**Table 1 - Truth table for the [3,4,1] neural network architecture.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X1 | X2 | A1 | A2 | A3 | y |
| 0 | 1 | ~0 | ~1 | ~0 | 0 |
| 0 | 0 | ~1 | ~1 | ~1 | 1 |
| 1 | 0 | ~0 | ~1 | ~0 | 0 |
| 1 | 1 | ~0 | ~0 | ~0 | 1 |
|  |  | (NOTX1)AND(NOTX2) | (NOTX1)OR(NOTX2) | (NOTX1)AND(NOTX2) |  |

**Are the extracted features the same as that given on Slide 25? Discuss.**

No. After training the neural net on multiple occasions the activations do not always coincide with a simple Boolean expression. Sometimes the activations hold values of 0.7, sometimes 0.2. After running it multiple times, the truth able above seemed to come up multiple times after training. Since the activations have higher dimensionality to work with through the use of an extra hidden neuron, they can adopt higher order features that are not as simple as values approaching boolean 1 or 0.

**Can you suggest a way to run the algorithm so that it will produce the same features as those given on Slide 25?**

In slide 25 the neural network learns higher order features of (X1)AND(X2) and (NOTX1)AND(NOTX2). Assuming you cannot micromanage specific neurons, one way to produce the same features would be to penalize weights that deviate from one another through regularization. Since the activations work when they are both largely positive (a1) and largely negative (a2), you could penalize the difference between the weights squared. This is due to the nature of the AND statements of the higher order features, which work well when both weights are large and both positive/negative. By squaring the regularization, you would cause them to change rapidly and prevent them from approaching zero during training. This is due to the nature of their derivatives (2θ versus Constant). Since bias values vary throughout, it wouldn’t make sense to regularize those.

A regularization method was added which attempts to drive the neural net to learn higher order features as those obtained in slide 25, and it was largely unsuccessful. It did not converge, therefore another hyperparameter which modifies the rate of regularization may be used. No other alternative regularization methods are suspected of working.

**Run your Matlab code using the logistic cost function and compare the results with that obtained using the Euclidean cost function as done in (1) above.**

After looking at the differences in cost function curves, a number of differences exist. Firstly, the logistic cost function requires smaller learning rate to converge. When tested with higher rates, convergence was not found even when the epochs increased. The logistic cost function also produces much higher error rates, but that is simply the nature of the cost function itself (logarithmic) versus squared Euclidean distance (difference squared). Both cost functions converged around the same number of epochs.