Stock Prices Forecasting Using LSTMs

Submitted
by

Pinaki Basu (19115087) **Paritosh Sanadhya (19115083)**
Arpit Kumar Pandey (19115032) **Rajat Raj (19115096)**
Davinder Singh (19115046) **Aman Soni (19115020)**

Supervisor

Prof. Ashish Kothyari

Department of Electrical Engineering
Indian Institute of Technology Roorkee
2022

Declaration

We hereby declare that the work which is presented here, entitled **Stock Prices Forecasting Using LSTMs**, submitted for the completion of course EEN 300: Industry Oriented Problem. We also declare that we have been doing my work under the supervision and guidance of **Prof. Ashish Kothyari, Department of Electrical Engineering, Indian Institute of Technology Roorkee**. The matter presented in this report is not submitted for award of any other degree of institute or any other institutes.

Date: 19/04/2022

[illegible]	Paritosh	[illegible]	Rajat	[illegible]	Aman Soni
Pinaki Basu	**Paritosh Sanadhya**	**Arpit Kumar Pandey**	**Rajat Raj**	**Davinder Singh**	**Aman Soni**
19115087	**19115083**	**19115032**	**19115096**	**19115046**	**19115020**

Certificate

This is to certify that the above statement made by the candidate is true to the best of our knowledge and belief.

Signature
Name of the Supervisor(s)
Designation
Department of Electrical Engineering
Indian Institute of Technology Roorkee

Acknowledgement

We would like to express our deepest gratitude to our project supervisor Prof. Ashish Kothyari, Assistant Professor, Department of Electrical Engineering, IIT Roorkee, for his able guidance and encouragement.

We are extremely grateful to Prof. G.N. Pillai, Head of department, Electrical Engineering, for providing us with the facilities required for completing the project.

We are very thankful to the course coordinator Prof. Ashish Kumar Behra, for his constant encouragement and support. We would like to express our gratitude to all teaching faculty in the Department of Electrical Engineering, whose ideas during evaluations aided us in completing our project.

We want to express our gratitude towards our batchmates for their support throughout the duration of the project. Last but not least, we want to express our gratitude to everyone who has helped us complete this project successfully, whether directly or indirectly.

Project members:

Pinaki Basu (19115087)
Paritosh Sanadhya (19115083)
Arpit Kumar Pandey (19115032)
Rajat Raj (19115096)
Davinder Singh (19115046)
Aman Soni (19115020)

Abstract

Our project's objective is to predict the stock-market prices with the help of Deep learning architectures, specifically LSTMs.

The ability to predict stock prices accurately is crucial for investors to maximize their profits. Usually traditional-approach based stock forecasts are driven by the stock price pattern, fundamental analysis of the company, volume changes, to a much extent by market news/rumors. Most of the time these types of forecasts are unreliable, and lead to big losses for the investors. Thus this traditional method of stock forecasting has a very low accuracy, and mainly depends on a person's skill and capability.

Now with the advancement of technology and use of digital computers, stock price prediction has become more efficient with the help of machine learning. Artificial Neural Network (ANN) has made it possible to accurately predict stock prices. Stock prices aren't just randomly generated numbers but rather can be analyzed as a sequence of discrete-time data. So with the right choice of model and proper training methods for forecasting time series data. We studied different types of models and tried a few variations. The three types of variation we worked upon are Stacked-LSTM, LSTM-GRU, CNN-LSTM. We have used IBM dataset from the official IBM website in training of our models. The results from these models are more accurate than those obtained from traditional methods.

Table of Contents

List of Figure

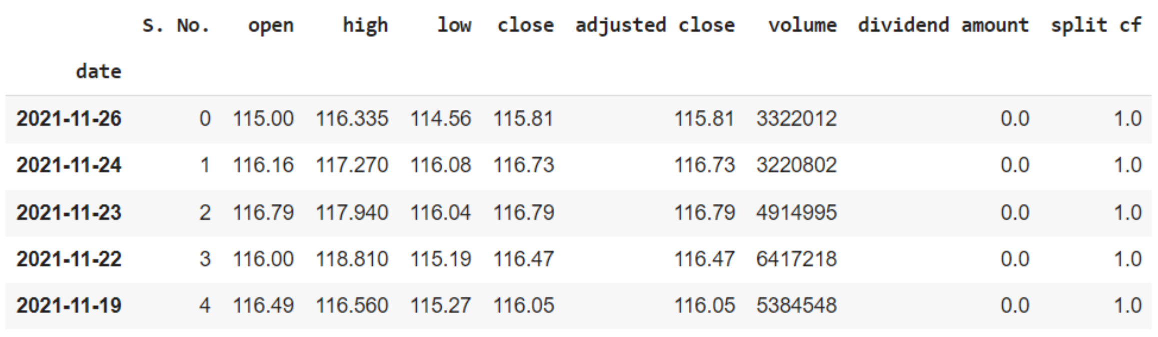
Chapter 1. Introduction And Architectures

1.1 Introduction

The stock market is a dynamic system where traders can buy and sell stocks and equities using various virtual platforms with the interface provided by brokers. Stock price doesn't remain constant over time because of supply and demand. With the fluctuation in the demand of stocks lead to change in stock price. From the beginning of the stock market, predicting the accurate future stock price is the primary goal of investors. To make accurate predictions, investors have to predict based on technical analysis, such as company' chart, stock market indices etc . It is not easy for investors to analyze and forecast the market by using all the information. Therefore, to predict trends in the stock market, many Artificial Intelligence have been developed.

Time-series analysis is one of the commonly used techniques which is used in many real world applications for future prediction such as weather forecasting and financial market prediction. It uses the continuous data in the time frame to forecast the future in the next time frame. Over the time various algorithms have been developed for time series prediction. In present time the popular algorithms are based on Recurrent Neural Network (RNN). A Recurrent Neural Network is a type of neural network that contains loops that can store information within the network. Recurrent Neural Networks use their reasoning from previous experiences to inform the upcoming events. In a basic RNN unit, there are 2 gates namely - input and output gate. As RNN is recursive, gradients get smaller and smaller, then finally vanish. This situation is called gradient vanishing. This problem makes RNN not the best choice and this is the motive of LSTM and GRU. LSTM, an advanced version, contains an extra Forget Gate. LSTM provides extra parameters which leads to more controllability and thus, gives better results. GRU is another option whose aim is to solve the vanishing gradient problem. GRU doesn't have cell state and output gate, therefore it has less parameters then LSTM. Also, powerful feature extraction algorithms like CNN can also be used before passing the data to the LSTM layers to provide better results.

The dataset used here is of IBM stock prices, containing the valuation of 5555 trading xdays. The 'Close' feature has been used as an input parameter, along with a window width of 40 days, with empirical results determining that this value provides the best results.



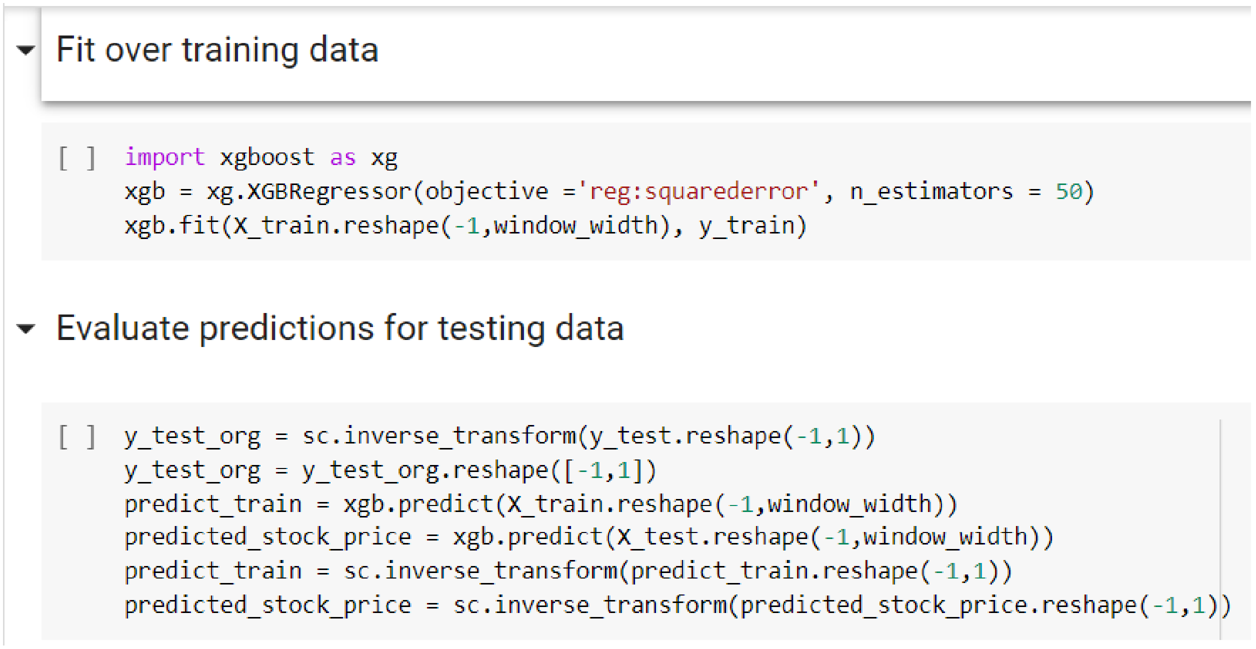
	S. No.	open	high	low	close	adjusted close	volume	dividend amount	split cf
date									
2021-11-26	0	115.00	116.335	114.56	115.81	115.81	3322012	0.0	1.0
2021-11-24	1	116.16	117.270	116.08	116.73	116.73	3220802	0.0	1.0
2021-11-23	2	116.79	117.940	116.04	116.79	116.79	4914995	0.0	1.0
2021-11-22	3	116.00	118.810	115.19	116.47	116.47	6417218	0.0	1.0
2021-11-19	4	116.49	116.560	115.27	116.05	116.05	5384548	0.0	1.0

Figure 1: IBM Dataset

1.2 A Model without RNN

Firstly, a model was built that used the XGBoost technique to predict the stock prices. Extreme gradient boosting, or XGBoost is a gradient-boosting algorithm making use of decision trees to perform classification/regression on a given dataset efficiently. It builds upon an ensemble learning method, where, instead of using just one decision tree, the dataset is divided and trained using multiple “Weak Learners”, whose outputs are then compiled to give the final output. This ensures diversity in training and reduces variance. Under ensemble learning comes two major categories, Bagging and Boosting. XGBoost, as the name suggests, is a boosting algorithm where models are built while reducing errors from earlier models and boosting the influence of the models that performed well. When training is done using the gradient descent method, it is known as gradient-boosting. XGBoost employs further optimizations on top of gradient-boosting like parallel processing, regularization, tree pruning, and handling missing values. As mentioned earlier, XGBoost uses decision trees, which is a method having a tree-like structure that works on the principle of conditions, where many decision nodes are present, leading to leaf nodes that determine the final decision.

XGBoost is a powerful technique used in a multitude of machine learning applications. In this case, the model is deployed to predict the stock prices.



```
import xgboost as xg
xgb = xg.XGBRegressor(objective ='reg:squarederror', n_estimators = 50)
xgb.fit(X_train.reshape(-1,window_width), y_train)
```

```
y_test_org = sc.inverse_transform(y_test.reshape(-1,1))
y_test_org = y_test_org.reshape([-1,1])
predict_train = xgb.predict(X_train.reshape(-1,window_width))
predicted_stock_price = xgb.predict(X_test.reshape(-1,window_width))
predict_train = sc.inverse_transform(predict_train.reshape(-1,1))
predicted_stock_price = sc.inverse_transform(predicted_stock_price.reshape(-1,1))
```

Figure 2: XGBoost Code

The algorithm gives a Mean Squared Error of 237.81 and the R2 Score of 0.722, which is definitely nowhere near acceptable standards with respect to the requirements in the field.

1.3 Deep Model Architectures

1.3.1 Recurrent Neural Network (RNN)

With limited success of traditional Machine Learning methods, various Deep Learning architectures are explored. Drawing parallels with the functioning of neurons in the brain, Artificial Neural Networks (ANN) consists of a number of neurons stacked on each layer. The output of each layer is passed through as input into the next layer, using some non-linear activation function. The weights on each layer are adjusted and learnt using back-propagation. A Recurrent Neural-Network (RNN) is a subdomain of ANN, which specializes in training and performance of time-series data. Time-series data are a set of mathematical values obtained at equally spaced time intervals and thus along with the various input parameters, are also a function of time. An RNN model takes a contiguous series of values in input to predict the output.

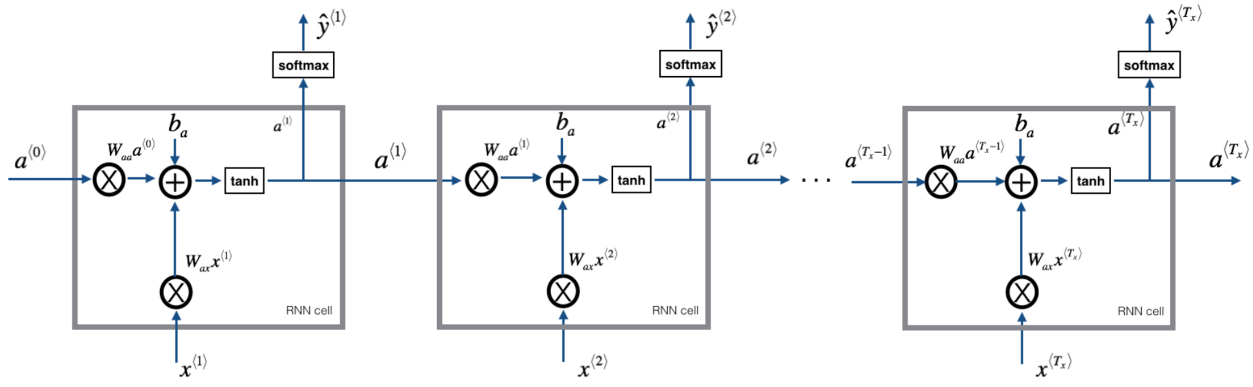


Figure 3: RNN Architecture

In the figure attached above, it can be inferred that to predict the output at an instant $y^{<Tx>}$ is not only dependent on input $x^{<Tx>}$ but also on values less than T_x, like $x^{<1>}$ and $x^{<2>}$ in this case (upto a specific window width), connected to the model with the help of activation state $a_{<Tx-1>}$. This property of RNN helps to introduce the ability to recognize and understand patterns in the network. The ability to inherently introduce a new parameter of time dependency in the model is what makes RNN an excellent base for predicting stock prices. An RNN model learns its parameters with the help of Backpropagation through time (BPTT). However, as can be intuitively guessed, the introduction of sequential modeling introduces a large set of parameters, which makes it difficult for a simple RNN model to understand the intricacies of long-term pattern-matching. This is due to the phenomenon of vanishing/exploding gradients, and thus restricts the usage of RNN to only short-range pattern recognition. However, further modifications in the RNN architecture have made it possible to identify even long-term dependencies in a given time-series data and are thus used as fundamental blocks for prediction of stock prices.

1.3.2 Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) is an improvement over the standard RNN architecture, which specializes in the handling of long-term dependencies in a time-series dataset by addressing the problem of exploding/ vanishing gradient. It does so with the help of 3 gates - an update gate (also called the input gate in some papers), a forget gate and the output gate. The forget gate is the gate responsible for deciding which information is important enough to be retained, with the help of a sigmoid function.

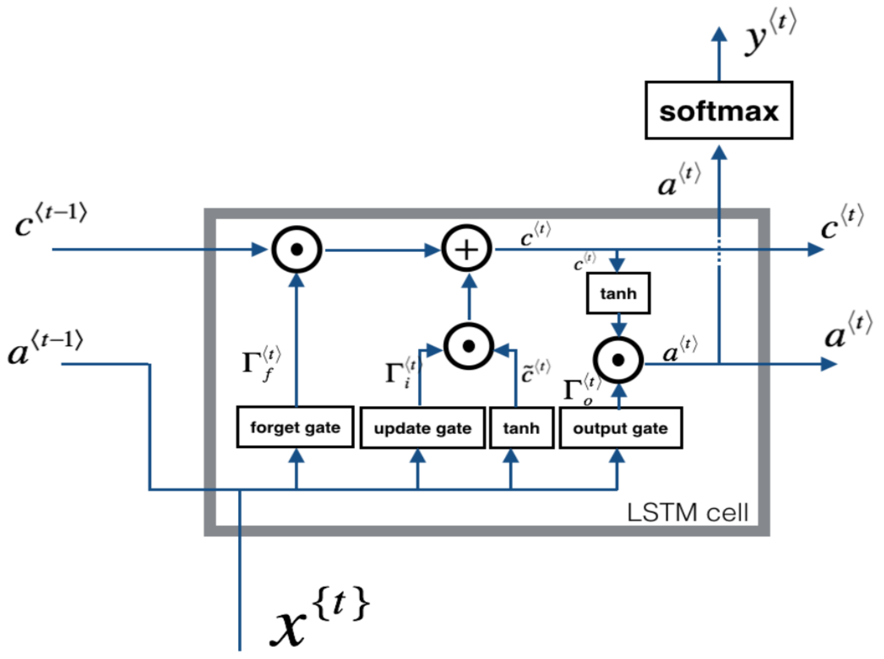


Figure 4: LSTM Architecture

As can be seen from the figure illustrated above, each cell in an LSTM contains an extra input parameter $c^{<t-1>}$ known as cell state, in addition to the activation state $a^{<t-1>}$. It is the cell state of $c^{<t>}$ that determines which information is to be remembered.

The equations governing the operations in LSTM are listed below: -

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[a_{t-1}, x_t] + b_u)$$

$$G_f = \sigma(W_f[a_{t-1}, x_t] + b_f)$$

$$G_o = \sigma(W_o[a_{t-1}, x_t] + b_o)$$

$$c_t = G_u * \tilde{c}_t + G_f * c_{t-1}$$

$$a_t = G_o * tanh(c_t)$$

Figure 5: LSTM Update Operations

1.3.3 Gated Recurrent Unit (GRU)

Gated Recurrent Units (GRU), like LSTM, is an improvement over the original basic RNN-architecture used to tackle the issue of vanishing/ exploding gradients so that long-term dependencies in the dataset can be learned and considered while predicting the output. However, unlike LSTM which uses 3 gates, a GRU cell uses only 2 gates i.e. an update gate and a reset gate. The update gate decides which information to retain, while the reset gate decides which information to forget.

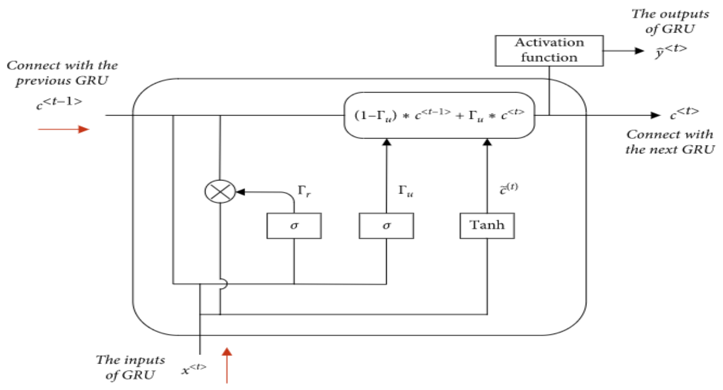


Figure 6: GRU Architecture

As can be seen in the figure as well, GRU takes only the current state $x^{<t>}$ and the cell state $c^{<t-1>}$ as input, and does not include the activation state $a^{<t-1>}$ like LSTM. This implies GRU requires less number of parameters per cell to learn and predict the output as compared to LSTM, making it light-weighted and easier to train. The performance comparison of LSTM and GRU is comparable and completely case dependent. The Mathematical equations governing the operations in GRU are listed below: -

$$\tilde{c}_t = \tanh(W_c[G_r * c_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[c_{t-1}, x_t] + b_u)$$

$$G_r = \sigma(W_r[c_{t-1}, x_t] + b_r)$$

$$c_t = G_u * \tilde{c}_t + (1 - G_u) * c_{t-1}$$

$$a_t = c_t$$

Figure 7: GRU Update Operations

1.3.4 Convolutional Neural Network (CNN)

Convolutional Neural Networks or CNNs are Neural Networks containing convolutional layers, and are the de facto models for image processing requirements. However, their use cases extend far beyond and could also be applied to predict time series data accurately. A CNN picks out the most significant features in the input. Hence it is commonly used in feature engineering. CNNs are very efficient as they make use of weight sharing, which significantly minimizes the number of parameters. The convolutional layer and the pooling layer are the two layers that make up a Convolutional Neural Network. Initially, convolution is performed on the input with the help of a filter, which extracts the most significant features. Next, this data is passed through a pooling layer to ensure the input dimensions do not grow exponentially, thus applying a kind of filter to the output extracted from the convolutional layer.

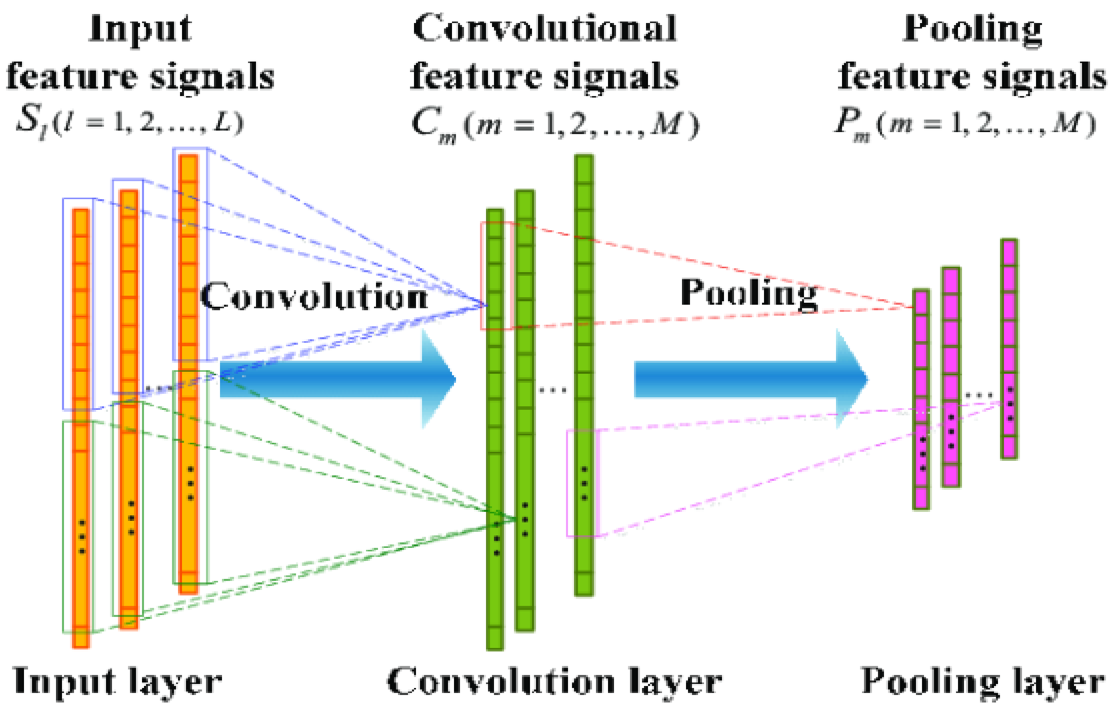
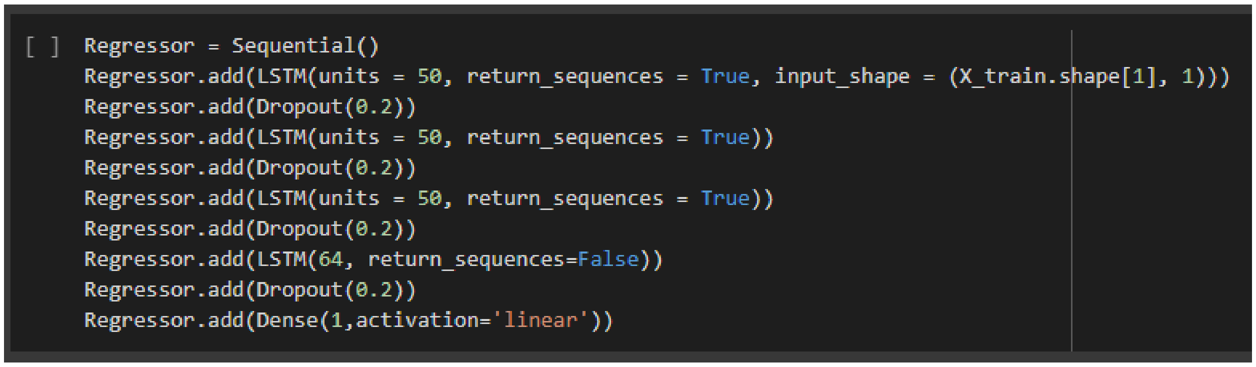

Input feature signals
$S_l (l = 1, 2, \ldots, L)$
Convolutional feature signals
$C_m (m = 1, 2, \ldots, M)$
Pooling feature signals
$P_m (m = 1, 2, \ldots, M)$
Convolution
Pooling
Input layer
Convolution layer
Pooling layer

Figure 8: Working of CNN

Chapter 2. Methodology And Results

2.1 Method 1 : Using stacked LSTM

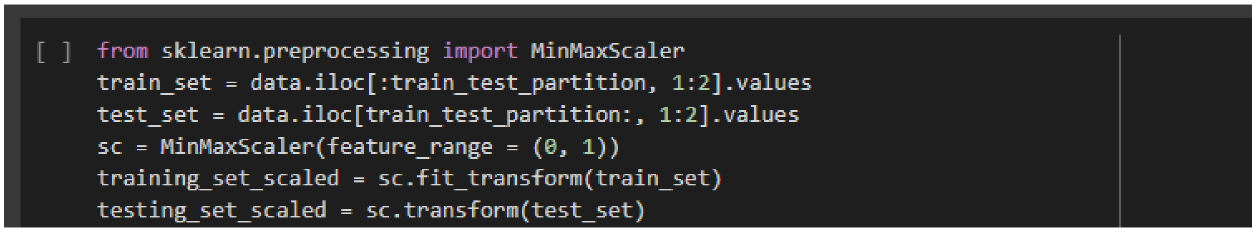
The first deep-learning model is the one designed using stacked LSTMs. We have used 4 layers of LSTM networks stacked one upon another and followed it up with a dense layer to map correctly the dimensionality of the input to the output.



```
[ ]  Regressor = Sequential()
     Regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
     Regressor.add(Dropout(0.2))
     Regressor.add(LSTM(units = 50, return_sequences = True))
     Regressor.add(Dropout(0.2))
     Regressor.add(LSTM(units = 50, return_sequences = True))
     Regressor.add(Dropout(0.2))
     Regressor.add(LSTM(64, return_sequences=False))
     Regressor.add(Dropout(0.2))
     Regressor.add(Dense(1,activation='linear'))
```

Figure 9: Stacked LSTM code

Data from the real world is very inconsistent, sometimes incomplete, and has some noise. Therefore it needs to be pre-processed to make maximum use of it. It is the first crucial step in a machine learning application. The technique we are going to use for preprocessing is data scaling, this technique scales down numerical features within a specific range. To utilize this functionality we are going to use the MinMaxScaler class.



```
[ ]  from sklearn.preprocessing import MinMaxScaler
     train_set = data.iloc[:train_test_partition, 1:2].values
     test_set = data.iloc[train_test_partition:, 1:2].values
     sc = MinMaxScaler(feature_range = (0, 1))
     training_set_scaled = sc.fit_transform(train_set)
     testing_set_scaled = sc.transform(test_set)
```

Figure 10: Data Preprocessing

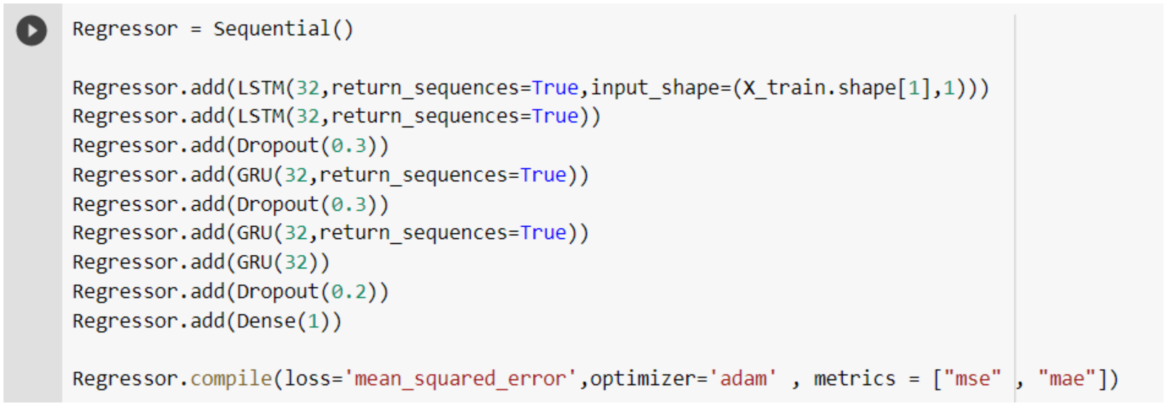
Out of all the things that can go wrong in any ML model, overfitting is one of the most common and most frequent errors. To overcome overfitting we have various techniques like data augmentation, feature selection, cross-validation, regularization, dropout layer, etc. Regularization is one of them which applies a penalty to the input features with heavy coefficients which helps in reducing the model variance and model complexity. To reduce the complexity of the model, some number of layer outputs are randomly ignored, or “dropped out” is used. In this case, 0.2 dropout is used as it is the empirical solution for our model.

The aim of the deep learning model is to minimize a loss function, which in our case is the Mean-squared error (MSE). This loss is minimized through gradient descent. For optimization purposes, we are going to use Adam Optimizer. Other optimizers of gradient descent, Stochastic Gradient Descent (SGD), and Mini-Batch Gradient Descent don't do well in deep learning, because for each weight the learning parameter is the same. Here comes Adaptive Gradient Descent (AdaGrad), which is a technique to provide adaptive gradients to different weights. AdaGrad doesn't do well on the dense dataset, so here Adam Optimizer came into the picture, it is a momentum estimation algorithm. Adam combines the power of AdaGrad, which works well

for sparse gradients and RMSprop, which has excellent online setting characteristics to converge as per expected theoretical characteristics.
Another important aspect that drastically affects the performance of the model is hyperparameter tuning. Hyperparameters are parameters in deep learning that one decides and sets before training begins and whose values remain the same over the period of training. In deep learning models, the common hyperparameters involved are train-test split ratio, optimizer algorithm, activation function in neural networks, choice of activation function, number of hidden layers, number of activation units in each layer, dropout layer, bach size, epoch. The values of hyperparameters are determined with the help of RandomSearch class of Keras tuner library, available in tensorflow framework. Our stacked LSTM model is found to work best with a batch-size of 64 and 15 epochs.

2.2 Method 2 : Using LSTM-GRU

The second deep-learning model designed is basically an improvement over the stacked-LSTM model used earlier, by introduction of GRU units in some layers. For the LSTM-GRU model, 2 layers of 32 LSTM cells have been stacked one upon another, whose output is fed to 2 GRU layers. The output, much like the first case, is obtained from the final layer by applying a dense layer for dimension consistency. Steps taken to encounter regularization are similar to that taken in the first method, by inserting dropout after each layer. Learning, as is the case with most deep-learning models, is processed through Adam with the aim of minimizing the MSE loss function, and hyperparameters are tuned using RandomSearch. The layers of GRU and LSTM have been empirically found out and this configuration produces the best result.


```
Regressor = Sequential()

Regressor.add(LSTM(32,return_sequences=True,input_shape=(X_train.shape[1],1)))
Regressor.add(LSTM(32,return_sequences=True))
Regressor.add(Dropout(0.3))
Regressor.add(GRU(32,return_sequences=True))
Regressor.add(Dropout(0.3))
Regressor.add(GRU(32,return_sequences=True))
Regressor.add(GRU(32))
Regressor.add(Dropout(0.2))
Regressor.add(Dense(1))

Regressor.compile(loss='mean_squared_error',optimizer='adam' , metrics = ["mse" , "mae"])
```
Figure 11: LSTM-GRU code

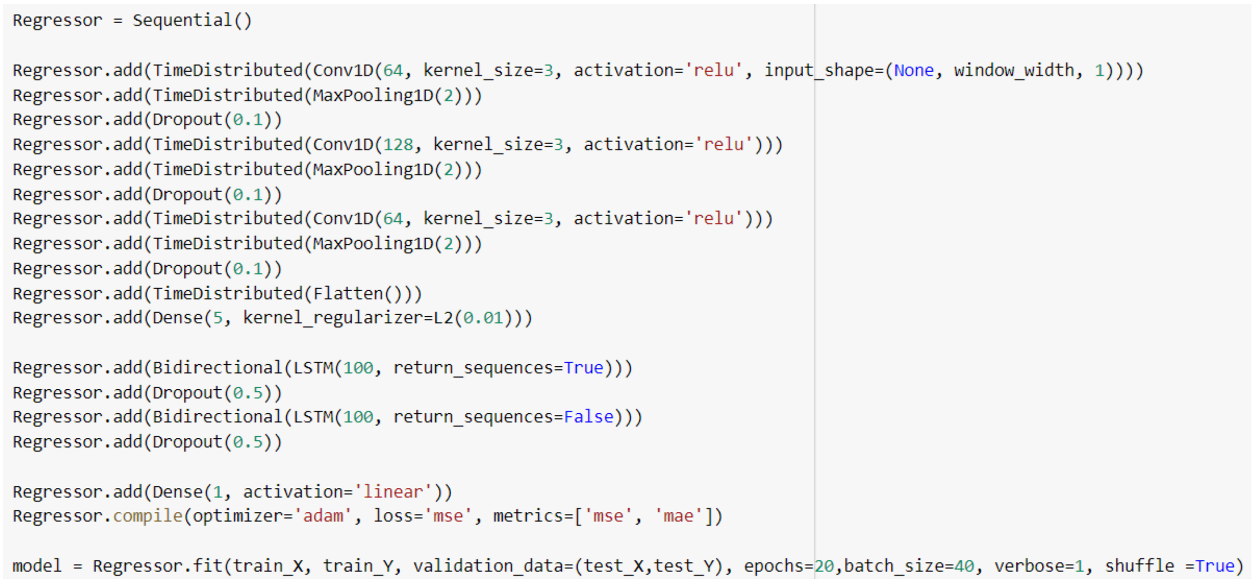
2.3 Method 3 : Using CNN-BiLSTM

The third deep-learning model used explores a completely new domain - using CNN and bidirectional LSTM for stock price prediction. CNN, although overwhelmingly used in the fields of image-processing, can also be used in the fields of time-series forecasting as a 1D convolution on a time series data roughly computes the weighted moving average and applies filters to the sequential data, which can then be inferred to predict the trend. The powerful feature-extraction properties of CNN from not only past, but also future values, together combined by a bidirectional LSTM makes a powerful DL model.

Initially, the data has been scaled with respect to first-element in the window, so that the input values remain closer to zero which is beneficial for CNN layers. Thus the final predicted value is again unscaled with inverse transformation for getting the actual predicted value of the stock price.

The sequential model consists of 3 2D CNN layers with 64, 128 and 64 filters respectively. Rectified Linear(ReLu) activation fuction is used in all three CNN layers. Each CNN layer is stacked with adjacent MaxPooling layer for downscalng the processed data for extracting the most important features. Dropout layers are introduced with each CNN layer to improve regularization.

The processed data is thus fed to 2 layers of Bidirectional LSTM with 100 units each for extraction of past as well as future trends. All the layers before BiLSTM are timedistributed for introducing the dimension of time, required for BiLSTM layers. Ultimately, a Dense layer with linear activation is used to calculate the final prediction.



```
Regressor = Sequential()

Regressor.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu', input_shape=(None, window_width, 1))))
Regressor.add(TimeDistributed(MaxPooling1D(2)))
Regressor.add(Dropout(0.1))
Regressor.add(TimeDistributed(Conv1D(128, kernel_size=3, activation='relu')))
Regressor.add(TimeDistributed(MaxPooling1D(2)))
Regressor.add(Dropout(0.1))
Regressor.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu')))
Regressor.add(TimeDistributed(MaxPooling1D(2)))
Regressor.add(Dropout(0.1))
Regressor.add(TimeDistributed(Flatten()))
Regressor.add(Dense(5, kernel_regularizer=L2(0.01)))

Regressor.add(Bidirectional(LSTM(100, return_sequences=True)))
Regressor.add(Dropout(0.5))
Regressor.add(Bidirectional(LSTM(100, return_sequences=False)))
Regressor.add(Dropout(0.5))

Regressor.add(Dense(1, activation='linear'))
Regressor.compile(optimizer='adam', loss='mse', metrics=['mse', 'mae'])

model = Regressor.fit(train_X, train_Y, validation_data=(test_X,test_Y), epochs=20,batch_size=40, verbose=1, shuffle =True)
```

Figure 12: CNN-BiLSTM Code

Chapter 3. Results

The stacked-LSTM model predicts the data with a Mean Square Error (MSE) of 76.88 and an R^2 score of 0.931, an exponential improvement from the XgBoost model, with same quantum of training data.

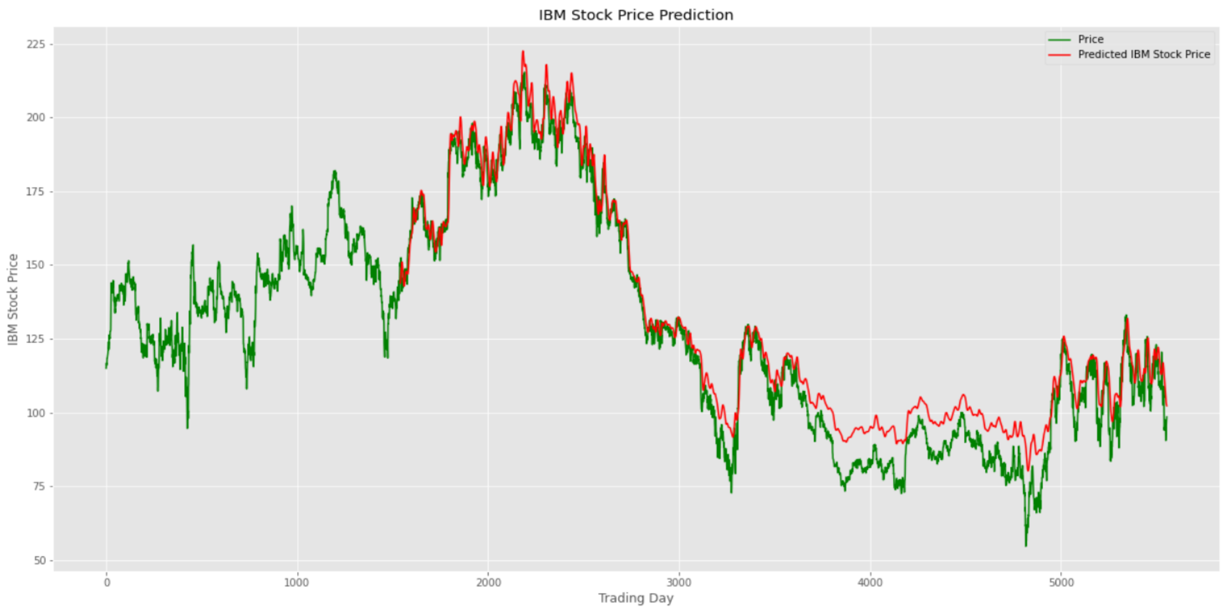


Figure 13: Results using Stacked-LSTM Model

Using the bottom 2 layers of GRU instead of LSTM further improved the various performance metrics - MSE improving to 33.42 and R^2 score of 0.975.

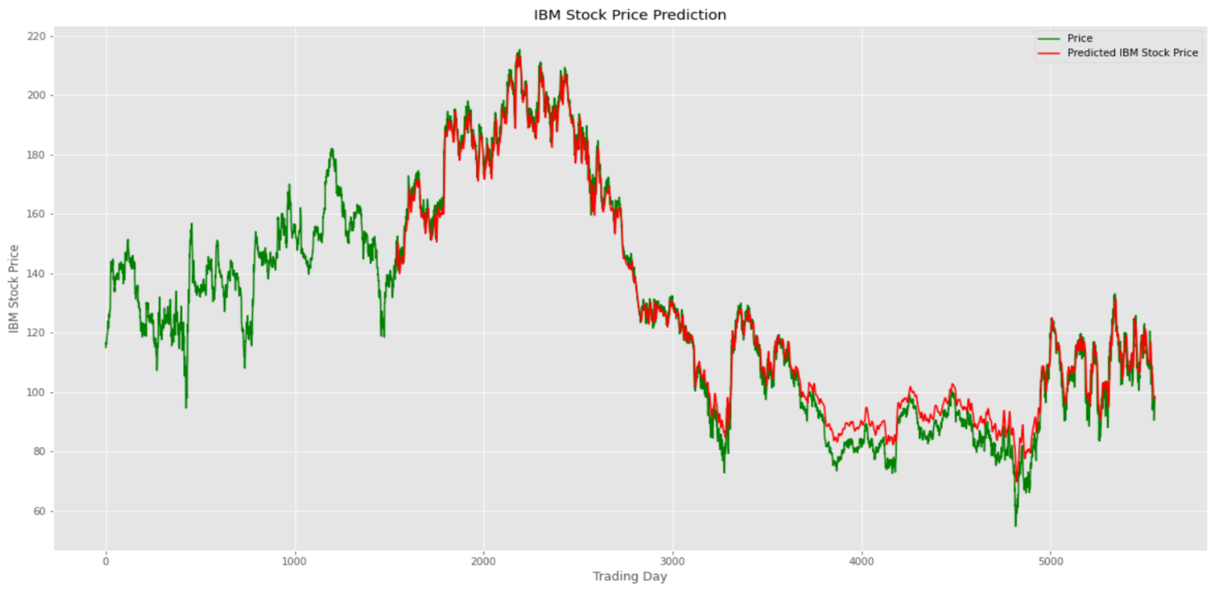


Figure 14: Results using LSTM-GRU Model

The third model, which used CNN-BiLSTM for predicting stock prices from not only past, but also future trends performed the best on our dataset, with an MSE of only 17.14 and R^2 score of 0.989.

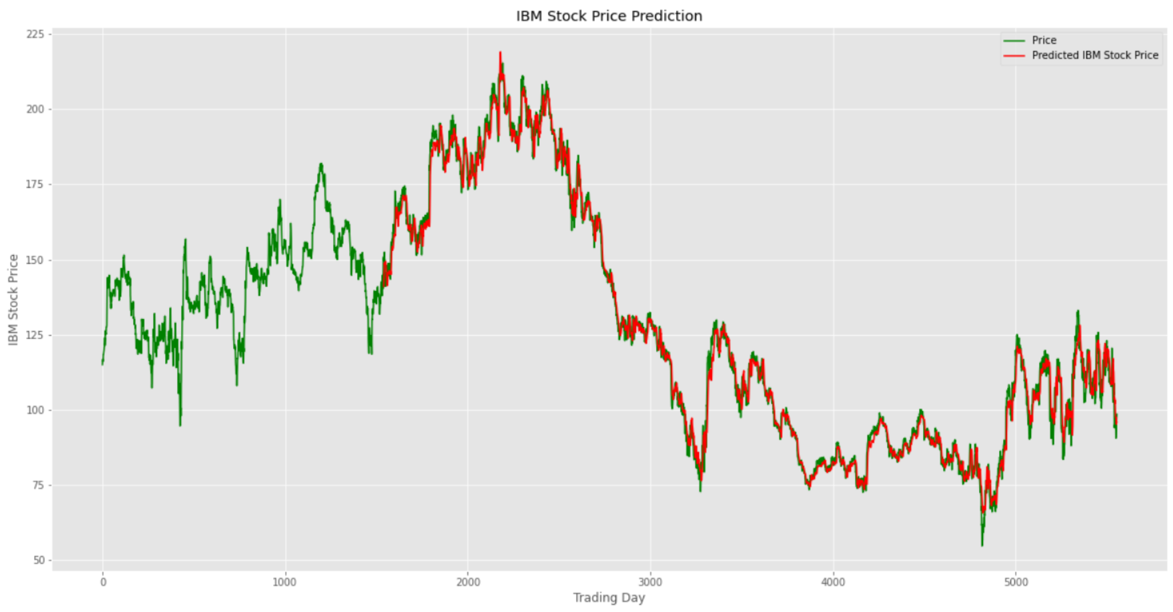


Figure 15: Results using CNN-BiLSTM Model

Chapter 4. Conclusion and Future Works

Stock-price forecasting, despite all its volatility and dependence on multiple parameters, can be modeled as a time-series data and predicted with an acceptable level of accuracy with the help of various deep learning architectures. For the IBM dataset, the best results were obtained by combining the power of feature extracting architectures like CNN with the power of strong long-range pattern-detecting architectures on time-series data like LSTM. These models comprehensively outperform the traditional Machine Learning models like XgBoost.

However, despite all these successes, the model still lacks on certain fronts and can be a topic of future research. Neither of the three models deployed includes the parameters of qualitative factors such as national policy and news, and its effect on increasing/ decreasing stock prices. One area of future research, thus can be to explore the field of sentiment analysis and integrate the findings found to the model, so as to further improve the accuracy of the model.

Chapter 5. References

[1] Selvin, Sreelekshmy & Ravi, Vinayakumar & Gopalakrishnan, E. A & Menon, Vijay & Kp, Soman. (2017). Stock price prediction using LSTM, RNN and CNN-sliding window model. 1643-1647. 10.1109/ICACCI.2017.8126078.

[2] S. O. Ojo, P. A. Owolawi, M. Mphahlele and J. A. Adisa, "Stock Market Behaviour Prediction using Stacked LSTM Networks," 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC), 2019, pp. 1-5, doi: 10.1109/IMITEC45504.2019.9015840.

[3] Lu, Wenjie & Li, Jiazheng & Li, Yifan & Sun, Aijun & Wang, Jingyang. (2020). A CNN-LSTM-based model to forecast stock prices. Complexity. 2020. 1-10. 10.1155/2020/6622927.

[4] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[5] Dey, Rahul & Salem, Fathi. (2017). Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks.

[6] Keras Tuner Documentation.

[7] Recurrent Neural Networks for Prediction : Learning Algorithms, Architectures And Stability - by Danilo Mandic and Jonathan A. Chambers.