Module Code: COC102

Module Title: Advanced AI Systems

Title: ANN Implementation

Student Name: Max Pearson

Student ID: B123103

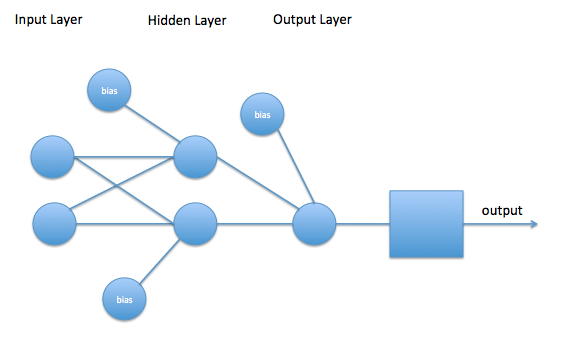
Introduction

The task was to construct a program called a Neural Network Multi-Layer Perceptron using a back propagation algorithm. This algorithm calculates errors and finds classification 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

* **Network**: Main Neural Network Class.
* **Nodes**: Contains activations and delta for each node.
* **Weighting:** Holds weights and weight changes for each layer.
* **Data** (Processing data from CWK.xlsx and formatting for use with NN.

Variables and Objects

* Network
  + ***learning\_rate***: Variable parameter increasing and decreasing based on epoch performance.
  + ***momentum:*** Used in changing weight based on neurons change.
  + ***nm\_i***: Number of Input nodes.
  + ***nm\_h***:Number of Hidden nodes.
  + ***nm\_o***: Number of Output nodes.
  + ***weights***: Container for Hidden and Output weights.
  + ***changes***:Container for Hidden and Output weight changes.
  + ***inputLayer***: Input nodes holding input activation.
  + ***hiddenLayer***: Hidden nodes holding input activation and delta weighting.
  + ***outputLayer***: Output nodes holding input activation and delta weighting.
* 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
* Weighting
* Nodes

Methods

* NN
  + ***feed\_forward:*** Back propagation algorithm function that sets activations for each node/neuron layer by layer until it reaches the output layer.
  + ***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.
  + ***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.
* Data
* Nodes
* Weighting

Functions

Program formulation

Feed Forward

Back Propagation

Update Weights

ANN Configuration

Flood data is taken from CWKData.xlsx by the class named ***Data***. This populates an object that 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 1.

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 1* but can take any number of different inputs and outputs.

Figure 1.

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

*LR* = Learning Rate parameter

*Mom* = Momentum parameter

*Hidden* = Hidden Nodes

*Inputs* = Input Nodes

*Data Set* = Number of examples taken from CWKData.xlsx

*Epoch Max* = Set Termination Value

Epoch Termination = Did the Network Terminate before Epoch Max?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Network** | **LR** | **Mom** | **Hidden** | **Inputs** | **Data Set** | **Epoch Max** | **Epoch**  **Termination** |
| **1** | 0.5 | 0.9 | 8 | 8 | 500 | 1000 |  |
| **2** | 0.5 | 0.6 | 6 | 8 | 400 | 800 |  |
| **3** | 0.5 | 0.4 | 4 | 8 | 300 | 600 |  |
| **4** | 0.4 | 0.9 | 8 | 7 | 500 | 1000 |  |
| **5** | 0.4 | 0.6 | 6 | 7 | 400 | 800 |  |
| **6** | 0.4 | 0.4 | 4 | 7 | 300 | 600 |  |
| **7** | 0.3 | 0.9 | 8 | 6 | 500 | 1000 |  |
| **8** | 0.3 | 0.6 | 6 | 6 | 400 | 800 |  |
| **9** | 0.3 | 0.4 | 4 | 6 | 300 | 600 |  |
| **10** | 0.2 | 0.9 | 8 | 5 | 500 | 1000 |  |
| **11** | 0.2 | 0.6 | 6 | 5 | 400 | 800 |  |
| **12** | 0.2 | 0.4 | 4 | 5 | 300 | 600 |  |
| **13** | 0.1 | 0.9 | 8 | 4 | 500 | 1000 |  |
| **14** | 0.1 | 0.6 | 6 | 4 | 400 | 800 |  |
| **15** | 0.1 | 0.4 | 4 | 4 | 300 | 600 |  |

*Performance Rating* = (Accuracy of Predictions) 0-10

*Time to Execute* = (measured in milliseconds)

*Training Set* = % of Data Set for Training Network

*Validation Set* = % of Data Set for Validation of Network

*Test Set* = % of Data Set for Testing Network

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Network** | **Performance Rating** | **Time To Execute** | **Training Set** | **Validation Set** | **Test Set** |
| **1** |  |  | 70 | 15 | 15 |
| **2** |  |  | 70 | 20 | 10 |
| **3** |  |  | 70 | 10 | 20 |
| **4** |  |  | 70 | 15 | 15 |
| **5** |  |  | 70 | 20 | 10 |
| **6** |  |  | 70 | 10 | 20 |
| **7** |  |  | 70 | 15 | 15 |
| **8** |  |  | 70 | 20 | 10 |
| **9** |  |  | 70 | 10 | 20 |
| **10** |  |  | 70 | 15 | 15 |
| **11** |  |  | 70 | 20 | 10 |
| **12** |  |  | 70 | 10 | 20 |
| **13** |  |  | 70 | 15 | 15 |
| **14** |  |  | 70 | 20 | 10 |
| **15** |  |  | 70 | 10 | 20 |

Network 1

Network 2

Network 3

Network 5

Network 6

Network 7

Network 8

Network 9

Network 10

Network 11

Network 12

Network 13

Network 14

Network 15