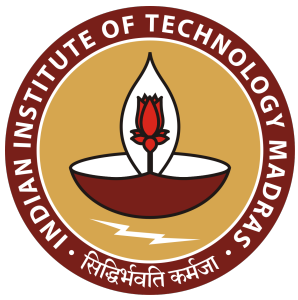
BT3041 Analysis and Interpretation of Biological Data

BT3041 AIBD Assignment 1

N Sowmya Manojna | BE17B007
Department of Biotechnology,
Indian Institute of Technology, Madras

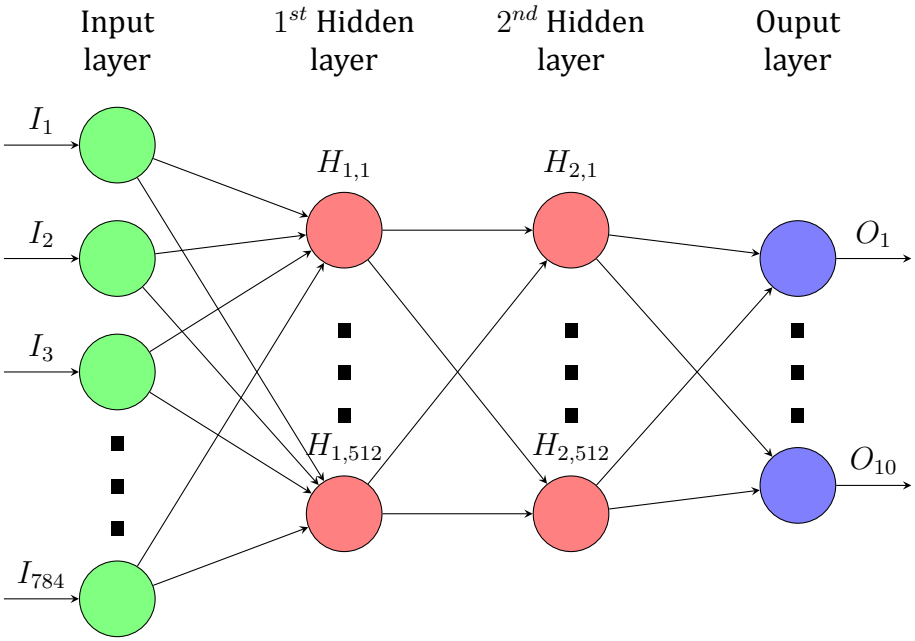


1 Question 1 - MLP

The solution to this question is coded using MATLAB.
Codes for this question have been written using MATLAB 2020a which performs automatic repmatting of matrices or vectors in matrix operations.

1.1 Network Architecture

The neural architecture is as follows:



The network comprises of:

- A input layer with 784 neurons
- Two hidden layers comprising of 512 neurons each
- An output layer with 10 neurons

1.2 Activation functions

- All the activation functions used are logistic activation functions.
- The sigmoid midpoint is placed at 0 and
- The logistic growth rate is made 15 fold.

The function used is:

$$f(x; c1, c2) = \frac{1}{1 + \exp^{-c1(x-c2)}}$$

Where, c1 denotes the midpoint and c2 denotes the growth rate of the function.
A steep growth rate is used because, most of the `z1,z2` values lie between the range -1 and 1. The sigmoidal function is largely linear in this phase and prevented the network from learning non-linear relations. Hence, in order to introduce non-linearity the logistic growth rate was increased and the parameters used were `c1=15, c2=0`.

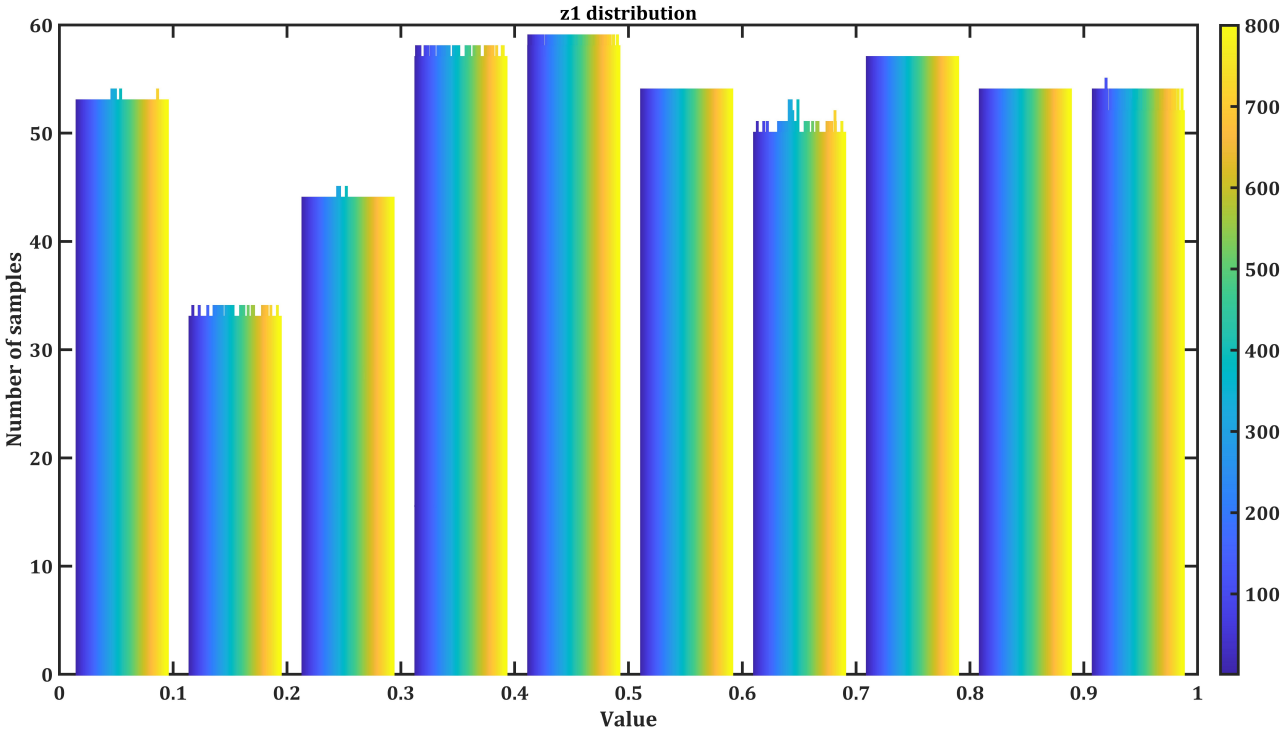


Figure 1: Distribution of values in `z1`

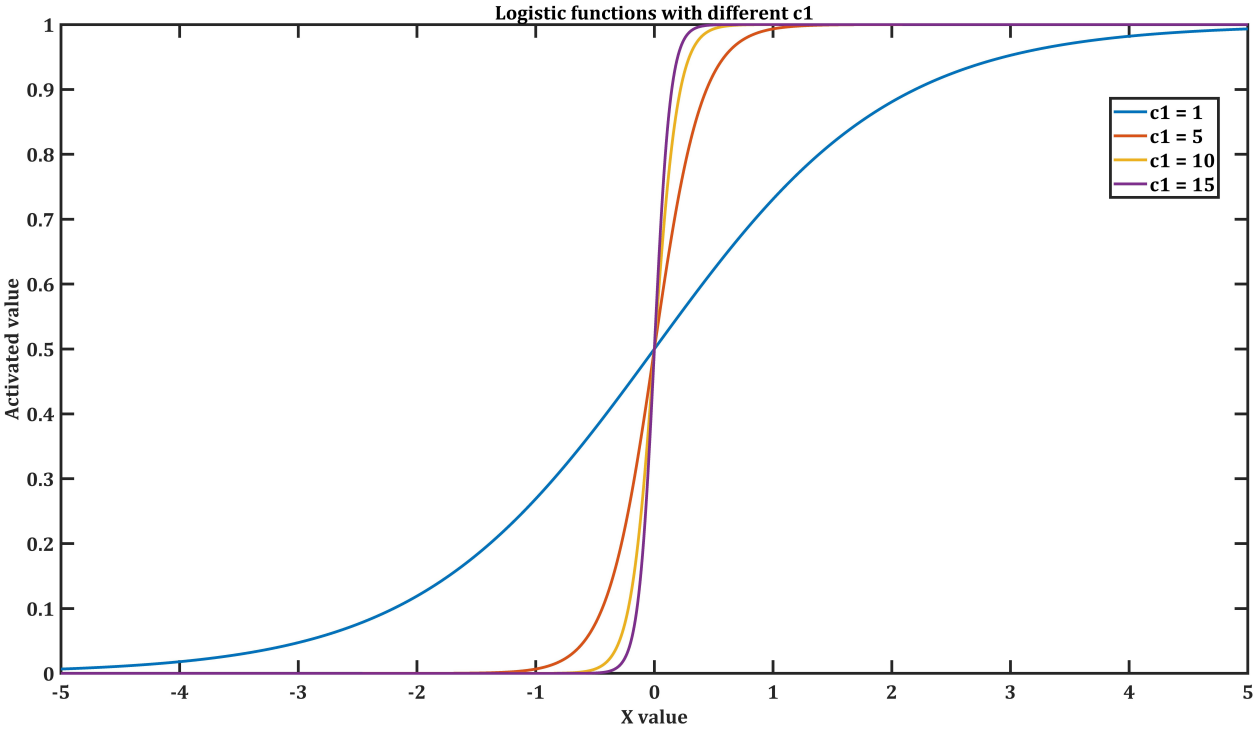


Figure 2: Plots of logistic functions with varying slope

The activation function with the constants is:

$$f(x) = \frac{1}{1 + \exp^{-15x}}$$

1.3 Loss Function

Multi-class Mean Squared Error has been used as the loss function.
The equation for the same is (vector notation):

$$loss = \frac{1}{2}||(predicted\_class\_vector - true\_class\_vector)||^2$$

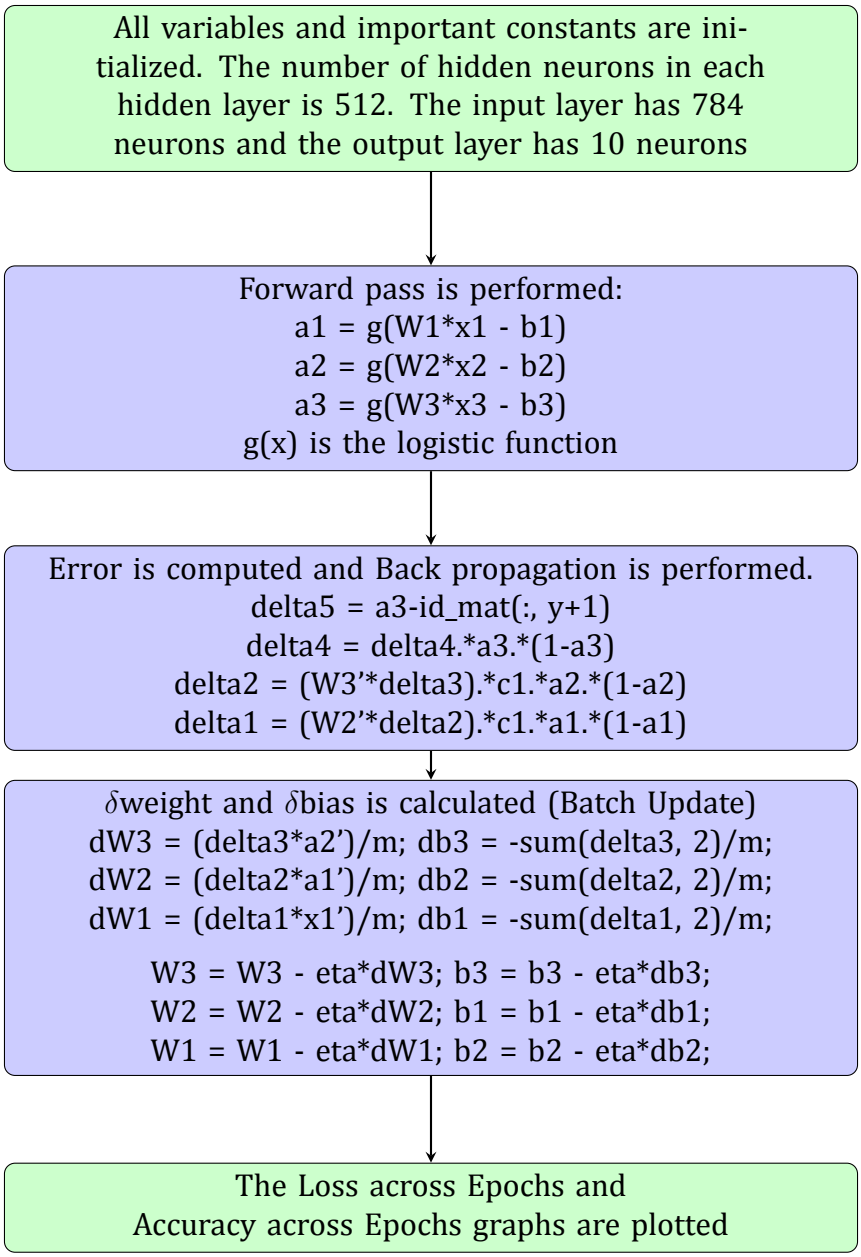
The `predicted_class_vector` and the `true_class_vector` are both of size $10 * 1$ where, an entry having 1 indicates that the predicted or actual number (i.e.)

- $(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$ means 0
- $(0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$ means 1
- $(0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T$ means 2 and so on

1.4 Learning Algorithm

The multi-class feed-forward network uses back-propagation for learning.

The flowchart of the algorithm is as follows:



The detailed algorithms is as follows:

```
% Accessing the images, labels and normalizing the images
x1 = double(trainImages')/255;
y = double(trainLabels');
t1 = double(testImages')/255;
yt = double(testLabels');

% Initializing the number of train and test samples
m = 800;
mt = 200;

% Defining all essential constants
% - eta_max and eta_min for the exponential eta decay function
% - epochs for training
% - fact (factor) for determinig the rate of eta decay
% - c1 rate of logistic function
% - save_every to save the accracy values after a fixed number of epochs
eta_max = 0.3;
eta_min = 0.01;
epochs = 6000;
fact = 5000;
c1 = 15;
c2 = 0;
save_every = 10;

% Defining the size of the network
n1 = size(x1,1);
n2 = 512;
n3 = 512;
n4 = 10;

% Initializing the weights, biases and additional variables
% to store the accuracy and
W1 = zeros(n2, n1);
W2 = zeros(n3, n2);
W3 = zeros(n4, n3);

b1 = rand(n2, 1);
b2 = rand(n3, 1);
b3 = rand(n4, 1);

id_mat = eye(n4);
train_accuracy = zeros(ceil(epochs/save_every),1);
test_accuracy = zeros(ceil(epochs/save_every),1);

% Start training
for i = 1:epochs
    % Forward pass
    z1 = W1*x1 - b1;
    a1 = sigmf(z1, [c1, 0]);

    z2 = W2*a1 - b2;
    a2 = sigmf(z2, [c1, 0]);

    z3 = W3*a2 - b3;
    a3 = sigmf(z3, [1, 0]);

    % Train and test accuracy calculation
    if rem(i, save_every) == 0
```

```
        fprintf('Epoch: %d; eta: %f \n', i, eta);
        [¬, idx] = max(a3);
        train_accuracy(i/save_every) = sum(idx' == y+1)/m;

        % Testing
        t2 = sigmf(W1*t1 - b1, [c1,0]);
        t3 = sigmf(W2*t2 - b2, [c1,0]);
        t4 = sigmf(W3*t3 - b3, [1,0]);
        [¬, idx] = max(t4);
        test_accuracy(i/save_every) = sum(idx' == yt + 1)/mt;
    end

    % Backpropagation
    d4 = a3 - id_mat(:, y+1);
    d3 = d4.*c1.*a3.*(1-a3);
    d2 = (W3'*d3).*c1.*a2.*(1-a2);
    d1 = (W2'*d2).*c1.*a1.*(1-a1);

    dW3 = (d3*a2')/m;
    dW2 = (d2*a1')/m;
    dW1 = (d1*x1')/m;
    db3 = -sum(d3, 2)/m;
    db2 = -sum(d2, 2)/m;
    db1 = -sum(d1, 2)/m;

    % get eta based on epoch
    eta = get_exp_eta(i, eta_max, eta_min, fact);

    % Weight and bias update
    W3 = W3 - eta*dW3;
    W2 = W2 - eta*dW2;
    W1 = W1 - eta*dW1;
    b3 = b3 - eta*db3;
    b2 = b2 - eta*db2;
    b1 = b1 - eta*db1;
end
```

1.5 Learning Rate

A combination of exponential decay and step learning rate schedule is used. The initial learning rate (maximum learning) rate is 0.2 and the final learning rate (minimum) is 0.01.

The learning rate schedule is plotted below:

The learning rate schude is as follows:

$$\eta = \begin{cases} \eta_{min} + (\eta_{max} - \eta_{min}) * \exp^{\frac{-epochs}{fact*0.5}} & epochs < fact \\ \eta_{min} & otherwise. \end{cases}$$

The η_{min} and η_{max} used in this model are 0.01 and 0.3 respectively.

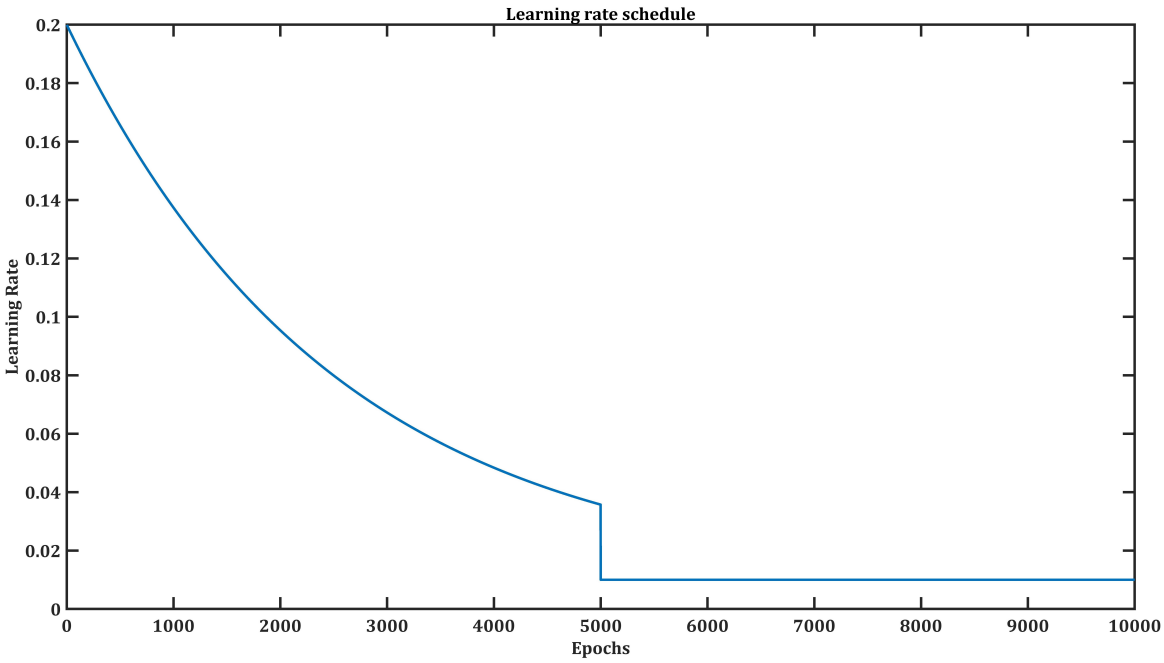


Figure 3: Learning rate schedule used in the model

1.6 Loss and Accuracy testing

The accuracy obtained after training for 6000 epochs is as follows:

- Train accuracy: 99.5%
- Test accuracy: 87%

The final loss obtained after training for 6000 epochs is 0.0108. The loss distribution across epochs is as follows:

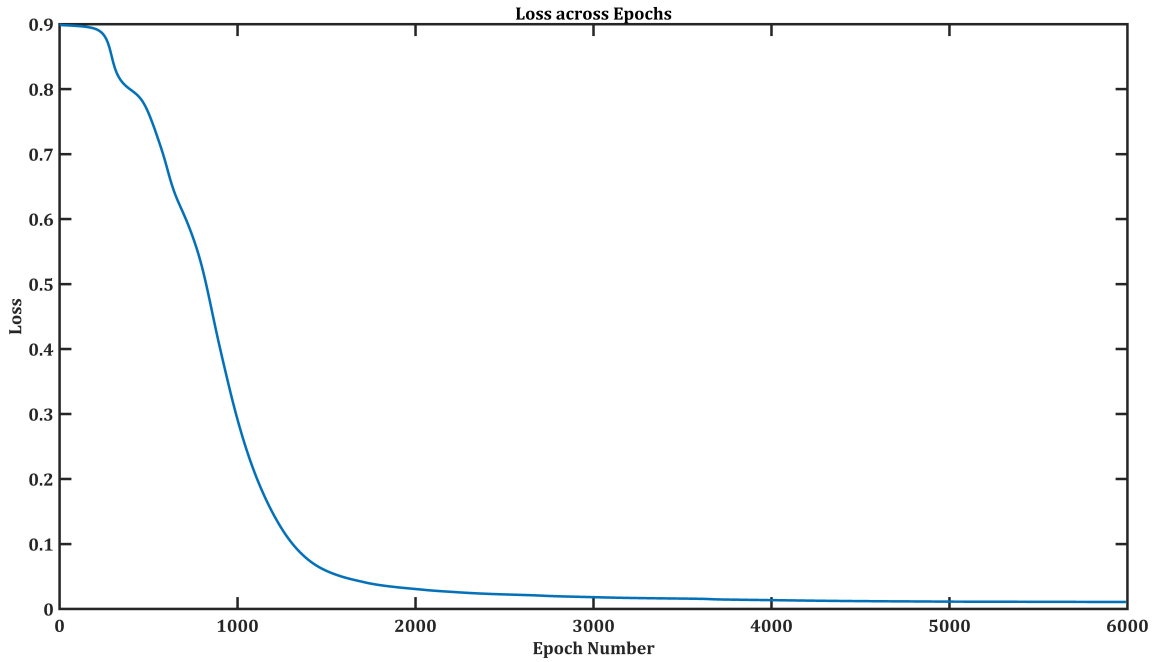


Figure 4: Loss function value across epochs

The test and train accuracy distribution across epochs is as follows:

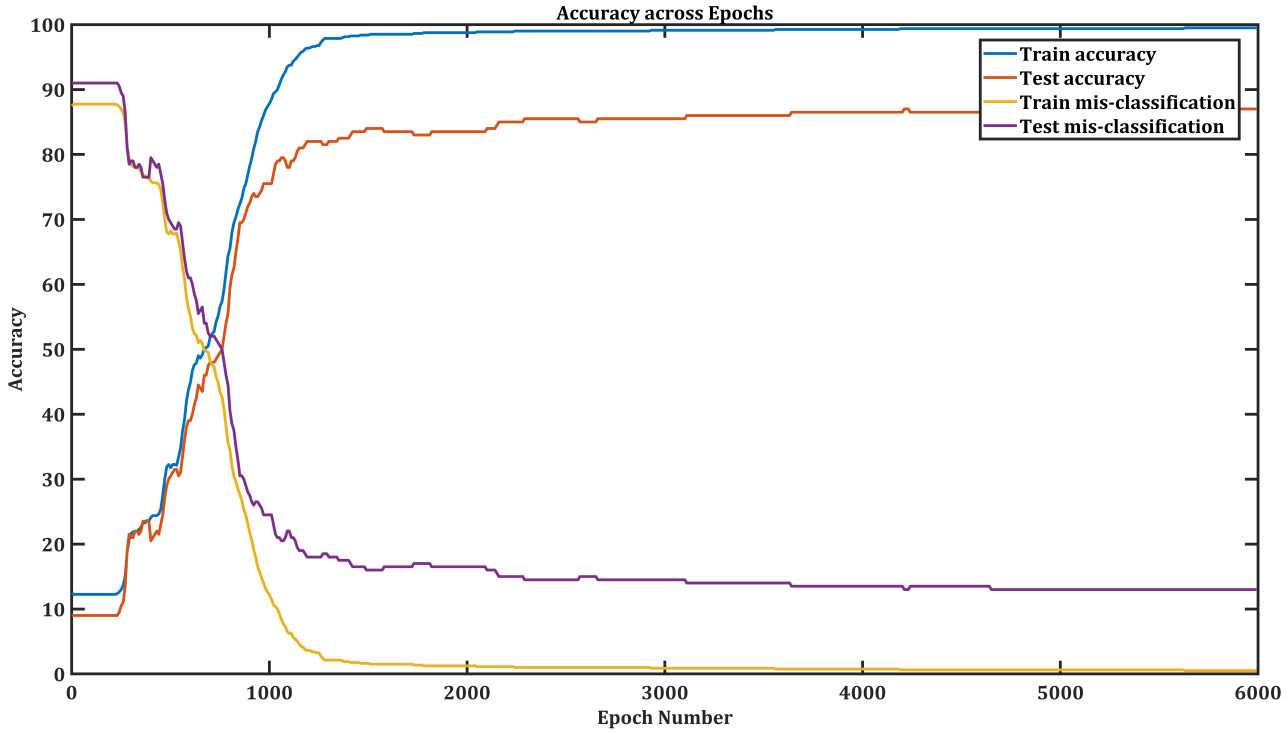


Figure 5: Accuracy across epochs

2 Question 2 - DBSCAN

The solution to this problem is coded using Python3 and the Jupyter framework is used for the same.
All the libraries used for this question can be installed by using the following code (inside question_2 folder): `$ pip3 install -r requirements.txt`

2.1 Parameters

The parameters used to obtain 4 clusters is:

- Epsilon radius (ϵ) = 0.3
- `min_points` = 40

2.2 Procedure

- The distance between all pairs of points was calculated.
- The number of neighbors a point has is calculated by considering all distances that are less than `eps`.
- Points that have more than `min_points` number of neighbors are labeled **core points**.
- Points that have less than `min_points` number of neighbors, but are reachable from core points are labeled **neighbor points**.
- Points that have less than `min_points` number of neighbors and are not neighbor points are labeled **noise points**.

- Cluster allocation is done using a Breadth First Search (BFS) like spanning through the core points. All the accessible core points and their neighbors are assigned the same cluster number.
- Once all the accessible core points and their neighbors are given a cluster label, the current cluster label is incremented and the cluster allocation is done using the next set of core points.

2.3 Results

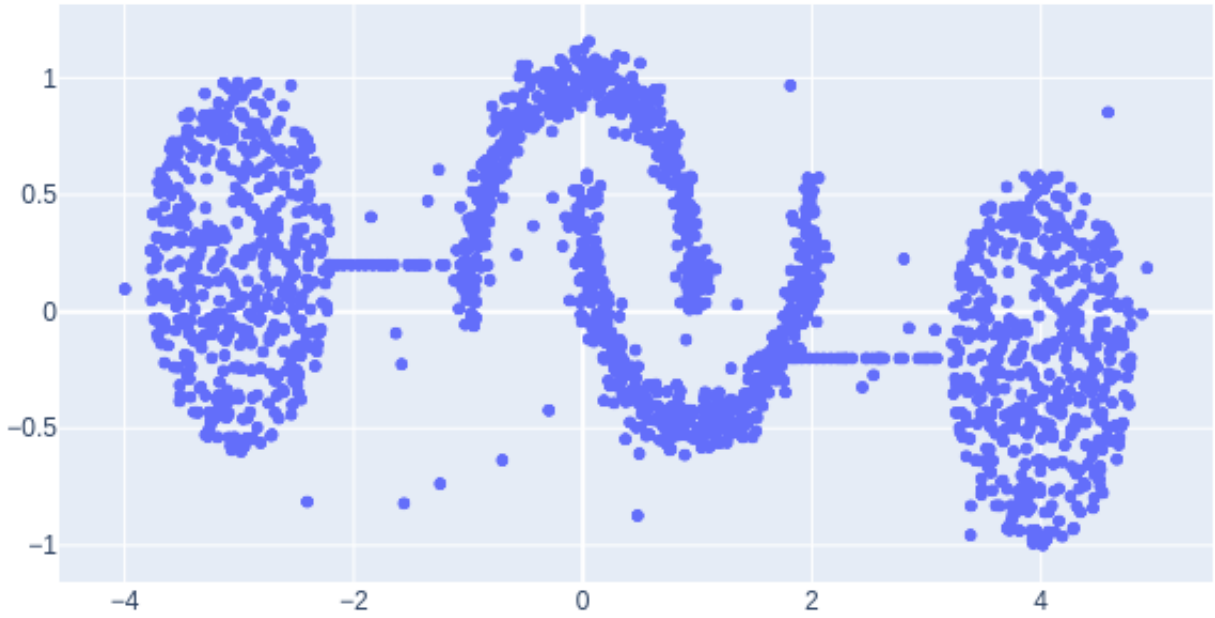


Figure 6: The initial unclustered data

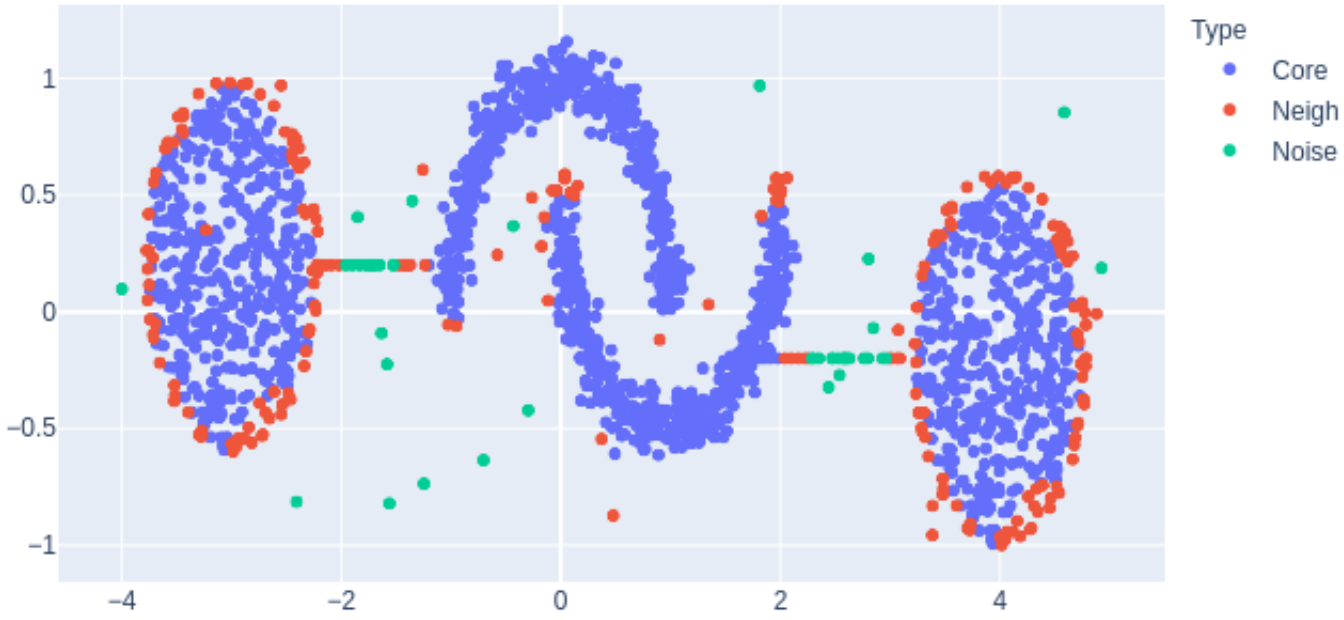


Figure 7: Core, Neighbor and Noise points labeled

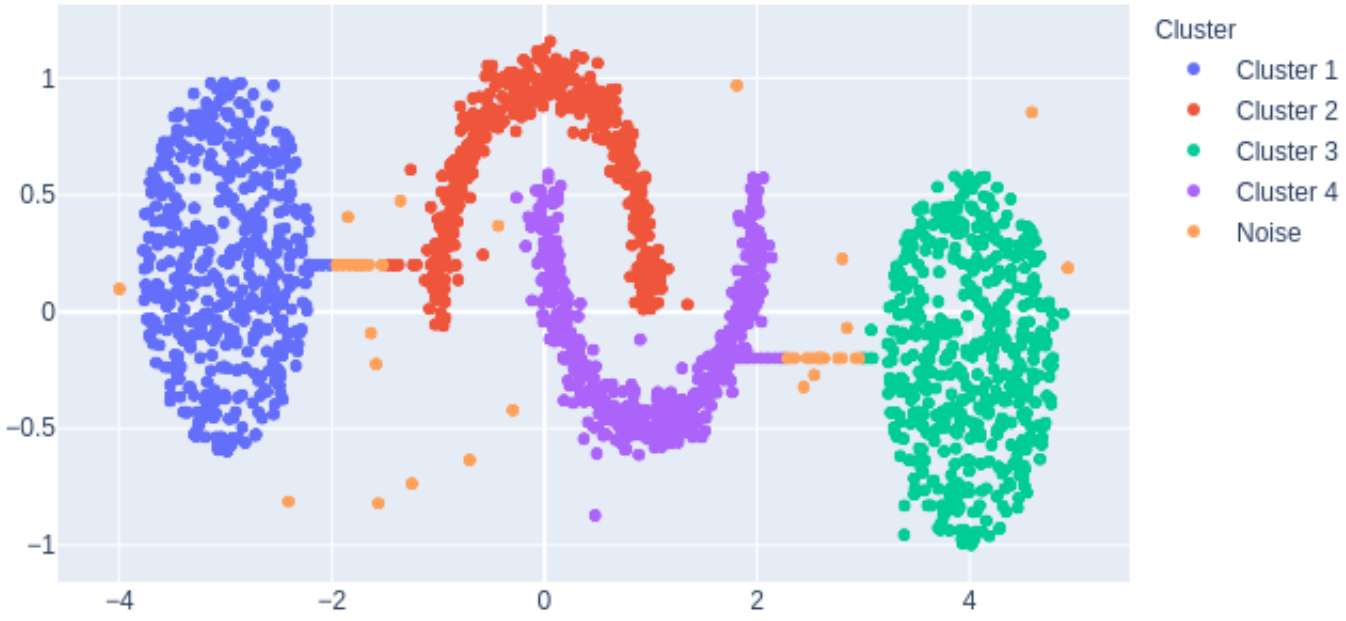


Figure 8: Final clustered data

The distribution of points in clusters is as follows:

- Cluster 1: 509 points
- Cluster 2: 468 points
- Cluster 3: 512 points
- Cluster 4: 466 points
- Noise: 45 points