25-05-2022

ALGORITHMIC TRADING

CMRCET B.Tech 2018-22 CSE

ABSTRACT

Algorithmic trading is a method of executing orders using automated pre-programmed trading instructions for the things such as time and price. This type of trading attempts to boost the speed and computational resources of computers compared to human traders. These encompass a variety of trading strategies, which are based on formulas and results from mathematical finance, and often rely on specialized software. Stock market is a probabilistic trading where we can gain and have possibility of loss. Investing in stock market trading might be a risky task. So, we can use algorithmic trading which uses the earlier trends of a particular stock and help us predicting investing in the stock. Algorithmic trading using machine learning techniques to increase the probability of profit because it uses the technical analysis of stock, price action strategies, seasonal trends and help us to predict which time is better to invest in stocks. So, we will not fall in debts anymore. We are developing an algorithm using machine learning and deep learning techniques like SMA (Simple Moving Average), Arima, RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory) for predictions.

1. INTRODUCTION

1.1 Motivation for work

Stock price prediction is an attractive area where we gain more profits to develop the business. With a successful model for stock prediction, we can gain insight about market behaviour over time. Machine learning technique will be an efficient method to solve the problem of predicting stock movements.

1.2 Objective

To design efficient system for predicting the stock market movement using ML. Specifically, use LSTM method to Predict stock price based on the historical data. Compare existing methods to show the superiority of LSTM.

1.3 Problem statement

Stock prices are volatile in nature and price depends on a range of factors. The main aim of this project is to predict stock prices using Long short-term memory (LSTM). Predict stock market movement using machine learning techniques to improve the profit percentage in the trading

1.4 Scope & limitations

1. We predict the stock market up to one month.

2. It is hard to predict in the pandemic situations

2. RELATED WORK

2.1 Performance matrix formulas

MSE = $\frac{1}{n}\sum_{i=1}^{n}(yi-pi)^2$

MSE = mean squared error

n = number of data points

Yi = observed values

Pi = predicted values

RMSE= $\sqrt{\frac{1}{n}}\sum_{i=1}^{n}(yi-pi)^2$

RMSD = root mean square deviation

$\boldsymbol{R^2}$ = 1-RSS/TSS

R^2 = coefficient of determination

RSS = sum of squares of residuals

TSS = total sum of squares

2.2 Simple moving average (SMA) [1]

A simple moving average (SMA) is an arithmetic moving average calculated by adding recent prices and then dividing that figure by the number of time periods in the calculation average.

Formula:

SMA = (A1+A2+⋯+An) / n

An= The price of an asset

n = The number of total periods

For example, this is how you would calculate the simple moving average with the following closing prices over a 15-day period.

Week One (5 days): 20, 22, 24, 25, 23

Week Two (5 days): 26, 28, 26, 29, 27

Week Three (5 days): 28, 30, 27, 29, 28

2.2.1 SMA implementation results

The platform we used is Google colab.

We have implemented SMA for the dates between 08-2018 and 07-2019. We have tested SMA for different stocks datasets like google, amazon, apple, tesla

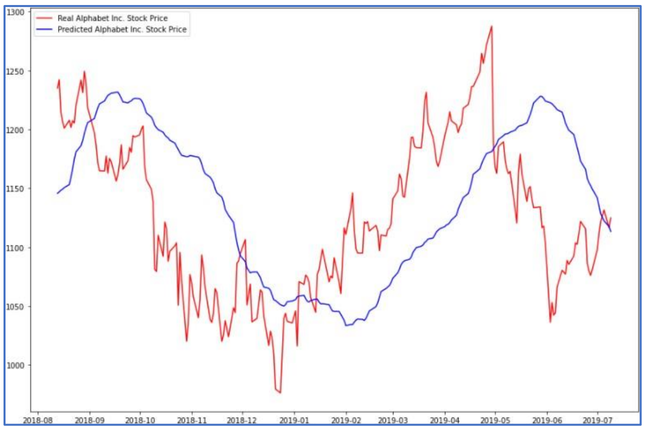

Fig 2.1: SMA-Google dataset

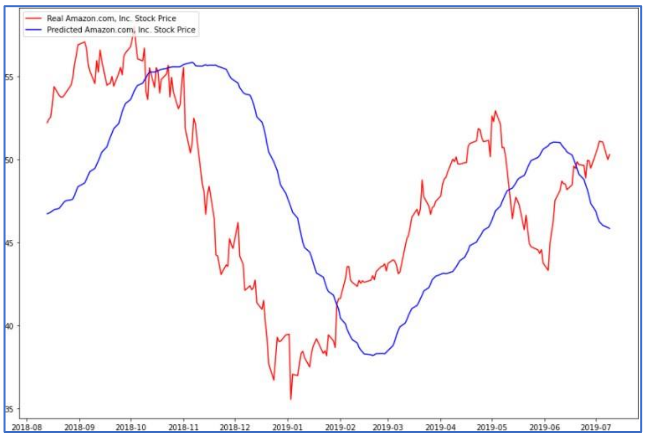

Fig 2.2: SMA-Amazon dataset

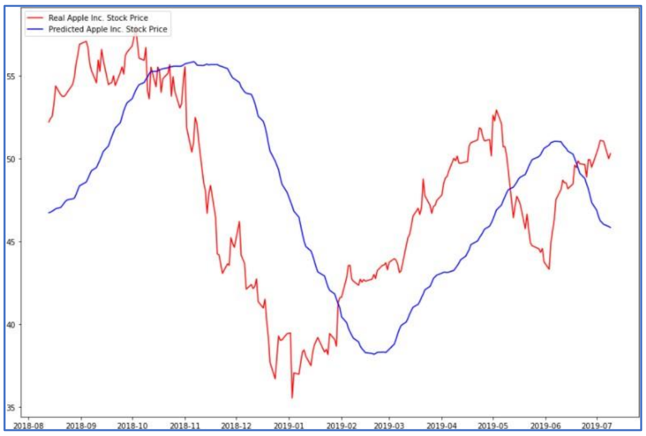

Fig 2.3: SMA-Apple dataset

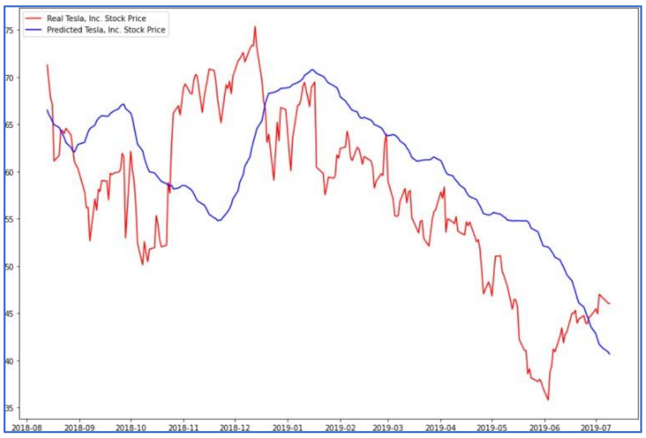

Fig 2.4: SMA-Tesla dataset

2.2.2 SMA performance metrics

Stock	MSE	RMSE	R^2
TSLA	34719.06597	186.33052	-1.78882
GOOG	19929.37611	141.17144	-0.07731
AAPL	172.47748	13.13306	-1.23221
AMZN	61294.70557	247.57767	-0.14051

Table 2.1: SMA performance metrics

NOTE: MSE and RMSE values are extremely high and R^2 is too low, it denotes that this model is not preferable for satisfactory results

2.3 Auto regressive integrated moving average (ARIMA) [2]

This acronym is descriptive, capturing the key aspects of the model itself.

Briefly, they are:

AR: Autoregression. A model that uses the dependent relationship between an observation and number of lagged observations.

I: Integrated. Differencing of raw observations (e.g., subtracting an observation from an observation at the previous time step) to make the time series stationary.

MA: Moving Average. It uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components specifies in the model as a parameter. A standard notation ARIMA (p, d, q) where the parameters are substitutes with integer values to quickly indicate the specific ARIMA model. Parameters of the ARIMA model are as follows:

p: Number of lag observations included in the model, also called the lag order.

d: Number of times the raw observations differenced, also called the degree of differencing.

q: The size of the moving average window, also called the order of moving average.

ARIMA Highlights

ARIMA models are known to be robust and efficient in financial time series forecasting

It constantly outperformed complex structural models in short-term prediction

But they work only for a particular time series data

It is a linear model that is it cannot manage nonlinear behaviour of stock market

2.3.1 ARIMA implementation results

We have implemented ARIMA for the dates between 11-2021 and 03-2022. We evaluated ARIMA for different stocks datasets like google, amazon, apple, tesla

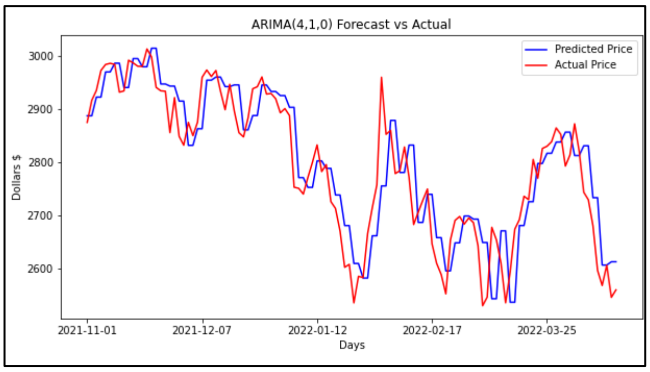

Fig 2.5: ARIMA-Google dataset

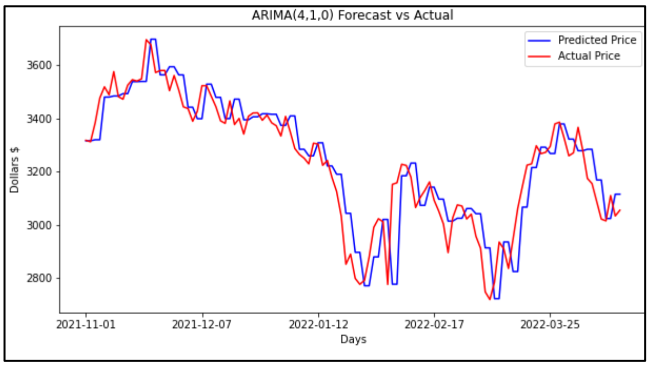

Fig 2.6: ARIMA-Amazon dataset

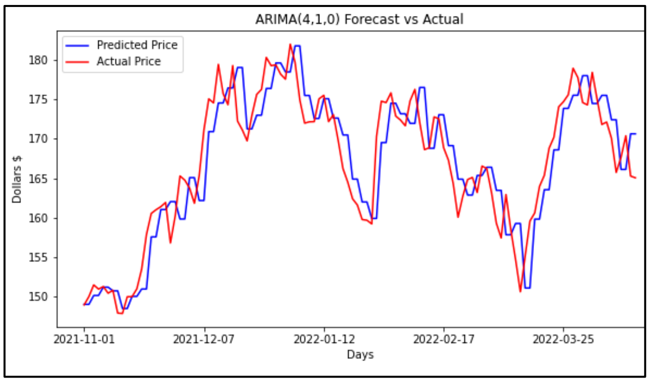

Fig 2.7: ARIMA-Apple dataset

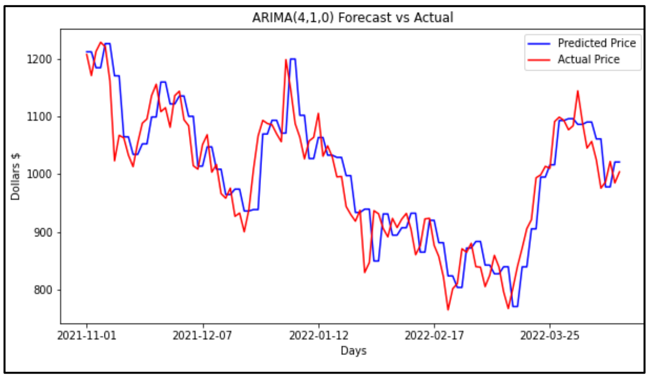

Fig 2.8: ARIMA-Tesla dataset

2.3.2 ARIMA performance metrics

Stock	MSE	RMSE	R^2
TSLA	3773.97858	61.43271	0.79599
GOOG	2398.89653	48.97853	0.80730
AAPL	14.24036	3.77364	0.81571
AMZN	9812.72205	99.05918	0.81741

Table 2.2: ARIMA performance metrics

NOTE: MSE and RMSE values are extremely high and R^2 is high, it denotes that this model is slightly preferable for satisfactory results

2.4 Recurrent neural networks (RNN) [3]

Recurrent neural networks (RNN) are a class of neural networks that are helpful in modelling sequence data. Derived from feedforward networks, exhibit similar behaviour to how human brains function. Simply put recurrent neural networks produce predictive results in sequential data that other algorithm cannot. RNNs are a powerful and robust type of neural network and belong to the most promising algorithms in use because it is the only one with an internal memory.

Like other deep learning algorithms, recurrent neural networks are old. They were initially created in the 1980's, but only in recent years have we seen their true potential. An increase in computational power along with the massive amounts of data that we now must work with, and the invention of long short-term memory (LSTM) in the 1990s, has really brought RNNs to the foreground.

Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what is coming next. Therefore, they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

In a feed-forward neural network, the information only moves in one direction from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice. Feed-forward neural networks have no memory of the input they receive and are bad at predicting what is coming next. Because a feed-forward network only considers the current input, it has no notion of order in time. It simply cannot remember anything about what happened in the past except its training

In a RNN the information cycles through a loop. When it decides, it considers the current input and what it has learned from the inputs it received previously

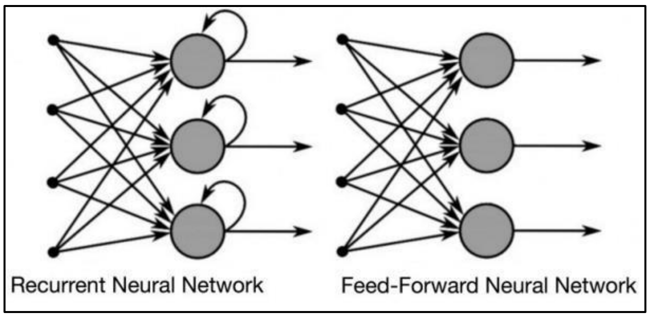

Fig 2.9: Difference between Feed Forward and RNN **[3]**

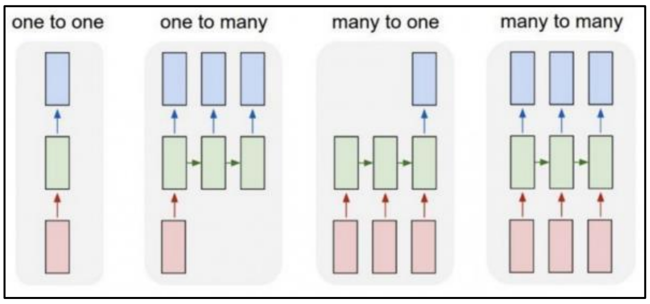

Fig 2.10: RNN mapping inputs to outputs **[3]**

2.4.1 RNN implementation results

We have implemented RNN for dates between 08-2018 and 07-2019. We have evaluated RNN for different stocks datasets like google, amazon, apple, tesla.

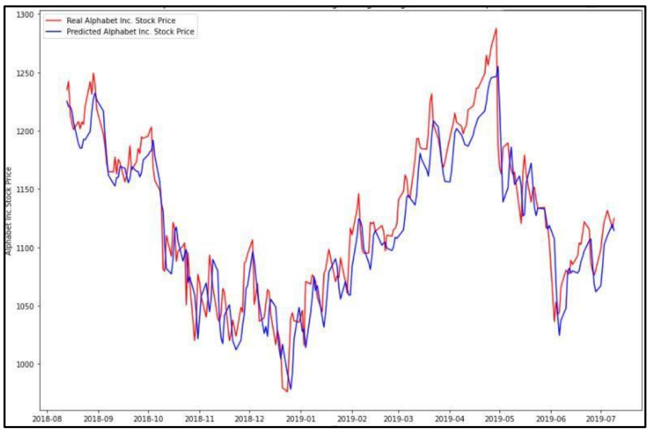

Fig 2.11: RNN-Google dataset

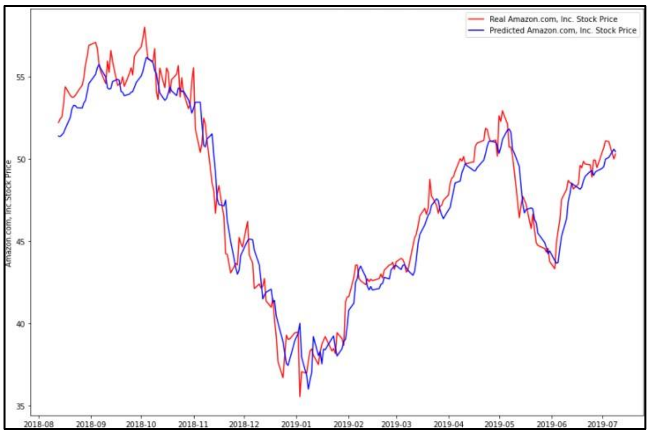

Fig 2.12: RNN-Amazon dataset

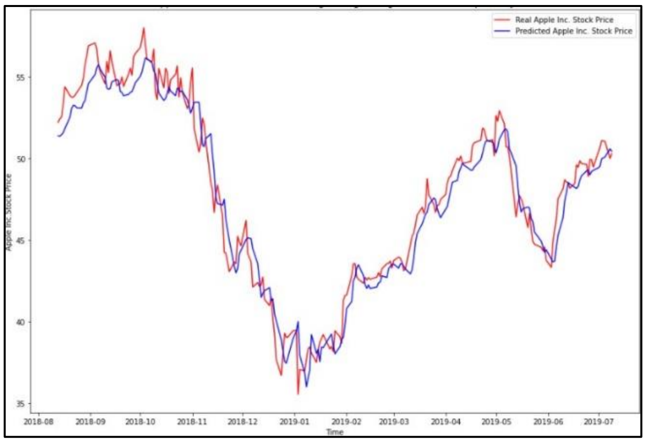

Fig 2.13: RNN-Apple dataset

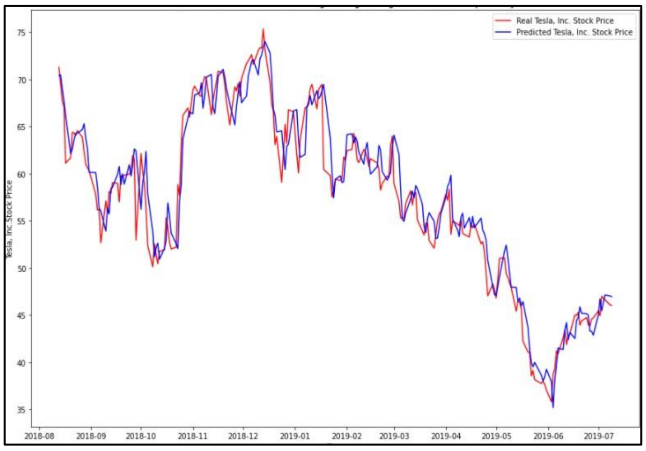

Fig 2.13: RNN-Tesla dataset

2.3.2 RNN performance metrics

Stock	MSE	RMSE	R^2
TSLA	804.60987	32.95856	0.77471
GOOG	522.30054	20.86819	0.72851
AAPL	18.03839	4.24716	0.76654
AMZN	1791.87195	73.76498	0.83640

Table 2.3: RNN performance metrics

NOTE: MSE and RMSE values are high and R^2 is high, it denotes that this model is slightly preferable for satisfactory results

3. PROPOSED SYSTEM

Long short-term memory (LSTM) **[5]** is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network can process not only single data points, but also entire sequences of data. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM model networks are well-suited to classifying, processing, and making predictions based on time series data, since there can be lags of unknown duration between notable events in a time series. LSTM deal with the vanishing gradient problem that encounters when training traditional RNNs.

RNN model can keep track of arbitrary long-term dependencies in the input sequences. The problem with RNNs is computational, when training a vanilla RNN using backpropagation, the long-term gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. RNNs using LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged.

3.1 Vanishing gradient problem

- Ø It occurs when training artificial neural networks with gradient-based learning methods
- Ø In such methods, during each iteration of training each of the neural network's weights receives an update
- Ø In some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value.
- Ø In the worst case, this may completely stop the neural network from further training.

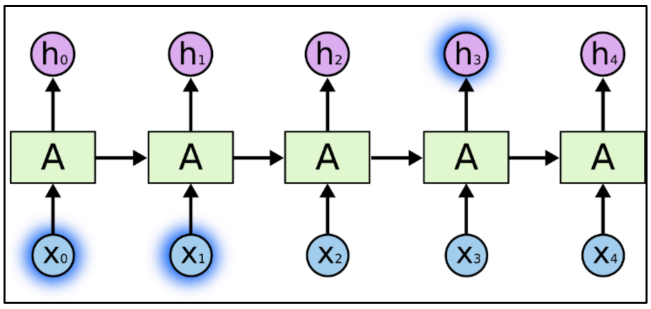

Fig 3.1: RNNs learn to use the past information **[3]**

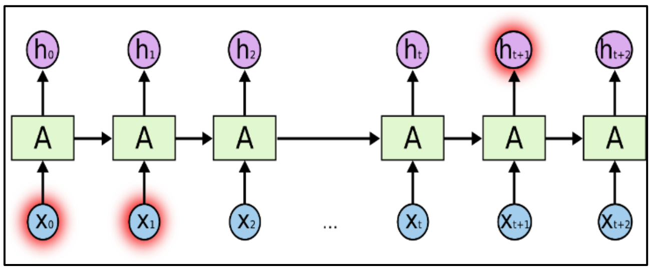
h_0 h_1 h_2 h_t h_{t+1} h_{t+2}
A A A A A A
X_0 X_1 X_2 ... X_t X_{t+1} X_{t+2}

Fig 3.2: When gap grows, RNN become unable to learn to connect the information. **[3]**

3.2 LSTM highlights

□ LSTM is an advanced type of RNN

□ Proficient in learning about long-term dependencies.

□ It solves the vanishing gradient problem

□ LSTM is useful on time-series and predictions.

□ LSTMs are good at processing sequences of data

3.3 LSTM model

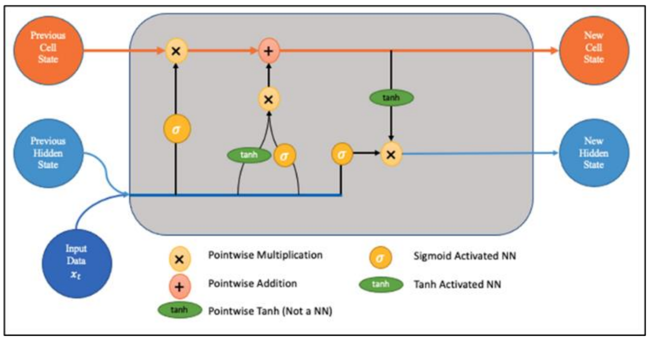

Fig 3.3: LSTM model **[5]**

The output of an LSTM at a particular point in time is dependent on three things:

- ○ The current long-term memory of the network known as the cell state
- ○ The output at the previous point in time known as the previous hidden state
- ○ The input data at the current time step

LSTM series of 'gates' control the information in a sequence of data. Three gates in LSTM are:

• forget gate,

• input gate and

• output gate.

3.3.1 Forget gate

Here we will decide which bits of the cell state (long term memory of the network) are useful given both the previous hidden state and new input data

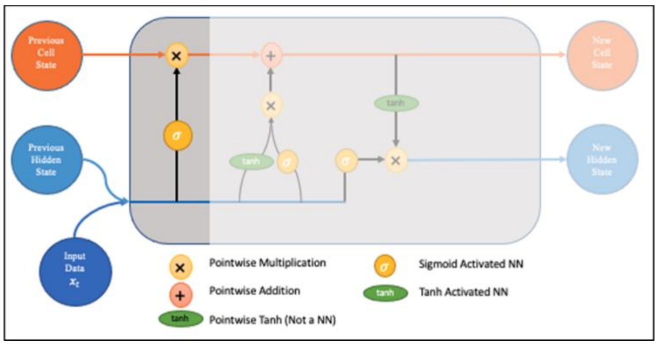

Fig 3.4: Forget gate **[5]**

The previous hidden state and the new input data are fed into a neural network. This network generates a vector where each element is in the interval [0,1] (ensured by using the sigmoid activation). This network (within the forget gate) is trained so that it outputs close to zero when a component of the input is deemed irrelevant and closer to 1 when relevant. It is useful to think of each element of this vector as a sort of filter/sieve which allows more information through as the value gets closer to one.

These output values have sent up and pointwise multiplied with the previous cell state. This pointwise multiplication means that components of the cell state which is irrelevant for the forget gate network will multiplies by a number close to zero and thus will have less influence on the following steps.

3.3.2 Input gate

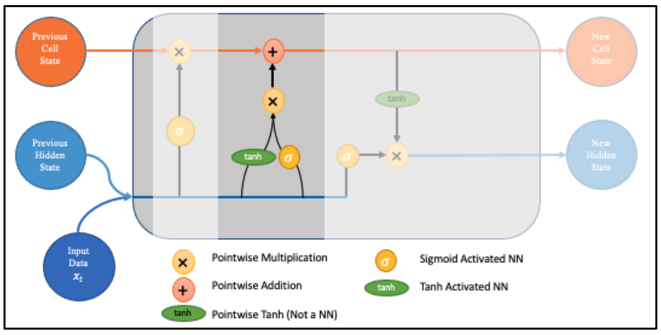

Fig 3.5: Input gate **[5]**

Both the new memory network and the input gate are neural networks in themselves, and both take the same inputs, the previous hidden state, and the new input data.

The new memory network is a tanh activated neural network which has learned how to combine the previous hidden state and new input data to generate a 'new memory update vector'. This vector contains information from the new input data given the context from the previous hidden state. This vector tells us how much to update each component of the long-term memory (cell state) of the network given the new data. We use a tanh here because its values lie in [-1,1] and so can be negative. The possibility of negative values here is necessary if we wish to reduce the impact of a component in the cell state.

However, in part 1 above, where we generate the new memory vector, there is a big problem, it does not check if the new input data is even worth remembering. This is where the input gate comes in. The input gate is a sigmoid activated network which acts as a filter, identifying which components of the 'new memory vector' are worth retaining. This network will output a vector of values in [0,1] (due to the sigmoid activation), allowing it to function as a filter through pointwise multiplication. Like what we saw in the forget gate, an output near zero is telling us we do not want to update that element of the cell state.

The output of parts 1 and 2 are pointwise multiplied. This causes the magnitude of added information we decided on in part 2 to be regulated and set to 0 if necessary. The resulting combined vector is then added to the cell state, resulting in the long-term memory of the network being updated.

3.3.3 Output gate

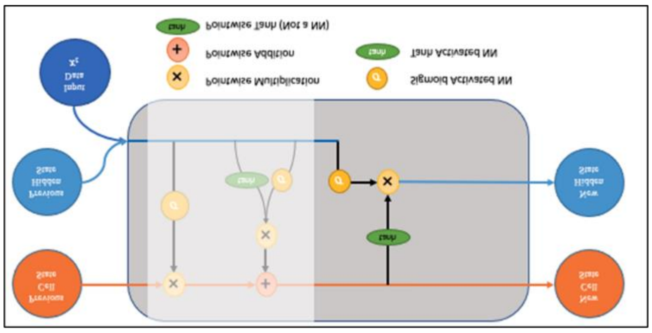

Fig 3.6: Output gate **[5]**

This ensures that only necessary information will be an output (saved to the new hidden state). However, before applying the filter, we pass the cell state through a tanh to force the values into the interval [-1,1].

- Apply the tanh function to the current cell state pointwise to obtain the squished cell state, which now lies in [-1,1].
- Pass the previous hidden state and current input data through the sigmoid activated neural network to obtain the filter vector.
- Apply this filter vector to the squished cell state by pointwise multiplication.
- Output the new hidden state

3.4 LSTM Applications [5]

1. Robot control
2. Time series prediction
3. Speech recognition
4. Grammar learning
5. Handwriting recognition
6. Human action recognition
7. Prediction in business process management
8. Prediction in medical care pathways
9. Semantic parsing
10. Market prediction

4. DESIGNING

- We reimplemented LSTM model to predict the stock market movement.
- We designed the LSTM model that it predicts the real-world stock instead of dummy data or manipulated data
- We designed our project to evaluate against the different real world stock market datasets.
- The LSTM model evaluates against different combinations of parameters to know the better combinations of parameters with which the model can predict the best stock movement
- We designed the project to help the investors to trade according to the stock movement predicted by the LSTM model
- Client can observe the future one-month stock and do better trading

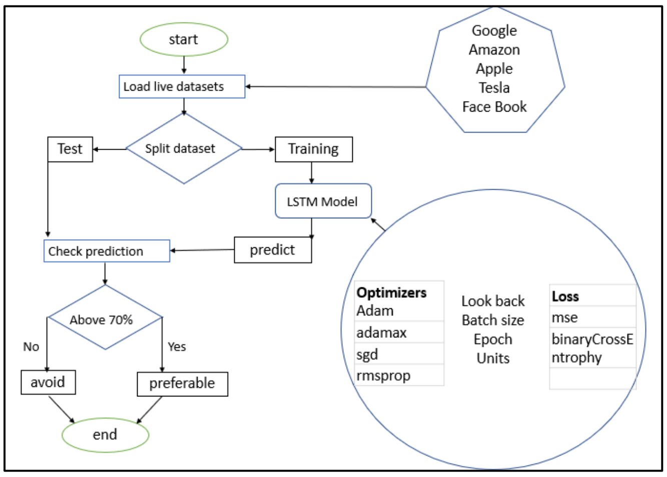

Fig 4.1: Project design

4.1 Design description

Load Live Datasets:

we are loading Google, Amazon, Apple, Tesla, Face Book real live datasets from yahoo website.

Split Dataset:

We have divided the dataset into two datasets

1. Training data set - 70%

2. Test data set - 30%

Model parameters:

We have considered different parameters

- Batch size
- Epoch
- Units
- **Different optimizers:**
  - adam
  - adamdelta
  - sgd
  - rmsprop
- **Different loss measures:**
  - mse
  - binaryCrossEntropy

Model prediction

We get prediction data as an output from the model

Check model performance

Compare the predicted data set with the test data set. Calculate the mode performance using performance matrix formulas

Result

The model with above 70% accurate performance is a preferable model. We should observe future one-month predicted data and do better trading. The model with below 70% accurate performance is a non-preferable model

5. RESULTS

5.1 Find best parameters combinations

5.1.1 Optimizers

5.1.1.1 Adam

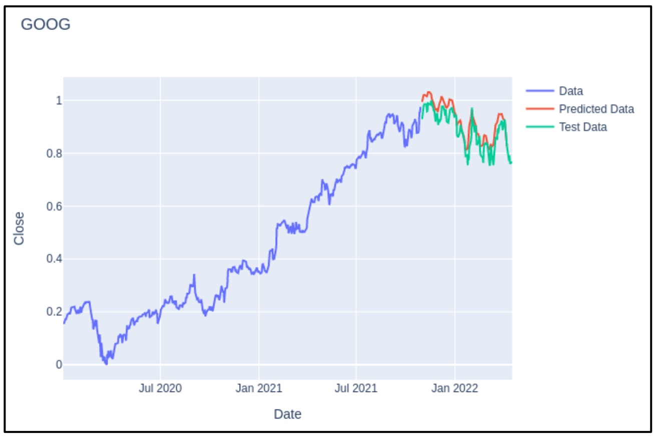

Fig 5.1: Adam optimizer for Google dataset

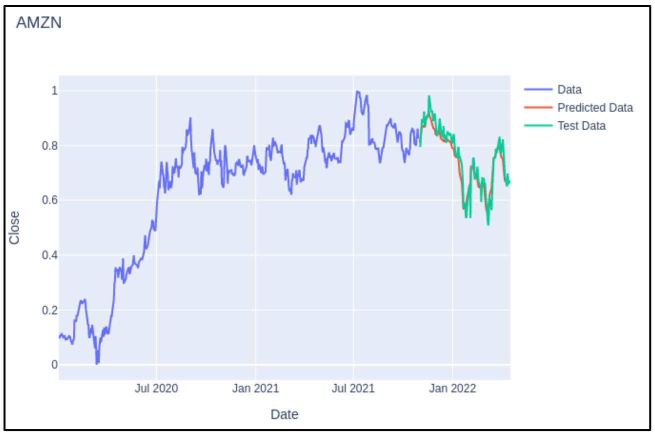

Fig 5.2: Adam optimizer for Amazon dataset

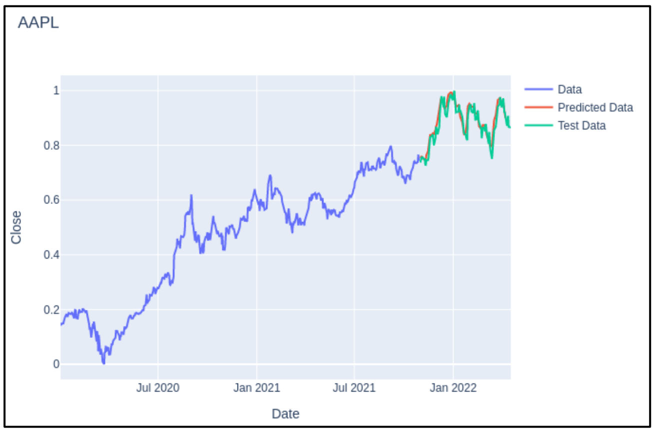

Fig 5.3: Adam optimizer for Apple dataset

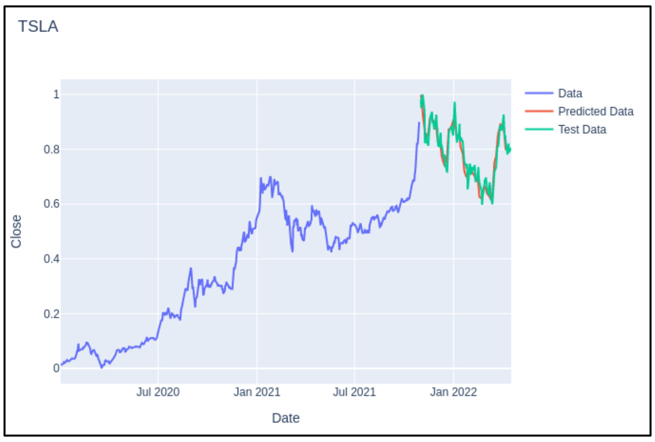

Fig 5.4: Adam optimizer for Tesla dataset

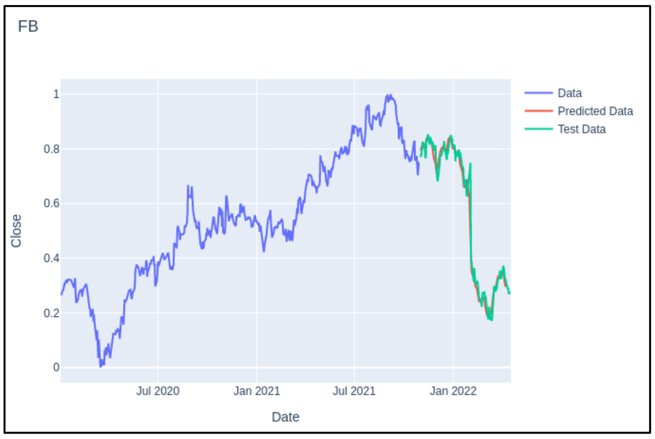

Fig 5.5: Adam optimizer for Facebook dataset

Stock	MSE	RMSE	R^2
GOOG	0.00174	0.04174	0.60804
TSLA	0.00107	0.03274	0.88945
AAPL	0.00059	0.02445	0.88233
AMZN	0.00133	0.03655	0.89638
FB	0.00163	0.04044	0.97316

Table 5.1: Adam optimizer performance metrics

Note: MSE and RMSE values are low and R^2 is values are high, it denotes that Adam optimizer is preferable for satisfactory results

5.1.1.2 Rmsprop

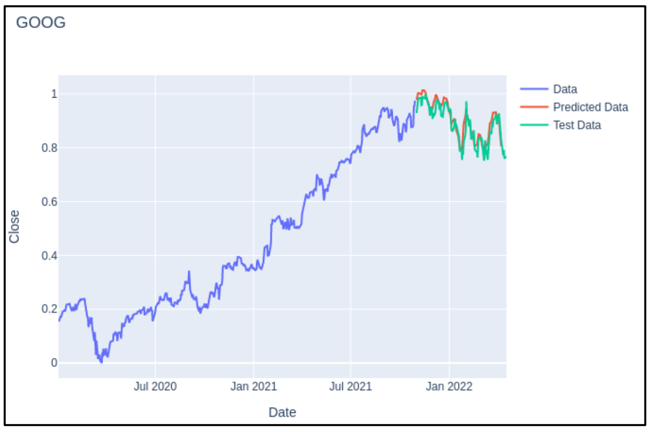

Fig 5.6: Rmsprop optimizer for Google dataset

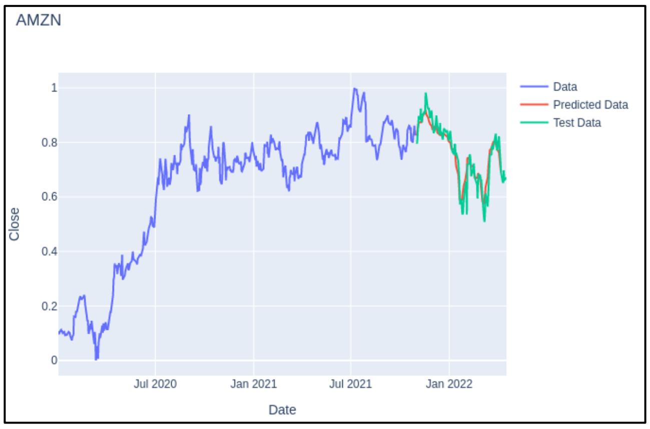

Fig 5.7: Rmsprop optimizer for Amazon dataset

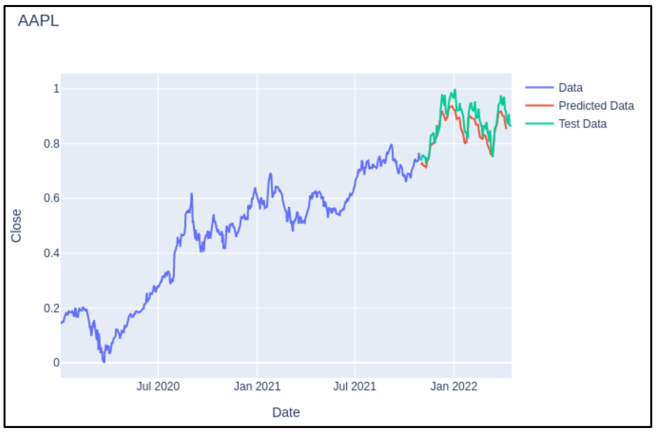

Fig 5.8: Rmsprop optimizer for Apple dataset

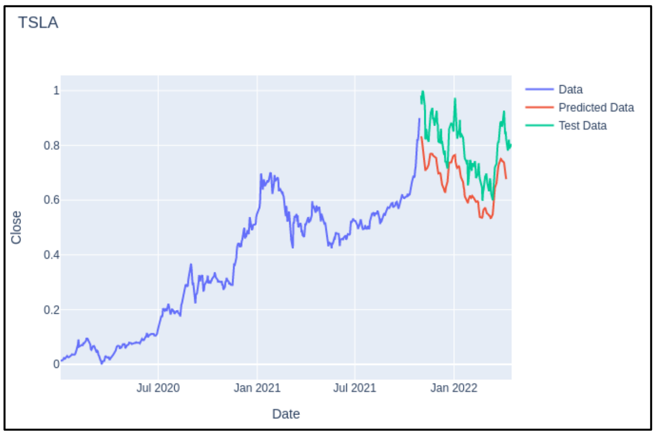

Fig 5.9: Rmsprop optimizer for Tesla dataset

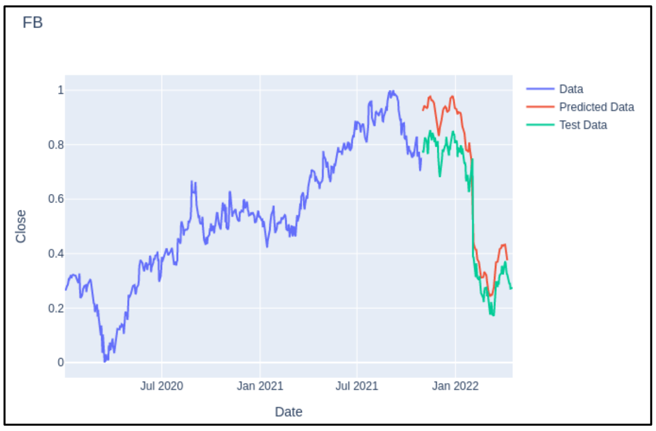

Fig 5.10: Rmsprop optimizer for Facebook dataset

Stock	MSE	RMSE	R^2
GOOG	0.00087	0.02956	0.80344
TSLA	0.01732	0.13161	-0.7861
AAPL	0.00168	0.04104	0.66852
AMZN	0.00148	0.03856	0.88471
FB	0.01263	0.11241	0.79269

Table 5.2: Rmsprop optimizer performance metrics

Note: MSE and RMSE values are low and all R^2 values are not high, it denotes that **Rmsprop** optimizer is not preferable for satisfactory results.

5.1.1.3 SGD

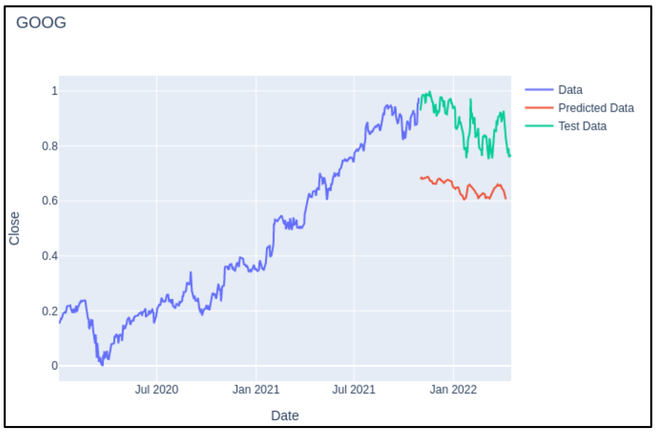

Fig 5.11: SGD optimizer for Google dataset

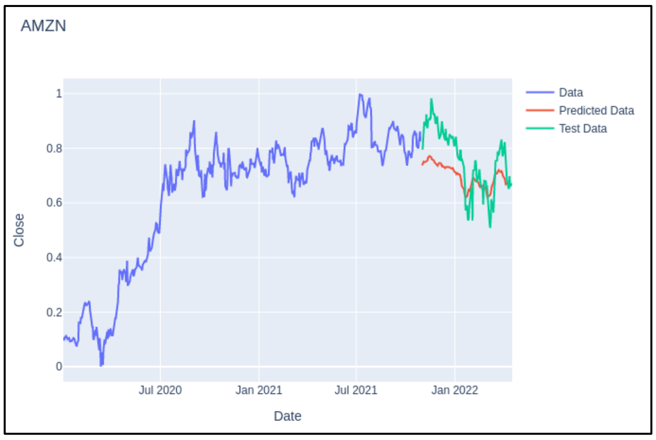

Fig 5.12: SGD optimizer for Amazon dataset

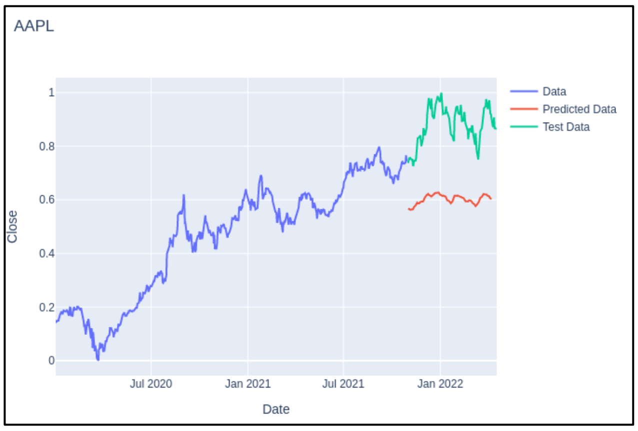

Fig 5.13: SGD optimizer for Apple dataset

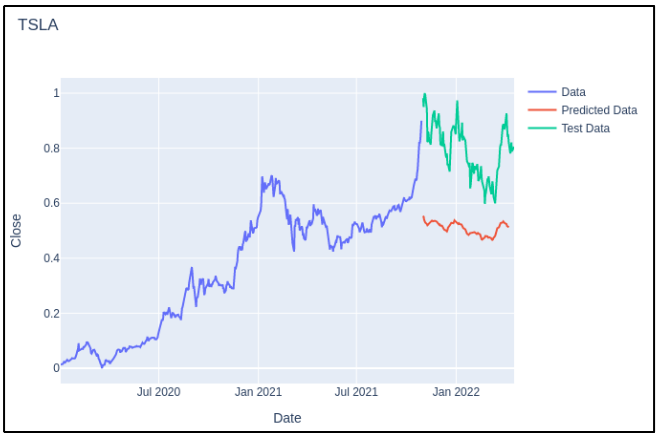

Fig 5.14: SGD optimizer for Tesla dataset

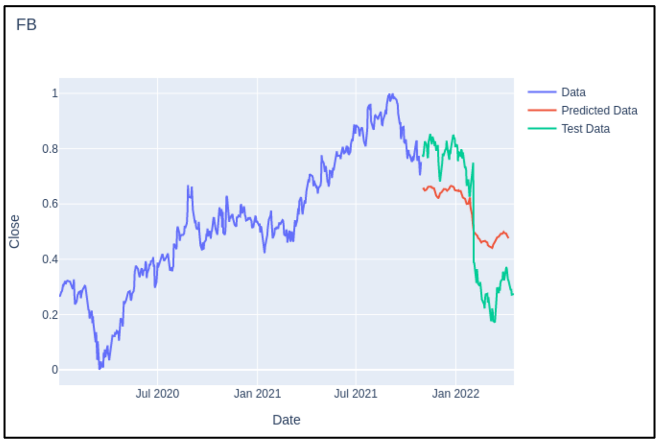

Fig 5.15: SGD optimizer for Facebook dataset

Stock	MSE	RMSE	R^2
GOOG	0.06160	0.24819	-12.8524
TSLA	0.08883	0.29804	-8.15954
AAPL	0.08054	0.28381	-14.8517
AMZN	0.00996	0.09982	0.227499
FB	0.02732	0.16531	0.551634

Table 5.3: SGD optimizer performance metrics

Note: MSE and RMSE values are low but R^2 values are not high, it denotes that **SGD** optimizer is not preferable for satisfactory results

We have noticed that Adam optimizer is the best optimizer because MSE and RMSE values are extremely low and R^2 values are high

5.1.2 Loss measures

5.1.2.1. Binary cross entropy

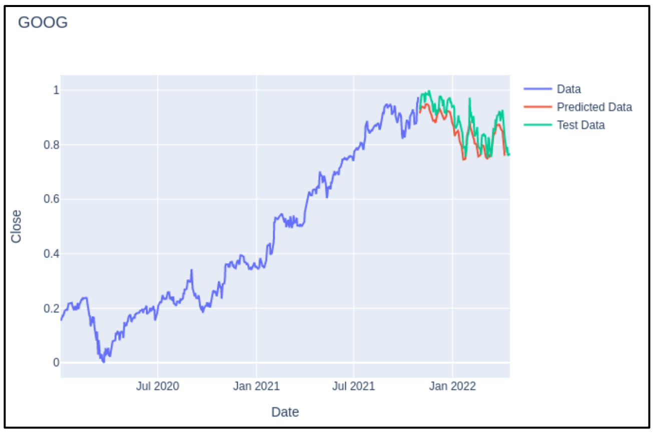

Fig 5.16: Binary Cross Entropy for Google dataset

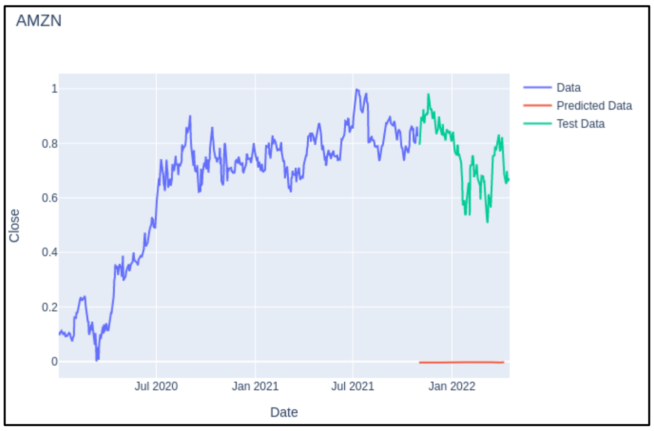

Fig 5.17: Binary Cross Entropy for Amazon dataset

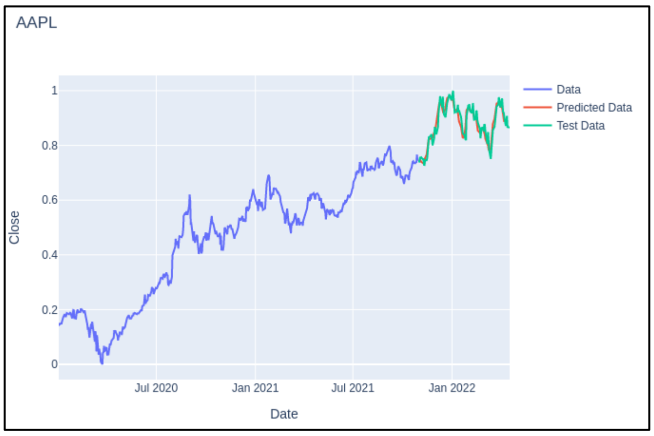

Fig 5.18: Binary Cross Entropy for Apple dataset

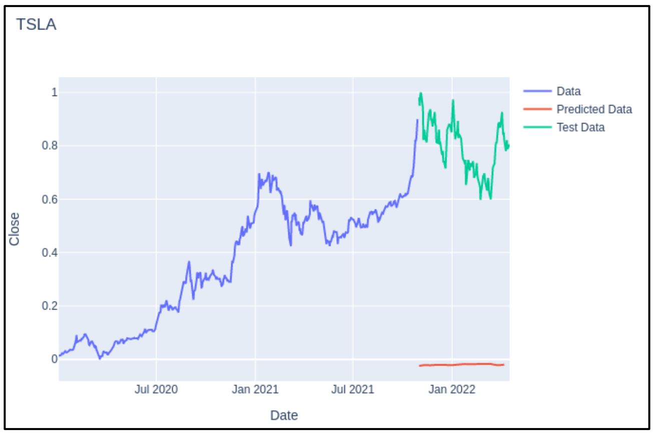

Fig 5.19: Binary Cross Entropy for Tesla dataset

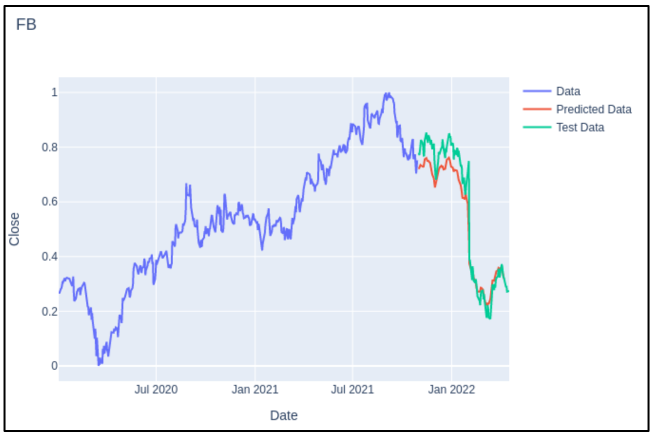

Fig 5.20: Binary Cross Entropy for Facebook dataset

Stock	MSE	RMSE	R^2
GOOG	0.00210	0.04583	0.52766
TSLA	0.67542	0.82184	-68.6425
AAPL	0.00050	0.02239	0.90129
AMZN	0.59969	0.77439	-45.4885
FB	0.00434	0.06588	0.92879

Table 5.4: Binary Cross Entropy performance metrics

Note: MSE and RMSE values are low but R^2 values are not high, it denotes that **Binary Cross Entropy** is not preferable for satisfactory results

5.1.2.2. MSE

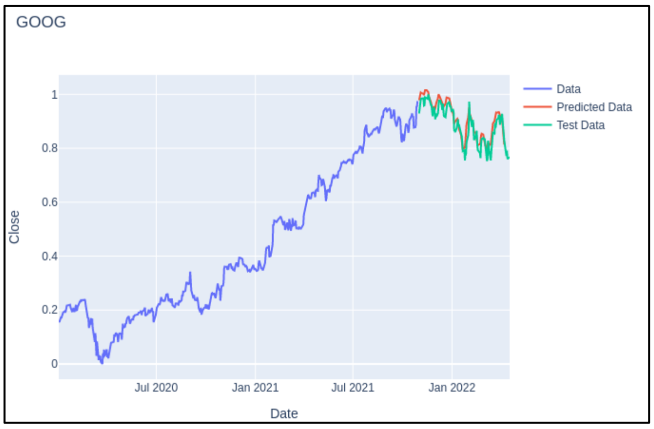

Fig 5.21: MSE for Google dataset

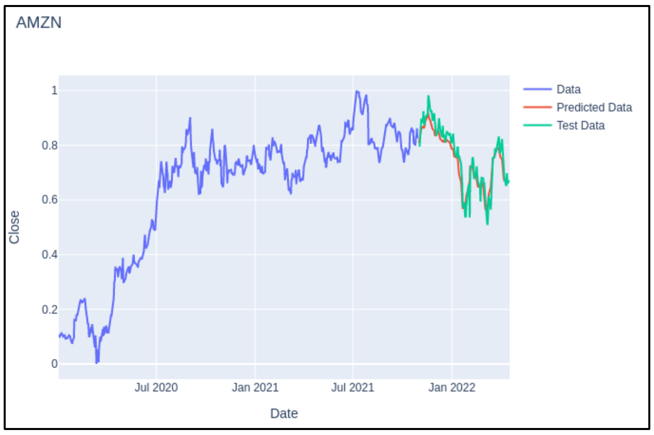

Fig 5.22: MSE for Amazon dataset

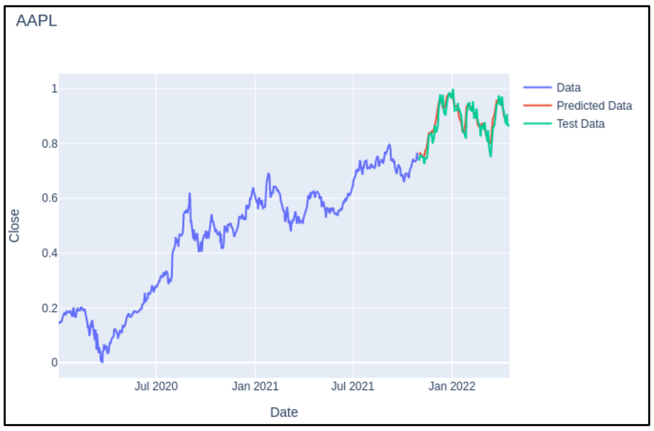

Fig 5.23: MSE for Apple dataset

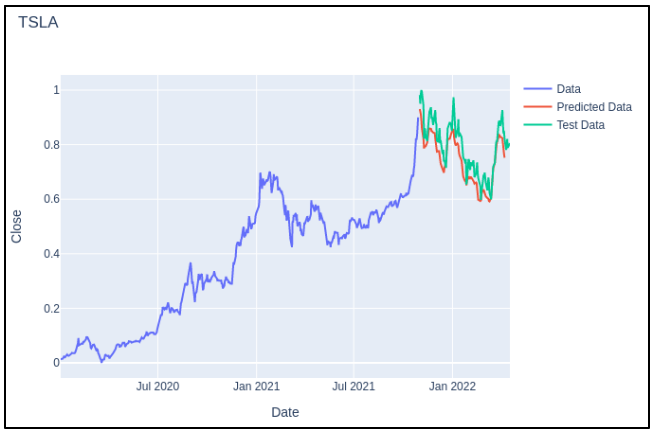

Fig 5.24: MSE for Tesla dataset

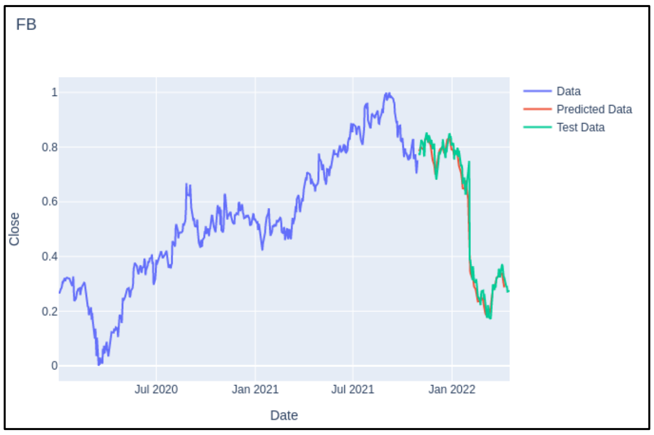

Fig 5.25: MSE for Facebook dataset

Stock	MSE	RMSE	R^2
GOOG	0.00089	0.02997	0.79798
TSLA	0.00370	0.06082	0.61847
AAPL	0.00058	0.02428	0.88390
AMZN	0.00149	0.03861	0.88442
FB	0.00189	0.04350	0.96894

Table 5.5: MSE performance metrics

Note: MSE and RMSE values are low and R^2 values are high, it denotes that **MSE** is preferable for satisfactory results

We have noticed that MSE is the best loss measure

5.2 LSTM performance for different datasets

5.2.1 From 2015-01-01 to 2022-05-01

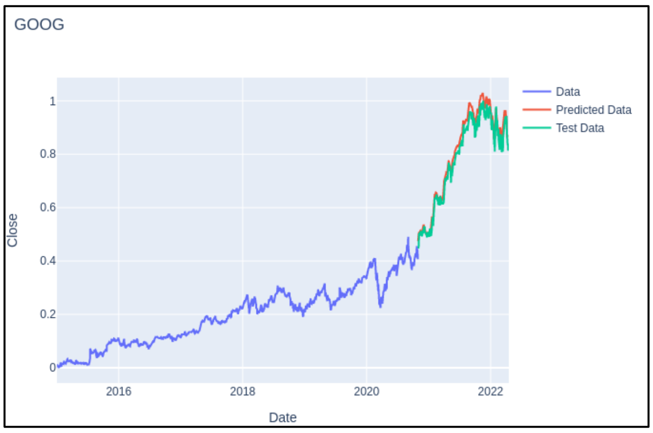

Fig 5.26: LSTM for Google dataset 2015-22

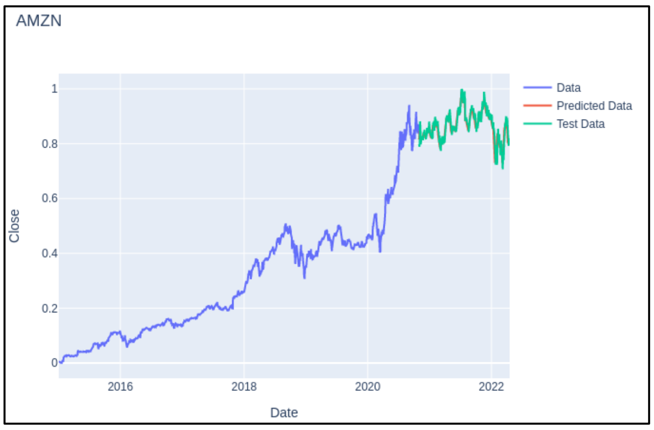

Fig 5.27: LSTM for Amazon dataset 2015-22

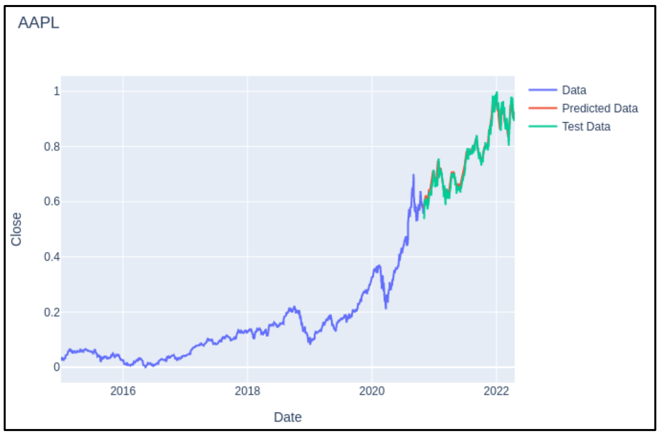

Fig 5.28: LSTM for Apple dataset 2015-22

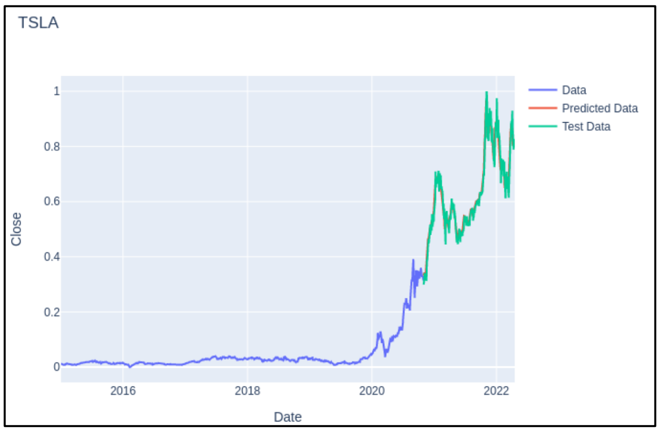

Fig 5.29: LSTM for Tesla dataset 2015-22

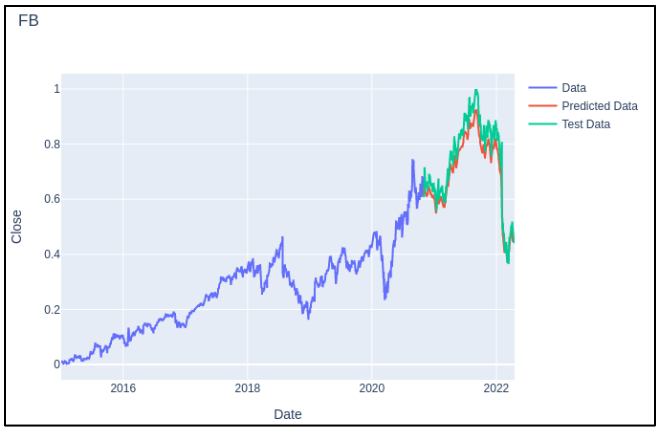

Fig 5.30: LSTM for Facebook dataset 2015-22

Stock	MSE	RMSE	R^2
GOOG	0.00095	0.03084	0.96407
TSLA	0.00075	0.02746	0.96667
AAPL	0.00037	0.01934	0.97205
AMZN	0.00030	0.01741	0.89628
FB	0.00258	0.05083	0.88969

Table 5.5: LSTM performance for dataset 2015-22

Note: MSE and RMSE values are low and R^2 values are high, it denotes that **LSTM** is preferable for satisfactory results

5.2.2 From 2020-01-01 to 2022-05-01

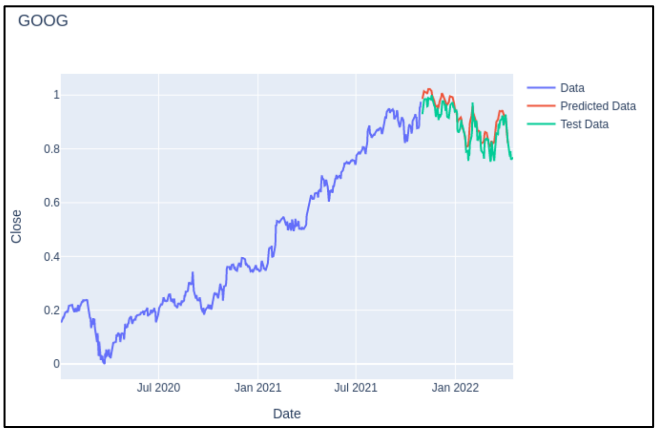

Fig 5.31: LSTM for Google dataset 2020-22

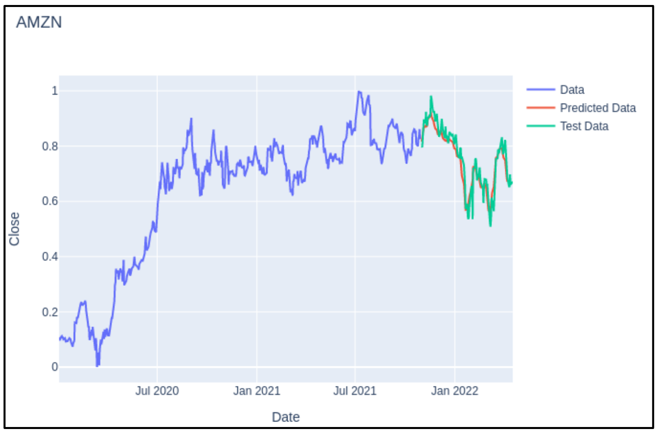

Fig 5.32: LSTM for Amazon dataset 2020-22

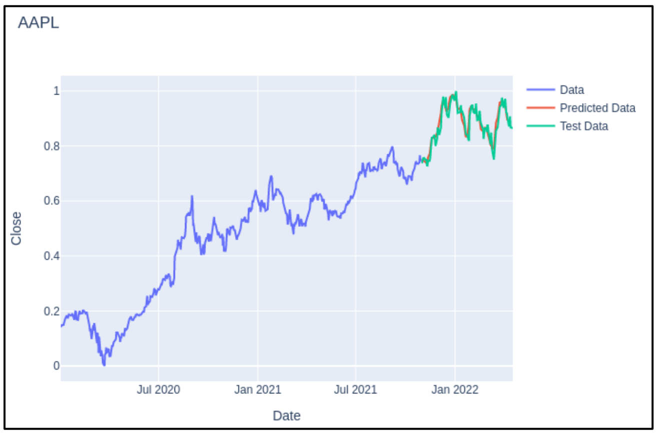

Fig 5.33: LSTM for Apple dataset 2020-22

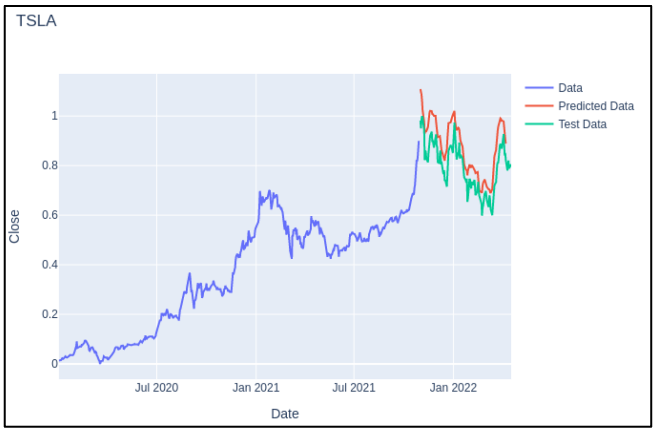

Fig 5.34: LSTM for Tesla dataset 2020-22

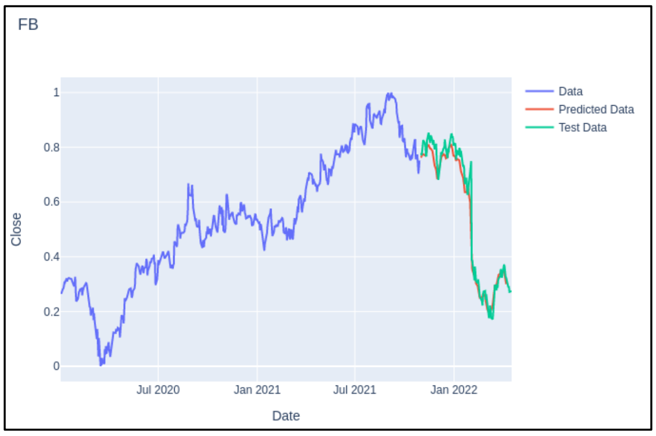

Fig 5.35: LSTM for Facebook dataset 2020-22

Stock	**MSE**	**RMSE**	R^2
GOOG	0.00129	0.03595	0.70934
TSLA	0.00798	0.08937	0.57644
AAPL	0.00048	0.02211	0.90374
AMZN	0.00144	0.03802	0.88789
FB	0.00219	0.04686	0.96396

Table 5.6: LSTM performance for dataset 2020-22

Note: MSE and RMSE values are low and R^2 values are high, it denotes that **LSTM** is preferable for satisfactory results

5.3 Compare LSTM to other models

Dataset range 2015-22

Note: MSE and RMSE values should be low and R^2 values should be high

Stock	Model	MSE	RMSE	R^2	Best
GOOG	LSTM	0.00156	0.041	0.80	LSTM
	RNN	5022.38	70.86	0.72	
	ARIMA	3773.91	61.43	0.79	
	SMA	19929.3	141.1	-0.0	
AMZN	LSTM	0.00133	0.036	0.89	LSTM
	RNN	8791.87	93.76	0.83	
	ARIMA	9812.72	99.05	0.81	
	SMA	61294.7	247.5	-0.14	
AAPL	LSTM	0.00059	0.024	0.88	LSTM
	RNN	18.0383	4.247	0.76	
	ARIMA	14.2403	3.773	0.81	
	SMA	172.477	13.13	-1.23	
TSLA	LSTM	0.00107	0.032	0.88	LSTM
	RNN	2804.60	52.95	0.77	
	ARIMA	2398.89	48.97	0.80	
	SMA	34719.0	186.3	-1.78	

Table 5.7: Compare LSTM to other models

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

From the above tables and figures we can infer that LSTM model with ADAM optimizer performs much better than other models such as ARIMA, SMA and RNN. We got satisfactory results that is above 80% accurate stock prediction.

6.2 Future work

We add new features like finding the local minima and local maxima to buy and sell the stock respectively, add API's or some other operations to let the algorithm do the trading for us.

7. REFERENCES

[1] [For SMA description and formula]

- https://www.investopedia.com/terms/s/sma.asp

[2] [For ARIMA terms and details]

- https://journal.jis-institute.org/index.php/ijmhrr/article/view/235
- https://www.investopedia.com/terms/a/autoregressive-integrated-moving-averagearima.asp

[3] [For RNN details and images]

- https://www.ijert.org/research/future-stock-price-prediction-using-recurrent-neural-network-lstm-and-machine-learning-IJERTCONV9IS05065.pdf
- https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[4] [For SMA and ARIMA implementation]

- https://github.com/0xpranjal/Stock-Prediction-using-different-models/blob/main/Notebooks/1.%20Time%20Series%20Forecasting%20with%20Naive%2C%20Moving%20Averages%2C%20and%20ARIMA.ipynb

[5] [For LSTM details and images]

- https://www.researchgate.net/publication/348390803_Stock_Price_Prediction_Using_LSTM
- https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/
- https://en.wikipedia.org/wiki/Long_short-term_memory#:~:text=Long%20short%2Dterm%20memory%20(LSTM,networks%2C%20LSTM%20has%20feedback%20connections.
- https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

[6] [For LSTM implementation]

- https://towardsdatascience.com/time-series-forecasting-with-recurrent-neural-networks-74674e289816
- https://medium.com/visionary-hub/using-lstms-to-predict-future-stock-prices-61f4458fc860