**CSSE 413: Artificial Intelligence**

Neural Networks Lab Manual

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Basics**

[8 pts] **trainNetwork()**implemented.

1. [1 pt] Given a *learning rate* of 0.1 and a *threshold* of 0.5, how many training episodes does the network take to learn Boolean **and**? \_\_3\_\_
2. [1 pt] Keep the learning rate at 0.1 and lower the threshold until you can learn in one episode. Keep the threshold values to whole 1/10ths. What *threshold* value enabled the network to learn in one episode? \_\_0.2\_\_
3. [1 pt] Reset the *threshold* to 0.5. Now, modify the *learning rate*, again in 1/10th increments until the network can learn and in one episode. What is the value of the *learning rate* that enables the network to learn in one episode? \_\_0.3\_\_
4. [2 pts] What is the relationship between the *learning rate* and the *threshold* values in the context of the prior two experiments? Please have a look at the weights of the network.

In both situations, the weights add up to more than the threshold.

1. [2 pt] Modify the training data so as to attempt to learn the **xor** Boolean function. Ensure you reset the learning rate to 0.5 and the threshold to 0.1. Then experiment with different values for the *learning rate* and the *threshold* value. In light of your answer to question (4), what appears to be the problem with the perceptron network when attempting to learn **xor**? Again, please have a look at the weights of the network.

In order to learn XOR, the input to the last neuron must be high when only one input neuron is active. However, because the calculation adds the activations of the previous layer together, that cannot happen as it will be high when both inputs are active.

**XOR Experiments**

1. [8 pts] Run XOR.java. Currently, the number of training episodes is to 1,000, the learning rate is set to 0.1 and there is a single hidden layer which has three nodes. Based on the example XOR net we saw in the slides, we should be able to develop a network that implements XOR reliably As it turns out, this is not the case with the current set-up of the network. The testNetwork() method prints the actual output and the desired output. The printWeights() method shows the weights after learning. Experiment with the learning rate and the number of training episodes to determine values for which the network trains reliably. Add to your report three sets of values with which you were able to train the network efficiently and reliably. Briefly state what you consider to be reliable.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Set | Learning Rate | Training Episodes | Output for example 1 | Output for example 2 |
| 1 | 0.1 | 12000 |  |  |
| 2 | 0.2 | 5000 |  |  |
| 3 | 0.3 | 2500 |  |  |

What relationship exists between among the parameters? Answer below.

As the threshold increases, the number of episodes required decreases.

1. [12 pts] Modify the XOR.java file so that the network uses a step activation function rather than a sigmoid activation function. Hint: Recall that the derivative of the step function is not defined for all values, hence, you need to modify trainNetwork(). You should be able to learn XOR in fewer than 20 training episodes. Experiment with the number of training episodes, learning rate, threshold of the activation function and hidden layer size. Below, add three sets of values with which you were able to train the network efficiently and reliably.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training episodes | Learning rate | Threshold | Hidden layer size |
| 1 | 15 | 0.1 | 0.5 | 3 |
| 2 | 15 | 0.1 | 0.5 | 2 |
| 3 | 10 | 0.2 | 0.7 | 2 |

Briefly state what you consider to be reliable.

Having a large learning rate and small episode count often trains quickly but is unreliable. The best arrangement is small learning rate, higher episode count, higher threshold, and at least two hidden nodes.

**Parity Bit**

[25 points] ParityNet.java successfully implements 7-bit parity bit.

**Digit Recognition**

1. [35 points] Experiment with the various parameters so as to efficiently and reliably train the network. Document the number of training episodes required to obtain the sort of precision you deem sufficient. Please justify your decision for the precision. Additionally, include the values of the following parameters: the range of the initial weights, the learning rate and the number and size of the hidden layers.

Number of training episodes: \_\_\_10\_\_

Learning rate: \_\_\_0.1\_\_

Weight offset: \_\_\_0.0\_\_

Number of hidden layers: \_\_\_1\_\_

Size of hidden layers: \_\_\_8\_\_

Global error after training: \_\_ 0.01608425900002581\_\_\_

Overall accuracy on test set: \_\_\_0.9045\_\_

Precision and justification of reasonable precision. Please elaborate below.

I experimented with many different configurations and found that this configuration gives the highest results most consistently. The lowest accuracy I managed to get was 0.6403. As for the size of the hidden layer, I used 8 as a rough number of the different “parts” of a number. For example, both 8 and 9 have a circle on the upper half, so I assume that there is a neuron in the hidden layer that is active when there is a circle on the upper half .