Assignment 1 : Neural Network Implementation

Emil Biju - Sundar Raman P - Kommineni Aditya (**Group 16**)

March 3, 2020

1 Assignment objectives

- Function approximation of 2-D input
- Single-label multi-class classification for 2-D Input
- Single-label multi-class classification for image data

2 Important Libraries & Modules used

- `matplotlib` : Python 2D plotting library.
- `numpy` : Python library for numerical computations.
- `pandas` : A python module which enables robust handling of data.

3 Implementation of Neural Network Explained

This section aims to give an insight into the manner in which the forward and back propagation for the neural networks have been implemented. The computation of the forward iteration and backward gradients have been vectorized which improve the training time of the model.

3.1 Forward Pass

In the forward pass, the equations to calculated the node outputs are as follows:

Let the network have L layers and a_l h_l denote the pre-activation and output of the layers respectively where l denotes the layer number. Below, W_{l-1} refers to the weight matrix between the layer l-1 and l .

h_i = Input sample

i = 1 to L :

$$a_i = W_{l-1}h_{l-1}$$
$$h_i = g(a_i)$$

3.2 Backward Pass

From the output layer node values, we calculate the error for that particular training sample. Then, we use the below equations to obtain the gradients of various weight parameters. Let the model have L layers

$\nabla \mathrm{E}_{a_L} = \nabla \mathrm{E}_{h_L} (\partial \mathrm{h}_L / \partial \mathrm{a}_L)$

i = L to 1 :
$\nabla \mathrm{E}_{W_{i-1}} = \nabla \mathrm{E}_{h_i}\ h_{i-1}$
$\nabla \mathrm{E}_{h_{i-1}} = W^T \nabla \mathrm{E}_{a_i}$
$\nabla \mathrm{E}_{a_{i-1}} = \nabla \mathrm{E}_{h_{i-1}}\ (\partial \mathrm{h}_{i-1} / \partial \mathrm{a}_{i-1})$
weight update rule

From the above equations, we can find the gradients of any number of hidden layered neural network.

3.3 Weight Update Method

Once we have the output of the layers given the input, we must calculate the changes to the weights which must be made in order to reach the local minima. There are various weight update methods but, below are the rules which have been used in the assignment.

Delta Method - This is the simplest weight update rule wherein the learning parameter value remains constant throughout the training process. The weight update rule is as follows :

$$w(m+1) = w(m) - \eta \frac{\partial E}{\partial w}$$

Generalized Delta Rule - This method takes into account the previous weight changes and in general is faster than the delta rule. We have another parameter known as momentum in addition to the learning rate. The weight update rule is as follows :

$$w(m+1) = w(m) - \eta \frac{\partial E}{\partial w} + \alpha \Delta w(m-1)$$

Adam Optimiser - This rule takes into account both the first and second moment of the gradients of the weights. This method in general is superior to the previously mentioned methods. Here, q and r are intermediate variables where storing just one old values saves us from recursion each time. g stands for the gradient of error with respect to weights, m denotes the iteration number (which continuously increases in parameter update with each update), p1 and p2 are hyper parameters (played around with and finally

takes values very close to 1) and epsilon is set to 1e-8 to avoid a run time error due to denominator becoming zero.

$$q_w(m) = p_1 * q_w(m-1) + (1-p_1) * g_w(m)$$
$$r_w(m) = p_2 * r_w(m-1) + (1-p_2) * g_w^2(m)$$
$$a_w(m) = \frac{q_w(m)}{(1-p_1^m)}$$
$$b_w(m) = \frac{r_w(m)}{(1-p_2^m)}$$
$$w(m+1) = w(m) - \eta \frac{a_w(m)}{\epsilon + \sqrt{b_w(m)}}$$

3.4 Derivative of softmax

The softmax function which is in final output layer of the model to give output probabilities for single label multi class classification problems (2) (3). It's derivative is as shown below which was used for figuring out the weight update rule. j is index of node which corresponds to correct output, i is a placeholder for any output index, $s^{'}$ is derivative of softmax and y is the final softmax output.

$$if(i == j)$$
$$s^{'} = y_i * (1 - y_i)$$
$$else$$
$$s^{'} = -y_i * y_j$$

4 Function approximation of 2-D input

In this section, we are given a 2-D input, which is passed through a neural network with 2 hidden layers to generate a single output. The input layer has 2 nodes to accept the 2-D input. Various values for the number of hidden layer nodes were experimented with and the final results are given below. The Mean Squared Error loss function is used for training and evaluation. The best results have been obtained with the following configuration:

Learning rate: 0.05
Momentum : 0.8
Number of nodes in the hidden layers : 10 and 8
Activation function for hidden layers : sigmoid
Activation function for output layer : linear

Weight update rule : Generalised Delta Rule
Parameter of linear activation : 1.0

4.1 Implementation details

- The input data is normalized before passing into the model. This prevents the issue of large exponents while applying the activation function.
- The weights are initialized as standard Gaussian random variables using the `np.random.randn` function.
- The `tanh` and `sigmoid` activation functions provide almost similar results with the `sigmoid` activation function giving slightly lower average loss.
- For finding the surface plot of the desired function, a scatter plot of the data points provided was made and the triangulation method was applied to separate the plane into triangles. Then the `plot_trisurf` method is applied to get the surface plot.
- For obtaining the surface plot of the approximated function, a meshgrid is created with equally spaced values over the range of input data. Our model predicts value at each coordinate and the surface plot is generated using the resulting tuples.

4.2 Results

- **Aveage MSE Loss achieved on training data** = 15.58
- **Average MSE Loss achieved on validation data** = 17.42

4.3 Inference

- Increasing the learning rate beyond a point causes the training error to increase after many iterations. This is because the weight update becomes so large that the newly updated paramter corresponds to a larger error on the other side of the Loss vs Parameter graph.
- After a large number of epochs, the validation error starts to increase, while the training error keeps decreasing. This is due to overfitting.
- The validation and training errors in our final model are quite close and decrease continuously during training. This implies that there is no overfitting.

4.4 Plots

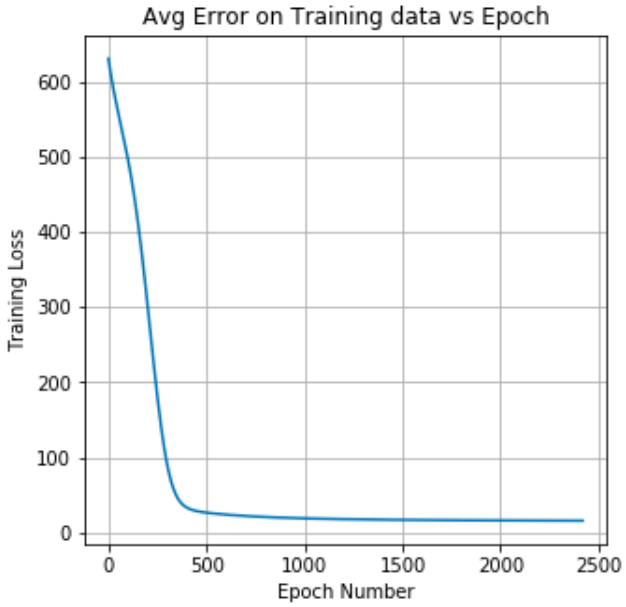


Figure 1: Average Error vs Epoch

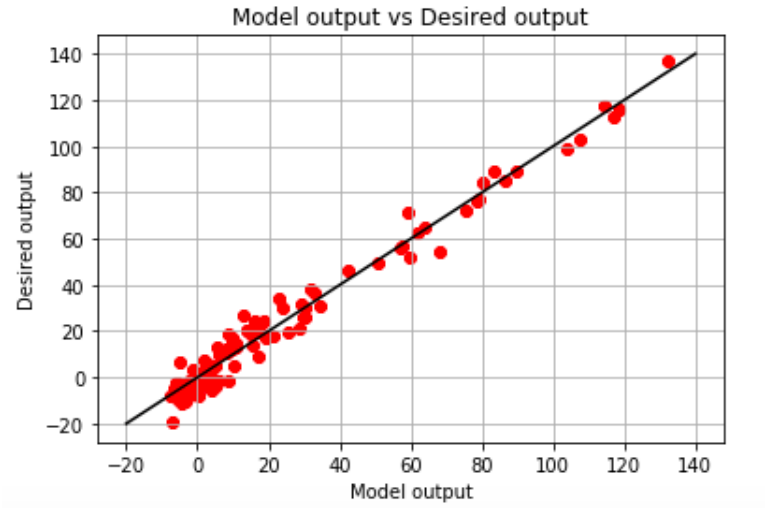


Figure 2: Scatter plot of Model output and Desired output for Training data

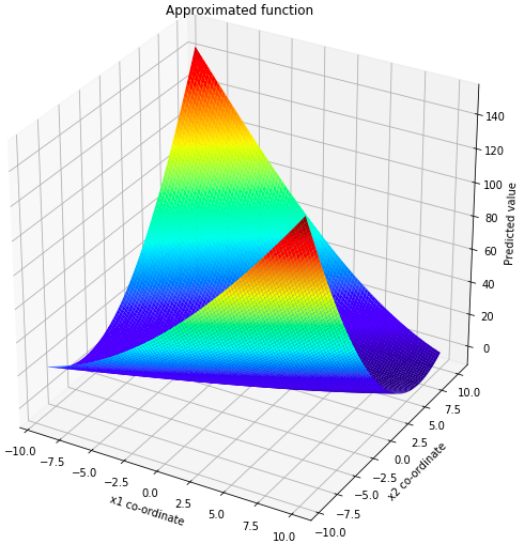


Figure 3: Surface Plot of Approximated function

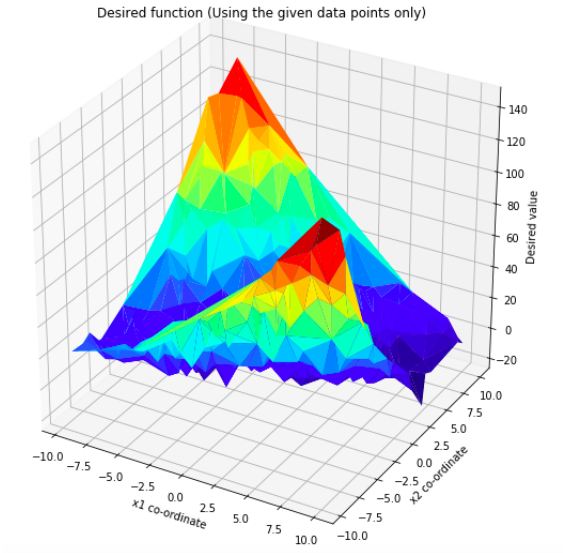


Figure 4: Surface Plot of Desired function

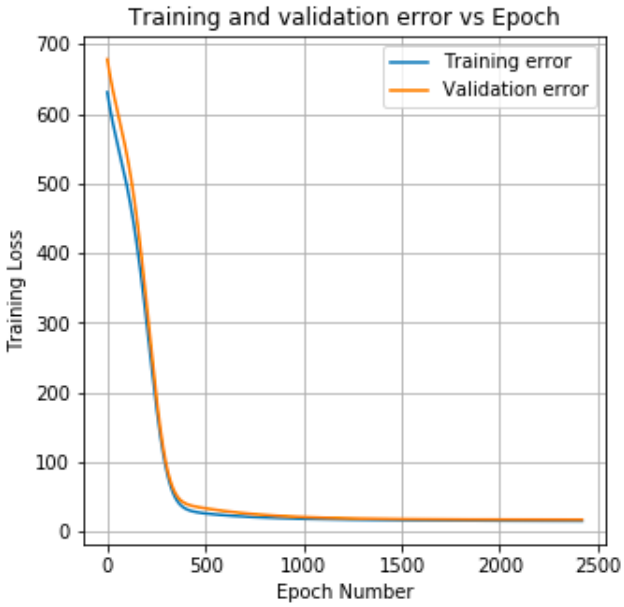


Figure 5: Training and validation error vs epochs

5 Classification task for 2-D data

5.1 Model Parameters and Implementation

This section describes the implementation of the neural network for classification of a non linear 2-D data. Firstly, the data was split into train and test in the ratio of 3:1. Following which, the output data was one hot vectorized to facilitate the implementation of softmax output activation function.

Akin to the previous problem, the weights have been intialised randomly using the *numpy.random.randn* function. We could ensure that the same set of weights are generated while testing over a range of learning parameters through setting numpy.random.seed to a certain integer value.

The stop parameter for the training was set to the difference between the previous epoch error and present epoch error to be less than 1e-5.

The values of the hyper parameters which yielded the best performance are as given below :

Learning rate: 1e-3
Momentum : 0.8
Number of nodes in each hidden layer : 5
Activation function for hidden layers : tanh

Activation function for output layer : softmax
Weight update rule : generalised delta rule
Accuracy on validation set : 0.831

5.2 Observations

5.2.1 Complexity of the model

We need to obtain the values of the hidden layers for which the model gives the optimal performance and doesn't overfit. From the plots below, we can see that as the complexity of the model increases beyond 5 nodes per hidden layer, the model starts to overfit.

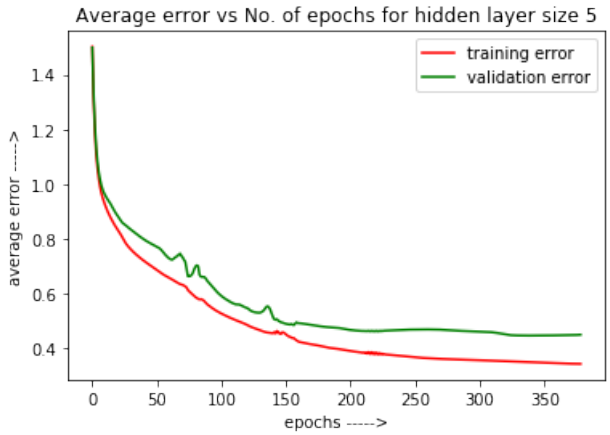


Figure 6: Average error vs No. of epochs for 5 hidden layer nodes

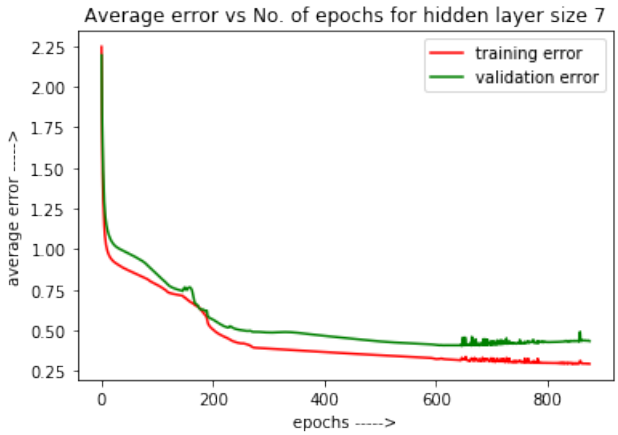


Figure 7: Average error vs No. of epochs for 7 hidden layer nodes

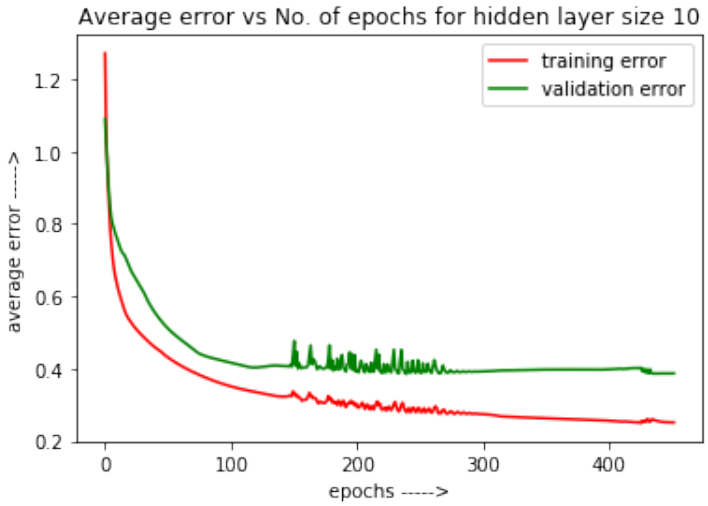


Figure 8: Average error vs No. of epochs for 10 hidden layer nodes

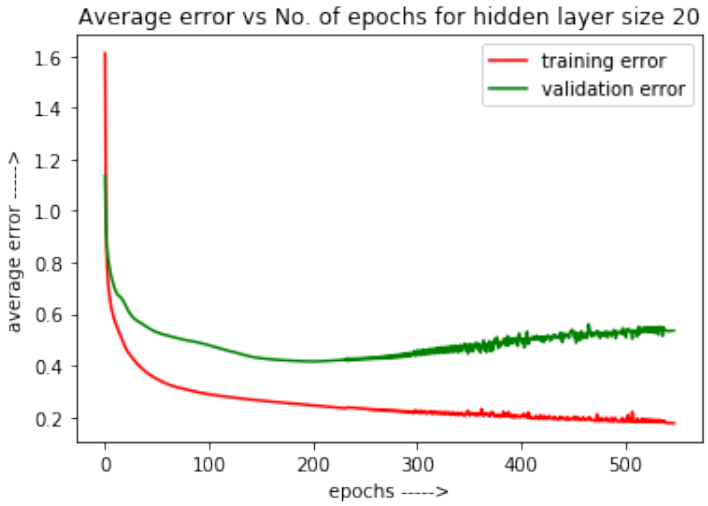


Figure 9: Average error vs No. of epochs for 20 hidden layer nodes

The above observation could be attributed to the fact that since the number of training examples are low in number and we have an empirical measure that the number of training samples needed is 10 times the number of parameters to learn, for higher number of hidden layer nodes, the model starts to overfit.

5.2.2 Surface Plots of hidden layer and output nodes

Output Node 1 Below are the surface plots for the Output Layer Node 1 across various epochs.

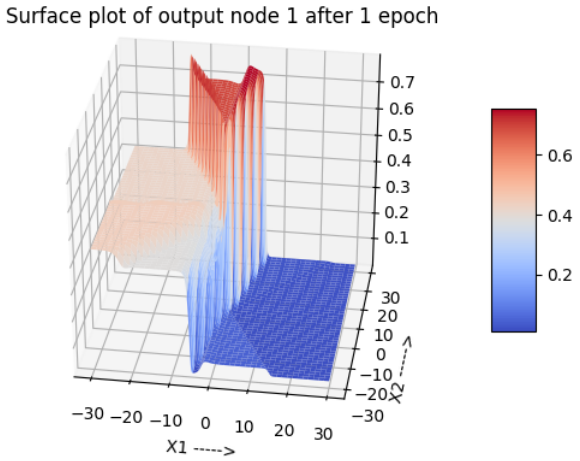


Figure 10: Surface Plot for Output Layer Node 1 after 1 epoch

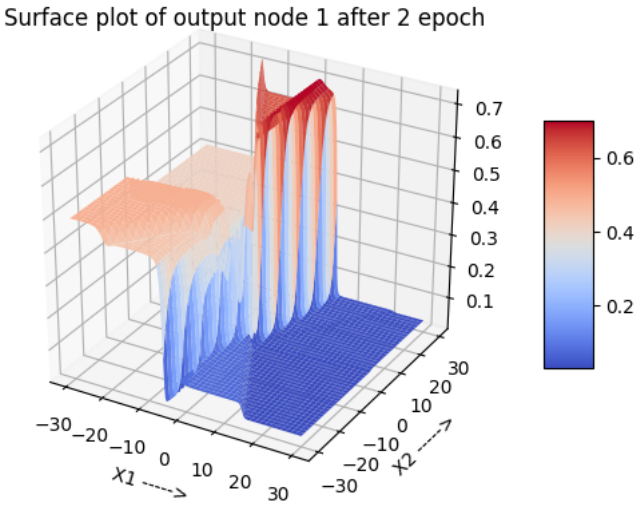


Figure 11: Surface Plot for Output Layer Node 1 after 2 epochs

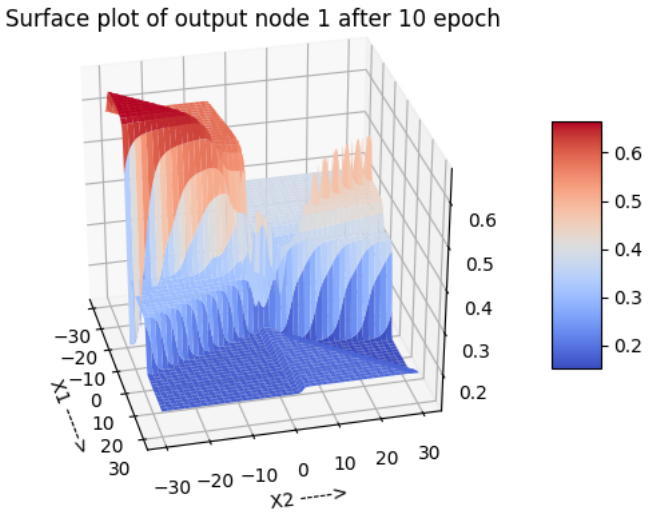


Figure 12: Surface Plot for Output Layer Node 1 after 10 epochs

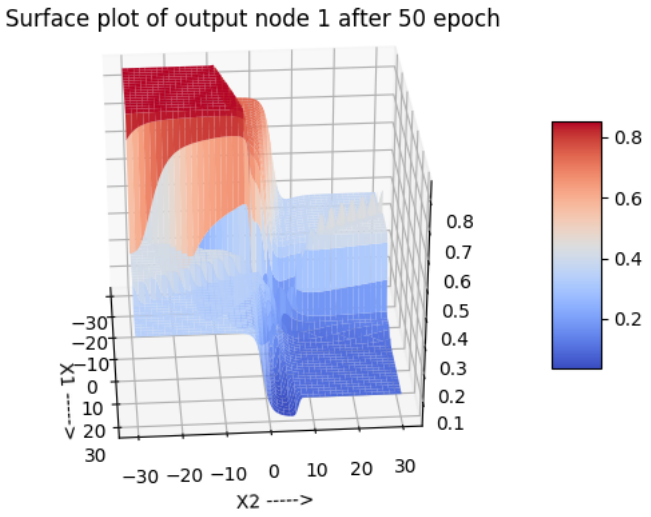


Figure 13: Surface Plot for Output Layer Node 1 after 50 epochs

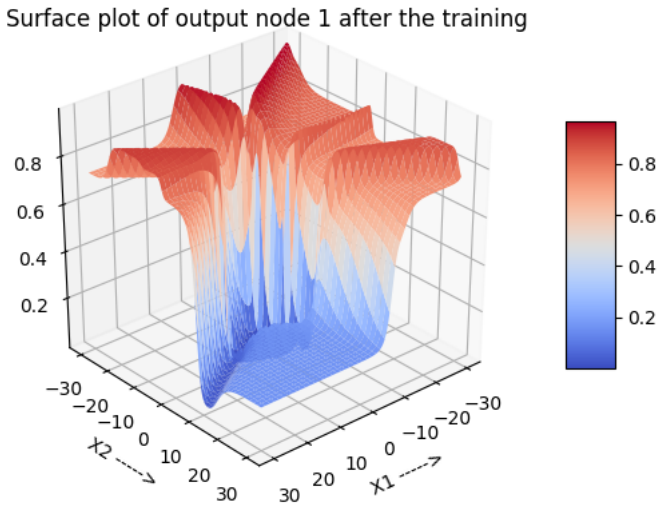


Figure 14: Surface Plot for Output Layer Node 1 after training is complete

Output Node 2 Below are the surface plots for the Output Layer Node 2 across various epochs.

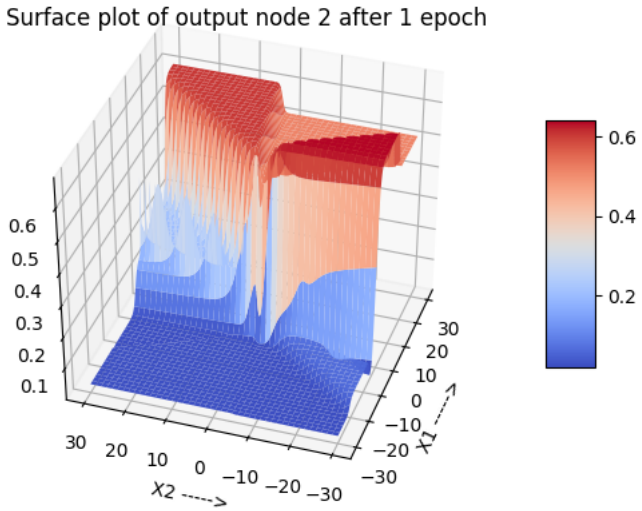


Figure 15: Surface Plot for Output Layer Node 2 after 1 epoch

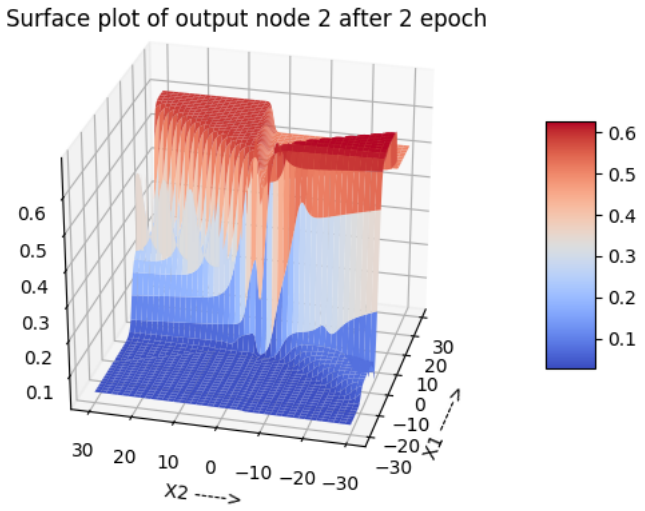


Figure 16: Surface Plot for Output Layer Node 2 after 2 epochs

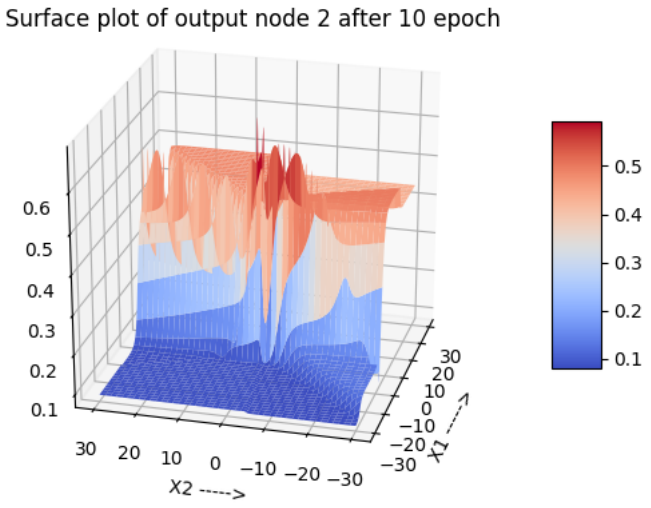


Figure 17: Surface Plot for Output Layer Node 2 after 10 epochs

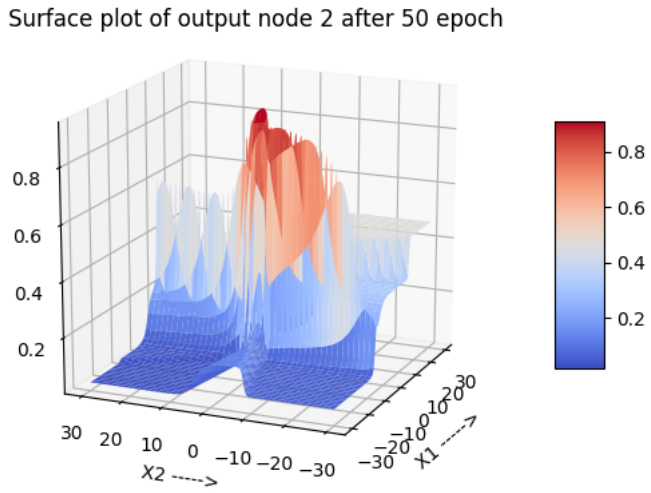


Figure 18: Surface Plot for Output Layer Node 2 after 50 epochs

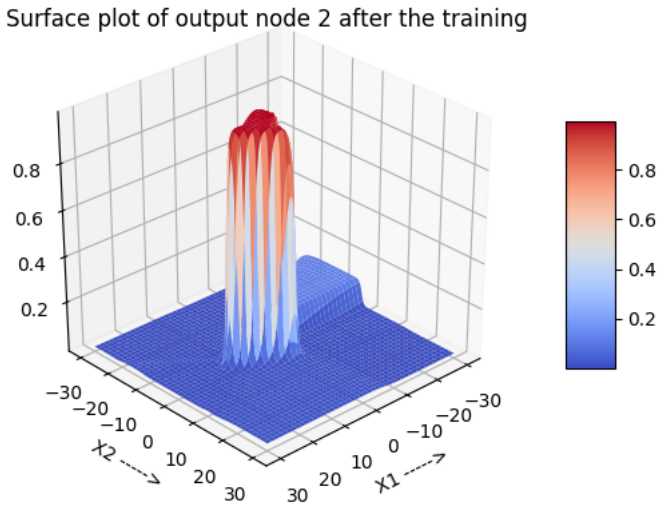


Figure 19: Surface Plot for Output Layer Node 2 after training is complete

Output Node 3 Below are the surface plots for the Output Layer Node 3 across various epochs.

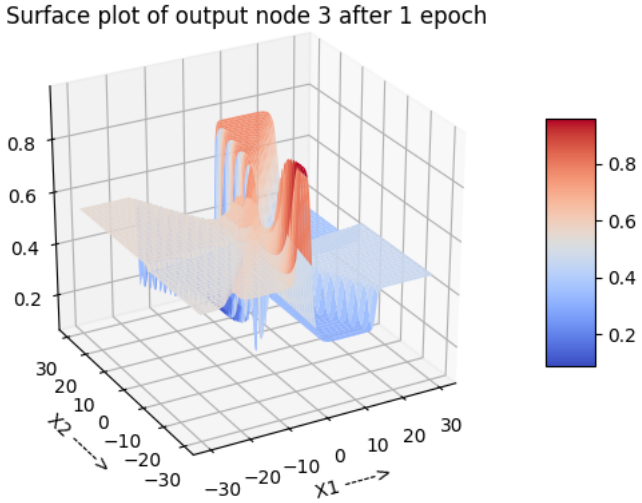


Figure 20: Surface Plot for Output Layer Node 3 after 1 epoch

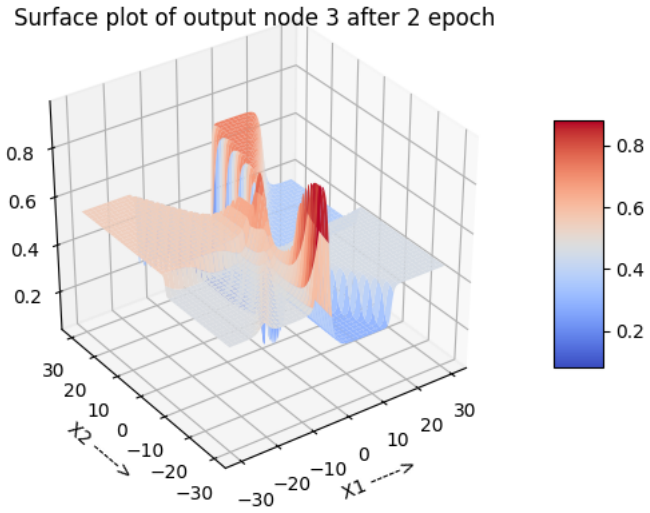


Figure 21: Surface Plot for Output Layer Node 3 after 2 epochs

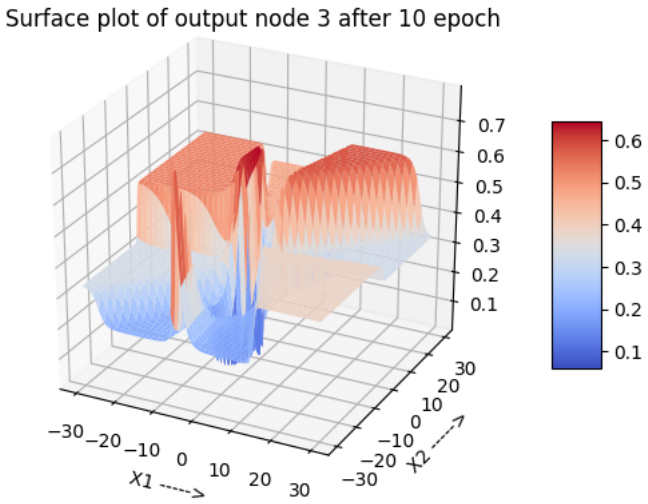


Figure 22: Surface Plot for Output Layer Node 3 after 10 epochs

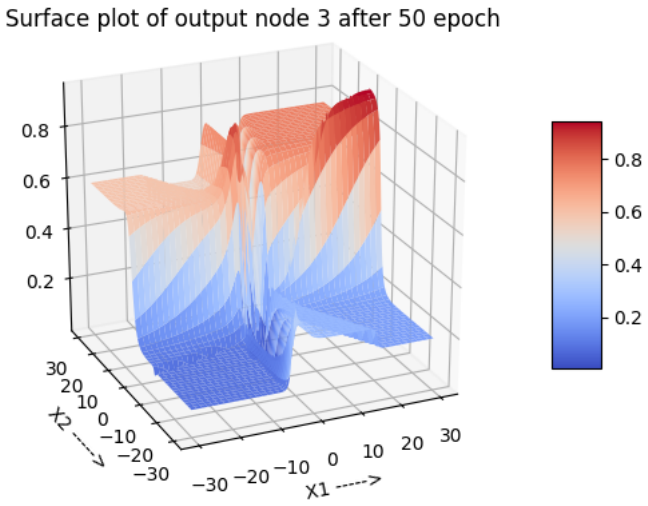


Figure 23: Surface Plot for Output Layer Node 3 after 50 epochs

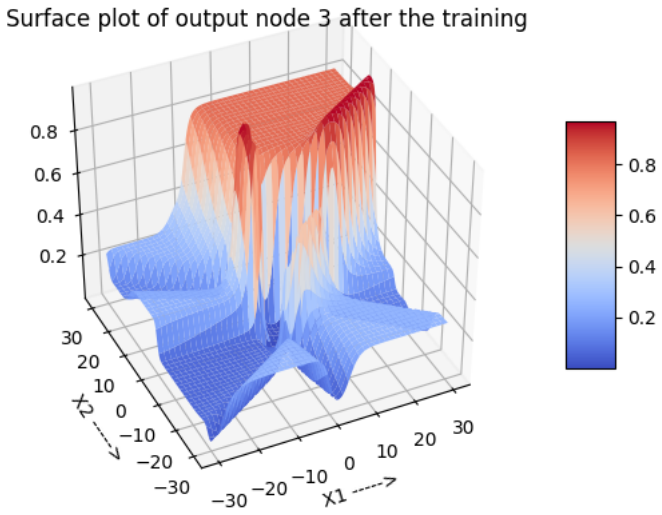


Figure 24: Surface Plot for Output Layer Node 3 after training is complete

Hidden Layer 1 Node 1 Below are the surface plots for the Hidden Layer 1 Node 1 across various epochs.

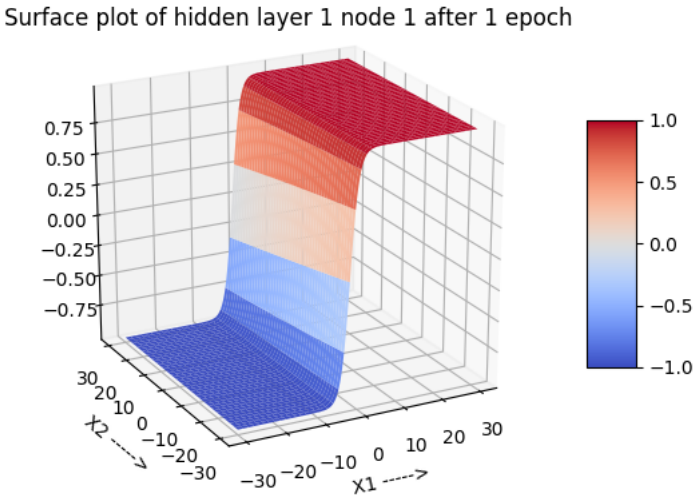


Figure 25: Surface Plot for Hidden Layer 1 Node 1 after 1 epoch

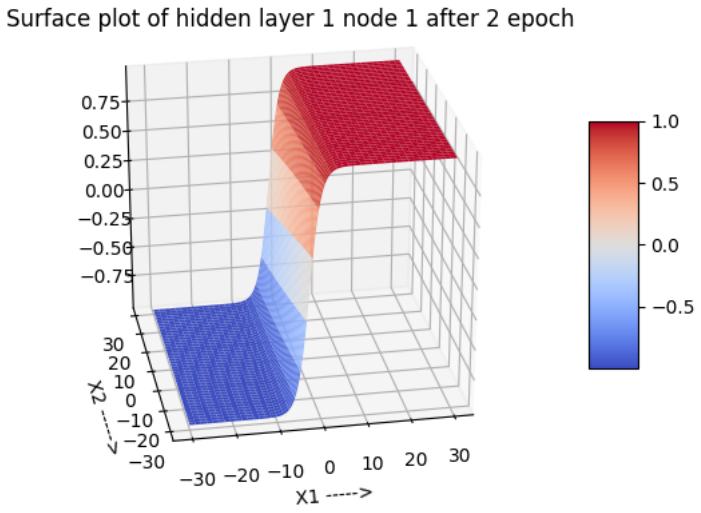


Figure 26: Surface Plot for Hidden Layer 1 Node 1 after 2 epochs

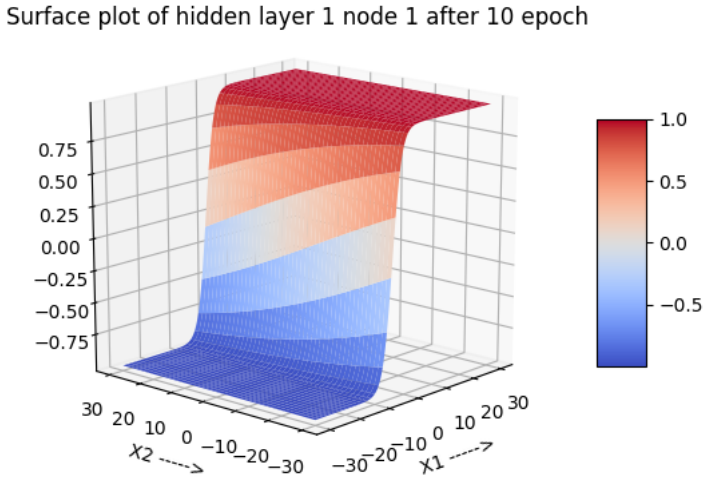


Figure 27: Surface Plot for Hidden Layer 1 Node 1 after 10 epochs

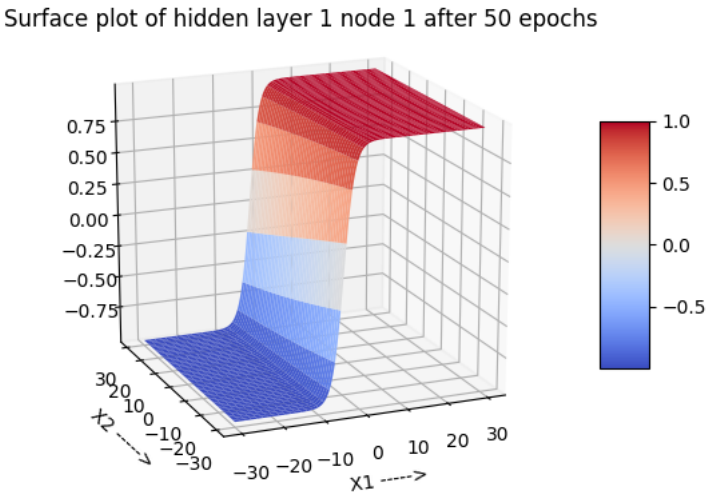


Figure 28: Surface Plot for Hidden Layer 1 Node 1 after 50 epochs

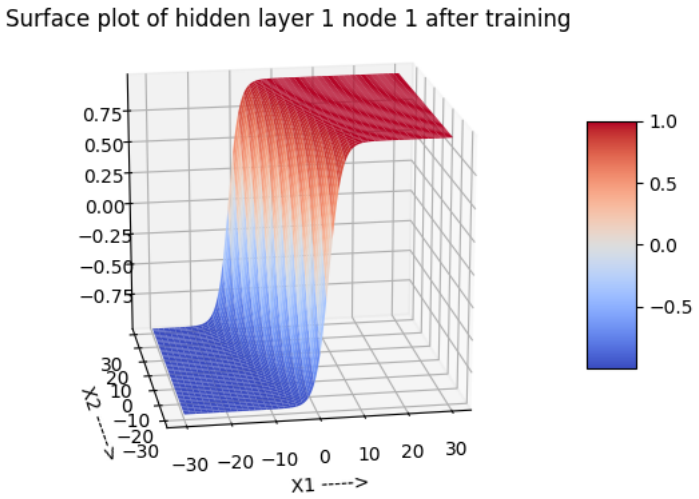


Figure 29: Surface Plot for Hidden Layer 1 Node 1 after completion of training

Hidden Layer 1 Node 2 Below are the surface plots for the Hidden Layer 1 Node 2 across various epochs.

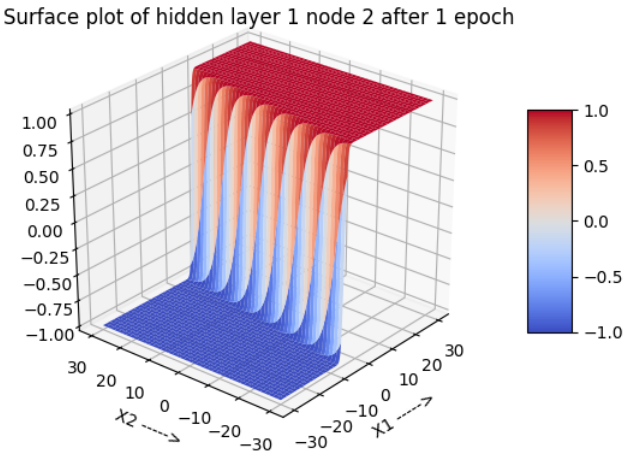


Figure 30: Surface Plot for Hidden Layer 1 Node 2 after 1 epoch

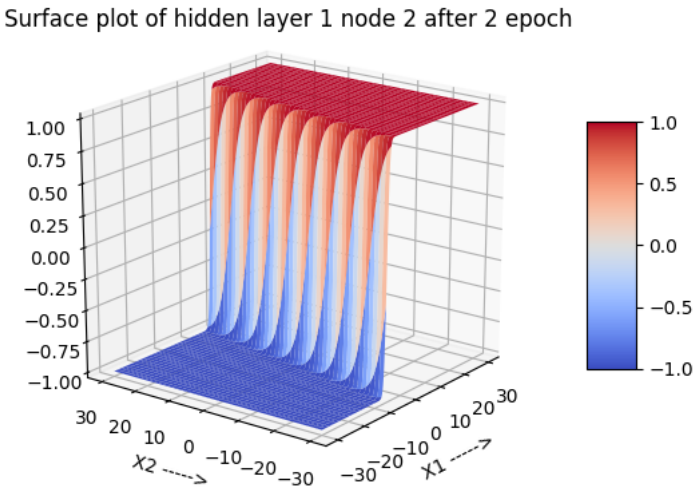


Figure 31: Surface Plot for Hidden Layer 1 Node 2 after 2 epochs

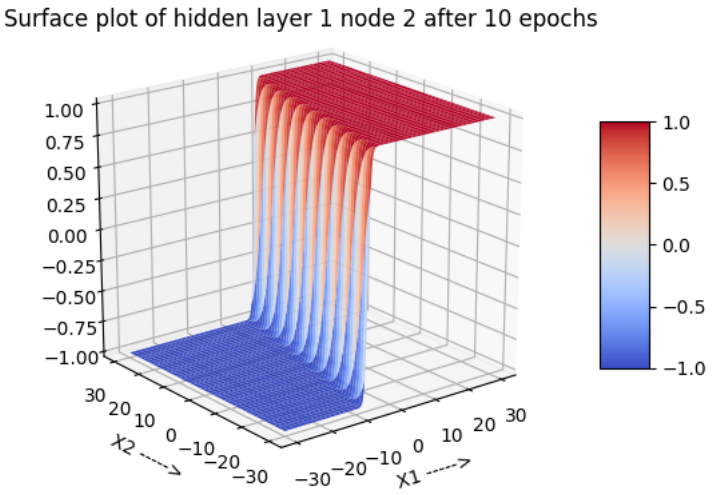


Figure 32: Surface Plot for Hidden Layer 1 Node 2 after 10 epochs

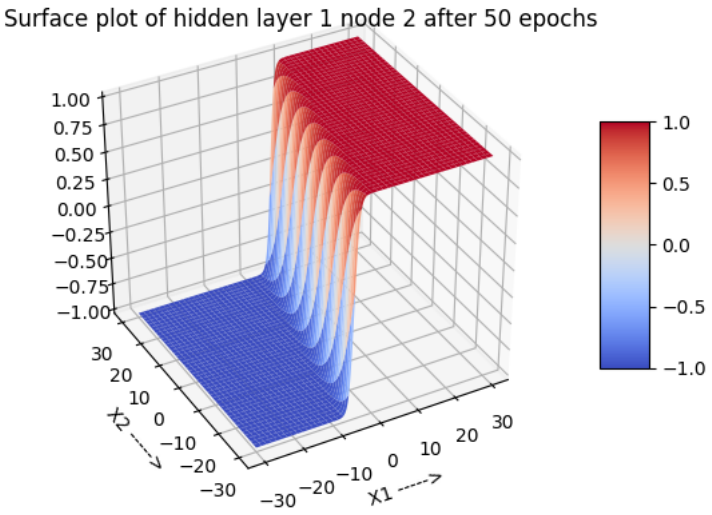


Figure 33: Surface Plot for Hidden Layer 1 Node 2 after 50 epochs

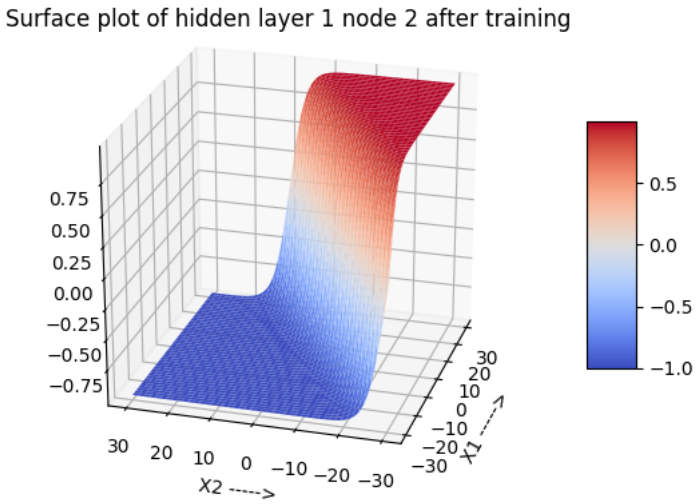


Figure 34: Surface Plot for Hidden Layer 1 Node 2 after completion of training

Hidden Layer 1 Node 3 Below are the surface plots for the Hidden Layer 1 Node 3 across various epochs.

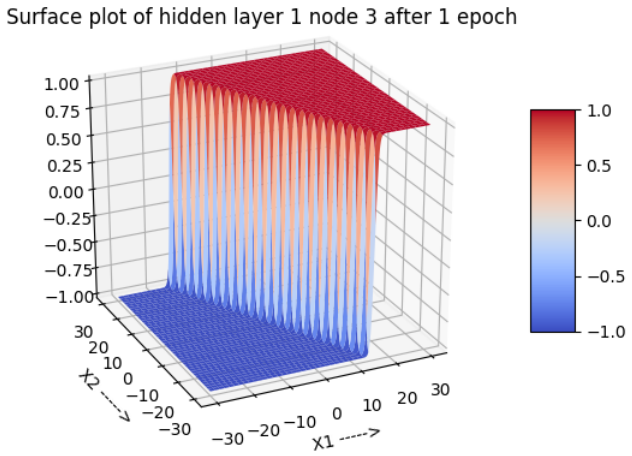


Figure 35: Surface Plot for Hidden Layer 1 Node 3 after 1 epoch

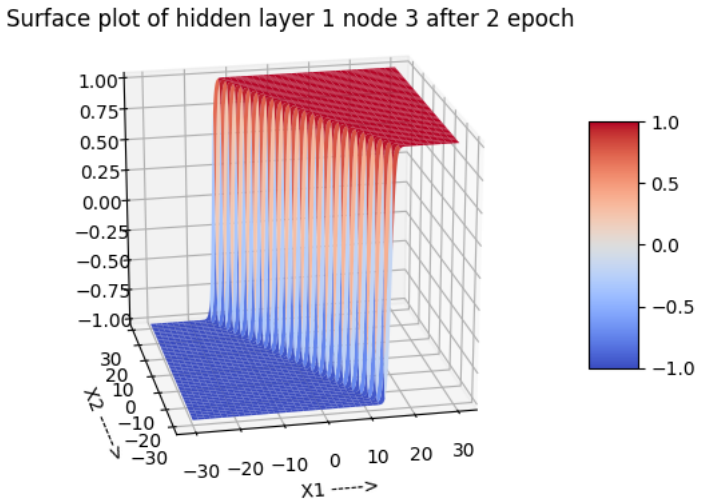


Figure 36: Surface Plot for Hidden Layer 1 Node 3 after 2 epochs

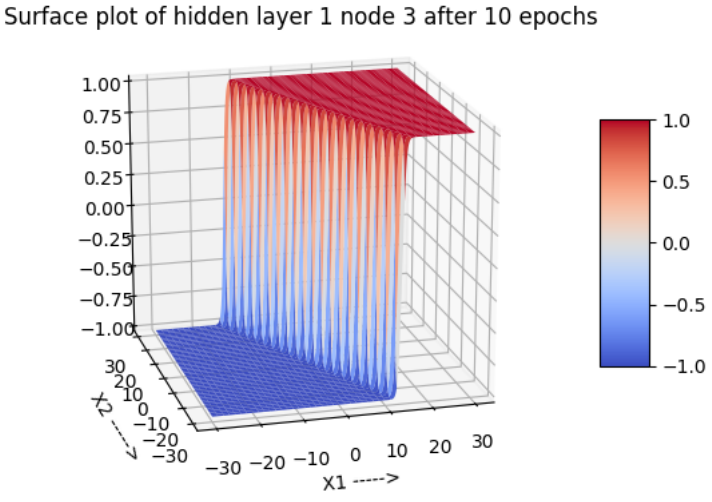


Figure 37: Surface Plot for Hidden Layer 1 Node 3 after 10 epochs

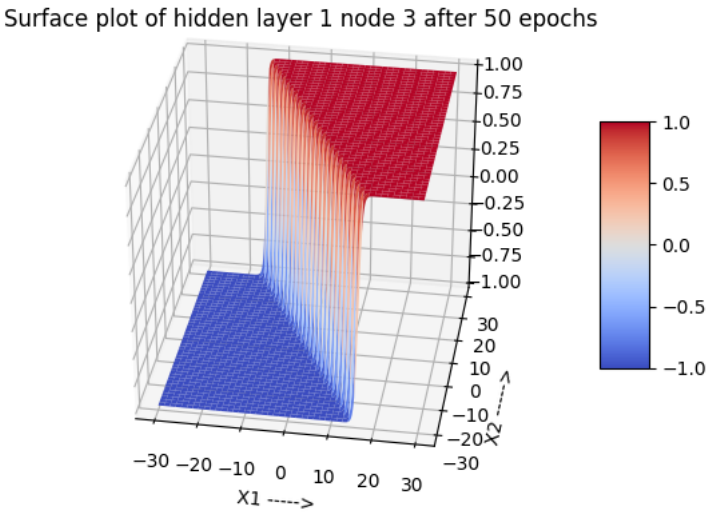


Figure 38: Surface Plot for Hidden Layer 1 Node 3 after 50 epochs

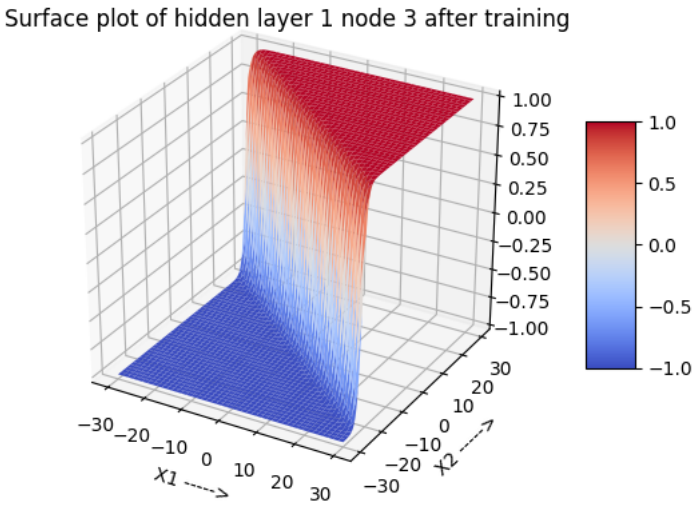


Figure 39: Surface Plot for Hidden Layer 1 Node 3 after completion of training

Hidden Layer 1 Node 4 Below are the surface plots for the Hidden Layer 1 Node 4 across various epochs.

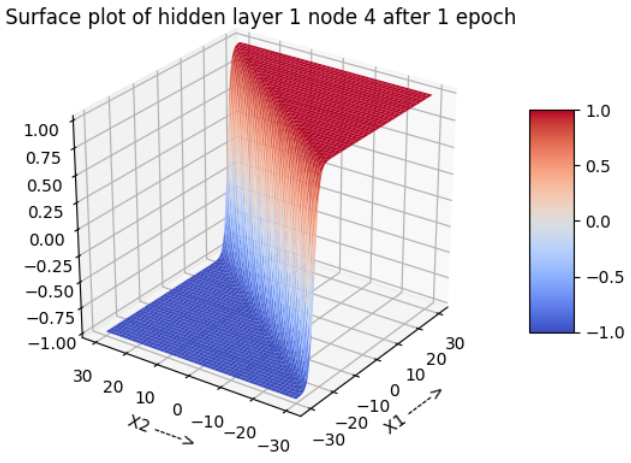


Figure 40: Surface Plot for Hidden Layer 1 Node 4 after 1 epoch

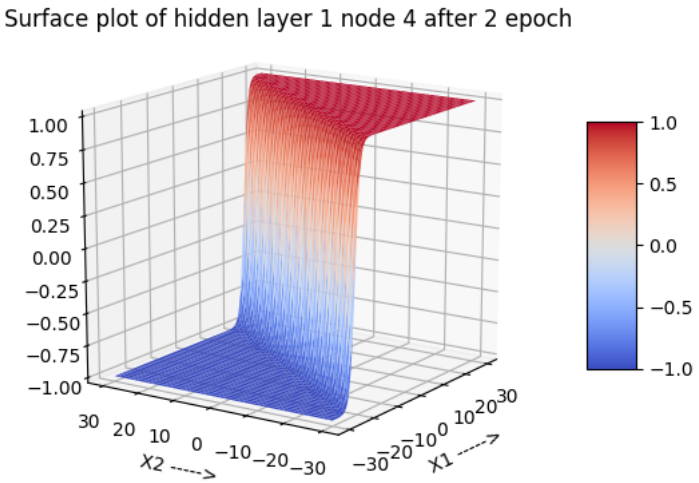


Figure 41: Surface Plot for Hidden Layer 1 Node 4 after 2 epochs

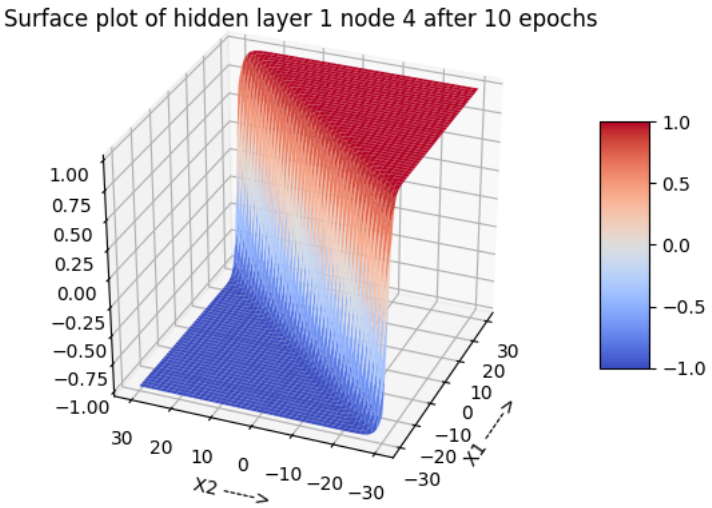


Figure 42: Surface Plot for Hidden Layer 1 Node 4 after 10 epochs

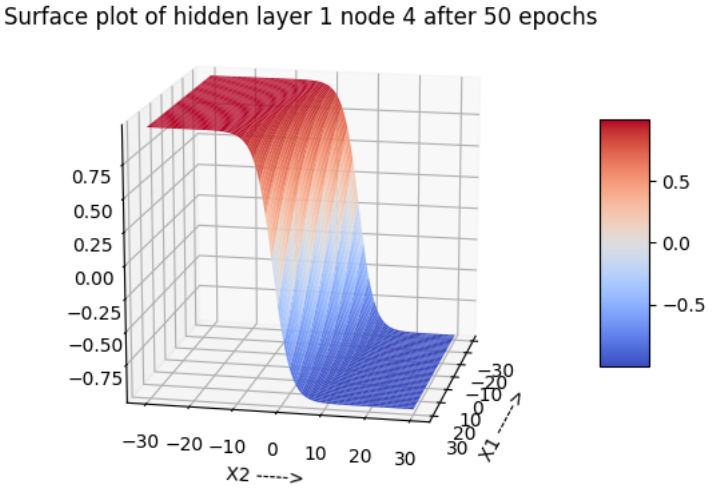


Figure 43: Surface Plot for Hidden Layer 1 Node 4 after 50 epochs

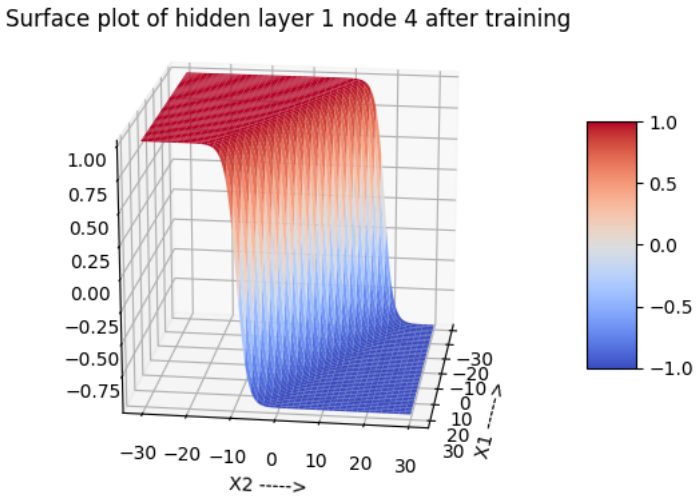


Figure 44: Surface Plot for Hidden Layer 1 Node 4 after completion of training

Hidden Layer 1 Node 5 Below are the surface plots for the Hidden Layer 1 Node 5 across various epochs.

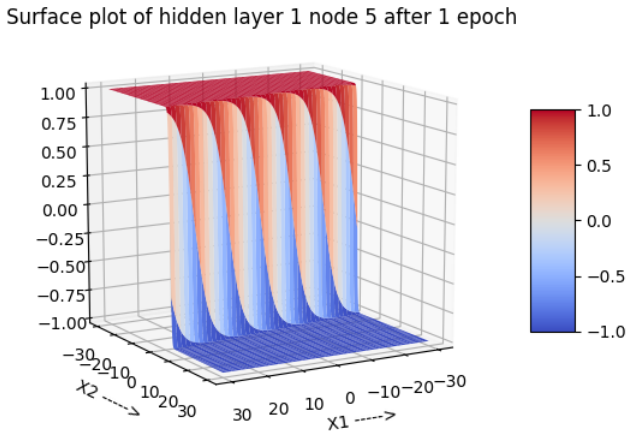


Figure 45: Surface Plot for Hidden Layer 1 Node 5 after 1 epoch

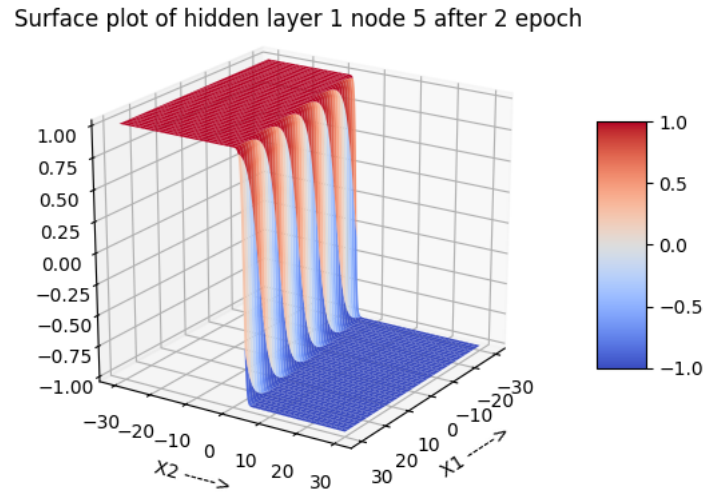


Figure 46: Surface Plot for Hidden Layer 1 Node 5 after 2 epochs

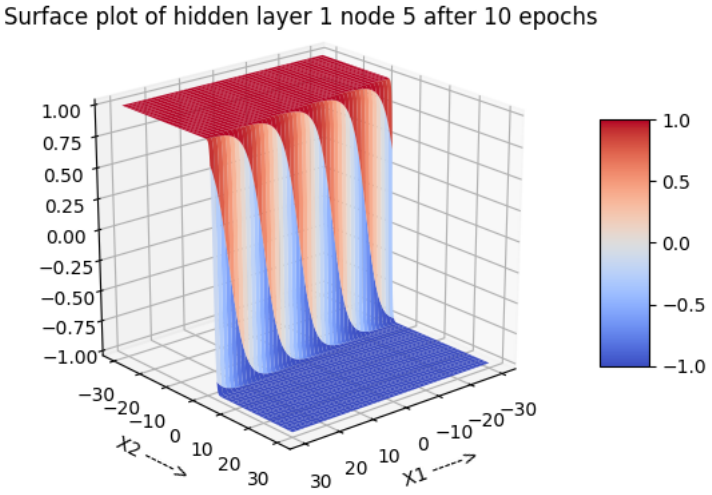


Figure 47: Surface Plot for Hidden Layer 1 Node 5 after 10 epochs

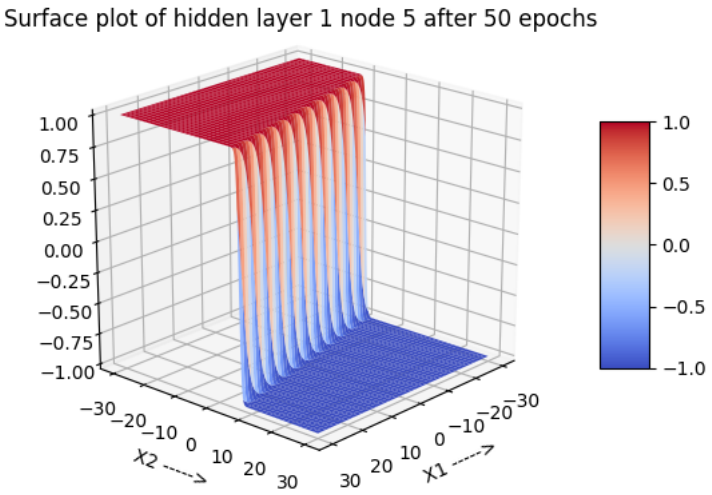


Figure 48: Surface Plot for Hidden Layer 1 Node 5 after 50 epochs

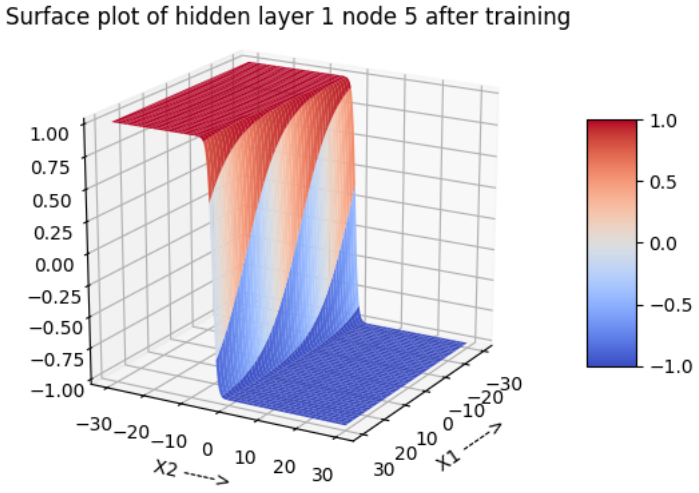


Figure 49: Surface Plot for Hidden Layer 1 Node 5 after completion of training

Hidden Layer 2 Node 1 Below are the surface plots for the Hidden Layer 2 Node 1 across various epochs.

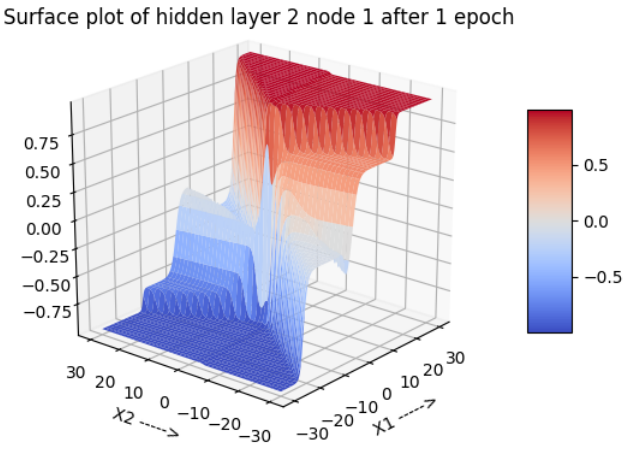


Figure 50: Surface Plot for Hidden Layer 2 Node 1 after 1 epoch

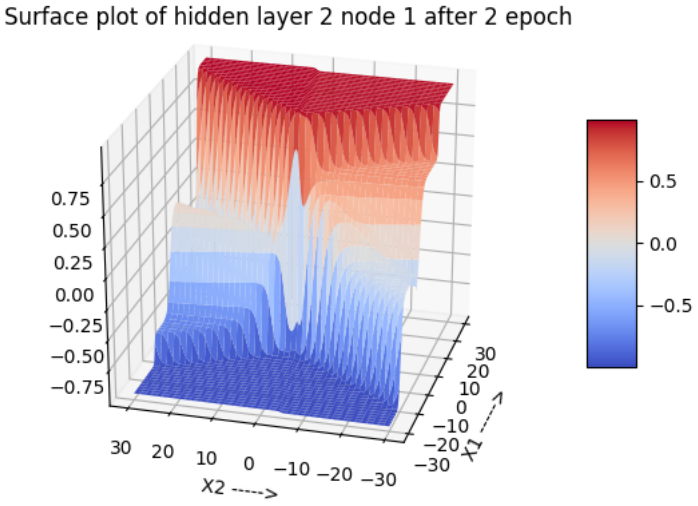


Figure 51: Surface Plot for Hidden Layer 2 Node 1 after 2 epochs

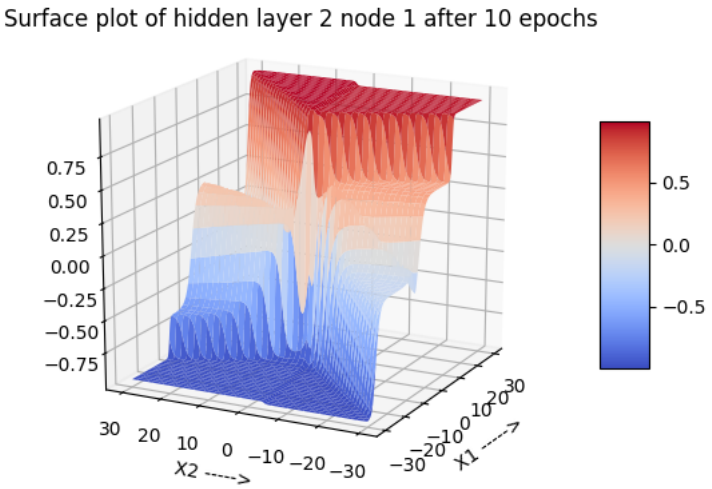


Figure 52: Surface Plot for Hidden Layer 2 Node 1 after 10 epochs

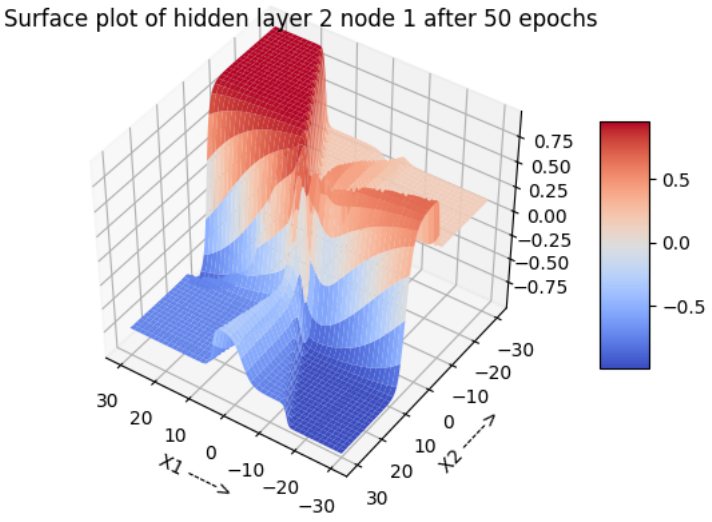


Figure 53: Surface Plot for Hidden Layer 2 Node 1 after 50 epochs

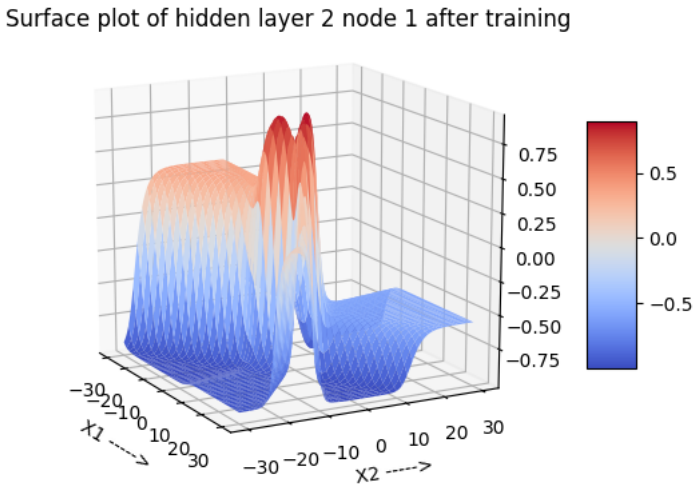


Figure 54: Surface Plot for Hidden Layer 2 Node 1 after completion of training

Hidden Layer 2 Node 2 Below are the surface plots for the Hidden Layer 2 Node 2 across various epochs.

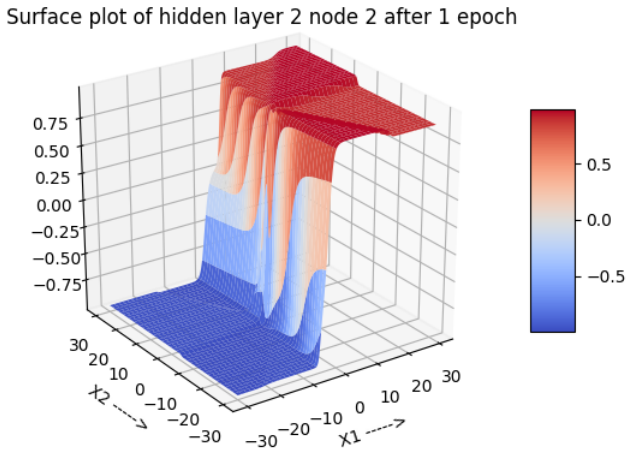


Figure 55: Surface Plot for Hidden Layer 2 Node 2 after 1 epoch

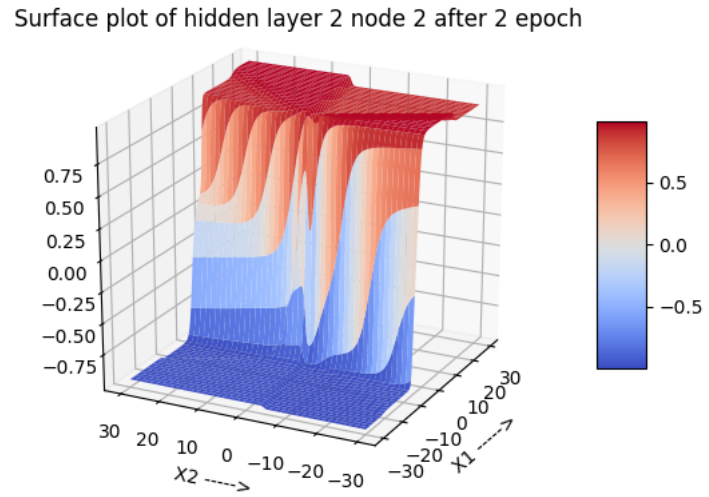


Figure 56: Surface Plot for Hidden Layer 2 Node 2 after 2 epochs

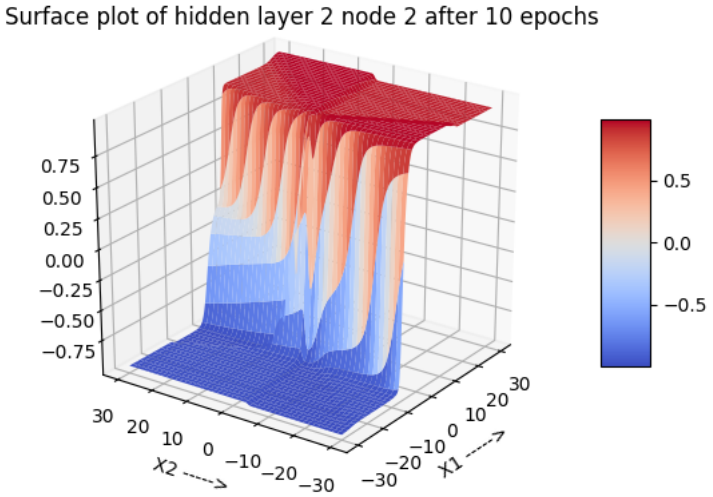


Figure 57: Surface Plot for Hidden Layer 2 Node 2 after 10 epochs

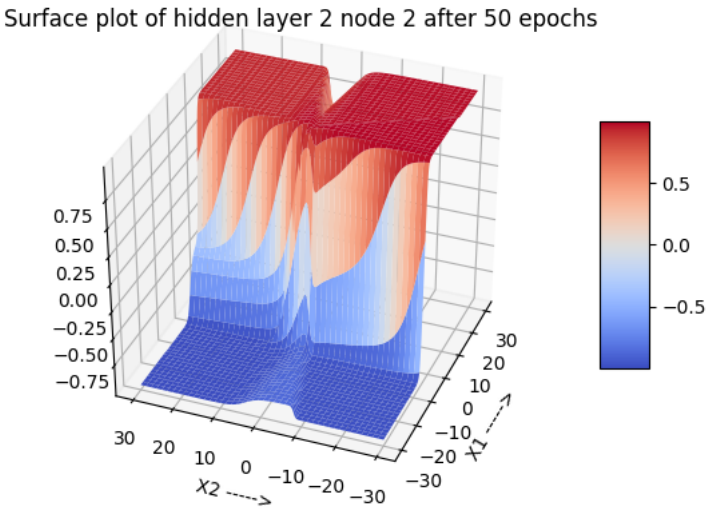


Figure 58: Surface Plot for Hidden Layer 2 Node 2 after 50 epochs

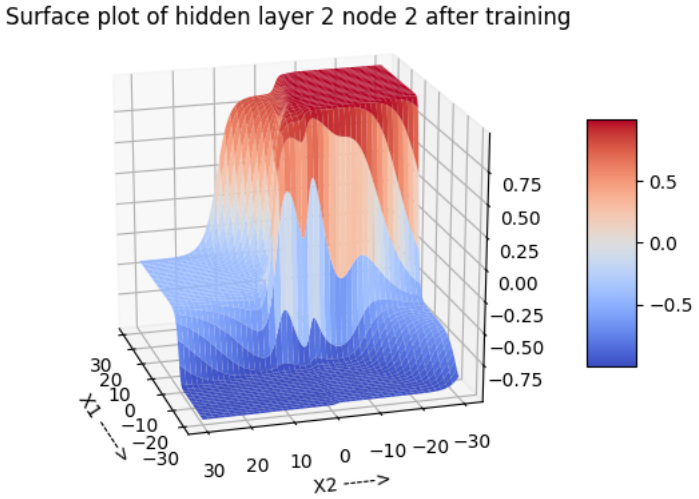


Figure 59: Surface Plot for Hidden Layer 2 Node 2 after completion of training

Hidden Layer 2 Node 3 Below are the surface plots for the Hidden Layer 2 Node 3 across various epochs.

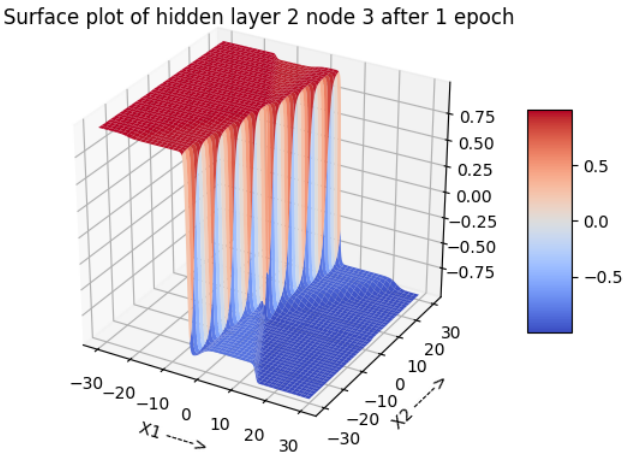


Figure 60: Surface Plot for Hidden Layer 2 Node 3 after 1 epoch

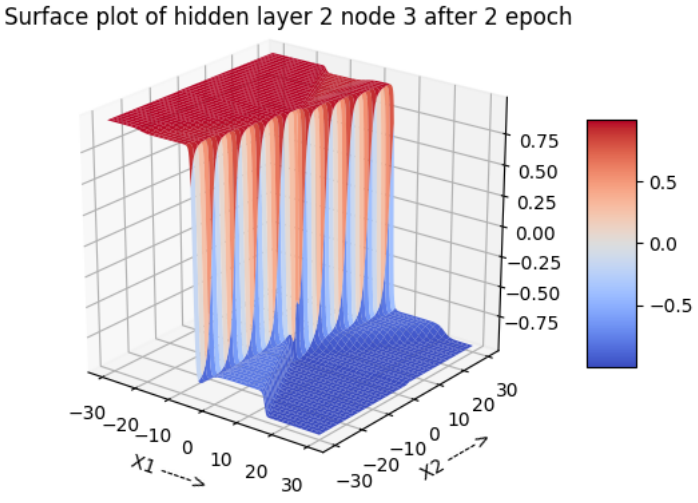


Figure 61: Surface Plot for Hidden Layer 2 Node 3 after 2 epochs

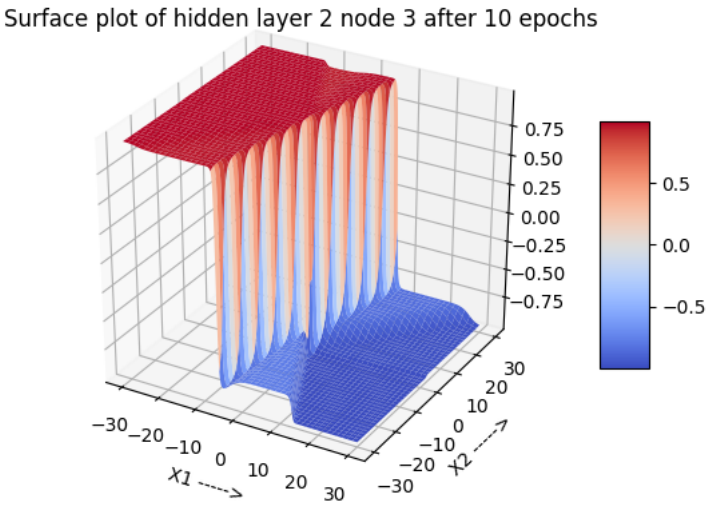


Figure 62: Surface Plot for Hidden Layer 2 Node 3 after 10 epochs

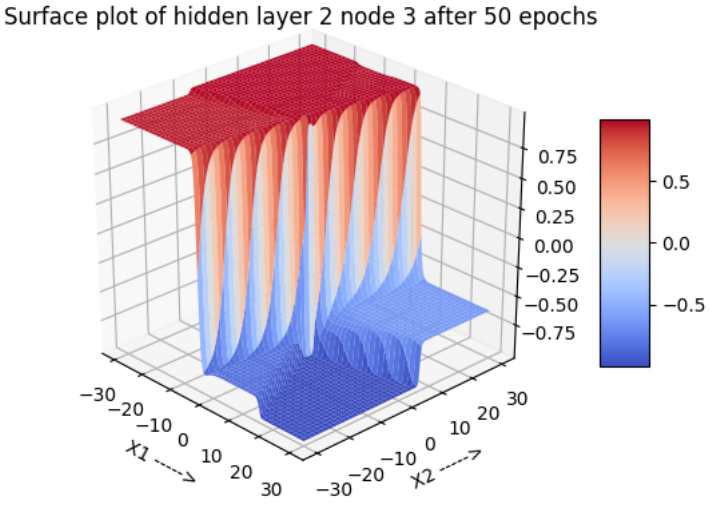


Figure 63: Surface Plot for Hidden Layer 2 Node 3 after 50 epochs

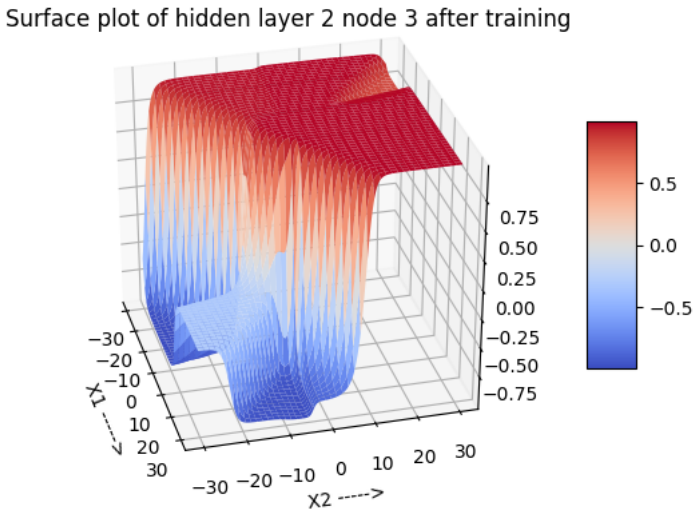


Figure 64: Surface Plot for Hidden Layer 2 Node 3 after completion of training

Hidden Layer 2 Node 4 Below are the surface plots for the Hidden Layer 2 Node 4 across various epochs.

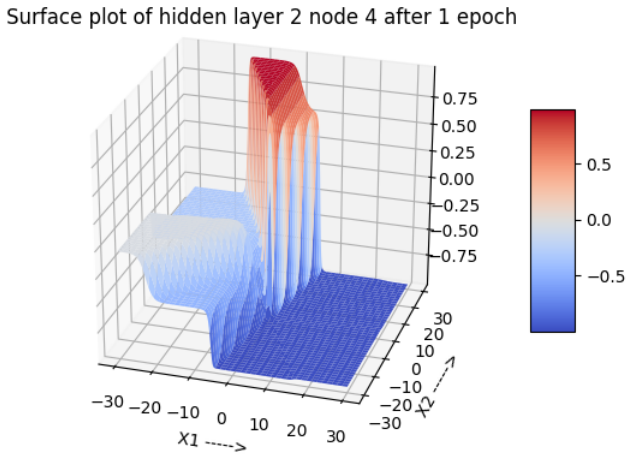


Figure 65: Surface Plot for Hidden Layer 2 Node 4 after 1 epoch

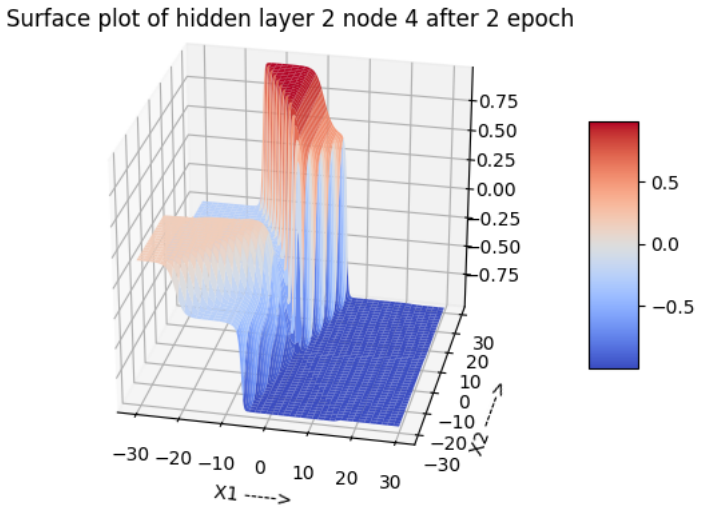


Figure 66: Surface Plot for Hidden Layer 2 Node 4 after 2 epochs

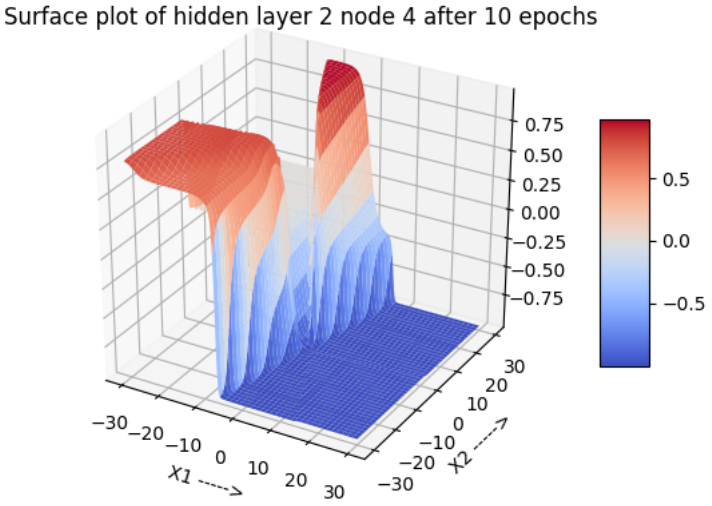


Figure 67: Surface Plot for Hidden Layer 2 Node 4 after 10 epochs

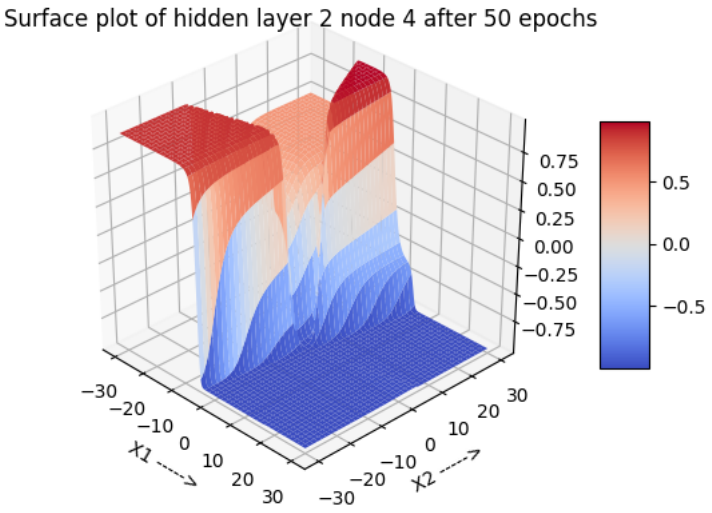


Figure 68: Surface Plot for Hidden Layer 2 Node 4 after 50 epochs

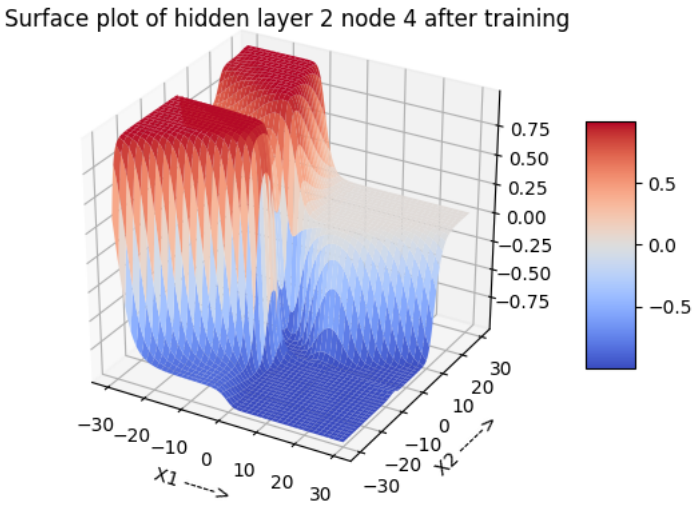


Figure 69: Surface Plot for Hidden Layer 2 Node 4 after completion of training

Hidden Layer 2 Node 5 Below are the surface plots for the Hidden Layer 2 Node 5 across various epochs.

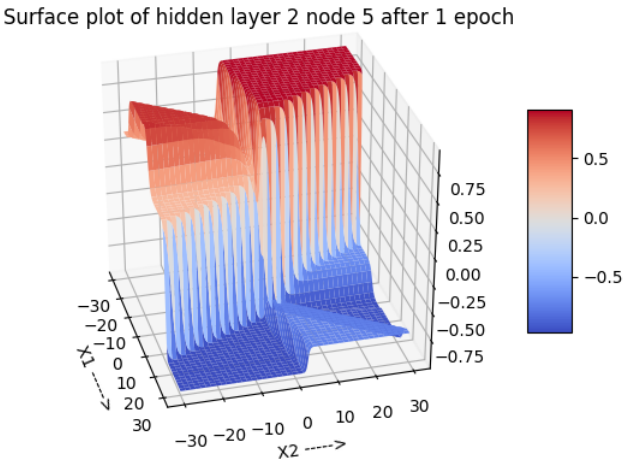


Figure 70: Surface Plot for Hidden Layer 2 Node 5 after 1 epoch

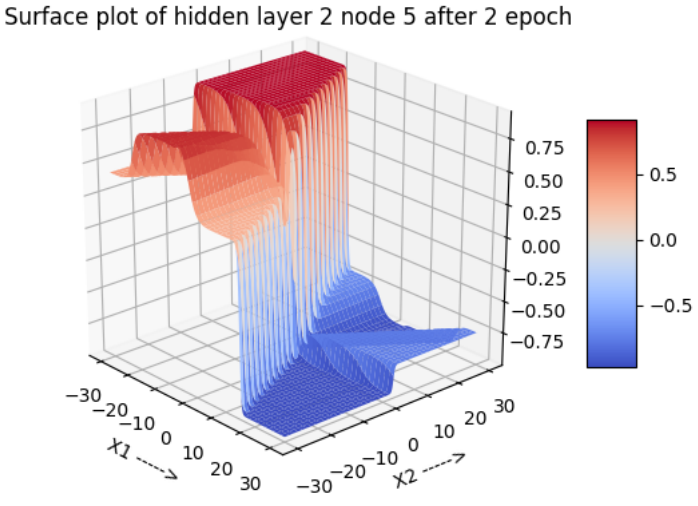


Figure 71: Surface Plot for Hidden Layer 2 Node 5 after 2 epochs

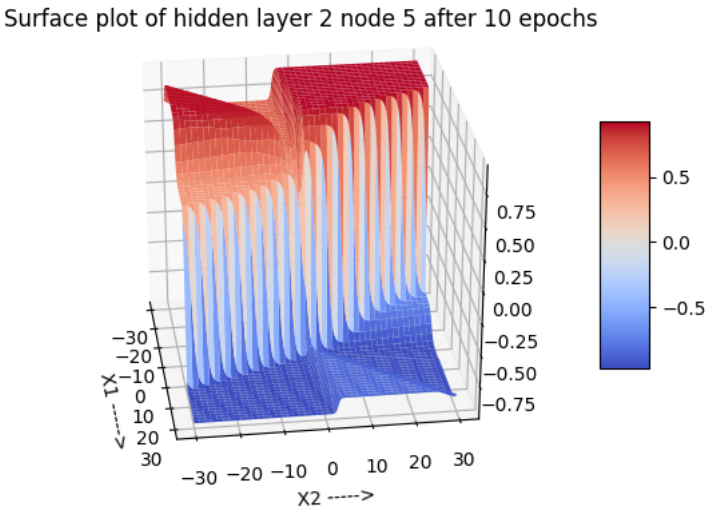


Figure 72: Surface Plot for Hidden Layer 2 Node 5 after 10 epochs

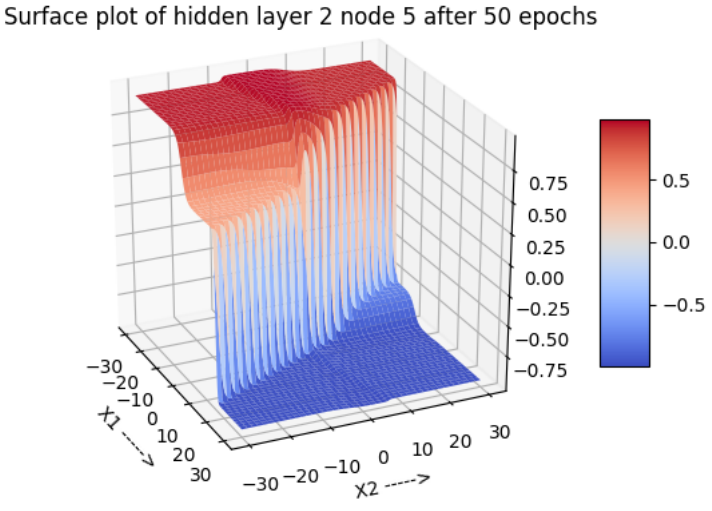


Figure 73: Surface Plot for Hidden Layer 2 Node 5 after 50 epochs

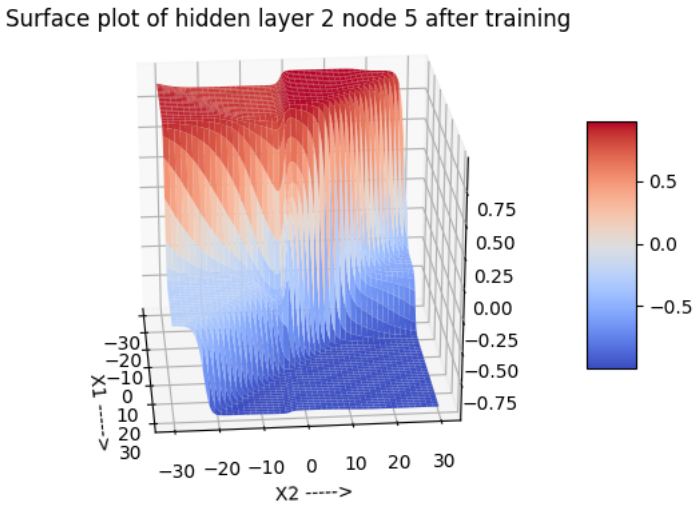


Figure 74: Surface Plot for Hidden Layer 2 Node 5 after completion of training

5.2.3 Decision Region

Firstly, let us represent the input data in terms of a scatter plot to get an idea of the distribution of the data.

Figure 75: Distribution of the entire data set

From the figure above, we can see that the 3 different classes form overlapping clusters which require non linear decision regions to separate.

Now, we plot the decision region of the input vector space. The plot below

shows the demarcated regions for the three classes which are representative of the data points plot above[Figure 73].

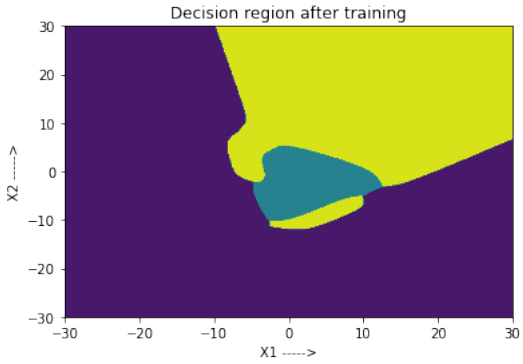


Figure 76: The decision region after the training of the model

5.3 Inference

- As the value of the learning rate is decreased, the average error curve becomes slightly more smooth and the abrupt changes in the values are absent. However, a very small value of the learning parameter increases the training time drastically. Hence, we would need to find an optimal learning rate, here found to be 1e-3.
- For very small values of the momentum, the method wasn't much different from the delta rule. However, as the value of the delta was increased, the average error curve became more smooth.
- The model overfit the data when the hidden layer nodes exceed a value of 5.

6 Classification task for image data

In this sub-part, we perform single-label multi-class classification of image data. The 5 classes which are included in the classification are deer, dog, plane, ship and truck.

Firstly, the original image dimensions were reduced to feature vectors post which, Principal Component Analysis (PCA) was performed on the feature vectors to obtain the most important features which accounted for majority of variance in the data. The code for the PCA has been included in the submission. PCA was import from sklearn's decomposition. Using pca.fit-transform, the feature extracted 1500*512 data was transformed to 1500*20,

a set of values of linearly uncorrelated variables (n-pca=20 was a hyper parameter that was experimented with). PCA applied on direct image pixels, output of feature extraction without PCA directly fed into the model etc. were tried out but despite the accuracy being higher in such cases, they either did not make analytical sense or resulted in over fitting because of huge model complexity (unavailability of 10*(NO-OF-PARAMETERS) is one major reason for the same which can be avoided using data augmentation).

Similar to the previous classification method, the outputs were one hot encoded prior to training. The output layer activation function was softmax to obtain the outputs as probabilities.

While comparing the convergence across various weight update rules, it was ensured that the weight initialisations were the same.

The values of the parameters which gave the best accuracy results for the image classification are as follows :

Learning rate: 1e-3
Momentum : 0.9
Number of nodes in input layer (after PCA) : 20
Number of nodes in hidden layer 1 : 10
Number of nodes in hidden layer 2 : 5
Number of nodes in output layer : 5
Activation function for hidden layers : tanh
Activation function for output layer : softmax
Weight update rule : delta rule, generalised delta rule, adam

6.1 Plots of average error values for different optimiser methods

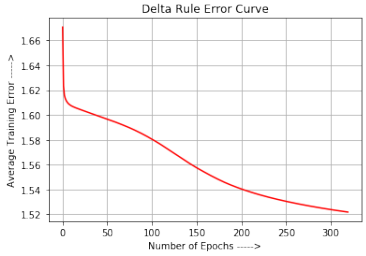
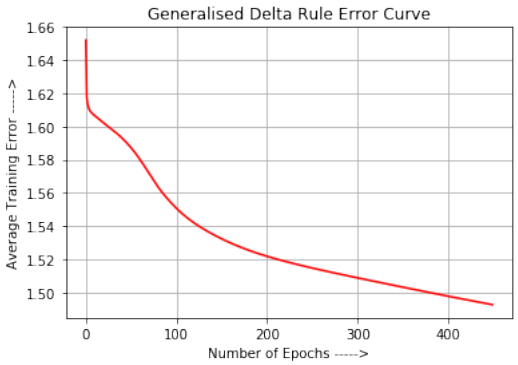


Figure 77: Average error plot for delta rule



From the plots above, we can observe that the delta rule has the least smooth descent followed by the generalised delta curve and the adam optimiser has extremely smooth descent which we prefer.

6.2 Confusion Matrices for weight update rules

The confusion matrices for different weight update rules (*delta*, *generalized delta* and *adam* rule) taking are tabulated below. Here, `deer` is assumed to be class 0, `dog` is class 1, `plane` is class 2, `ship` is class 3 and `truck` is class 4.

Adam Optimizer *Accuracy on validation set* : 0.28

CM	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4
Actual 0	25	14	16	11	9
Actual 1	12	22	11	17	13
Actual 2	22	12	21	9	11
Actual 3	13	15	11	23	13
Actual 4	9	16	17	18	15

Delta Rule *Accuracy on validation set* : 0.27

CM	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4
Actual 0	21	14	12	17	11
Actual 1	16	19	15	9	116
Actual 2	14	12	25	9	19
Actual 3	14	15	12	23	11
Actual 4	4	16	28	13	14

Generalised Delta Rule *Accuracy on validation set* : 0.24

CM	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4
Actual 0	13	19	7	24	12
Actual 1	12	20	17	17	9
Actual 2	10	20	24	5	16
Actual 3	12	24	8	21	10
Actual 4	4	24	26	9	12

6.3 Inference

- Adam Optimiser has a better performance in the sense that the descent of the average error is much smoother over the other methods such as generalised delta and delta function.
- The accuracy on the validation set is low could be attributed to the fact the number of parameters to be estimated are proportional to the number of training set examples which makes it extremely hard for the model to learn from the limited dataset.