CS 6910 Fundamentals of Deep Learning

Assignment 1 Analysis Report

March 3, 2020

Report by **Team 19**

ME17B084 Nishant Prabhu, **ME17B068** Soumyadeep Mondal, **ME16B177** Saye Sharan

---

Contents

---

1 Submission Overview

The functionalities offered by each script in `Team19_Sub.zip` has been summarized below.

File	Description	Contents
`models.py`	Defines the `Network` class with feedforward, backpropagation, training and prediction functions among few others.	`<class> Network`
`layers.py`	Defines model layers as classes with necessary attributes (used by optimizers).	`<class> Input` `<class> Dense`

`activations.py`	Defines activation functions for layers as classes with methods to compute activation values and derivatives.	`<class> Linear`
		`<class> Sigmoid`
		`<class> Tanh`
		`<class> ReLU`
		`<class> Softmax`

`losses.py`	Defines loss functions as classes with methods to compute their values and derivatives.	`<class> RMSE` `<class> CrossEntropy`

`metrics.py`	Defines metrics to monitor model's training progress at the end of each epoch.	`<func> RMSE`
		`<func> CrossEntropy`
		`<func> Accuracy`

`optimizers.py`	Defines optimizers responsible for better model convergence. Typically uses local gradients or previous weight updates at every layer.	`<class> SGD`
		`<class> AdaGrad`
		`<class> AdaDelta`
		`<class> Adam`

`plotting.py`	Defines functions to plot distributions of data points, decision surfaces or model performance trends.	`<func> SpatialPlot`
		`<func> ParityPlot`
		`<func> CallbackPlot`
		`<func> DecisionRegionPlot`
		`<func> ConfusionMatrix`
		`<func> LayerOutputPlot`

`utils.py`	Defines helper functions used for data preprocessing.	`<class> OneHotEncoder`
		`<class> MinMaxScaler`
		`<class> ImFeatureExtractor`

2 Function Approximation

Data with 2 features related through a non-linear output function was provided for our model to learn. In the following sections, we have described our preprocessing steps, model description and model learning outcomes.

2.1 Data Preprocessing

No preprocessing was performed on the data. We avoided scaling it to allow direct usage of new data while testing model performance (which could potentially lie outside the range used for scaling training data).

2.2 Model Description

Epochs	`30,000`
Training mode	Pattern
Loss function	`RMSE`
Performance Metric	`RMSE`
Optimizer	`Generalized Delta Rule (SGD)` `(eta = 2e-06, momentum = 0.9)`

```
layers = [
    Input(units = 2, label = 'Input'),
    Dense(units = 8, activation = Sigmoid(), label = 'Hidden_1'),
    Dense(units = 6, activation = Sigmoid(), label = 'Hidden_2'),
    Dense(units = 1, activation = Linear(), label = 'Output')
]
```

Here are the results after running the model.

Training data size	`(100, 2)`
Validation data size	`(300, 2)`
Final average training RMSE	`30.130948`
Final average validation RMSE	`95.496457`

2.3 Average error (RMSE) over epochs

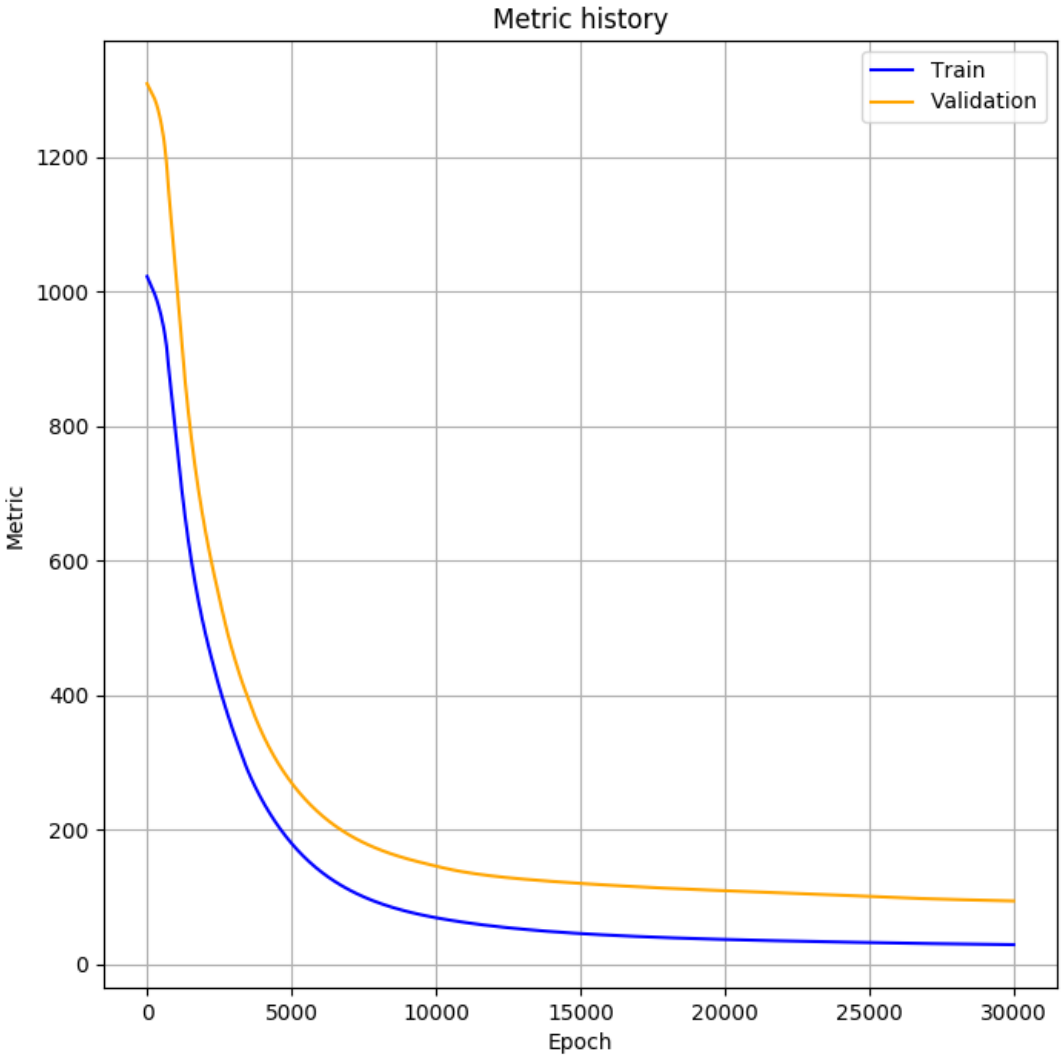


Figure 1: Trend of average RMSE over epochs

Both training and validation errors reduce smoothly, with the validation error always being slightly higher than training error. We train for 30,000 epochs since the model doesn't always reach errors as low as these due to random weight initializations.

2.4 Prediction Scatter

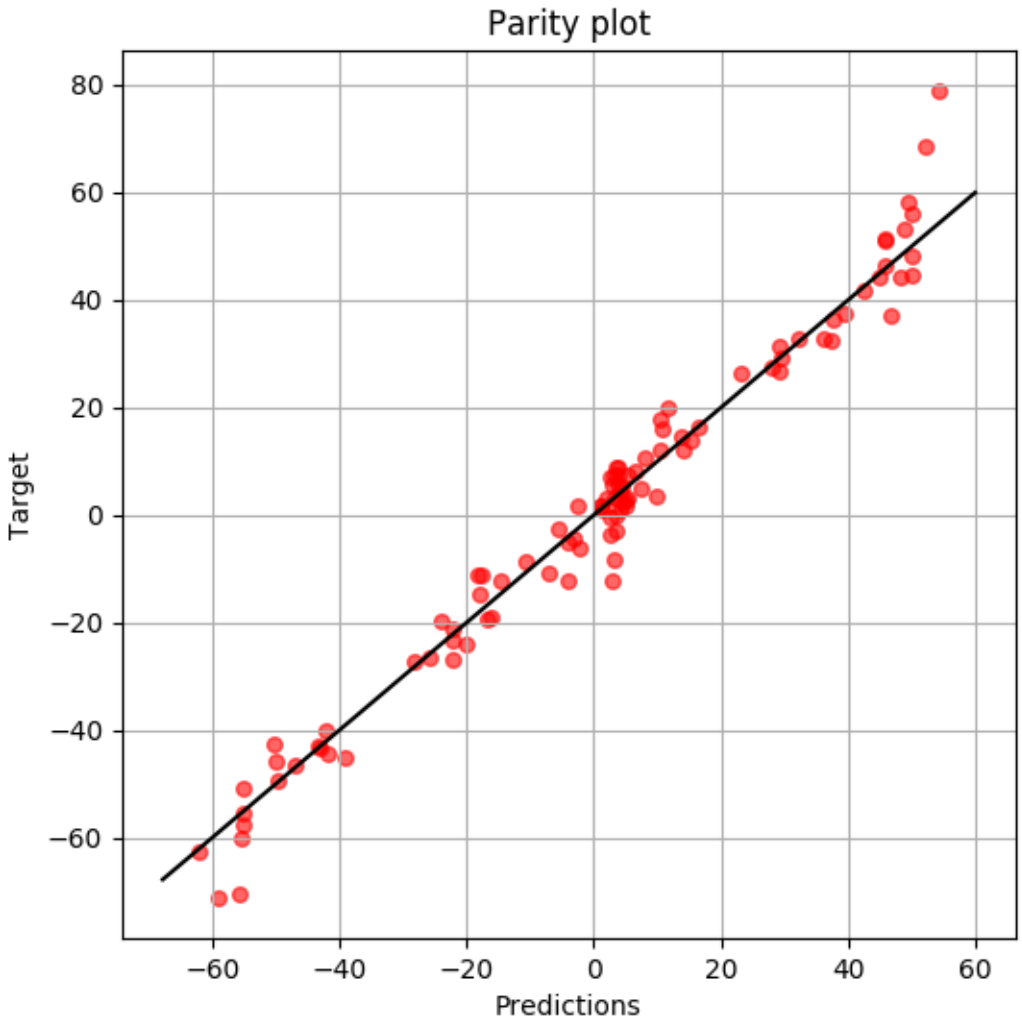


Figure 2: Scatter of actual output and predictions

Most of our predictions are close to actual values. Data with small feature values have been slightly underestimated while data with large feature values have been slightly overestimated.

2.5 Actual Data vs. Prediction Distribution

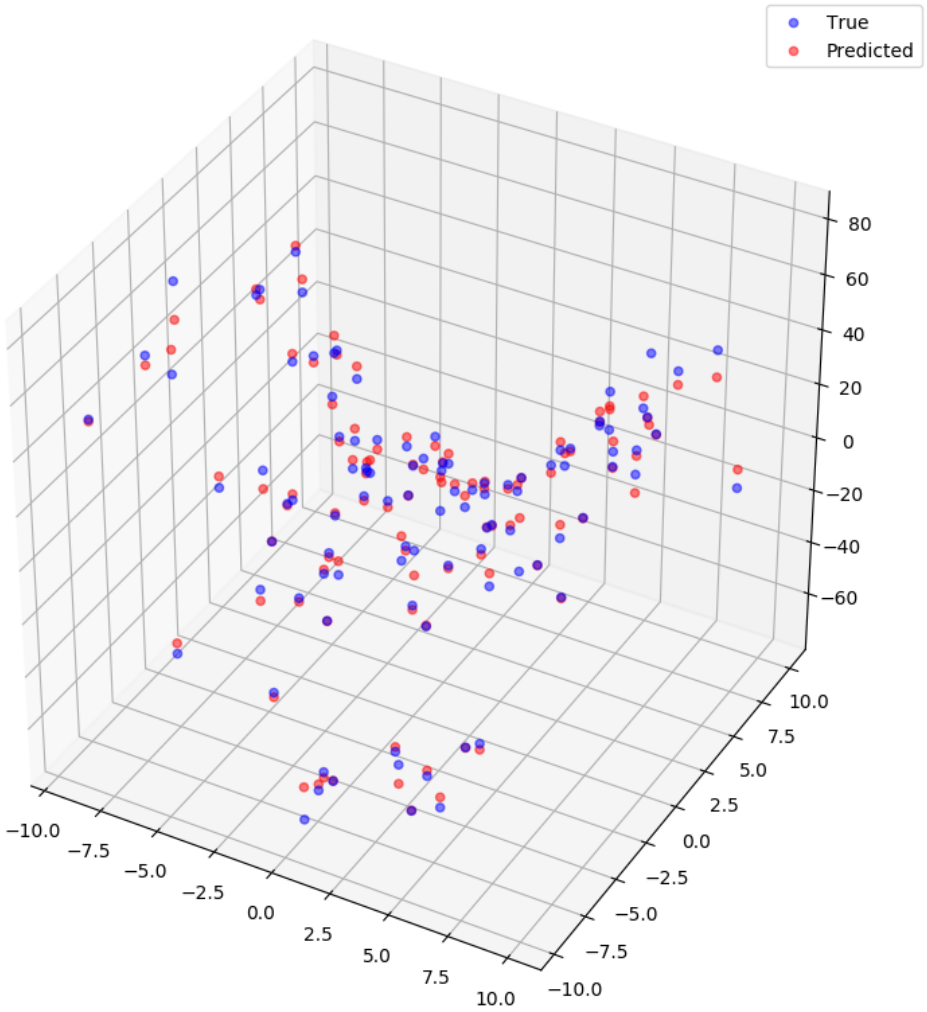
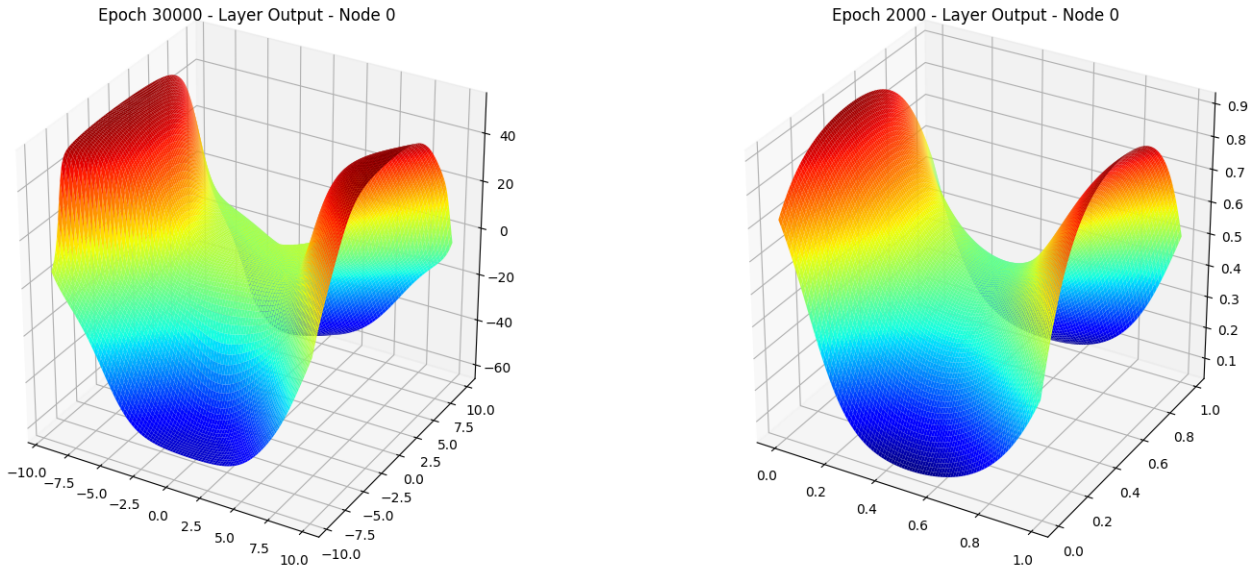


Figure 3: Distribution of actual data (blue) vs. predictions (red)

2.6 Approximated Function Surface



(a) Prediction surface for unscaled data

(b) Prediction surface for scaled data

Figure 4: Approximated function surface: Scaled vs. unscaled data

The approximated surface looks similar to the expected surface, of a saddle function. We also performed the same task with data scaled between 0 and 1. In that case, the approximated function resembles the actual function much better, as shown in the plot on the right. Scaled data also need much fewer epochs (2000) to converge reasonably, as compared to unscaled data.

3 2D Nonlinear Classification

Data with 2 features belonging to one of three linearly inseparable classes was provided for our model to learn. In the following sections, we have described our preprocessing steps, model description and model learning outcomes.

3.1 Data Preprocessing

Features (in both training and validation datasets) were scaled to lie between 0 and 1 using `MinMaxScaler`. Target was one-hot encoded using `OneHotEncoder` from `utils.py`.

We tried training the model with unscaled features, in which case it was struggling to reach above 35% accuracy. This could have happened due to features already lying in the saturation zones of the hidden activation functions (sigmoid).

3.2 Model Description

Epochs	`200`
Training mode	Pattern
Loss function	`CrossEntropy`
Performance Metric	`Accuracy`
Optimizer	`Generalized Delta Rule (SGD)` `(eta = 0.01, momentum = 0.9)`

```
layers = [
    Input(units = 2, label = 'Input'),
    Dense(units = 5, activation = Sigmoid(), label = 'Hidden_1'),
    Dense(units = 5, activation = Sigmoid(), label = 'Hidden_2'),
    Dense(units = 3, activation = Softmax(), label = 'Output')
]
```

Here are the results after running the model.

Training data size	`(316, 2)`
Validation data size	`(135, 2)`
Final average training CE loss	`0.006162`
Final average validation CE loss	`0.006253`
Final average training Accuracy (ratio)	`1.000000`
Final average validation Accuracy (ratio)	`1.000000`

3.3 Decision Regions

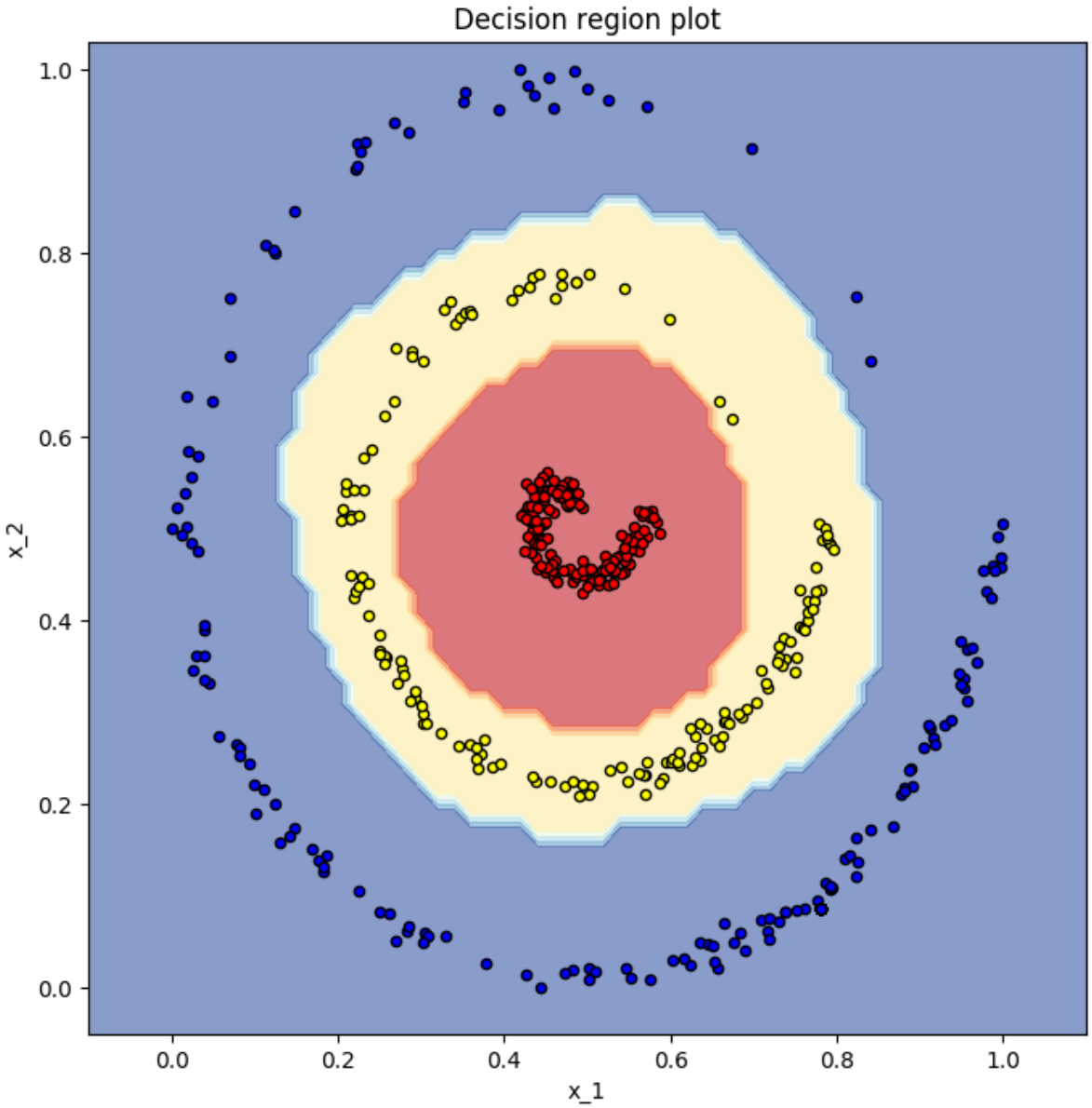
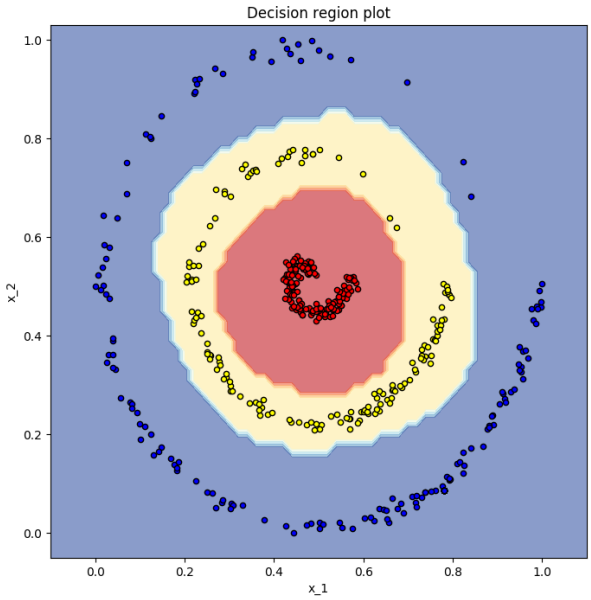
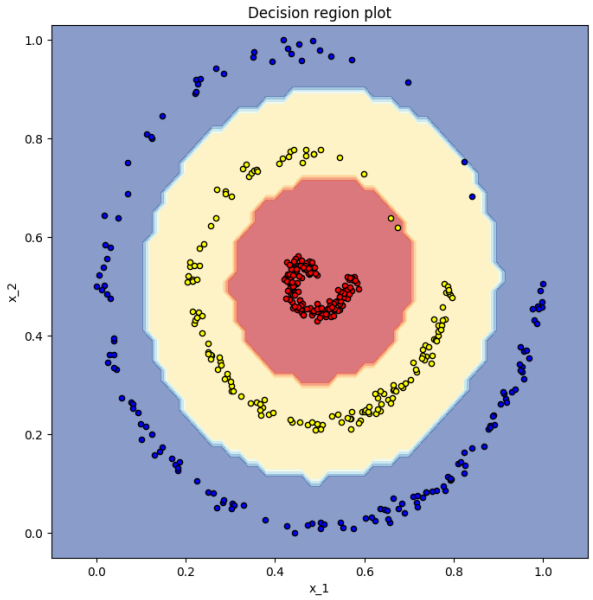


Figure 5: Decision regions learnt by MLFFNN classifier

There are some observations worth noting for this visualization, as explained on the next page.



(a) Decision regions learnt by MLFFNN



(b) Decision regions learnt by ν-SVC

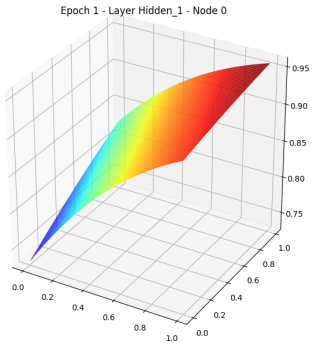
Figure 6: Decision region comparison

Since the classes in our data are well separated, the MLFFNN model (left) has perfectly characterized all decision boundaries. Our model was able to achieve 100% accuracy on the data (training and validation). However, the points lying in the yellow region have been confined to an excessively narrow region, which could sometimes cause misclassifications.

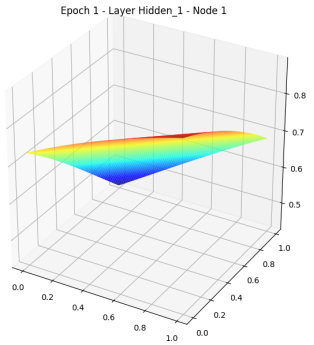
For comparison, on the right we have decision boundaries learnt by a Support Vector Classifier (NuSVC, `scikit-learn`) with radial basis function kernel. Although it has less than 100% accuracy (4 misclassifications), the boundaries are much more conservative.

3.4 Layer Output Surface Plots

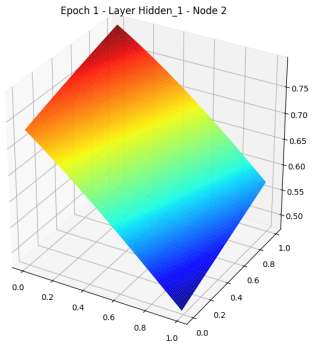
Epoch 1



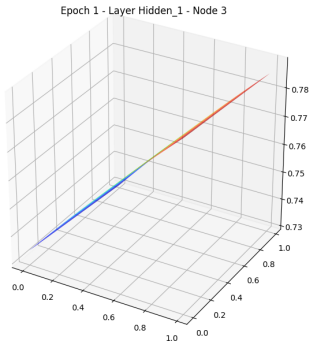
(a) Hidden layer 1, Node 0



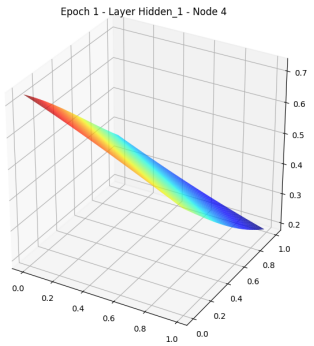
(b) Hidden layer 1, Node 1



(c) Hidden layer 1, Node 2

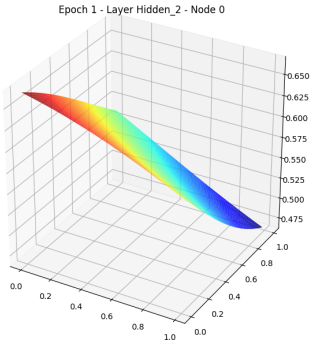


(d) Hidden layer 1, Node 3

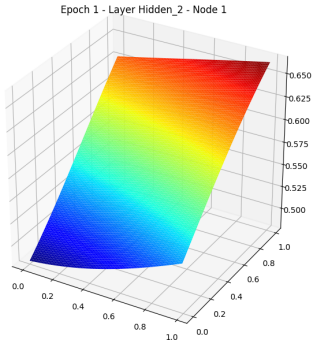


(e) Hidden layer 1, Node 4

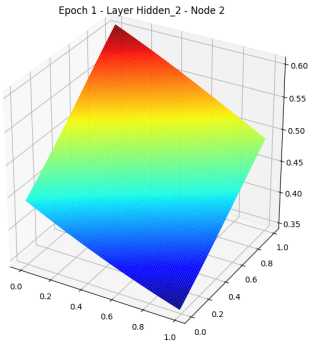
Figure 7: Epoch 1, Hidden layer 1



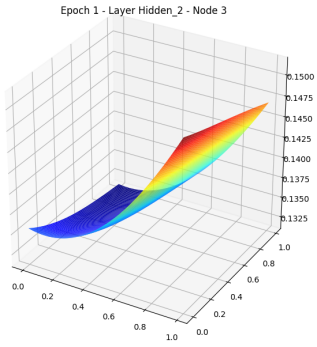
(a) Hidden layer 2, Node 0



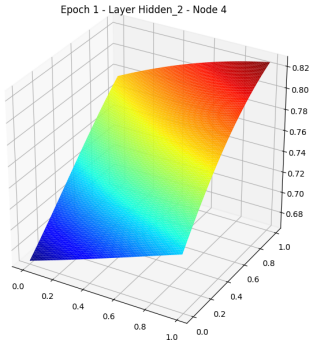
(b) Hidden layer 2, Node 1



(c) Hidden layer 2, Node 2

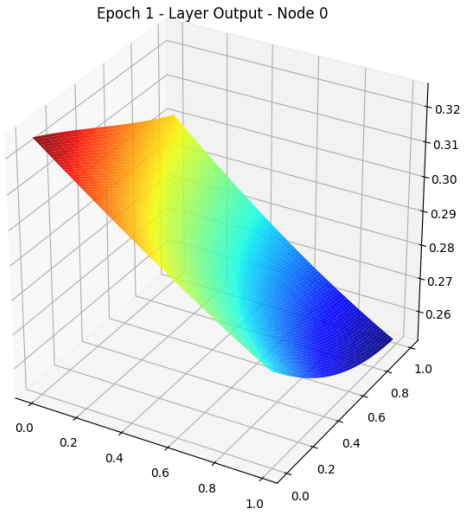


(d) Hidden layer 2, Node 3

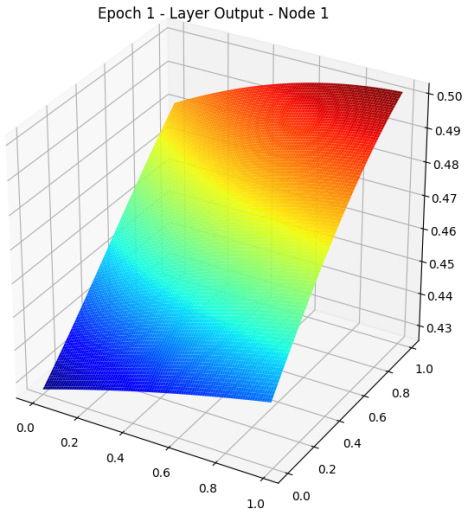


(e) Hidden layer 2, Node 4

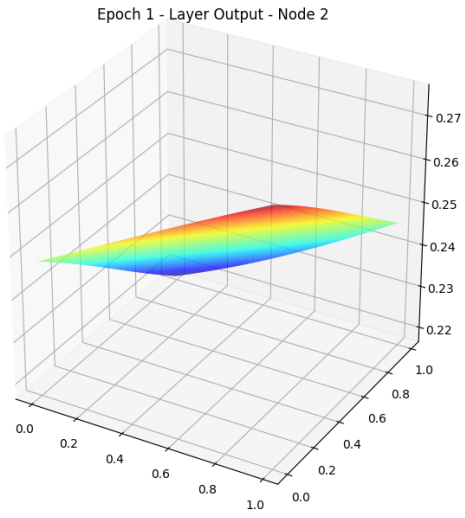
Figure 8: Epoch 1, Hidden layer 2



(a) Output layer, Node 0



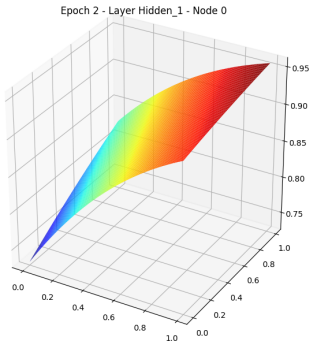
(b) Output layer, Node 1



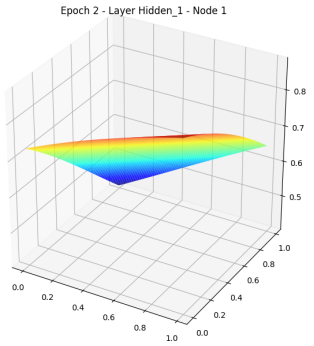
(c) Output layer, Node 2

Figure 9: Epoch 1, Output layer

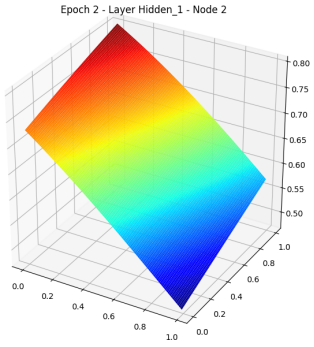
Epoch 2



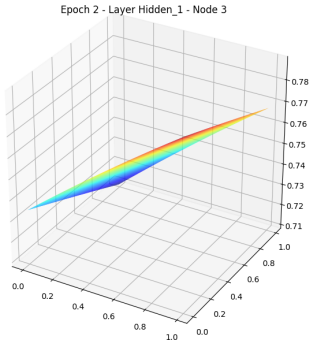
(a) Hidden layer 1, Node 0



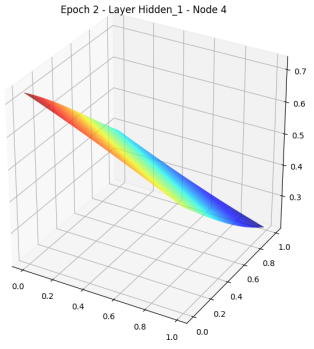
(b) Hidden layer 1, Node 1



(c) Hidden layer 1, Node 2

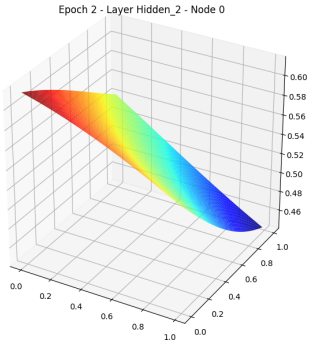


(d) Hidden layer 1, Node 3

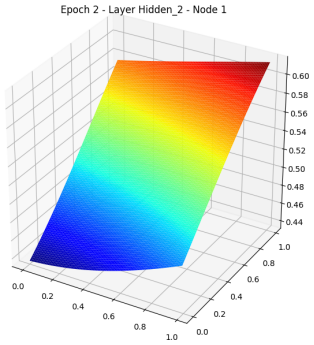


(e) Hidden layer 1, Node 4

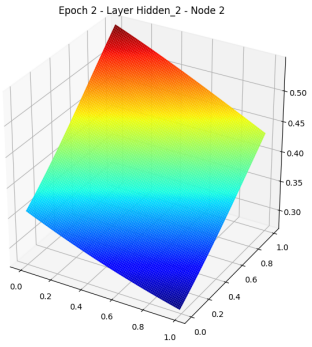
Figure 10: Epoch 2, Hidden layer 1



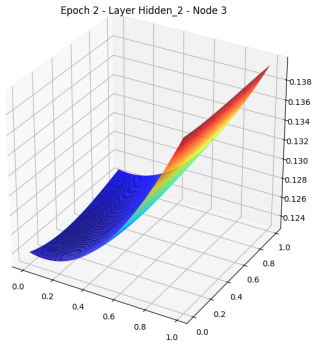
(a) Hidden layer 2, Node 0



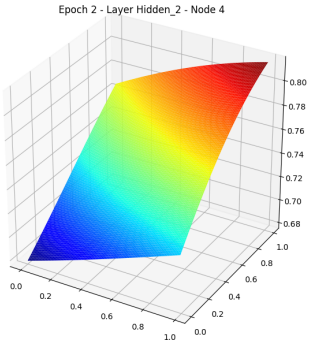
(b) Hidden layer 2, Node 1



(c) Hidden layer 2, Node 2

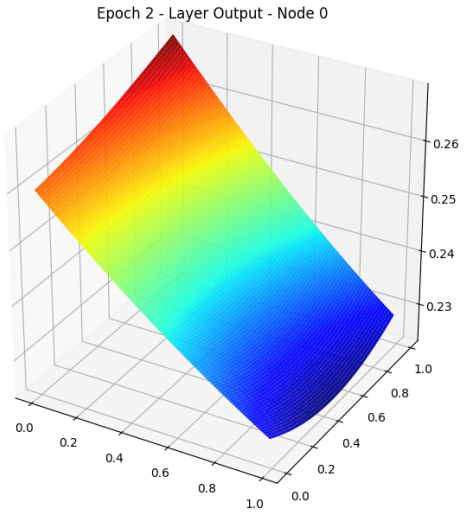


(d) Hidden layer 2, Node 3

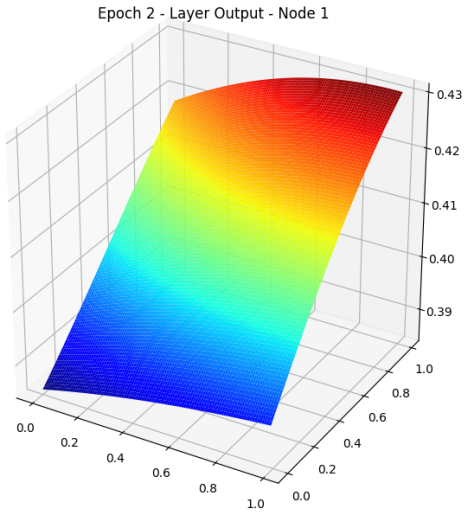


(e) Hidden layer 2, Node 4

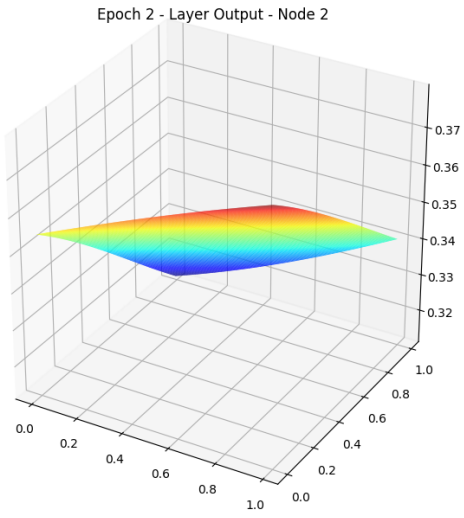
Figure 11: Epoch 2, Hidden layer 2



(a) Output layer, Node 0



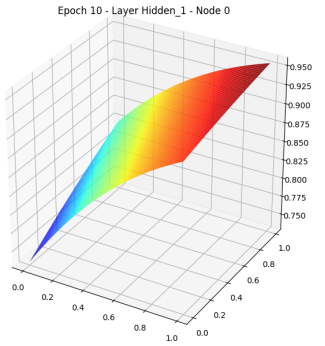
(b) Output layer, Node 1



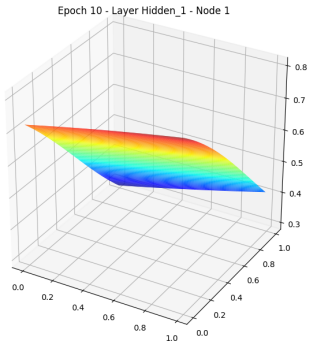
(c) Output layer, Node 2

Figure 12: Epoch 2, Output layer

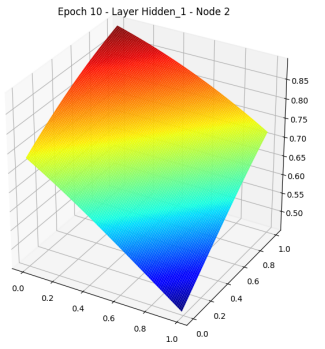
Epoch 10



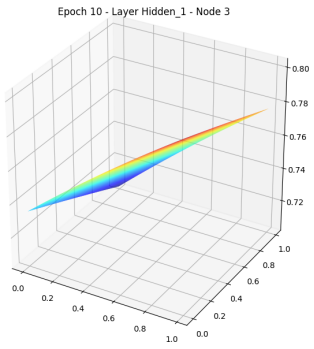
(a) Hidden layer 1, Node 0



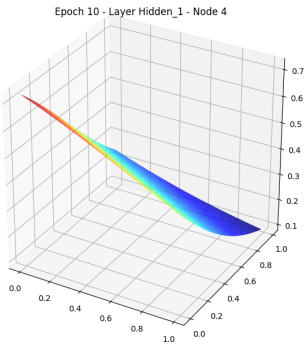
(b) Hidden layer 1, Node 1



(c) Hidden layer 1, Node 2

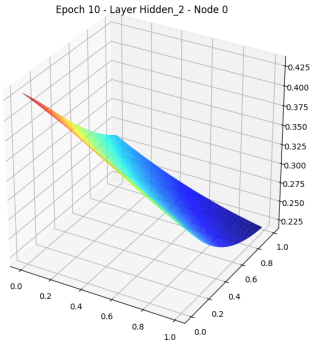


(d) Hidden layer 1, Node 3

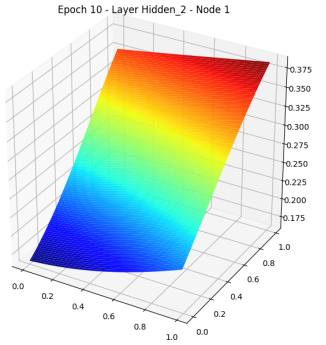


(e) Hidden layer 1, Node 4

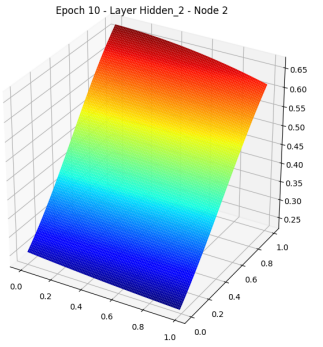
Figure 13: Epoch 10, Hidden layer 1



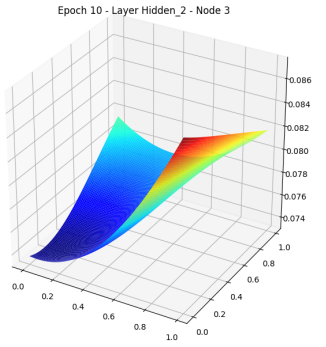
(a) Hidden layer 2, Node 0



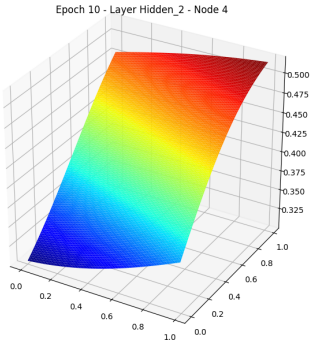
(b) Hidden layer 2, Node 1



(c) Hidden layer 2, Node 2

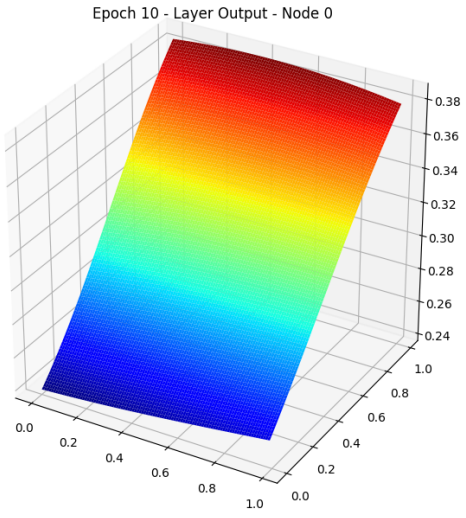


(d) Hidden layer 2, Node 3

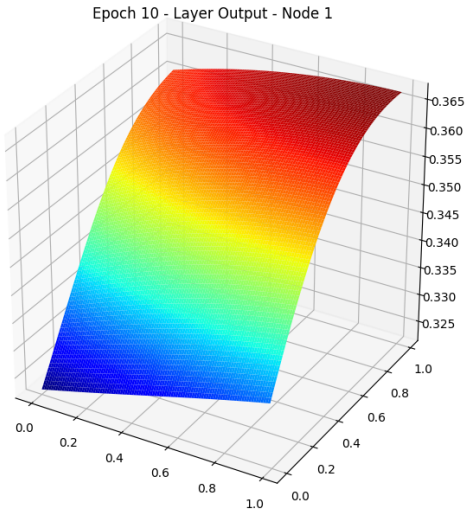


(e) Hidden layer 2, Node 4

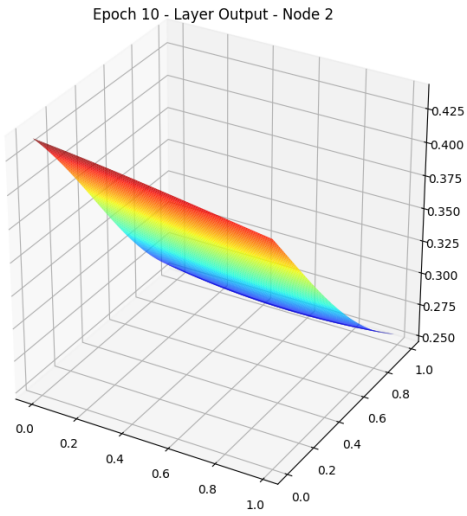
Figure 14: Epoch 10, Hidden layer 2



(a) Output layer, Node 0



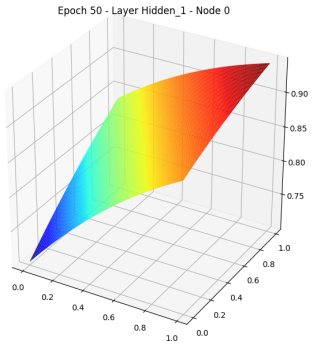
(b) Output layer, Node 1



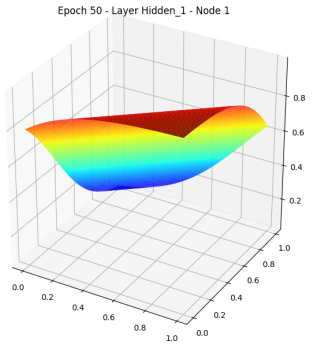
(c) Output layer, Node 2

Figure 15: Epoch 10, Output layer

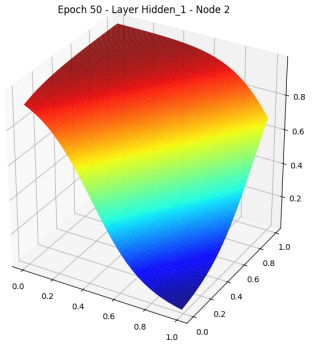
Epoch 50



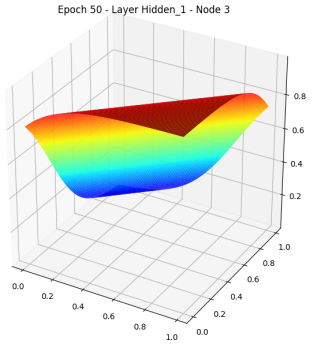
(a) Hidden layer 1, Node 0



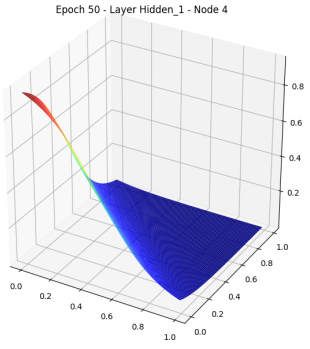
(b) Hidden layer 1, Node 1



(c) Hidden layer 1, Node 2

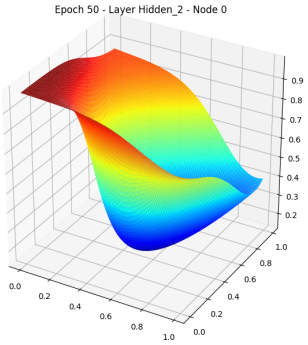


(d) Hidden layer 1, Node 3

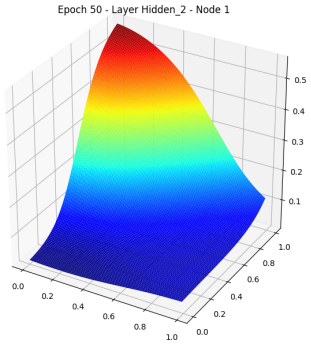


(e) Hidden layer 1, Node 4

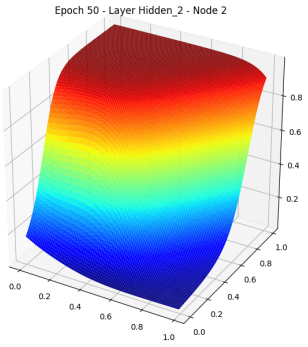
Figure 16: Epoch 50, Hidden layer 1



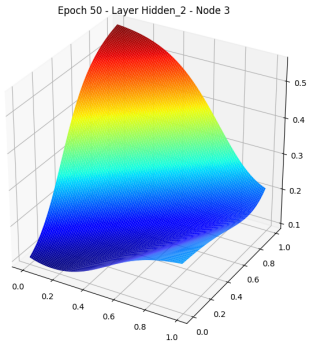
(a) Hidden layer 2, Node 0



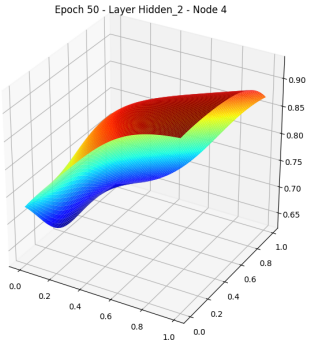
(b) Hidden layer 2, Node 1



(c) Hidden layer 2, Node 2

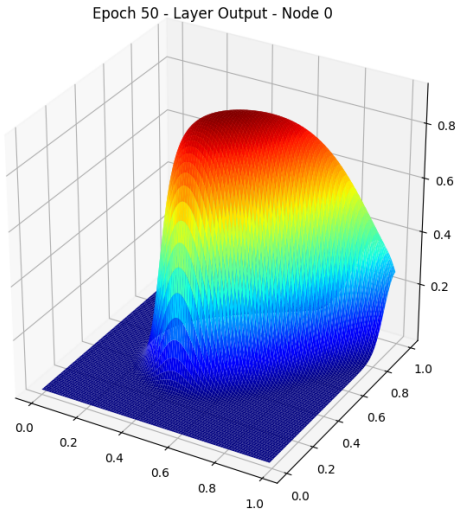


(d) Hidden layer 2, Node 3

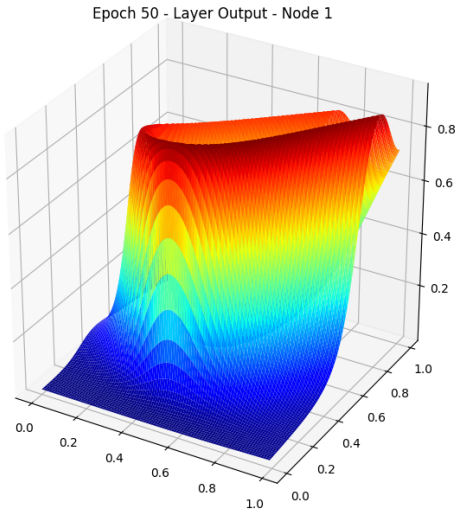


(e) Hidden layer 2, Node 4

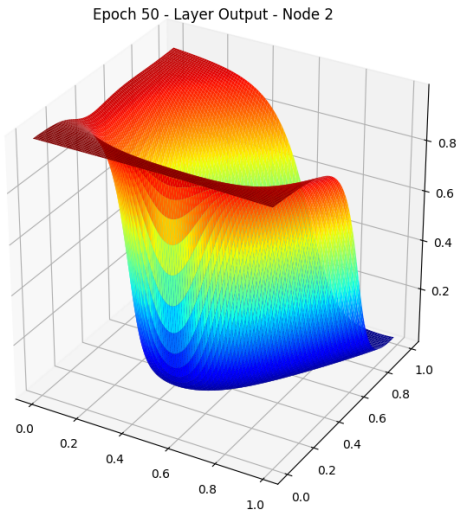
Figure 17: Epoch 50, Hidden layer 2



(a) Output layer, Node 0



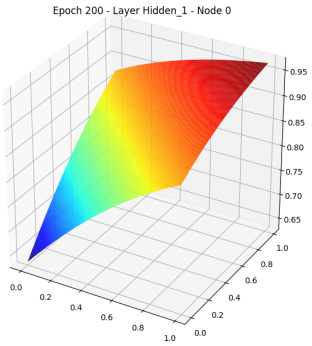
(b) Output layer, Node 1



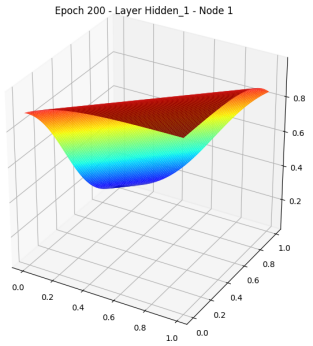
(c) Output layer, Node 2

Figure 18: Epoch 50, Output layer

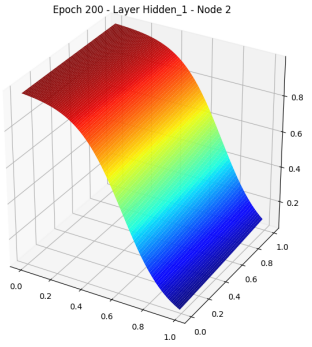
Epoch 200 (End of training)



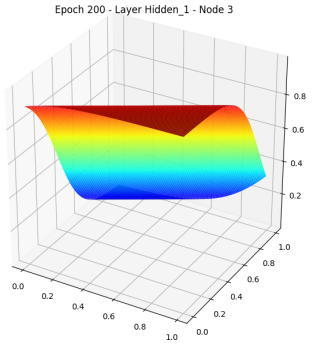
(a) Hidden layer 1, Node 0



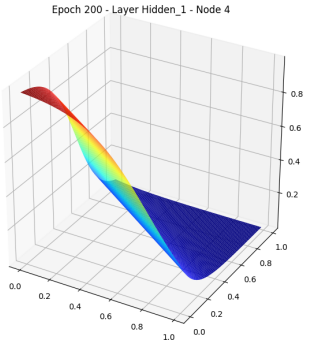
(b) Hidden layer 1, Node 1



(c) Hidden layer 1, Node 2

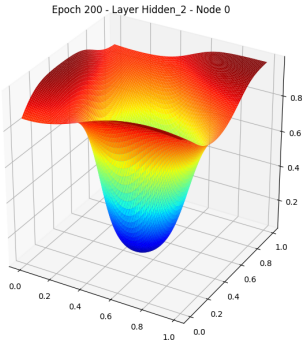


(d) Hidden layer 1, Node 3

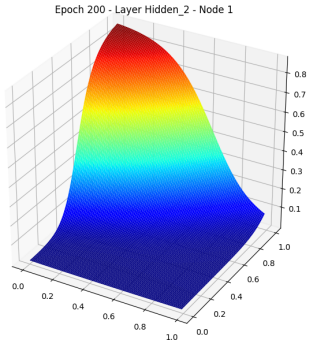


(e) Hidden layer 1, Node 4

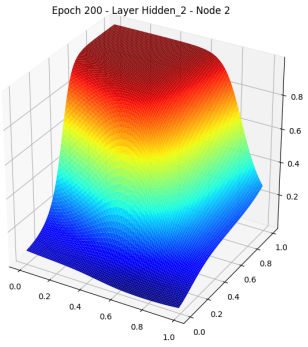
Figure 19: Epoch 200, Hidden layer 1



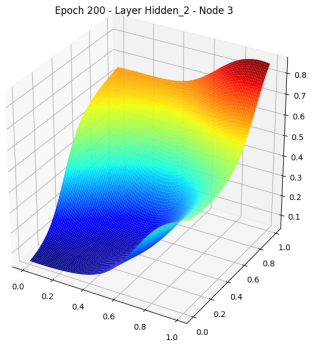
(a) Hidden layer 2, Node 0



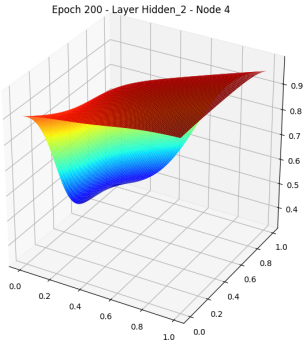
(b) Hidden layer 2, Node 1



(c) Hidden layer 2, Node 2

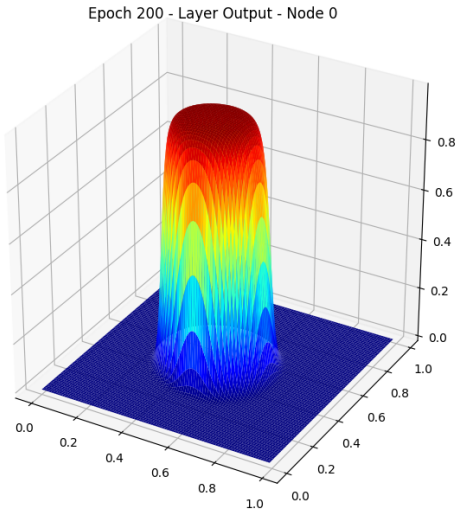


(d) Hidden layer 2, Node 3

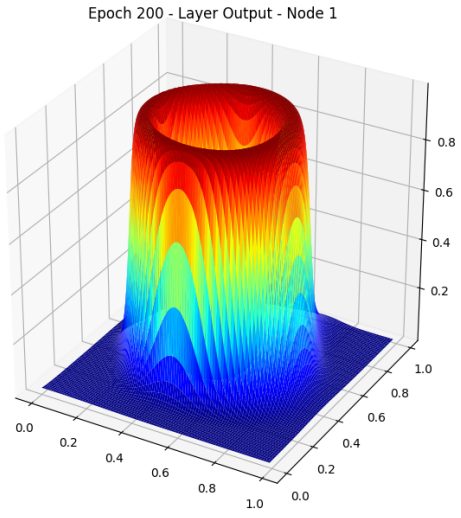


(e) Hidden layer 2, Node 4

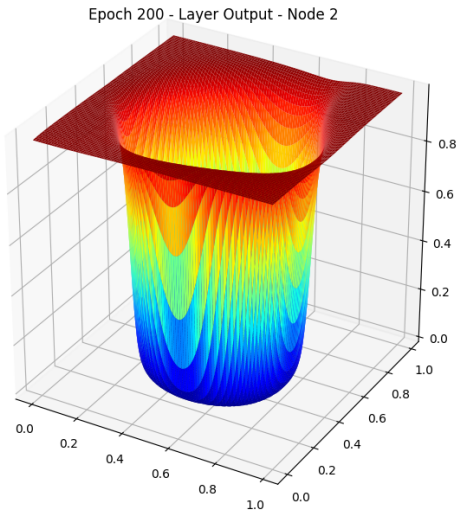
Figure 20: Epoch 200, Hidden layer 2



(a) Output layer, Node 0



(b) Output layer, Node 1



(c) Output layer, Node 2

Figure 21: Epoch 200, Output layer

Typically, our network reached 100% accuracy at the end of 100 epochs. It was trained until 200 epochs to account for bad weight initializations, hence giving enough opportunity to converge to the local minimum.

4 Image Classification

Image data for 5 classes (300 images per class) was provided in the form of pixel intensities encoded in `npy` files. Scripts to extract features (by passing them through convolutional layers) were provided. After running the scripts on the provided files, 512 features for each image were obtained which were used for training. Weights initialization and learning rates were same for all tests. In the following sections, we have described our preprocessing steps, model description and model learning outcomes. Inferences have been stated at the end of the section.

4.1 Data Preprocessing

No preprocessing was performed on the features. Target was one-hot encoded using `OneHotEncoder` from `utils.py`.

4.2 Model Description

Epochs	`500`
Training mode	Pattern
Loss function	`CrossEntropy`
Performance Metric	`Accuracy`
Optimizer 1	`Delta Rule` `(eta = 0.001)`
Optimizer 2	`Generalized Delta Rule (SGD)` `(eta = 0.001, momentum = 0.9)`
Optimizer 3	`Adam` `(eta = 0.001, rho_1 = 0.9, rho_2 = 0.999)`

```
layers = [
    Input(units = 512, label = 'Input'),
    Dense(units = 64, activation = Sigmoid(), label = 'Hidden_1'),
    Dense(units = 64, activation = Sigmoid(), label = 'Hidden_2'),
    Dense(units = 5, activation = Softmax(), label = 'Output')
]
```

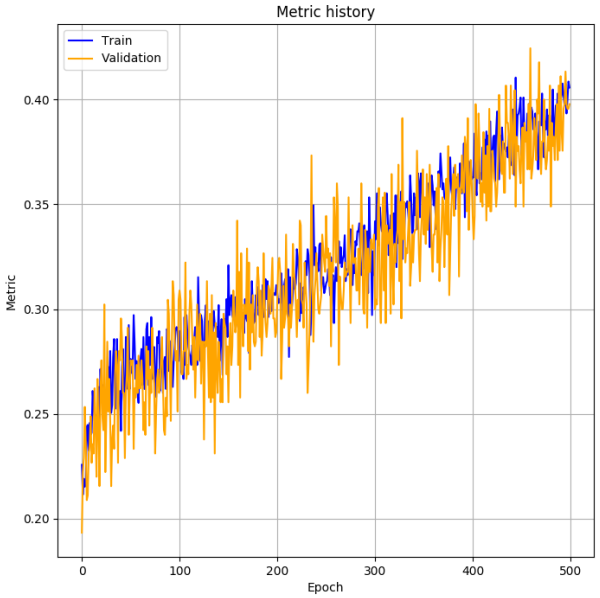
Here are the results after running the model.

	Training	**Validation**
Final Delta Rule Accuracy (ratio)	`0.411111`	`0.403333`
Final SGD Accuracy (ratio)	`0.999048`	`1.000000`
Final Adam Accuracy (ratio)	`1.000000`	`1.000000`

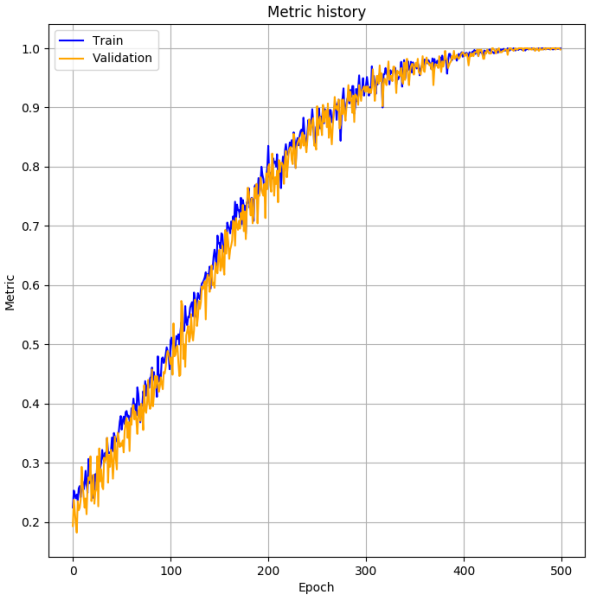
4.3 Performance Measures

Accuracy Trends

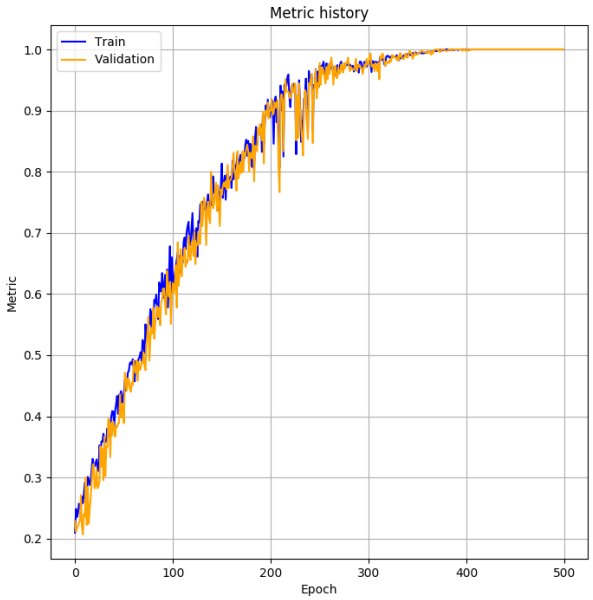
Indirectly, this tells us the error rate of each optimizer over epochs.



(a) Delta Rule



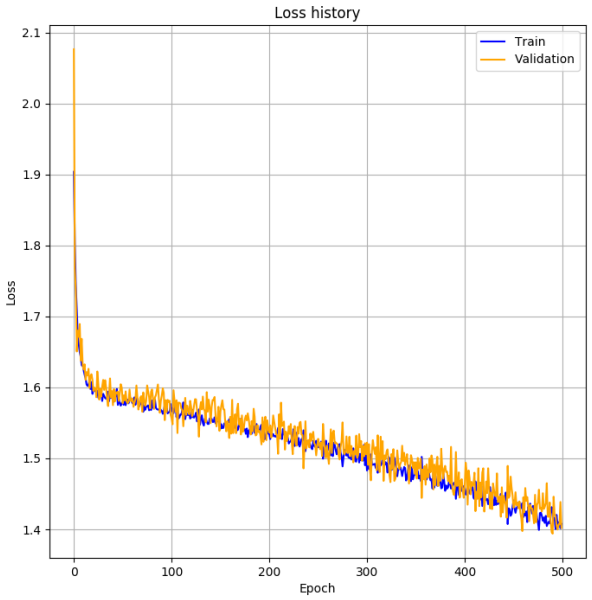
(b) Generalized Delta Rule



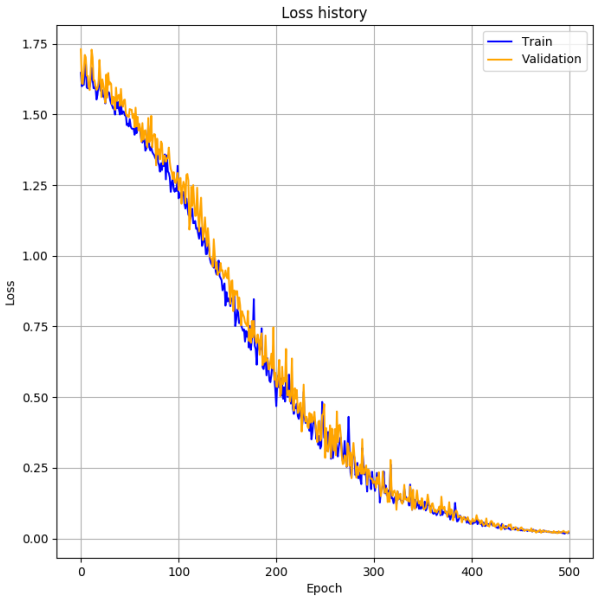
(c) Adam

Figure 22: Accuracy trends of the three optimizers

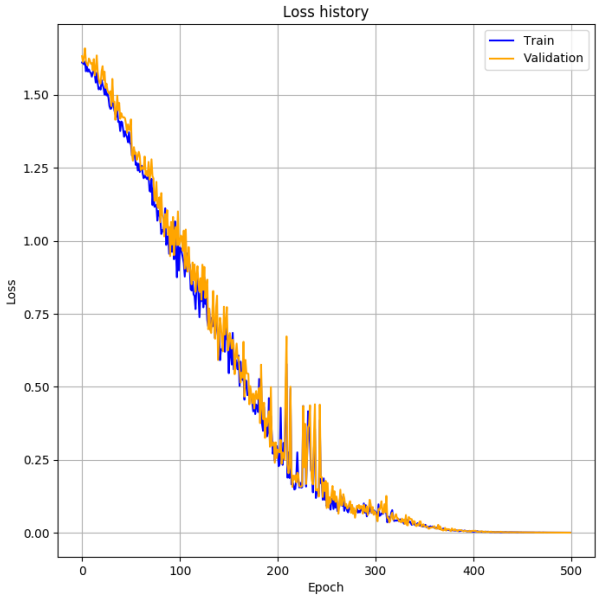
Loss Trends



(a) Delta Rule



(b) Generalized Delta Rule



(c) Adam

Figure 23: Loss trends of the three optimizers

Confusion Matrices
Annotated values depicted number of images with that property.

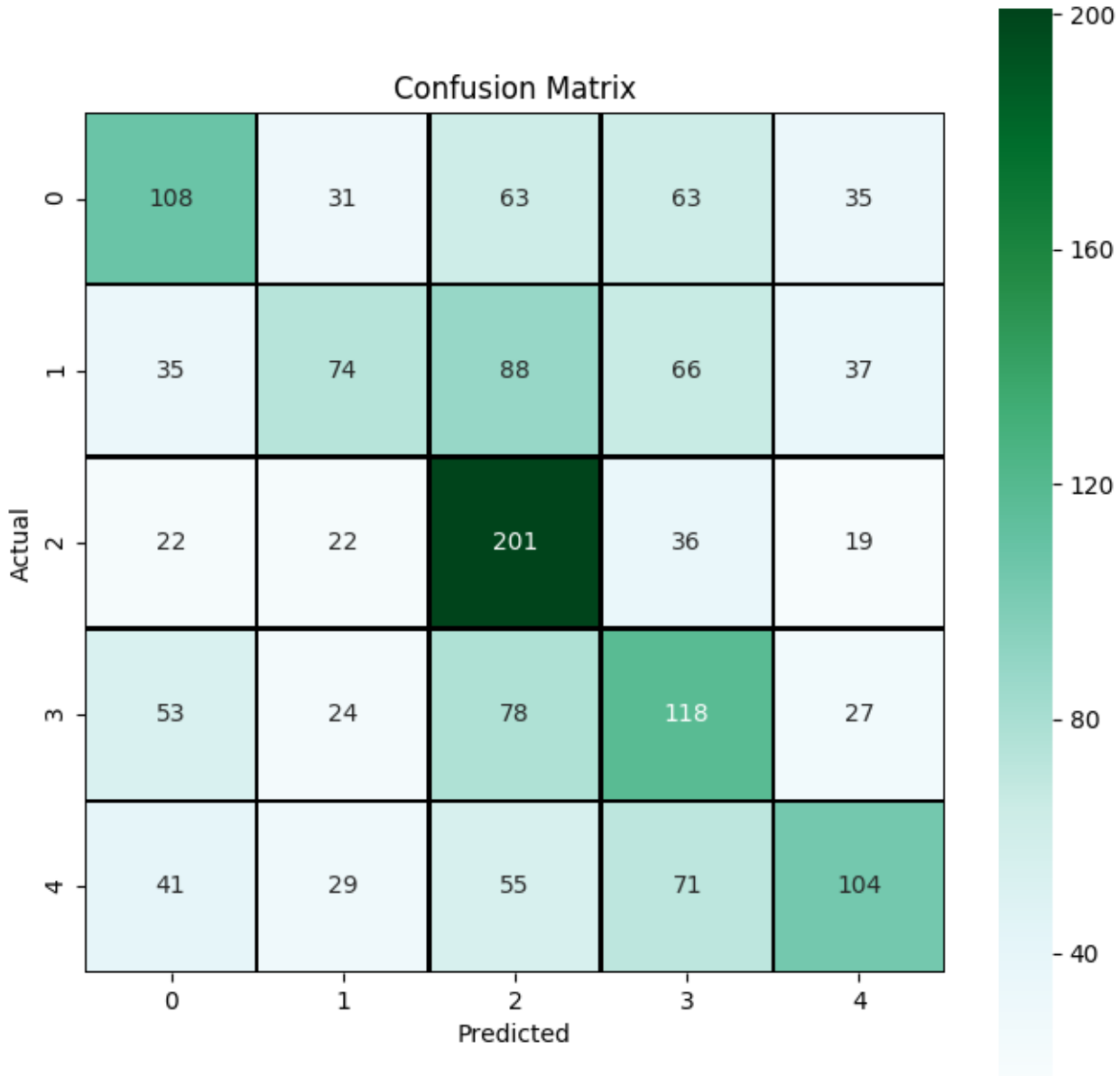


Figure 24: Delta Rule

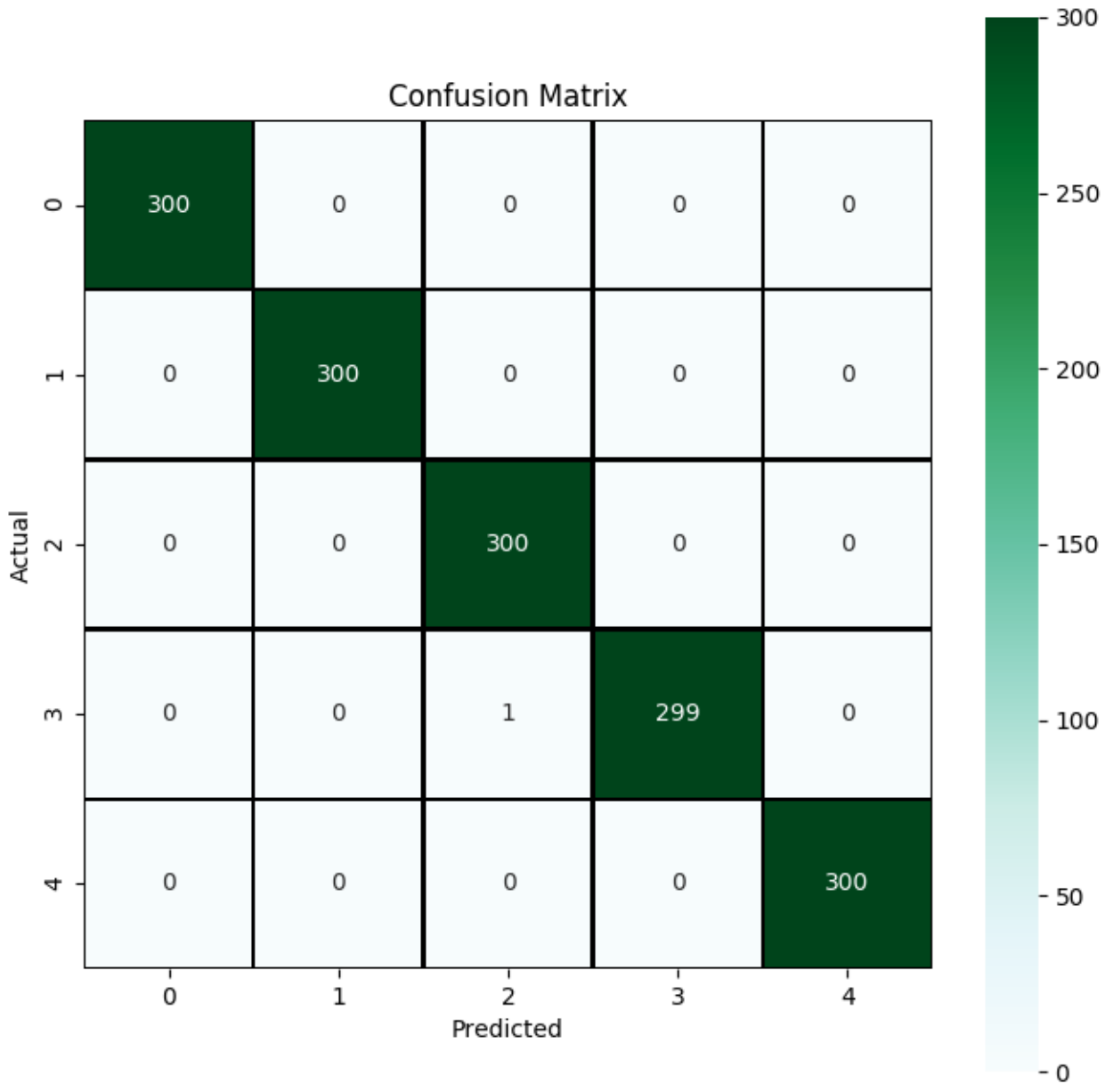


Figure 25: Generalized Delta Rule

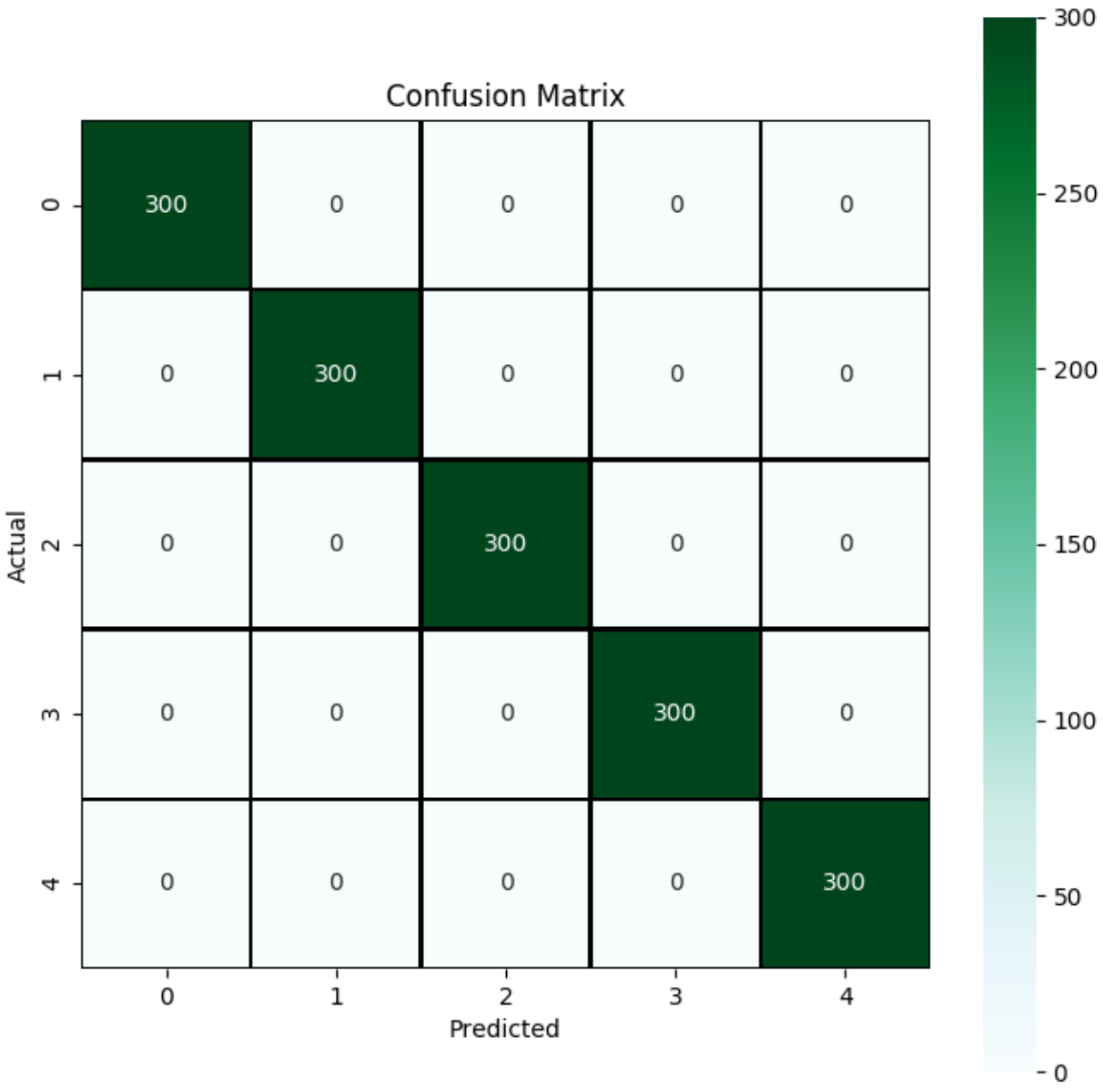


Figure 26: Adam

4.4 Inferences

The following facts can be noted from the trends above.

- Delta rule moves towards convergence very slowly, reaching only about 40% accuracy while Generalized Delta Rule and Adam reach close to 100% accuracies. Adam converges faster than Generalized Delta Rule.
- Generalized Delta Rule converged very smoothly for the considered configuration. Adam faced some disturbance halfway through, but quickly recovered. Delta Rule proceeds very haphazardly, moving around the same values of loss over several epochs.
- In general, Adam > Generalized Delta Rule > Delta Rule from overall performance point of view.

5 Work in Progress

The following features are being added to the architecture (and will hopefully be functional before the presentations).

- **Model saving**: To avoid training the model again and again, this feature will store the weights and biases of the model in a python dictionary, which will be saved as a `pkl` file. Every layer defined in `layers.py` has a `set_weights()` function which allows the user to set an external numpy matrix as the weights for that layer. This functionality will be extended to the entire model.
- **Overfitting detector**: If the gap between validation loss and training loss begins to increase and does so for certain number of epochs (specified by a hyperparameter called `patience`), the model will stop training and retain model parameters which it had when overfitting just began. This mechanism has been adapted from `CatBoost`, a GBM module developed by Yandex.
- **Early stopping**: If it so happens that loss value stabilizes reasonably (changes less than some percentage of itself after every epoch), then this feature will stop training and store the parameters which it possesses at that stage. This mechanism has been adapted from popular GBM modules such as `XGBoost` (dmlc) and `LightGBM` (Microsoft). Hopefully, it will work well for our architecture and save us some training time.

6 Endnote

Our neural network framework has been succesful in capturing simple non-linearities for regression and classification tasks. It can also handle high dimensional data without problems, as in the case of image classification task.