# **Typographic Conventions**

Functions (script) names starting with Capital letters correspond to “GUI functions”

Functions (script) names starting with small letters correspond to “core mathematical functions” used to develop the network

All variable names start with capital letters

# **Functions used**

## **helm\_train.m**

Used for training the ELM.

### **Syntax**

[stack,TrainingAccuracy,Actual,Expected] = helm\_train(train\_x,train\_y,SizeMatrix,NumEpochs,ActivationFunction)

### **Input Arguments:**

Train\_x: Training inputs (Dimensions: Number of entries Number of features)

Train\_y: Training targets (Dimensions: Number of entries Number of classes)

SizeMatrix: Number of layers and number of neurons in each hidden layer

ActivationFunction: Activation function of neurons. User can select from ‘sigmoid’, ‘sine’, ‘hardlim’, ‘tribas’ and ‘radbas’

NumEpochs: Number of training epochs

### **Output Arguments:**

stack: Trained network (weights of each layer)

Actual: Training labels predicted by network

Expected: Training labels

TrainingAccuracy: Prediction Accuracy on training data

## **helm\_test.m**

### **Syntax**

[Actual, TestingAccuracy, Expected] = helm\_test(Test\_x, Test\_y, stack,ActivationFunction)

### **Input Arguments**

**Test\_x:** Testing inputs (Dimensions: Number of entries Number of features)

**Test\_y:** Testing targets (Dimensions: Number of entries Number of classes)

**stack:** Trained network (weights of each layer)

**ActivationFunction:** Activation function of neurons. User can select from ‘sigmoid’, ‘sine’, ‘hardlim’, ‘tribas’ and ‘radbas’

### **Output Arguments:**

**Actual:** Testing labels predicted by network

**Expected:** Testing labels

**TestingAccuracy:** Prediction Accuracy on testing data

## **Sparse\_elm\_autoencoder**

### **Sytntax**

x = sparse\_elm\_autoencoder(A,b,lam,itrs)

### Input arguments

**A:** Input of the encoder

**b:** output of encoder

**lam:** tolerance

**itrs:** number of epochs

### **Output**

**x:** linear weights

### **Description**

ELM auto-encoder estimates the weights by assuming that output of the auto-encoder is actually a transformed version of the input. Let’s say ‘A’ is the input matrix and ‘b’ is the output matrix. By definition, ‘b’ is a transformed version of ‘A’ and we solve following equation by Moore-penrose method to estimate linear weights ‘x’

# **Data format**

Let’s say we have total M =1000 data entries and 5 classes. Let’s say each class is described by N=10 features, to . We can place entries of the two classes in one matrix randomly.

Input data will look like as following

.

.

.

For target data, there should be M (2 in this case) column, each column belongs to a class. Input and target data will look like

Input Data Target Data

Class 1 Class2 Class3 Class4 Class5

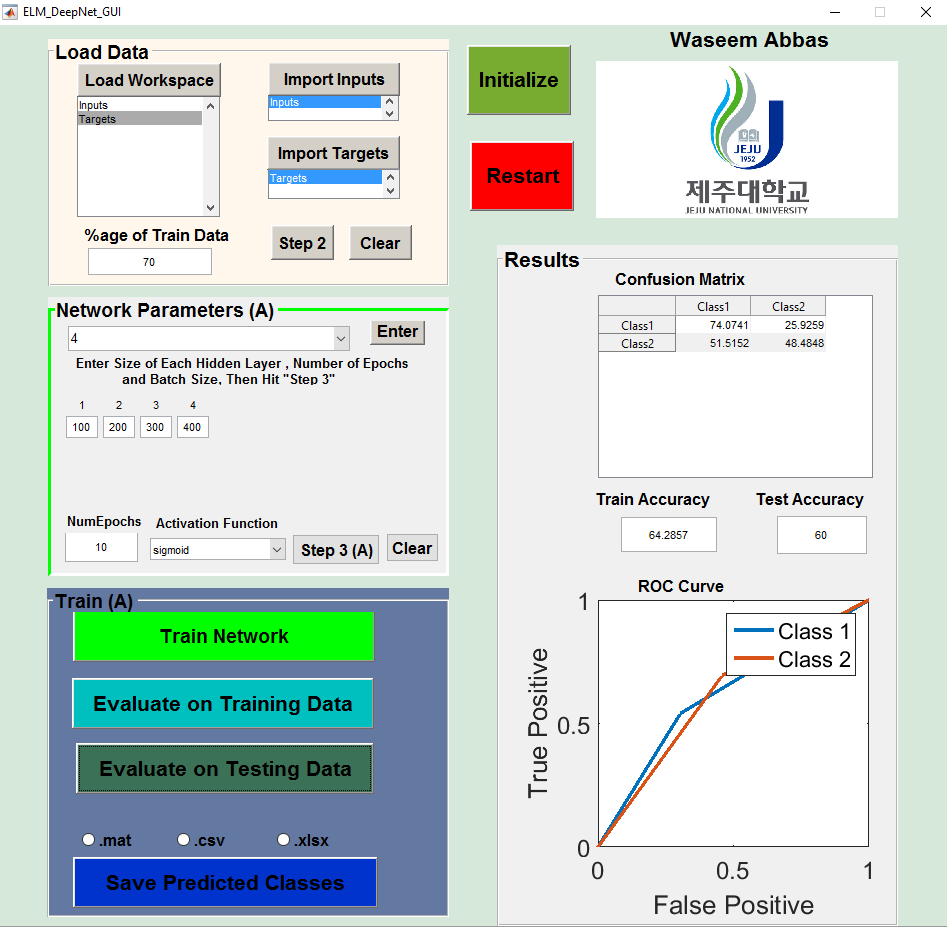
.

.

.

### GUI

Run ‘ELM\_DeepNet\_GUI.m’



**4**

**3**

**2**

**1**

There are 4 parts of the GUI, Load Data, Network Parameters, Train and Results

## **Load Data**

This panel is used to load data from workspace. To import in puts, first click “load workspace” and then select the inputs. Then click on ‘import inputs’ button. Similarly, select targets and then click ‘Import Targets’. Once inputs and targets are imported, click on ‘step 2’. It will activate the next panel

## **Parameters**

This panel is used to set network parameters. Use following steps to properly initialize network parameters

1. Select number of hidden layers and click ‘Enter’
2. Let’s say you have selected 4 hidden layers, you will see 4 input boxes created below. Each correspond to a hidden layer. Enter number of neurons in each layer
3. Input number of epochs, and select an activation function
4. Once you have completed step a-c, click on ‘step 3’. Network parameters will be saved

## **Train**

This panel is used to train and evaluate the network. First, hit ‘Train Network’ button. When network is trained, it can be evaluated on train data and test data. The results can also be saved.

## **Results**

In this panel, confusion matrix and ROC curve can be observed