Assignment 3

In this assignment i have taken stock prices of microsft from yahoo finance as a dataset . We will implement below structures to predict the prices of the stock.
1) Plain backpropogation
2) Backpropogation through time
3) LSTM
4) GRU

step 1

Import all the required libraries

In [ ]:

```
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from torch import nn, optim
import time
```

step2

Read the required csv file in the dataframe . Please note that we will use Close value in the data for prediction of stock proce.

In [2]:

```
df = pd.read_csv('MSFT.csv')
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062055	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064271	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065379	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.063717	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.062609	47894400

In [3]:

```
df=df.drop(['Date', 'Open','High','Low','Adj Close','Volume'], axis=1)
```

In [4]:

```
df.head()
```

Out[4]:

	Close
0	0.097222
1	0.100694
2	0.102431
3	0.099826
4	0.098090

In [5]:

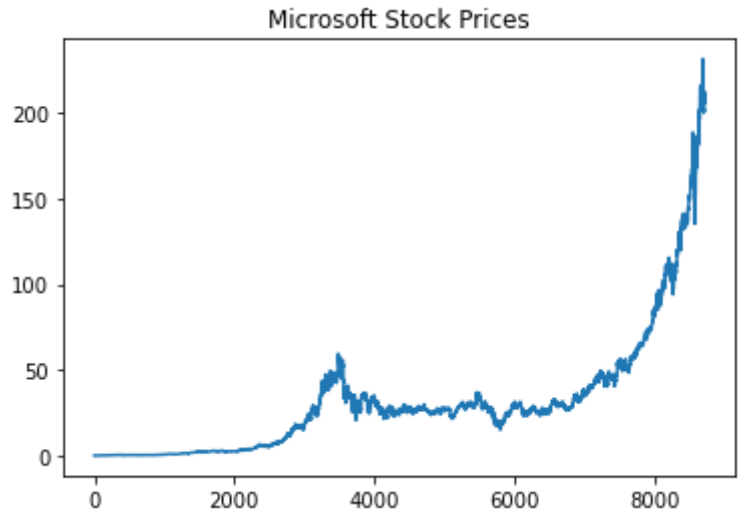
```
df.isnull().sum().sum()
```

Out[5]:

0

In [6]:

```
plt.plot(df['Close'])
plt.title("Microsoft Stock Prices");
```



In [7]:

```
len(df.index)
```

Out[7]:

8716

step 3

In this step we will split the dataset into trainset and testset. Since the data is timeseries data we will not use random function to split the data. We have used 70 percent of the data as a train data and rest of them as a train data.

In [8]:

```
test_data_size = 2716
train_data_set = df[:-test_data_size]
test_data_set = df[-test_data_size:]
```

In [9]:

```
len(train_data_set.index)
```

Out[9]:

```
6000
```

In [10]:

```
len(test_data_set)
```

Out[10]:

```
2716
```

step 4

We will scale the data in order to improve performance of the gradient descent. Please note that we have to use different scaler for train data and test data since the min and max for both of them will be different and both of them belongs to different time.

In [11]:

```
scaler = MinMaxScaler()
train_data_scaler = scaler.fit(train_data_set.to_numpy())
train_data = train_data_scaler.transform(train_data_set.to_numpy())
test_data_scaler = scaler.fit(test_data_set.to_numpy())
test_data = test_data_scaler.transform(test_data_set.to_numpy())
```

step 5

In the time series based data we will use last n input to predict the output at the current time . So inorder to do that we have created a function called create sequence which will take the input data and based on the sequence length it will use last "sequence length" values as input and current value as a output . In the next cell we have converted train and test data into input and output . After that we have converted those input and output into tensor.

In [12]:

```
def create_sequences(data, seq_length):
    xs = []
    ys = []

    for i in range(len(data)-seq_length-1):
        x = data[i:(i+seq_length)]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)

    return np.array(xs), np.array(ys)
```

In [13]:

```
seq_length = 5
X_train, Y_train = create_sequences(train_data, seq_length)
X_test, Y_test = create_sequences(test_data, seq_length)


X_train = torch.from_numpy(X_train).float()
Y_train = torch.from_numpy(Y_train).float()

X_test = torch.from_numpy(X_test).float()
Y_test = torch.from_numpy(Y_test).float()
```

step 6

This is a feed forward neural network.which is a plain backpropogation algorithm . For the first hidden layer we have used 12 nodes and for second hidden layer we have used 32 nodes and the last layer is the output.

In [14]:

```
class FeedForward(nn.Module):
    def __init__(self,input_size,output_size):
        super(FeedForward, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_sizes = [12,32]
        self.feedforward = nn.Sequential(nn.Linear(self.input_size,self.hidden_size
s[0]),
                                nn.Sigmoid(),
                                nn.Linear(self.hidden_sizes[0],self.hidden_sizes[1]),
                                nn.Sigmoid(),
                                nn.Linear(self.hidden_sizes[1],self.output_size))

    def forward(self,x):
        out = self.feedforward(x)
        return out
```

step 7

In this class we have used simple recurrent neural network which uses backpropogation through time . It's simpler than LSTM.We have used tanh as a non linear function and number of hidden nodes and number of layers are taken as a input from user.

In [15]:

```
import torch.nn as nn
class StockRNNModel(nn.Module):
    def __init__(self, inputSize, hiddenSize, numLayers):
        super(StockRNNModel,self).__init__()
        self.numLayers = numLayers
        self.batchSize = 1
        self.hiddenSize = hiddenSize
        self.RNN = nn.RNN(input_size=inputSize,
                          hidden_size=hiddenSize,
                          num_layers=numLayers,
                          nonlinearity='tanh',
                          batch_first=True)
        self.linear = nn.Linear(hiddenSize,1)

    def forward(self,x):
        hState = torch.zeros([self.numLayers,x.size(0), self.hiddenSize])
        x, h = self.RNN(x,hState)
        out = self.linear(x[:,-1,:])
        return out
```

step 8

In this class we have used LSTM class from pytorch library . We have taken number of hidden dimension and number layers as a input from user. We applied one liner neural network on output of the LSTM.

In [16]:

```
class StockLSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(StockLSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad
_()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad
_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.linear(out[:, -1, :])
        return out
```

step 9

In this class we have used GRU class from pytorch library . We have taken number of hidden dimension and number layers as a input from user. We applied one liner neural network on output of the GRU. GRU has different gate structure than LSTM so it does not require any cell state.

In [17]:

```
class StockGRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(StockGRUModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad
_()
        out, (hn) = self.gru(x, (h0.detach()))
        out = self.linear(out[:, -1, :])
        return out
```

step 10

In this step we will train all the 4 model and will plot the (epoch vs (train and test accuracy)) graph along with that we will also track the execution time and MSE for each model.

In [19]:

```
def train_model(model,X_train,Y_train,X_test,Y_test,epochs):
    loss_function = torch.nn.MSELoss(reduction='mean')
    optimizer = torch.optim.Adam(model.parameters(),lr=1e-3)
    train_histogram = np.zeros(epochs)
    test_histogram = np.zeros(epochs)

    for t in range(epochs):
        Y_pred = model(X_train)
        loss = loss_function(Y_pred.float(),Y_train)
        train_histogram[t] = loss.item()

        Y_test_pred = model(X_test)
        test_loss = loss_function(Y_test_pred.float(),Y_test)
        test_histogram[t] = test_loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    return model.eval(),train_histogram,test_histogram
```

In [20]:

```
execution_time = []
mse = []
```

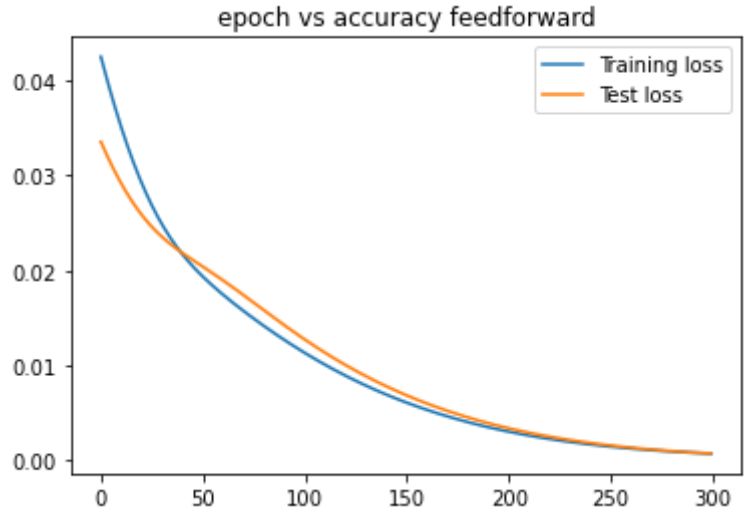
In [21]:

```
start_time = time.time()
stock_feed_forward = FeedForward(5,1)
epochs = 300
model_feed_forward,train_feed_forward,test_feed_forward = train_model(stock_feed_fo
rward,
                                                                       X_train.resha
pe(5994,5),
                                                                       Y_train,
                                                                       X_test.reshap
e(2710,5),
                                                                       Y_test,
                                                                      epochs)


mse.append(test_feed_forward[epochs-1])
execution_time.append(time.time()-start_time)
```

In [22]:

```
plt.plot(train_feed_forward, label="Training loss")
plt.plot(test_feed_forward, label="Test loss")
plt.title("epoch vs accuracy feedforward")
plt.legend();
```

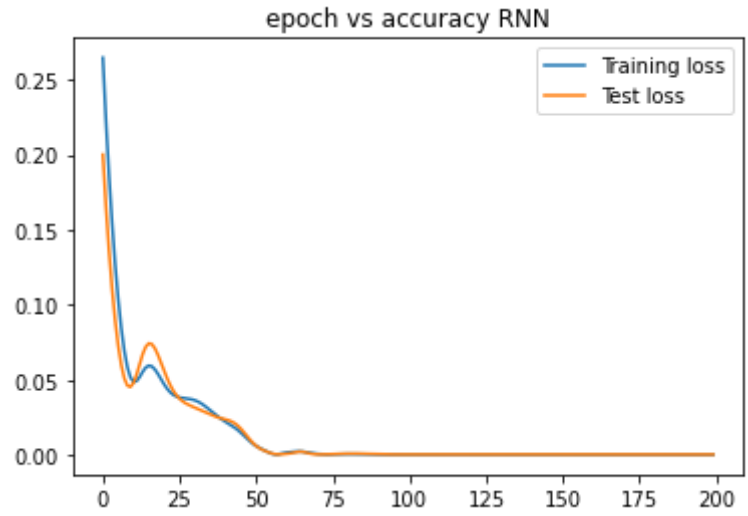


In [23]:

```
epochs = 200
start_time = time.time()
stock_rnn = StockRNNModel(1,32,2)
model_rnn,train_rnn,test_rnn = train_model(stock_rnn,X_train,Y_train,X_test,Y_test,
epochs)
mse.append(test_rnn[epochs-1])
execution_time.append(time.time()-start_time)
```

In [24]:

```
plt.plot(train_rnn, label="Training loss")
plt.plot(test_rnn, label="Test loss")
plt.title("epoch vs accuracy RNN")
plt.legend();
```

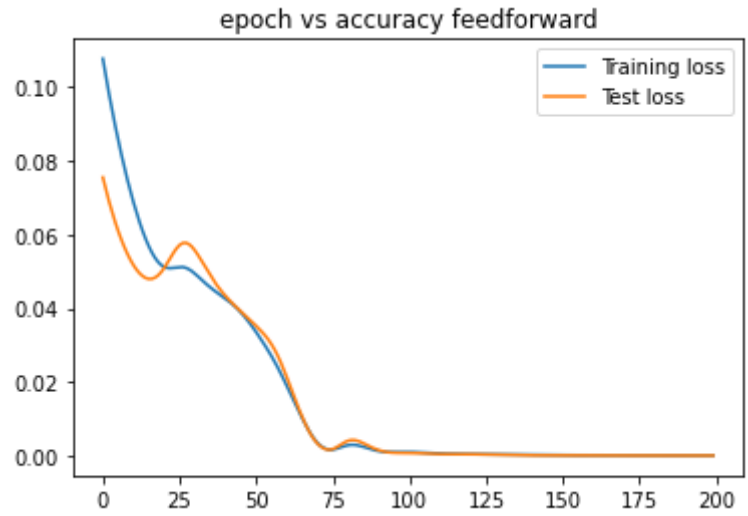


In [25]:

```
epochs = 200
start_time = time.time()
stock_lstm = StockLSTMModel(1,32,2,1)
model_lstm,train_lstm,test_lstm = train_model(stock_lstm,X_train,Y_train,X_test,Y_t
est,epochs)
mse.append(test_lstm[epochs-1])
execution_time.append(time.time()-start_time)
```

In [26]:

```
plt.plot(train_lstm, label="Training loss")
plt.plot(test_lstm, label="Test loss")
plt.title("epoch vs accuracy feedforward")
plt.legend();
```

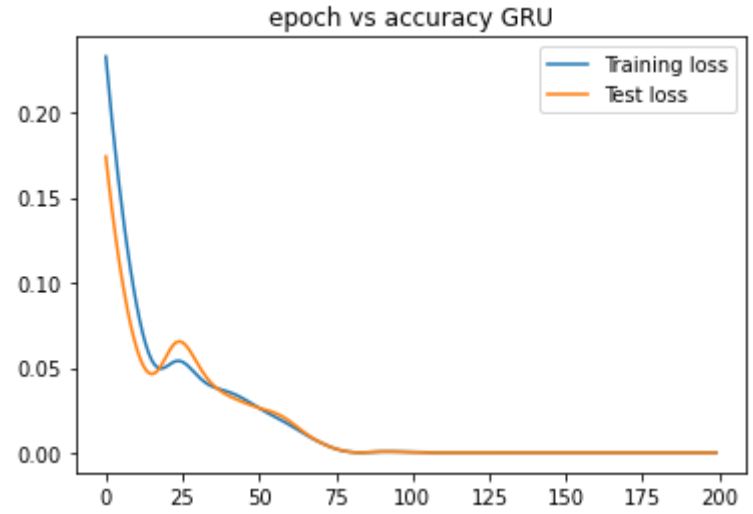


In [27]:

```
epochs = 200
start_time = time.time()
stock_gru= StockGRUModel(1,32,2,1)
model_gru,train_gru,test_gru = train_model(stock_gru,X_train,Y_train,X_test,Y_test,
epochs)
mse.append(test_gru[epochs-1])
execution_time.append(time.time()-start_time)
```

In [28]:

```
plt.plot(train_gru, label="Training loss")
plt.plot(test_gru, label="Test loss")
plt.title("epoch vs accuracy GRU")
plt.legend();
```



step 11

As we can see trainng time for LSTM was largest while simpler structures like feed forward neural network and RNN have lower training time . But When we compare the MSE we can see that GRU has the lowest MSE while feed forward neural network has highest MSE.

In [29]:

```
execution_time
```

Out[29]:

```
[0.4710850715637207, 8.243815422058105, 39.99551439285278, 28.313562154
769897]
```

In [30]:

```
mse
```

Out[30]:

```
[0.0007012145360931754,
 0.0003164022054988891,
 0.0001843045902205631,
 0.00010075954196508974]
```

In [ ]:

```

```