Advanced Statistical Algortihms (MA691)

Name - Kartikeya Singh
Roll Number - 180123021

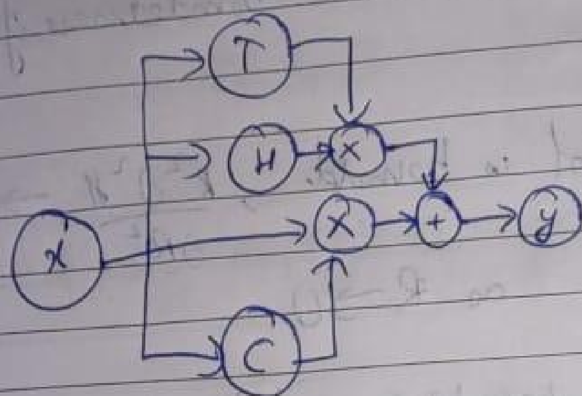
Question 1

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler()

df = pd.read_csv("BitcoinPrice.csv")
df_norm = df.drop(['Timestamp'], 1, inplace=True)

prediction_days = 20

df_train= df[:len(df)-prediction_days]
df_test= df[len(df)-prediction_days:]

training_set = df_train.values
training_set = min_max_scaler.fit_transform(training_set)

x_train = training_set[0:len(training_set)-1]
y_train = training_set[1:len(training_set)]
x_train = np.reshape(x_train, (len(x_train), 1, 1))

num_units = 4
activation_function = 'sigmoid'
optimizer = 'adam'
loss_function = 'mean_squared_error'
```

```
batch_size = 5
num_epochs = 100

regressor = Sequential()

regressor.add(LSTM(units = num_units, activation = activation_function,
input_shape=(None, 1)))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = optimizer, loss = loss_function)

regressor.fit(x_train, y_train, 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 = regressor.predict(inputs)
predicted_price = min_max_scaler.inverse_transform(predicted_price)

plt.figure(figsize=(25, 25), dpi=80, facecolor = 'w', edgecolor = 'k')

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

plt.title('BTC Price Prediction', fontsize = 40)
plt.xlabel('Time', fontsize=40)
plt.ylabel('BTC Price(USD)', fontsize = 40)
plt.legend(loc = 'best')
plt.show()
```


BTC Price Prediction
Real BTC Price
Predicted BTC Price
68000
66000
64000
62000
60000
58000
56000
BTC Price(USD)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Time

Q1. In highway network, two non-linear transformations T and C are introduced, T represents the transform gate and C represents the carry gate.



Highway Circuit.

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

$$C = 1 - T$$

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

We can have below conditions for particular T values →

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

→ When $T=0$, we pass the input as output directly which creates an information highway. So this is called highway network.

Question 2

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from torch.autograd import variable

training_set = pd.read_csv('./u1.base', delimiter = '\t')
training_set = np.array(training_set, dtype = 'int')
test_set = pd.read_csv('./u1.test', delimiter = '\t')
test_set = np.array(test_set, dtype = 'int')

# Getting the number of users and movies
nb_users = int(max(max(training_set[:,0]),
max(test_set[:,0])))
nb_movies = int(max(max(training_set[:,1]),
max(test_set[:,1])))

# converting the data into array with users in lines and
movies in column
```

```python
def convert(data):
    new_data = []
    for id_users in range(1, nb_users+1):
        id_movies = data[:,1][data[:,0] == id_users]
        id_ratings = data[:,2][data[:,0] == id_users]
        ratings = np.zeros(nb_movies)
        ratings[id_movies - 1] = id_ratings
        new_data.append(list(ratings))
    return new_data
training_set = convert(training_set)
test_set = convert(test_set)

# converting the data into torch tensor
training_set = torch.FloatTensor(training_set)
test_set = torch.FloatTensor(test_set)

# converting the ratings into binary rating 0 for (dislike)
and 1 for (like)
training_set[training_set == 0] = -1
training_set[training_set == 1] = 0
training_set[training_set == 2] = 0
training_set[training_set >=3 ] = 1
test_set[test_set == 0] = -1
test_set[test_set == 1] = 0
test_set[test_set == 2] = 0
test_set[test_set >=3 ] = 1

# Creating the architecture of the Neural Network
class RBM():
    def _init_(self, nv, nh):
        self.W = torch.randn(nh, nv)
        self.a = torch.randn(1, nh)
        self.b = torch.randn(1, nv)
```

```python
    def sample_h(self, x):
        wx = torch.mm(x, self.W.t())
        activation = wx + self.a.expand_as(wx)
        p_h_given_v = torch.sigmoid(activation)
        return p_h_given_v, torch.bernoulli(p_h_given_v)
    def sample_v(self, y):
        wy = torch.mm(y, self.W)
        activation = wy + self.b.expand_as(wy)
        p_v_given_h = torch.sigmoid(activation)
        return p_v_given_h, torch.bernoulli(p_v_given_h)
    def train(self, v0, vk, ph0, phk):
        #print("temp shape: ")
        temp = torch.mm(v0.t(), ph0) - torch.mm(vk.t(),
phk)

        #print(temp.shape)
        #print("Wshape: ")
        #print(self.W.shape)
        self.W += temp.t()
        self.b += torch.sum((v0 - vk), 0)
        #print(self.b.shape)
        self.a += torch.sum((ph0 - phk), 0)
nv = len(training_set[0])
print(nv)
nh = 512
batch_size = 100
rbm = RBM(nv, nh)

# Training the RBM model
nb_epoch = 10
for epoch in range(1, nb_epoch + 1):
    train_loss = 0
    s = 0.
```

```
    for id_user in range(0, nb_users - batch_size, 
batch_size):
        vk = training_set[id_user:id_user+batch_size]
        v0 = training_set[id_user:id_user+batch_size]
        ph0,_ = rbm.sample_h(v0)
        for k in range(10):
            _,hk = rbm.sample_h(vk)
            _,vk = rbm.sample_v(hk)
            vk[v0<0] = v0[v0<0]
        phk,_ = rbm.sample_h(vk)
        #print(v0.shape)
        #print(vk.shape)
        #print(ph0.shape)
        #print(phk.shape)
        rbm.train(v0, vk, ph0, phk)

        train_loss += torch.mean(torch.abs(v0[v0>=0] - 
vk[v0>=0]))
        s += 1.
    print('epoch: '+str(epoch)+' loss: '+str(train_loss/s))

# Testing the RBM Model
test_loss = 0
s = 0.
arr = []
for id_user in range(nb_users):
    v = training_set[id_user:id_user+1]
    vt = test_set[id_user:id_user+1]
    if len(vt[vt>=0]) > 0:
        _,h = rbm.sample_h(v)
        arr.append(np.reshape(h.numpy(), nh))
        _,v = rbm.sample_v(h)
        test_loss += torch.mean(torch.abs(vt[vt>=0] -
```

```
v[vt>=0]))
        s += 1.
print('test loss: '+str(test_loss/s))

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns


user_features = np.array(arr)
print(user_features.shape)
## FITTING KMEANS WITH 3 CLUSTERS
kmeans = KMeans(n_clusters=2,
random_state=0).fit(user_features)
## 2D VISUALIZING USING TSNE
user_features_tsne =
TSNE(n_components=2).fit_transform(user_features)
plt.figure(figsize=(16,10))
sns.scatterplot(
    x=user_features_tsne[:,0], y=user_features_tsne[:,1],
    hue=kmeans.labels_,
    palette=sns.color_palette("hls", 2),
    legend="full",
    alpha=0.3
)


# ____________DBN Method______________

import numpy as np
from dbn.models import UnsupervisedDBN
```

```python
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns


n_users = 943
n_movies = 1682
data = np.array([[None]*n_movies for _ in range(n_users)])
with open('ml-100k/u.data') as f:
    for line in f.readlines():
        uId,mId,rating,t = map(int,line.split('\t'))
        data[uId-1,mId-1] = rating
df = pd.DataFrame(data)
df = df.fillna(df.mean())

%cd deep-belief-network

dbn = UnsupervisedDBN(hidden_layers_structure=[256, 512],
                      batch_size=10,
                      learning_rate_rbm=0.06,
                      n_epochs_rbm=20,
                      activation_function='sigmoid')
dbn.fit(df.values)
user_features = dbn.transform(df.values)  ## 512
dimensional features

## FITTING KMEANS WITH 3 CLUSTERS
kmeans = KMeans(n_clusters=3,
random_state=0).fit(user_features)

## 2D VISUALIZING USING PCA
```

```
user_features_pca =
PCA(n_components=2).fit_transform(user_features)

plt.figure(figsize=(16,10))
sns.scatterplot(
    x=user_features_pca[:,0], y=user_features_pca[:,1],
    hue=kmeans.labels_,
    palette=sns.color_palette("hls", 3),
    legend="full",
    alpha=0.3
)
plt.show()

## 2D VISUALIZING USING TSNE
user_features_tsne =
TSNE(n_components=2).fit_transform(user_features)
plt.figure(figsize=(16,10))
sns.scatterplot(
    x=user_features_tsne[:,0], y=user_features_tsne[:,1],
    hue=kmeans.labels_,
    palette=sns.color_palette("hls", 3),
    legend="full",
    alpha=0.3
)
```

Question 4

```python
import numpy as np

rev = {'G':0, 'R':1, 'B':2}

A = np.array([[0.7,0.2,0.1],[0.3,0.5,0.2],[0.3,0.3,0.4]])
U = np.array([[70,20,10], [50,20,30], [40,40,20]])
B = [[] for i in range(3)]

for i in range(U.shape[0]):
   for j in U[i]:
       B[i].append(j/np.sum(U[i]))

B = np.array(B)
Pi = [0.6, 0.3, 0.1]
Y = ['R', 'R', 'G','G', 'B']
y = np.array([rev[i] for i in Y])

print(A, B)

def viterbi(y, A, B, Pi=None):

   K = A.shape[0]
   Pi = Pi if Pi is not None else np.full(K, 1 / K)
   T = len(y)
   T1 = np.empty((K, T), 'd')
   T2 = np.empty((K, T), 'B')

   print(y[0])
   print(B[:,0])

   T1[:, 0] = Pi * B[:, y[0]]
   T2[:, 0] = 0

   for i in range(1, T):
       T1[:, i] = np.max(T1[:, i - 1] * A.T * B[np.newaxis, :, y[i]].T, 1)
       T2[:, i] = np.argmax(T1[:, i - 1] * A.T, 1)

   x = np.empty(T, 'B')
   x[-1] = np.argmax(T1[:, T - 1])

   for i in reversed(range(1, T)):
       x[i - 1] = T2[x[i], i]

   return x, T1, T2
```

```
print(viterbi(y,A,B,Pi)[0])
```