HOMEWORK #2:

Non-Linear Classification using Neural Networks

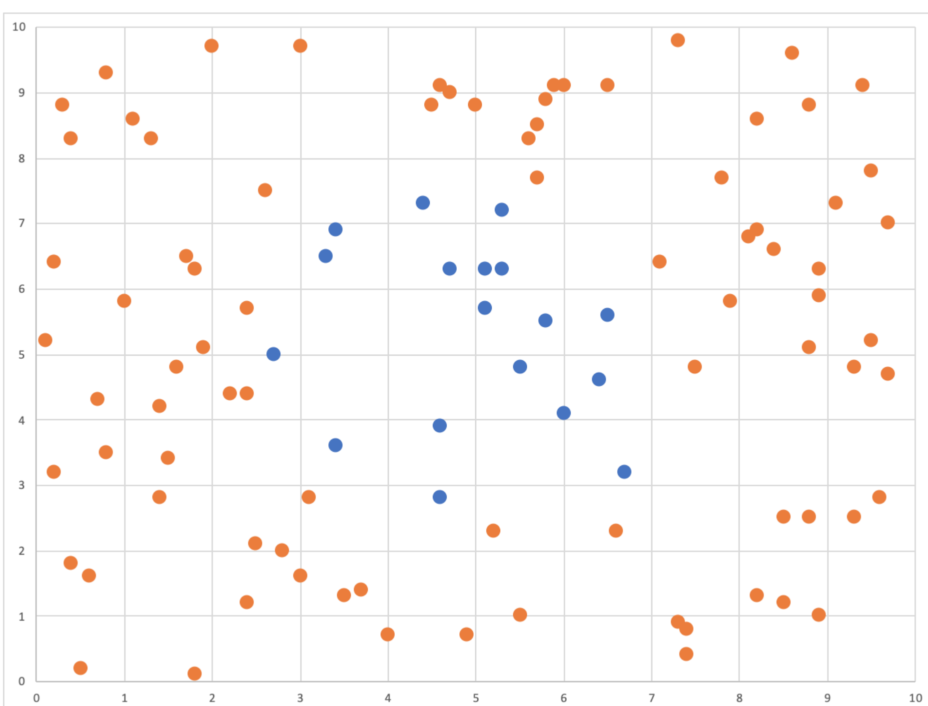
Due Date: Friday, December 17th, 11:59:59pm

Problem:

For this Homework you will perform non-linear classification using a small Neural Network.

Input:

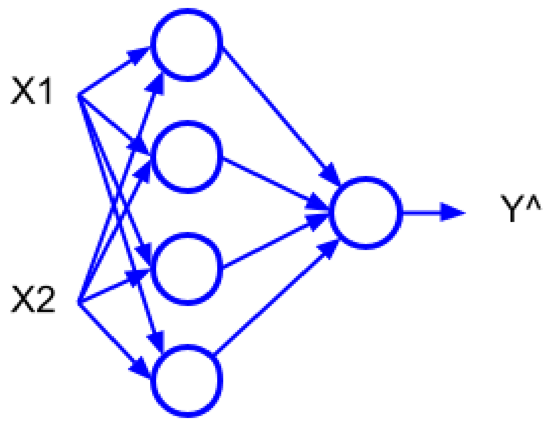
The file `data2.txt` contains the training data for a classification problem. Data consists of points in a plane categorized in one of two categories (labeled '0' or '1'). The first column and second columns characterize the point and the third column states the point's category. You can visualize the training data in the following scatter plot type '0' points are in blue and type '1' points are in orange:



Notice that this data is not "Linearly Separable", therefore a single neural unit is not enough to classify it. The file `valid2.txt` contains the **validation** data for this classification problem.

Back Propagation:

You will implement your Neural Network using **back-propagation**, and will use the **sigmoid** function as the activation function. Your network will need to have at least 2 layers. For this homework, use the following network architecture:



Initialize the weights of all the units randomly **in the range [-1..1]**. The learning rate η (eta) is left for you to decide. Experiment with different values and search for one that yields good results.

Validation:

After performing gradient descent on the training data, your program should evaluate the performance of your learner against the validation data by computing the sum-of-squares error between the validation data and your learner's predictions. *Do not use the validation data to train!*.

Submission Guidelines:

You will submit through **Canvas**:
Your submission should consists of the following components:

1. Your **program** files.- Submit all necessary files. Your main program file should be called '`learner2.X`' where X is the extension of whatever programming language you are using.
2. A **text** file, called '`learner2output.txt`', in which you report on the training run that achieved the best results. This file should **strictly** follow the format shown below.

Report file format:

The first line of the report text file is the values of the learned weights, space separated. The second line is the **Sum of Squares** error of the learned weights against the validation data. It is important that you follow the format because **these lines will be read by the auto-grader** to evaluate your program.

Sample 'learner2output.txt'

```
1.31   23.3   12.5
45.2   23.2   723.3
22.4   123.4   0.452
23.3   123.3   73.11
62.73   723.74   223.45   24.1   563.1
1237491.2831

CS-5001: HW#2
Programmer: Philip J. Fry

TRAINING
Using learning rate eta = 0.001
Using 10000 iterations.

LEARNED:
Input Units:
1.31   23.3   12.5
45.2   23.2   723.3
22.4   123.4   0.452
23.3   123.3   73.11
Output Unit:
62.73   723.74   223.45   24.1   563.1

VALIDATION
Sum-of-Squares Error = 1237491.2831
```

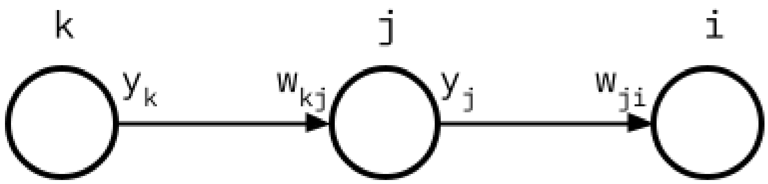
Pseudocode:

```
PROCEDURE BackPropagation
   E  : Set of examples, each of the form 〈 X, Y 〉  where:
      X = 〈 X1, X2, X3, ..., Xnx 〉

   NN : A sequence of Layers
   Ycap[ ny ] : output of the Neural Network

   REPEAT
      FOR EACH example e in E
         values := e.X
         FOR EACH layer L in NN, from input to output
            values := L.feedForward( values )

         Ycap := values
         FOR EACH output j in [0..ny]
            delta[j] := ( e.Y - Ycap[y] ) * Ycap[y] * ( 1 - Ycap[y] )
         FOR EACH layer L in NN, from output to input
            delta := L.backProp( delta )

   UNTIL Termination.
```

Formulae:

(from [link])

Consider the following slice of the inside of a neural network:



For a weight w_{kj} connecting a node in layer k to a node in layer j, the update formula is :

$$w_{kj} = w_{kj} + \Delta w_{kj}$$
$$\Delta w_{kj} = \eta * \delta_j * y_k$$

where:

η is the learning rate

y_k is the output of the node in layer k to which w_{kj} is applied.

δ_j is the "error contribution" associated with the neuron that contains w_{kj}

The δ error contribution is computed as follows:

If j is the output layer and t is the target value

$$\delta_j = (t - y_j) * y_j * (1 - y_j)$$

If j is a hidden layer:

$$\delta_j = (\Sigma_i \ \delta_i * w_{ji}) * y_j * (1 - y_j)$$

where δ_i's are the error contributions of the neurons in layer i.

Pseudocode 2 :

```
CONSTANTS
    eta

TYPES
    A Neural Layer consists of :
        size
        weights[][]     // weights of all nodes in this layer
        output[]        // output of this layer
        delta[]         // error contribution of this layer
        wchange[][]     // weight changes to be applied

GLOBALS
    E : set of examples : ⟨ x1, x2, x3, ..., xnx, target ⟩
    NNet[] : sequence of Neural Layers
        // NNet[0] is the output layer
        // NNet[nl] is the input layer

PROCEDURE FeedForward ( example e : ⟨ x1, x2, x3, ..., xnx, target ⟩ )

  Initialize NNet[nl].output[] to ⟨ x1, x2, x3, ..., xnx  ⟩

  FOR every layer j in NNet from layer nl-1 downto output layer 0
    compute NNet[j].output[] from NNet[j].weights[][] and NNet[j+1].output[]

END


PROCEDURE BackPropagation ( target )

  compute NNet[0].delta[] from NNet[0].output[] and target
  compute NNet[0].wchange[][] from eta, NNet[0].delta[] and NNet[1].output[]

  FOR every layer j in NNet from layer 1 upto layer nl-1

      compute NNet[j].delta[] from ( NNet[j-1].delta[], NNet[j-1].weights[][],
                                     NNet[j].output )

      compute NNet[j].wchange[][] from ( eta, NNet[j].delta[],
                                         NNet[j+1].output[] )

  FOR every layer j in NNet from layer 0 upto layer nl-1
      update NNet[j].weights[][] with NNet[j].wchange[][]
END
```

```
PROCEDURE NeuralNetworkLearner

  Initialize all NNet[].weights[][].

  REPEAT

    pick an example e : ⟨ x_1, x_2, x_3, ..., x_nx, target ⟩ from E
    FeedForward( e )
    BackPropagation( target )

  UNTIL termination conditions.
END
```

END.