Deep Learning Models for Stock Price Prediction

Main Objective

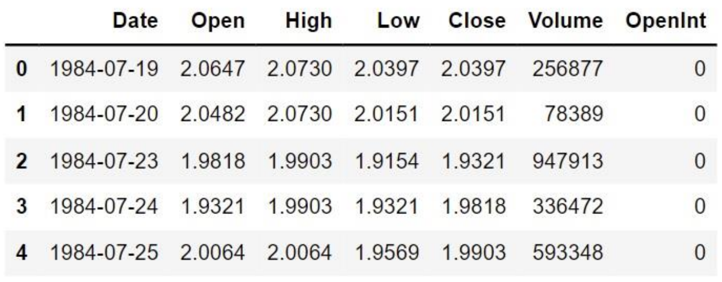
- In this report, we will analyze the dataset that has the stock prices of Analog Devices Inc (ADI). from 1984 to 2017.
- The objective of this report is to find out the best model that takes in as input a portion of the dataset and predicts the future behavior of the stock prices of ADI.
- We will use a variety of hyperparameter modifications of Recurrent Neural Networks. We will also use LSTM and GRU neural networks and find out the best model for this dataset.

Brief Description of the Dataset

- The dataset has 7 features. They are described below with their corresponding data types in brackets:
  1. Date (object): In yyyy-mm-dd format. The date goes from 1984-07-19 to 2017-11-10.
  2. Open (float64): Price of the stock in USD at the opening of the day.
  3. High (float4): High price of the stock in USD during the business day.
  4. Low (float64): Low price of the stock in USD during the business day.
  5. Close (float64): Price of the stock in USD at the closing of the day.
  6. Volume (int): Total number of shares of the stock that are traded on a day
  7. OpenInt (int): The number of open positions in the contract that have not yet been offset. All cells have the same value in this column, equal to 0.

- There are 8398 data points.

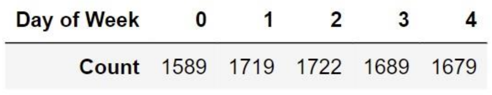
- Here is what a sample of the dataset looks like:



	Date	Open	High	Low	Close	Volume	OpenInt
0	1984-07-19	2.0647	2.0730	2.0397	2.0397	256877	0
1	1984-07-20	2.0482	2.0730	2.0151	2.0151	78389	0
2	1984-07-23	1.9818	1.9903	1.9154	1.9321	947913	0
3	1984-07-24	1.9321	1.9903	1.9321	1.9818	336472	0
4	1984-07-25	2.0064	2.0064	1.9569	1.9903	593348	0

Data Cleaning and Feature Engineering

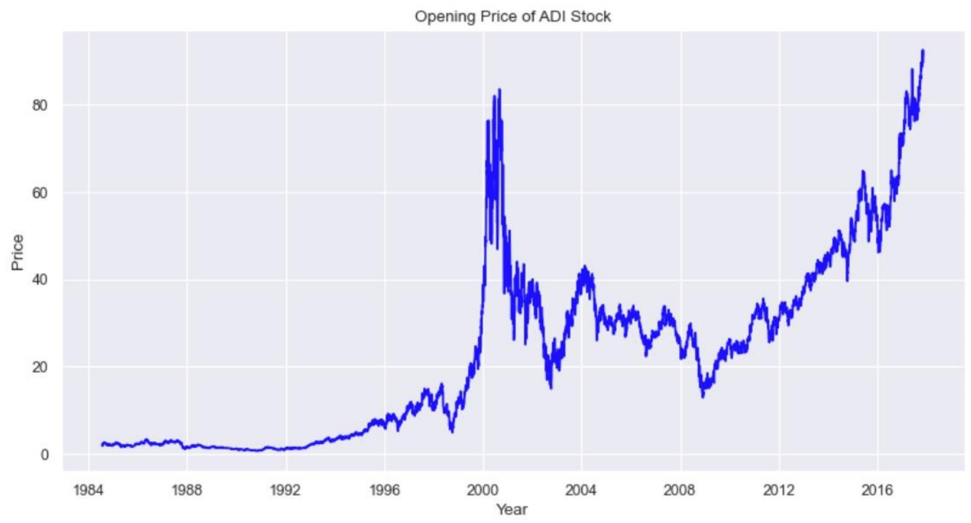
- The OpenInt column has only one value for every cell, equal to 0. This makes the column useless for any machine learning model and we consequently drop it.
- There are no missing values in any of the cells.
- We convert the Date column from object datatype to datetime[ns].
- We set this modified Date column as our index.
- Looking at the index, we find that it does not have a frequency.
- We use the dayofweek method for the index and find that the weekends are absent from our index, as expected. Moreover, some days during the regular week are also absent from our dataset as can be seen from the table below:



Day of Week	0	1	2	3	4
Count	1589	1719	1722	1689	1679

- To ensure we have a time series, we fill in the missing dates in our index. We do this by using the date_range method from the minimum date to the maximum date and setting this as our new index. There are now 12168 examples in our dataset.
- Since the newly introduced dates do not have any values for the features, we use the interpolate method to fill in these values.

- Now we look at the run sequence plots for the opening price of our stock.



- Instead of training and testing our models on the entirety of the dataset, we will only use a portion of the recent examples. Specifically, we'll try the last 1000 days and the last 500 days.
- In our deep learning models, we will choose one of the features in our dataset and train and test our model on this feature. We will try multiple features to decide which one is the most conducive to training our ML models.
- We then prepare a function that can convert any chosen series into the required form that keras can take as input for our various recurrent neural network models and their modifications.
- Subsequently, we split our dataset into a training and a testing split. We also need an input size since we are training on a time series. We set the training input, testing input, and testing output all equal to 15.

Deep Learning Models

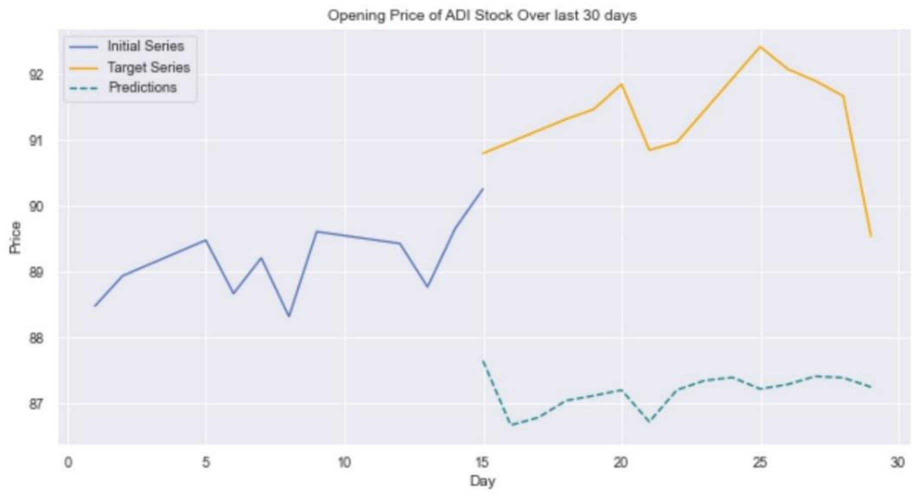
In this section, we will use various deep learning models to train and subsequently test our models on the opening values of the stock. For all our models, we will use the 'Open' column to train and test our models.

1. **Simple RNN (Base Case)**: We start with the most straightforward recurrent neural network. This network has two layers. The details of this model are given below.
    i) The first layer is the simple RNN. We use 10 cell units for this model.
    ii) The second layer is dense with an output of dimension 1 and the activation set to ReLU.
    iii) The model has 120 parameters for the simple RNN and 11 parameters for the dense layer. The total number of parameters is 131.

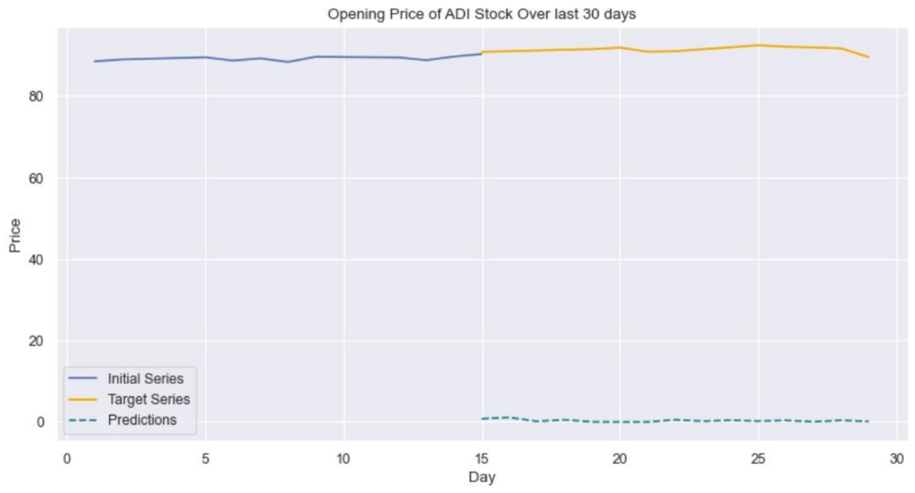
iv) We set the optimizer to Adam.
v) We use mean-squared error as the loss function.
vi) We set the batch size to 64 and train our model for 10 epochs.

Clearly, this model does a terrible job at predicting the future due to under-fitting. We now increase the number of cell units and the number of epochs for training our model

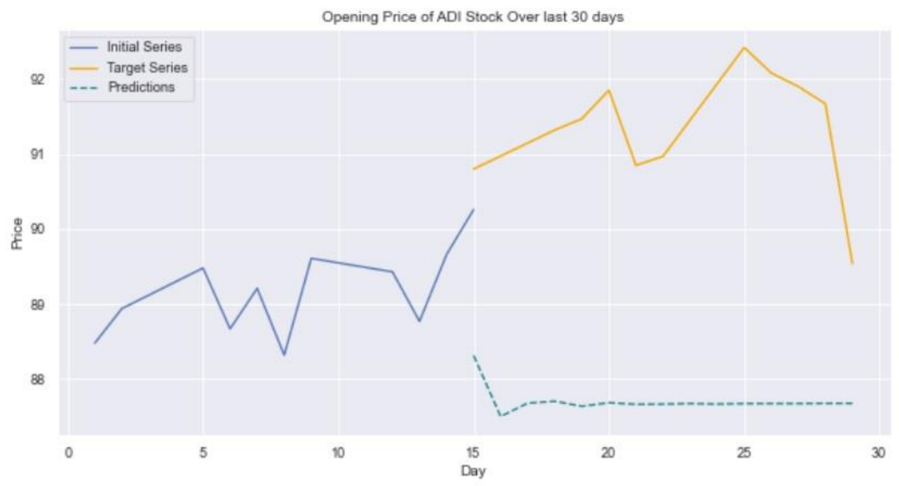
2. **Simple RNN - 2** : We now make small modifications to our previous model by increasing the cell units of the RNN to 200 from 10 and increase the number of training epochs to 10,000. This increases the total number of parameters to 40,400. We keep everything else the same as in the previous model to see if these changes will give us a better predictive model:



This model is much better than the previous one. While the model could still be significantly improved, it is much closer to the actual value and does manage to show small variations which resemble a Brownian motion. The RMSE for this model is 4.23.

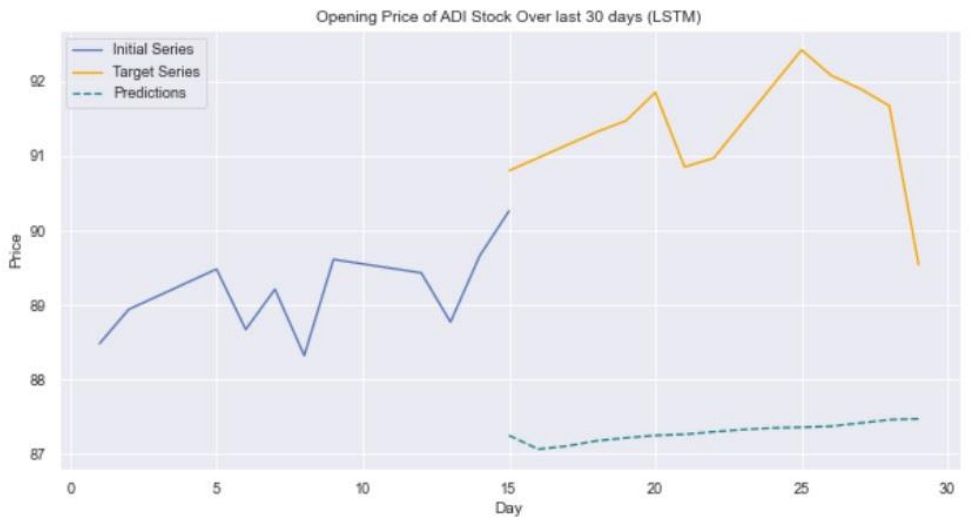


3. **Simple RNN - 3:** Since it appears that the previous model might be suffering from underfitting, we add more cell units to our models which increases the number of parameters. We increase the number of cell units to 250 from 200. This sets the number of parameters to 63,251. The RMSE for the model is 3.72. The predictive plot of the model has the form shown below.

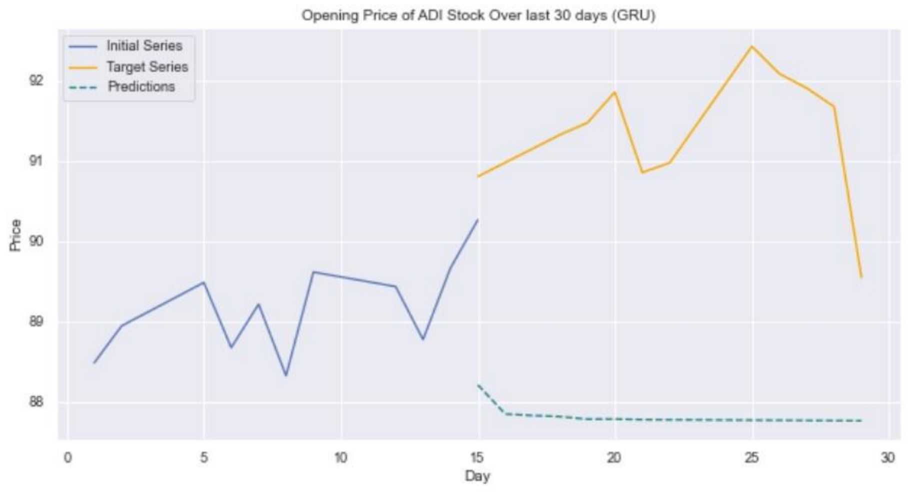


While the RMSE is lower than that for RNN-2, this model lacks the Brownian motion-like behavior that we expect from stock prices and in fact approaches a constant value. Thus, even though the RMSE is lower than the previous model, RNN-2 is preferable over this model.

4. **LSTM:** We now try an LSTM (Long-Short Term Memory) network. We set the number of cell units to 100 and cap the number of training epoch to 6,000 because LSTMs can take much longer to train compared to simple RNNs. The RMSE for the model is 4.12. The predictive plot on the training set for this model is shown below.

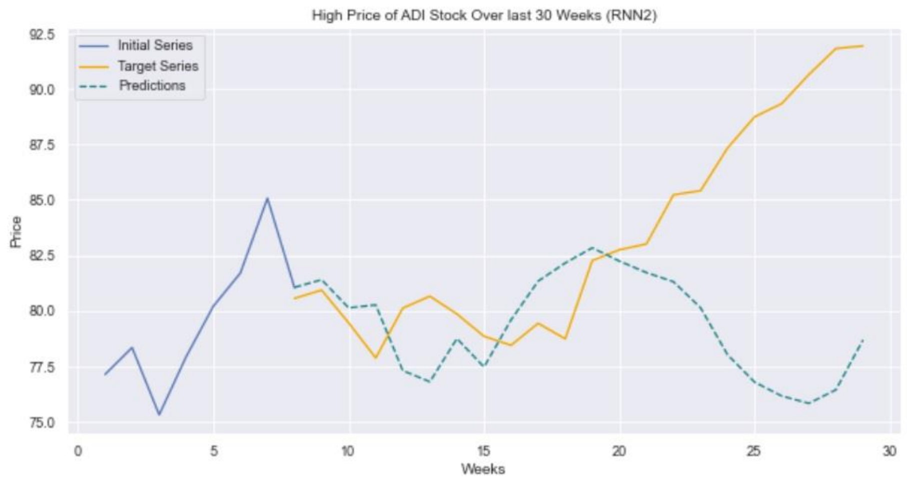


5. **GRU:** We now try a GRU (Gated Recurrent Unit) network. We set the number of cell units to 100 and the number of training epoch to 10,000. The RMSE for the model is 3.62. The predictive plot on the training set for this model is shown below.



6. **Simple RNN-2 (Weekly Sampled):** Now we have tried the simple RNN with a variety of cell units, LSTM and GRU. Based on these models we find that a simple RNN with 200 cell units and 10,000 training epochs is able to capture the fluctuations of stock price though it is not able to capture the trend and the error is a bit much for a good model. Therefore, we now try variations of the second model by changing the input size,

changing the feature over which we train or model and also by resampling the data at different frequencies.
We find that when we train the RNN-2 model (same number of cell units as RNN-2 but some hyperparameters are different which are explained) on weekly data, instead of training it on daily data, we get the best model. We use the last 200 weeks to train our model, use 8 weeks as training and testing input, and use 22 weeks for testing output. The weekly samples are taken by using the mean value of the stock price for that week.

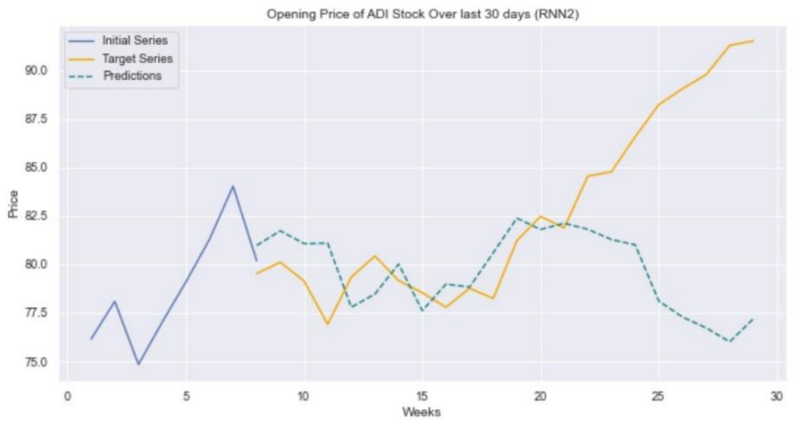


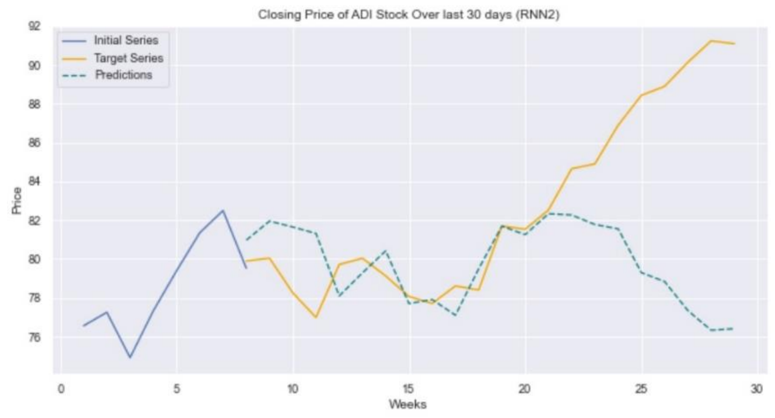
This model has a good predictive behavior for the first 12 weeks. However, after that the error increases significantly since it is plotted on a weekly scale and the fluctuations are larger than in the previous plots. The RMSE value for the first 12 weeks is only 2.03.

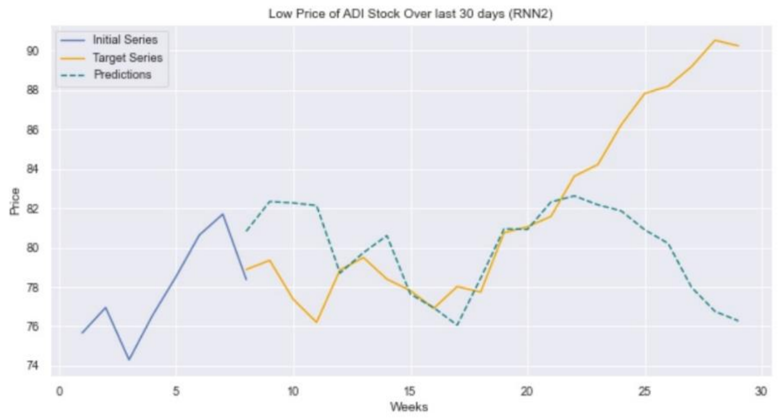
Best Model

The RMSE for models 2-5 are comparable and those small differences are not the best criterion for determining the model most suited to our needs. Instead, we choose the second model (RNN-2) as the best model among these four since it can capture the Brownian motion-like fluctuations. All the other models, while they have a marginally lower RMSE, become constant rather quickly. If we want to predict the stock behavior over a larger period of time, then this model is the best. However, the overall **best model is sixth one**, in which we used a modification of RNN-2 by weekly sampling the data. We got this model by trying out a large number of variations and we found that training the ‘High’ series with a weekly sample gave the best model.

The predictive plots for the ‘Open’, ‘Low’ and ‘Close’ columns based on the RNN-2 model are shown below:







The RMSEs for these three plots for the first 12 weeks of prediction are 1.87 (High), 1.90 (Low) and 2.60 (Closing).

We tried several other deep learning models (can be found in the attached Jupyter notebook) to figure out the best one, including changing other hyperparameters and also changing the sampling frequency of our time series. This particular model is able to capture the short-term trend rather nicely compared to the others.

Key Findings

Based on the deep learning models we tried (using different neural network structures and different hyperparameters), we found the best model and gained useful knowledge about other models as well.

- We found that our first simple RNN model has no real predicting power and is terribly underfit.
- Our second model has some predictive power and broadly shows the behavior of stock prices, although there is room for improvement.
- We tried to improve the performance of our second model by increasing the number of cell units to 250 from 200 but found no real changes. Thus, we need to significantly ramp up the number of cell units to correctly model the stock prices.
- We also tried LSTM and GRU. The RMSE values for these two models are similar to the previous ones but there doesn't appear to be any predictive power because they approach a constant value rather quickly.
- Trying a large number of variations of the second model, we find that sampling our data weekly is much better for our ML models since it smoothens the data. Furthermore, we experiment with the input size to find that a smaller input size produces better models. Specifically, an input size of 8 produces the best models for us.
- The resulting model has good predictive power for the first few weeks. In fact the predicted curve is quite close to the actual curve for the first 12 weeks.
- After that period, there is a divergence between the actual values and the predicted values and the model loses any predictive power.
- Overall, we have found a good model for predicting the prices for a quarter, which makes our model quite practical.

Next Steps

Many of our models suffer from underfitting to some degree. RNNs can be sped up significantly using GPUs since they can do many computations in parallel, and in our case, that would allow us to significantly increase the number of parameters in our models. This would allow us to better model the stock price.

There are a large number of hyperparameters that went into our models and we couldn’t possibly check the performance as we change all of them. A longer version of this project could vary many of these hyperparameters such as the learning rate for gradient descent, changing the activation functions etc., and observe the corresponding effect on our models. We could also change the loss function and the optimizer for the gradient descent to Adam or Adagrad and record any difference in our model. Overall, there are a large number of possibilities of further steps that can be undertaken for a more in-depth study.

We attach the python code used to prepare this report as an appendix. The code is not necessary to follow the report but can be used to verify any statements made in the report.