CS6910: Fundamentals of Deep Learning
Assignment-1: Neural Network Implementation

Team 1: Arnav Anil Mhaske (CS17110), Samuel Jeyseelan (CS17B026) and Vamsi Krishna Valluru (CS17B045)

Contents:

1) Design and Implementation Details:

a) We modularized our code implementation for the assignment:

i) Neural network module: We designed an API that is similar to that of Keras. It has the following methods apart from the state variables:

(1) `__init__()` constructor that initializes the weights, biases, activation values etc.

(2) `loadwb()and compile()` function that loads weights from existing stored weights.

(3) `fwprop_fn()` function that does the forward cycle.

(4) `computeDelta()` function that computes the delta vector for each layer.

(5) `deltaOptimizer(), genDeltaOptimizer() and adamOptimzer()` form the back propogation functions.

(6) `crossEntropy() and sumOfSquares()` form the loss functions.

(7) `Tanh(), Softmax() and Linear()` form the activation functions.

(8) `fit() and fit_r()` form the training functions for classification and regression respectively.

(9) `test() and test_r()` form the testing functions using existing waits.

ii) `Driver_2D.py, Driver_funapp.py and Driver_Img.py` form the driver files for each of the three datasets.

iii) `Test_2D.py, Test_funapp.py and Test_image.py` form the testing files for each of the three datasets.

b) The theoretical basis for the forward and back propagation functions(All the optimisers use a common "delta") was formed using the following formulas derived by us:

i) Forward cycle:

(1) $\mathbf{a}^{[l]} = \mathbf{W}^{[l]\mathbf{T}}\, \mathbf{s} + \mathbf{b}^{[l]}$

(2) $\mathbf{s}^{[l]} = f(\, \mathbf{a}^{[l]})$

ii) Backward cycle:

(1) $\boldsymbol{\delta}^{[l]} = \mathbf{t} - \mathbf{s}^{[l]}$

(2) $\boldsymbol{\delta}^{[l]} = \text{element-wise-product}(\mathbf{da}^{[l]}, \mathbf{W}^{[l+1]}\boldsymbol{\delta}^{[l+1]})$ $\quad l > L$

(3) $\Delta\mathbf{W}^{[l]} = \eta\mathbf{s}^{[l-1]}\boldsymbol{\delta}^{[l]\mathbf{T}}$

Legend:

1) $\mathbf{a}^{[l]}, \mathbf{da}^{[l]}, \mathbf{s}^{[l]}$ – Activation value, its derivative, Output of the l*th* layer
2) $\mathbf{W}^{[l]}$ – Weights for the l*th* layer
3) $\boldsymbol{\delta}^{[l]}$ – $(-dE/d\mathbf{a}^{[l]})$
4) η – Learning rate
5) $\mathbf{t}$ – Expected output (for classification a 1-hot vector)

2) Function Approximation task:

a) In this task, we used the following layer architecture:
   i) [(2, nothing), (50, tanh), (50, tanh), (1, linear)]

b) The non-linearity is introduced by the tanh-sigmoid function. The output layer though, has a linear activation function so as to reflect the actual values of the simulated function.

c) We used the Sum of Squared loss function and the Generalized Delta optimizer.

d) We noticed that the performance peaked when the hidden layers were set to 50 nodes each.

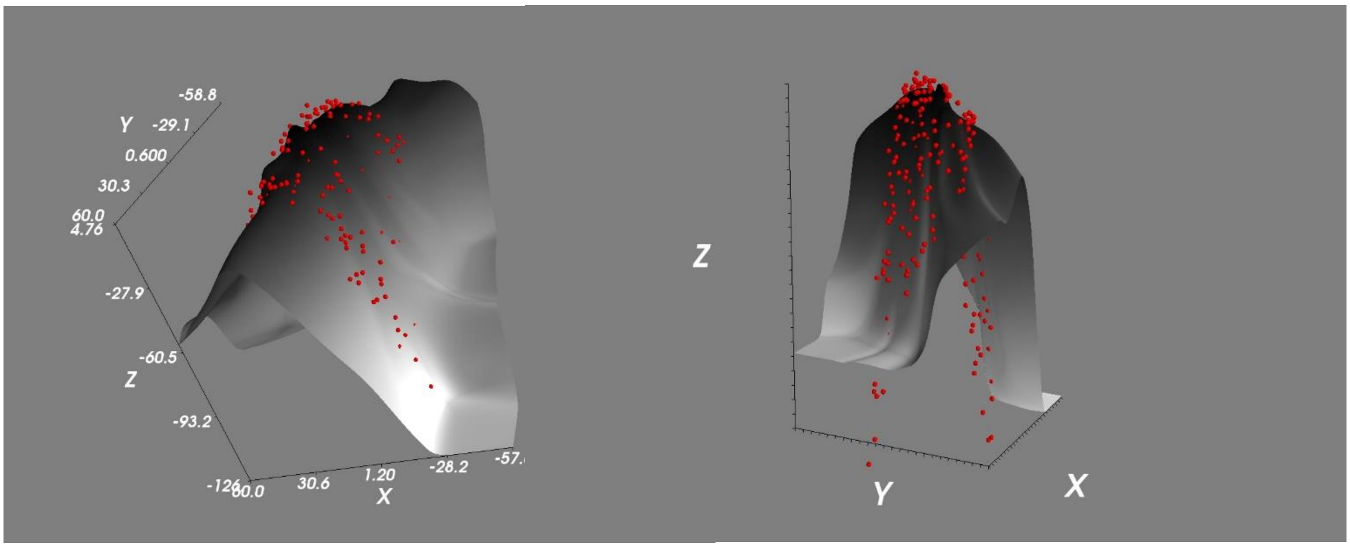


Figure 2.1: The above figures represent the Expected function points vs the Predicted surface 3-D plot. The red points are the expected function outputs and the grey 3-D surface is the neural network's predicted function. We presented two different viewpoints of the same surface and also note that the left image has been scaled slightly. Since the outlier points are very sparse, the predicted surface does not match the expected value towards the tail end of the function. It appeared to match it as the network complexity increased but due to limited training samples, we restricted the number of parameters.

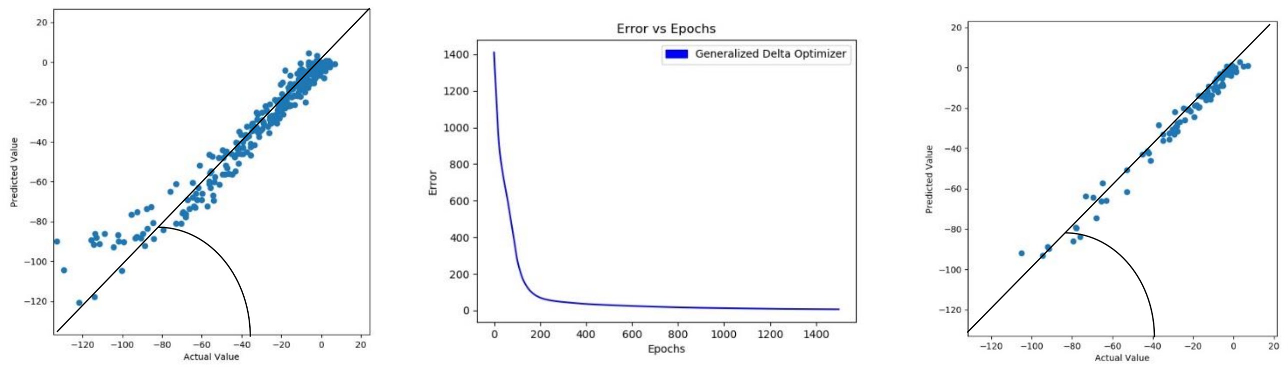


Figure 2.2: The figure to the left and the right represent the Model output vs the Desired output graph for the validation and training sets respectively. The central figure represents the Error vs Epochs curve. As expected the Model output vs the Desired output graphs give us a 45^o line (y = x). Notice that both validation set has a few points off of the line towards the bottom left. This can be empirically attributed to the fact that most of the training examples were concentrated towards the top right of the graph. The central graph depicts the speedy convergence followed by the minute advances towards the 0 error.

3) **Classification of 2-D data task:**
   a) In this task, we used the following layer architecture:
      i) [(2, nothing), (6, tanh), (6, tanh), (3, softmax)]
   b) The non-linearity is introduced by the tanh-sigmoid function. The output layer though, has a softmax activation function so as to keep the output values < 1.0 and also have the sum of all the output values = 1.0.
   c) We used the Cross Entropy loss function and the Generalized Delta optimizer, although we did try and run it using the Adam optimizer as well.

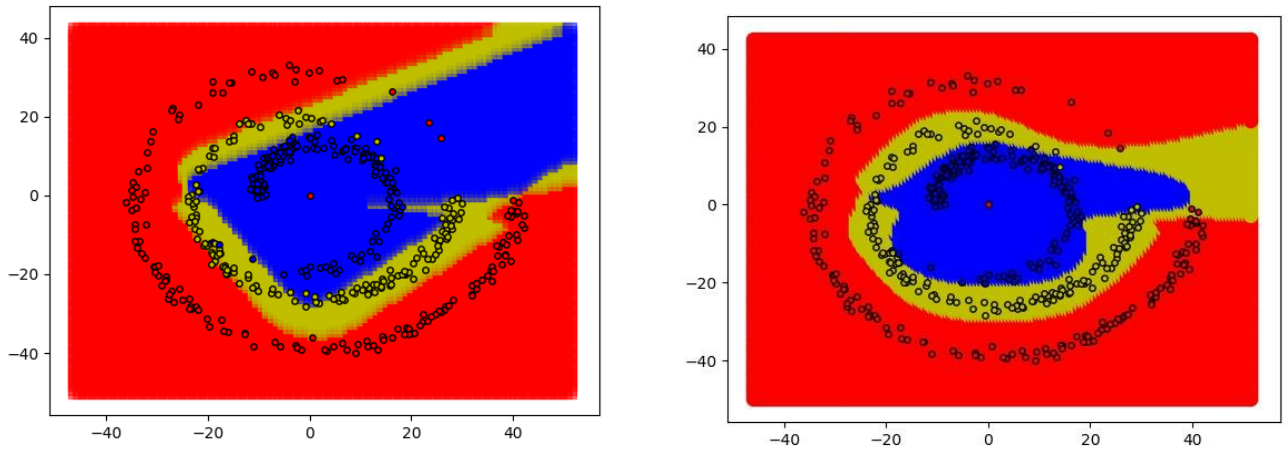


Figure 3.1: The figure on the left and right denote the decision boundaries produced by the Generalized delta and Adam optimizers. Notice the difference in the decision boundaries both learnt. Clearly the minima was found faster by the Adam optimizer due to its superior weight update procedure. The similarity between the two is that the top right side extends to infinity. This can be attributed to the sparsity of training points in that region.

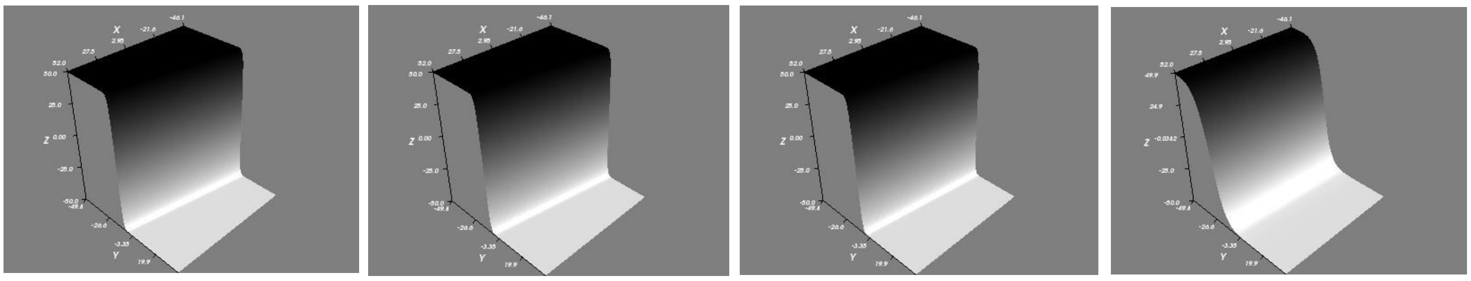


Figure 3.2: Figures for layer 1, node 0 for epochs 1, 2, 100 and 400 from left to right.

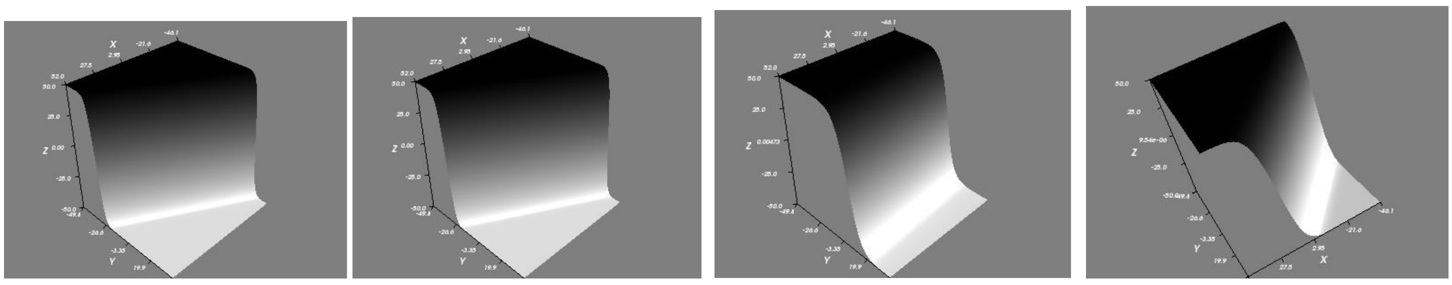


Figure 3.3: Figures for layer 1, node 4 for epochs 1, 2, 100 and 400 from left to right.

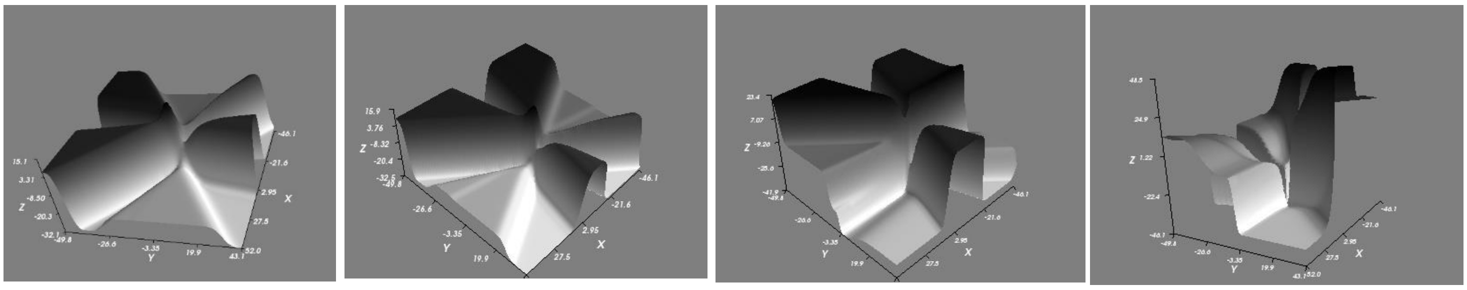


Figure 3.4: Figures for layer 2, node 0 for epochs 1, 2, 100 and 400 from left to right.

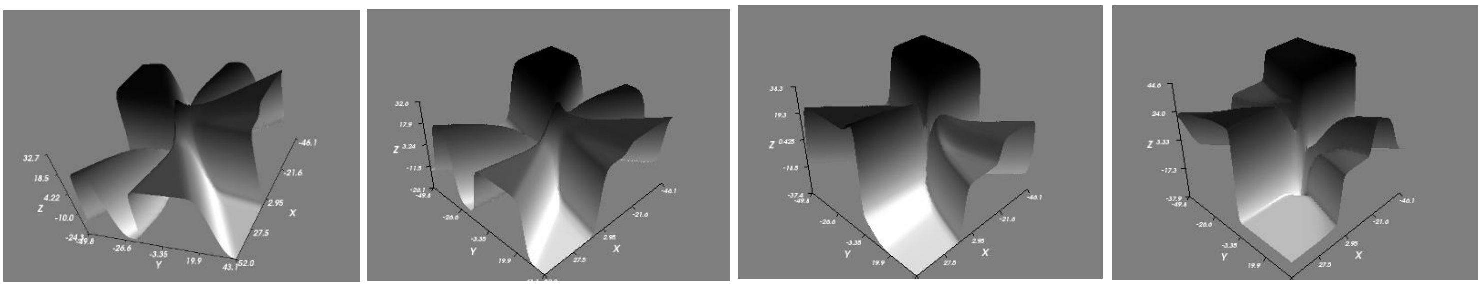


Figure 3.5: Figures for layer 2, node 4 for epochs 1, 2, 100 and 400 from left to right.

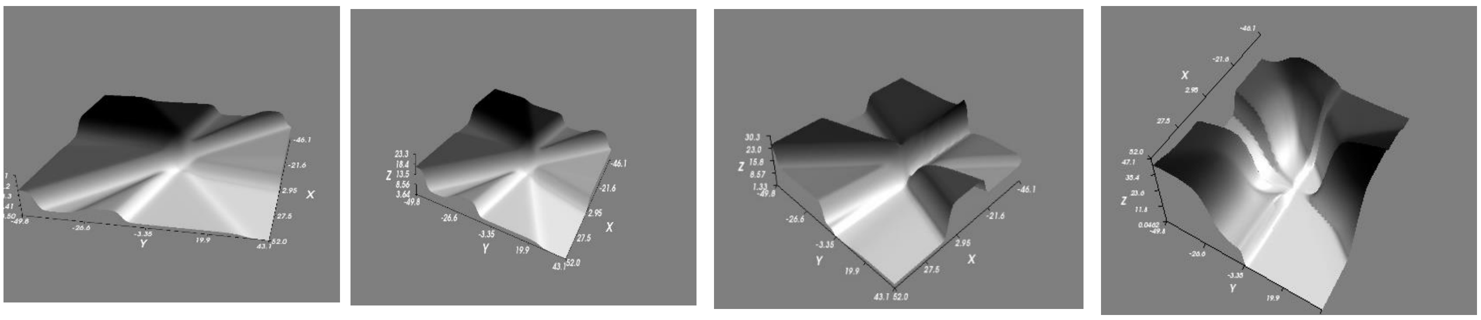


Figure 3.6: Figures for layer 3, node 0 for epochs 1, 2, 100 and 400 from left to right.

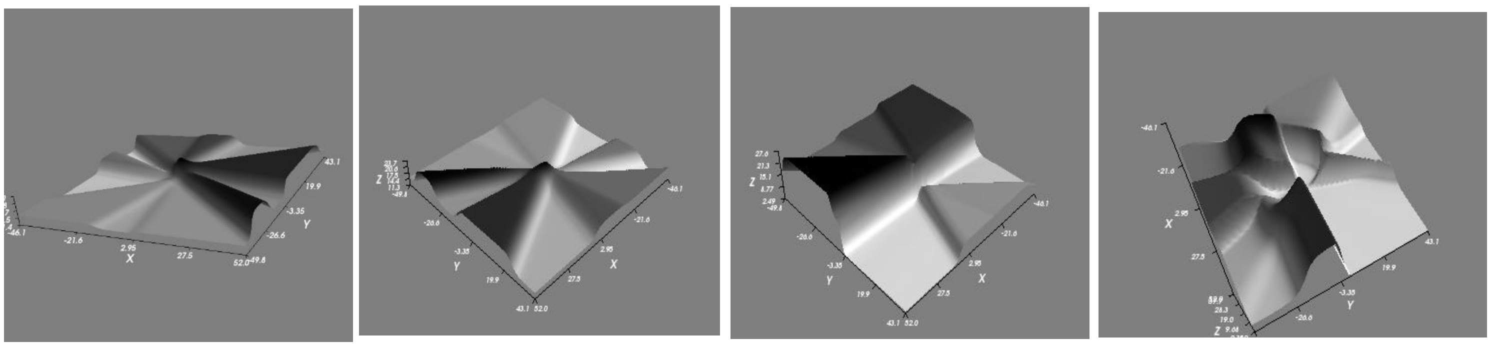


Figure 3.7: Figures for layer 3, node 1 for epochs 1, 2, 100 and 400 from left to right.

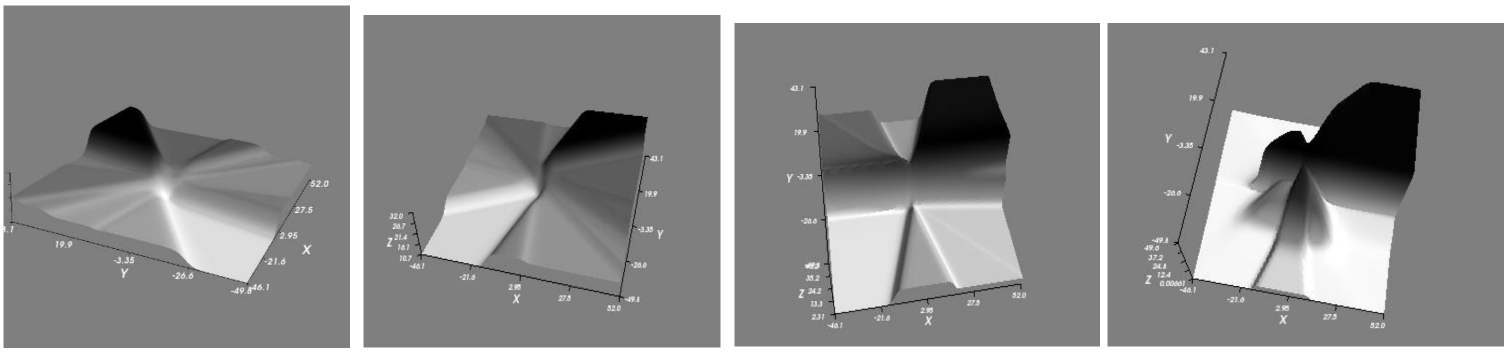
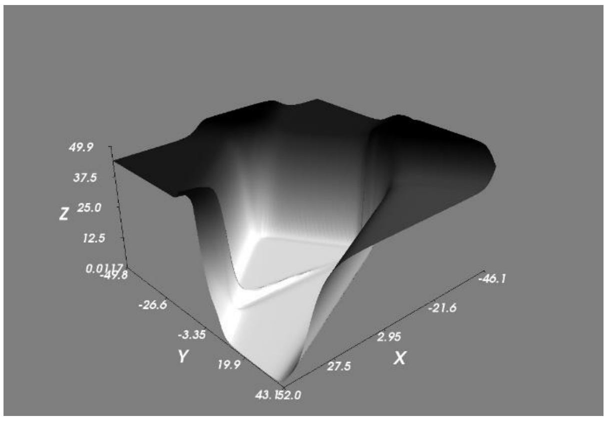
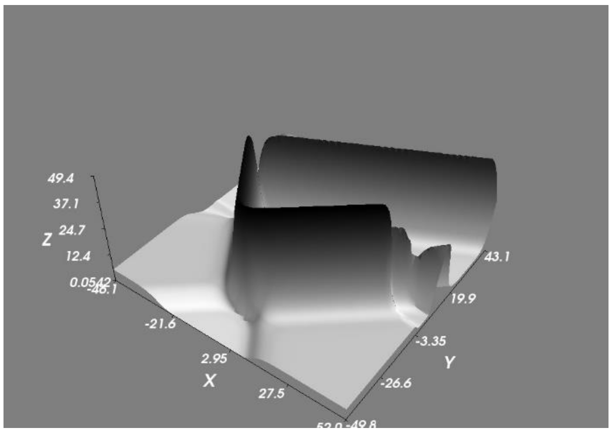


Figure 3.8: Figures for layer 3, node 2 for epochs 1, 2, 100 and 400 from left to right.





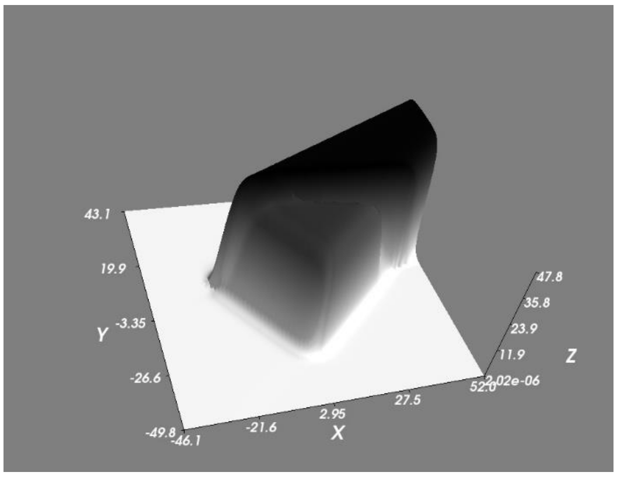


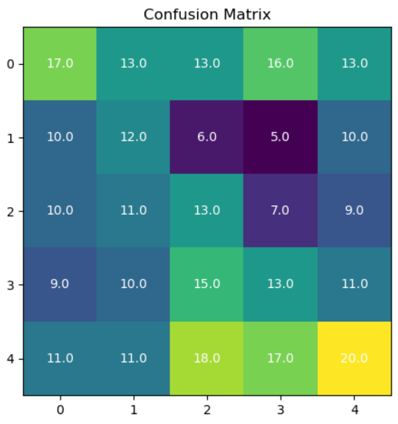
Figure 3.9: Go from left to right, top to bottom. The first one represents class 0 final output, second, class 1 final output and third, class 2 final output at the end of training. It clearly shows the different regions where each class probability dominates. As can be observed, class zero peaks at the edge, class 1 peaks at the walls (in the middle) and class 2 peaks at the centre.

4) Classification of Image data task:

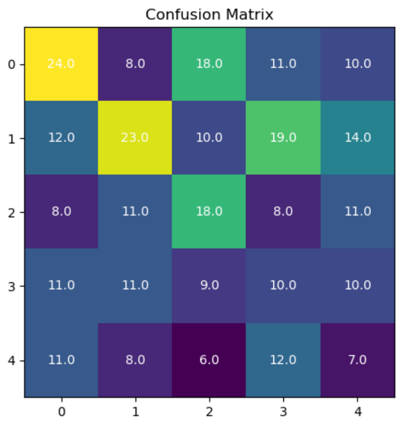
a) In this task, we used the following layer architecture:
   i) [(512, nothing), (5, tanh), (5, tanh), (5, softmax)]

b) The non-linearity is introduced by the tanh-sigmoid function. The output layer though, has a softmax activation function so as to keep the output values < 1.0 and also have the sum of all the output values = 1.0.

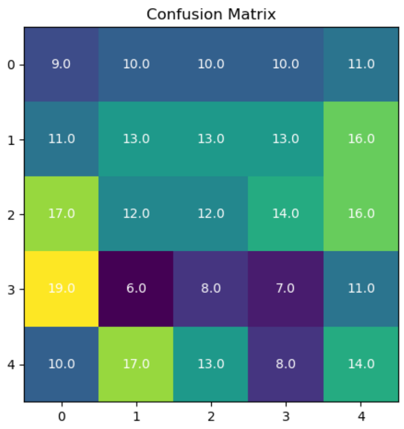
c) We used the Cross Entropy loss function and the Delta, Generalized Delta and Adam optimizers.


Confusion Matrix

	0	1	2	3	4
0	17.0	13.0	13.0	16.0	13.0
1	10.0	12.0	6.0	5.0	10.0
2	10.0	11.0	13.0	7.0	9.0
3	9.0	10.0	15.0	13.0	11.0
4	11.0	11.0	18.0	17.0	20.0


Confusion Matrix

	0	1	2	3	4
0	24.0	8.0	18.0	11.0	10.0
1	12.0	23.0	10.0	19.0	14.0
2	8.0	11.0	18.0	8.0	11.0
3	11.0	11.0	9.0	10.0	10.0
4	11.0	8.0	6.0	12.0	7.0


Confusion Matrix

	0	1	2	3	4
0	9.0	10.0	10.0	10.0	11.0
1	11.0	13.0	13.0	13.0	16.0
2	17.0	12.0	12.0	14.0	16.0
3	19.0	6.0	8.0	7.0	11.0
4	10.0	17.0	13.0	8.0	14.0

Figure 4.1: The above figures are the confusion matrices for the image data classification task where the (left to right) first image is that of delta, generalized delta and the Adam optimizers on the validation set. The accuracies are very low for all cases. This probably indicates that our model was overfitting.

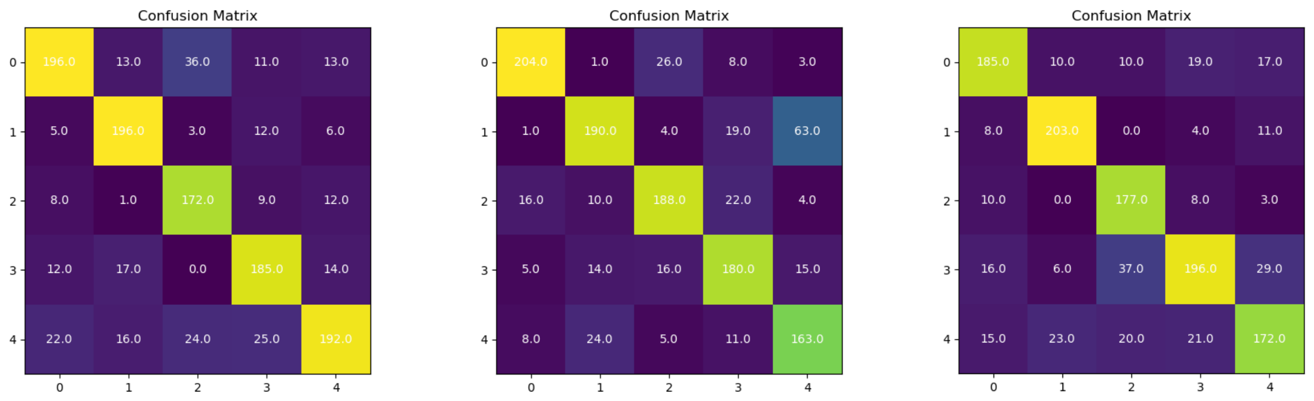


Figure 4.2: The above figures are the confusion matrices for the image data classification task where the (left to right) first image is that of delta, generalized delta and the Adam optimizers on the train set. The accuracies are very high for all cases. This reinforces our conclusion that our model was overfitting.

d) We tried many possible fixes to avoid overfitting and improve the validation set classification accuracy:
   i) We did PCA (Principle component analysis) to different dimensions (25, 10 and 5).
   ii) We tried reducing the number of nodes per hidden layer.
   iii) We tried reducing the number of hidden layers themselves.
   iv) We tried classic L-2 Regularization to no effect (the weights were not very large to begin with)
   v) We tried the dropout technique, also to no effect. (This was our most complicated attempt)

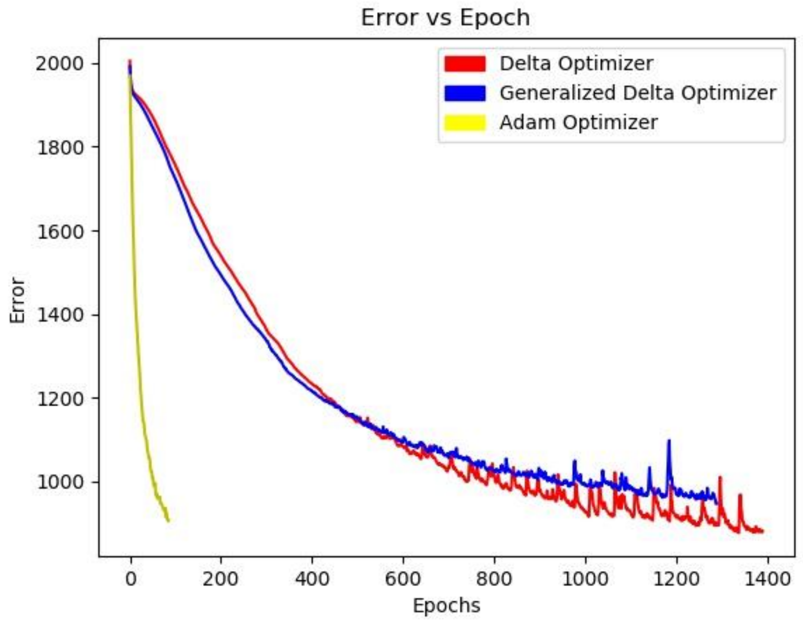


Figure 4.3: The above figure is the Error vs Epoch graph that plots the curve for all the three optimizers – Delta, Generalized Delta and Adam. As you can see Adam (Yellow) converges very quickly, whereas Delta and Generalized Delta take a while, with oscillations towards the end. This means that they oscillate around a minima. Also we ran the three optimizers till we reached a 75% accuracy.

Acknowledgements:

1) Credits to Dr.Chandrasekhar and his lecture notes.
2) Credits to general discussions with Suhas P, Sriram V, Dev S and Harshit.
3) None of the code has been plagiarised and all the matrix derivations were done on our own. The exception would be the use of snippets from Stackoverflow to help with plot formatting.