Neel Zadafiya (1115533)

Machine Learning - Home Assignment 2 - Part 2

Goal:

Implement Incremental Extreme Learning Machine to deepen the understanding of the random network.

Task 1:

- Classification

Answer: Classification.py

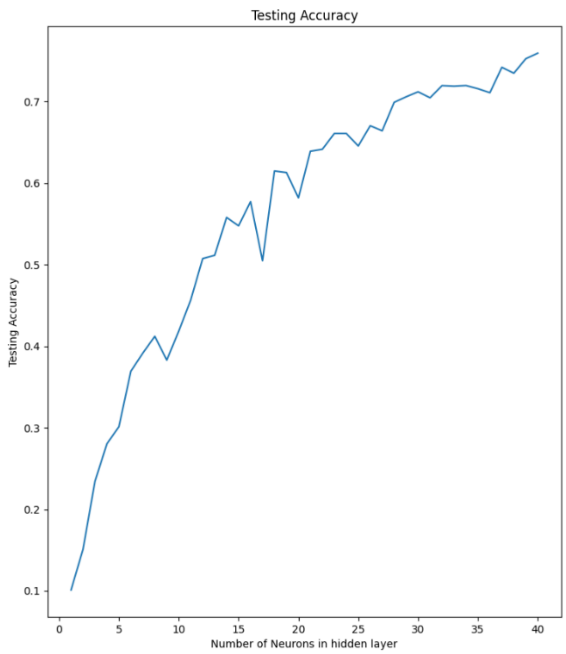
```
#Import libraries

import hpelm
from keras.datasets import mnist
from keras.utils import to_categorical
import numpy as np
from numpy import random
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import time
import matplotlib.pyplot as plt

#Lists to store results
Train_T = []
Test_E  = []

#Load mnist data
(x_train,y_train),(x_test,y_test) = mnist.load_data()

#Scale data
x_train = x_train.astype(np.float32) / 255.0
x_train = x_train.reshape(-1,28*28)

x_test = x_test.astype(np.float32) / 255.0
x_test = x_test.reshape(-1,28*28)

# 1 hot encoding
y_train = to_categorical(y_train,10).astype(np.float32)
y_test = to_categorical(y_test,10).astype(np.float32)

#=================================================================

def calculateE(y,t):

    p = np.zeros_like(t)
    p[np.arange(len(t)), t.argmax(1)] = 1

    hit = 0
    miss = 0
```

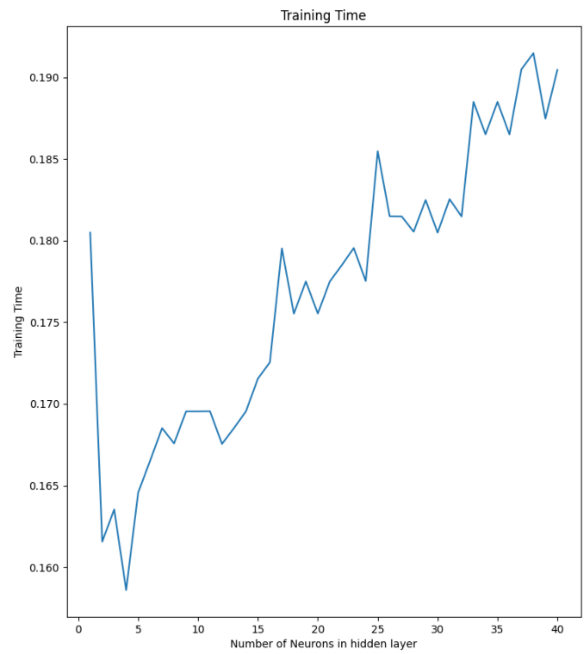
```
    #Calculate accuracy
    for i in range(len(t)):

        if np.where(p[i]==1)==np.where(y[i]==1):
            hit = hit + 1
        else:
            miss = miss + 1

    return hit/(hit+miss)


#================================================================
#Initialization

Lmax = 40
L = 0
E = 0
ExpectedAccuracy = 0

while L < Lmax and E >= ExpectedAccuracy:

    #Increase Node
    L = L + 1

    #Calculate Random weights, they are  already addded into model using
hpelm library
    w = random.rand(784,L)

    #Initialize model
    model = hpelm.ELM(28*28,10)
    model.add_neurons(L,'sigm')

    start_time = time.time()

    #Train model
    model.train(x_train,y_train,'ml')

    Train_T.append(time.time() - start_time)

    #Calculate output weights and intermediate layer
    BL_HL = model.predict(x_test)

    #Calculate new accuracy
    E = calculateE(y_test,BL_HL)

    Test_E.append(E)

    #Print result
    print("Hidden Node",L,"Accuracy :",E)

#====================================================================

#Find best accuracy
L = Test_E.index(max(Test_E)) + 1

#Define model
model = hpelm.ELM(28*28,10)
```

```
model.add_neurons(L,'sigm')

start_time = time.time()
model.train(x_train,y_train,'ml')
print('Training Time :',time.time() - start_time)

start_time = time.time()
BL_HL = model.predict(x_train)
print('Testing Time :',time.time() - start_time)

#Calculate training accuracy
E = calculateE(y_train,BL_HL)
print('Training Accuracy :',E)
print('Testing Accuracy  :',max(Test_E))

#==================================================================

#Plot Data

plt.subplot(1, 2, 1)    #Generate graph for ANN
plt.plot(range(1,Lmax+1),Test_E)
plt.title('Testing Accuracy')
plt.xlabel('Number of Neurons in hidden layer')
plt.ylabel('Testing Accuracy')


plt.subplot(1, 2, 2)    #Generate graph for CNN
plt.plot(range(1,Lmax+1),Train_T)
plt.title('Training Time')
plt.xlabel('Number of Neurons in hidden layer')
plt.ylabel('Training Time')


plt.show()
```

Ourput:

```
Hidden Node 1 Accuracy : 0.101
Hidden Node 2 Accuracy : 0.1509
Hidden Node 3 Accuracy : 0.2343
Hidden Node 4 Accuracy : 0.2804
Hidden Node 5 Accuracy : 0.3013
Hidden Node 6 Accuracy : 0.3694
Hidden Node 7 Accuracy : 0.3916
Hidden Node 8 Accuracy : 0.4122
Hidden Node 9 Accuracy : 0.383
Hidden Node 10 Accuracy : 0.4185
Hidden Node 11 Accuracy : 0.4567
Hidden Node 12 Accuracy : 0.5075
Hidden Node 13 Accuracy : 0.5116
Hidden Node 14 Accuracy : 0.558
Hidden Node 15 Accuracy : 0.5477
Hidden Node 16 Accuracy : 0.5774
Hidden Node 17 Accuracy : 0.505
Hidden Node 18 Accuracy : 0.615
Hidden Node 19 Accuracy : 0.6131
Hidden Node 20 Accuracy : 0.582
Hidden Node 21 Accuracy : 0.6392
Hidden Node 22 Accuracy : 0.6416
Hidden Node 23 Accuracy : 0.661
Hidden Node 24 Accuracy : 0.661
Hidden Node 25 Accuracy : 0.6457
Hidden Node 26 Accuracy : 0.6705
Hidden Node 27 Accuracy : 0.6643
Hidden Node 28 Accuracy : 0.6992
Hidden Node 29 Accuracy : 0.706
Hidden Node 30 Accuracy : 0.7121
Hidden Node 31 Accuracy : 0.7048
Hidden Node 32 Accuracy : 0.7197
Hidden Node 33 Accuracy : 0.719
Hidden Node 34 Accuracy : 0.7198
Hidden Node 35 Accuracy : 0.7159
Hidden Node 36 Accuracy : 0.711
Hidden Node 37 Accuracy : 0.7422
Hidden Node 38 Accuracy : 0.7349
Hidden Node 39 Accuracy : 0.7527
Hidden Node 40 Accuracy : 0.7595
Training Time : 0.1894996166229248
Testing Time : 0.26226305961608887
Training Accuracy : 0.7538166666666667
Testing Accuracy  : 0.7595
```





Task 2:

- Regression

Answer: Regression.py

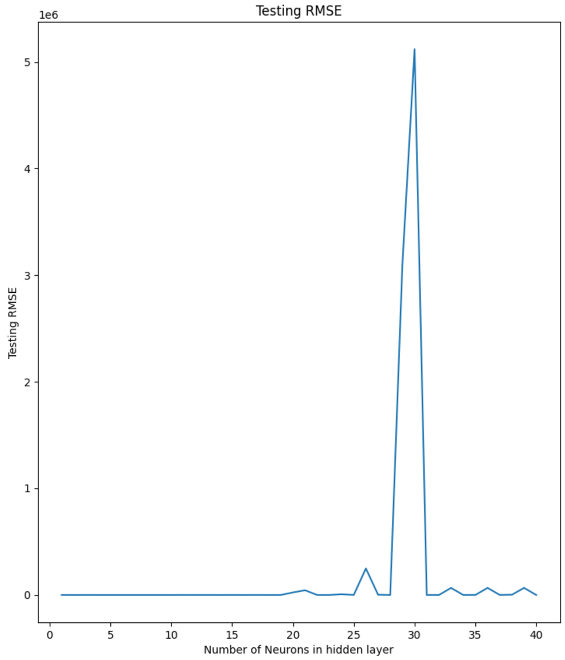
```
#Import libraries

import hpelm
from keras.datasets import mnist
from keras.utils import to_categorical
import numpy as np
from numpy import random
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import time
import matplotlib.pyplot as plt

#Lists to store results
Train_T = []
Test_E  = []

##Load wine testing UCI data data
data = np.genfromtxt('winequality-white.csv', dtype = float, delimiter = ';')

#Delete heading
data = np.delete(data,0,0)

x = data[:,:11]
y = data[:,-1]
```

```
#Train test split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.33,random_state=42)

#============================================================

def calculateE(y,t):

    #Calculate RMSE
    return mean_squared_error(y, t)


#============================================================
#Initialization

Lmax = 40
L = 0
E = 0
ExpectedAccuracy = 0

while L < Lmax and E >= ExpectedAccuracy:

    #Increase Node
    L = L + 1

    #Calculate Random weights, they are  already addded into model using
hpelm library
    w = random.rand(11,L)

    #Initialize model
    model = hpelm.ELM(11,1)
    model.add_neurons(L,'sigm')

    start_time = time.time()

    #Train model
    model.train(x_train,y_train,'r')

    Train_T.append(time.time() - start_time)

    #Calculate output weights and intermediate layer
    BL_HL = model.predict(x_test)

    #Calculate new EMSE
    E = calculateE(y_test,BL_HL)

    Test_E.append(E)

    #Print result
    print("Hidden Node",L,"RMSE :",E)

#==================================================================

#Find best RMSE
L = Test_E.index(min(Test_E)) + 1
```

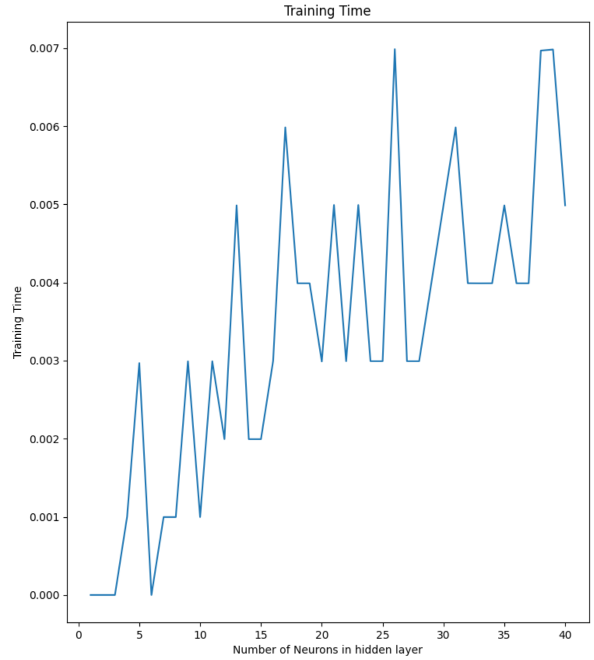
```
print()
print()
print()
print()

#Define model
model = hpelm.ELM(11,1)
model.add_neurons(L,'sigm')

start_time = time.time()
model.train(x_train,y_train,'r')
print('Training Time :',time.time() - start_time)

start_time = time.time()
BL_HL = model.predict(x_train)
print('Testing Time :',time.time() - start_time)

#Calculate training RMSE
E = calculateE(y_train,BL_HL)
print('Training RMSE :',E)
print('Testing RMSE  :',min(Test_E))

#=====================================================================

#Plot Data

plt.subplot(1, 2, 1)    #Generate graph for ANN
plt.plot(range(1,Lmax+1),Test_E)
plt.title('Testing RMSE')
plt.xlabel('Number of Neurons in hidden layer')
plt.ylabel('Testing RMSE')


plt.subplot(1, 2, 2)    #Generate graph for CNN
plt.plot(range(1,Lmax+1),Train_T)
plt.title('Training Time')
plt.xlabel('Number of Neurons in hidden layer')
plt.ylabel('Training Time')


plt.show()
```

Output:

```
  B = np.linalg.lstsq(HH, HT)[0]
Training Time : 0.003989458084106445
Testing Time : 0.0029916763305664062
Training RMSE : 0.7130317508834515
Testing RMSE  : 0.7242310914703839
```



Testing RMSE
1e6
Testing RMSE
Number of Neurons in hidden layer



Training Time
Training Time
Number of Neurons in hidden layer