**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)

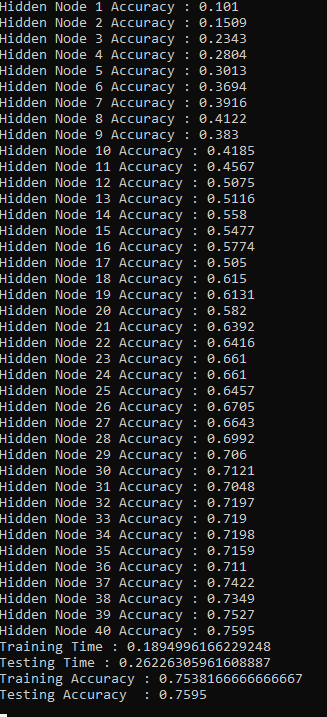
plt.title('Training Time')

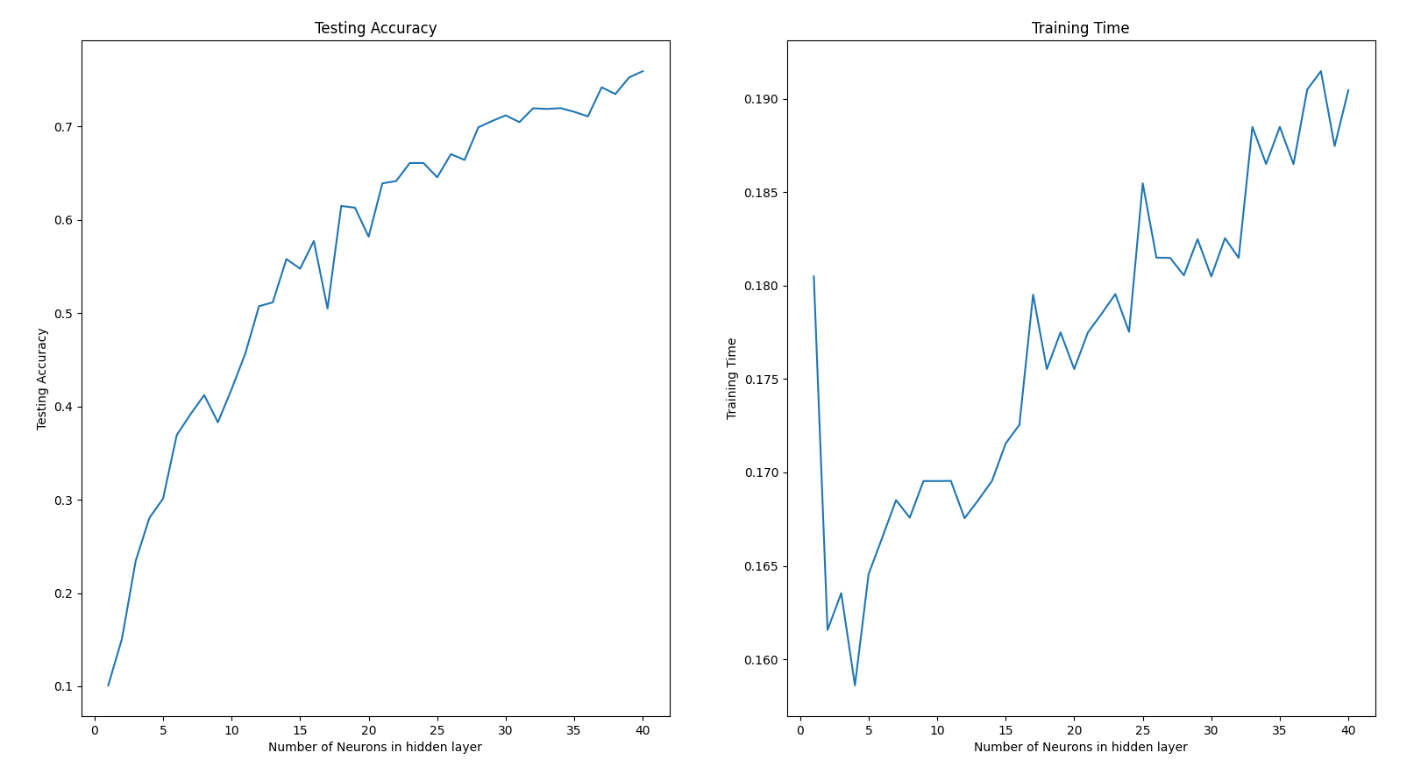
plt.xlabel('Number of Neurons in hidden layer')

plt.ylabel('Training Time')

plt.show()

Ourput:





**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:

