Module Code: COC102

Module Title: Advanced AI Systems

Title: ANN Implementation

Student Name: Max Pearson

Student ID: B123103

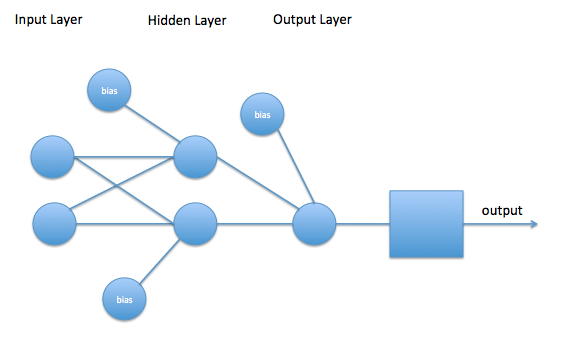
Introduction

The task was to construct a program called a Neural Network Multi-Layer Perceptron using a back propagation algorithm. This algorithm calculates errors by using mathematical functions such as: the hyperbolic tangent function, the logistic function and chain rule to derive changes from neuron to neuron.

The program works by setting 3 sets of Neurons called input, hidden and output. These sets or ‘layers’ can communicate with each other through weights, which are initially random.

*Figure1* is an example of a Multi Layer Perceptron with 2 inputs nodes, 2 hidden nodes and 1 output node, including bias for input and hidden.

Figure 1.



1. Implementation

Program Details

Classes

* **NN** ( Main Neural Network )
* **Layers** (Layers for Input, Hidden and Output neurons)
* **Layer** ( Neuron container for Input, Hidden and Output neurons)
* **Neuron** (node for calculating errors and predictions)
* **Network Instance** (Instance of Layers for viewing network state at any given time/epoch)
* **Data** (Processing data from CWK.xlsx and formatting for use with NN)

Variables and Objects

* NN
  + ***states*** : An array containing network layer instances for each epoch.
  + ***learning\_rate***: Variable parameter increasing and decreasing based on epoch performance.
  + ***Momentum:*** Used in changing weight based on neurons change.
  + ***Network****:* Uses Layers class to manipulate Neurons contained within each Layer.
* Layers
  + ***layers:*** Object containing multiple Layer Objects.
  + ***NumNeurons:*** Number of neurons contained in network.
  + ***NumLayers:*** Number of layers contained in network.
* Layer
  + ***name:*** Name of Layer.
  + ***Neurons:*** An array containing multiple Neuron Objects.
* Neuron
  + ***activation:*** Activation value for node/neuron.
  + ***n:*** Index of node/neuron to find weight of connection.
  + ***delta:*** Delta value for node/neuron.
  + ***SUM:*** The summation of all nodes/neurons connected in layer below.
  + ***Change:*** Change is the value of activation \* delta and is used for momentum in changing weight connections.
* Data
  + ***Document:*** Object storage of data from File
  + ***tagLine:*** Array containing data column names.
  + ***maxN:*** Max Value of all data for use of normalisation.
  + ***minN:*** Min Value of all data for use of normalisation.

Methods

* NN
  + ***captureState:*** Using Network Instance class clone Network Layers state and store within Network states array.
  + ***feed\_forward:*** Back propagation algorithm function that sets activations for each node/neuron, by looping through (*layer+1*) and finding the sum of weights in all weight connection (*layer*) using ***feed\_layer***.
  + ***feed\_layer:*** This function takes a layer from ***feed\_forward*** function and loops through the layer back of that given layer to return the sum of activations.
  + ***backPropagation:*** Back propagation main function that starts of at the output layer and proceed back through layers until it reaches input, while taking activations and assigning delta weights. This function uses chain rule to derive delta weighting.
  + ***updateWeights:*** Update each weight connection on each layer though each node starting from layer 1, each update is done based on the delta and activation values on each node/neuron.
  + ***updateLayerForward:*** Used as part of ***updateWeights*** to minimise code length this function is passed a layer and iterates through the next layer up to find appropriate connections between these weights and update.
  + ***weightDecay:*** In order to minimise the chances of weights becoming too large this function decays weight based on the epoch number, i.e if the epoch number is currently small the weight decay will be large, if the epoch is large the weight decay will be small.
  + ***simulated\_annealing:***
  + ***train:*** Giving the network a normalised set of training examples containing inputs and outputs, using the functions ***feed\_forward***, ***backPropagation***, ***updateWeights*** and ***bold\_driver***. the functions are able to set activations and delta weights which in turn calculate and print out errors and associated learning rate.
  + ***test:*** Giving the network a normalised set of test examples with inputs and outputs, using ***feed\_forward*** the network is able to attain predictions based upon the weightings and activations obtained through testing and the activation of the final layer.
  + ***getError:*** Calculate and return error based upon target and prediction obtained by the network.
  + ***bold\_driver:*** Change the ***learning\_rate*** variable based upon the increase or decrease of the error calculation. Increasing rate by 1% upon decrease of error, decreasing by 50% upon increase of error.
  + ***getLayer:*** Return networks layer via index
  + ***getLayerNeurons:*** Return neurons array in layers.
  + ***getWeights:*** Return networks weight via indexing.
* Layers
  + ***addLayer:*** Add Layer to Network layers and set Neurons based upon parameters passed into function.
  + ***updateLayerSize:*** When new layer is added this function is called to increase ***NumNeurons*** and ***NumLayers***.
  + ***\_\_updateWeightSpace (private function ):*** Create 2 dimensional array of weights based upon layer size and neurons.
* Layer
  + ***\_\_createLayer (private function ):*** Create variables and Objects associated with this class.
* Neuron
  + ***activate:*** Take neuron SUM and pass through activation function.
* Network Instance
  + ***set:*** Once network layers instance is created append to states in Network
  + ***get:*** Get network layers state via epoch index once network is trained.
* Data:

Functions

Program formulation

Feed Forward

Back Propagation

Update Weights

System Diagram

Figure 2

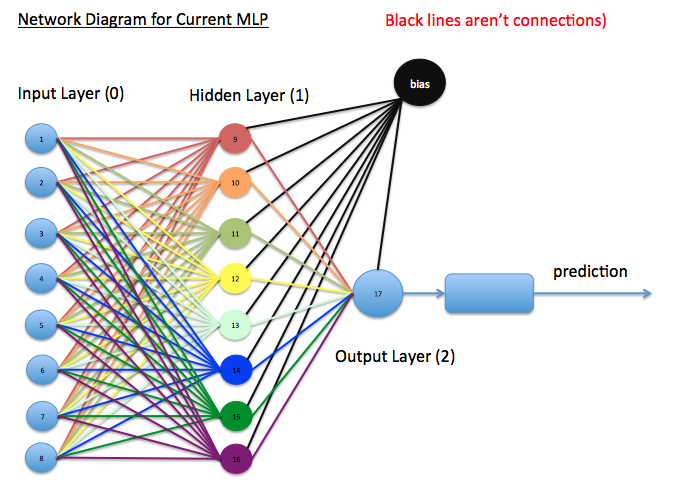
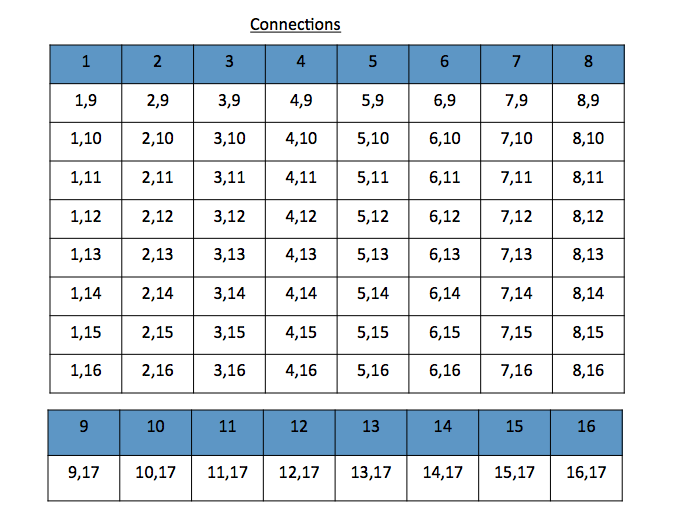


Figure 3.



ANN Configuration

Flood data is taken from CWKData.xlsx by the class named ***Data***. This populates an object which is restructured into a 2 dimensional array through the function ***createNormalisedData.*** The data set is then useable by the Neural Network and is normalised between [0,1]. To allow the use of normalisation between [-1,1] the program also includes *tanh* and *tanh’* functionality. Once the data set is normalised it looks as shown in Figure 4.

The Neural Network consists of 3 layers: input, hidden and output, each layer is its own object with multiple Neuron objects. The Network initially has a learning rate of 0.5 and a momentum of 0.9, which is subject to change by the function ***bold\_driver***. The ANN can only be trained and tested with an array formed as *Figure 4* but can take any number of different inputs and outputs.

Figure 4.

XOR = [

[[1,0],[1]],

[[1,1],[0]]

]

Data= [ [

[0.0068449795275427805,

0.000111555178380698,

0.00021707150891862612,

9.963008895833298e-06,

0.003090865453445421,

8.670215837796286e-05,

0.0076734135276339535,

0.20100374807522625 ], [ 0.002088242304387711 ] ]

];

Network Details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Initial LR. | Initial Mom. | Hidden nodes | Input nodes |
| **Network 1**  ( sigmoid ) | 0.5 | 0.9 | 8 | 8 |
| **Network 2**  ( tanh ) | 0.4 | 0.7 | 7 | 7 |
| **Network 3**  (sigmoid) | 0.5 | 0.9 | 6 | 6 |
| **Network 4**  ( tanh ) | 0.4 | 0.7 | 5 | 5 |

Network Performances

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Network 1 | Network 2 | Network 3 | Network 4 |
| epochs taken to terminate. |  |  |  |  |
| prediction accuracy. |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |