Stock Prediction

Using

Recurrent Neural Network

NEURAL NETWORKS COURSE

Dr. Katan Foroush

Tahsin Ilkhas Zadeh 40422034 Zeinab Khosravi 99422067

Stock Prediction Using Neural Network

Machine Learning became very useful to the Stock Market Forecasting over the last years, and today, many investment companies are using Machine Learning to make decisions in the Stock Market.

The successful prediction of a stock's future cost could return noteworthy benefit.

Stock Prediction Using Neural Network

- Different types of RNNs are taken in forecasting stock trend in the previous years.

- In this approach, a stock price prediction method is proposed with Recurrent Neural Networks (RNN) models.

Datasets

The datasets used to fit to models:

Apple dataset

- This dataset contains Apple's (AAPL) stock data for the last 10 years (from 2010 to date) reached from Nasdaq

Google dataset

- This dataset contains Google's stock data for the last years (to 2017)

Understanding the Problem

Understanding the Problem

forecast stock price is about use Sequential data, where the past data matters and influence directly in our predictions.

We are going to give to our model a Stock Price history of datasets, it will work on this data to identify patterns and make a lot of calculus to, and give us the predictions.

Understanding the Problem

- In addition to the start and end values given by the start and end dates, values such as interval and period can be given.

Understanding the Problem

Since the data is sent as a list of candlesticks, it has 6 values:

- Date
- Open
- Close
- High
- Low
- Volume

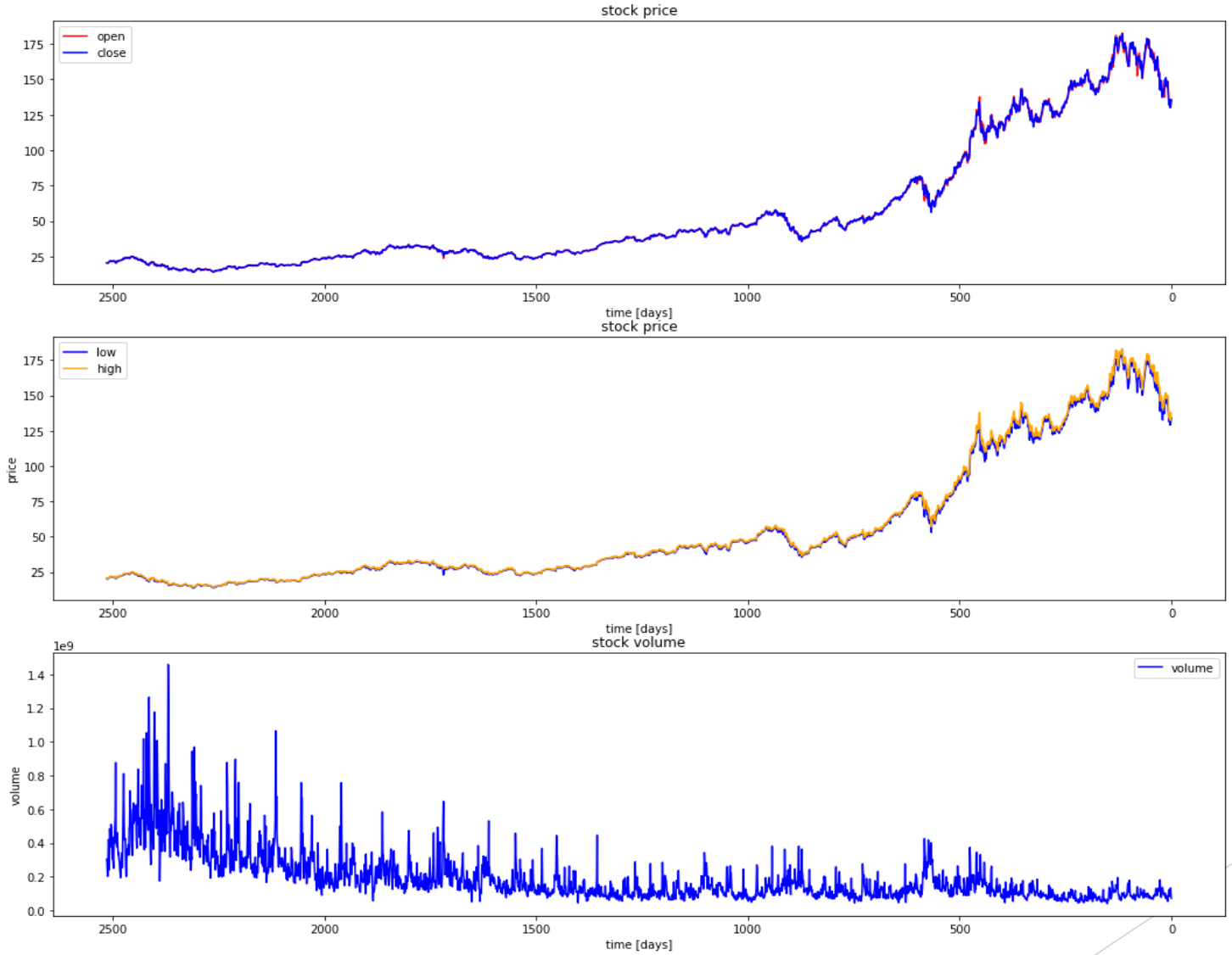
Understanding the Problem

The Apple dataset head lines:

Date	Close/Last	Volume	Open	High	Low
2022-06-22	$135.35	73409230	$134.79	$137.76	$133.91
2022-06-21	$135.87	81000490	$133.42	$137.06	$133.32
2022-06-17	$131.56	134520300	$130.065	$133.079	$129.81
2022-06-16	$130.06	107961500	$132.08	$132.39	$129.04
2022-06-15	$135.43	91532970	$134.29	$137.34	$132.16

Understanding the Problem

Apple open/close and high/low graph



Understanding the Problem

The google dataset head lines:

Date	Open	High	Low	Close	Volume
2012-01-03	325.25	332.83	324.97	663.59	7,380,500
2012-01-04	331.27	333.87	329.08	666.45	5,749,400
2012-01-05	329.83	330.75	326.89	657.21	6,590,300
2012-01-06	328.34	328.77	323.68	648.24	5,405,900
2012-01-09	322.04	322.29	309.46	620.76	11,688,800

Understanding the Problem

Understanding the Problem

The time interval inside each candle, that is from open to close, is a few minutes, called the interval.

With the help of the period value, it is possible not to give the start, end dates, and for example request the data of the last 24 hours.

The risk of the stock

We are going to analyze the risk of the stock.

In order to predict the risk, we need to inspect the daily changes of the stock, and not just its absolute value

Then find the daily return of the stock on average.

Moving Average (MA) Lines

Moving averages are one of the most commonly used technical indicators in stock, futures and forex trading.

Market analysts and traders use moving averages to help identify trends in price fluctuations, smoothing out the noise and short-lived spikes (from news and earnings announcements, for example) for individual securities or indexes.

Most Commonly-Used Moving Averages

The 5 -, 10 -, 20 - and 50 - day moving averages are often used to spot near-term trend changes.

Changes in direction by these shorter-term moving averages are watched as possible early clues to longer-term trend changes.

Crossovers of the 50-day moving average by either the 10-day or 20-day moving average are regarded as significant.

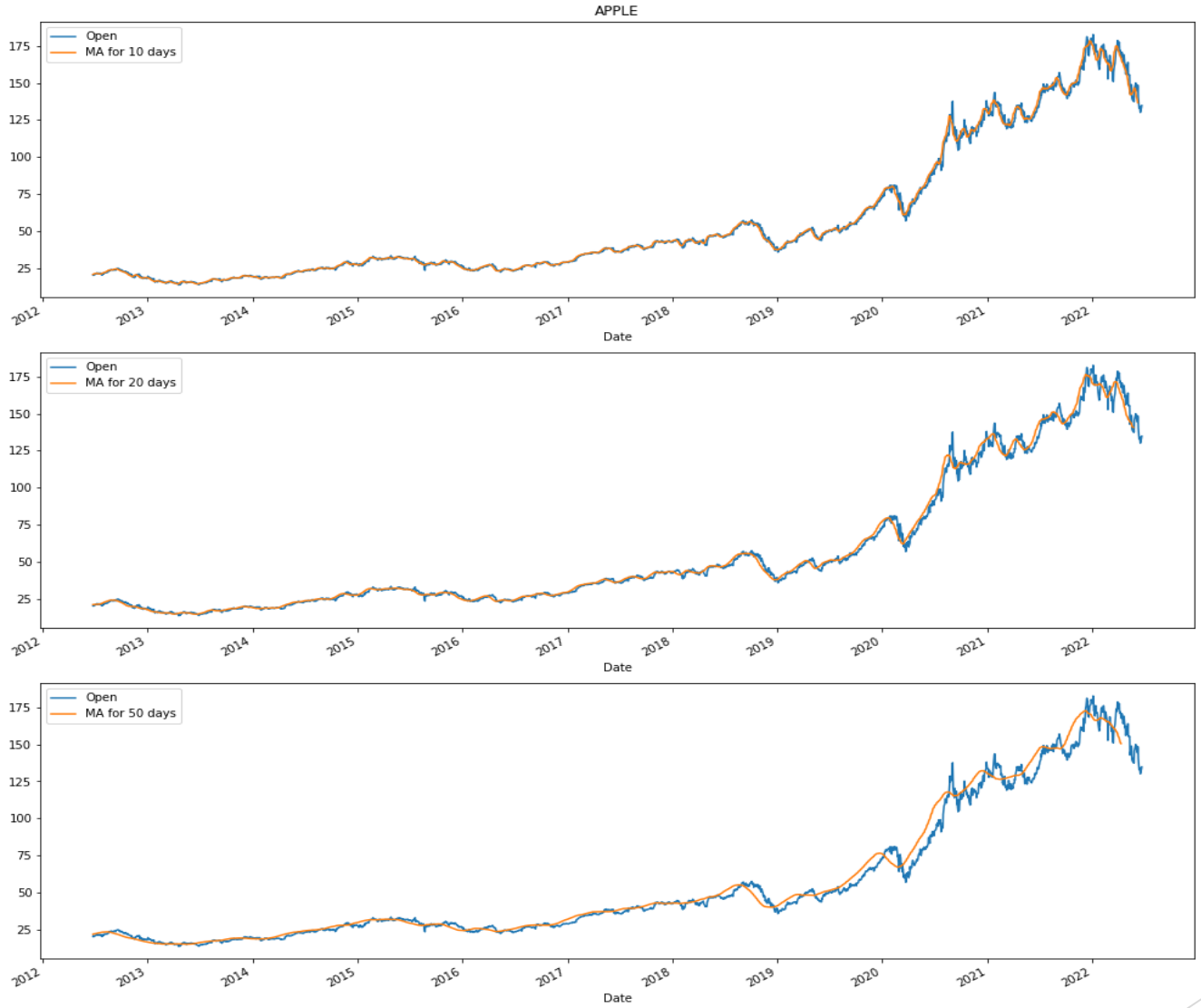
The 10-day moving average plotted on an hourly chart is frequently used to guide traders in intraday trading.

Types of Moving Averages

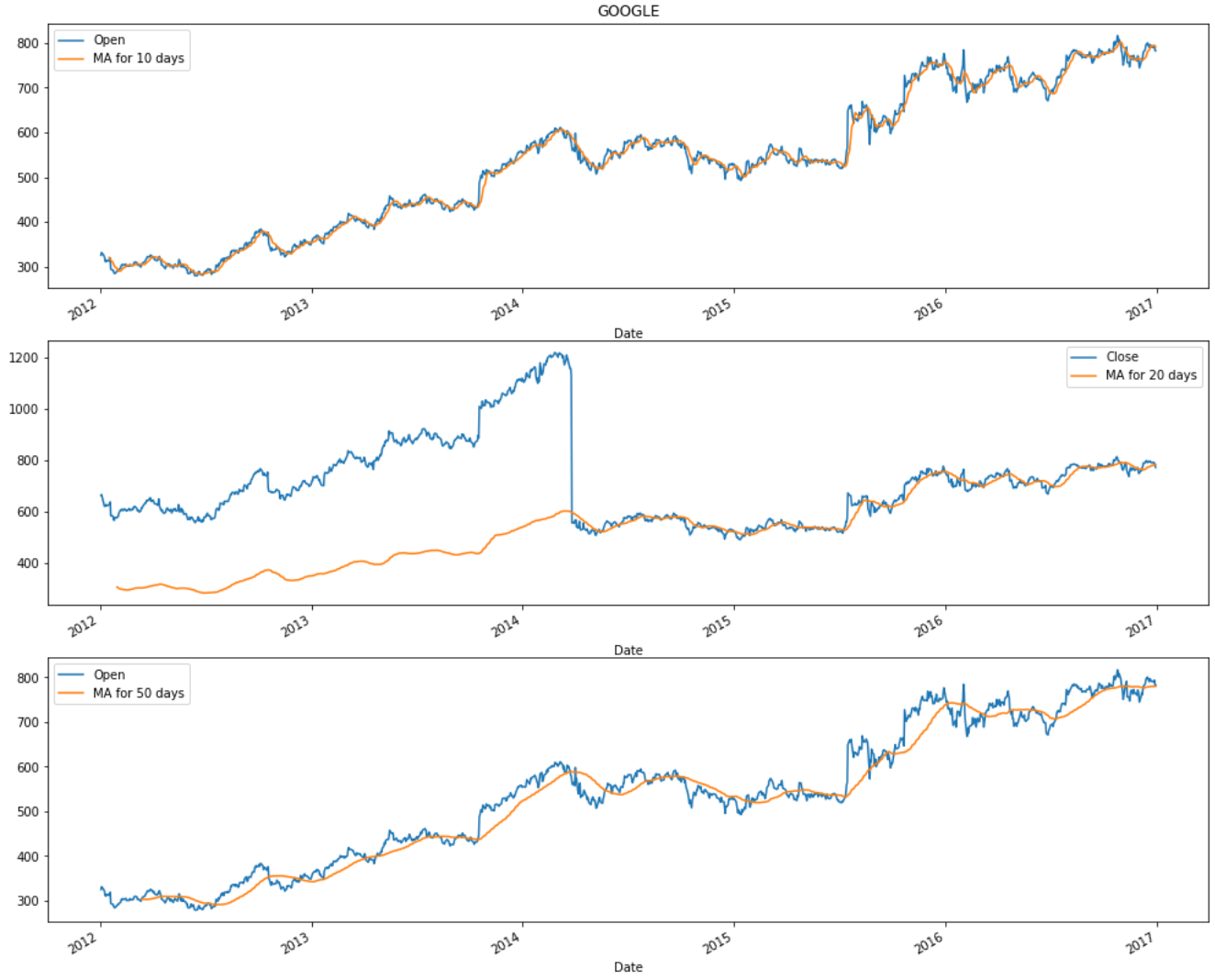
Most moving averages are some form of the simple moving average (SMA), which is the average price over a given time period

As a general guideline, if the price is above a moving average, the trend is up. If the price is below a moving average, the trend is down.

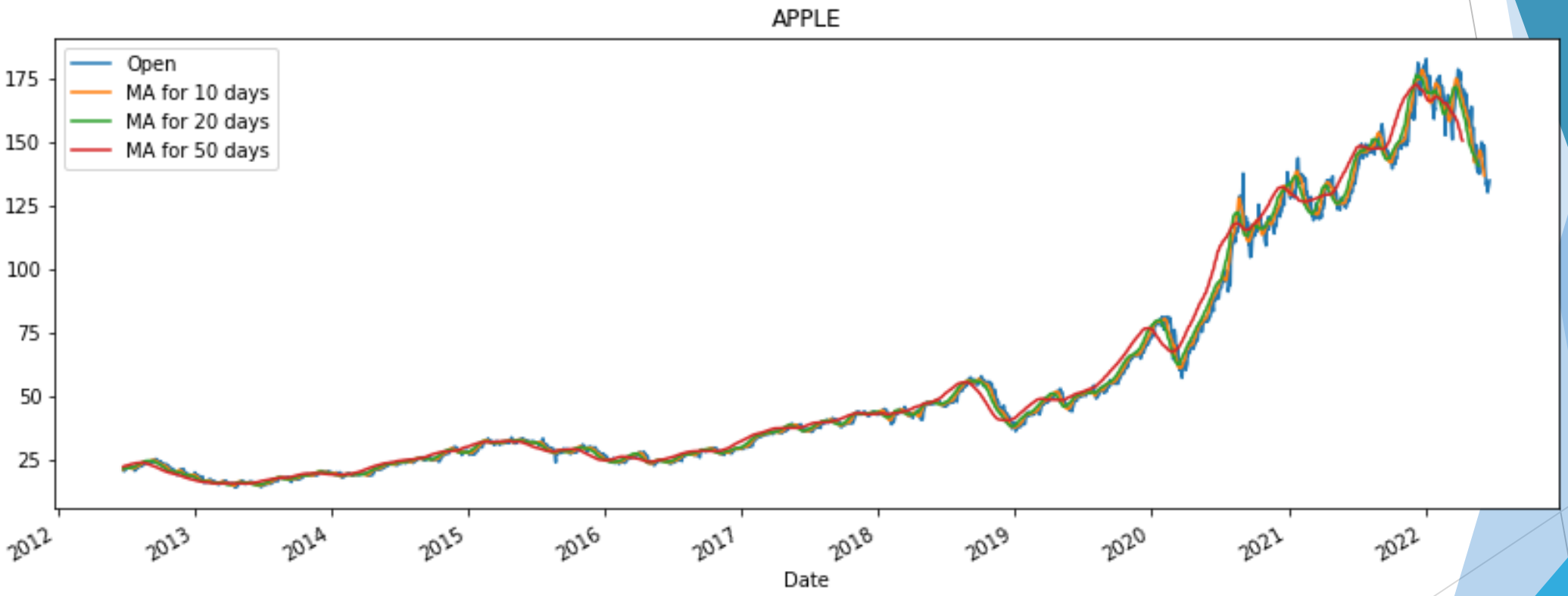
Moving Averages for Apple dataset



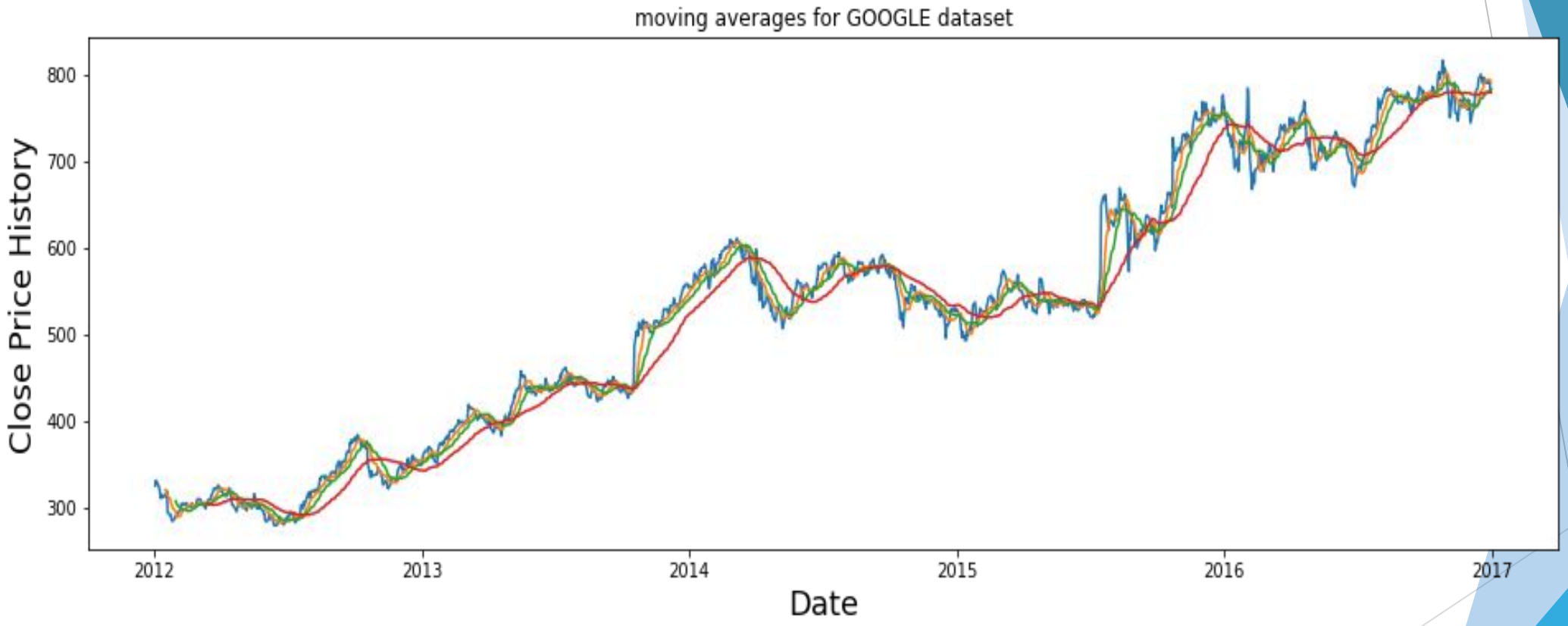
Moving Averages for Google dataset



Moving Averages for Apple dataset



Moving Averages for Google dataset



Models

RNN models

RNN Is a type of artificial neural network where connections between nodes form a sequence. This allows temporal dynamic behavior for time sequence.

Long Short Term Memory models

From the simulation results, it can be noted that using RNN models i.e. LSTM, and BI-LSTM with proper hyper-parameter tuning, can forecast future stock trend with high accuracy.

Recurrent Neural Network (RNN) models

we have tested 2 kind of models for each dataset:

1. Long Short Term Memory (LSTM) model

2. Bidirectional long short term memory, Bi-LSTM model

LSTM model

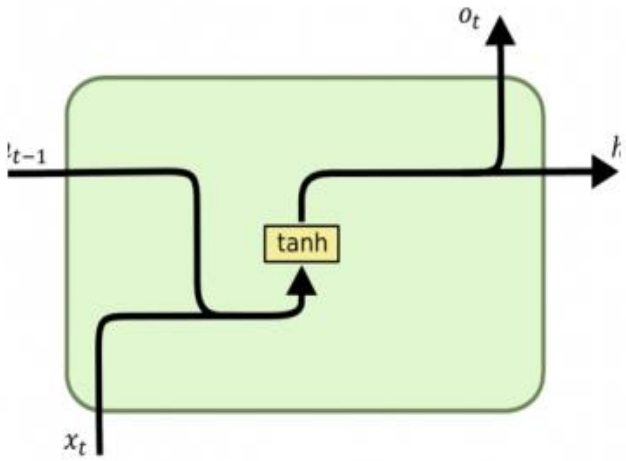
LSTM Models

There are 3 types of vanilla recurrent neural network:

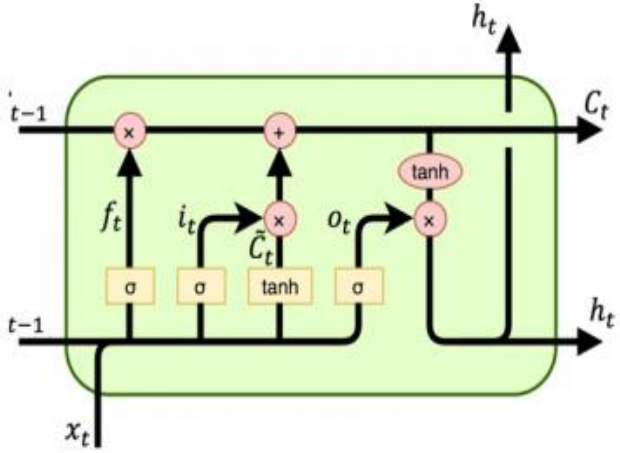
- the simple (RNN)
- gated recurrent unit (GRU)
- long short term memory unit (LSTM).

LSTM Models

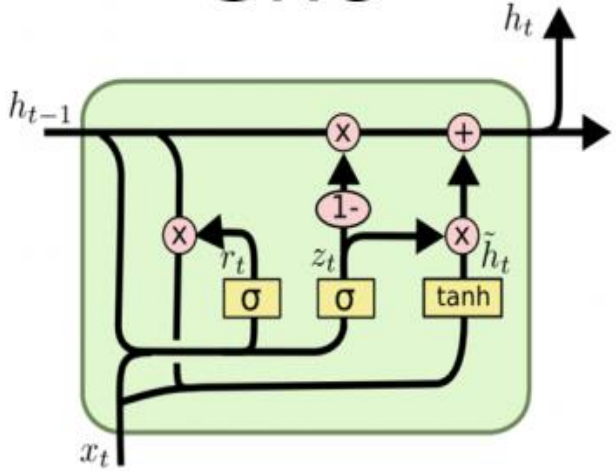
RNN



LSTM



GRU



LSTM Models

In regular RNN, the problem frequently occurs when connecting previous information to new information

Long short term memory networks, usually called LSTM

are a special kind of RNN.

They were introduced to avoid the

long-term dependency problem.

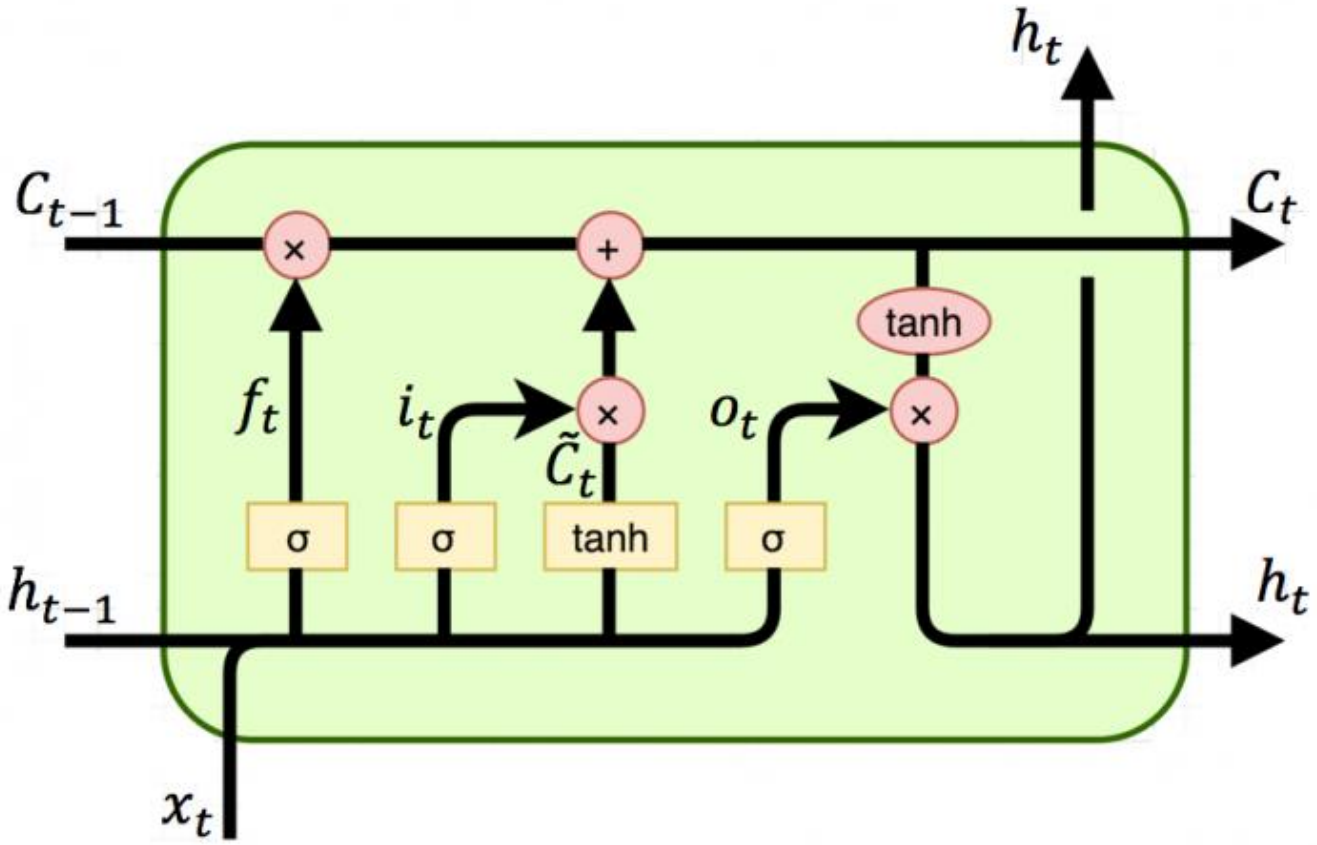
LSTM Models

- Recurrent networks have an **internal state** that can represent context information.
- They **keep information about past inputs** for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data.

LSTM Models

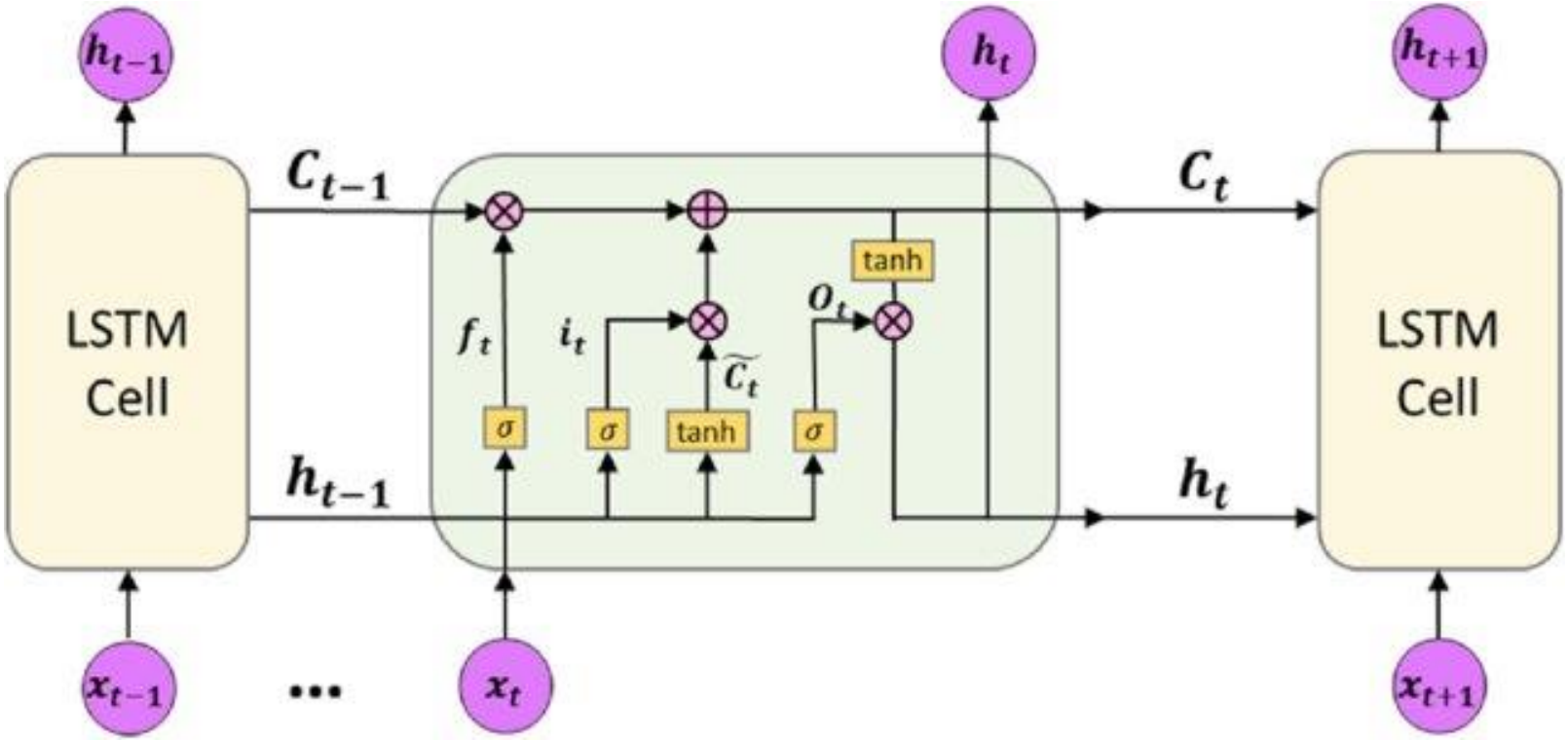
Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of **learning order dependence** in sequence prediction problems.

LSTM Models



LSTM core

LSTM Models



A LSTM layer structure for a time step, with a detailed calculation illustration shown in the LSTM cell at time step t.

LSTM Models

Only the hidden state is passed into the output layer. The memory cell is internal.

LSTMs have three types of gates that control the flow of information.

- input gates
- forget gates
- output gates

LSTM Models

Only the hidden state is passed into the output layer. The memory cell is internal.

The hidden layer output of LSTM includes the hidden state and the memory cell

LSTMs can alleviate vanishing and exploding gradients.

Bi-LSTM model

BI-LSTM Models

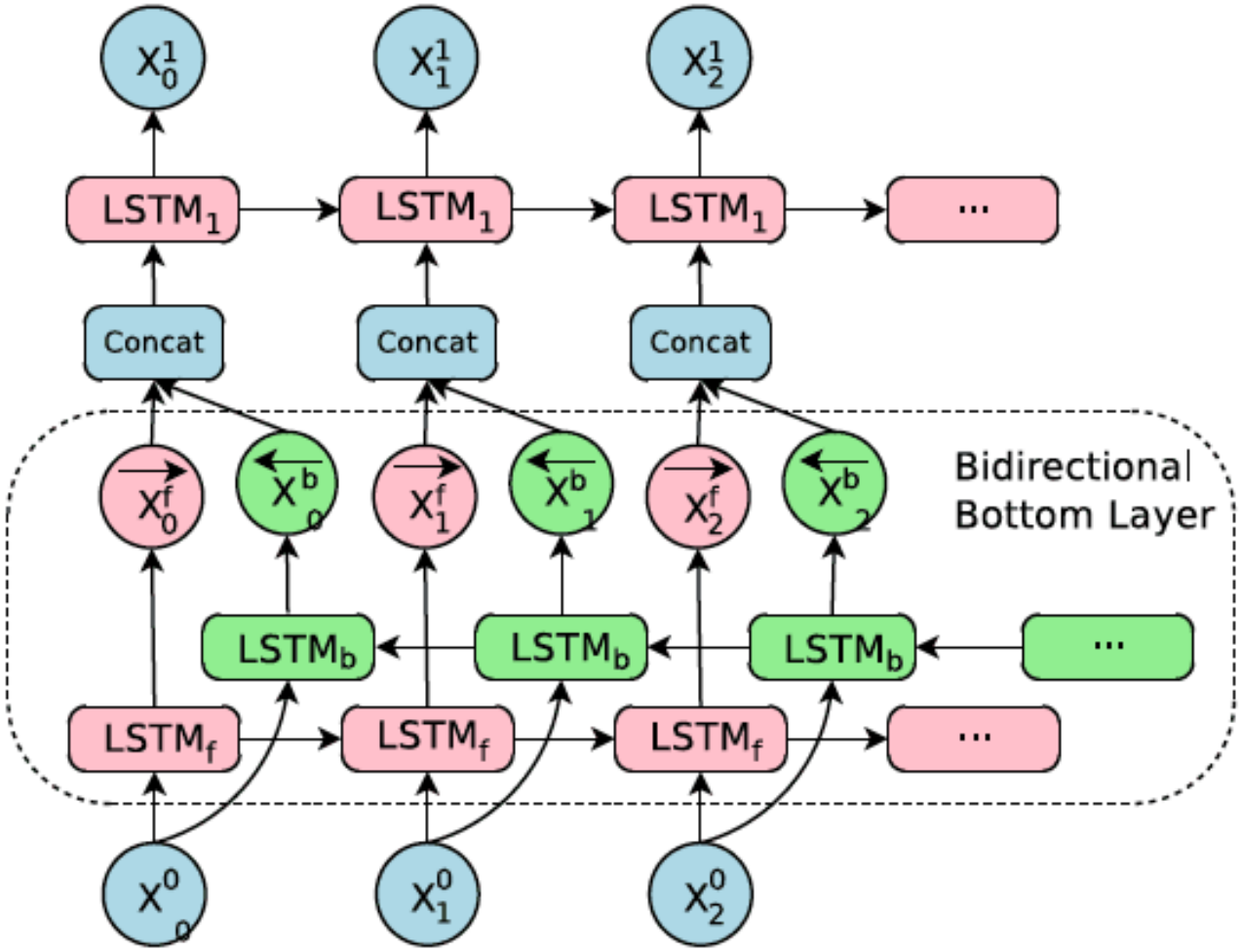
Bidirectional long-short term memory (bi-lstm)

the process of making any neural network to have the sequence information in both directions

backwards (future to past)

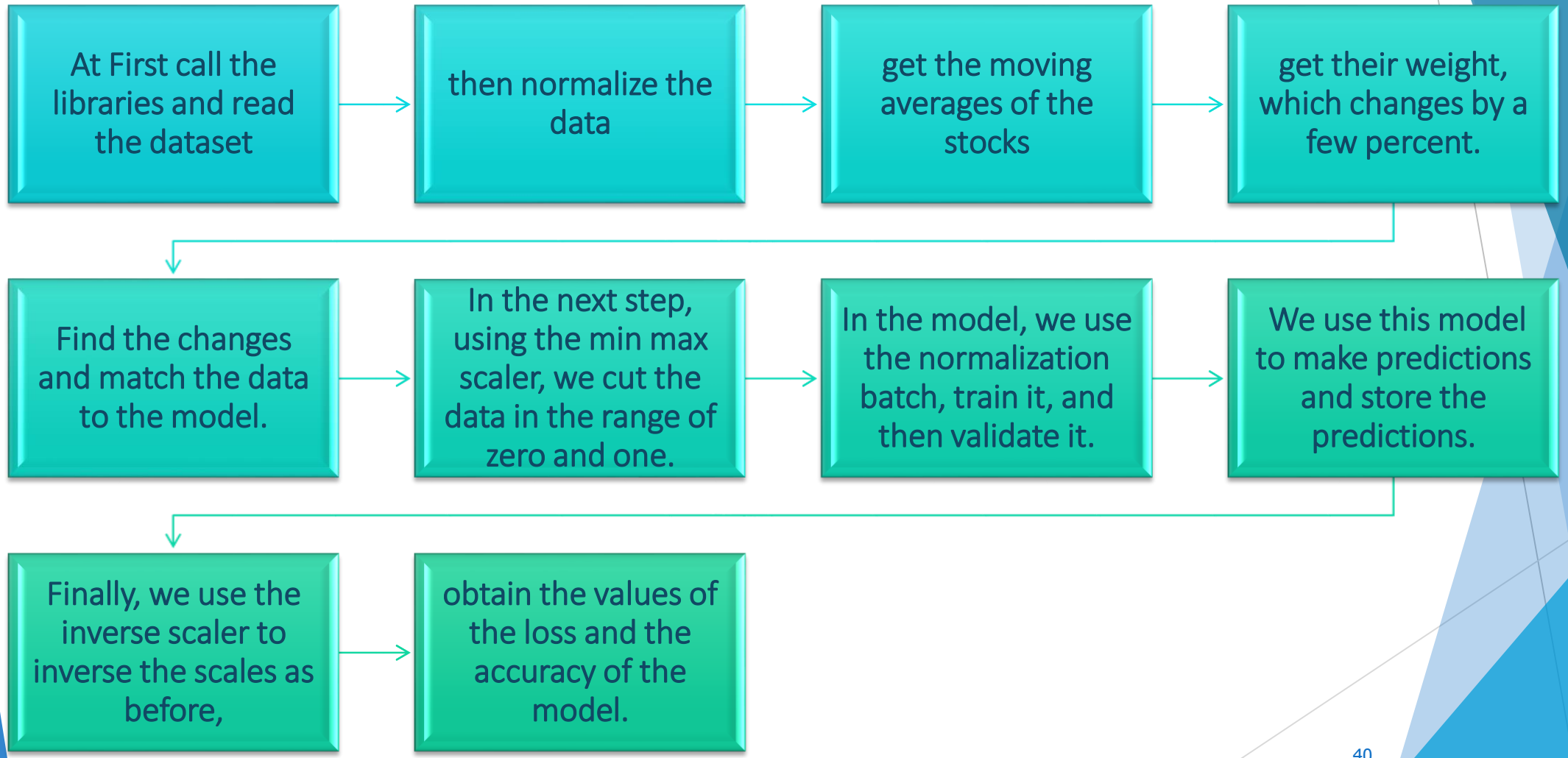
Or forward (past to future)

BI-LSTM Models



The procedure

The procedure



The model setups

define the shape of the input dataset:

- Number of time steps, the number of lags in the data frames we set in Step #2.

define the number of units, and the number of hyper parameters of the LSTM.

- The higher number, more parameters in the model.

The model setups

Define the dropout rate, which is used to prevent overfitting.

Specify the output layer to have a linear activation function.

Then we also define the optimization function and the loss function.

- Again, tuning these hyper parameters to find the best option would be a better practice.

Apple Dataset

Models and Results

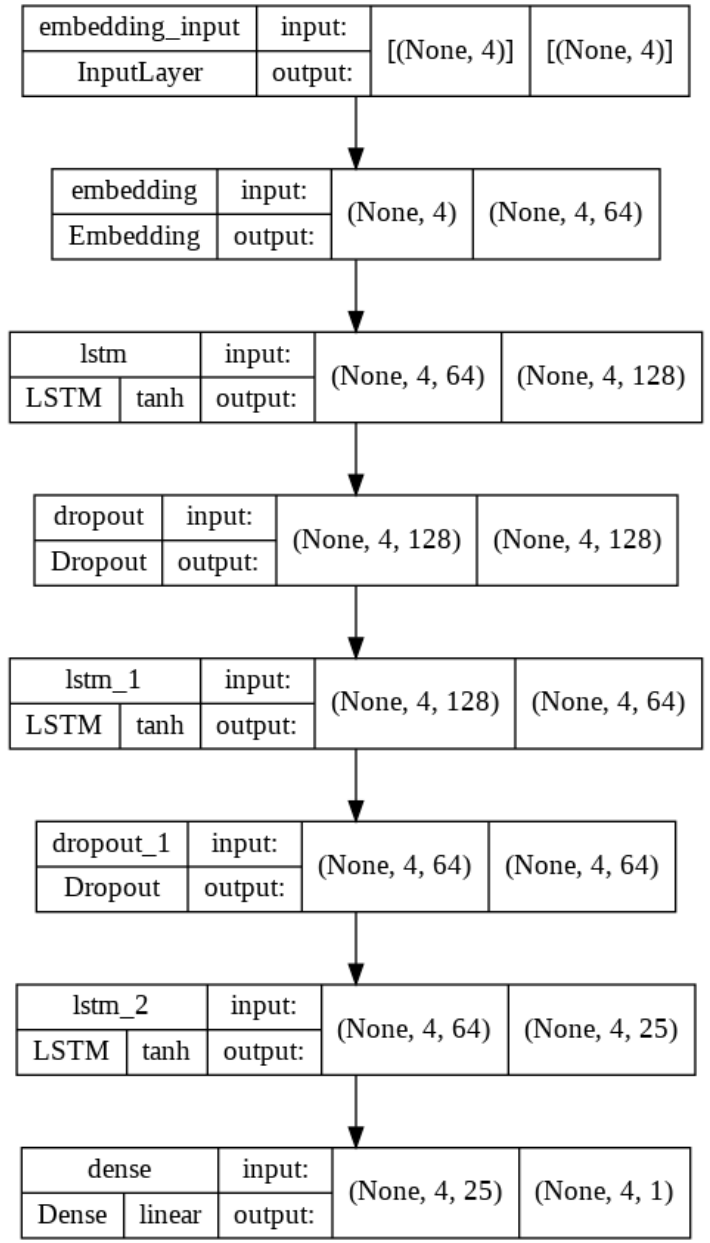
The LSTM Model

LSTM Model NO.1 for Apple dataset

```
# Build the LSTM model
model = Sequential()
model.add(Embedding(15212,64,input_length=x_train.shape[1]))
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               dropout=0.2, recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(Dropout(0.6))
model.add(LSTM(64,  return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
               recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(Dropout(0.6))
model.add(LSTM(25 , return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
               recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
```

LSTM Model NO.1



LSTM Model NO.1 for Apple dataset

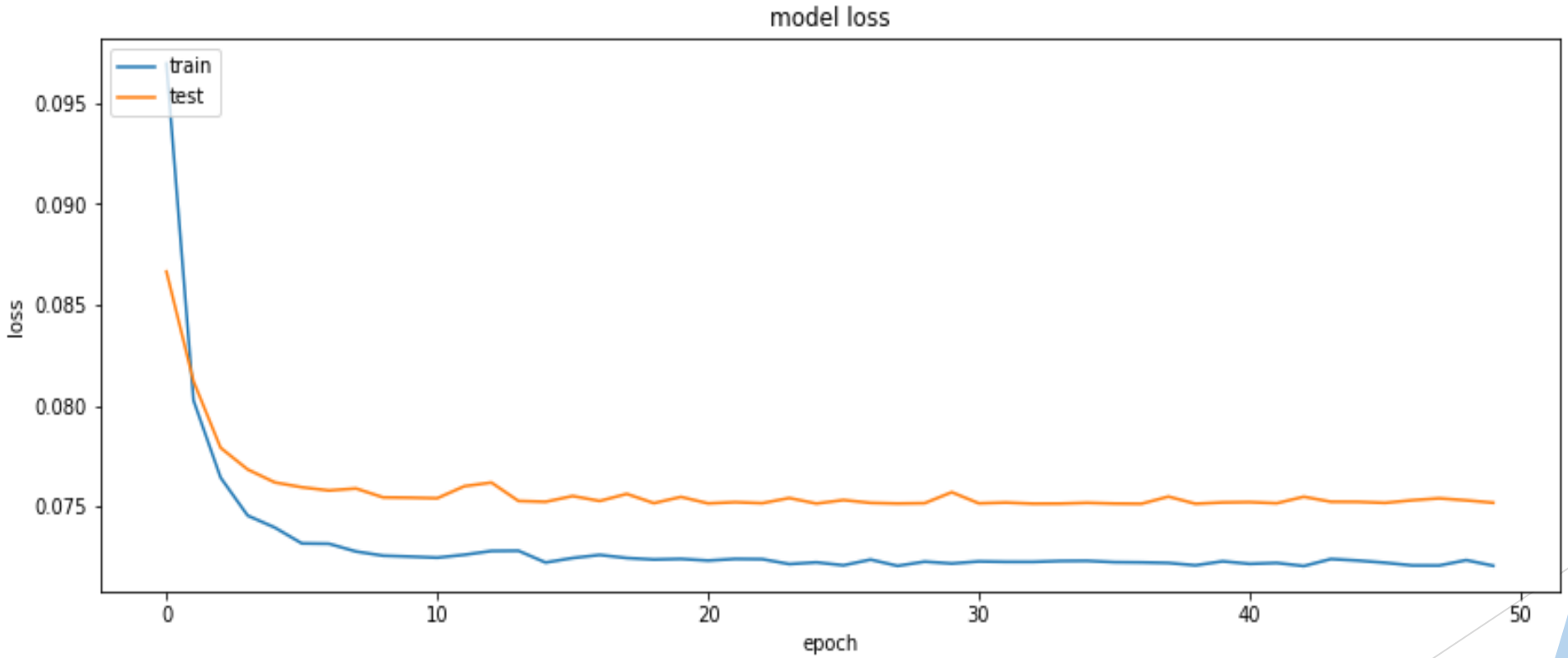
We trained our model with the training data over bellow parameters :

- epochs = 50
- with a batch size = 32
- optimizer= Adam
- loss= mean squared error (MSE)
- Validation split=0.25
- Drop out = 0.6

Accuracy of LSTM model no.1 = 0.00088417332153767 35

LSTM Model NO.1 for Apple dataset

- The Loss graph for LSTM model no.1



LSTM Model NO.2 for Apple dataset

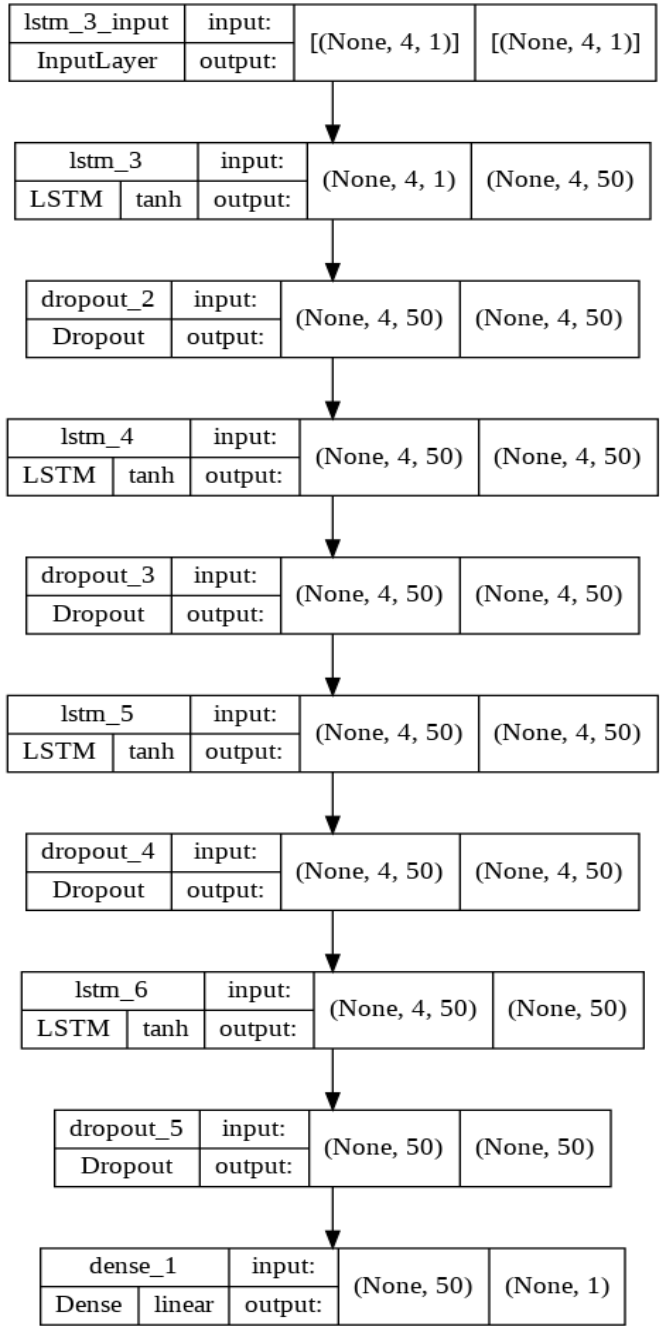
We changed model layer architecture and parameters to compare the results and fitted it over same condition on apple dataset. So, the second model is as below:

```
# Building Model:
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(units=50, return_sequences=True,input_shape=(x_train.shape[1], 1)))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.LSTM(units=50, return_sequences=True))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.LSTM(units=50, return_sequences=True))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.LSTM(units=50))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(units=1))
model.summary()
```

LSTM Model NO.2

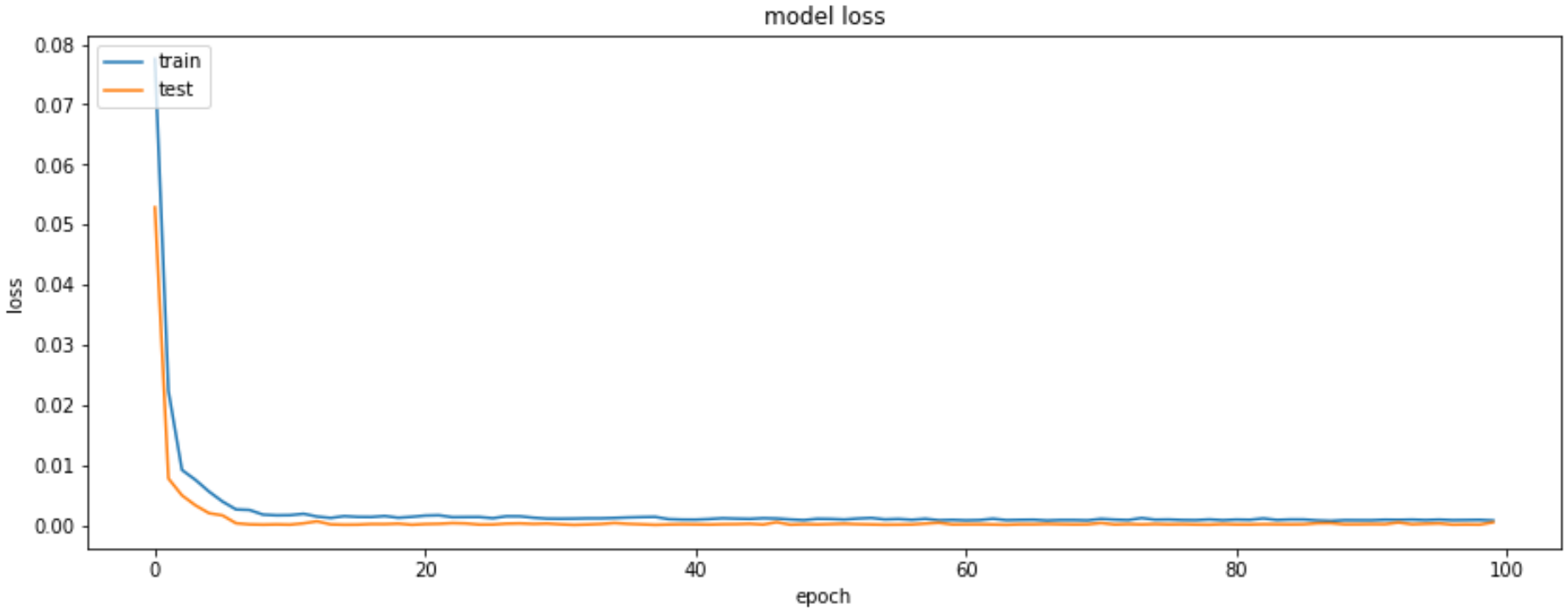
The Accuracy of LSTM no.2

0.000884173321537673 5



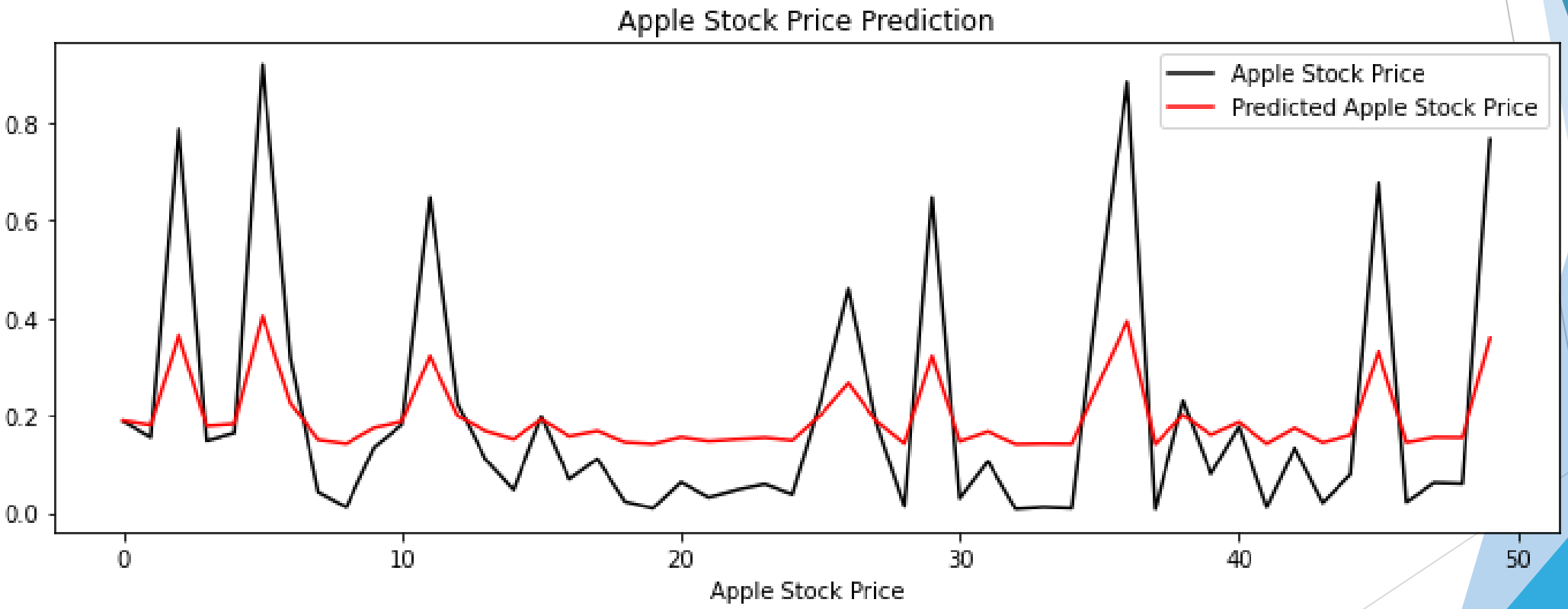
LSTM Model NO.2 for Apple dataset

- The Loss graph for LSTM model no.2



LSTM Model NO.2 for Apple dataset

- The prediction



The Bi-LSTM Model

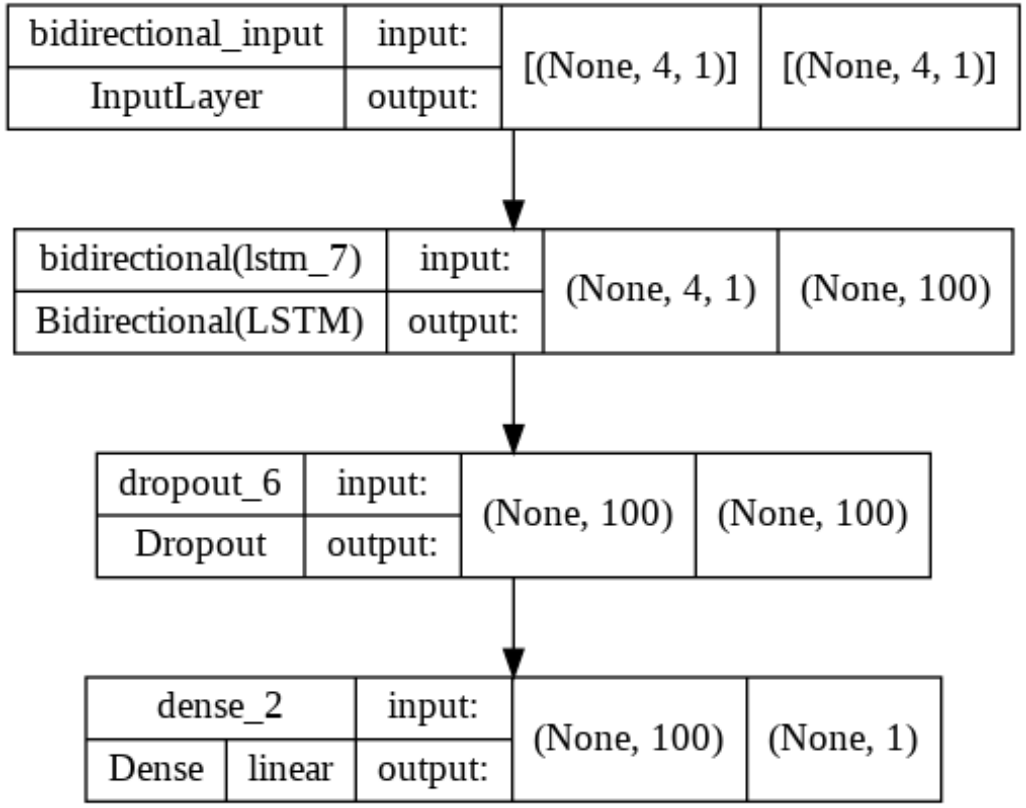
Bi-LSTM Model for Apple dataset

- The bidirectional model used for apple dataset:

```
# define model
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='tanh'), input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.build((x_train.shape[1],1))
model.add(Dense(1))
```

Bi-LSTM Model for Apple dataset

- The bidirectional model architecture



Bi-LSTM Model for Apple dataset

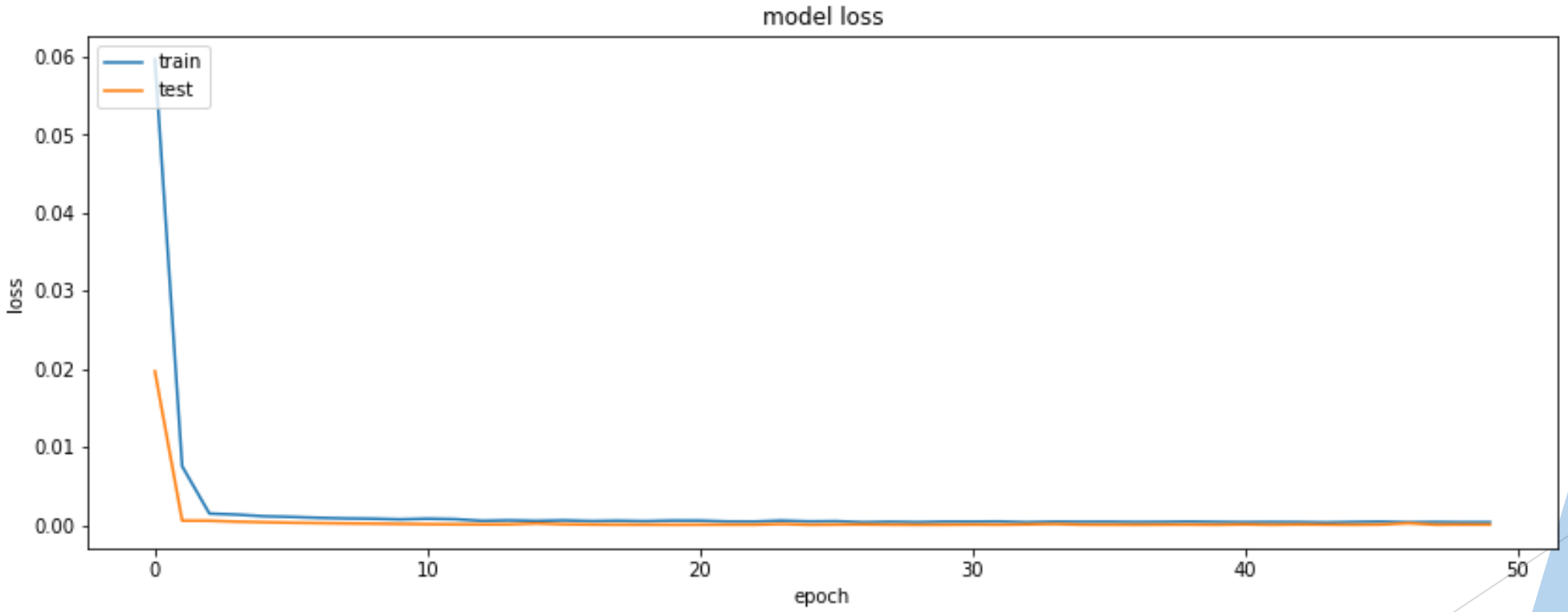
We trained our model with the training data over bellow parameters :

- epochs = 50
- with a batch size = 32
- optimizer= Adam
- loss= mean squared error (MSE)
- Validation split=0.2
- Drop out = 0.2

Accuracy of Bi-LSTM model = 0.0008285004296340048

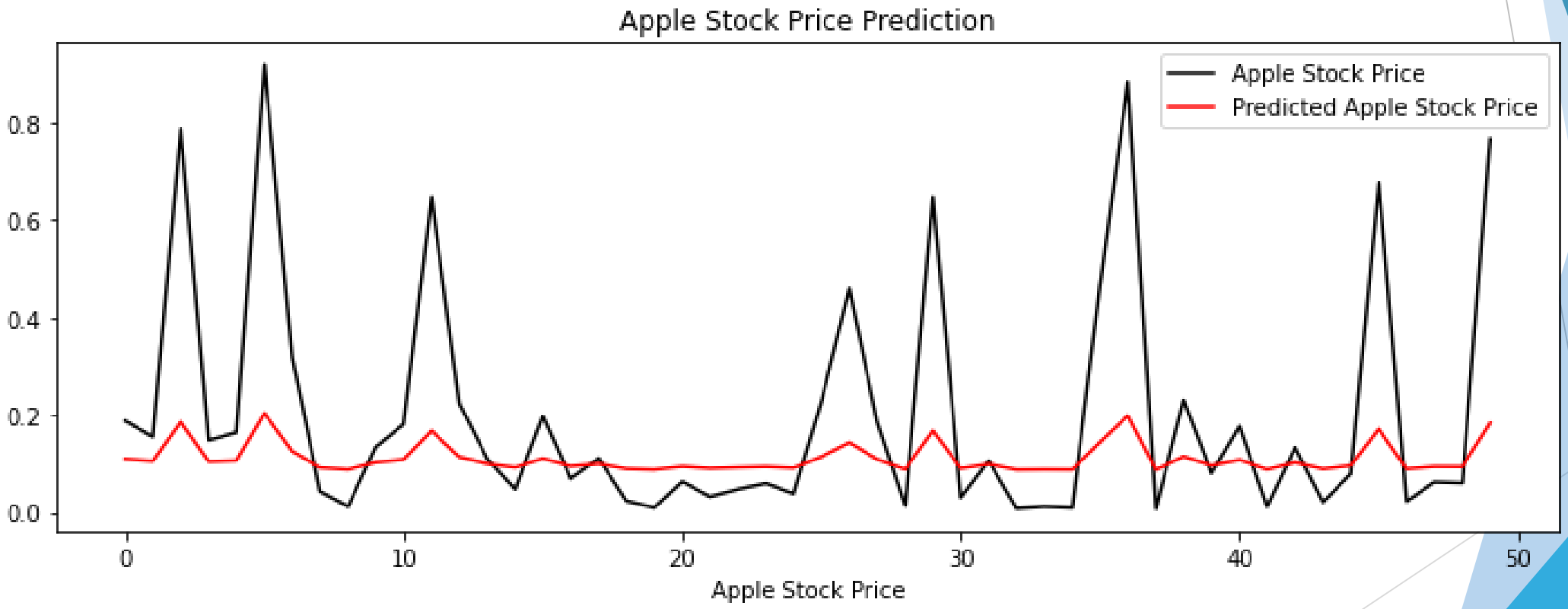
Bi-LSTM Model for Apple dataset

- The Loss graph



Bi-LSTM Model for Apple dataset

- The prediction



Google Dataset

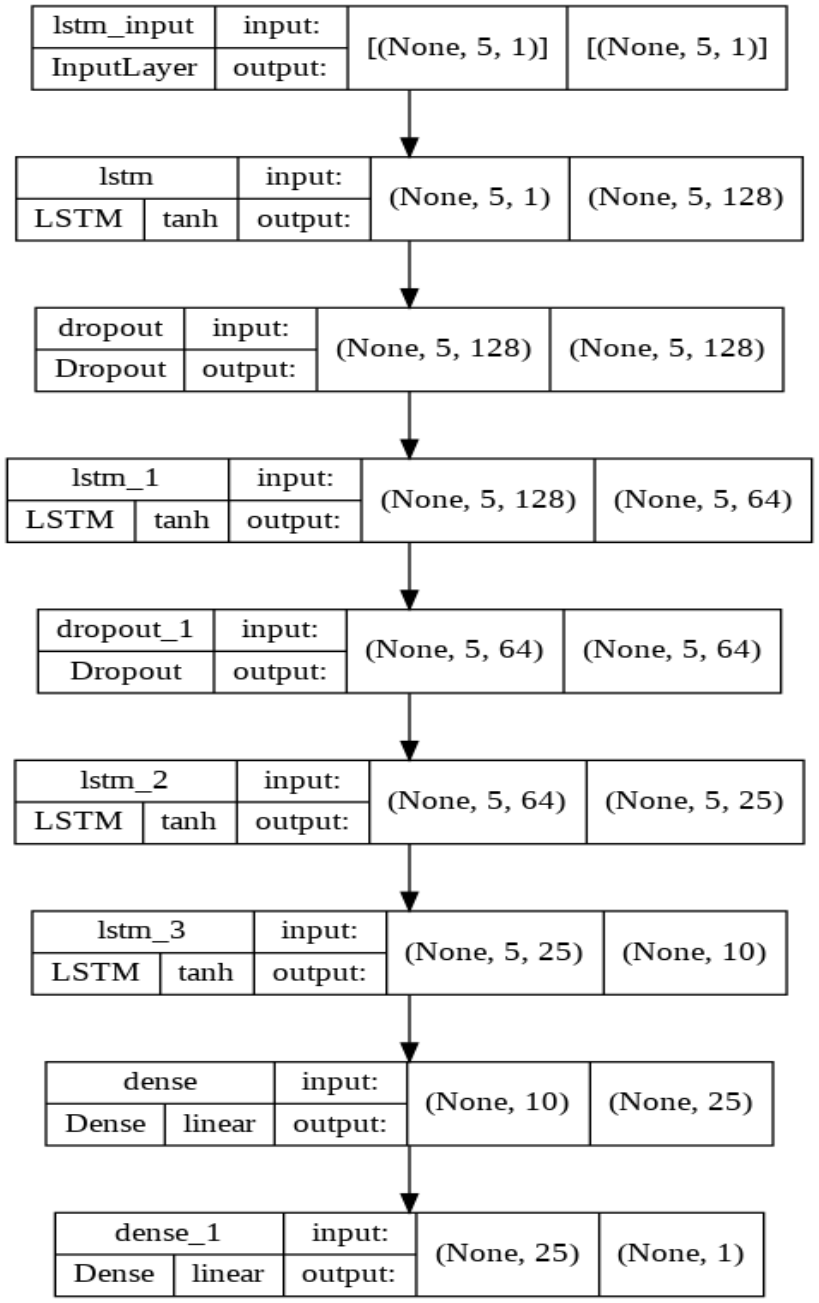
Models and Results

The LSTM Model

LSTM Model NO.1 for Google dataset

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(Dropout(0.6))
model.add(LSTM(64,  return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
               recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(Dropout(0.6))
model.add(LSTM(25 , return_sequences=True, input_shape= (x_train.shape[1], 1),
               activation='tanh', recurrent_activation='sigmoid', use_bias=True,
               kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal',
               bias_initializer='zeros', unit_forget_bias=True,
               kernel_constraint=None, recurrent_constraint=None, bias_constraint=None,
               recurrent_dropout=0.2,
               return_state=False, go_backwards=False,))
model.add(LSTM(10, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

LSTM Model NO.1



LSTM Model NO.1 for Google dataset

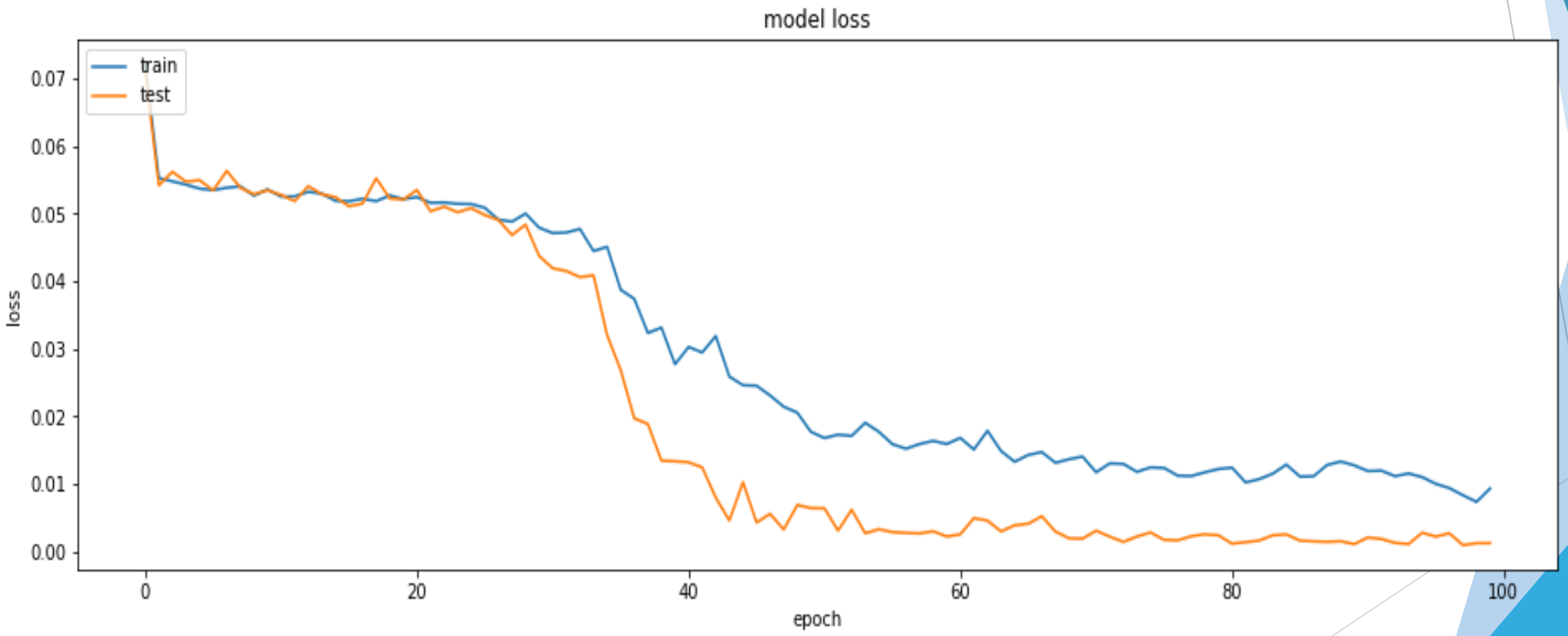
We trained our model with the training data over bellow parameters :

- epochs = 100
- with a batch size = 64
- optimizer= Adam
- loss= mean squared error (MSE)
- Validation split=0.10
- Drop out = 0.6

Accuracy of LSTM model no.1 = 0.0014749262481927872

LSTM Model NO.1 for Google dataset

- The Loss graph for LSTM model no.1



LSTM Model NO.2 for Google dataset

We changed model layer architecture and parameters to compare the results and fitted it over same condition on apple dataset. So, the second model is as below:

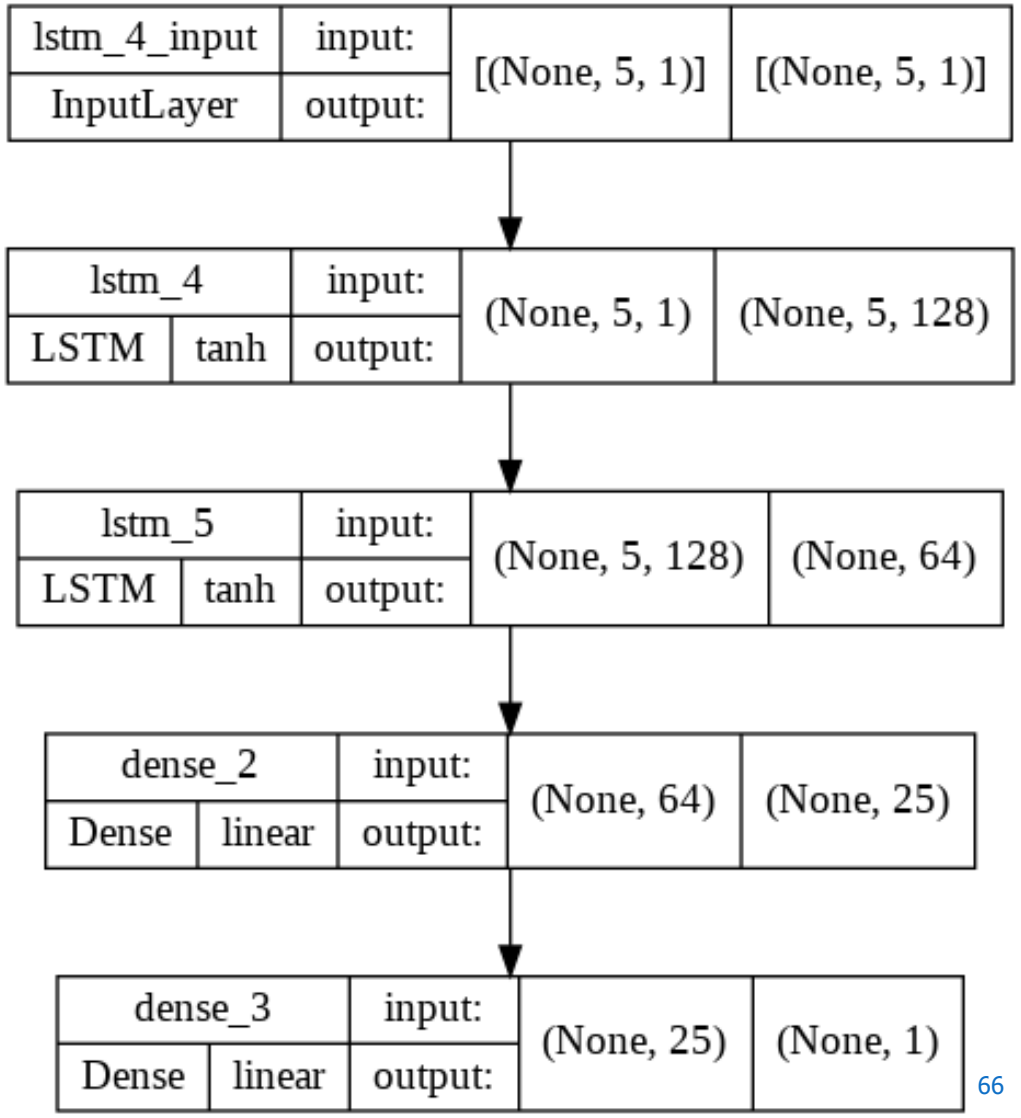
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.summary()
```

LSTM Model NO.2 for Google dataset

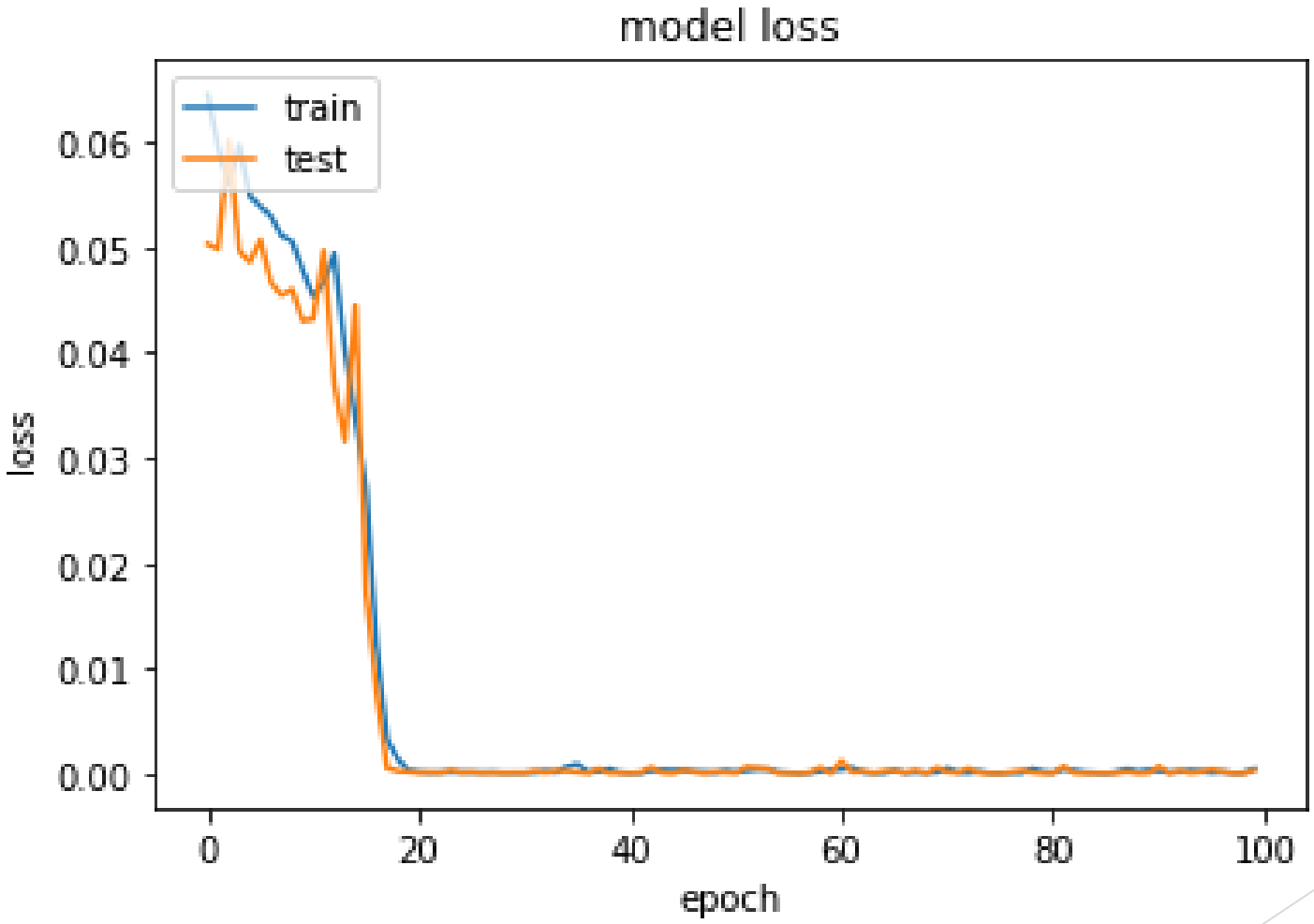
The Accuracy of LSTM no.2

0.0017699114978313446



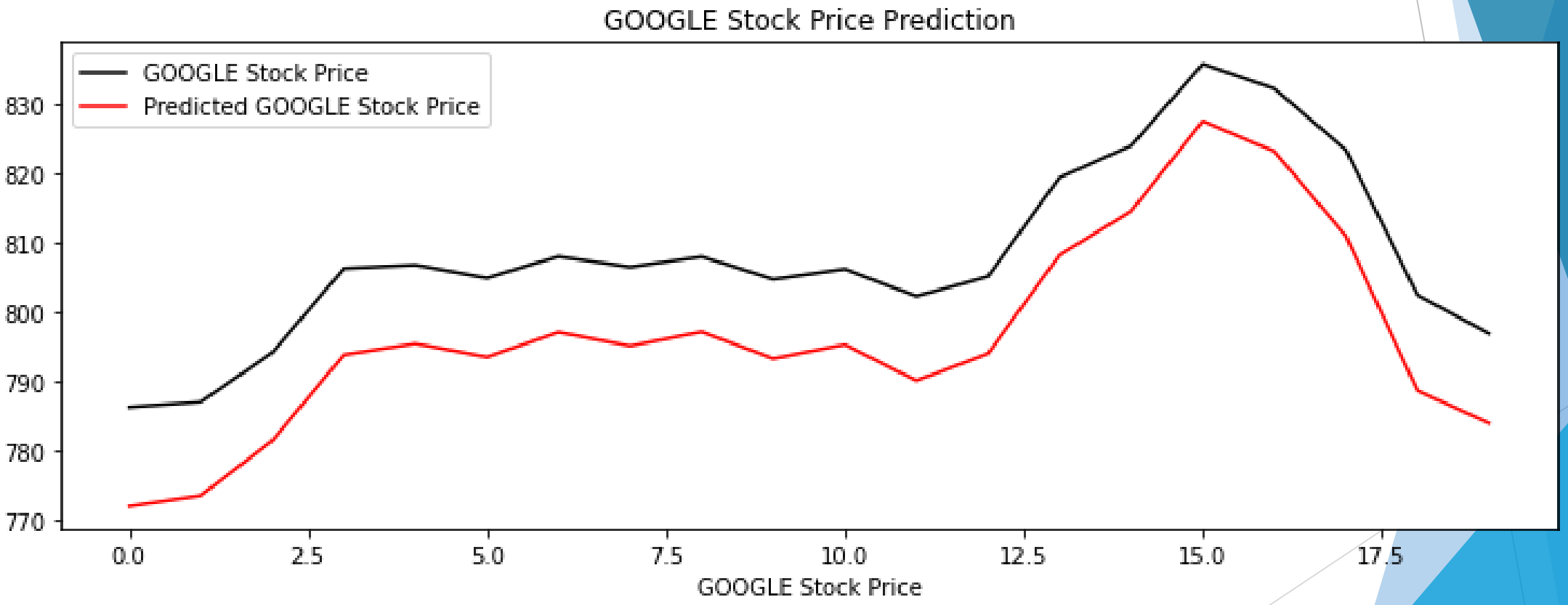
LSTM Model NO.2 for Google dataset

- The Loss graph for LSTM model no.2



LSTM Model NO.2 for Google dataset

- The prediction



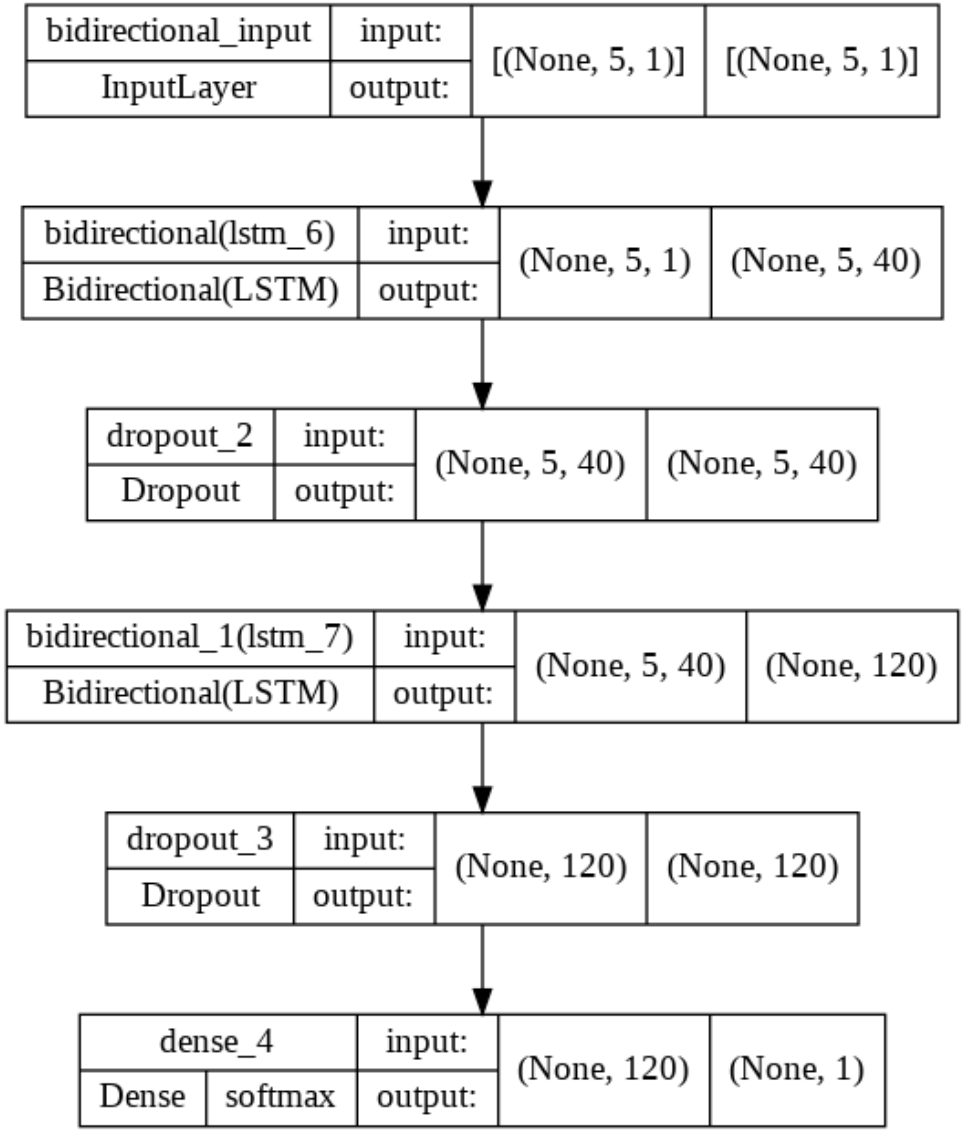
The Bi-LSTM Model

Bi-LSTM Model for Google dataset

- The bidirectional model used for google dataset:

```
# define model
model = Sequential()
model.add(Bidirectional(LSTM(20,return_sequences=True, activation='softmax'), input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.6))
model.add(Bidirectional(LSTM(60,return_sequences=False, activation='linear')))
model.add(Dropout(0.6))
model.add(Dense(1,activation='softmax'))
model.build()
```

Bi-LSTM Model for Google dataset



Bi-LSTM Model for Google dataset

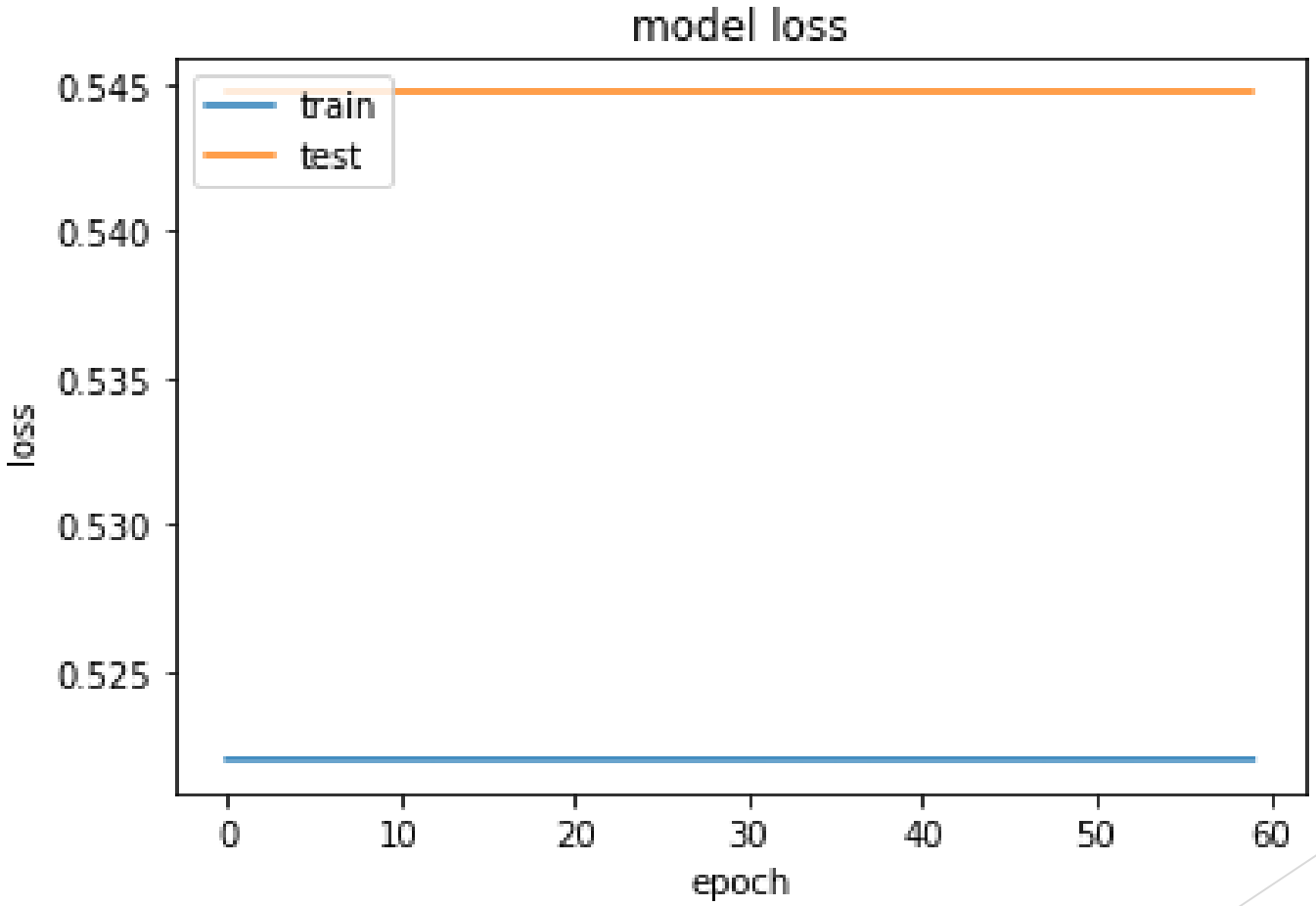
We trained our model with the training data over bellow parameters :

- epochs = 50
- with a batch size = 32
- optimizer= Adam
- loss= mean squared error (MSE)
- Validation split=0.25
- Drop out = 0.6

Accuracy of Bi-LSTM model = 0.0018975331913679838

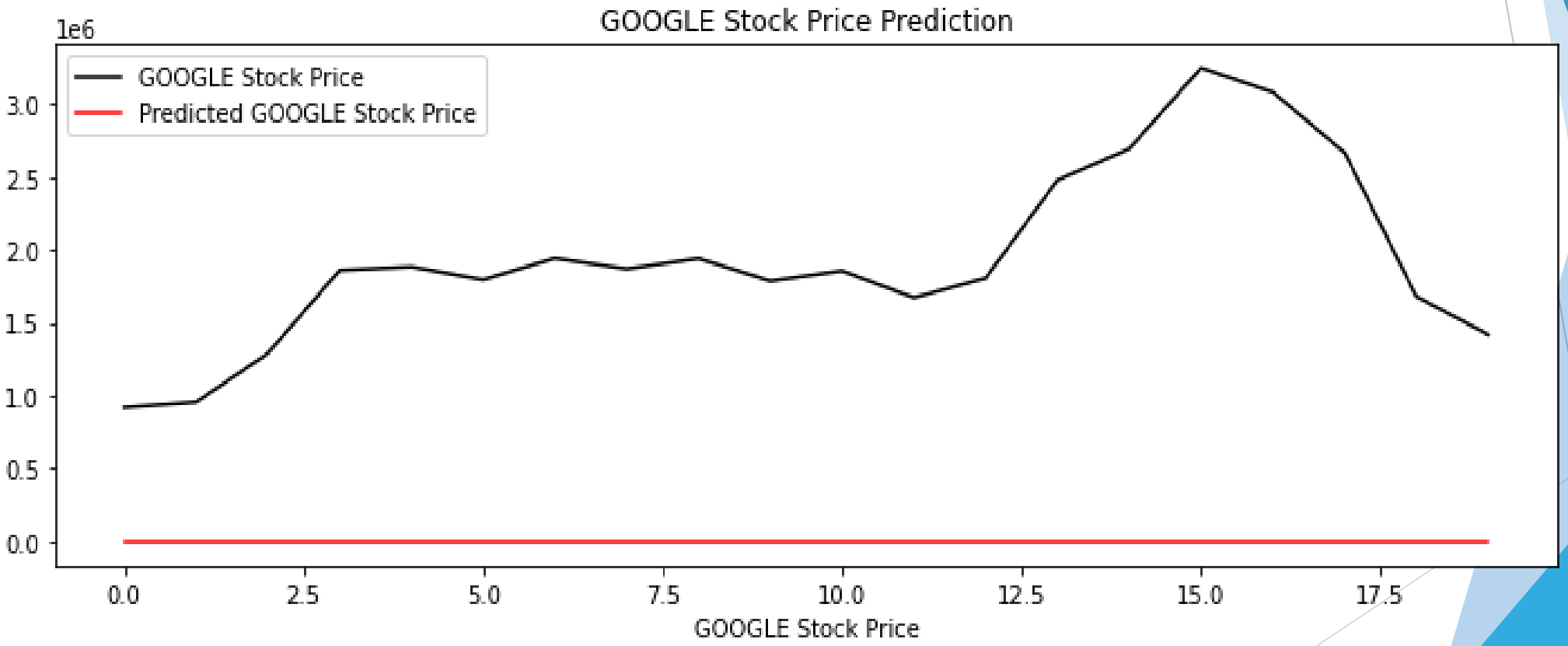
Bi-LSTM Model for Google dataset

- The Loss graph



Bi-LSTM Model for Google dataset

- The prediction



Conclusion

In this approach, the goal was to predict stock prices using a recursive artificial neural network.

For this purpose, several model architectures for stock price forecasting were presented. Also, tried to test different model parameters such as number of layers, nodes and drop outs and fitting over several batch size, epoch size, optimizer and different activation functions to determine the effect of different parameters in accuracy of the model prediction.

Here ,the LSTM models had better stock predictions, but with proper layers and hyper parameters, the Bi-LSTM model should also give good results

Thank you

Tahsin Ilkhas Zadeh : 40422034

Zeinab Khosravi : 99422067