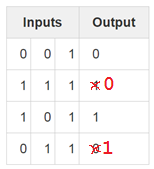
# Outline

Train a Backpropagation neural network using following training data

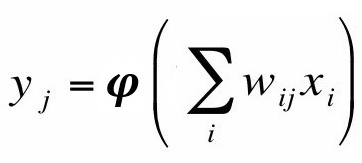
and network topology:

* 3 inputs
* 4 hidden neurons
* 1 output neuron
* Sigmoid activation function
* Weights pre-set (normally set randomly but for debugging purposes pre-set)

Training phases:

Feed forward

* From input to output

Back ward phase 

* Error

Output error

Hidden layer error

* Update weights

Reference

<https://iamtrask.github.io/2015/07/12/basic-python-network/>

<http://neuralnetworksanddeeplearning.com/>

# Create a BackPropagation Application

1. Open Visual Studio 2015
2. File->New->Project
3. Installed->Templates->Visual C#->Windows->Classic Desktop->Console Application
4. Change
   1. Name : BackPropagation
   2. Location : D:\Projects\DotNET\
   3. Solution name: BackPropagation

Create directory for solution ticked

1. Click OK
2. Copy and replace Program.cs with following code (page 2 to page 4)

using System;

namespace BackPropagation

{

class Program

{

const int NUM\_INPUT = 3;

const int NUM\_OUTPUT = 1;

const int NUM\_HIDDEN = 4;

const int NUM\_SETS = 4;

static double[,] X = new double[NUM\_SETS, NUM\_INPUT]

{ { 0, 0, 1 },

{ 0, 1, 1 },

{ 1, 0, 1 },

{ 1, 1, 1 }

};

static double[,] Y = new double[NUM\_SETS, NUM\_OUTPUT] { { 0 }, { 1 }, { 1 }, { 0 } };

static double[,] wH = new double[NUM\_INPUT, NUM\_HIDDEN]

{

{ -0.16595599, 0.44064899, -0.99977125, -0.39533485 },

{ -0.70648822, -0.81532281, -0.62747958, -0.30887855 },

{ -0.20646505, 0.07763347, -0.16161097, 0.370439 },

};

static double[,] wO = new double[NUM\_HIDDEN, NUM\_OUTPUT]

{

{ -0.5910955 }, { 0.75623487 }, { -0.94522481 }, { 0.34093502 }

};

static void Main(string[] args)

{

double[,] yH; // output from hidden layer

double[,] yO; // output from output layer

for (int iter = 0; iter < 100000; iter++)

{

// Forward progragation - from input->hidden->output

yH = sigmoidDot(X, wH);

yO = sigmoidDot(yH, wO);

// Backward progragation - from output->hidden

// Output error

double[,] eO = diff(Y,yO);

double[,] deltaO = delta(eO, yO);

// Hidden-layer error

double[,] eSumH = dot(deltaO, transpose(wO));

double[,] deltaH = delta(eSumH, yH);

// Update output

wO = updateWeights(wO,yH,deltaO);

// Update hidden-layer

wH = updateWeights(wH, X, deltaH);

// Display error every 1000 iterations

if (iter %1000 == 0)

{

double meanError = 0;

for (int n = 0; n < NUM\_OUTPUT; n++)

{

for (int w = 0; w < NUM\_SETS; w++)

{

meanError += Math.Abs(eO[w, n]);

}

meanError = meanError / (NUM\_OUTPUT \* NUM\_SETS);

}

Console.WriteLine(iter + " iteration: Mean output error:" + meanError);

}

}

// Network now trained

Console.WriteLine("Trained Hidden Weights : ");

printArray(wH);

Console.WriteLine("Trained Output Weights : ");

printArray(wO);

Console.Read();

}

static double[,] updateWeights(double[,] w, double[,] previousOutput, double[,] delta)

{

int NUM\_ROWS = w.GetLength(0);

int NUM\_COLUMS = w.GetLength(1);

double learningRate = 0.1;

double[,] temp = dot(transpose(previousOutput), delta);

for (int r = 0; r < NUM\_ROWS; r++)

{

for (int c = 0; c < NUM\_COLUMS; c++)

{

w[r, c] += learningRate \* temp[r, c];

}

}

return w;

}

static double[,] sigmoidDot(double[,] x, double[,] s)

{

int NUM\_ROWS = x.GetLength(0); // down

int NUM\_INPUTS = x.GetLength(1); // across

int NUM\_COLUMS = s.GetLength(1); // across

double[,] sigmoid = new double[NUM\_ROWS, NUM\_COLUMS];

for (int c = 0; c < NUM\_COLUMS; c++)

{

for (int r = 0; r < NUM\_ROWS; r++)

{

double sum = 0.0;

for (int w = 0; w < NUM\_INPUTS; w++)

{

sum += x[r, w] \* s[w, c];

}

sigmoid[r, c] = 1 / (1 + Math.Exp(-sum));

}

}

return sigmoid;

}

static double[,] dot(double[,] x, double[,] s)

{

int NUM\_ROWS = x.GetLength(0); // down

int NUM\_INPUTS = x.GetLength(1); // across

int NUM\_COLUMS = s.GetLength(1); // across

double[,] result = new double[NUM\_ROWS, NUM\_COLUMS];

for (int c = 0; c < NUM\_COLUMS; c++)

{

for (int r = 0; r < NUM\_ROWS; r++)

{

double sum = 0.0;

for (int w = 0; w < NUM\_INPUTS; w++)

{

sum += x[r, w] \* s[w, c];

}

result[r, c] = sum;

}

}

return result;

}

static double[,] transpose(double[,] x)

{

int NUM\_NODES = x.GetLength(0); // down

int NUM\_INPUTS = x.GetLength(1); // across

double[,] result = new double[NUM\_INPUTS, NUM\_NODES];

for (int node = 0; node < NUM\_NODES; node++)

{

for (int w = 0; w < NUM\_INPUTS; w++)

{

result[w, node] = x[node, w];

}

}

return result;

}

static double[,] diff(double[,] Y, double[,] v)

{

double[,] result = new double[NUM\_SETS, NUM\_OUTPUT];

for (int s = 0; s < NUM\_SETS; s++)

{

for (int y = 0; y < NUM\_OUTPUT; y++)

{

result[s, y] = Y[s, y] - v[s, y];

}

}

return result;

}

static double[,] delta(double[,] error, double[,] output)

{

int NUM\_SETS = error.GetLength(0);

int NUM\_NODES = error.GetLength(1);

double[,] result = new double[NUM\_SETS, NUM\_NODES];

for (int x = 0; x < NUM\_SETS; x++)

{

for (int y = 0; y < NUM\_NODES; y++)

{

result[x, y] = error[x, y] \* output[x, y] \* (1 - output[x, y]);

}

}

return result;

}

static void printArray(double[,] w)

{

for (int i = 0; i < w.GetLength(0); i++)

{

Console.Write("[ ");

for (int k = 0; k < w.GetLength(1); k++)

{

Console.Write(w[i, k].ToString() + " ");

}

Console.Write(" ]");

Console.WriteLine();

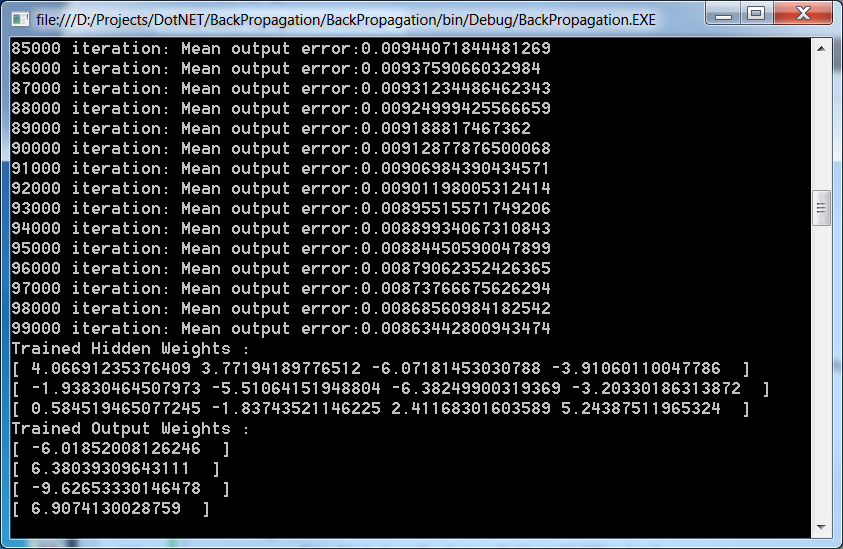
}

}

}

}

1. Run the program and if you get trained hidden weights (and mean error) as below, congratulations you have trained (possibly) your 1st neural network.



Study the code and see if you can see how the equations are implemented

What are methods **dot**(x,y) and **transpose**(x) lines 124 & 147 doing?

1. Update the learningRate (0.00 to 1.00) and observe how the Mean Error changes.
2. Update the application with these weights and see what output you get if you enter the same or different input values.