|  |  |  |  |
| --- | --- | --- | --- |
| ***CS2NN16 MLP Lab 2 Report Sheet 2018/19*** | | | |
| **Student Number: 26032996** | **Date: 21/11/18** | | |
| **Introduction** | | **Mark / 2** | |
| In this practical we will be looking at completing a multi layer network using up a hidden signoidal layer, along  with a linear output layer. This will require using the output layer to calculate the hidden layers errors, and use  other relations between them to calculate weights and inputs for the output layer. | | | |
| **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 |
| MLP - XOR Learn Rate 0.50 Momentum 0.00 Seed 0  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  Epoch 200 : SSE 0.2620 : %Correct 50  Epoch 400 : SSE 0.2479 : %Correct 75  Epoch 600 : SSE 0.2440 : %Correct 75  Epoch 800 : SSE 0.2634 : %Correct 50  Epoch 1000 : SSE 0.2668 : %Correct 50  Epoch 1200 : SSE 0.2517 : %Correct 50  Epoch 1400 : SSE 0.2430 : %Correct 50  Epoch 1600 : SSE 0.2398 : %Correct 50  Epoch 1800 : SSE 0.1589 : %Correct 75  Epoch 2000 : SSE 0.2838 : %Correct 75  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.349 0  0 1 1 0.142 0  1 0 1 0.655 1  1 1 0 0.360 0  Over Set : SSE 0.2769 : %Correct 75  Weights 1.61915 3.36151 -3.02206 -2.75987 4.87463 -5.27845 -2.14600 1.73904 1.18745 | |
| **Initial Random Seed 0; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| MLP - XOR Learn Rate 0.50 Momentum 0.80 Seed 0  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  Epoch 50 : SSE 0.2606 : %Correct 25  Epoch 100 : SSE 0.2398 : %Correct 75  Epoch 150 : SSE 0.2474 : %Correct 50  Epoch 200 : SSE 0.2690 : %Correct 50  Epoch 250 : SSE 0.2643 : %Correct 25  Epoch 300 : SSE 0.2548 : %Correct 50  Epoch 350 : SSE 0.2406 : %Correct 75  Epoch 400 : SSE 0.1998 : %Correct 75  Epoch 450 : SSE 0.1732 : %Correct 75  Epoch 500 : SSE 0.3366 : %Correct 50  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.417 0  0 1 1 0.204 0  1 0 1 0.605 1  1 1 0 0.589 1  Over Set : SSE 0.3275 : %Correct 50  Weights 0.29311 -0.05061 -2.28608 3.89053 -3.13152 -2.08994 0.57433 2.98829 -2.67378 | |
|  | |
| **Initial Random Seed 1000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| MLP - XOR Learn Rate 0.50 Momentum 0.80 Seed 1000  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.587 1  0 1 1 0.590 1  1 0 1 0.588 1  1 1 0 0.590 1  Over Set : SSE 0.2578 : %Correct 50  Epoch 50 : SSE 0.2620 : %Correct 0  Epoch 100 : SSE 0.2603 : %Correct 0  Epoch 150 : SSE 0.2595 : %Correct 0  Epoch 200 : SSE 0.2593 : %Correct 0  Epoch 250 : SSE 0.2591 : %Correct 0  Epoch 300 : SSE 0.2587 : %Correct 0  Epoch 350 : SSE 0.2581 : %Correct 25  Epoch 400 : SSE 0.2561 : %Correct 25  Epoch 450 : SSE 0.2430 : %Correct 75  Epoch 500 : SSE 0.2140 : %Correct 75  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.279 0  0 1 1 0.585 1  1 0 1 0.592 1  1 1 0 0.642 1  Over Set : SSE 0.2071 : %Correct 75  Weights -2.28983 -0.94516 -0.23645 -0.30590 -2.36824 -2.39925 0.63494 -0.91269 -3.54260 | | |
| **Initial Random Seed 5000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| MLP - XOR Learn Rate 0.80 Momentum 0.80 Seed 5000  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.435 0  0 1 1 0.455 0  1 0 1 0.404 0  1 1 0 0.428 0  Over Set : SSE 0.2563 : %Correct 50  Epoch 50 : SSE 0.2693 : %Correct 0  Epoch 100 : SSE 0.2682 : %Correct 0  Epoch 150 : SSE 0.2678 : %Correct 0  Epoch 200 : SSE 0.2554 : %Correct 75  Epoch 250 : SSE 0.1987 : %Correct 75  Epoch 300 : SSE 0.0941 : %Correct 100  Epoch 350 : SSE 0.1875 : %Correct 75  Epoch 400 : SSE 0.1847 : %Correct 75  Epoch 450 : SSE 0.1834 : %Correct 75  Epoch 500 : SSE 0.1827 : %Correct 75  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.339 0  0 1 1 0.952 1  1 0 1 0.330 0  1 1 0 0.334 0  Over Set : SSE 0.1694 : %Correct 75  Weights -4.86914 -6.28099 5.62308 6.73683 0.07199 4.81780 4.00852 5.45199 -4.72248 | |
| **Initial Random Seed 2000; Lrate 0.5 mmtum 0.8; train 500 epochs** | / 1 |
| MLP - XOR Learn Rate 0.50 Momentum 0.80 Seed 2000  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.415 0  0 1 1 0.365 0  1 0 1 0.393 0  1 1 0 0.351 0  Over Set : SSE 0.2668 : %Correct 50  Epoch 50 : SSE 0.2649 : %Correct 0  Epoch 100 : SSE 0.2619 : %Correct 0  Epoch 150 : SSE 0.2601 : %Correct 0  Epoch 200 : SSE 0.2595 : %Correct 0  Epoch 250 : SSE 0.2592 : %Correct 0  Epoch 300 : SSE 0.2591 : %Correct 0  Epoch 350 : SSE 0.2591 : %Correct 0  Epoch 400 : SSE 0.2590 : %Correct 0  Epoch 450 : SSE 0.2590 : %Correct 0  Epoch 500 : SSE 0.2589 : %Correct 0  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.507 1  0 1 1 0.508 1  1 0 1 0.509 1  1 1 0 0.509 1  Over Set : SSE 0.2499 : %Correct 50  Weights -2.59686 -0.54256 -0.32577 -1.71910 0.01385 -0.12019 0.03292 -0.28093 0.08388 | |
| **Initial Random Seed 1000; Lrate 0.3 mmtum 0.8; train 1000 epochs** | / 1 |
| MLP - XOR Learn Rate 0.30 Momentum 0.80 Seed 1000  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.587 1  0 1 1 0.590 1  1 0 1 0.588 1  1 1 0 0.590 1  Over Set : SSE 0.2578 : %Correct 50  Epoch 100 : SSE 0.2575 : %Correct 0  Epoch 200 : SSE 0.2566 : %Correct 0  Epoch 300 : SSE 0.2559 : %Correct 0  Epoch 400 : SSE 0.2556 : %Correct 0  Epoch 500 : SSE 0.2557 : %Correct 0  Epoch 600 : SSE 0.2555 : %Correct 0  Epoch 700 : SSE 0.2551 : %Correct 0  Epoch 800 : SSE 0.2546 : %Correct 25  Epoch 900 : SSE 0.2535 : %Correct 50  Epoch 1000 : SSE 0.2329 : %Correct 75  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.384 0  0 1 1 0.532 1  1 0 1 0.541 1  1 1 0 0.577 1  Over Set : SSE 0.2276 : %Correct 75  Weights -1.97289 -0.85337 -0.28311 -0.53628 -1.90580 -1.94622 0.38509 -1.14184 -1.95173 | |
| **Initial Random Seed 0; Lrate 0.3 mmtum 0.8; train 1000 epochs** | / 1 |
| MLP - XOR Learn Rate 0.30 Momentum 0.80 Seed 0  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  Epoch 100 : SSE 0.2542 : %Correct 25  Epoch 200 : SSE 0.2362 : %Correct 75  Epoch 300 : SSE 0.2624 : %Correct 50  Epoch 400 : SSE 0.2617 : %Correct 25  Epoch 500 : SSE 0.2444 : %Correct 50  Epoch 600 : SSE 0.2289 : %Correct 75  Epoch 700 : SSE 0.1596 : %Correct 75  Epoch 800 : SSE 0.2568 : %Correct 75  Epoch 900 : SSE 0.0403 : %Correct 100  Epoch 1000 : SSE 0.4965 : %Correct 50  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.983 1  0 1 1 0.882 1  1 0 1 0.871 1  1 1 0 0.995 1  Over Set : SSE 0.4965 : %Correct 50  Weights -0.45682 4.13241 -5.09230 2.38500 7.50589 -4.60919 1.58781 -4.28804 4.50450 | |
| **Initial Random Seed 250; Lrate 0.3 mmtum 0; train 1000 epochs** | / 1 |
| MLP - XOR Learn Rate 0.30 Momentum 0.00 Seed 250  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.282 0  0 1 1 0.338 0  1 0 1 0.297 0  1 1 0 0.348 0  Over Set : SSE 0.2832 : %Correct 50  Epoch 100 : SSE 0.2579 : %Correct 50  Epoch 200 : SSE 0.2583 : %Correct 50  Epoch 300 : SSE 0.2585 : %Correct 50  Epoch 400 : SSE 0.2586 : %Correct 25  Epoch 500 : SSE 0.2586 : %Correct 25  Epoch 600 : SSE 0.2587 : %Correct 25  Epoch 700 : SSE 0.2586 : %Correct 50  Epoch 800 : SSE 0.2584 : %Correct 50  Epoch 900 : SSE 0.2578 : %Correct 25  Epoch 1000 : SSE 0.2562 : %Correct 50  Inputs Targets Raw Ops Outputs  x1 x2 XOR XOR XOR  0 0 0 0.545 1  0 1 1 0.509 1  1 0 1 0.507 1  1 1 0 0.462 0  Over Set : SSE 0.2485 : %Correct 75  Weights -0.18081 -1.23466 -0.57582 2.12628 -1.07827 -1.34413 -0.40004 0.27839 0.50673 | |

|  |  |
| --- | --- |
| **Discussion (on code and results)** | **Missing .. Ok ... Excellent ;**  / 3 |
| The code functions mostly as planned. All calculations are correct except for the weightedDeltas function.  This error was not performed completely but luckily this does not ruin the affect of too many other  functions. However, this does affect the results heavily as most of them have come up incorrect. | |
| **Conclusion** | **Missing .. Ok ... Excellent ;** / 3 |
| Overall, the relationship between the sigmoidal and linear layer in a multi layer network has become  more clear and my understand of them have been deepened. The only misunderstanding from this is the  calculation for the weighted errors for the hidden layer. When using this for a single neuron it worked  correctly however did not for the use of multiple neurosn | |

|  |  |  |
| --- | --- | --- |
| **Self Evaluation (answer yes/no/maybe or n/a)** | **Your View** | **Markers View** |
| My code works fully | no |  |
| 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 (mostly) |  |
| Feedback from last session was useful | yes |  |

|  |
| --- |
| **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** |
|  | |