SE2NN11 MLP Lab 1 Report Sheet 2014/15				
Student Name: Abdelrahmane Bray	**Marks**			
Date: 21/10/2014	**Missing**	**Poor**	**Ok**	**Good**
Introduction	0	1	2	3

The aim of this practical is to develop, through coding, a functional program which can simulate an artificial neural network.

Most of the program already is functional, and all that is expected during this practical is to program key methods which take care of the learning algorithm.

The methods relating to calculating the output for single-layered linearly-activated networks as well as single-layered sigmoidal-activated networks are fully implemented during the practical.

Single-layered networks function in the same way a single perceptron would, each neuron has its own output, weights and targets but all neurons share the same inputs. Single layers can thus learn to solve different problems from the same inputs.

Output of Untrained LinearLayerNetwork network	**Missing**	**Poor**	**Correct**
	0	1	2

```
Richard J. Mitchell's Perceptron Network Program
        Adapted by Abdelrahmane Bray [Autumn 2014]

######################


Network is: for Linear-activation
Initial weights seed [0]
Learning rate: [0.2]. Momentum: [0]

MENU:: Select one of the following:
[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit
>t
       Inputs           Targets           Actuals           Rescaled
      0    0 :     0    0    0 :    0.2   0.3   0.4 :    0    0    0
      0    1 :     0    1    1 :    0.5   0.4   0.6 :    0    0    1
      1    0 :     0    1    1 :    0.7   0.8   0.5 :    1    1    0
      1    1 :     1    1    0 :      1   0.9   0.7 :    1    1    1
Mean Sum Square Errors are   0.195  0.125  0.265
% Correct Classifications     75    75    50
```

```
SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>l

        Epoch     0
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265

        Epoch     1
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265

        Epoch     2
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265

        Epoch     3
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265

        Epoch     4
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265

        Epoch     5
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265
```

```
          Epoch      6
AndOrXor : Mean Sum Square Errors are    0.195   0.125   0.265


SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>
```

LinearLayerNetwork Functions – mark scheme	Code			Comments		
Function ReturnTheWeights	0	1	2	0	1	2

```
///<summary>
/// Copies the array of weights stored in the network-object to the currentWeights array
///
///<argument="double currentWeights[]">Array containing the current weights</argument>
///</summary>
void LinearLayerNetwork::ReturnTheWeights (double currentWeights[]) {


        //Copies as many elements as there are weights, from the weights array, to the currentWeights array
        dcopy (numWeights, weights, currentWeights);
}
```

Function FindDeltas	0	1	2	3	0	1	2

```
///<summary>
/// Finds and stores the deltas from the errors. It is assumed the size of errors-array and the size of deltas-array are equal.
/// Equation (for a linear system):
/// delta = Error
///
///<argument="const double errors[]">Array of errors used to find deltas</argument>
///</summary>
void LinearLayerNetwork::FindDeltas (const double errors[]) {

	//only copying has to be done, there are as many errors as there are outputs and as many outputs as there are neurons
	dcopy(numNeurons, errors, deltas);

}
```

Function ChangeAllWeights	0	1	2	3	4	0	1	2	3

```
///<summary>
/// Calculates and stores the new weights from the errors.
/// Equation (for a linear system):
/// new weight = old weight + ( (error * input * learning_rate) + momentum + old change in weight)
///
///<argument="const double inputs[]">Array of inputs used to find the new weights</argument>
///<argument="const double learningParameters[]"> Array containing the parameters: {learning-rate, momentum}</argument>
///</summary>
void LinearLayerNetwork::ChangeAllWeights (const double inputs[], const double learningParameters[]) {

        //Used to keep track of the current input
        double current_input;

        //Used to keep track of which weight is being used
        int weight_index = 0;

        //For each neuron in the layer
        for(int neuron_index=0; neuron_index < numNeurons; neuron_index++)
        {
```

```
            //For each input in the input-array
            for(int input_index=0; input_index < numInputs + 1; input_index++)
            {
                //IF bias weight
                if((input_index % (numInputs + 1))==0) current_input = 1;
                else current_input = inputs[input_index - 1];

                //Equate  (delta * input * learning_rate)  ADD  (momentum * previous_delta)
                deltaWeights[weight_index] = (current_input * deltas[neuron_index] * learningParameters[0])
                                                                  + (deltaWeights[weight_index] * learningParameters[1]);

                //New weight = old weight + change in weight
                weights[weight_index] += deltaWeights[weight_index];

                //Move on to the next weight
                weight_index++;
            }
        }
}
```

Program output with default weights, after training with:	**Missing**	**Poor**	**Close**	**Correct**
a learning rate of 0.1 and momentum of 0.3	0	1	2	3

Richard J. Mitchell's Perceptron Network Program

Adapted by Abdelrahmane Bray [Autumn 2014]

######################

Network is: for Linear-activation

Initial weights seed [0]

Learning rate: [0.2]. Momentum: [0]

MENU:: Select one of the following:

[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit

>c

Enter Learning Rate: [range 0 to 1]   0.1

Enter Momentum: [range 0 to 1]   0.3

######################

Network is: for Linear-activation

```
Initial weights seed [0]
Learning rate: [0.1]. Momentum: [0.3]

MENU:: Select one of the following:
[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit
>t
       Inputs           Targets           Actuals          Rescaled
     0    0 :      0    0    0 :    0.2   0.3   0.4 :     0    0    0
     0    1 :      0    1    1 :    0.5   0.4   0.6 :     0    0    1
     1    0 :      0    1    1 :    0.7   0.8   0.5 :     1    1    0
     1    1 :      1    1    0 :      1   0.9   0.7 :     1    1    1
Mean Sum Square Errors are   0.195   0.125   0.265
% Correct Classifications     75    75    50

SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>l

       Epoch     0
AndOrXor : Mean Sum Square Errors are    0.185    0.13   0.338

       Epoch     1
AndOrXor : Mean Sum Square Errors are    0.121   0.103   0.332
```

```
        Epoch    2
AndOrXor : Mean Sum Square Errors are    0.107  0.0959   0.331

        Epoch    3
AndOrXor : Mean Sum Square Errors are      0.1  0.0917   0.329

        Epoch    4
AndOrXor : Mean Sum Square Errors are   0.0961  0.0887   0.327

        Epoch    5
AndOrXor : Mean Sum Square Errors are   0.0929  0.0865   0.326

        Epoch    6
AndOrXor : Mean Sum Square Errors are   0.0904  0.0848   0.324

SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>p
       Inputs            Targets            Actuals             Rescaled
      0     0 :     0     0     0 :  -0.13   0.349   0.443 :    0     0     0
      0     1 :     0     1     1 :  0.242   0.706   0.502 :    0     1     1
      1     0 :     0     1     1 :  0.329   0.822   0.441 :    0     1     0
```

1 1 : 1 1 0 : 0.701 1.18 0.5 : 1 1 1 Mean Sum Square Errors are 0.0682 0.068 0.252 % Correct Classifications 100 100 50 SELECT: [L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort. >			
Weights of network after training	0	1	2
SELECT: [L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort. >w -0.13,0.459,0.373,0.349,0.473,0.357,0.443,-0.00206,0.0596, SELECT: [L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort. > Weights are = {-0.13, 0.459, 0.373, 0.349, 0.473, 0.357, 0.443, -0.00206, 0.0596}			

Code for SigmoidalLayerNetwork Functions – mark scheme	**Code**				**Comments**		
Function SigmoidalLayerNetwork CalcOutputs	0	1	2	3	0	1	2

```
///<summary>
///        Calculates the outputs of the sigmoidal layer
/// Equation:
/// sum = input * weight
/// output = 1 / (1 + exp( - sum ) )
///
///<argument="const double inputs[]">Array containing the inputs</argument>
///</summary>
void SigmoidalLayerNetwork::CalcOutputs(const double inputs[]) {
        // Calculate outputs being Sigmoid (WeightedSum of ins)

        //Tracks which weight is being accessed
        int weight_index = 0;

        for (int neuron_counter=0; neuron_counter < numNeurons; neuron_counter++)
        {
                //Processes each neuron in order
                outputs[neuron_counter] = weights[weight_index++];

                //The summation is done here
```

```
		for (int input_counter=0; input_counter < numInputs; input_counter++)
		{
			outputs[neuron_counter] += inputs[input_counter] * weights[weight_index++];

			outputs[neuron_counter] = 1 / (1 + exp( -1 * output[neuron_counter]));
		}
	}
}
```

Function SigmoidalLayerNetwork FindDeltas	0	1	2	0	1	2

```
///<summary>
///	Calculates the outputs of the sigmoidal layer
/// Equation:
/// Temp_output = input * weight
/// output = 1 / (1 + exp( - sum ) )
///
///<argument="const double inputs[]">Array containing the inputs</argument>
///</summary>
void SigmoidalLayerNetwork::CalcOutputs(const double inputs[]) {
	// Calculate outputs being Sigmoid (WeightedSum of ins)

	//Makes use of inheritance to find outputs as done with linear-activation networks
	LinearLayerNetwork::CalcOutputs(inputs);
```

```
        for (int neuron_counter=0; neuron_counter < numNeurons; neuron_counter++)
        {
                //Actual output = 1 / (1 + exp( - Temp_output ) )
                outputs[neuron_counter] = 1 / (1 + exp( -1 * outputs[neuron_counter]));
        }
}
```

Program output with default weights: **a learning rate of 0.15 and momentum of 0.4 :** **show state before, during and then after training**	**Missing**	**Wrong**	**Correct**
	0	1	2

Before

```
Richard J. Mitchell's Perceptron Network Program
        Adapted by Abdelrahmane Bray [Autumn 2014]

######################


Network is: for Linear-activation
Initial weights seed [0]
Learning rate: [0.2]. Momentum: [0]

MENU:: Select one of the following:
[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit
>n
SELECT NETWORK:
[L]inear. [S]igmoidal. [X]OR. [O]ther non-Separable. [C]lassifier. [N]umerical Probability.
>s

######################
```

```
Network is: for Sigmoidal-activation
Initial weights seed [0]
Learning rate: [0.2]. Momentum: [0]

MENU:: Select one of the following:
[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit
>c
Enter Learning Rate: [range 0 to 1]    0.15
Enter Momentum: [range 0 to 1]         0.4

######################



Network is: for Sigmoidal-activation
Initial weights seed [0]
Learning rate: [0.15]. Momentum: [0.4]

MENU:: Select one of the following:
[T]est Network. Set [N]etwork. Set Learning-[C]onstants. [I]nitialise Random Seed. [Q]uit
>t
        Inputs          Targets          Actuals          Rescaled
```

```
      0      0 :      0      0      0 :    0.55   0.574   0.599 :      1      1      1
      0      1 :      0      1      1 :   0.622   0.599   0.646 :      1      1      1
      1      0 :      0      1      1 :   0.668    0.69   0.622 :      1      1      1
      1      1 :      1      1      0 :   0.731   0.711   0.668 :      1      1      1
Mean Sum Square Errors are    0.302   0.168   0.268
% Correct Classifications       25      75      50

SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>l


            Epoch      0
AndOrXor : Mean Sum Square Errors are    0.301   0.168   0.272


            Epoch    200
AndOrXor : Mean Sum Square Errors are   0.0577  0.0369   0.255


            Epoch    400
AndOrXor : Mean Sum Square Errors are   0.0318   0.018   0.255


            Epoch    600
AndOrXor : Mean Sum Square Errors are   0.0212  0.0114   0.255
```

```
          Epoch    800
AndOrXor : Mean Sum Square Errors are   0.0156 0.00819   0.255


          Epoch   1000
AndOrXor : Mean Sum Square Errors are   0.0122 0.00634   0.255


SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>p
         Inputs              Targets              Actuals              Rescaled
      0      0 :      0      0      0 :  0.00302    0.12   0.505 :      0      0      1
      0      1 :      0      1      1 :    0.119  0.926   0.502 :      0      1      1
      1      0 :      0      1      1 :    0.119  0.926   0.499 :      0      1      0
      1      1 :      1      1      0 :    0.858  0.999   0.496 :      1      1      0
Mean Sum Square Errors are   0.0122 0.00633    0.25
% Correct Classifications      100     100      50


SELECT:
[L]earn. [P]resent Data. [C]hange Learning Constants. Find [W]eights. [S]ave Learnt Data. [A]bort.
>w
-5.8,3.8,3.8,-1.99,4.52,4.52,0.0217,-0.0238,-0.014,
```

Discussion (on code and results)	**Missing .. Ok ... Excellent**				
	0	1	2	3	4

Linearly-activated neurons can solve the problems of boolean OR and AND in a small amount of epochs, but not that of XOR.

This is because those problems are linearly separable.

Sigmoidal-activated neurons can solve the problems of OR, AND and XOR, but in a large amount of epochs.

The learning rate affects the magnitude of the change in weights.

Momentum also affects the magnitude of the change in weights, but the momentum grows in magnitude when learning is done in the right direction.

Most of the code had to be changed, as the variable name used were not intuitive to read, and on several occasions bad programming practice was in place.

Conclusion	**Missing .. Ok ... Excellent**				
	0	1	2	3	4

To conclude, the linearly activated neurons can learn the “simple” problems of boolean OR and AND, but not that of XOR because it is not possible to separate the target outputs into distinct sets. This problem can be overcome using a sigmoidal activation network, however the learning takes much longer.

Momentum and learning rate affect the rate at which the weights change, and when correctly utilised, allow the neuron to learn quickly and effectively. Smaller learning rates allow for better “fine-tuning” of the weights, whereas greater learning rates allow for faster learning.

Real-world applications of single-layered networks are scarce, as most problems it can solve can be easily overcome using other methods.

However, multiple-layers of perceptrons may have a very numerous amount of applications, most of which are in pattern-manipulation.

Self Evaluation (answer yes/no/maybe)	Your View	Markers View
My code works fully	YES	
My code is clear and concise	YES	
Each function has good comments explaining what it does and its arguments	YES	
The code implementing the functions are well explained	YES	
I understand the code in the library module	YES	

Write below any issues you have or any questions you would like answered

Markers Comments	Total Mark / 45