CS6910 Assignment 1

Lakshya J(EE19B035), Neham Jain(EE19B084), Nisharg Manvar(EE19B094)

March 2022

1 Function Approximation Task

In this task, we are asked to create a Multi Layer Feed Forward Neural Network of two layers where the Learning Rate and the number of neurons in each layer are hyper-parameters.

We are supposed to use Stochastic Gradient Descent as the optimizer and Mean Squared Error Loss for the loss function.
tanh was used as the activation function after every layer except for the last output layer.

I ran the model for 2000 epochs on batch sizes of 16, 32, 64, 128 for varying learning rates and number of neurons.

Learning rates used: [1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
Number of neurons: [50, 100, 256, 512, 784, 1024, 2048]

Out of all these cases, I observed the least loss for the validation set in the case of batch size of 16, learning rate = 0.01 and number of neurons = 512.

Here is the result obtained:
$Learning\ Rate:\ 0.01$
$No.\ of\ Neurons:\ 50$
$Validation\ loss:\ 0.0003136$

I tested this model on the test set to obtain final results.

1.1 Plots Obtained

First, I plot the 3D plot of the actual function.

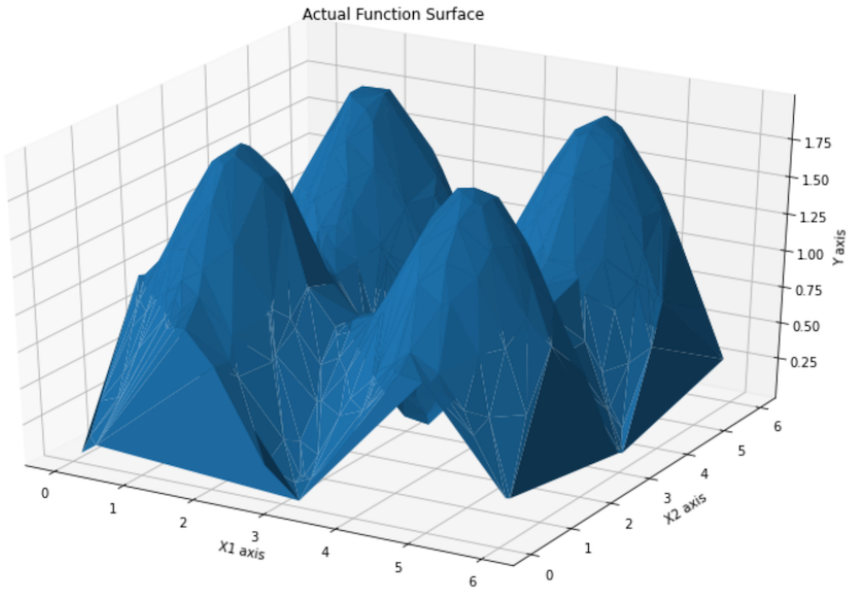


Figure 1: Actual Plot

Next I plot the output obtained after training the model for various epochs. I plot the output of the model for epochs 1, 2, 10, 50, 500, 1000, 1500 and 2000(when the training is stopped). It is observed that as the model moves through epochs, it is able to approximate the output better and give a smoother 3D output.

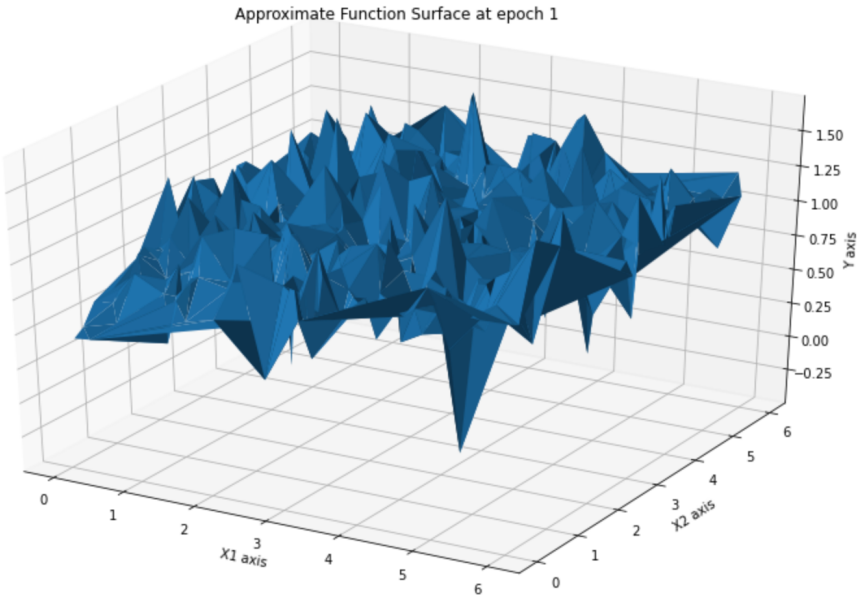


Figure 2: Plot of all data applied on the model after 1 epoch

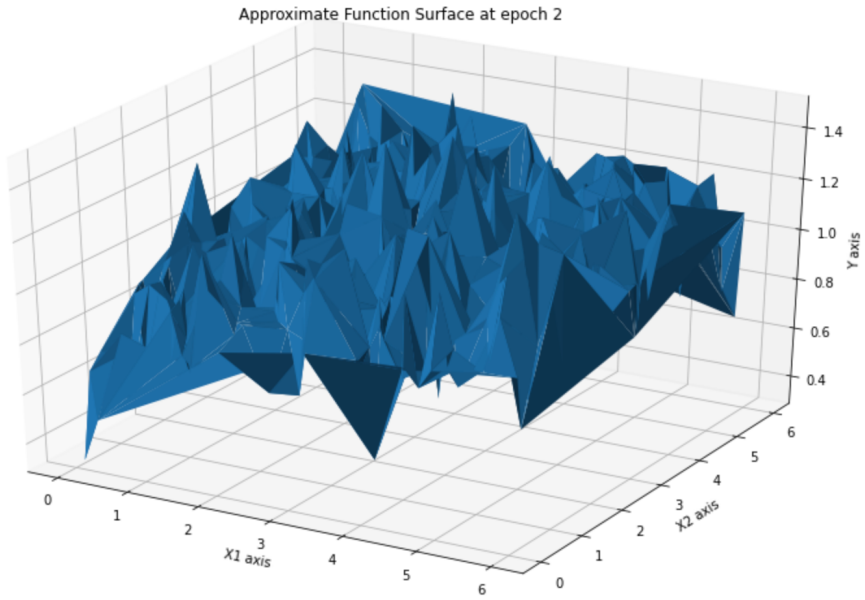


Figure 3: Plot of all data applied on the model after 2 epoch

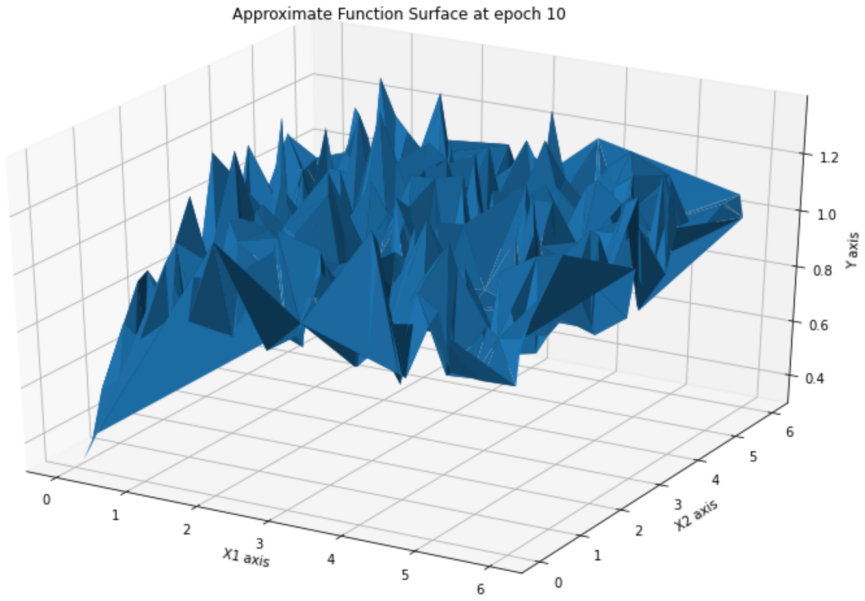


Figure 4: Plot of all data applied on the model after 10 epoch

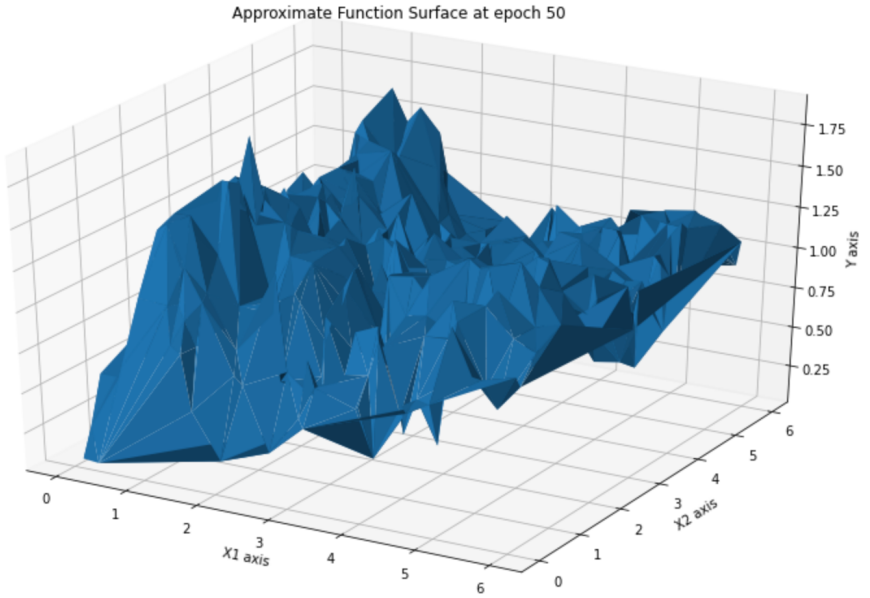


Figure 5: Plot of all data applied on the model after 50 epoch

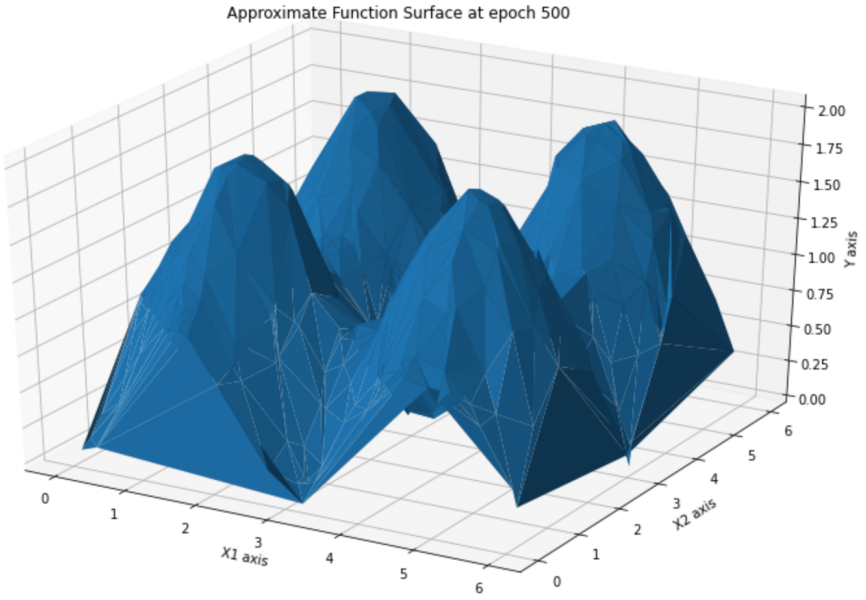


Figure 6: Plot of all data applied on the model after 500 epoch

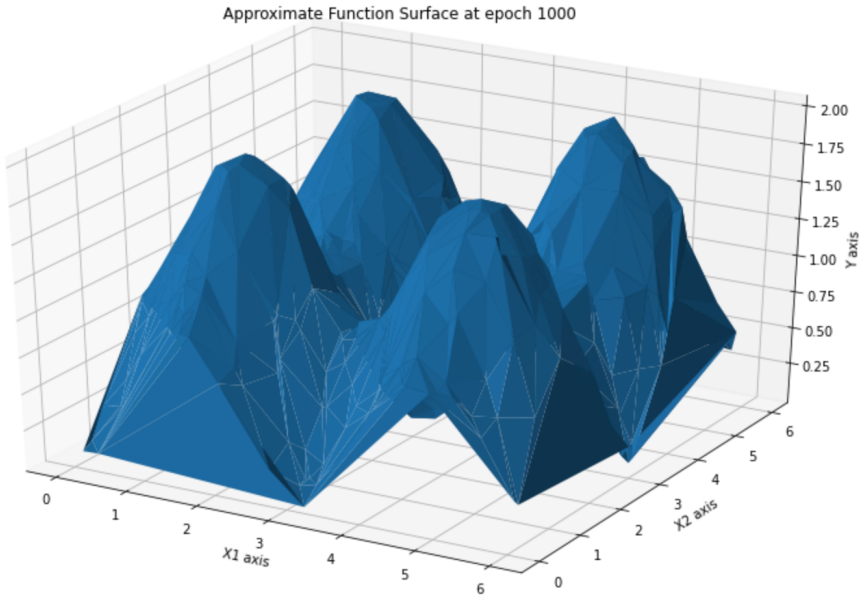


Figure 7: Plot of all data applied on the model after 1000 epoch

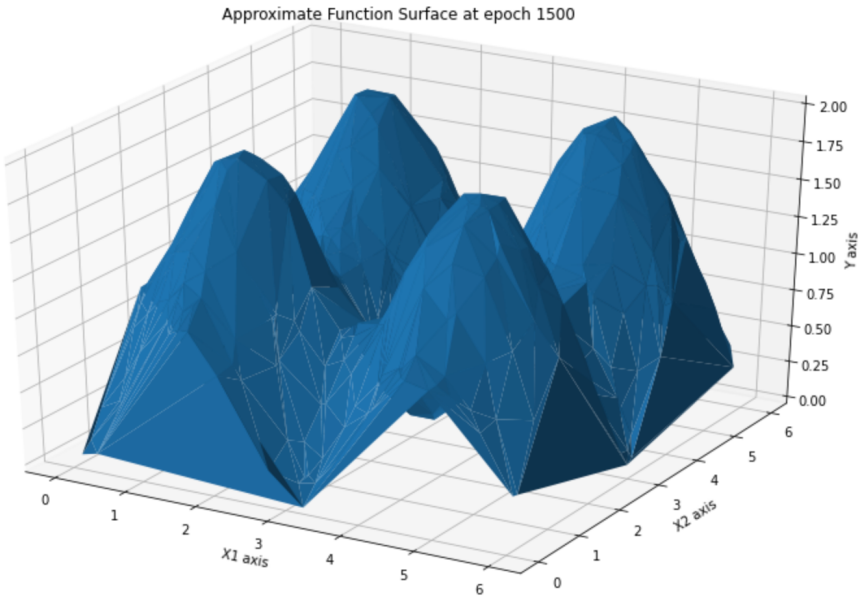


Figure 8: Plot of all data applied on the model after 1500 epoch

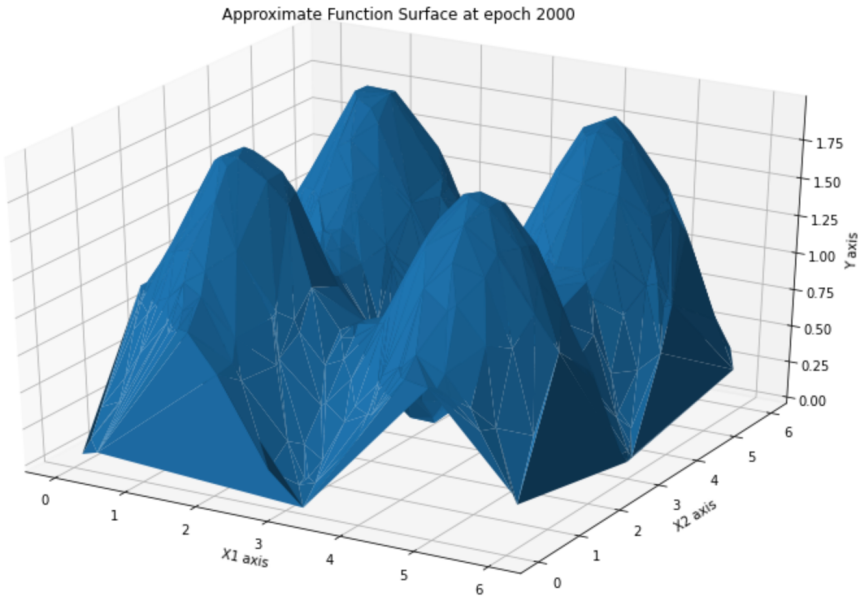


Figure 9: Plot of all data applied on the model after 2000 epoch

Next I plot the error plot for the t

1.2 Implementation Details:

To prevent the model from overfitting (blah blah blah)

raining data, to see how well the training data learns with multiple epochs. We observe that the loss steeply drops within the first few epochs and them asymptotically reaches to a value after which it is not able to learn much, and oscillates around the found local optimum.

Figure 10: Training data error vs Number of epochs

The scatter plot after after training the model for 1500 epochs with the above parameters is as follows.

1.3 Observations

- As the number of epochs increases, the average error decreases. After a point, the output just oscillates around the local optimum.
- Train average loss: 0.00167
- Test average loss: 0.00114
- The scatter plot showa that the data points almost lie along the $x = y$ line, showing the model has learnt well.
- The final plot obtained at 2000 epochs almost resembles the actual function plot, except it is more sharper at the edges.

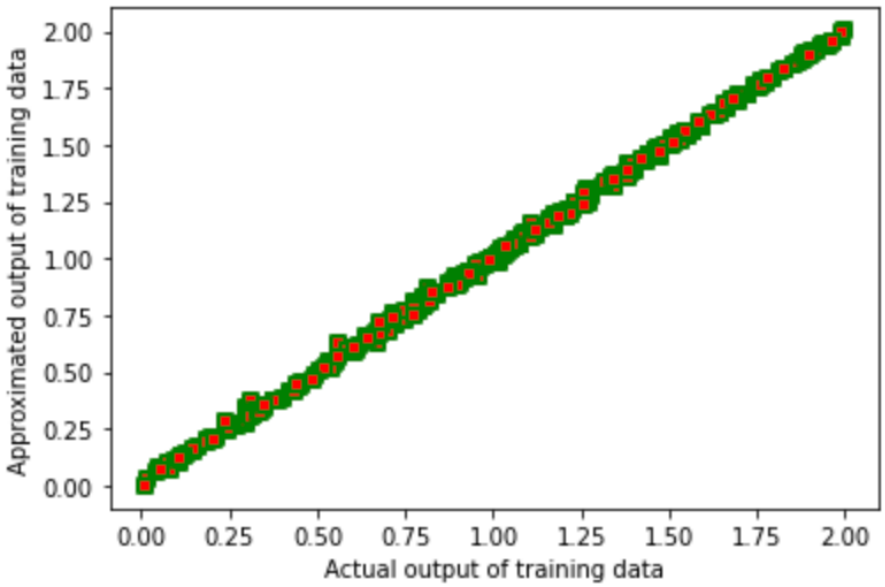


Figure 11: Scatter plot of actual output vs model obtained output of training data

- The validation loss varies with the epochs as follows:

  Validation average loss at epoch 0000: 0.217057
  Validation average loss at epoch 0100: 0.106351
  Validation average loss at epoch 0200: 0.050664
  Validation average loss at epoch 0300: 0.005119
  Validation average loss at epoch 0400: 0.006670
  Validation average loss at epoch 0500: 0.002086
  Validation average loss at epoch 0600: 0.001214
  Validation average loss at epoch 0700: 0.001714
  Validation average loss at epoch 0800: 0.001274
  Validation average loss at epoch 0900: 0.001025
  Validation average loss at epoch 1000: 0.000955
  Validation average loss at epoch 1100: 0.000678
  Validation average loss at epoch 1200: 0.000656
  Validation average loss at epoch 1300: 0.000648
  Validation average loss at epoch 1400: 0.000698
  Validation average loss at epoch 1500: 0.000476
  Validation average loss at epoch 1600: 0.003656
  Validation average loss at epoch 1700: 0.000401
  Validation average loss at epoch 1800: 0.000398
  Validation average loss at epoch 1900: 0.000314

- Final Train average loss: 0.00041
- Final Test average loss: 0.00031

2 Classifying Image Data

The images in the dataset given to us have been mapped to features of dimension 60. The features are then fed into our Multi Layer Feed Forward Neural Network model to classify the images into five distinct classes. The architecture of our model is given in Section 1. We replace the last layer of the model with a linear layer containing five output nodes. The output nodes are the logits of the image belonging to each of the distinct classes.

We have to compare the performance and the number of epochs taken for convergence for different weight update algorithms such as delta, generalized delta and Adam. Delta rule is simple stochastic gradient descent with momentum of 0. Generalised delta contains non zero momentum value Adam algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

2.1 Hyper Parameter Optimization

We tune the hyperparameters such as learning rate, number of neurons in hidden layer 1, number of neurons in hidden layer 2 in our experiments. We use the validation data to tune the hyperparameters. We use grid search method to tune our hyperparameters.

2.2 Results

2.2.1 Comparison of weight update rules using same hyperparameters

We choose a fixed set of hyperparameters for our model. We use the same weight initialisation while comparing the different weight update rule. We plot the training loss, validation loss, training accuracy and validation accuracy with respect to epochs. The hyperparameters that we choose for the model is 64 as the number of neurons in hidden layer 1, 64 as the number of neurons in hidden layer 2, 1e-4 as the learning rate and 0.8 as the momentum.

2.2.2 Confusion Matrix and Training Loss for optimum hyperparameters

We optimise the hyperparameters using the validation data. Using these optimum hyperparameters, we plot the confusion matrix and the training loss variation for different weight update rules.

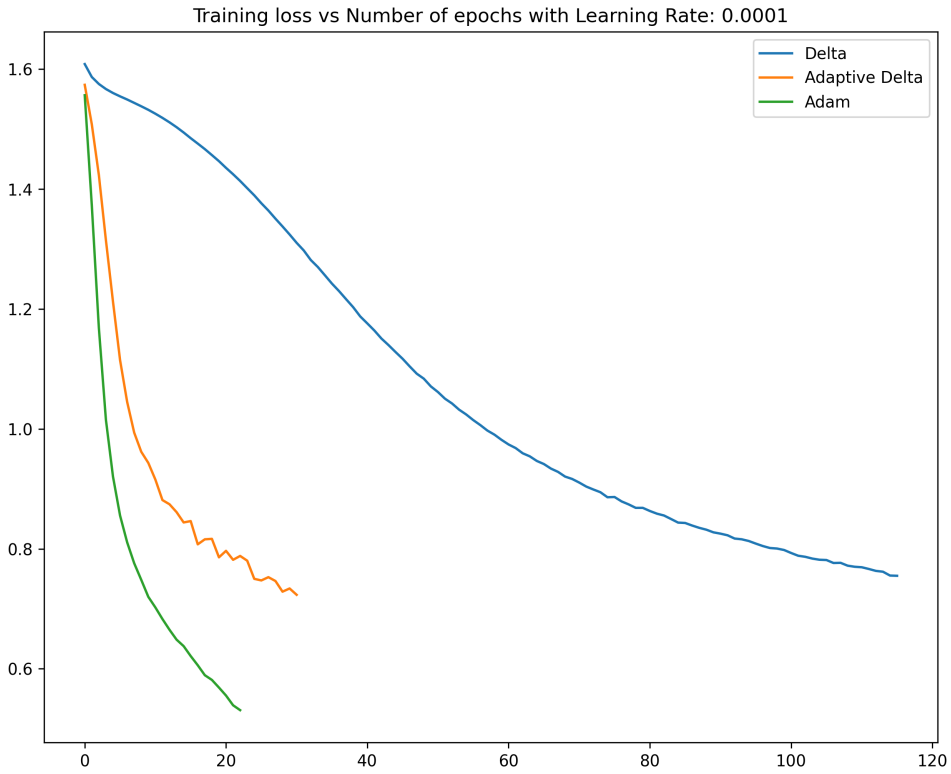


Figure 12: Comparing the training loss vs number of epochs for different weight update rules utilising same hyperparameters

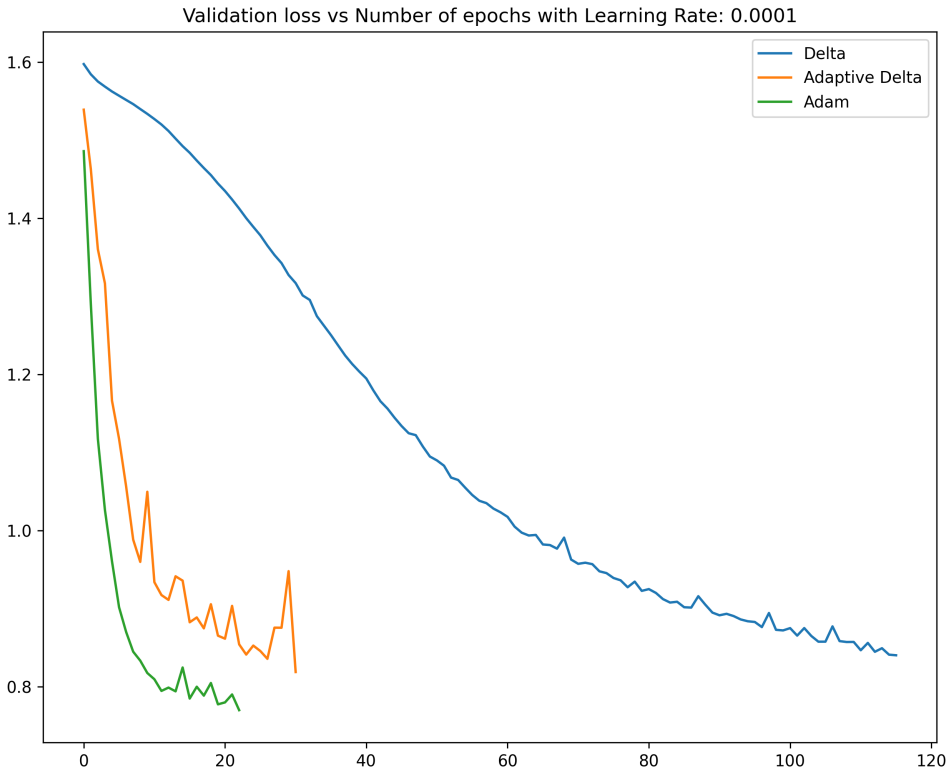


Figure 13: Comparing the validation loss vs number of epochs for different weight update rules but utilising the same hyperparameters

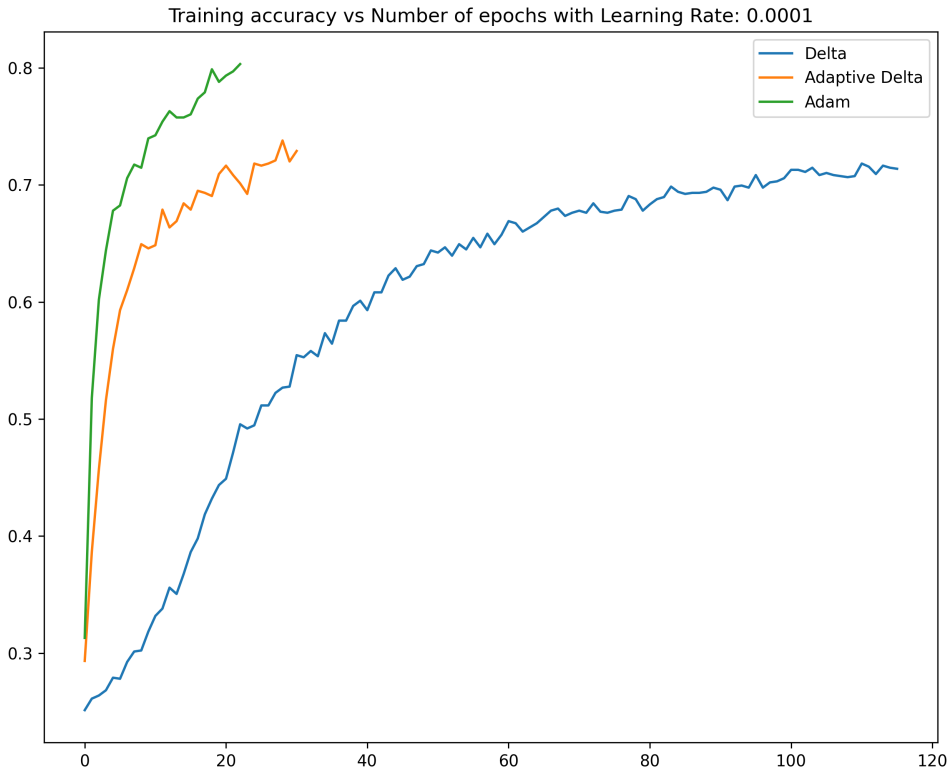


Figure 14: Comparing the training accuracy vs number of epochs for different weight update rules but utilising the same hyperparameters

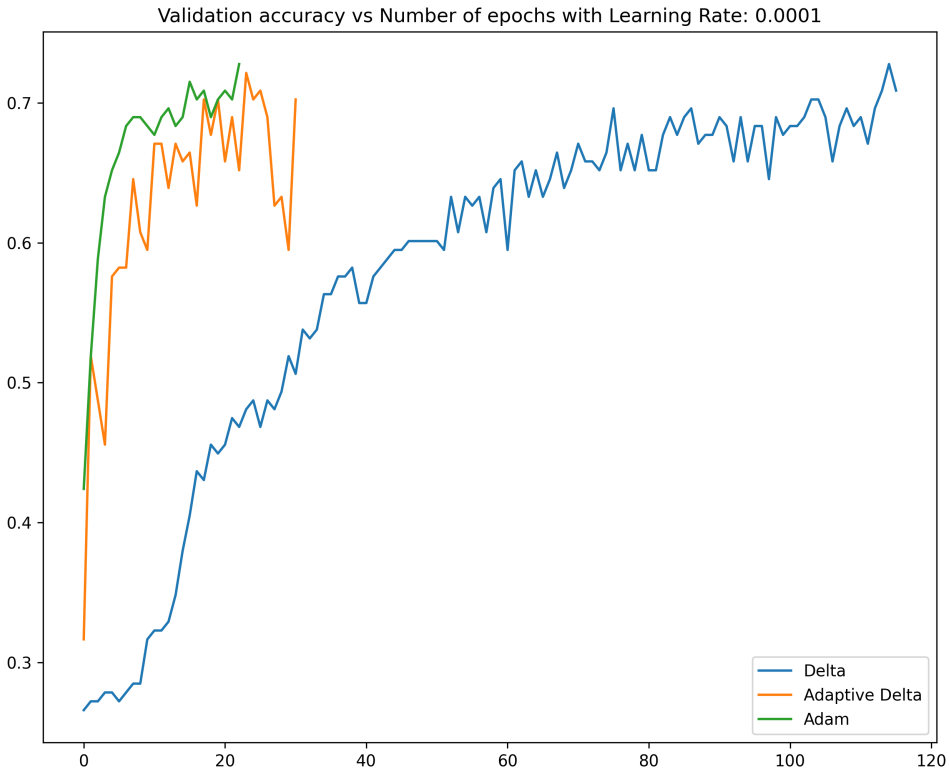
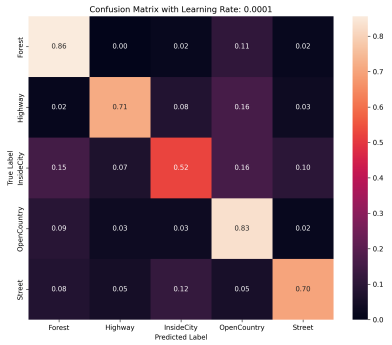
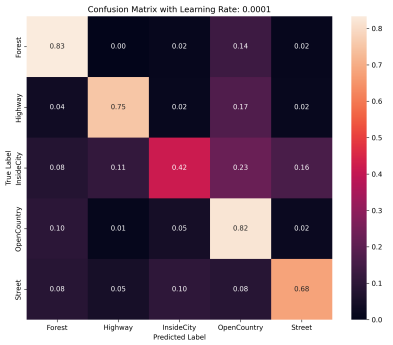


Figure 15: Comparing the validation accuracy for different weight update rules but utilising the same hyperparameters

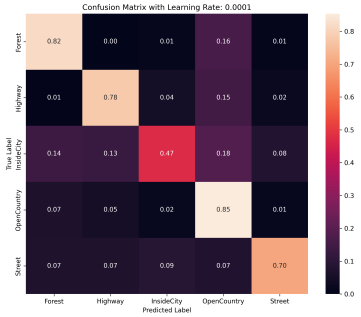


(a) Training Data

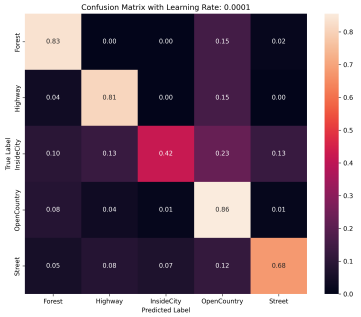


(b) Test Data

Figure 16: Confusion Matrix for Delta Rule on training and test dataset

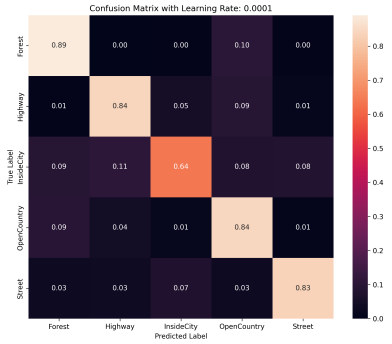


(a) Training Data

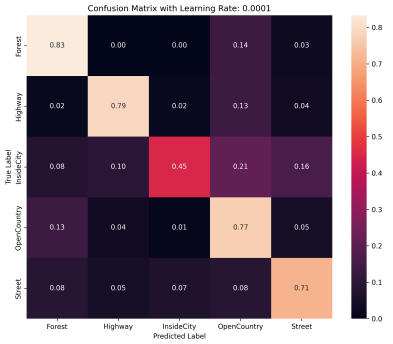


(b) Test Data

Figure 17: Confusion Matrix for Generalised Delta Rule on training and test dataset



(a) Training Data



(b) Test Data

Figure 18: Confusion Matrix for Adam Rule on training and test dataset

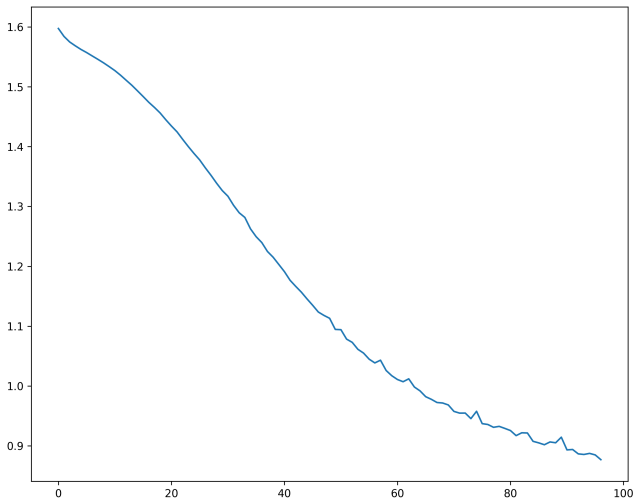


Figure 19: Training loss vs number of epochs for delta rule

Using our most optimal hyperparameters, we plot the training loss and validation loss variation for different weight update rules.

2.2.3 Number of Epochs taken for different weight update rules

The epochs taken for convergence for different weight update rules are given in Table 1

	Weight update rule		
Hyperparameters	Delta	Generalised Delta	Adam
(64,64,$1e^{-4}$,0.8)	116	35	**22**
(32,32,$1e^{-5}$,0.9)	500	261	**88**
(16,16,$1e^{-3}$,0.8)	11	6	**2**

Table 1: Comparison of number of epochs taken for different weight update rules. The format of hyperparameters is (number of neurons in hidden layer 1,number of neurons in hidden layer 2,learning rate,momentum (for generalised delta and adam))

2.3 Observations

- We observe that Adam converges faster than weight update rules such as delta rule and it is only slightly better than generalized delta rule.
- One interesting observation that we make is that even though Adam converges faster, delta generalizes better than Adam and thus results in slightly improved final performance.

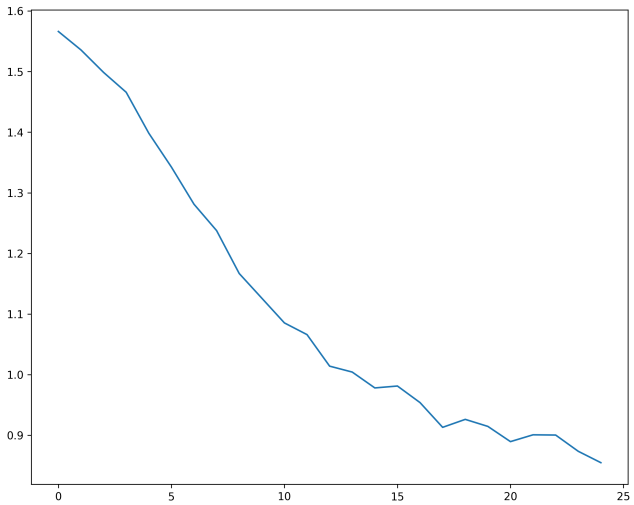


Figure 20: Training loss vs number of epochs for generalised delta rule

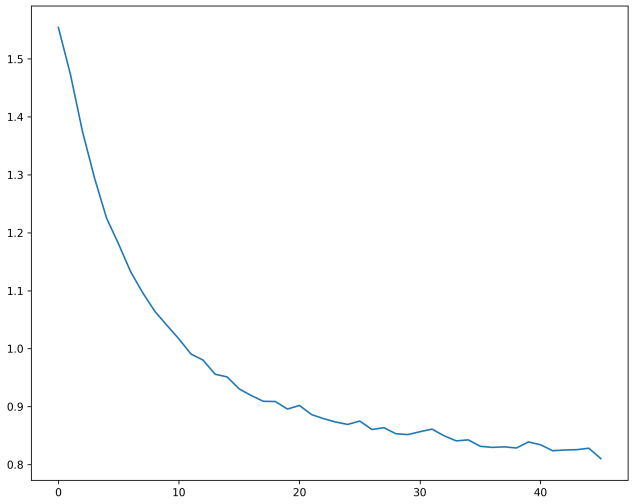


Figure 21: Training loss vs number of epochs for Adam rule

- Also, the validation loss of delta and generalized delta rule is more stable compared to Adam rule.
- The reason why Adam converges faster is that it adaptively selects a separate learning rate for each parameter. Parameters that would ordinarily receive smaller or less frequent updates receive larger updates with Adam
- In the confusion matrix, we observe that the model gets confused between classes such as insidecity and street which is to be expected as both classes are visually similar
- When the learning rate is lesser than $1e^{-4}$, the time taken for convergence (especially for delta rule) is much longer.
- When the value of momentum is less such as 0.2-0.5, time taken for generalised delta and delta is quite similar.
- Also, overfitting is very much prelavent as number of neurons increase in the hidden layer. To mitigate that, we use early stopping rule. x''