MSA 8600 Deep learning Final Project– Group 5

Project Report, April 18th, 2023

Project 1: Predicting Microsoft Closing Price using RNN

Project 2: Reinforcement Learning game Mind Tracker

Team Members:

Shreyashi Mukhopadhyay

Swathy Raja

Jibi Jacob

Anesia McCarthy

Project 1: Microsoft Stock Price Prediction using RNN.

Introduction

The stock market has enormously large historical data that varies with trade date, which is time-series data. LSTM and GRU models within RNN based models are widely used for sequence prediction problems and have proven to be extremely effective as they can store past important information and forget the information that is not.

In this project we have applied simple and composite RNN models like LSTM, CNN-LSTM, BiLSTM, CNN-BiLSTM & GRU in conjunction with NEURAL Prophet to predict the closing price of the Microsoft stock.

Microsoft Corporation (MSFT) is one of the largest technology companies in the world, with a market capitalization of over $2 trillion. As a leading provider of software, hardware, and cloud services, Microsoft's performance is closely watched by investors, analysts, and market participants.

In this context, predicting Microsoft's stock price movements can be of great interest to traders, investors, and financial analysts. Predicting stock prices is a challenging task that involves analyzing various economic and financial indicators, company-specific data, news events, and market trends, among other factors.

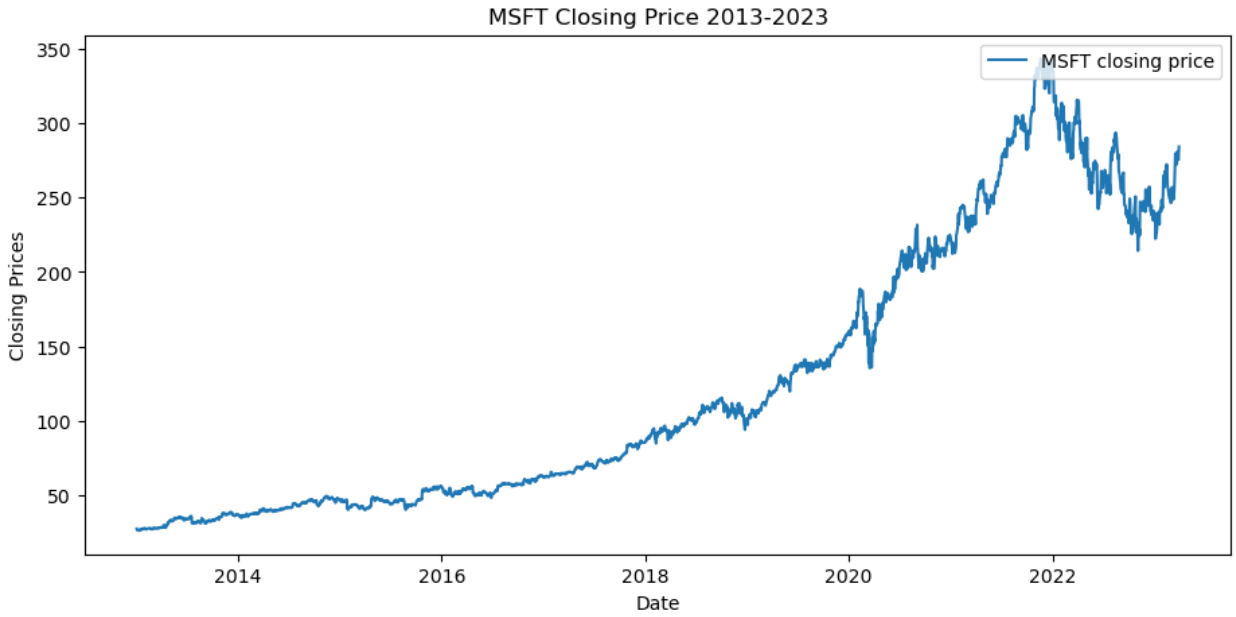
In recent years, machine learning has emerged as a powerful tool for predicting stock prices, using algorithms that can learn from historical data to identify patterns and make predictions about future prices. Machine learning models can analyze large amounts of data, including news articles, social media sentiment, and economic indicators, to generate predictions with high accuracy. However, stock price prediction is a complex and uncertain task, and machine learning models must be carefully designed and evaluated to avoid overfitting and improve performance.

Data Exploration

The historical stock price data for Microsoft (MSFT) was collected from Yahoo Finance using the yfinance library within pandas. This extracted 9348 rows of data and dividends, and stock splits.

For the purpose of our prediction only closing price was utilized panning the time frame January 1st, 2013 to March 31st, 2023.The data consisted of 6 columns including open price, highest price, lowest price, closing price, volume. The closing price is the last price at which the stock is traded during the regular trading day. A stock’s closing price is the standard benchmark used by investors to track its performance over time. The chosen dataset consisted of 2579 records and the data had no null values.

Figure 1



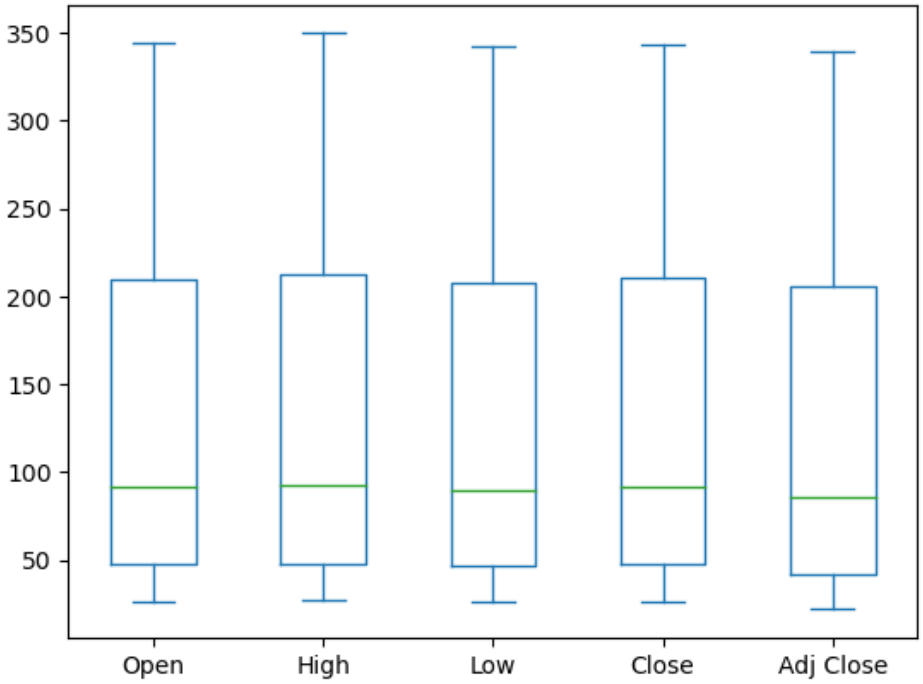
Data Description:

```
#   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Open        2579 non-null   float64
 1   High        2579 non-null   float64
 2   Low         2579 non-null   float64
 3   Close       2579 non-null   float64
 4   Adj Close   2579 non-null   float64
 5   Volume      2579 non-null   int64
```

Data Null Values:

```
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
```

Microsoft Stock Price Descriptive Statistics:



Microsoft Stock Data Descriptive Statistics 2013-2023

	Open	High	Low	Close	Adj	Volume
count	2579	2579	2579	2579	2579	2.58E+03
mean	124.9148	126.1763	123.6076	124.9573	120.2674	3.23E+07
std	91.0774	92.08854	90.00056	91.08922	91.99527	1.67E+07
min	26.49	26.75	26.28	26.46	21.76367	7.43E+06
25%	47.285	47.72	46.725	47.325	41.50475	2.25E+07
50%	91.21	92.73	89.74	91.49	86.13831	2.84E+07
75%	210.06	212.3	208.045	210.195	205.4011	3.67E+07
max	344.62	349.67	342.2	343.11	339.0756	2.48E+08

The Microsoft stock data descriptive statistics from 2013-2023 indicate the minimum stock closing price was $26.46 while the maximum stock closing price was $339.0756.
The mean closing price was recorded at $124 with a standard deviation of $91. The 25th percentile was recorded at $47 and the 50th percentile was recorded at $91 while the 75th percentile was recorded at $210.

Trend Indicators

Simple Moving Average

Simple moving averages (SMAs) are used to chart the long-term trajectory of a stock or other security, while ignoring the noise of day-to-day price movements. This allows MSFT_dfaders to compare medium- and long-term MSFT_dfends over a larger time horizon. For example, if the 200-day SMA of a security fall below its 50-day SMA, this is usually interpreted as a bearish death cross pattern and a signal of further declines. The opposite pattern, the golden cross, indicates potential for a market rally. The SMA for our data was calculated using a window size of both 30 and 60.

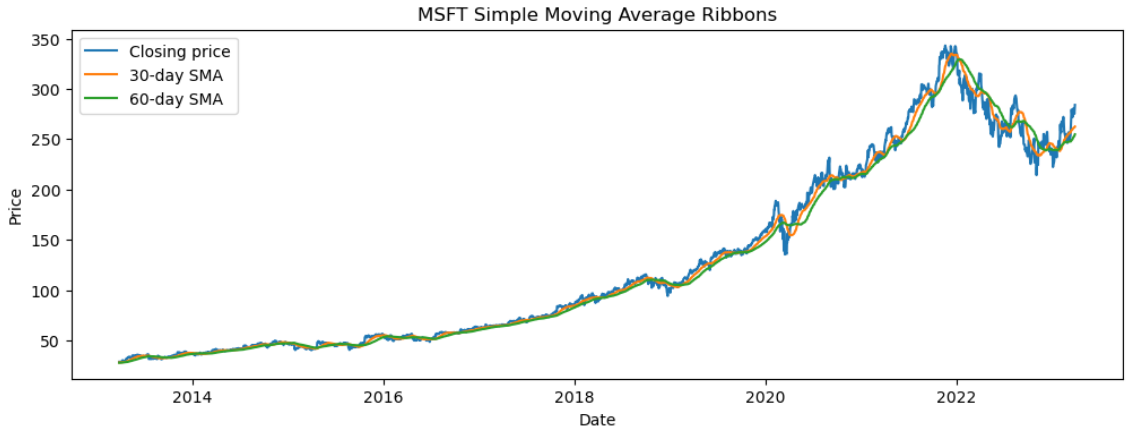


Figure 2

Exponential Moving Average

Exponential Moving Average (EMA) is a technical analysis indicator that places greater weight on recent price data points, compared to Simple Moving Average (SMA) which considers all data points equally. EMA is calculated by taking a weighted average of the closing prices over a specified time period, with more weight given to the most recent price data points. The weightage is determined by the chosen smoothing factor or the span, which can be adjusted based on the trader's preference. EMA is often used to identify trends and potential buy/sell signals, as it responds more quickly to changes in the market compared to SMA. Similarly, to the SMA, EMA was calculated using the window sizes 30 and 60.

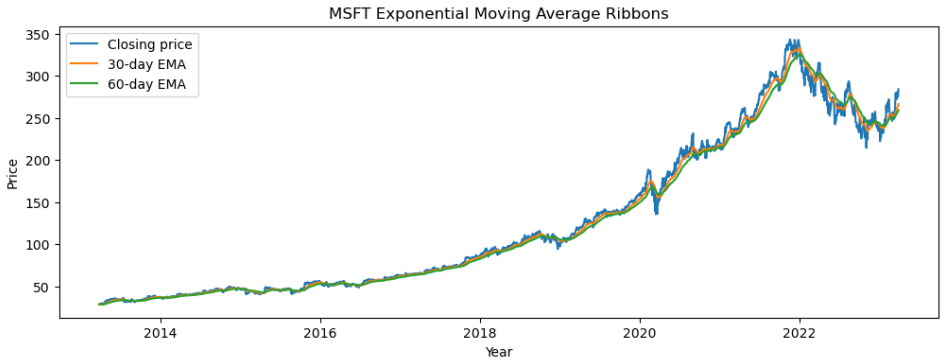
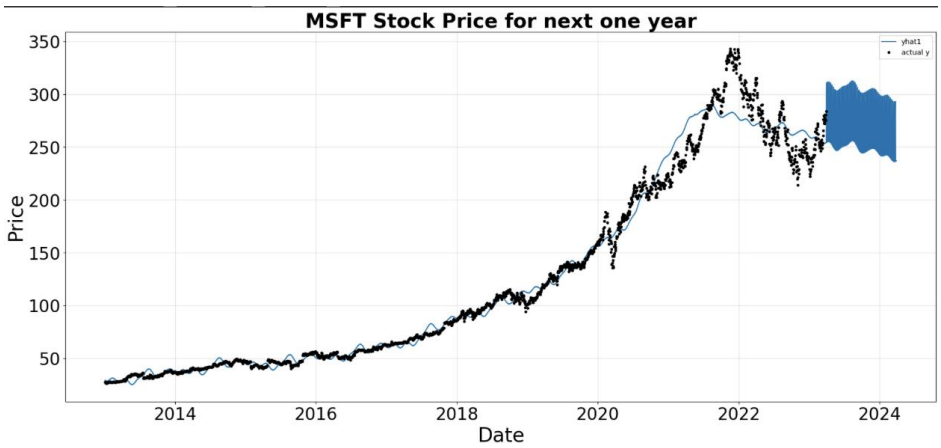


Figure 3

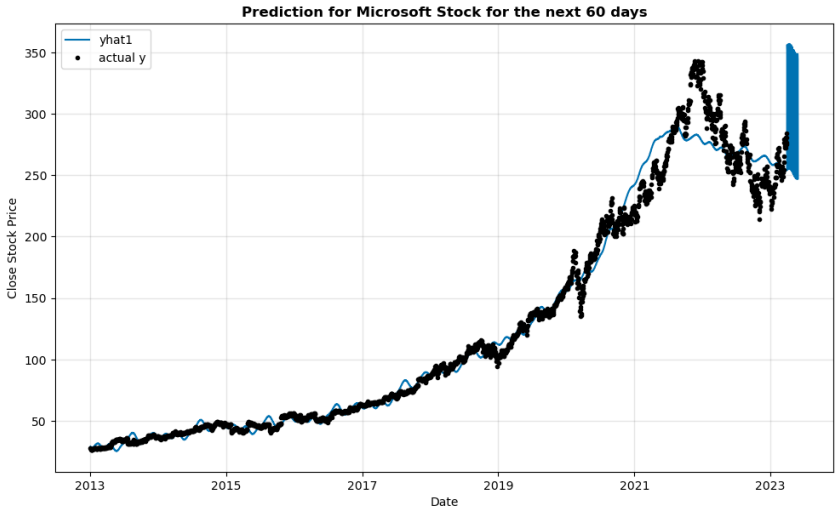
Neural Prophet

Neural Prophet is a deep learning time series forecasting library that uses neural networks to model and predict time series data. It is based on the PyTorch library and is capable of modeling complex time series data, including seasonal and non-linear trends, and can handle missing data as well. Neural Prophet provides a simple and intuitive API for training, evaluation, and prediction of time series models. It is also highly customizable, allowing users to specify their own model architectures and loss functions. In the context of stock prediction, Neural Prophet can be used to build and train neural network models to forecast stock prices based on historical stock data.

The first prophet model casts predictions for the next year. The model is trained for 147 epochs, with a learning rate selected based on a range test. The output of the model is then used to generate predictions for the future data, which is plotted against the actual data to visualize the forecasted trend. Respectively, model 2 makes predictions for the next 60 days.



Prophet 1



Prophet 2

To evaluate the metrics of the prophet models the values of four metrics were used: SmoothL1Loss, MAE, RMSE, and RegLoss. These metrics are commonly used to evaluate the accuracy of time-series models. The results show the values for each metric at each epoch during the training process. The metrics decrease over time as the model learns to make better predictions. In the final epoch, the model achieved a SmoothL1Loss of 0.001211, an MAE of 8.153888, an RMSE of 12.904163, and a RegLoss of 0.0. These metrics indicate that the model is making accurate predictions on the given data.

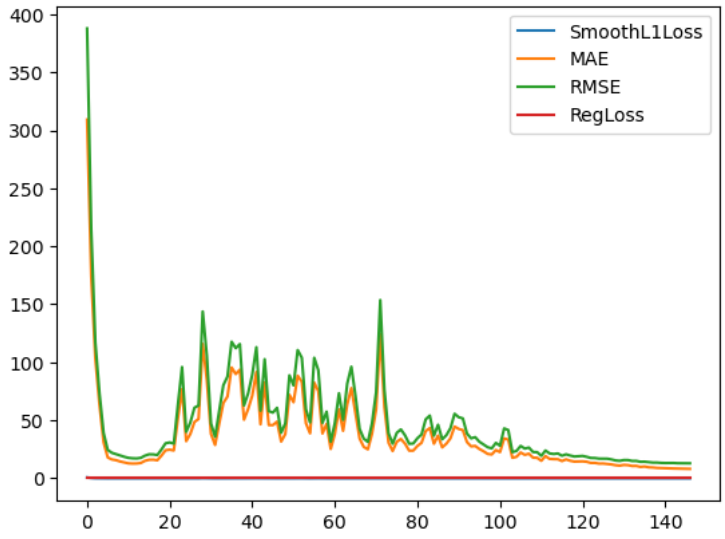


Figure 4

RNN Models

Data Pre-processing:

1. In the first step, we import the Microsoft stock data from 2013 -2023.

2. In step 2, we filter the data on the close price column.

3. Filtered Data length : 2579

4. We apply a Train Test Split: 80:20

5. Train data length : 2063

6. Validation data length : 516

7. Scale the data using min-max scaler in the range (0,1)

8. Define the Time steps =60

9. Reshape the data from a 2D array to a 3D array in the shape with the format of [samples, time steps and features].

10. Final shape of the train and test data is:

```
x_train.shape =  (2004, 60, 1)
y_train.shape =  (2004,)
x_test.shape =  (515, 60, 1)
y_test.shape =  (515, 1)
```

LSTM

LSTM stands for Long Short-Term Memory and is a type of neural network architecture that is commonly used for processing sequential data, such as time series. It was designed to address the vanishing gradient problem that can occur in traditional recurrent neural networks (RNNs) when trying to learn long-term dependencies in data. LSTM networks consist of memory cells that can selectively remember or forget information, allowing them to maintain information over long periods of time. They are commonly used in applications such as speech recognition, natural language processing, and time series forecasting.

Our model shows the architecture of the LSTM neural network model used for time series forecasting. It has two LSTM layers with 50 neurons each, followed by dropout layers with a rate of 0.2 to prevent overfitting. The output of the second LSTM layer is passed through two fully connected dense layers, one with 25 units and the other with a single unit, which produces the final

predicted value. The total number of trainable parameters in the model is 31,901. The optimizer used for training the model is Adam, and the loss function is mean squared error, with mean absolute error also being tracked as a metric during training.

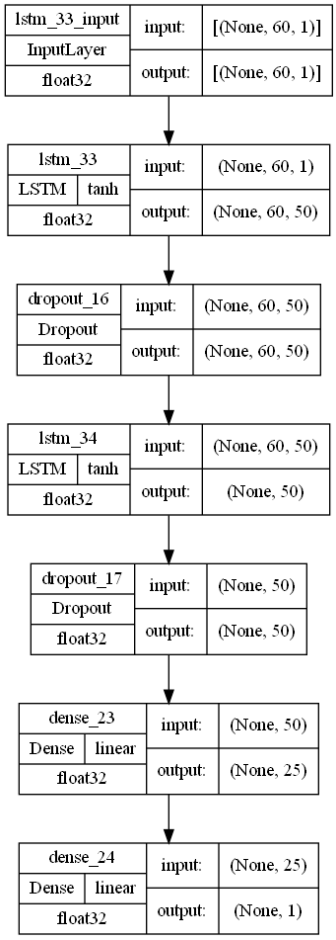


Figure 5

Results

The model appears to perform well in terms of both training and validation loss (measured by mean squared error) and mean absolute error. The loss and MAE both decrease over time, indicating that the model is learning and improving as it trains. The training loss (0.000068) and validation loss (0.000398) are both very low, which suggests that the model is fitting the data well and is not overfitting to the training data. The mean absolute error (MAE) measures the average absolute difference between the predicted and actual values, and the low MAE values (0.0058 for training and 0.0145 for validation) indicate that the model's predictions are generally close to the true values.

Performance Metrics:
Mean Absolute Error = 0.00608946243301034
Loss = 7.177415682235733e-05

RMSE: 2.0462493600197207
MSE: 4.1871364433811165
R2: 0.9233665109977625

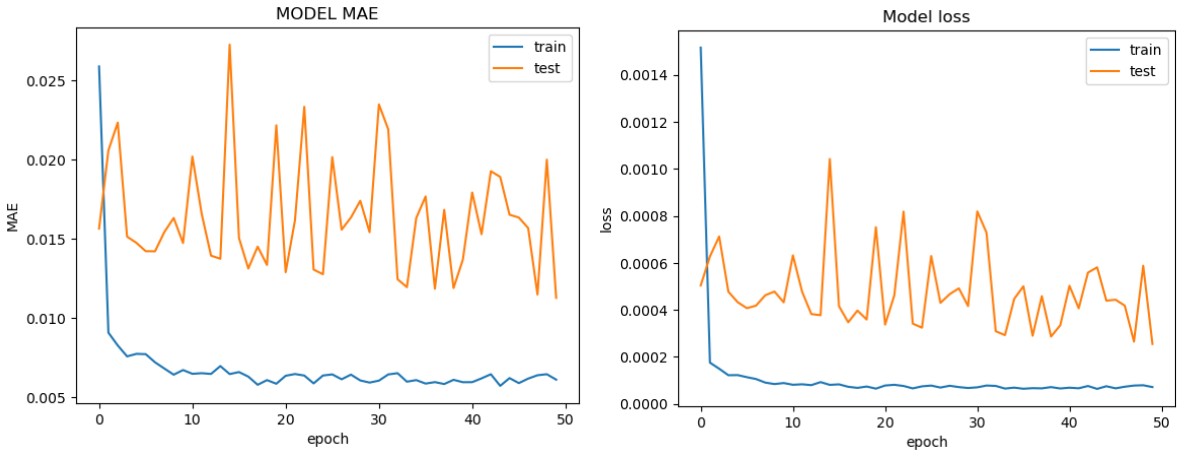


Figure 6

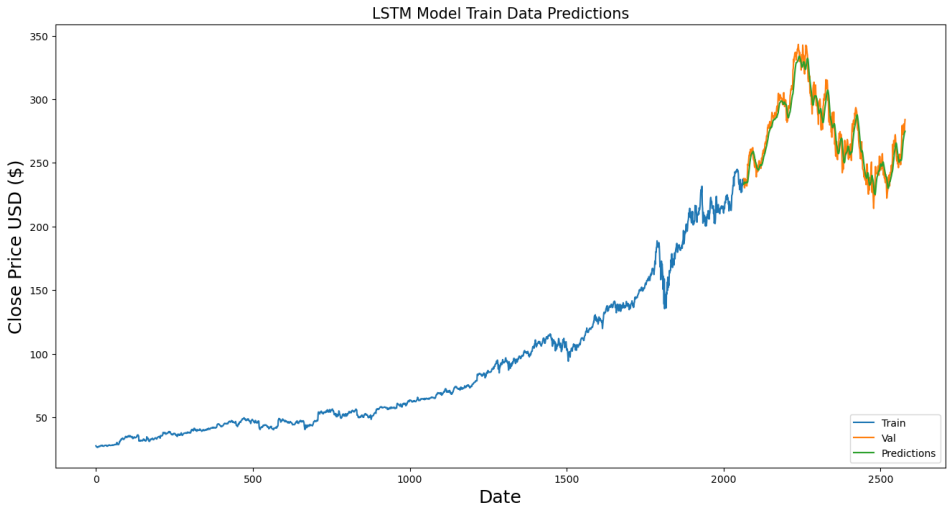


Figure 7

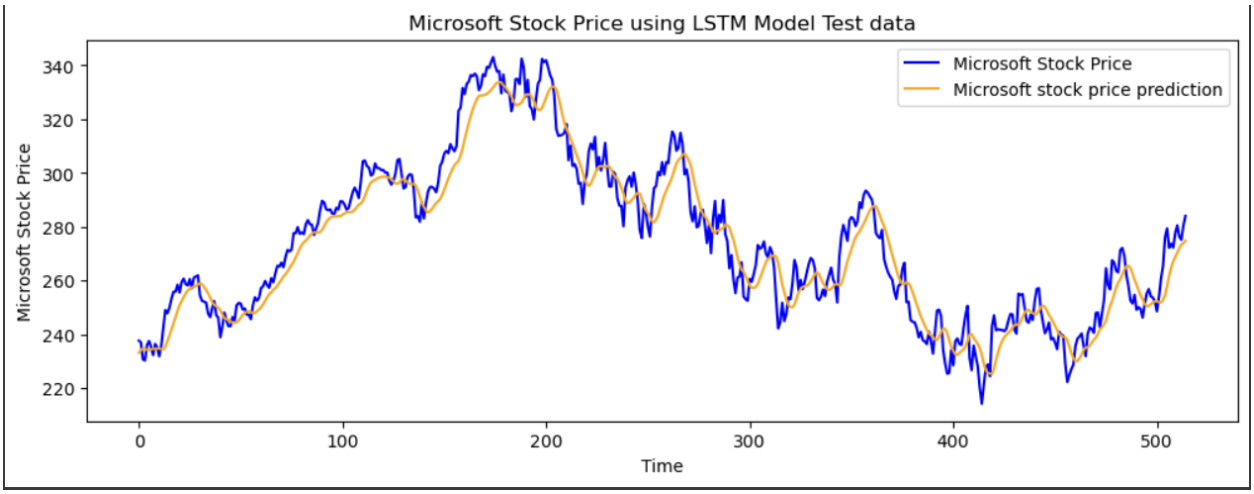


Figure 8

CNN LSTM

CNN-LSTM is a hybrid neural network architecture that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The CNN component processes input sequences by learning and extracting relevant features from them, and the LSTM component uses these features to model the temporal dependencies and capture long-term dependencies between inputs. The output of the CNN is fed to the LSTM as a sequence of feature maps, which are then processed by the LSTM layer. The CNN-LSTM architecture is commonly used for time-series data analysis, such as speech recognition and video analysis, where the input data has both spatial and temporal dimensions.

This model defines a sequential neural network model composed of several layers including a Conv1D layer, a MaxPooling1D layer, two LSTM layers, and a Dense layer. The model takes a 3D input tensor of shape (batch_size, timesteps, input_dim) where batch_size is the number of samples in each batch, timesteps is the length of each input sequence, and input_dim is the number of features in each input sample.

The Conv1D layer applies 64 filters of size 3 to the input tensor and uses the hyperbolic tangent activation function. The MaxPooling1D layer then applies max pooling with a pool size of 2. Next, two LSTM layers are added with 50 units each and using the hyperbolic tangent activation function. The return_sequences parameter of the first LSTM layer is set to True, which means that the output of this layer will be a 3D tensor which will be passed as input to the next LSTM layer. Finally, a Dense layer with a single unit is added to output the predicted value. The model is compiled using the Adam optimizer and mean squared error (MSE) as the loss function. Additionally, mean absolute error (MAE) is specified as a metric to monitor during training.

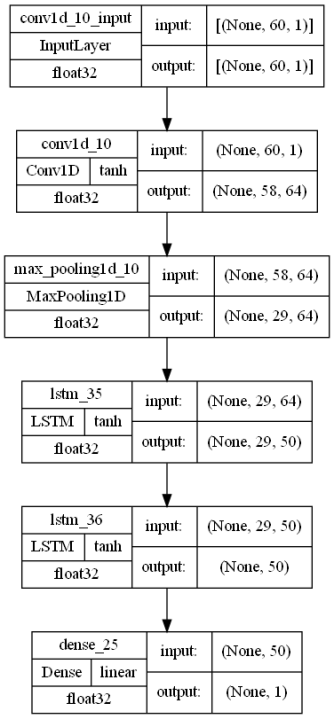


Figure 9

Results

The results show the training and validation loss (measured by mean squared error, MSE) and mean absolute error (MAE) for each epoch of the neural network model. The training loss and MAE decrease gradually with each epoch, indicating that the model is improving its fit to the training data. The validation loss and MAE also decrease, but not as consistently as the training loss, indicating some overfitting of the model to the training data.

The best validation loss and MAE are achieved in epoch 3 with a loss of 7.0972e-04 and MAE of 0.0190. The model then has a slight increase in validation loss and MAE for several epochs before a slight decrease in epoch 7. After that, the validation loss and MAE generally increase or remain stagnant, indicating the model may have started to overfit the data.

Performance Metrics:
MAE = 0.003856693161651492
LOSS = 2.8419215595931746e-05
RMSE: 14.491268002408223
MSE: 209.9968483176204
R2: 0.6760112846567832

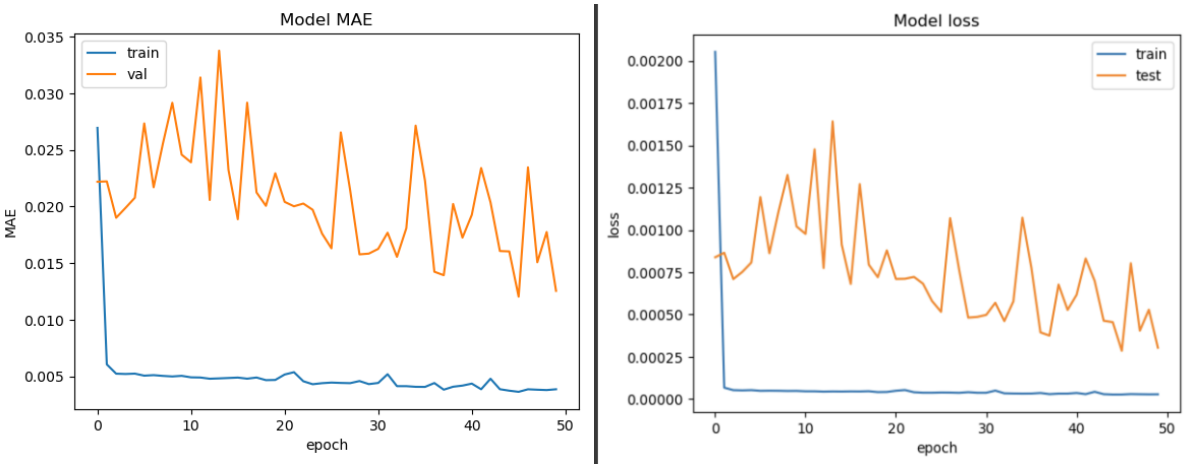


Figure 10

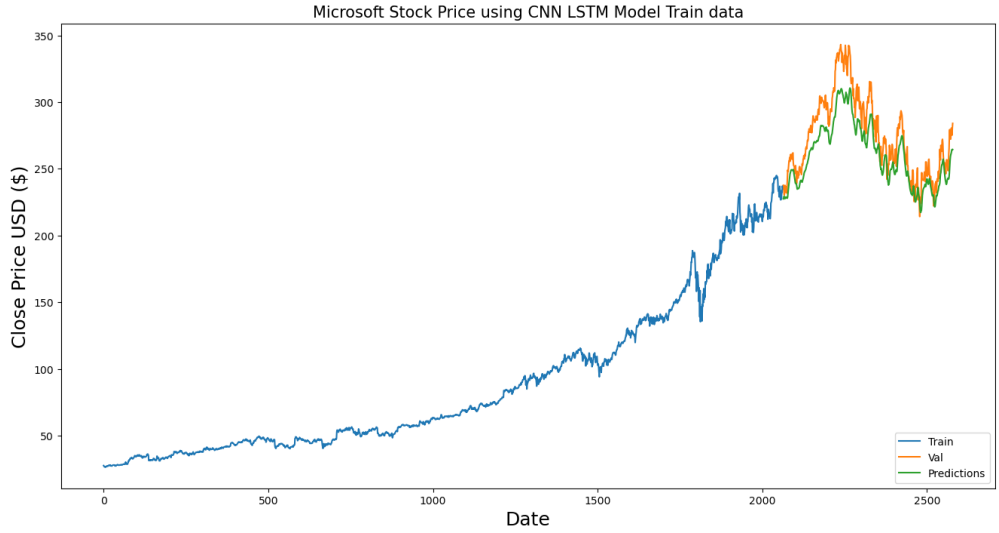


Figure 11

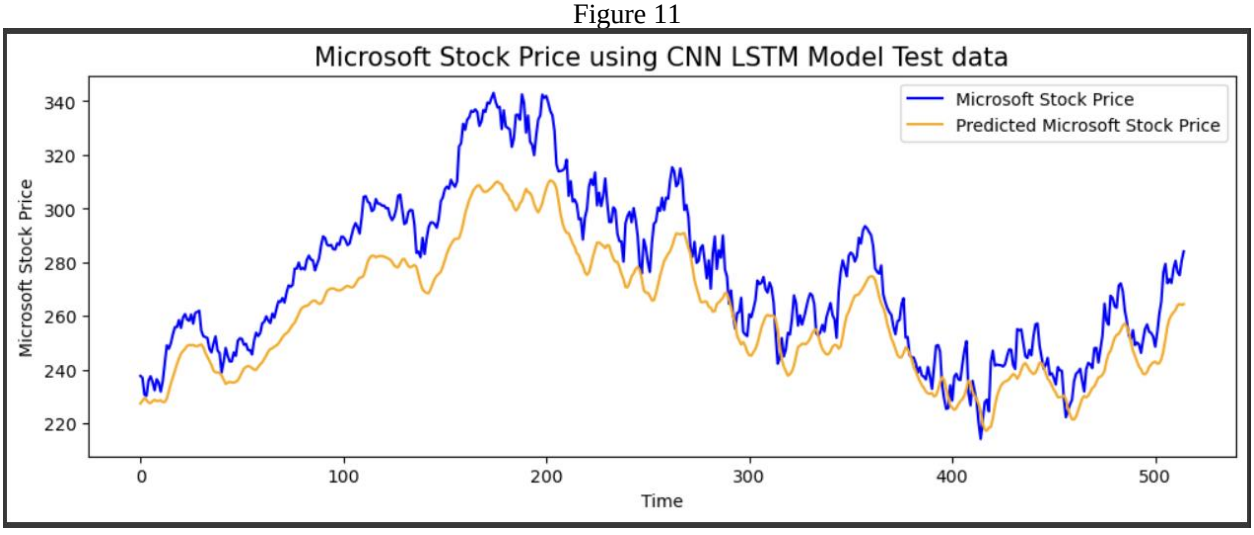


Figure 12

BiLSTM

Bidirectional LSTM (BiLSTM) is a type of neural network architecture that consists of two separate LSTMs, one processing the input sequence in a forward direction and the other in a backward direction. By combining information from both directions, BiLSTMs can capture more complex temporal patterns and dependencies in sequential data, making them especially useful for tasks such as speech recognition, natural language processing, and time series forecasting.

This model compiles a Sequential model in Keras for a bidirectional LSTM neural network with 50 hidden units and 'tanh' activation function. The input shape of the network is (number of time steps, 1) and it includes a dropout layer with a rate of 0.2. The model is then compiled with mean squared error (MSE) loss function, Adam optimizer and mean absolute error (MAE) as a metric. The summary function displays the architecture of the model including the layers, the output shape of each layer, the number of parameters in the model, and other information.

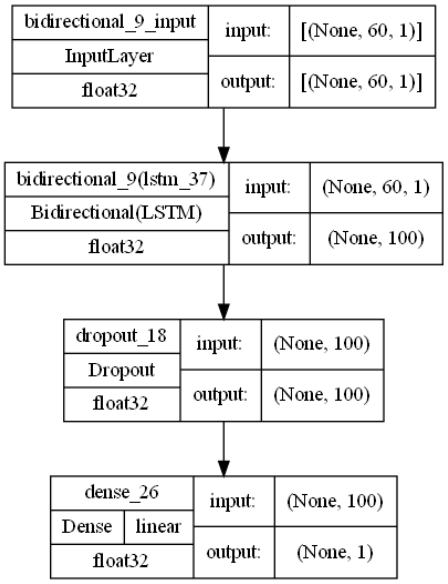


Figure 13

Results

As the number of epochs increases, the model becomes better at predicting the time series data, which is reflected in the decrease of the loss and MAE. In this case, we can see that the loss and MAE decrease rapidly in the first few epochs, and then begin to level off, suggesting that the model may have reached its optimal performance. The final values of the loss and MAE on the validation data are 0.0006 and 0.0151 respectively. These values indicate that the model can predict the time series data with a high degree of accuracy, as the MAE is relatively small compared to the range of the time series data.

Performance Metrics:
MAE = 0.0049415649846196175
LOSS = 4.929312490276061e-05
RMSE: 6.270157067752579
MSE: 39.31486965428762
R2: 0.8685028494168638

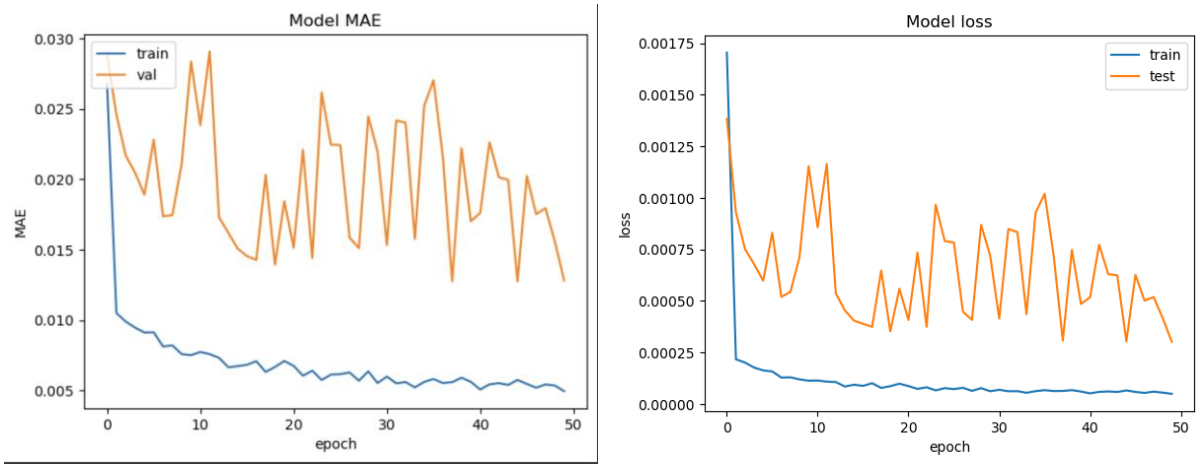


Figure 14

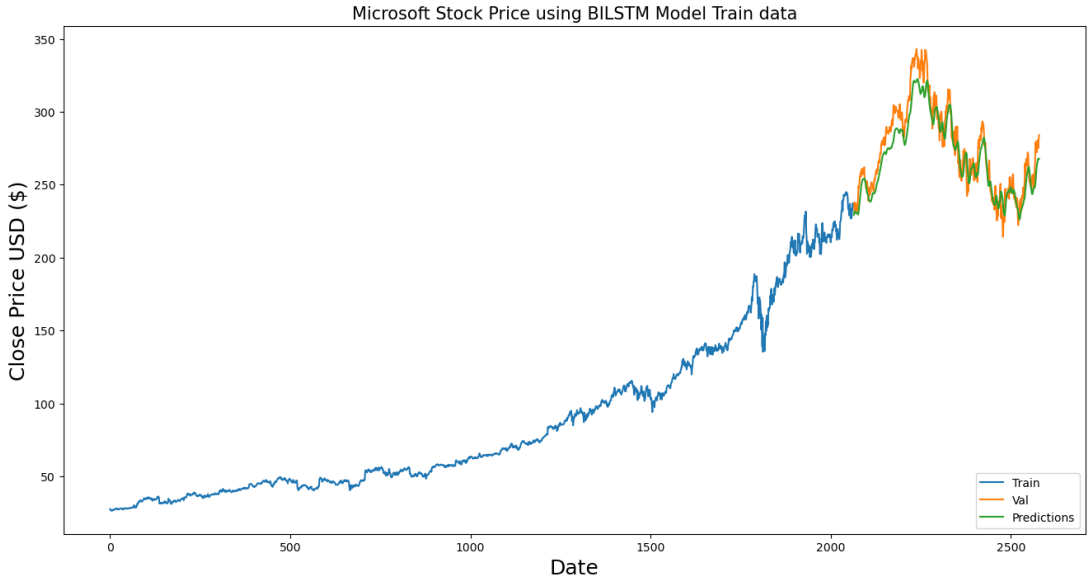


Figure 15

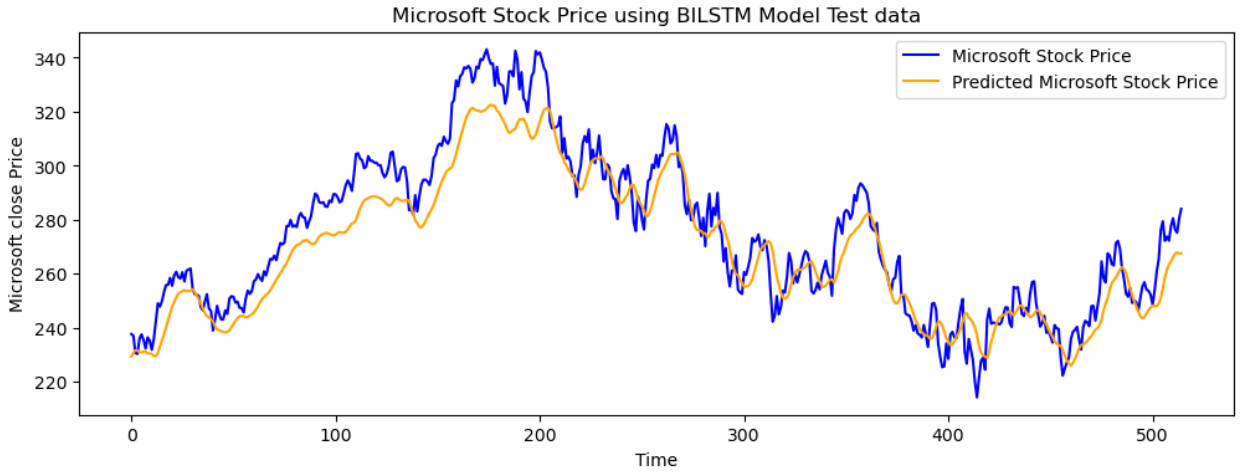


Figure 16

CNN BiLSTM

Similarly, to CNN-LSTM, CNN-BiLSTM is a type of deep learning model that combines two popular neural network architectures: Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (BiLSTM) networks. The CNN layers in the model extract important features from input data by sliding a filter over it, while the BiLSTM layers capture the sequence information by processing it in both forward and backward directions. The combination of these two architectures allows CNN-BiLSTM to effectively process both spatial and sequential information, making it well-suited for tasks such as image captioning, sentiment analysis, and speech recognition.

The CNN-BiLSTM builds a sequential model using Keras with four layers. The first layer is a one-dimensional convolutional layer with 64 filters and a kernel size of 3, with the hyperbolic tangent activation function. The input shape of the layer is defined by the shape of the training data. The second layer is a max-pooling layer with a pool size of 2. The third layer is a bidirectional LSTM layer with 64 units, which returns sequences. A dropout layer is added to randomly exclude 20% of the neurons during training to prevent overfitting. The fourth layer is another bidirectional LSTM layer with 64 units, which does not return sequences. Finally, a dense layer with one unit is added to produce the model's output. The model is compiled using the mean squared error loss function, the Adam optimizer, and the mean absolute error metric.

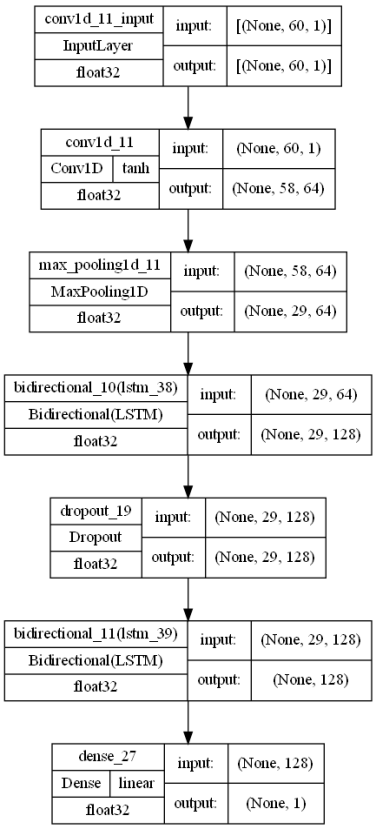


Figure 17

Results

The results suggest that the model is performing well in terms of minimizing the loss function and improving the mean absolute error on both the training and validation sets. Over the 50 epochs, the loss and MAE on both sets are decreasing, indicating that the model is learning from the training data and generalizing well to the validation data. The training loss started at 0.0013 and decreased to 2.2957e-05, while the validation loss started at 0.0009 and decreased to 0.0002. The MAE on the training set started at 0.0209 and decreased to 0.0023, while the MAE on the validation set started at 0.0230 and decreased to 0.0145. These values suggest that the model is successfully learning the patterns in the data and making accurate predictions.

Performance Metrics:
MAE = 0.0037112226709723473
LOSS = 2.6556523152976297e-05
RMSE: 17.294883772470417
MSE: 299.1130047032606
R2: 0.5765541678562096

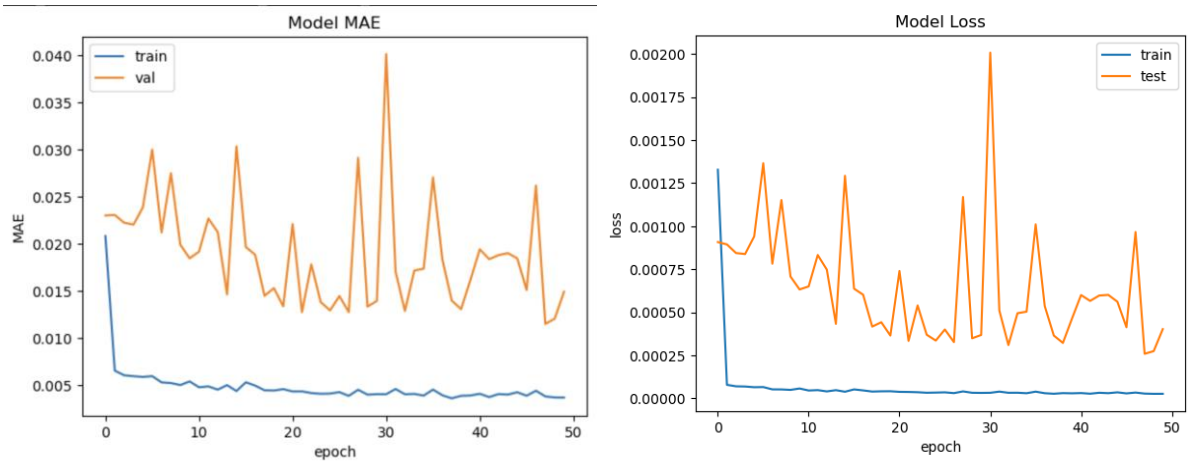


Figure 17

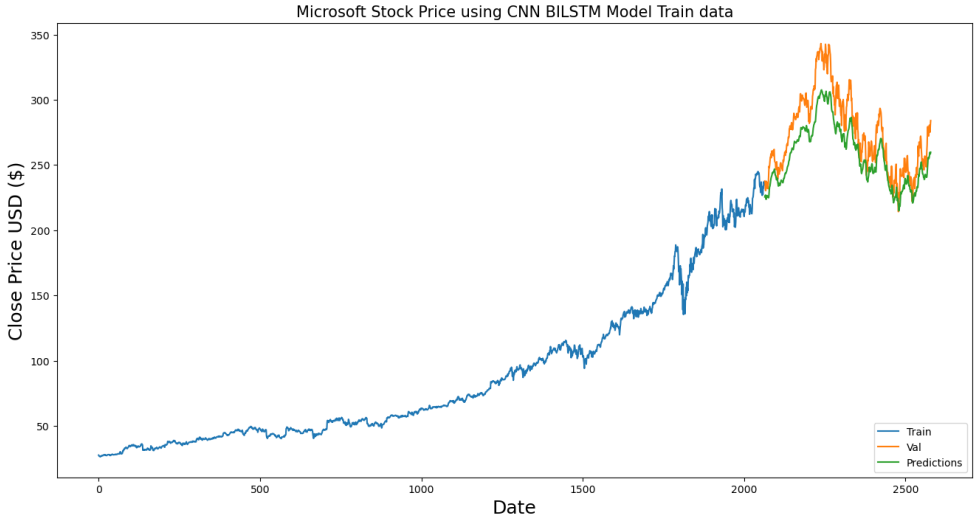


Figure 18

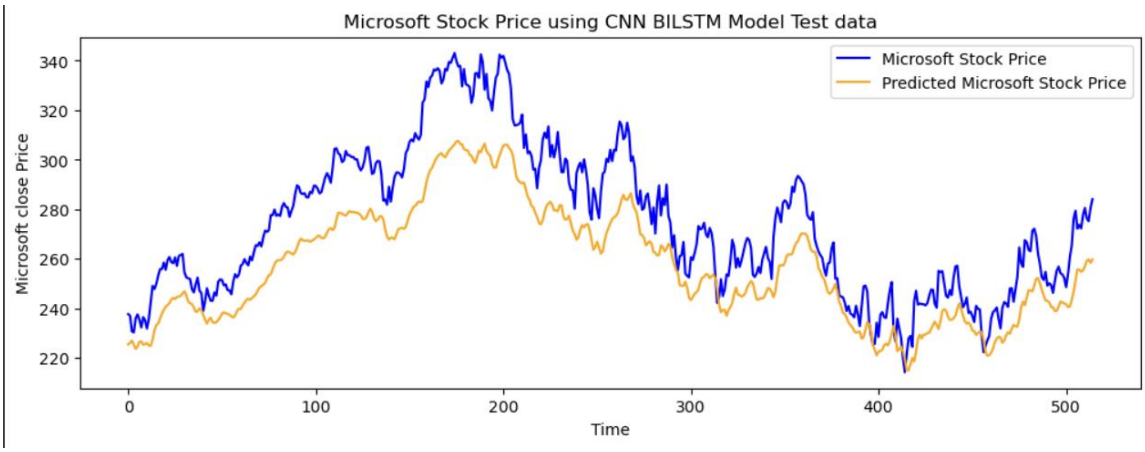


Figure 19

GRU

GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture. It was introduced by Cho et al. in 2014 as a modification to the original recurrent neural network (RNN) to address the vanishing gradient problem. GRU units typically have fewer parameters than other types of RNNs, making them computationally efficient, while still being able to capture long-term dependencies in sequences. They achieve this by incorporating gating mechanisms that regulate the flow of information in and out of the cell state, allowing the network to selectively update or forget information based on the input.

This GRU-based neural network model has a total of 53,901 parameters. It consists of four GRU layers, each with 50 units and the activation function 'tanh'. The first three GRU layers return sequences, while the fourth one returns only the last output. There is a dropout layer with a rate of 0.2 after the fourth GRU layer to prevent overfitting. The output layer is a single neuron with a linear activation function. The model was compiled with the mean squared error (MSE) loss function, the Adam optimizer, and the mean absolute error (MAE) metric.

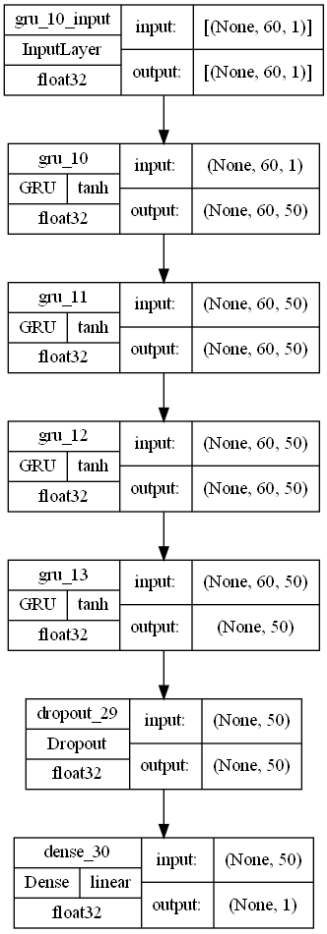


Figure 20

Results

The model performed quite well with both the training and validation loss and MAE decreasing over the epochs. However, the validation loss and MAE are slightly worse than the training loss and MAE.The validation MAE reached a minimum of 0.0118 after 14 epochs before gradually increasing again, indicating that the model may have started to overfit to the training data.

Performance Metrics:
MAE = 0.004897931590676308
LOSS = 4.8686317313695326e-05
RMSE: 2.454215010855962
MSE: 6.0231713195107295
R2: 0.9585839154493931

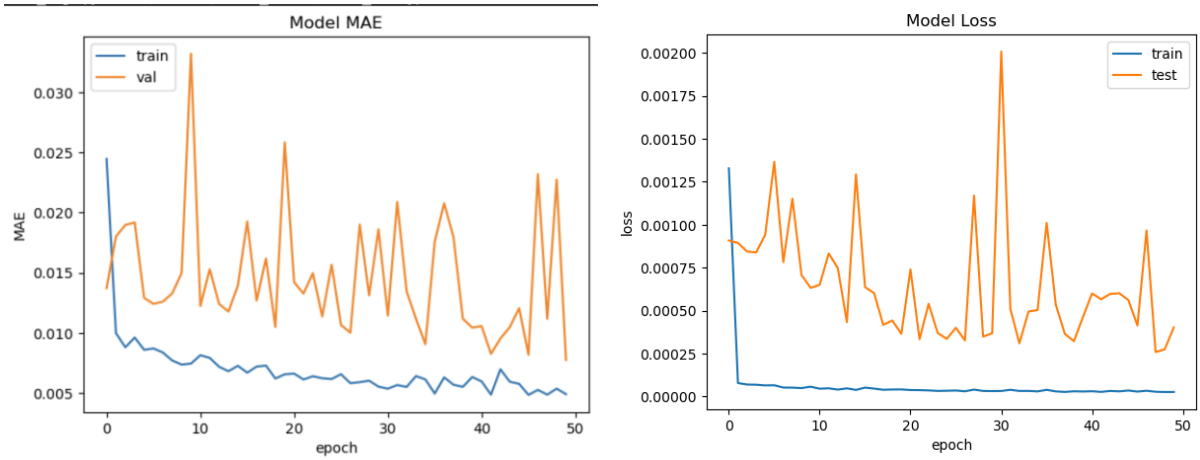


Figure 21

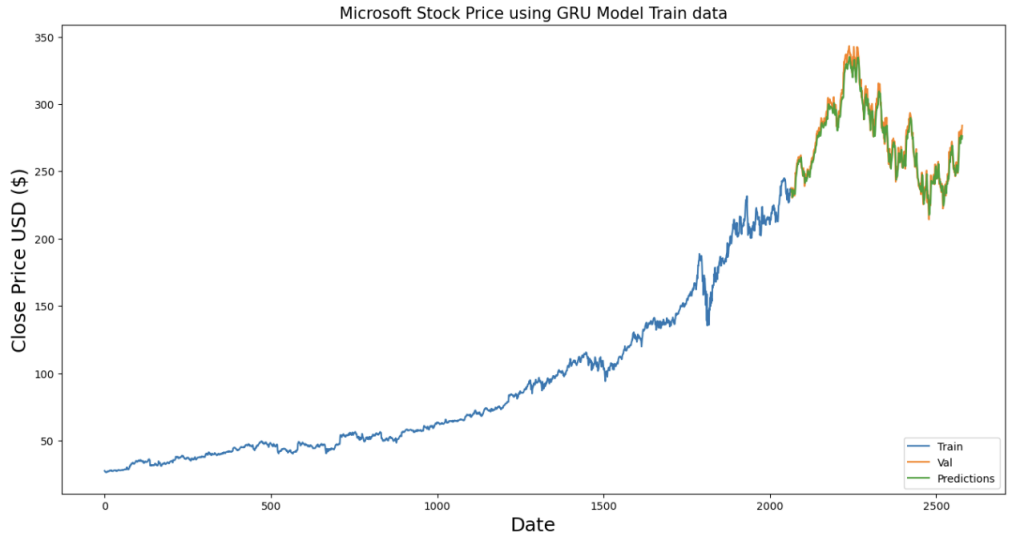


Figure 22

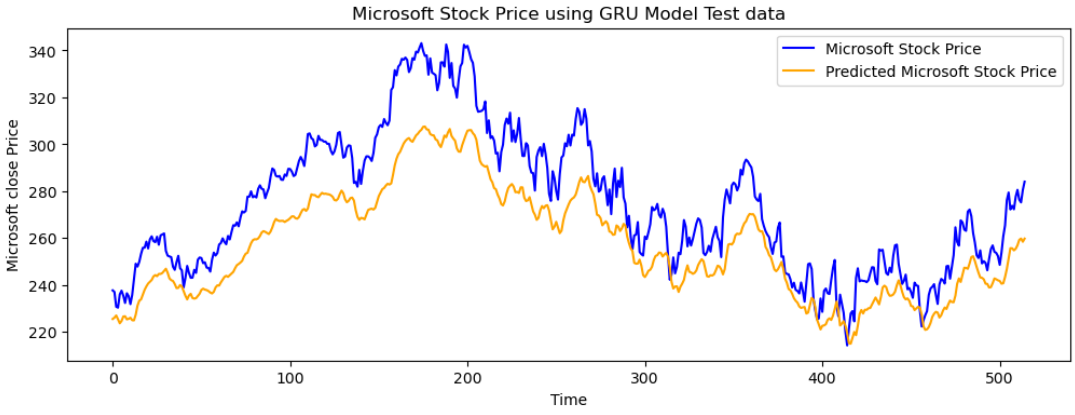


Figure 23

Conclusion

Based on the given performance metrics, it is difficult to make a clear determination on which model performed the best, as different metrics can give different insights into the performance of a model. However, comparing the MAE, RMSE, and R2 scores of the five models, we concluded

that the GRU model performed the best, with relatively low MAE and RMSE, and the highest R2 score. It can also be compared to the LSTM model, which has a slightly lower MSE but a lower R2.

In conclusion, we have discussed many different types of recurrent neural network models for time-series forecasting. These models have shown promising results in accurately predicting future values based on past observations in various applications, including speech recognition, natural language processing, and video analysis. Each model has a unique architecture that combines different types of layers and techniques to capture long-term dependencies and extract relevant features from input data.

Overall, these models have demonstrated high accuracy in time-series forecasting tasks, with low mean squared error and mean absolute error values. However, in the future, it is important to monitor for overfitting and choose appropriate hyperparameters and optimization methods to ensure that the model is generalizing well to new data.

Reinforcement Learning

1. Introduction

Reinforcement learning is a type of machine learning which involves learning through erroneous interactions with the environment. In reinforcement learning, an agent learns to act in a way that will maximize a cumulative reward signal. The agent's objective is to learn an ideal policy that converts states into behaviors that maximize the anticipated cumulative reward over time.
Mind Tracker is a multi-player game developed with Reinforcement learning. It draws inspiration from the well-known game "Among Us." Consider 5 players. One among them will be a psychic which is the AI, who will be in disguise. Each player will have their own skills and a unique ability. The different skills may include speed- where they can complete a work very quickly, ability to solve puzzles, ability to multitask. They also have a separate list of tasks to complete. The tasks depend on the environment. For instance, level 1 might occur on a farm. Plowing the ground, planting seeds, watering crops, harvesting, feeding cows and hens, gathering milk and eggs, etc. are just a few of the different players' goals. Like level 1, level 2 can take place in a battlefield with the goals being to gather weapons, capture the enemies, etc. The players should work as a team to achieve their goals and try to avoid being influenced by psychic.

The role of psychic is to influence the other 4 players and achieve his/her objective which may be building a barn (which includes many small tasks) through them. Each level will be completed if either all the players achieve the assigned tasks without being influenced by the psychic or if the psychic's objectives are achieved through other players.

2. Basic Components

2.1. Actions

The decisions the AI agent can make while playing the game are called actions. When playing a psychic, the agent's actions are focused on manipulating and controlling the other participants. The primary actions consist of:

a) Select a target player to influence: The agent chooses a certain player to concentrate its influence efforts on, considering the player's role, abilities, and goals.
b) Choose an influence technique: The agent decides on an influence technique, such as persuasion, deception, or intimidation, depending on the target player's actions and the agent's understanding of possible defenses.
c) Make use of psychic powers: The agent makes use of psychic powers to influence the surroundings, cause disturbance, or gain an advantage over other players.
d) Monitor and assess player behavior: The agent continuously monitors the other players to learn about their behaviors, communication styles, and level of achievement of their goals. The agent can prioritize its tasks and modify its influence techniques with the aid of this information.

2.2. States

States serve as a representation of the game's current state, storing essential data about the psychic, other players, and the game's overall development. The following are important state variables:

a) Psychic's objectives and progress: The agent's aims and the extent to which they have been accomplished.
b) Information on the aims and development of other players: Details on the ambitions and development of other players.
c) Psychic's influence success rate: an indicator of how effective the agent has been at persuading other players that can be used to assess the efficacy of a strategy.

2.3. Rewards

a) Reward points will be given to the psychic if it is able to influence other players.
b) Bonus points will be awarded to the psychic if it is able to influence a player before they complete their first task.
c) Reward points will be awarded to the psychic if it is able to successfully complete its objective.
d) Negative points for psychic if the attempt to influence is unsuccessful.
e) Negative points for psychic if other players achieve their objective.

3. Hypothesis for action rule, Reward and State distribution

We can use a deep Q-network (DQN) or a Proximal Policy Optimization (PPO) algorithm to learn the optimal strategy for the psychic.
DQN algorithm learns the Q- function which will analyze the environment and the current state. It takes the input as the current state of the game and gives the expected reward for each action taken in that state.
PPO works like DQN. Here, input is the state of the game, and the output is the probability of taking each possible action in that state. It will help the psychic to maximize not only the immediate rewards but also the long-term rewards. In this way it helps the psychic to choose the best policy that can maximize the rewards in the long run.
ReLU activation function can be used in the hidden layers to introduce a non-linearity component to model the complex relationships between the game states and actions. An LSTM layer (Long Short-term memory) can be used in the neural network, so that the psychic remembers important details about the states and actions of the game. An attention mechanism can also be used so that the psychic can focus on the important details from the state of the game.

4. Final reward

The final reward is a cumulative total of all the positive and negative rewards. The objective of the psychic is to maximize the positive rewards by successfully influencing other players and achieving its objectives through them and minimize the negative rewards by lowering unsuccessful influence attempts and prevent other players from completing their task.

5. Analysis Procedure

a) The first step includes deciding the environments, states, action, task etc for the game. Once these have been finalized, it is important to preprocess the data into a format that can be used for Reinforcement Learning Algorithms.

b) After Preprocessing the data, we must initialize the neural network with random weights. This neural network will help the psychic take the best action for a given state.
c) Setting the exploration rate, learning rate, and discount factor: Learning rate determines how rapidly the neural network modifies its weights in response to rewards. The importance of future rewards in comparison to immediate rewards is determined by the discount factor. Exploration rate balances the best-known action and exploring new action.

d) Based on the exploration rate, the psychic chooses either the best-known action or explores new action best suited for that state.

e) After the psychic chooses an action, the new state and reward transitions should be updated in PPO and DQN along with updating network weight.

f) A batch of the transitions will be sampled, and network weights will be updated with the goal of minimizing the difference between predicted and actual Q values.

g) The process is repeated until the psychic attains a satisfactory performance.

6. Conclusion

In conclusion, the creation of an AI agent to assume the role of the psychic in a multiplayer, asymmetric psychological strategy game offers a demanding and engaging gaming experience for human participants. The AI agent can successfully learn and adjust its methods to a range of game conditions and player behaviors by applying reinforcement learning algorithms like Deep Q-Network (DQN) or Proximal Policy Optimization (PPO) and concentrating on the correlations between actions, states, and rewards. An appropriate neural network design and an intricate balance between exploration and exploitation allow the agent to continuously enhance its performance and offer fun gameplay. Because of this AI-driven approach to the psychic function, a novel gameplay experience is produced that challenges the conventions of multiplayer strategy games.