CS 536: Machine Learning – Presentation

Topic: Time Series Prediction Problem — Detect future values of stock prices of a set of companies using LSTM RNN

Likhit Garimella lg836 | Aniket Anvekar ava66 | Manasvini Nittala mn777 | Sahithi Reddy Sakinala ss4362 | Jahnavi Manchala jm2658

(1) Problem Statement

Problem: To predict the upward and downward trends that exist in a set of companies say 100, taking Google's stock price for demo.

Model: To do so, the model that we will implement is RNN and LSTM (long short-term memory). We will make an LSTM that will try to capture the downward and upward trend of Google stock price. For this, LSTM is the most powerful model that can do this. It performs well better than the traditional RMI model.

How to approach the problem: Train the RNN and LSTM models on 10 years of the Google stock price, from beginning of 2012 to the end of 2022. And then, based on this training and based on the correlations identified by the RNN and LSTM of the Google stock price, we will try to predict for the first month of 2023.

(2) Part-1: Data Preprocessing

Importing the training dataset csv file and take the first 2 columns from it.

Dataset: The dataset is taken from Yahoo Finance website in csv format. The dataset consists of Open, High, Low and Close Prices of Google stocks from January 2012 to December 2022 — total 2768 rows. It includes information like opening price, high price, closing price, low price and volume of the stock sold on each financial day.

Data Preprocessing

First check the dataset for any missing data and null values. Impute if any are found, and then normalise the data and extract the features.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2768 entries, 0 to 2767
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    2768 non-null   object
 1   Open    2768 non-null   float64
 2   High    2768 non-null   float64
 3   Low     2768 non-null   float64
 4   Close   2768 non-null   float64
 5   Volume  2768 non-null   int64
dtypes: float64(4), int64(1), object(1)
memory usage: 129.9+ KB
```

Here we can see the information about the data, and there are no null values, so we do not impute the data.

(3)

Feature Scaling: 2 effective ways of applying feature scaling — standardisation and normalisation (equations). While building our RNN, we apply normalisation.

Importing min max scaler class from Scikit learn library.

Then apply normalisation, by getting the minimum stock price and maximum stock price from the input training set, to be able to apply it in the normalisation formula.

Now, we get the normalised stock prices between 0 and 1 — the last value tends to go to 1 as the stock prices are increasing.

(4)

Creating a data structure with 60 timesteps and 1 output: At each time T, the RNN is going to look at the 60 stock prices before time T — that means, based on the stock prices between 60 days before time T and time T, the RNN will try to learn and understand the correlations and trends. And based on this, it will predict the next output, that is, the stock price at time (T+1).

Why choose 60 timesteps: Tried 20, 30, 40 which weren't enough to capture enough trends. Ended up with 60 being the best. Also 60 timesteps is 60 previous financial days. 1 month has 20 financial days, so 60 timesteps correspond to 3 months.

x_train: input of the neural network — 60 previous stock prices before time T.

y_train: output of the neural network — stock price for time (T+1).

Convert x_train and y_train lists to numpy arrays and pass them into the RNN.

In x_train table, 0th row values correspond to the previous 60 stock prices upto the 60th financial day. 1st row values correspond to the previous 60 stock prices upto the 61st financial day. (For every i and i+1 row, there would be 59 values in common).

y_train table contains the stock price at time T+1, that is, the set of 61st predicted stock prices for every corresponding 60 stock prices from the rows of x_train table.

(5)

Reshaping the data: Adding some more indicators so that it might predict better.

These dimensions are — number of observations (or) the batch size = 2768, number of time steps = 60, number of indicators = 1.

Methodologies

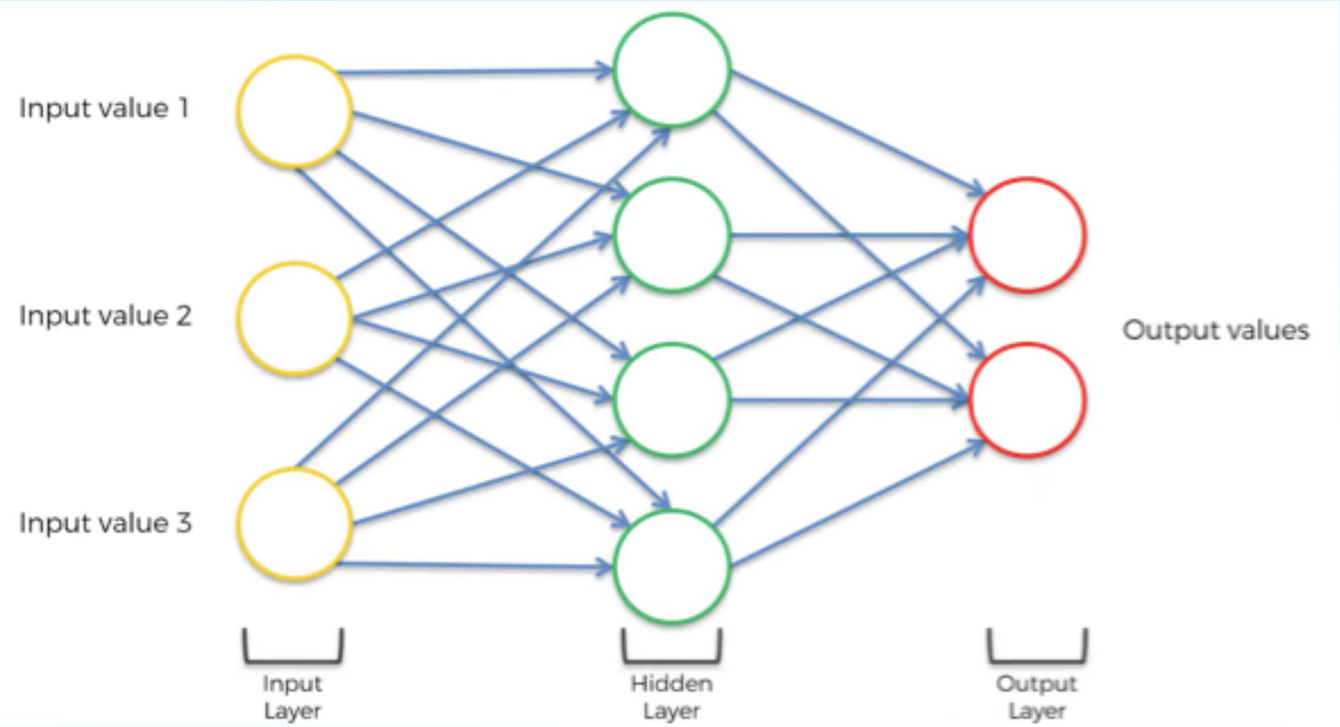
To build our model we are going to use the Simple RNN, LSTM and GRU, and compare all these models. All the input features are transformed into a specified range i.e. [0-1] using the min-max normalisation.

Four sequential layers have been stacked together and one dense layer is used to build the RNN model using Keras deep learning library. For each model a prediction error is calculated.

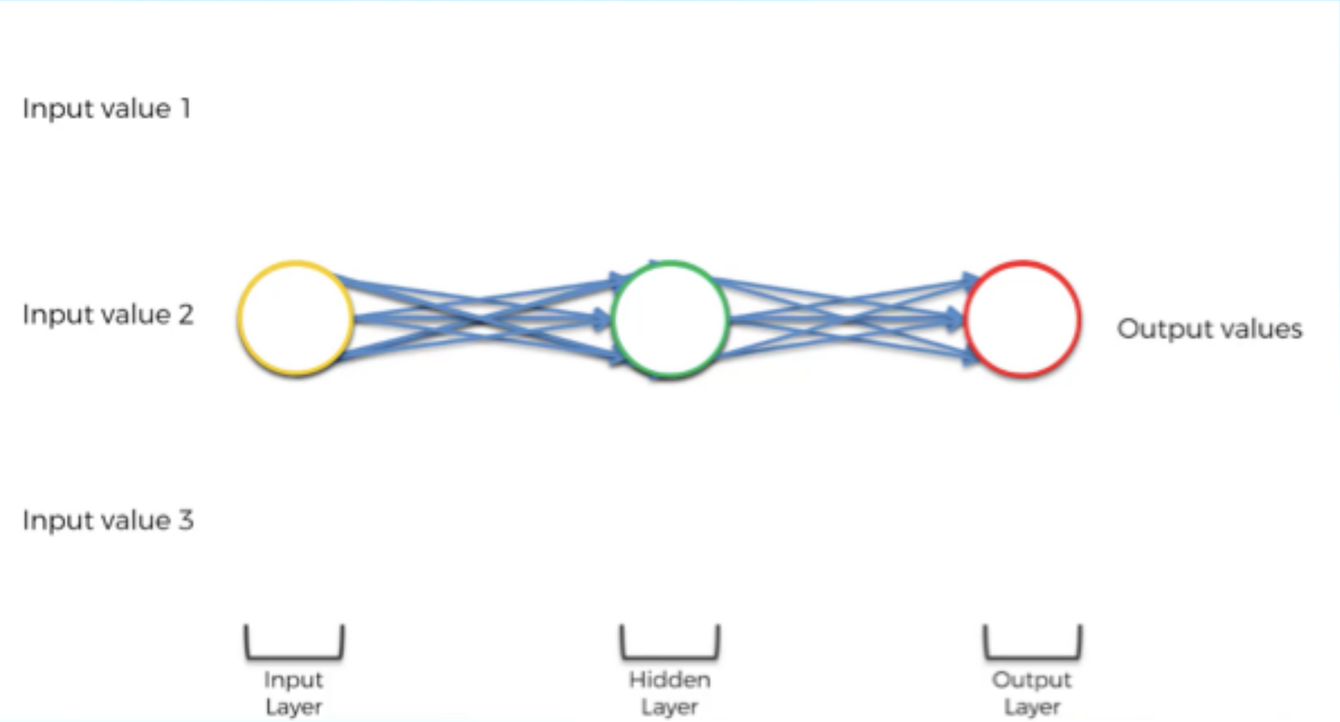
RNN: Recurrent neural networks (RNNs) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data.

RNN can't store long time memory, so the use of the Long Short-Term Memory (LSTM) based on "memory line" is proved to be very useful in forecasting cases with long time data.

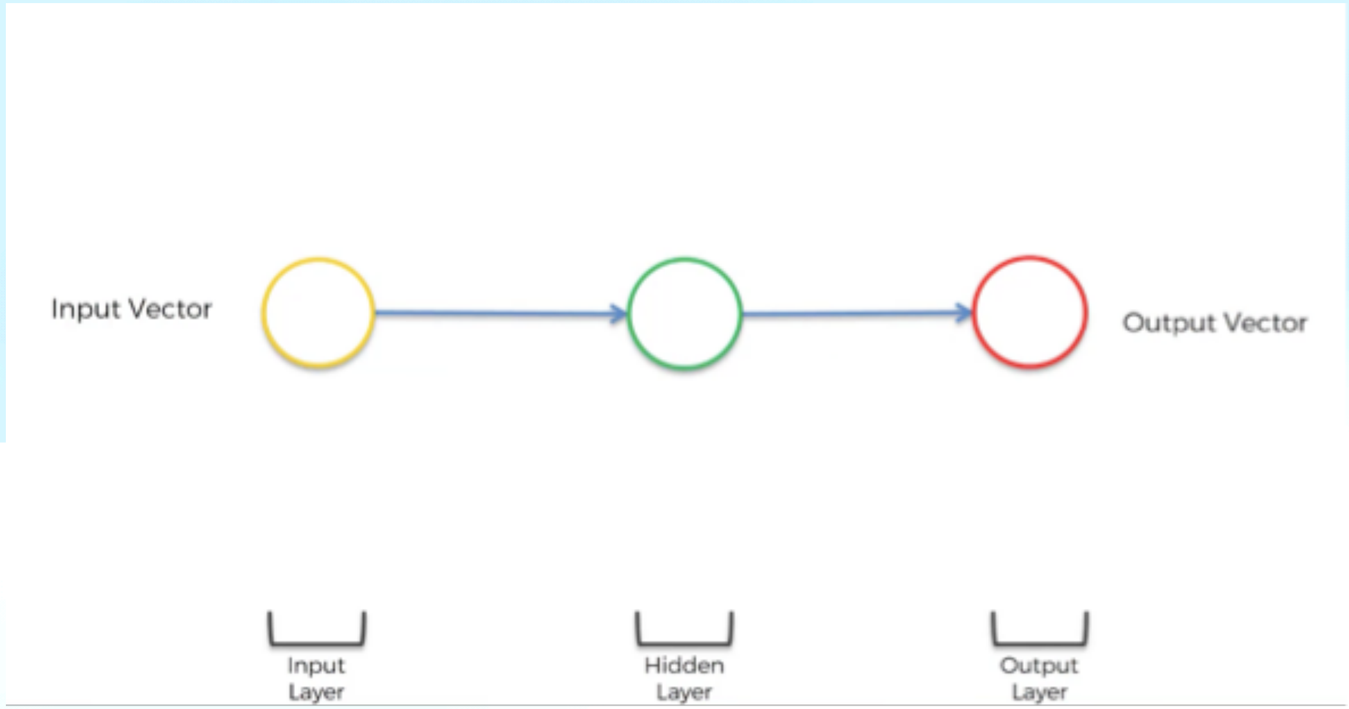
Representation of RNN



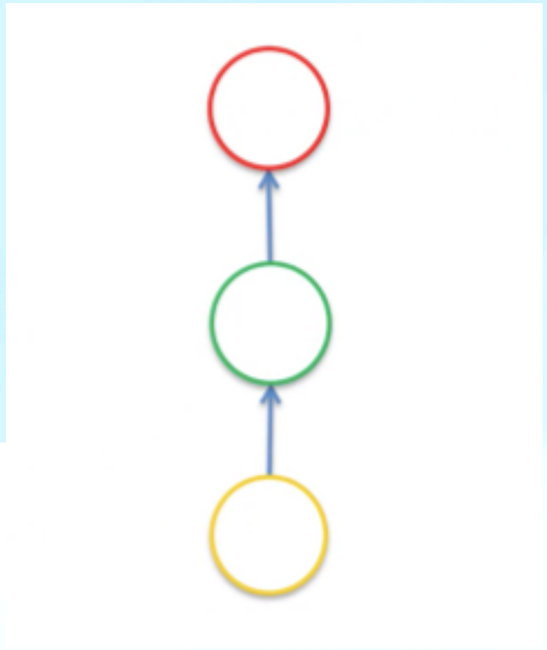
Representation of RNN



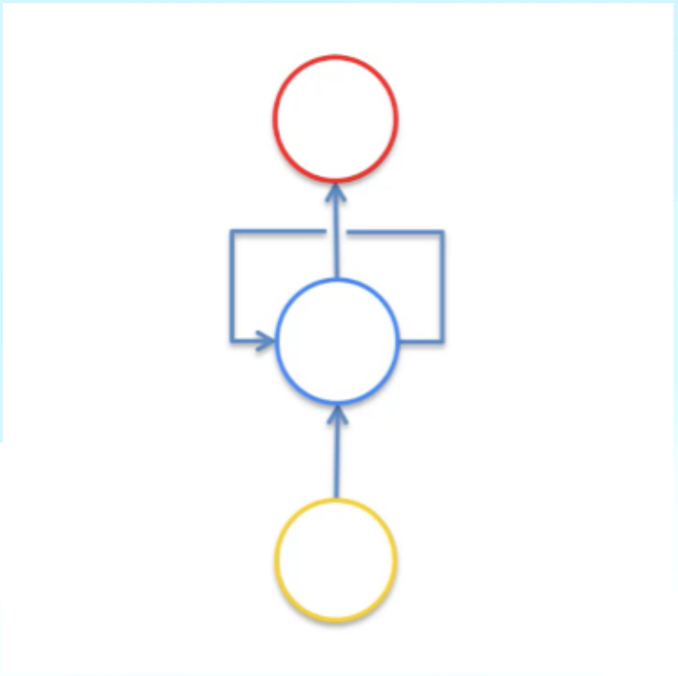
Representation of RNN



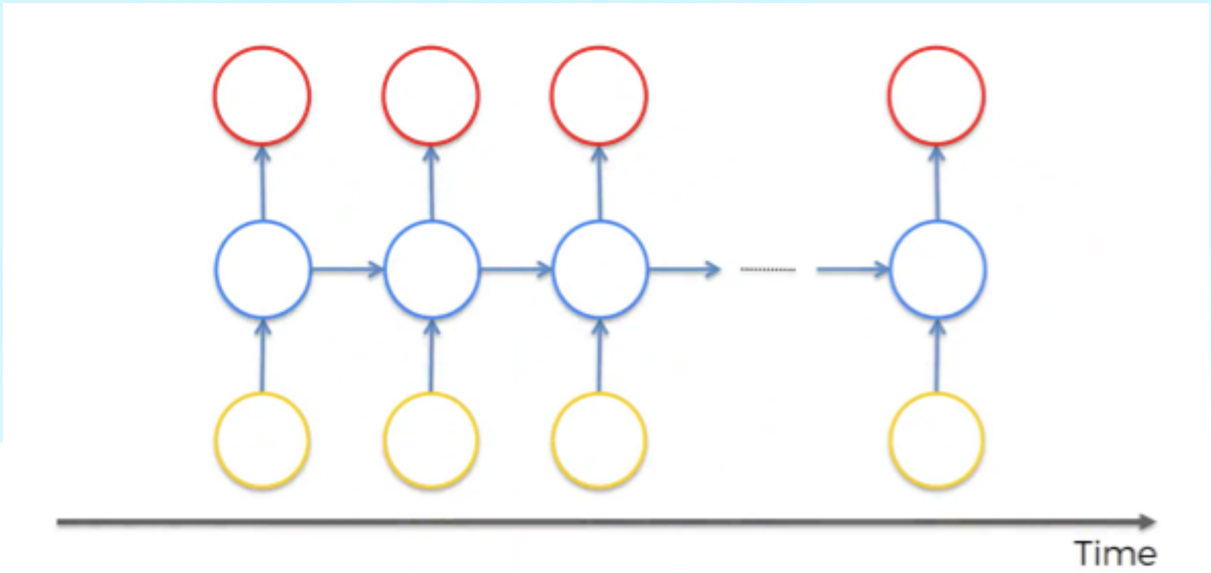
Representation of RNN



Representation of RNN



Representation of RNN



Methodologies

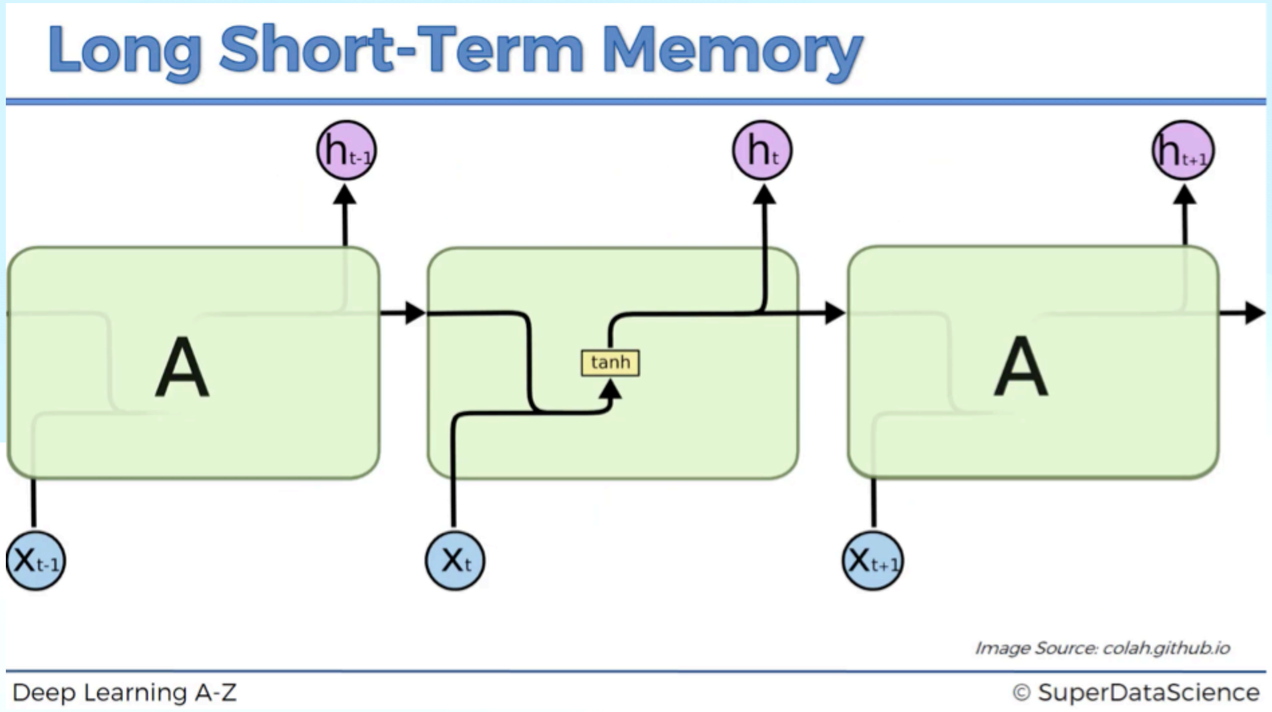
How LSTM solves the problem:

LSTM: LSTMs are a type of RNN that use special type of memory cell and gates to store and output information. LSTM networks combat the RNN's vanishing gradients or long-term dependence issue by ignoring useless data/information in the network. LSTM is effective at storing and accessing long-term dependencies. They are slower to train and run than other types of RNNs.

GRU: GRUs are simplified version of LSTMs that use single “update gate” to control the flow of information into the memory cell. GRUs are easier to train and faster to run than LSTMs, but they may not be as effective at storing and accessing long-term dependencies.

The key difference between RNNs, LSTMs, and GRUs is the way that they handle memory and dependencies between time steps.

Representation of LSTM



(6) Part-2: Building the RNN

Importing packages.

Sequential, to create a neural network object representing a sequence of layers.

Dense, to add the output layer.

LSTM, to add the LSTM layers.

Dropout, to add dropout regularisation — to avoid overfitting.

(7)

Adding the first LSTM layer: First LSTM layer takes 3 arguments.

(i) number of LSTM cells (or) memory units (or) neurons = 50. If we take less number of neurons like 3 or 5, we cannot capture the upward and downward trends very well. So, 50 neurons will get us a model with high dimensionality and better results.

(ii) return_sequences parameter set to True.

(iii) input_shape that contains the last two dimensions — time steps and indicators.

Lastly, adding some Dropout regularisation — 20% dropout.

(8)

Adding a second LSTM layer: Similar to how we added the first LSTM layer with the same number of 50 neurons, but don’t need to specify the input layer anymore. Similarly, adding 20% dropout

Adding a third LSTM layer: Completely similar how we added the second LSTM layer.

Adding a fourth LSTM layer: Similar to how we added 2nd and 3rd layers, but set return_sequences to False, because this is the last LSTM layer we’re adding and we are not returning anymore sequences.

(9)

Adding output layer: To add the last layer using Dense class with one input argument — number of neurons = 1, since output, which is the stock price at time T+1, has only one dimension.

Now, 2 things to do:

(i) Compiling the RNN with a powerful optimiser and the right loss, which will be the mean squared error.

(ii) Fit the RNN that we make, to our training set (x_train).

(10)

Compiling the RNN using the adam optimiser and mean squared error for loss.

Optimiser: Used adam optimiser because it's always safe, and powerful, and it always performs some relevant updates of the weights.

Loss: We are not doing classification in this case, so the loss is not going to be binary cross entropy. This is a regression problem because we have to predict a continuous values series. And the loss for this kinda problem is mean squared error.

(11)

Fitting the RNN — Connecting the neural network with our training set.

Input the 2 arguments into the regressor — x_train and y_train.

Connect the neural network using the fit method, and also execute the training over a certain number of epochs, that we will choose in the same fit method, which is = 100. 100 epochs because that's where we notice convergence. No convergence at 25 or 50 or 75. And batch size of 32 stock prices.

epochs — the number of iterations you want your neural network to be trained.

Convergence — Started with a loss of 3% 0.0307, in the last 10 epochs the loss stayed around the same level 0.0014.

(12) Part-3: Making the Predictions and Visualising the Results

First, getting the real stock price of 2023.

Importing the test dataset csv file and take the first 2 columns from it, and the training numpy array.

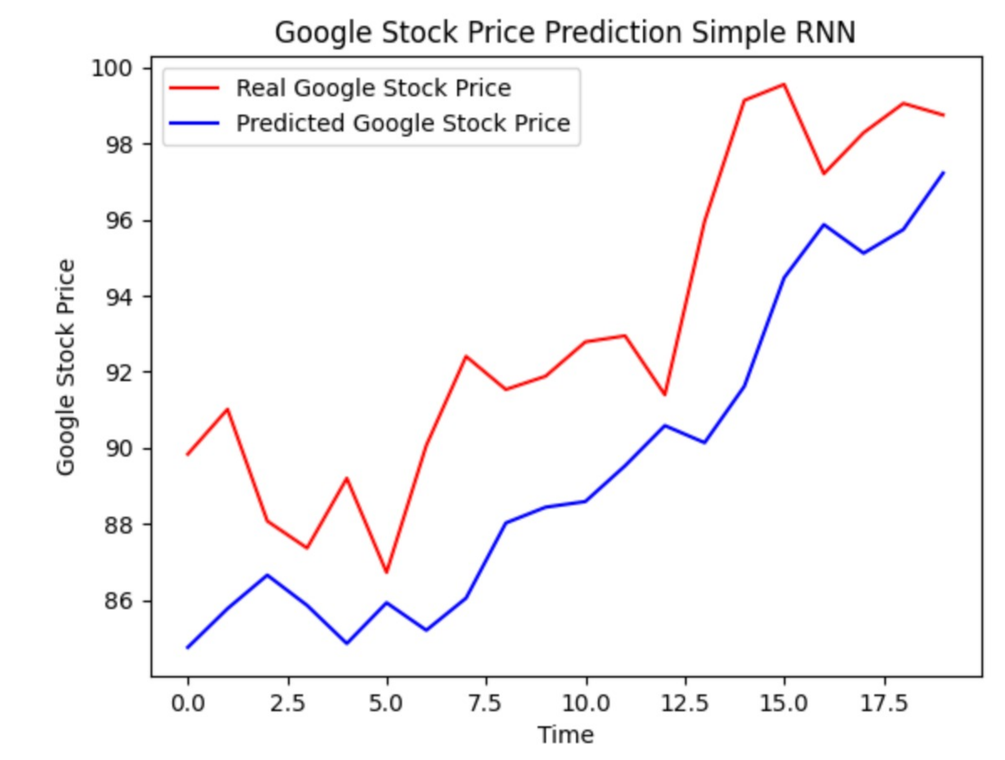
(13)

Now, getting the predicted stock price of 2023.

From both the training set (10 years data) and test set (first month of 2023).

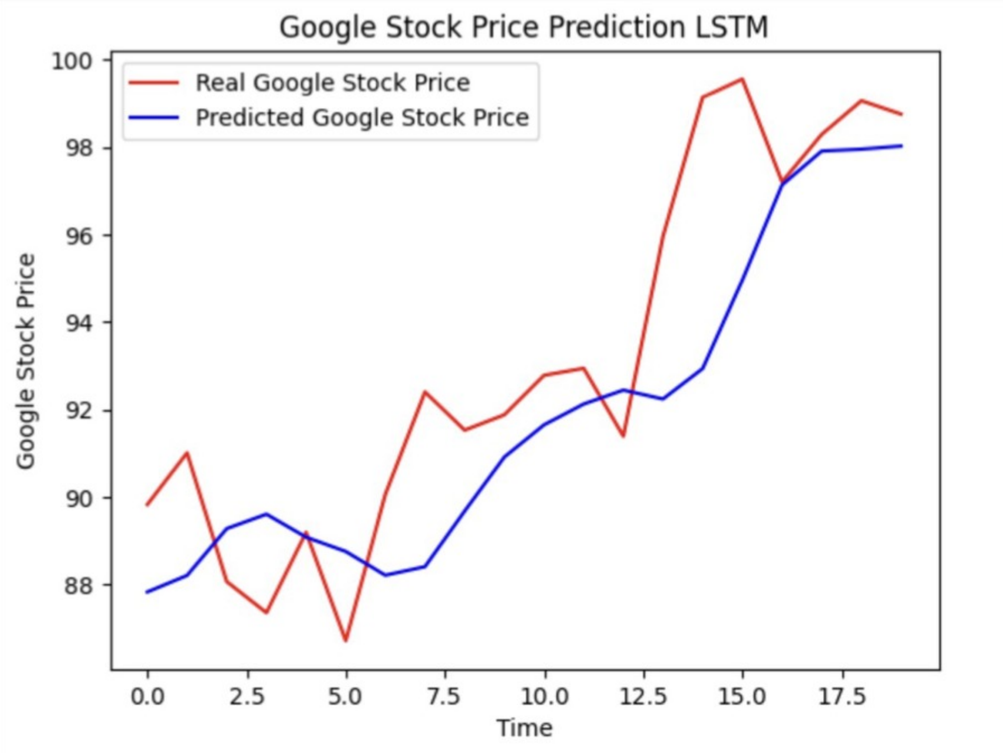
Graphs

10 years dataset, Google Stock Price Prediction using Simple RNN:



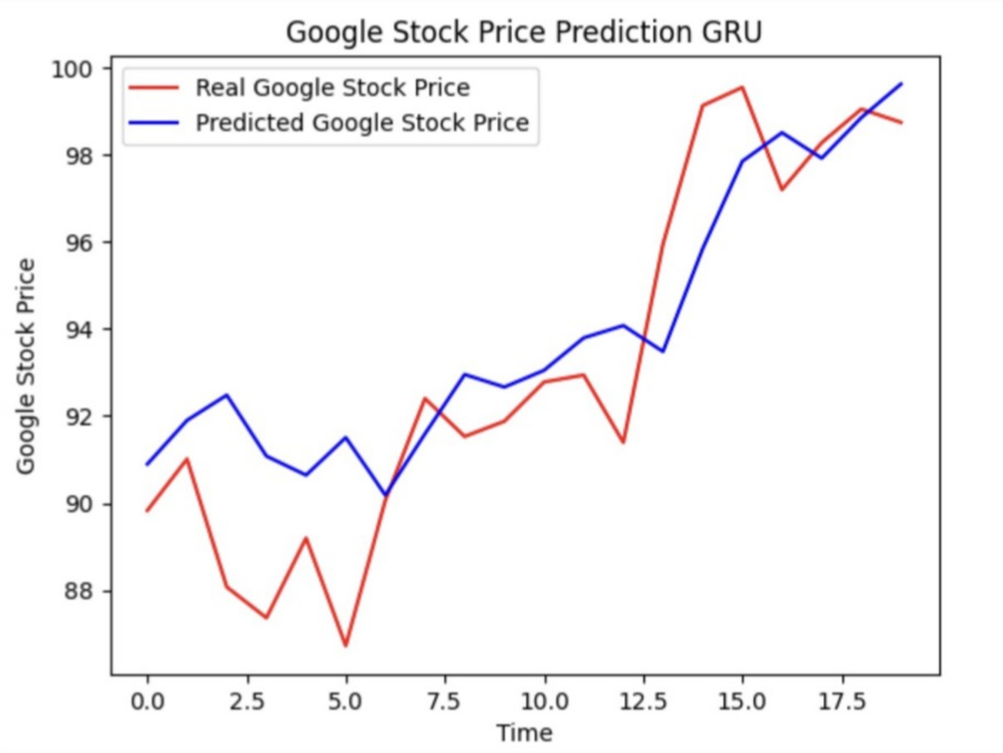
Graphs

10 years dataset, Google Stock Price Prediction using LSTM:



Graphs

10 years dataset, Google Stock Price Prediction using GRU:



Results — Numbers

```
Root mean squared error for simple RNN: 4.122156914877544
Root mean squared error for LSTM: 2.4942127075208815
Root mean squared error for GRU: 2.167411411712832
```

Learnings from the Project

→ Employing neural networks on time series data and predicting the future outcome i.e. the stock prices.

→ Data preprocessing — Normalising the data within a specific range.

→ Trained the RNN model by using the opening values of a given stock for that date. Compared different RNN models like LSTM & GRU to get the best possible result.

→ We identified the trends and behaviour of a particular variable over time.

→ Visualisation of the data — Comparing the actual and the predicted data over time and analysing the trend for each of them.

References — Research Papers

[1] Andrej Karpathy, Justin Johnson, Li Fei-Fei (2015, November 17). Visualizing and understanding recurrent networks. arXiv.org. Retrieved March 6, 2023, from https://arxiv.org/abs/1506.02078

[2] Andrej Karpathy (2015, May 21). The unreasonable effectiveness of recurrent neural networks. Retrieved March 6, 2023, from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[3] Klaus Greff, Rupesh K. Srivastava, Jan Koutn´ık, Bas R. Steunebrink, Jurgen Schmidhuber (2017, October 4). LSTM: A search space odyssey. arXiv.org. Retrieved March 6, 2023, from https://arxiv.org/abs/1503.04069

[4] Shi Yan (2016, March 13). Understanding LSTM and its diagrams. Retrieved March 6, 2023, from https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

[5] Christopher Olah (2015, August 27). Understanding LSTM Networks -- colah's blog. Retrieved March 6, 2023, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Sepp Hochreiter & Jurgen Schmidhuber (1997). Long Short-Term Memory. Retrieved March 6, 2023, from http://www.bioinf.jku.at/publications/older/2604.pdf

Thank You!