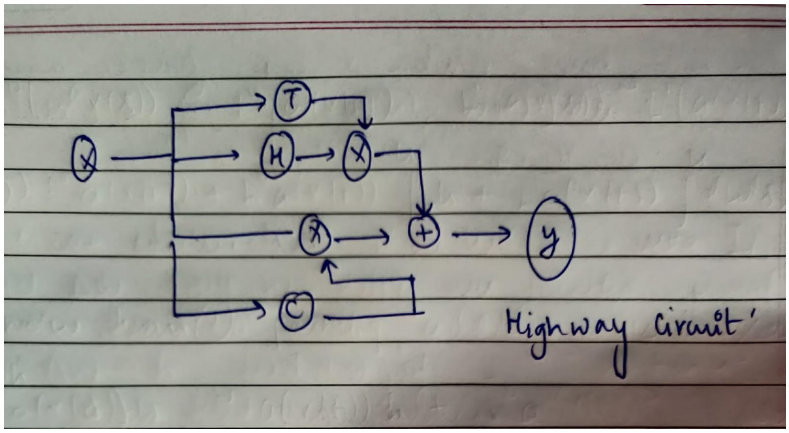
Advanced Statistical Algorithms MA691 Endsem

Name: **Naman Goyal**
Roll No: **180123029**

Ques. **1**



- In the highway network, 2 non linear transforms T,C are used.

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

- Where T -> Transform gate, C -> carry gate. Hence C = 1 - T.
- Put C = 1 - T in the above equations to yield

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

- Now we have below condition for T values:

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

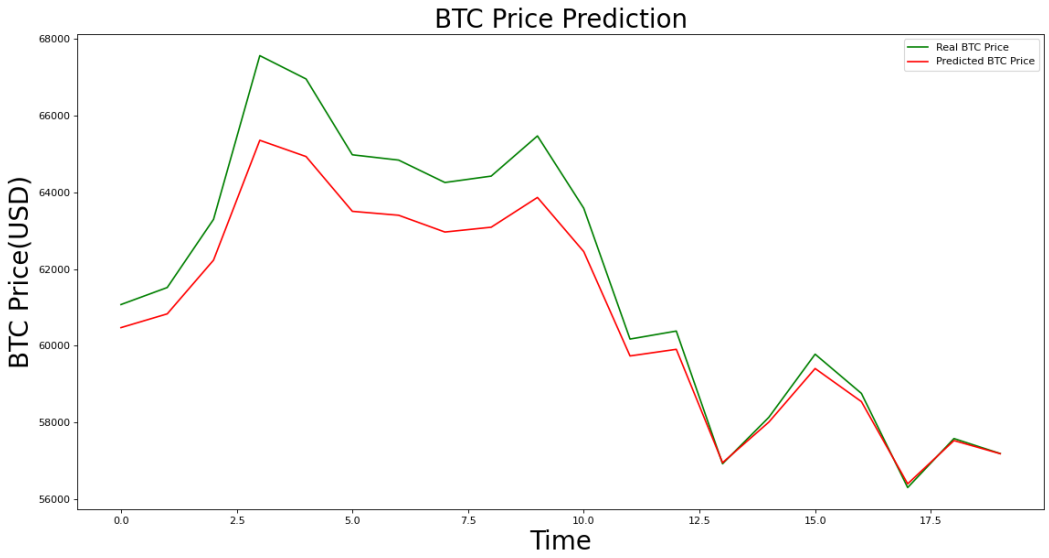
- When T = 0, we pass the input as output directly which creates an information highway called Highway network.

Code:

```
# Importing General libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

# Importing keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()

# Reading dataset
df = pd.read_csv("./BitcoinPrice.csv")
df_norm = df.drop(['Timestamp'], 1, inplace=True)
daysPrediction = 20
df_train= df[:len(df)-daysPrediction]
df_test= df[len(df)-daysPrediction:]

# Encoding categorical data
training_set = df_train.values
training_set = min_max_scaler.fit_transform(training_set)

x_set = training_set[0:len(training_set)-1]
y_set = training_set[1:len(training_set)]
x_set = np.reshape(x_set, (len(x_set), 1, 1))

num_units = 4
activation_function = 'sigmoid'
optimizer = 'adam'
loss_function = 'mean_squared_error'
batch_size = 5
num_epochs = 100

# Initialize the RNN
```

```
helper = Sequential()

# Adding the input layer and the LSTM layer
helper.add(LSTM(units = num_units, activation = activation_function,
input_shape=(None, 1)))

# Adding the output layer
helper.add(Dense(units = 1))

# Compiling the RNN
helper.compile(optimizer = optimizer, loss = loss_function)

# Using the training set to train the model
helper.fit(x_set, y_set, batch_size = batch_size, epochs = num_epochs)

test_set = df_test.values

inputs = np.reshape(test_set, (len(test_set), 1))
inputs = min_max_scaler.transform(inputs)
inputs = np.reshape(inputs, (len(inputs), 1, 1))

predicted_price = helper.predict(inputs)
predicted_price = min_max_scaler.inverse_transform(predicted_price)

plt.figure(figsize=(15, 15), dpi=80, facecolor = 'w', edgecolor = 'k')

plt.plot(test_set[:, 0], color='green', label='Real BTC Price')
plt.plot(predicted_price[:, 0], color = 'red', label = 'Predicted BTC
Price')

plt.title('BTC Price Prediction', fontsize = 25)
plt.xlabel('Time', fontsize=25)
plt.ylabel('BTC Price(USD)', fontsize = 25)
plt.legend(loc = 'best')
plt.show()
```

Output:

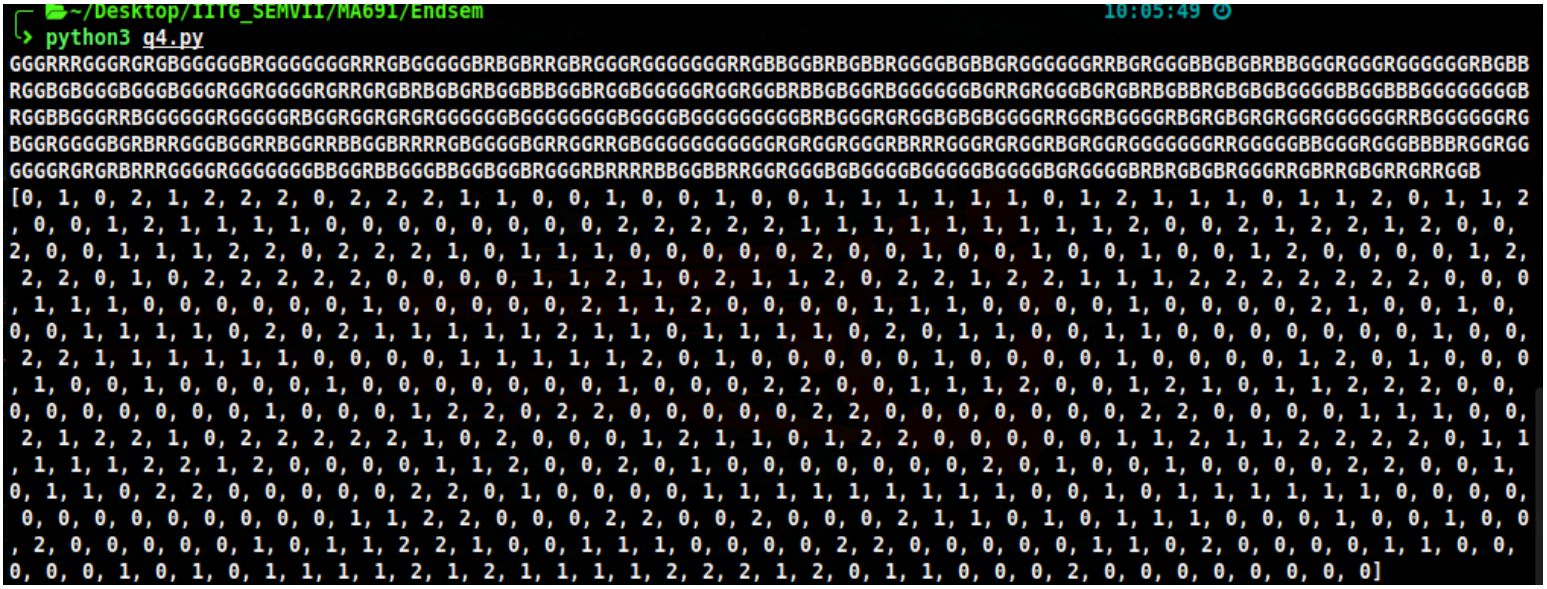


Ques. **4**

```
import random
import math

GeneratedSeq = ''
SeqUrn = []
ProbUrn = [0.6, 0.3, 0.1]
ProbSwitch = [[0.7, 0.2, 0.1], [0.3, 0.5, 0.2], [0.3, 0.3, 0.4]]
DistUrn = [[70, 20, 10], [50, 20, 30], [40, 40, 20]]
ReqSeq = 'RRGGB'

#Selecting the starting urn
def startUrn(ProbUrn):
 SeqUrn=[]
 u = random.uniform(0, 1)
 u -= ProbUrn[0]
 if u < 0:
     i = 0
 elif u-ProbUrn[1] < 0:
     i = 1
 else:
```

```
    i = 2
 SeqUrn.append(i)
 return SeqUrn,i

SeqUrn,i = startUrn(ProbUrn)

#picking a ball
flag = True
while flag:
   u = random.uniform(0, 1)
   u -= DistUrn[i][0]/100
   if u < 0:
       GeneratedSeq += 'G'
   elif (u-DistUrn[i][1]/100) < 0:
       GeneratedSeq += 'R'
   else:
       GeneratedSeq += 'B'
   if len(GeneratedSeq) >= 5:
       if GeneratedSeq[-5:] == ReqSeq:
           flag = False
           break

   u = random.uniform(0, 1)
   u -= ProbSwitch[i][0]
   if u < 0:
       i = 0
   elif u-ProbSwitch[i][1] < 0:
       i = 1
   else:
       i = 2
   SeqUrn.append(i)

print(GeneratedSeq)
print(SeqUrn)
```

Output:


~/Desktop/IITG_SEMVII/MA691/Endsem 10:05:49
python3 q4.py
GGGRRRGGGRGRGBGGGGGBRGGGGGGGRRRGBGGGGGBRBGBRRGBRGGGRGGGGGGGRRGBBGGBRBGBBRGGGGBGBBGRGGGGGGRRBGRGGGBBGBGBRBBGGGRGGGRGGGGGGRBGBB
RGGBGBGGGBGGGBGGGRGGRGGGGRGRRGRGBRBGBGRBGGBBBGGBRGGBGGGGGRGGRGGBRBBGBGGRBGGGGGGBGRRGRGGGBGRGBRBGBBRGBGBGBGGGGBBGGBBBGGGGGGGGB
RGGBBGGGRRBGGGGGGRGGGGGRBGGRGGRGRGRGGGGGGBGGGGGGGGBGGGGBGGGGGGGGGBRBGGGRGRGGBGBGBGGGGRRGGRBGGGGRBGRGBGRGRGGRGGGGGGGRRBGGGGGGRG
BGGRGGGGBGRBRRGGGBGGRRBGGRRBBGGBRRRRGBGGGGBGRRGGRRGBGGGGGGGGGGGRGRGGRGGGRBRRRGGGRGRGGRBGRGGRGGGGGGGRRGGGGGBBGGGRGGGBBBBRGGRGG
GGGGRGRGRBRRRGGGGRGGGGGGGBBGGRBBGGGBBGGBGGBRGGGRBRRRRBBGGBBRRGGRGGGBGBGGGGBGGGGGBGGGGBGRGGGGBRBRGBGBRGGGRRGBRRGBGRRGRRGGB
[0, 1, 0, 2, 1, 2, 2, 2, 0, 2, 2, 2, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 2, 1, 1, 1, 0, 1, 1, 2, 0, 1, 1, 2
, 0, 0, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 0, 2, 1, 2, 2, 1, 2, 0, 0,
2, 0, 0, 1, 1, 1, 2, 2, 0, 2, 2, 2, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 2, 0, 0, 0, 0, 1, 2,
 2, 2, 0, 1, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 1, 2, 1, 0, 2, 1, 1, 2, 0, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0
, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 1, 1, 2, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0,
0, 0, 1, 1, 1, 1, 0, 2, 0, 2, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 0, 2, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 0
, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 2, 0, 0, 1, 1, 1, 2, 0, 0, 1, 2, 1, 0, 1, 1, 2, 2, 2, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 1, 1, 1, 0, 0,
 2, 1, 2, 2, 1, 0, 2, 2, 2, 2, 2, 1, 0, 2, 0, 0, 0, 1, 2, 1, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 2, 2, 2, 2, 0, 1, 1
, 1, 1, 1, 2, 2, 1, 2, 0, 0, 0, 0, 1, 1, 2, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0, 0, 1,
0, 1, 1, 0, 2, 2, 0, 0, 0, 0, 0, 2, 2, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 0, 0, 0, 2, 2, 0, 0, 2, 0, 0, 0, 2, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0
, 2, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 2, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 1, 1, 0, 2, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
