of comparable models in other studies aiming to forecast stock closing prices. For instance, [3] found a MAE of 0.0148 with a single-layer RNN predicting SPY's daily closing prices with a fixed training window and an input size of 14 days, while our single-layer RNN produced a MAE of 0.011 predicting SPY's daily low prices with an input size of 15 days. This indicates that forecasting a stock's low prices may in fact be an easier task than forecasting a stock's close prices. Future research on forecasting stock prices with deep learning models should consider using the low price of a stock as opposed to the closing price.

The size of the input (e.g. 15 days vs 10 days) had little impact on model performance, especially for the CNN and CNN-RNN. Moreover, the results indicate that using input and prediction time-frames where the input and output variables lie in the same interval (day to day and hour to hour) lead to better model performance. The models produced lower relative error on hour to hour time-frame than the day to day time-frame, but this result is expected because the SPY has a larger range of movement each day as opposed to each hour.

In the future, I would like to explore the impact of additional sample features as well as explore other hybrid models, since the CNN-RNN was the best performer in this project. Additionally, I would like to experiment with other intraday input and prediction time-frames, such as 5-min to 5-min or 30-min to 30-min.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. ArXiv.org. https://arxiv.org/abs/1603.04467

[2] François, C. (2015). Keras. GitHub , GitHub Repository. https://doi.org/https://github.com/fchollet/keras

[3] Kamalov, F., Smail, L., Gurrib, I. (2020). Stock price forecast with deep learning. 2020 International Conference on Decision Aid Sciences and Application (DASA). https://doi.org/10.1109/dasa51403.2020.9317260

[4] M, H., E.A., G., Menon, V. K., K.P., S. (2018). NSE Stock Market Prediction Using Deep-Learning Models. Procedia Computer Science, 132, 1351–1362. https://doi.org/10.1016/j.procs.2018.05.050

[5] Mohan, S., Mullapudi, S., Sammeta, S., Vijayvergia, P., Anastasiu, D. C. (2019, April 1). Stock Price Prediction Using News Sentiment Analysis. IEEE Xplore. https://doi.org/10.1109/BigDataService.2019.00035

[6] Rezaei, H., Faaljou, H., Mansourfar, G. (2021). Stock price prediction using deep learning and frequency decomposition. Expert Systems with Applications, 169, 114332. https://doi.org/10.1016/j.eswa.2020.114332

[7] Shen, J., Shafiq, M. O. (2020). Short-term stock market price trend prediction using a comprehensive deep learning system. Journal of Big Data, 7(1). https://doi.org/10.1186/s40537-020-00333-6

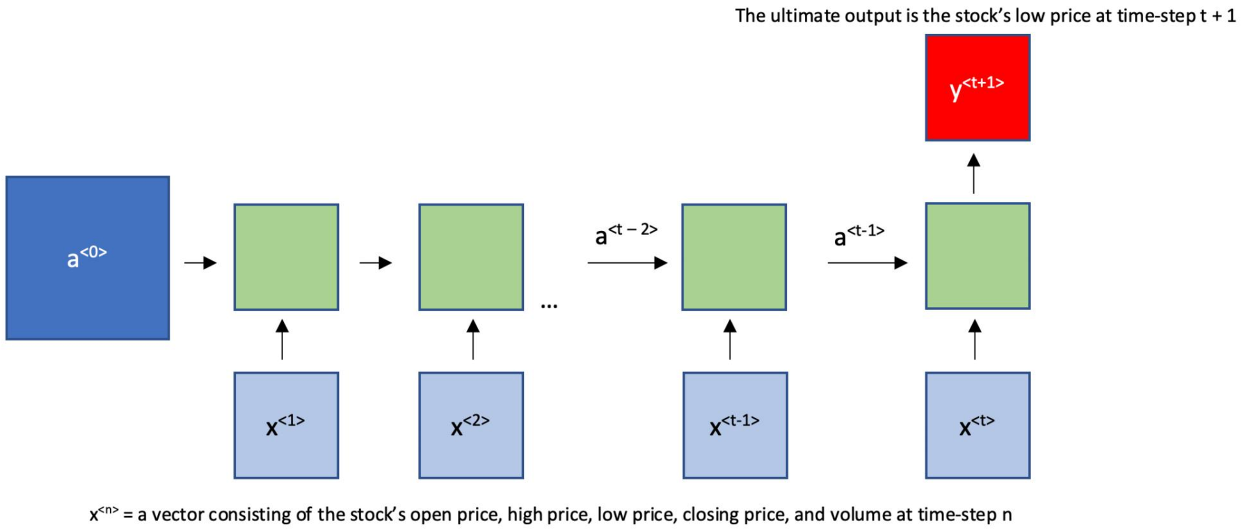
[8] Staffini, A. (2022). Stock Price Forecasting by a Deep Convolutional Generative Adversarial Network. Frontiers in Artificial Intelligence, 5. https://doi.org/10.3389/frai.2022.837596

7 Appendices

7.1 Deep Learning Models

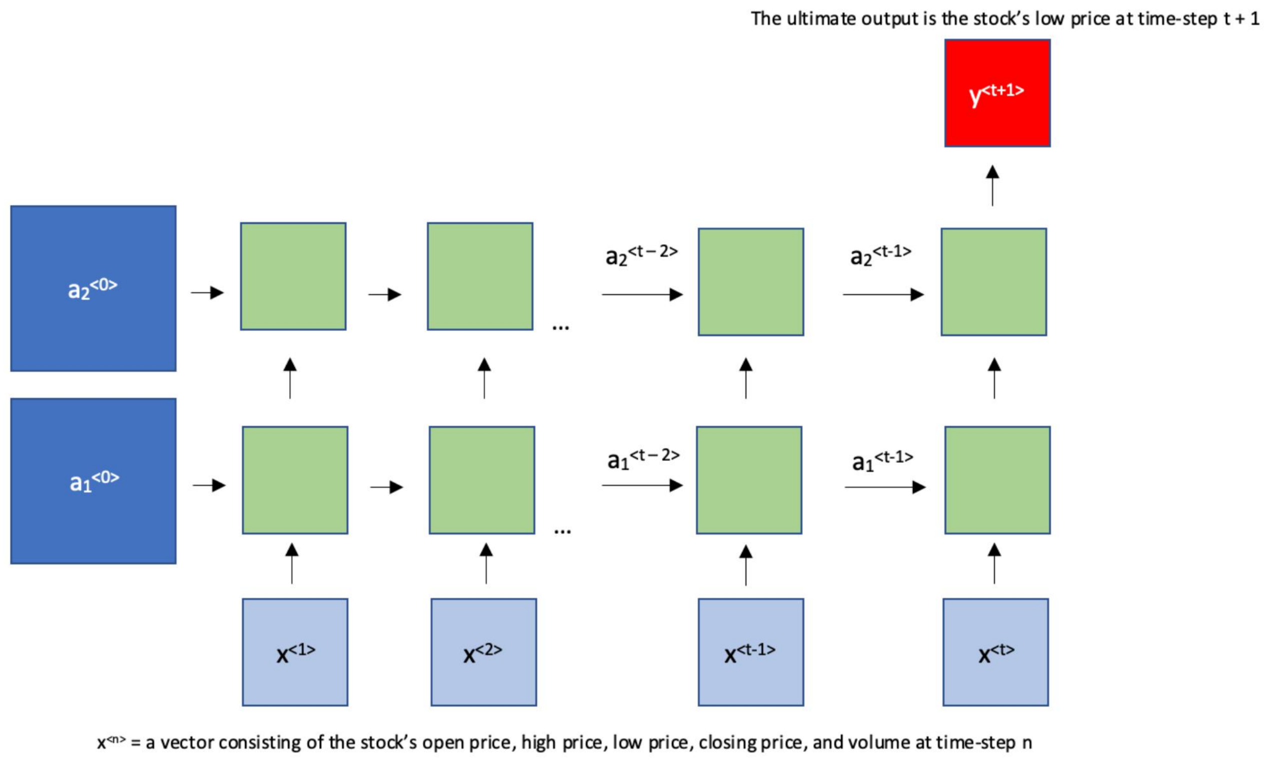
7.1.1 Single-Layer RNN

A ReLU activation function was applied before the final output.


The ultimate output is the stock's low price at time-step t + 1
$y^{<t+1>}$
$a^{<0>}$
$a^{<t-2>}$
$a^{<t-1>}$
...
$x^{<1>}$
$x^{<2>}$
$x^{<t-1>}$
$x^{<t>}$
$x^{<n>}$ = a vector consisting of the stock's open price, high price, low price, closing price, and volume at time-step n

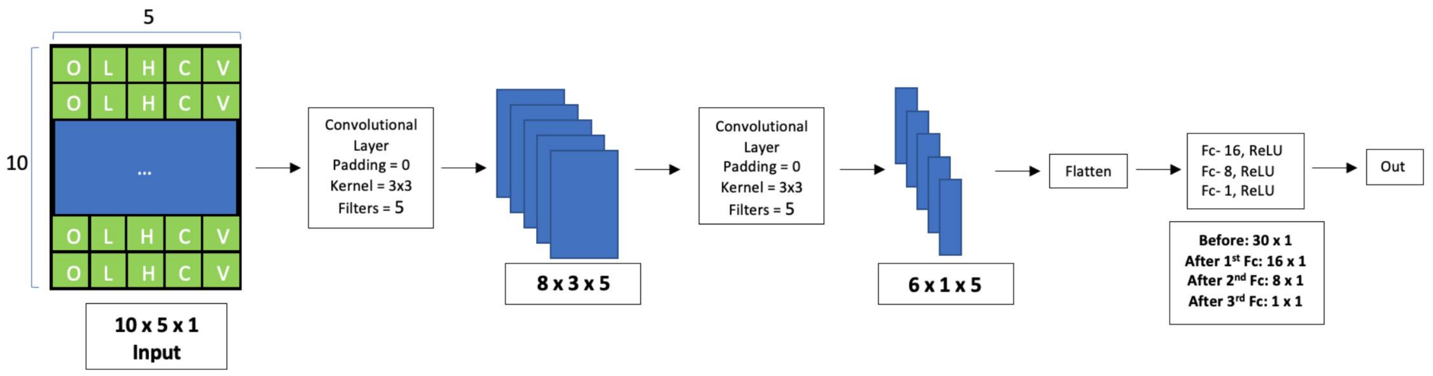
7.1.2 Two-Layer RNN

A ReLU activation function was applied before the outputs of both layers.


The ultimate output is the stock's low price at time-step t + 1
y<t+1>
a2<0>
a2<t – 2>
a2<t-1>
a1<0>
a1<t – 2>
a1<t-1>
...
x<1>
x<2>
x<t-1>
x<t>
x<n> = a vector consisting of the stock's open price, high price, low price, closing price, and volume at time-step n

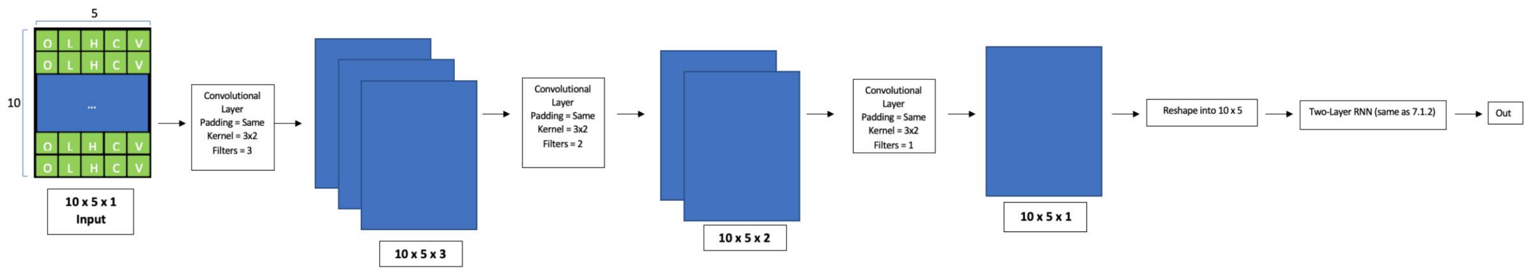
7.1.3 CNN

The diagram illustrates the CNN model with an input size of 10 x 5 x 1. For other input sizes, the layer sizes were modified slightly, but the overall number of layers and the type of layers remained consistent.



7.1.4 CNN-RNN Hybrid

The diagram illustrates the CNN-RNN hybrid model with an input size of 10 x 5 x 1. For other input sizes, the layer sizes were modified slightly, but the overall number of layers and the type of layers remained consistent.


5
10
O L H C V
...
10 x 5 x 1 Input
Convolutional Layer Padding = Same Kernel = 3x2 Filters = 3
10 x 5 x 3
Convolutional Layer Padding = Same Kernel = 3x2 Filters = 2
10 x 5 x 2
Convolutional Layer Padding = Same Kernel = 3x2 Filters = 1
10 x 5 x 1
Reshape into 10 x 5
Two-Layer RNN (same as 7.1.2)
Out