**Introduction to Artificial Intelligence**

**Project 2: Automated Part Sorting**

**Submitted by: Raj D. Devrukhkar**

**1. Network Design:**

(a) Network Diagram:

HN1

Y1

WHO

WIH

Y2

X1

Y3

X2

HNN

Y4

**Hidden Layer**

**Output Layer**

**Input Layer**

where, WIH = Weights from input nodes to hidden nodes

WHO = Weights from hidden nodes to output nodes

HNi = ith hidden node

N = Total number of hidden nodes.

**Fig. General 2-N-4 Neural Network Architecture**

**(b)** Comment on the sets of functions that the different architectures are capable of representing.

* The functions that an n-hidden node architecture is capable of representing would be the functions that divide the input space into their individual class regions with respect to the sigmoidal function. For example in our example (nuts and bolt), sufficient number of hidden nodes required is 4 since there are four regions and each region has approximately only inputs belonging to the same class. So we will need 4 sigmoidal curves arranged such that they divide the input space into their individual classes. So for an n-output classifies we should use at least a n-hidden node architecture.

**(c)** Which architecture did you expect to be most successful, and why?

* I expected the third architecture (2-10-4) to be the best because it had more hidden nodes and hence, more weights were updated for each example and expected the network to converge faster than the other networks. But I didn’t really know which architecture would give me the highest profits because when updating the weights we do not consider the cost matrix and hence the network tries to average the overall classes and is not biased towards any particular class to maximize profits.

**2. Results:**

**(a)** A plot of the learning curves for each network

**Fig: Learning curves for the 3 classifiers**

**(b)** Confusion Matrix and the Profit

Confusion Matrix for Architecture 2-2-4:

20 0 0 0

0 28 0 0

0 0 27 0

8 1 1 9

Profit: 8.98

Confusion Matrix for Architecture 2-5-4:

18 0 0 2

0 28 0 0

0 0 27 0

0 1 1 17

Profit: 8.44

Confusion Matrix for Architecture 2-10-4:

18 0 0 2

0 28 0 0

0 0 27 0

0 1 1 17

Profit: 8.44

**(c)** A plot of the training data set in feature space

**(d)** A plot of the class regions produced by each classifier.

**Fig: Class regions for the 2-2-4 Classifier**

**Fig: Class regions for the 2-5-4 Classifier**

**Fig: Class regions for the 2-10-4 Classifier**

**3.** **Discussion:** .

* The best performance can be considered in two ways

i) With respect to the profit earned: As seen from the above outputs, the classifier with 2 hidden nodes gives the highest amount of profit. I did not expect the first classifier (2-2-4) to give the best profit because in general I expected it not to be a good enough classifier (which is the case as seen from the epoch vs. SSE graph) because of the number of hidden nodes. Maybe the sample set is not properly chosen for this problem.

ii) With respect to learning and error in classification: The networks with 5 and 10 hidden nodes were better at classifying each sample to their respective classes. The confusion matrices of both the architectures are the same however, the class regions are different. The 3rd network seems to have a better classification since it has the least SSE of the three and the class regions seem to be proper and somewhat in sync with the input samples.

* In the class regions as the number of hidden nodes increase, the region of the class that covers a larger space in the feature space seems to be increasing and the ones concentrated in an area seems to be shrinking. I think it happens because after each epoch the classifier tries to get a better class boundary depending on the input values. So if we continue the updating of weights for a very large number of epoch then the classifier might over fit to the specific sample input. I think the major factor that contributes to this behaviour is the input sample set because the network will try to converge on the available input feature space. The samples may or may not do justice to an all the possible distinct input sets.
* While programming the train.py, I used a constant 0.5 for all initial weights. I assumed that the network will train it and converge the values of the weights at some point of time. But when I executed the code, I got different values as compared to the execution of the networks with random initial weights. The network did not nuts and scrap properly. The following were the confusion matrices

Using Architecture: 2-2-4

Confusion Matrix:

20 0 0 0

0 15 11 2

0 0 27 0

9 4 1 5

Profit: 6.12

Using Architecture: 2-5-4

Confusion Matrix:

19 0 0 1

0 15 11 2

0 0 27 0

8 3 1 7

Profit: 5.85

Using Architecture: 2-10-4

Confusion Matrix:

18 0 0 2

0 10 11 7

0 0 27 0

6 1 1 11

Profit: 4.48