Stock Price Predictions using Sequential Models

Charles Barnes, Kenyon Guo, Rafaela Melikian - UCLA Math 156

1 Abstract

We chose to look at the stock price histories of Tesla, Lockheed Martin, and Coca-Cola in an attempt to use neural networks to predict their future stock prices. We analyzed three types of neural network architectures: Long Short-Term Memory (LSTM), gated recurrent unit (GRU), and the simple recurrent neural network (RNN). Our LSTM model has four LSTM layers with Dropout layers of rate 0.2. Similarly, our GRU model has four GRU layers with Dropout layers of rate 0.2, and our RNN model has four RNN layers with a Dropout layers of rate 0.2. In each model, we followed these layers with a fully-connected layer before finishing with an output layer. All our models were compiled using the Adam optimizer and loss was determined by mean squared error. Across all companies, we discovered that the GRU and LSTM models performed best with the highest accuracy while the RNN model performed worst.

2 Introduction

Stock price prediction is an exceptionally difficult task. Countless factors affect a company's performance, including public perception, governmental influence, resource acquisition, among many others. Numerically analyzing all of these factors can prove impossible, and every company can be affected by the market in different ways. Knowing the lowest stock price of a company over a period of time is important for individuals looking to buy stocks, and knowing the highs is helpful for when the individual wants to sell their stocks in order to maximize their profits in realized gains or avoid large losses in value through realized losses. A method for prediction can be used by a company to predict their own future stock price, by a financial analyst, or by anyone looking to buy stocks at a low price and/or sell later on. Furthermore, accurate prediction could give us a heads-up on market crashes or market booms, thus potentially saving individuals from financial distress.

3 Background

The volatility of the stock market makes the stock market returns hard to predict, yet the consequences of being able to make accurate predictions are enticing. Thus, it is one of the most researched topics. Many scholars have looked at ways to analyze and make predictions using machine learning techniques in recent years. For example, [8] looked at two techniques—artificial neural networks and random forests—to predict the closing price for the next day on datasets for five companies: Pfizer, JPMorgan, Goldman Sachs, Nike, and Johnson and Johnson. They incorporated three aspects of financial data as inputs into the models: open, high, low, and close stock prices. They evaluated the models using root mean square error and mean absolute percentage error. For four out of the five companies, they concluded that the artificial neural network produced the most accurate results.

Another group we looked at reviewed approaches for stock market prediction using machine learning. [6] provided a broad look at different methods from several other papers and discuss the challenges with each method. They looked at traditional algorithms like SVM and Least Squares, as well deep learning techniques such as LSTM and CNN. By studying several papers, they concluded that traditional methods generally have sensitivity to outliers, deep-learning like RNNs and LSTMs are accurate but training in long and requires large memory requirements, and graph-based approaches are just not as accurate as traditional machine learning approaches.

The majority of the articles that we found involving sequential models used the LSTM model to predict stock prices over time and analyzed the differences between LSTM and simple models like a simple moving average of the last 50 days to predict the next days stock price [1] We decided that we wanted to compare the different recurrent neural network variants (Simple RNN, LSTM, and GRU) to compare the performance between these more advanced sequential models to see if there was a specific model that was most effective for predicting stock prices based on the past performance.

4 Methods and Theory

4.1 Method 1: RNNs

The most basic sequential model that we used in our stock price prediction efforts was the Recurrent Neural Network (RNN). Unlike the standard feed-forward neural networks where each layer takes in the data and passes it to the next layer, an RNN also has a short-term memory in that it uses the most recent cell state as another input which affects the current output. This allows the neural network to take advantage of the prior states to use the trends of the data to guide the decision making when the model is trained and being used to make predictions [2, 5].

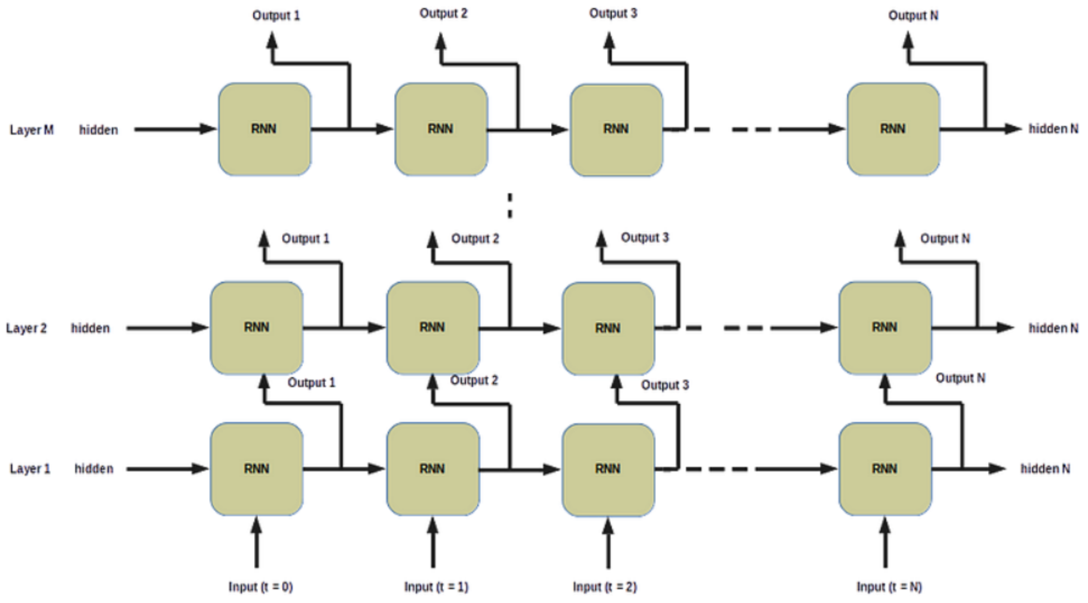

Fig. 1: RNN model representation [5]

This diagram shows a generalized simple RNN model with M layers and a sequence of N inputs that is fed into the model which affects the hidden state and calculates the output that will be passed to the next layer of the neural network [5]. Just like the layers of a feed-forward neural network, each layer of an RNN takes in the previous layer's inputs and does a linear calculation (with bias) on the input but it includes a function of the cell state in this calculation. Finally, the activation function normalizes the result and passes it to the next layer of the neural network. Due to the flexibility of this neural network, it can be used for a variety of tasks, but the added complexity of a cell state lends this type of neural network to be used specifically when the past data may have a significant impact on the future data. The network has to be set up carefully to ensure that the input is of the correct dimension, but beyond that, it is a very robust model that can be customized and combined with multiple layers for more complex tasks.

4.1.1 RNN model architecture

Like the multi-layer perceptron and similar feed-forward neural networks, the RNN model has a given number of nodes that take in information from the input and process it before passing the output to the next layer of the system. There are also hyperparameters such as "return_sequences," "input_shape," and "activation" that we had to set to construct our neural network.

In each of the four RNN layers, we set the output to be a 100-unit dimension because we wanted a robust network that could analyze the complicated trends of a stock's behavior in the market. Furthermore, as we wanted to test a simple RNN that took advantage of the previous state as well as the prior states to use the trends of the stock to improve predictions, we set the "return_sequences" argument to True so that the cell state would return the entire sequence of prior outputs when each data point is run through the network. Next, we set the "input_shape" to reflect the required dimension of the stock price data, and chose the "tanh" activation function to normalize the output data between -1 and 1, which allows the network to use multiple layers in succession due to these non-linearities in between the layers. We alternated the four RNN layers with dropout layers of rate 0.2, which set 20% of the input nodes to 0 while scaling up the remaining nodes passed to the next layer by 25% to maintain the total sum of the inputs. These dropout layers help to prevent overfitting which is important in a stock-price prediction model because trends can change at any time. Lastly, we included a fully-connected layer with an output of 100 units and one final dropout layer of rate 0.2 before passing the data into the output layer, which would output the predicted value of the stock at the current time. The final model had 80,701 trainable parameters.

4.2 Method 2: LSTM

One of the issues with a recurrent neural network is that due to the nature of backpropagation and the training of a recursive cell state, the models would run into a vanishing or exploding gradient problem where the model becomes untrainable for long-term memory because of the repeated multiplication of the gradient as the model needs to adjust its initial state before any of the inputs have been passed to it in order to produce a different output [3]. As a result, the Long-Short Term Memory model was developed as a way to work around this problem through a series of modifying gates (input, output, and forget) that force the cell state to only retain relevant information from the previous sequence to use when making the current prediction. These gates are the red non-linearities in Fig. 2 below [4]. During training, these gates learn which information is relevant to keep and have a sigmoid activation function that normalizes the data from the cell state between 0 and 1 to either "forget" the data in the cell state by multiplying it by 0 or "remember" it by multiplying it by 1, or any proportion in between for a degree of "memory" to pass to the next cell-state. Additionally, the blue non-linearities in Fig. 2 are the standard RNN architecture of an activation function that takes in both information from the newest input and the last cell state in order to output the newest information that will be passed to the next layer of the neural network.

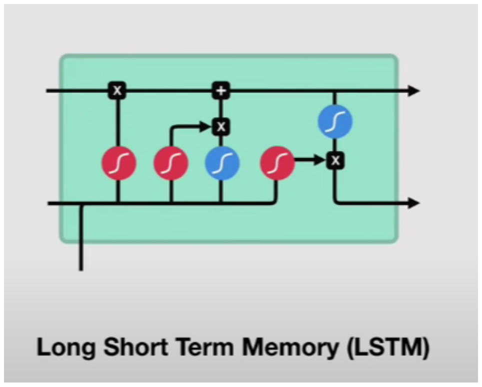

Fig. 2: LSTM cell representation [4]

4.2.1 LSTM model architecture

Because we wanted to maintain as much similarity between the models as possible, we used the same overall architecture of the neural network as the RNN model for the LSTM model, just with the four RNN layers swapped for LSTM layers instead. The LSTM layers had 4x as many trainable parameters as the RNN layers with the same hyperparameters, which increases the complexity and computational expense required to train the network. The final model had 292,901 trainable parameters.

4.3 Method 3: GRU

Our third model utilized the Gated Recurrent Unit, which is another variation of a modified RNN that was developed to combat the vanishing and exploding gradient problem that the RNN model faces. Instead of the cell state that the LSTM model uses, the GRU has a hidden state that is changed through 2 gates (update and reset) that work in a similar manner to the LSTM cell state gates. These gates are the red non-linearities in Fig. 3 below [4]. Over the course of training, these gates learn what information is relevant to keep in order to model long-term trends in sequential data. As a result, the less gates and tensor operations that the GRU requires compares to an LSTM model of comparable size leads to less trainable parameters in the overall model and quicker training than an LSTM model.

4.3.1 GRU model architecture

Once again, we used the same overall architecture of the neural network as the RNN and LSTM models for the GRU model, just with the four RNN/LSTM layers swapped for GRU layers instead. The GRU layers had over 3x as many trainable parameters as the RNN layers with the same hyperparameters, which increased the complexity and computational expense required to train the network over the RNN model but remained less complex than the LSTM model. The final model had 222,901 trainable parameters.

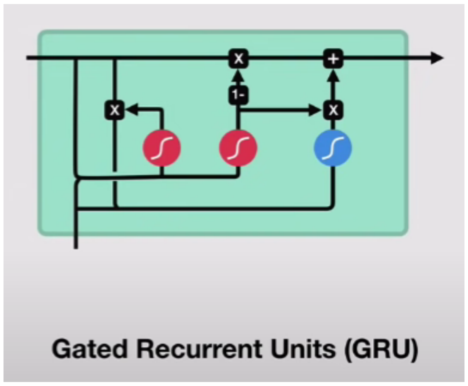

Fig. 3: GRU cell representation [4]

5 Dataset Description

We used three stock datasets that were sourced from Yahoo Finance: Lockheed Martin Corporation (LMT), Tesla, Inc. (TSLA), and The Coca-Cola Company (KO). The datasets each included the following 7 variables:

Variable	Explanation
Date	The day which these numbers refer to for the given stock
Open	The price of the stock at market open on the given day
High	The highest price of the stock on the given day
Low	The lowest price of the stock on the given day
Close	The price of the stock at market close on the given day
Adj. Close	Closing price adjusted for stock splits and dividends
Volume	Number of shares traded on that day

In our preprocessing to create Training and Testing datasets, we used the “Close” price of the stock as the metric for each day. For all the datasets, we selected the stock’s historical data from between January 1st, 2012 and July 1st, 2014, a 2.5 year period with 626 total business days. From here, we split the data so that the first two years (80%) would be the training data and the last 6 months (20%) would be the testing data, to model the performance of the stock over time. From there, we created sliding windows so that the training and testing datasets would include the “Close” price of the prior 60 days while the output training and testing datasets (which are used for training and validation) would just include the current day’s “Close” price [7].

6 Results

We used Root Mean Squared Error to evaluate the performance of each model over the testing dataset:

Stock	RNN - RMSE	LSTM - RMSE	GRU - RMSE
LMT	18.015	8.4569	9.5650
TSLA	119.26	116.92	117.86
KO	1.7197	1.7368	1.7948

The reason that we chose this error metric is because it allows us to magnify the effects of predicted datapoints that are extremely different from the corresponding actual value while maintaining the same units as the stock price ($). In my analysis of the Lockheed Martin Corporation dataset, I found that the RNN model had about twice the RMSE (18.015) as the LSTM and GRU models (8.4569 and 9.5650). My group members had different results and the effects were not as pronounced due to the differences in the datasets and the specific stock trends at the chosen testing dataset. We did find that the LSTM and GRU models had much more accurate predictions than our RNN models when comparing them to the actual stock price of the testing datasets of all three stocks. The graphical representation of this performance is shown below in Fig. 4 for Lockheed Martin Corporation and in the Appendix in Fig. 5 and Fig. 6 for Tesla, Inc. and The Coca-Cola Company.

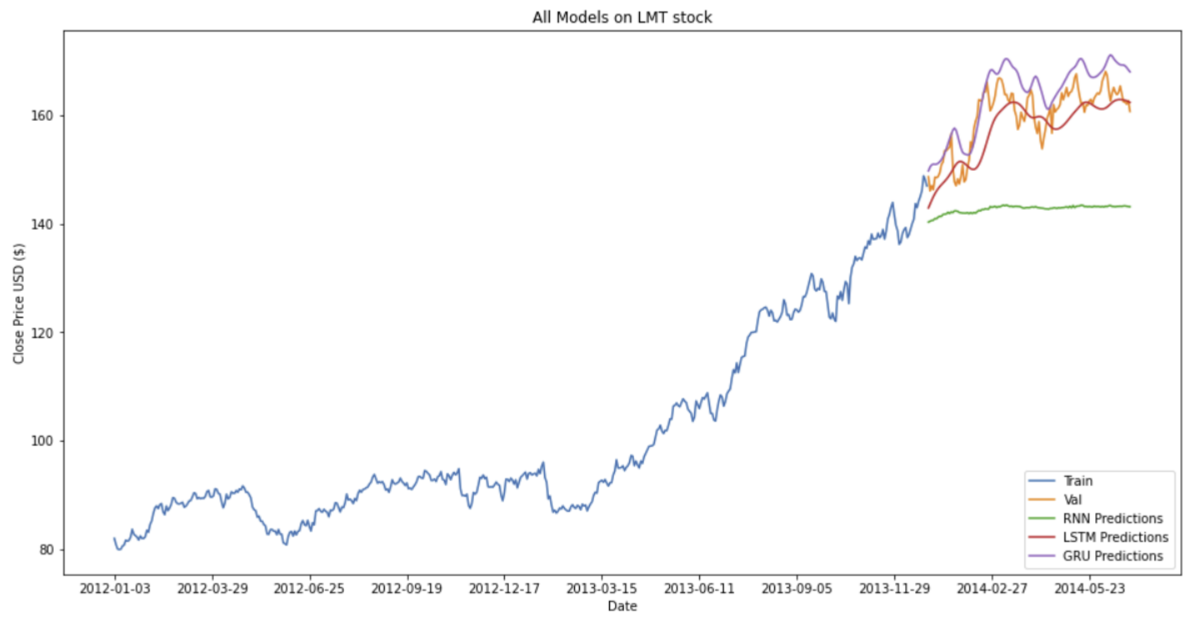

Fig. 4: Model performance of Lockheed Martin Corporation Stock Price over training/testing set

Intuitively, the performance discrepancy makes sense because of the vanishing and exploding gradient problem that the RNN faces compared to the LSTM and GRU models [3]. In the case of a vanishing gradient, as the model is trained through gradient descent and the gradient becomes smaller, backpropagation through time will cause this gradient to be multiplied over and over again, leading the actual change in weights to be extremely close to 0, thus causing the training of the model to effectively stop, leaving the model at the mercy of the randomized initialization of weights or with a half-trained model that cannot accurately predict the stock prices based on the past trends in this case. We see this problem clearly with the RNN predictions graph fairing significantly worse than the LSTM and GRU model predictions for the Lockheed Martin Corporation dataset.

7 Conclusion

The goal of our project was to analyze the differences between the various sequential models in relation to their degree of success in predicting stock prices. As every stock is different, and past behavior of a stock does not necessarily give any viable indications of future behavior, there is high variability in which model is the best performing stock price predictor but our conclusion is that the LSTM and GRU models were much more accurate compared to the simple RNN models.

Some next steps to improve confidence in stock price prediction would include additional hyperparameter testing and further variations in model architecture specific to each individual stock in order to tailor a model to a specific company's portfolio. Every stock and every company is different, and thus these models are not "one size fits all." In fact, the models are extremely inaccurate when used to predict the wrong stock and must be completely retrained over the specific time period in order to achieve viable results.

Additionally, the very nature of these sequential models makes it near impossible to predict long-term performance because each time step is dependent on the previous 60 time steps in our models. Thus, the only applicable field for these types of models is for daytraders, who may buy and sell stocks multiple times a day in the hopes of making a high volume of short-term gains. This is an unfortunate but possibly limitation of these sequential models because a truly accurate model that is capable of predicting several days in advance would cause massive upheaval in the economic outlook of the stock market if people were able to use sequential models to predict the long-term performance of a company's stock.

8 Data availability

We found the historical data for Lockheed Martin Corporation, Tesla, Inc., and The Coca-Cola Company on Yahoo! Finance, a freely accessible source of information for all publicly-traded stocks. One can easily view and download the data for such a company over any set of dates in a CSV file to use as they please.

9 Contributions Statement

Each individual selected a company to analyze: Rafaela looked at Tesla, Charles looked at Lockheed Martin, and Kenyon looked at Coca-Cola. We each found and downloaded our datasets. We collectively decided that in order for each of us to fully understand and apply the LSTM, GRU, and RNN architectures, we would form each model on our own dataset to practice. So, each of us made an LSTM model, GRU model, and RNN model and applied it to our respective dataset. We looked at every model that each of us made and decided to go with Kenyon's LSTM model and combined model, Rafaela's GRU model, and Charles's RNN model for the final submitted code. Thus, each model is the same for each dataset. We each adjusted parameters, such as the Dropout rates. For visualizations, Kenyon made the graphs which detail the training set, test set, and model performance. Rafaela made the graph that compares each model's performance with the test set alone. Charles helped with both and improved the formatting of each graph to give more relevant information on the axis labels and the overall readability.

Since there were only three of us, we each reviewed literature and discussed our findings together in order to collectively develop a game plan. We tested our own code pertaining to our own dataset, and later discussed what worked and what did. When we encountered coding issues, such as confusion with syntax, code not compiling, etc., we all looked at the issue together and usually one of us knew the remedy. Charles was especially helpful with debugging. Rafaela helped get the study room/meetings set up as well as writing the shared portions of the report while Charles and Kenyon provided edits. Kenyon always had great insight when it came to the intuition behind some of the concepts we were dealing with. Charles helped keep us on track throughout the past few weeks so that we didn't put work off until the last minute.

References

[1] Katherine (Yi) Li. *Predicting Stock Prices Using Machine Learning*. neptune.ai, July 2021. URL: `https://neptune.ai/blog/predicting-stock-prices-using-machine-learning`.

[2] Christopher Olah. *Understanding LSTM Networks – colah's blog*. Github.io, Aug. 2015. URL: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[3] Barak Or. *The Exploding and Vanishing Gradients Problem in Time Series*. Medium, Oct. 2020. URL: `https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22`.

[4] Michael Phi. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. www.youtube.com, Sept. 2018. URL: `https://www.youtube.com/watch?v=8HyCNIVRbSU`.

[5] Pritesh Prakash. *Understanding RNN Step by Step with PyTorch*. Analytics Vidhya, July 2021. URL: `https://www.analyticsvidhya.com/blog/2021/07/understanding-rnn-step-by-step-with-pytorch/`.

[6] Payal Soni, Yogya Tewari, and Deepa Krishnan. *Machine Learning Approaches in Stock Price Prediction: A Systematic Review*. 2022. URL: `https://iopscience.iop.org/article/10.1088/1742-6596/2161/1/012065/pdf`.

[7] Bee Guan Teo. *Stock Prices Prediction Using Long Short-Term Memory (LSTM) Model in Python*. The Handbook of Coding in Finance, Oct. 2021. URL: `https://medium.com/the-handbook-of-coding-in-finance/stock-prices-prediction-using-long-short-term-memory-lstm-model-in-python-734dd1ed6827`.

[8] Mehar Vijh et al. "Stock Closing Price Prediction using Machine Learning Techniques". In: *Procedia Computer Science* 167 (2020), pp. 599–606. DOI: `10.1016/j.procs.2020.03.326`.

10 Appendix

10.1 Hyperparameter selection

Hyperparameter	Value	Explanation
input_shape	(x_train.shape[1],1)	Match the dataset
units	100	Robustness of model
return_sequences	True	Include entire sequence of hidden states
activation	tanh	Default activation function for RNN/LSTM/GRU
Dropout	0.2	To ensure adequate overfitting protection
epochs	20	Performance/computational efficiency
batch_size	32	Performance/computational efficiency
optimizer	adam	Premier neural network optimizer
loss	mean_squared_error	Magnifies worst performance in regression/prediction
seed	1/123	Arbitrary, for reproducibility
sliding window size	60	12 weeks of data for long-term trends
learning_rate	0.001	Default adam optimizer learning rate

This table includes the full list of hyperparameters selected in our models. While many of them were chosen and the explanation is given in the report, the rest are explained in the table. Additionally, because each stock's performance and price ranges are wildly different, we used MinMaxScaler from the preprocessing library of Python's sklearn module to scale and normalize the data so that effects are not overly magnified in the neural network. After scaling the data and running it through the model, the predictions must then be returned back to their original values through an inverse transformation.

10.2 Other Visualizations

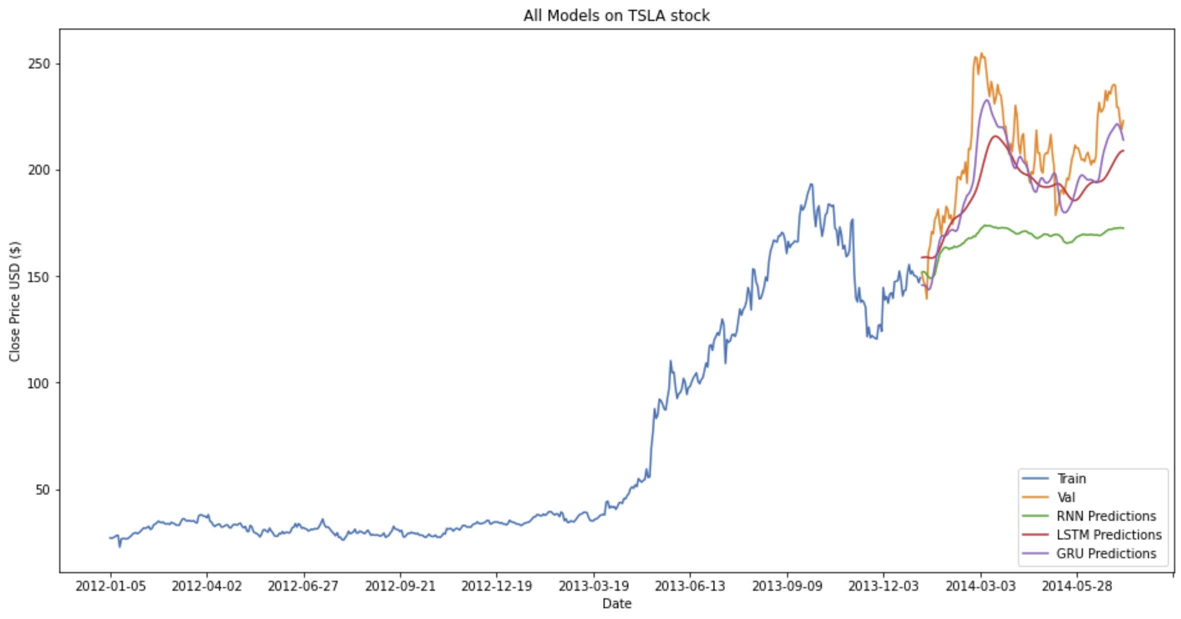

Fig. 5: Model performance of Tesla, Inc. Stock Price over training/testing set

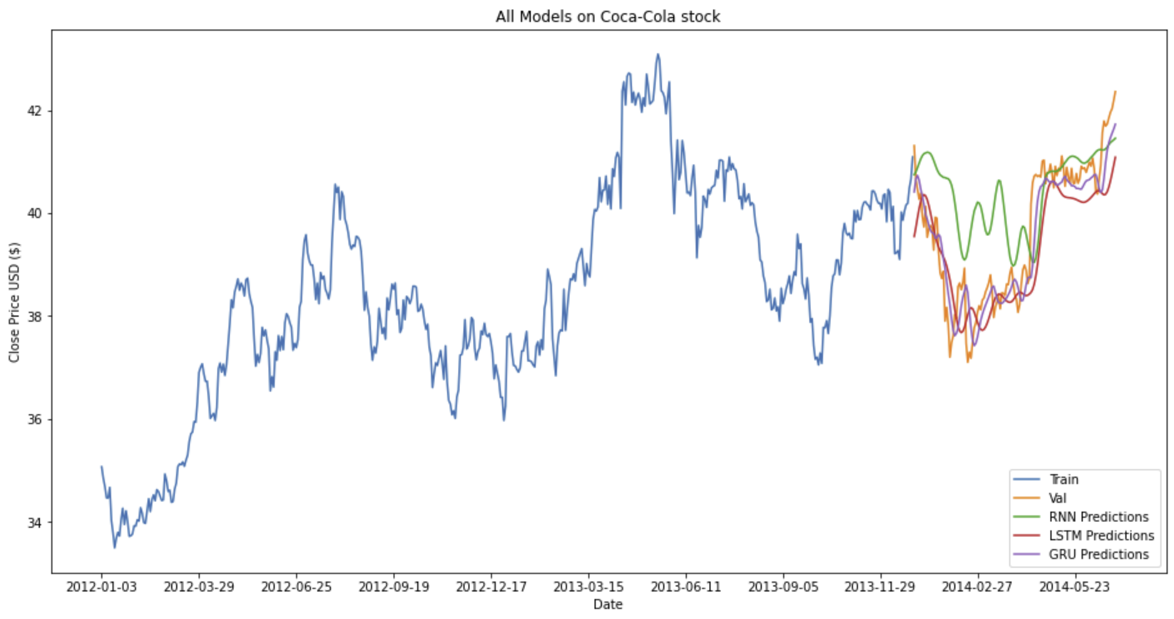

Fig. 6: Model performance of The Coca-Cola Company Stock Price over training/testing set