|  |  |
| --- | --- |
| ***CS2NN16 MLP Lab 1 Report Sheet 2018/19*** | |
| **Student Number: 26007252** | **Date : 31/10/18** |

|  |  |
| --- | --- |
| **Introduction** | **Mark / 3** |
| In this lab practical I will use the neural network code to create neural network layers which can solve logical problems such as AND and OR. Some of the methods of the code will be modified or implemented to add more functionality to the code. Both linear and sigmoidal network layers will be implemented and compared to demonstrate how each solves the logical problems. | |

|  |  |
| --- | --- |
| **Output of Untrained LinearLayerNetwork network** | **Mark / 1** |
| Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.200 0.300 0.400 0 0 0  0 1 0 1 1 0.500 0.400 0.600 1 0 1  1 0 0 1 1 0.700 0.800 0.500 1 1 1  1 1 1 1 0 1.000 0.900 0.700 1 1 1  Over Set : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Weights  Epoch 1 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 2 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 3 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 4 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 5 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 6 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Epoch 7 : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.200 0.300 0.400 0 0 0  0 1 0 1 1 0.500 0.400 0.600 1 0 1  1 0 0 1 1 0.700 0.800 0.500 1 1 1  1 1 1 1 0 1.000 0.900 0.700 1 1 1  Over Set : SSE 0.1950 0.1250 0.2650 : %Correct 50 75 75  Weights | |

|  |  |  |
| --- | --- | --- |
| **LinearLayerNetwork** | | |
| **getWeights** | **Code Mark / 2** | **Comment / 2** |
| /\*\*  \* return all the weights in the layer as a string each separated by spaces  \* **@return** s the string representation of all weights  \*/  **public** String getWeights() {  String s = "";  **for** (Double w:weights){ // for each weight in weights  s += String.*format*("%.5f", w) + " "; // Append to s the value of weight w which is formatted to 5 decimal points  }  **return** s; // return the result  } | | |

|  |  |
| --- | --- |
| **Weights of Untrained LinearLayerNetwork network** | **Mark / 1** |
| Weights 0.20000 0.50000 0.30000 0.30000 0.50000 0.10000 0.40000 0.10000 0.20000 | |

|  |  |  |
| --- | --- | --- |
| **findDeltas** | **Code / 2** | **Comments / 2** |
| /\*\*  \* find deltas using a list of errors  \* **@param** errors the list of errors for the neuron  \*/  **protected** **void** findDeltas(ArrayList<Double> errors) {  **for** (**int** cnt = 0; cnt < errors.size();cnt++){ // for each error in errors  deltas.set(cnt, errors.get(cnt)); // set the delta (of the same index as the error's index) as the error  }  } | | |

|  |  |  |
| --- | --- | --- |
| **changeAllWeights** | **Code / 2** | **Comments / 2** |
| /\*\*  \* change all the weights in layer  \* **@param** ins the list of inputs  \* **@param** learnRate the learning rate  \* **@param** momentum the amount of momentum  \*/  **protected** **void** changeAllWeights(ArrayList<Double> ins, **double** learnRate, **double** momentum) {  **for**(**int** neuron = 0; neuron < numNeurons; neuron++){ // for every neuron  changeOneWeight(neuron, -1, 1, learnRate, momentum); // calculate the change in weight for the bias weight of each neuron  **for** (**int** weight = 0; weight < ins.size(); weight++){ // for each weight between each input and the neuron (excluding the bias weight because that has already been accounted for)  changeOneWeight(neuron, weight, ins.get(weight), learnRate, momentum); // change a particular weight of the neuron using the input that the weight is connected to  }  }  } | | |

|  |  |  |
| --- | --- | --- |
| **changeOneWeight** | **Code / 3** | **Comments / 2** |
| /\*\*  \* change a given weight  \* **@param** wNeuron the index of the weight's neuron  \* **@param** wWeight the weight's index  \* **@param** theIn the input for the weight  \* **@param** learnRate the learning rate  \* **@param** momentum the amount of momentum  \*/  **private** **void** changeOneWeight(**int** wNeuron, **int** wWeight, **double** theIn, **double** learnRate, **double** momentum) {  /\*  \* the new wWeight is the index of that particular weight for its respective neuron  \*/  wWeight = weightIndex(wNeuron, wWeight);    /\*  \* the change in weight (wWeight) is the learning rate  \* multiplied by the delta of that particular neuron  \* multiplied by the input of that particular neuron  \* plus the momentum multiplied by the previous change in weight  \*/  changeInWeights.set(wWeight, (learnRate \* deltas.get(wNeuron) \* theIn + momentum\*changeInWeights.get(wWeight)));    /\*  \* After calculating the new change in weight the current weight  \* is equal to itself plus the new change in weight  \*/  weights.set(wWeight, weights.get(wWeight) + changeInWeights.get(wWeight));    } | | |

|  |  |  |
| --- | --- | --- |
| **weightIndex** | **Code / 1** | **Comments / 1** |
| /\*\*  \* return index into weights array list of the wWeight'th weight of the wNeuron'th neuron  \* **@param** wNeuron the index of the neuron  \* **@param** wWeight the index of the weight  \* **@return** the actual index in weights for this particular weight in this particular neuron  \*/  **private** **int** weightIndex (**int** wNeuron, **int** wWeight) {  /\*  \* the index of a weight for a particular neuron is that neuron's index  \* multiplied by the number of weights per neuron  \* plus the weight itself  \* plus 1 for the bias weight  \*/    **return** wNeuron \* (numWeights / numNeurons) + wWeight + 1;  } | | |

|  |  |
| --- | --- |
| **Modify main so learning rate is 0.1 momentum is 0.3. Paste below how the SSE changes during training, and the data set and weights after 10 epochs of training** |  |
| **Mark /3** |
| Epoch 1 : SSE 0.1849 0.1303 0.3380 : %Correct 75 75 50  Epoch 2 : SSE 0.1211 0.1031 0.3315 : %Correct 100 100 25  Epoch 3 : SSE 0.1066 0.0959 0.3310 : %Correct 100 100 25  Epoch 4 : SSE 0.1003 0.0917 0.3293 : %Correct 100 100 25  Epoch 5 : SSE 0.0961 0.0887 0.3274 : %Correct 100 100 25  Epoch 6 : SSE 0.0929 0.0865 0.3258 : %Correct 100 100 25  Epoch 7 : SSE 0.0904 0.0848 0.3244 : %Correct 100 100 25  Epoch 8 : SSE 0.0883 0.0836 0.3232 : %Correct 100 100 25  Epoch 9 : SSE 0.0867 0.0827 0.3223 : %Correct 100 100 25  Epoch 10 : SSE 0.0853 0.0820 0.3215 : %Correct 100 100 25  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 -0.169 0.332 0.474 -0 0 0  0 1 0 1 1 0.246 0.728 0.505 0 1 1  1 0 0 1 1 0.307 0.793 0.452 0 1 0  1 1 1 1 0 0.722 1.189 0.484 1 1 0  Over Set : SSE 0.0652 0.0657 0.2508 : %Correct 100 100 75  Weights -0.16924 0.47649 0.41478 0.33211 0.46119 0.39605 0.47393 -0.02181 0.03148 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Code for SigmoidLayerNetwork – mark scheme** | | | | |
| **SigmoidLayerNetwork CalcOutputs** | | **Code / 3** | **Comments / 2** | |
| /\*\*  \* calcOutputs of neuron  \* **@param** nInputs used to calculate the outputs  \*  \*/  **protected** **void** calcOutputs(ArrayList<Double> nInputs) {  **super**.calcOutputs(nInputs); // call the parent class calcOutputs method to get the outputs  **for** (**int** neuron = 0; neuron < numNeurons; neuron++){ // for each neuron  outputs.set(neuron, (1.0/(1.0 + Math.*exp*(-(outputs.get(neuron)))))); // set the output of the neuron as the sigmoid version using the output of the neuron  }  } | | | | |
| **SigmoidLayerNetwork findDeltas** | **Code / 2** | | | **Comments / 2** |
| /\*\*  \* find deltas using a list of errors  \* **@param** errors the list of errors which are used to calculate each delta  \*/  **protected** **void** findDeltas(ArrayList<Double> errors) {  // write code to set delta as error \* deriv activation  **for** (**int** neuron = 0; neuron < numNeurons; neuron++){ // for each neuron    /\*  \* assign the delta as the error multiplied by the output multiplied by (1.0 - the output)  \*/  deltas.set(neuron, errors.get(neuron) \* outputs.get(neuron) \* (1.0 - outputs.get(neuron)));  }  } | | | | |

|  |  |
| --- | --- |
| **Program output with default weights, a learning rate of 0.15 and momentum of 0.4 : show state *before*, *during* and then *after* 1000 epochs of training** | **Mark /3** |
| Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.550 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 0.690 0.622 1 1 1  1 1 1 1 0 0.731 0.711 0.668 1 1 1  Over Set : SSE 0.3021 0.1677 0.2682 : %Correct 25 75 50  Weights 0.20000 0.50000 0.30000 0.30000 0.50000 0.10000 0.40000 0.10000 0.20000  Epoch 100 : SSE 0.0914 0.0657 0.2555 : %Correct 100 100 0  Epoch 200 : SSE 0.0579 0.0371 0.2555 : %Correct 100 100 0  Epoch 300 : SSE 0.0415 0.0246 0.2555 : %Correct 100 100 0  Epoch 400 : SSE 0.0319 0.0180 0.2555 : %Correct 100 100 0  Epoch 500 : SSE 0.0256 0.0140 0.2555 : %Correct 100 100 0  Epoch 600 : SSE 0.0212 0.0114 0.2555 : %Correct 100 100 0  Epoch 700 : SSE 0.0180 0.0096 0.2555 : %Correct 100 100 0  Epoch 800 : SSE 0.0156 0.0082 0.2555 : %Correct 100 100 0  Epoch 900 : SSE 0.0137 0.0072 0.2555 : %Correct 100 100 0  Epoch 1000 : SSE 0.0122 0.0064 0.2555 : %Correct 100 100 0  Inputs Targets Raw Ops Outputs  x1 x2 AND OR XOR AND OR XOR AND OR XOR  0 0 0 0 0 0.003 0.120 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  Over Set : SSE 0.0122 0.0063 0.2500 : %Correct 100 100 50  Weights -5.79597 3.79637 3.79458 -1.98933 4.52081 4.51989 0.02171 -0.02381 -0.01400 | |

|  |  |
| --- | --- |
| **SigmoidLayerNetwork Other data Set**  **In GUI on Other data, set learning rate to 0.3 and momentum to 0.2**  **Initialise, Present, Learn, Present and show result below:** | **Mark / 2** |
| Sigmoid Layer : Other Learn Rate 0.30 Momentum 0.20 Seed 100  Inputs Targets Raw Ops Outputs  x y O1 O2 O1 O2 O1 O2  0.1 1.2 1 0 0.687 0.706 1 1  0.7 1.8 1 0 0.650 0.762 1 1  0.8 1.6 1 0 0.620 0.758 1 1  1.0 0.8 0 0 0.525 0.729 1 1  0.3 0.5 1 1 0.605 0.679 1 1  0.0 0.2 1 1 0.625 0.647 1 1  -0.3 0.8 1 1 0.710 0.664 1 1  -0.5 -1.5 0 1 0.562 0.517 1 1  -1.5 -1.3 0 1 0.716 0.470 1 0  Over Set : SSE 0.2055 0.3377 : %Correct 66 44  Epoch 50 : SSE 0.0902 0.0529 : %Correct 88 100  Epoch 100 : SSE 0.0782 0.0357 : %Correct 88 100  Epoch 150 : SSE 0.0705 0.0272 : %Correct 88 100  Epoch 200 : SSE 0.0645 0.0219 : %Correct 100 100  Epoch 250 : SSE 0.0595 0.0182 : %Correct 100 100  Epoch 300 : SSE 0.0551 0.0156 : %Correct 100 100  Epoch 350 : SSE 0.0512 0.0135 : %Correct 100 100  Epoch 400 : SSE 0.0477 0.0119 : %Correct 100 100  Epoch 450 : SSE 0.0445 0.0106 : %Correct 100 100  Epoch 500 : SSE 0.0417 0.0096 : %Correct 100 100  Inputs Targets Raw Ops Outputs  x y O1 O2 O1 O2 O1 O2  0.1 1.2 1 0 0.999 0.176 1 0  0.7 1.8 1 0 0.999 0.001 1 0  0.8 1.6 1 0 0.996 0.002 1 0  1.0 0.8 0 0 0.377 0.080 0 0  0.3 0.5 1 1 0.749 0.822 1 1  0.0 0.2 1 1 0.667 0.984 1 1  -0.3 0.8 1 1 0.997 0.875 1 1  -0.5 -1.5 0 1 0.001 1.000 0 1  -1.5 -1.3 0 1 0.213 1.000 0 1  Over Set : SSE 0.0402 0.0094 : %Correct 100 100 | |

|  |  |
| --- | --- |
| **SigmoidLayerNetwork Other data Set -**  **Tadpole plots** | **Mark / 1** |
|  | |

|  |  |
| --- | --- |
| **Discussion (on code and results)** | **Mark / 4** |
| It is interesting how with the Linear Neuron Network having the default learning rate and momentum the SSEs are decreasing at a lesser rate than that of the SSEs when the learning rate is 0.1 and the momentum is 0.3. | |
| **Conclusion** | **Mark / 4** |
| Momentum can be used to increase, or maintain, the rate at which the SSEs decrease even when decreasing the learning rate. Sigmoidal activation can take a greater number of epochs to reduce its SSEs but it can get those SSEs to a lower rate than that of linear activation. With sigmoidal activation the outputs for each problem were much closer to their actual target than that of the linear activation. When using a lower learning rate and a greater momentum, the linear neuron had found 100% correct outputs for the AND and OR problems by an earlier epoch than that of the neuron with the default learning rate and momentum. | |

|  |  |  |
| --- | --- | --- |
| **Self Evaluation (answer yes/no/maybe)** | **Your View** | **Markers View** |
| My code works fully | **Yes** |  |
| My code is clear and concise | **Maybe** |  |
| 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 these classes | **Yes** |  |

|  |
| --- |
| **Write below any issues you have or any questions you would like answered** |
|  |

|  |  |
| --- | --- |
| **Markers Comments** | **Total Mark / 50** |
|  | |