|  |  |  |  |
| --- | --- | --- | --- |
| ***CS2NN16 MLP Lab 2 Report Sheet 2018/19*** | | | |
| **Student Number:** | **Date:** | | |
| **Introduction** | | **Mark / 2** | |
| << write here an introduction >> | | | |
| **Weights of Untrained MultiLayerNetwork network** | | | **Mark / 1** |
| Weights 0.86252 -0.15580 0.28289 0.83499 -0.50600 -0.86445 0.03650 -0.43044 0.48121 | | | |

|  |  |  |
| --- | --- | --- |
| **Code for ‘weights’ MultiLayerNetwork Functions** | | |
| **numWeights** | **Code :**  / 1 | **Comments:**  / 1 |
| /\*\*  \* return how many weights there are in the network  \* **@return**  \*/  **public** **int** **numWeights**() {  **return** numWeights + nextLayer.numWeights(); //Gets the current number of weights and adds the next layers number of weights  } | | |
| **getWeights** | **Code / 2** | **Comments** : / 2 |
| /\*\*  \* return the weights in the whole network as a string  \* **@return** the string  \*/  **public** **String** **getWeights**() {  **return** **super**.getWeights() + nextLayer.getWeights(); //Gets the weights for the current layer and concatinates with the next layers weights  } | | |

|  |  |  |
| --- | --- | --- |
| **Code for calcOutputs function** | **Code :**  /2 | **Comments :**  /1 |
| /\*\*  \* calcOutputs of network  \* **@param** nInputs  \*  \*/  **protected** **void** **calcOutputs**(**ArrayList**<Double> nInputs) {  **super**.calcOutputs(nInputs); //Calculate outputs for this layer  nextLayer.calcOutputs(**super**.outputs); //Calculate outputs for next layer  } | | |

|  |  |  |
| --- | --- | --- |
| **Code for depositOutputs function** | **Code :**  /2 | **Comments :**  /1 |
|  |  |  |
| /\*\*  \* outputsToDataSet of the network to the data set  \* **@param** ct  \* **@param** d  \*/  **protected** **void** **outputsToDataSet** (**int** ct, **DataSet** d) {  d.storeOutputs(ct, nextLayer.outputs); //Stores the outputs of the next layer in the data set  } | | |

|  |  |
| --- | --- |
| **Outputs of Untrained MultiLayerNetwork network** | **Mark / 1** |
| Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.517 1  0 1 1 0.487 0  1 0 1 0.507 1  1 1 0 0.475 0  Over Set : SSE 0.2500 : %Correct 50 | |

|  |  |  |
| --- | --- | --- |
| **Code for ‘learning functions’ – mark scheme** | | |
| **LinearLayerNetwork weightedDeltas (version for one output)** | **Code :**  / 2 | **Comments :**  / 2 |
| /\*\*  \* Calculate the errors in the previous layer, being the deltas in this layer \* associated weights  \* this is used in the back propagation algorithm  \*  \* **@return** arraylist of errors  \*/  **public** **ArrayList**<Double> **weightedDeltas**() {  **ArrayList**<Double> **wtDeltas** = **new** ArrayList<Double>(); // create array for answer  **for**(**int** **i** = 0; i < numInputs; i++) {  wtDeltas.add(deltas.get(0) \* weights.get(i + 1));  } //Loops through each input (neuron of previous layer) and multiplies them by the weights  **return** wtDeltas;  } | | |
| **MultiLayerNetwork findDeltas** | **Code :**  / 3 | **Comments :**  / 2 |
| /\*\*  \* find the deltas in the whole network  \* **@param** errors  \*/  **protected** **void** **findDeltas**(**ArrayList**<Double> errors) {  nextLayer.findDeltas(errors); //find deltas for next layer using set errors  **super**.findDeltas(nextLayer.weightedDeltas()); //find deltas for this layer using weighted deltas of next  } | | |
| **MultiLayerNetwork changeTheWeights** | **Code :**  / 2 | **Comments :**  / 2 |
| /\*\*  \* change all the weights in the network, in this layer and the next  \* **@param** ins array list of the inputs to the neuron  \* **@param** learnRate learning rate: change is learning rate \* input \* delta  \* **@param** momentum momentum constant : change is also momentun \* change in weight last time  \*/  **protected** **void** **changeAllWeights**(**ArrayList**<Double> ins, **double** learnRate, **double** momentum) {  **super**.changeAllWeights(ins, learnRate, momentum); //change weights for this layer  nextLayer.changeAllWeights(**super**.outputs, learnRate, momentum); //change weights for next layer using outputs from this layer as inputs  } | | |

|  |  |
| --- | --- |
| **Program output DURING training with ‘Picton’s weights’, a learning rate of 0.4 and momentum of 0.7 (ie show SSE at specific epochs)** | **Mark / 1** |
| Epoch 200 : SSE 0.2611 : %Correct 50  Epoch 400 : SSE 0.0532 : %Correct 100  Epoch 600 : SSE 0.0044 : %Correct 100  Epoch 800 : SSE 0.0021 : %Correct 100  Epoch 1000 : SSE 0.0013 : %Correct 100  Epoch 1200 : SSE 0.0010 : %Correct 100  Epoch 1400 : SSE 0.0007 : %Correct 100  Epoch 1600 : SSE 0.0006 : %Correct 100  Epoch 1800 : SSE 0.0005 : %Correct 100  Epoch 2000 : SSE 0.0004 : %Correct 100 | |

|  |  |
| --- | --- |
| **Output of TRAINED network with ‘Picton’s weights’, a learning rate of 0.4 and momentum of 0.7 (ie output when you Present data to trained network)** | **Mark / 1** |

Inputs Targets Raw Ops Outputs

x1 x2 XOR XOR XOR

0 0 0 0.019 0

0 1 1 0.980 1

1 0 1 0.980 1

1 1 0 0.025 0

Over Set : SSE 0.0004 : %Correct 100

|  |  |  |
| --- | --- | --- |
| **Final Version of Code for weightedDeltas** | **Code :**  / 2 | **Comments :** / 2 |
| /\*\*  \* Calculate the errors in the previous layer, being the deltas in this layer \* associated weights  \* this is used in the back propagation algorithm  \*  \* **@return** arraylist of errors  \*/  **public** **ArrayList**<Double> **weightedDeltas**() {  **ArrayList**<Double> **wtDeltas** = **new** ArrayList<Double>(); // create array for answer  **for**(**int** **i** = 0; i < numInputs; i++) {  **double** **ans** = 0;  **for**(**int** **n** = 0; n < numNeurons; n++) {  ans+= deltas.get(n) \* weights.get(weightIndex(n, i));  } //Loops through each input (neuron of previous layer) and multiplies them by the weights  wtDeltas.add(ans);  }  **return** wtDeltas;  } | | |

|  |  |
| --- | --- |
| **Output of trained network with for other NLS problem : learning rate of 0.3 and momentum of 0.5 after 1000 epochs** | **Mark / 1** |
| Inputs Targets Raw Ops Outputs  x y O1 O2 O1 O2 O1 O2  0.1 1.2 1 0 0.902 0.635 1 1  0.7 1.8 1 0 0.976 0.111 1 0  0.8 1.6 1 0 0.974 0.123 1 0  1.0 0.8 0 0 0.953 0.298 1 0  0.3 0.5 1 1 0.884 0.705 1 1  0.0 0.2 1 1 0.845 0.775 1 1  -0.3 0.8 1 1 0.863 0.767 1 1  -0.5 -1.5 0 1 0.131 0.799 0 1  -1.5 -1.3 0 1 0.128 0.802 0 1  Over Set : SSE 0.1121 0.0878 : %Correct 88 88 | |

|  |  |
| --- | --- |
| **Output of trained network for XOR (in each of the following cases : train the network and then display the results when you (P)resent the data to the Trained Network.** | |
| **Initial Random Seed 0; Lrate 0.5 mmtum 0; train 2000 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 0; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
|  | |
| **Initial Random Seed 1000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 5000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 2000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 1000; Lrate 0.3 mmtum 0.8; train 1000 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 0; Lrate 0.3 mmtum 0.8; train 1000 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |
| **Initial Random Seed 250; Lrate 0.3 mmtum 0; train 1000 epochs** | / 1 |
| <<Paste the result when you present data to trained network >> | |

|  |  |
| --- | --- |
| **Discussion (on code and results)** | **Missing .. Ok ... Excellent ;**  / 3 |
| << write discussion here >> | |
| **Conclusion** | **Missing .. Ok ... Excellent ;** / 3 |
| << write conclusion here >> | |

|  |  |  |
| --- | --- | --- |
| **Self Evaluation (answer yes/no/maybe or n/a)** | **Your View** | **Markers View** |
| My code works fully |  |  |
| My code is clear and concise |  |  |
| Each function has good comments explaining what it does and its arguments |  |  |
| The code implementing the functions are well explained |  |  |
| I understand the code in the library module |  |  |
| Feedback from last session was useful |  |  |

|  |
| --- |
| **Write below any comments, issues you have, further clarification which would be useful or any questions you would like answered** |
|  |

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