DL Assignment - 1

Mohammad Aflah Khan, 2020082
Neemesh Yadav, 2020529

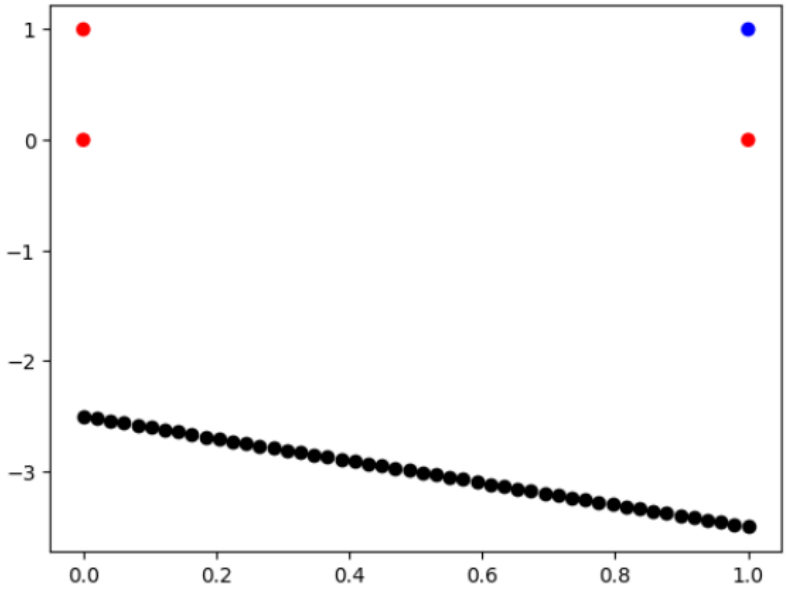
Contributions: The work was equally divided between the two members where one member worked on part 1, the other worked on part 3, and both pondered together upon part 2.

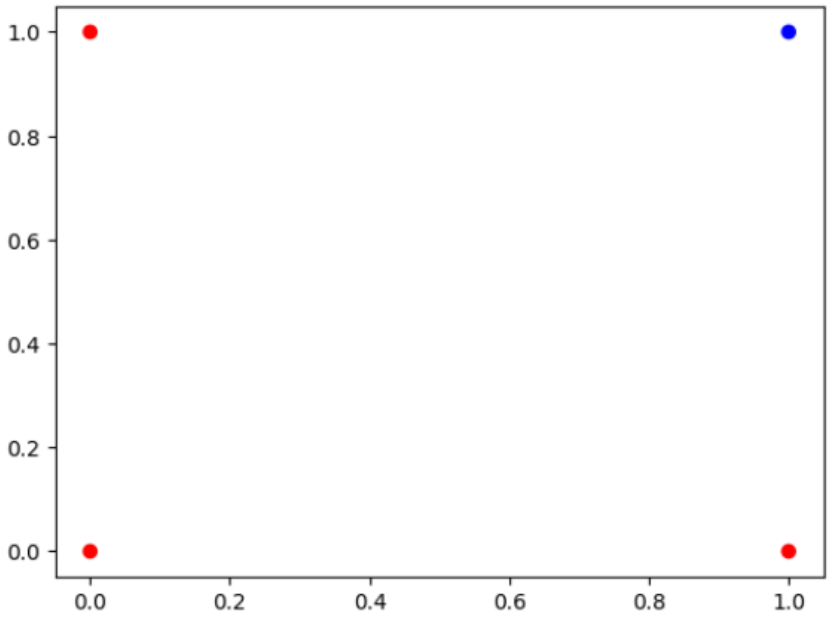
Note that when the weights are 0, there is no definite boundary to plot hence those images have no boundaries.
The weights for each iteration are shown as cell outputs in the jupyter notebooks, the model dumps are present in the Model_Saves folder under the respective part folder, and the Gradient Descent Logs are present under the Gradient_Descent_Logs folder for part 2.

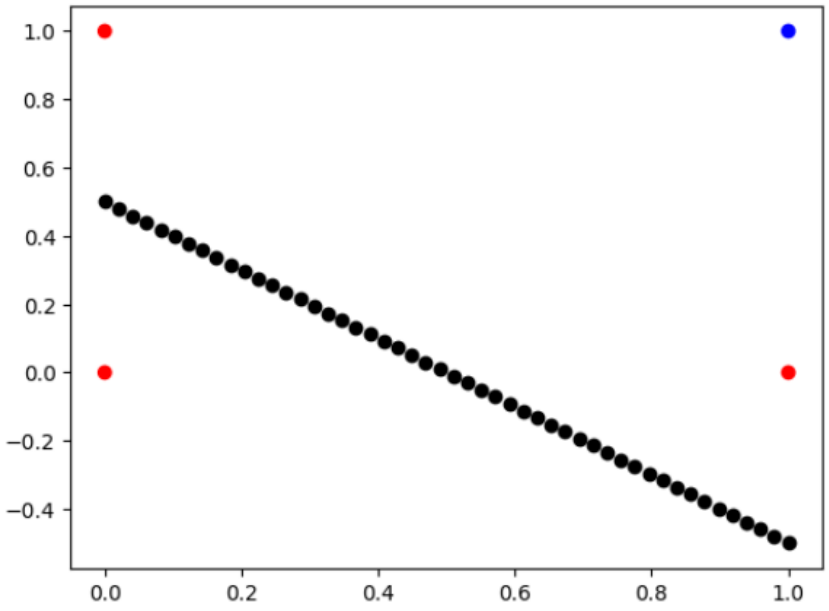
Part-1

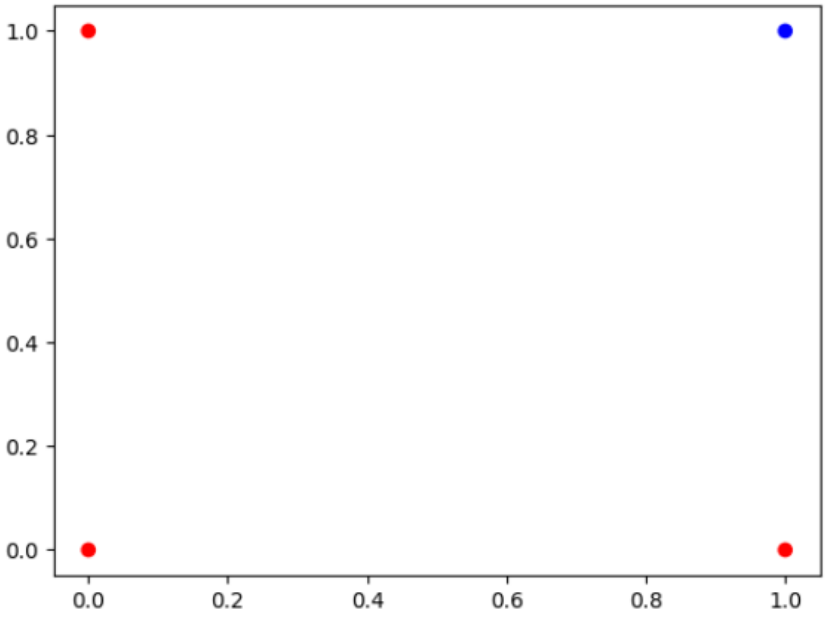
a) Steps taken to converge:
   i) AND: Converged at the 5th iteration
   ii) OR: Converged at the 5th iteration
   iii) NOT: Converged in 2 iterations

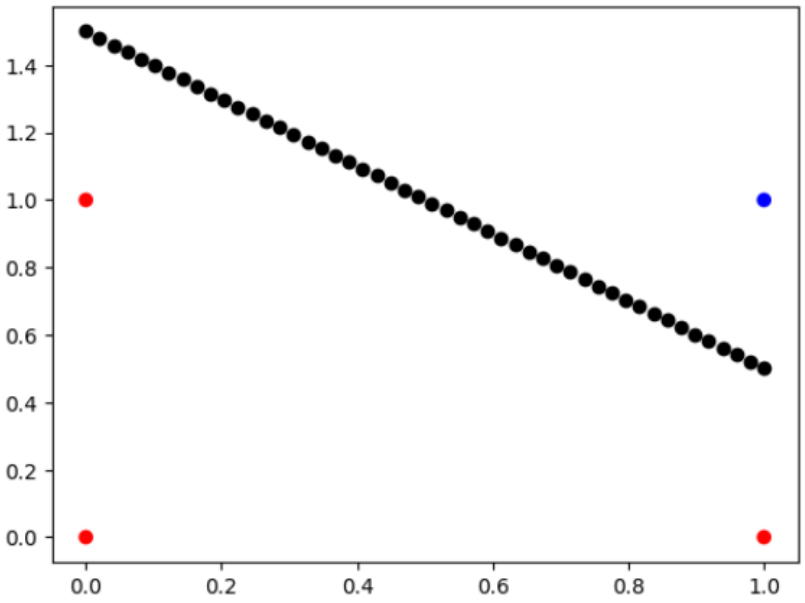
b) Plots:
   i) AND:



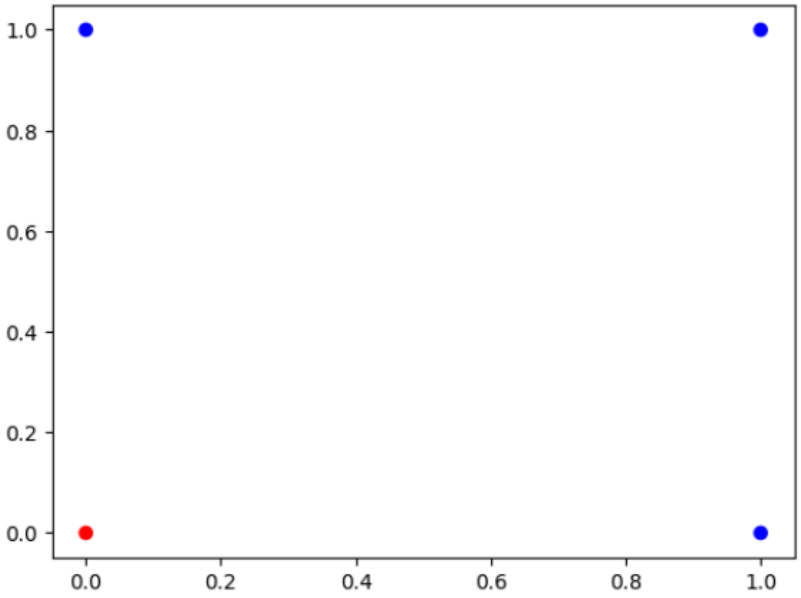

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

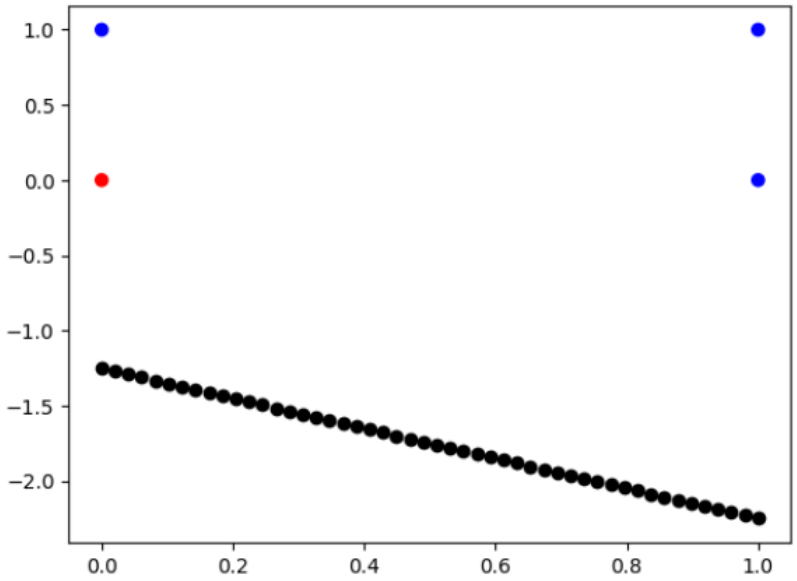

1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0.0
0.2
0.4
0.6
0.8
1.0

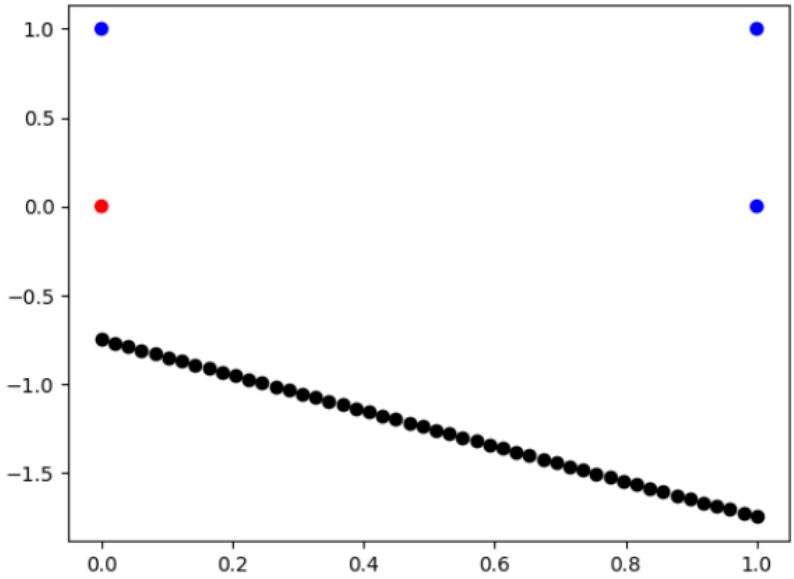

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

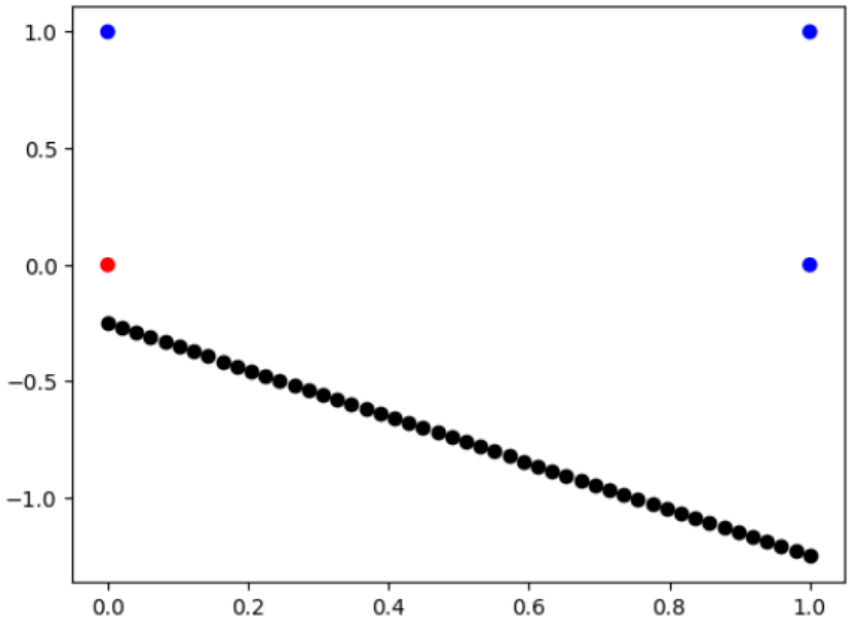

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

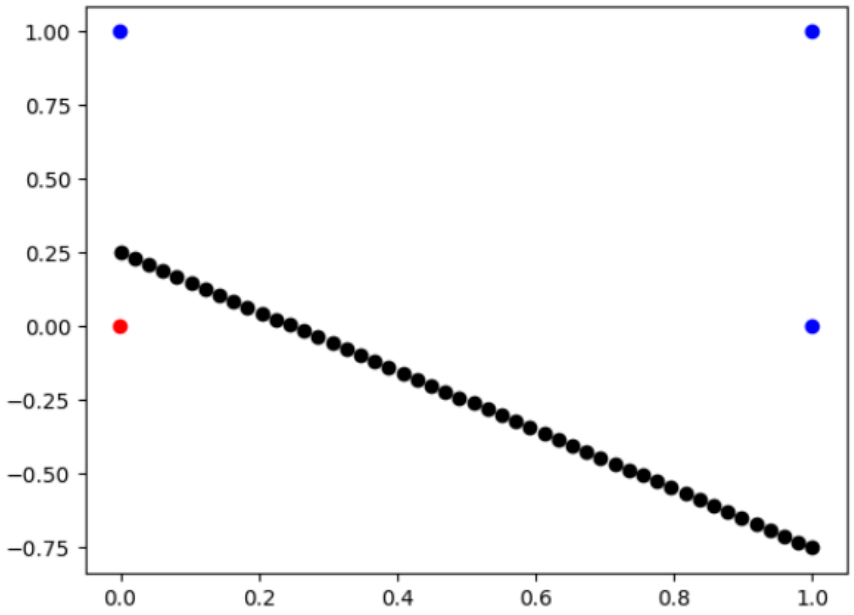
ii) OR:


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

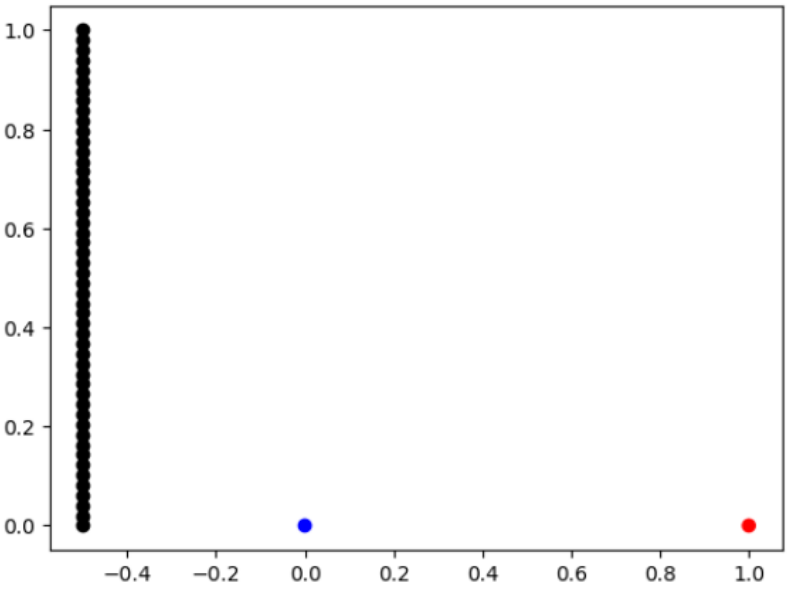

1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.2
0.4
0.6
0.8
1.0

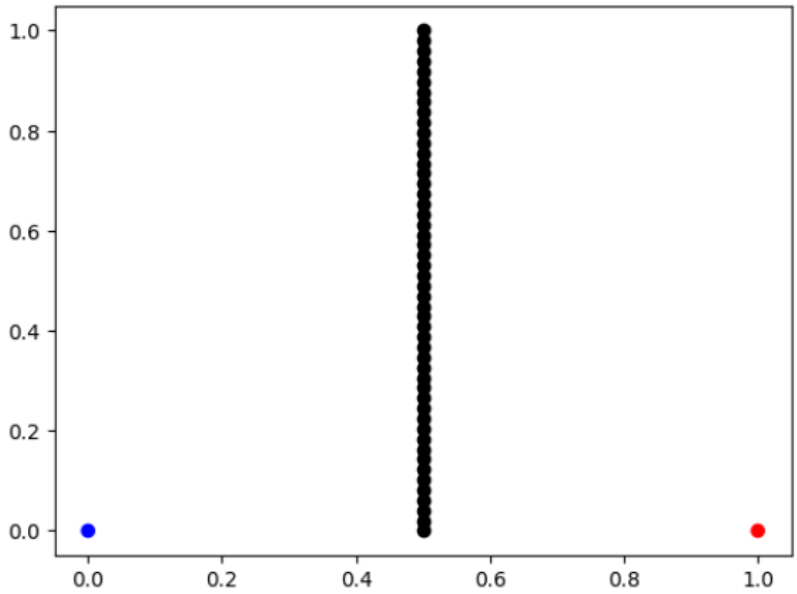

1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

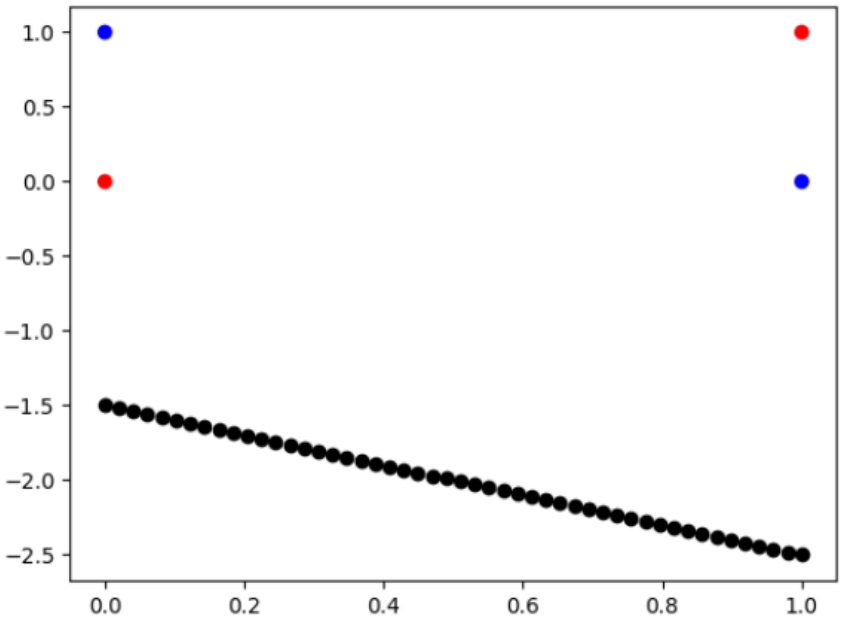

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0.0
0.2
0.4
0.6
0.8
1.0

iii) NOT:

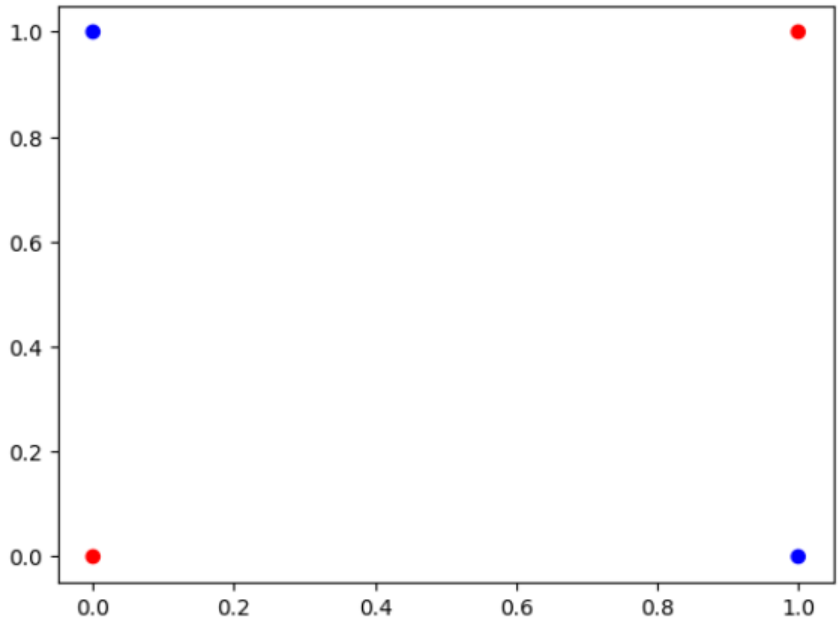

1.0
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0



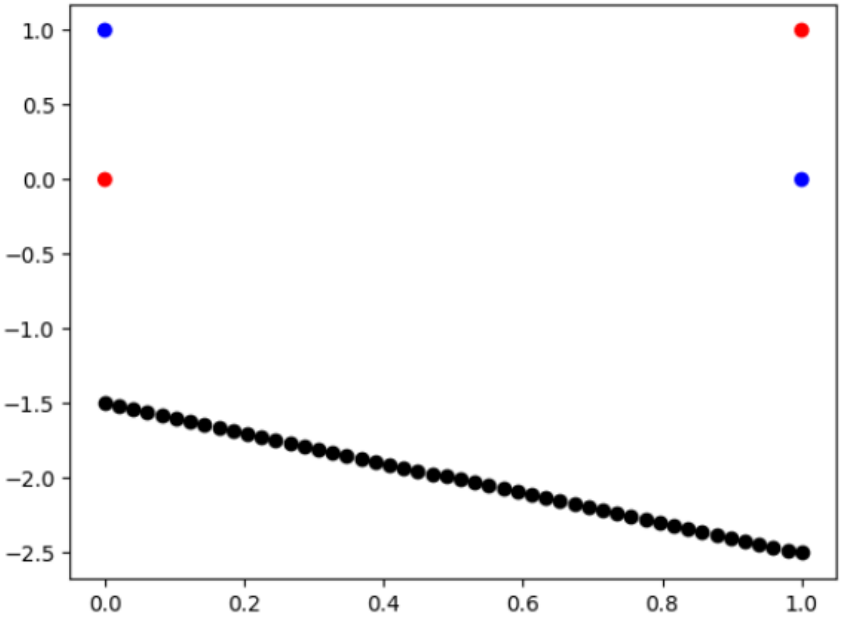
c) In only three iterations we can see that we start getting repetitive weights which implies that it will be stuck in a loop for as many iterations as we want, and will never be able to converge if we use a perceptron. Here, we tried to run our PTA on the XOR dataset for 10 epochs and we got the following Decision Boundary Plots and weights:
Iteration 1: w0, w1, w2 ~ -3, -2, -2



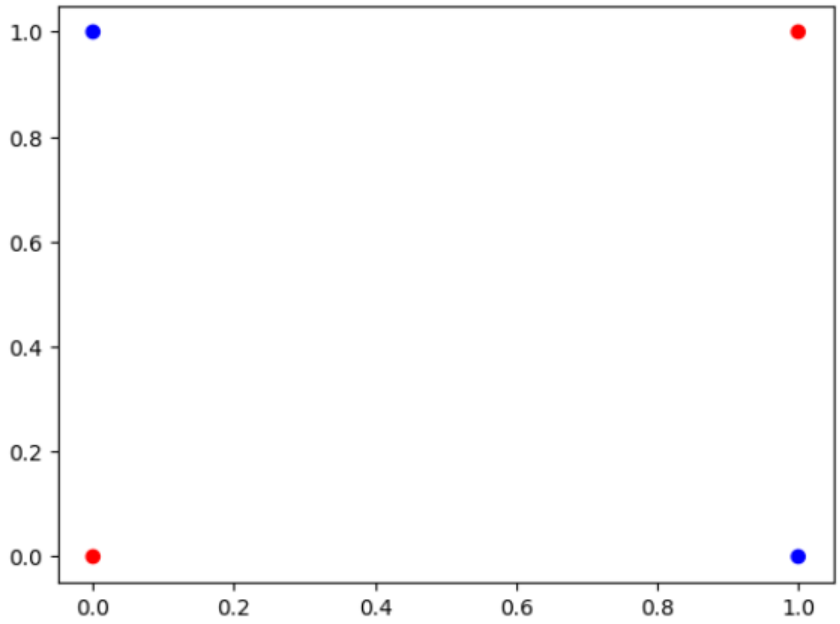
Iteration 2: w0, w1, w2 ~ 1, 0, 0



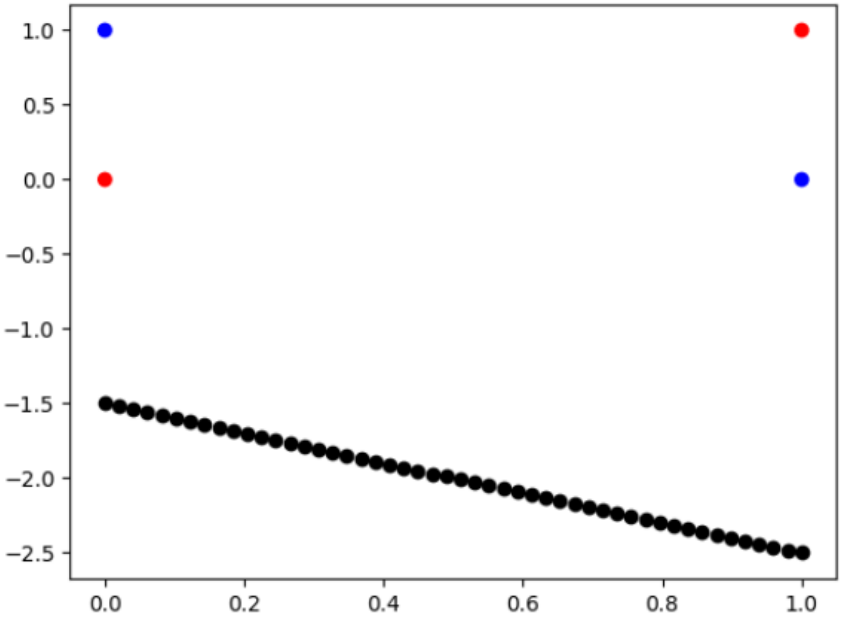
Iteration 3: w0, w1, w2 ~ -3, -2, -2



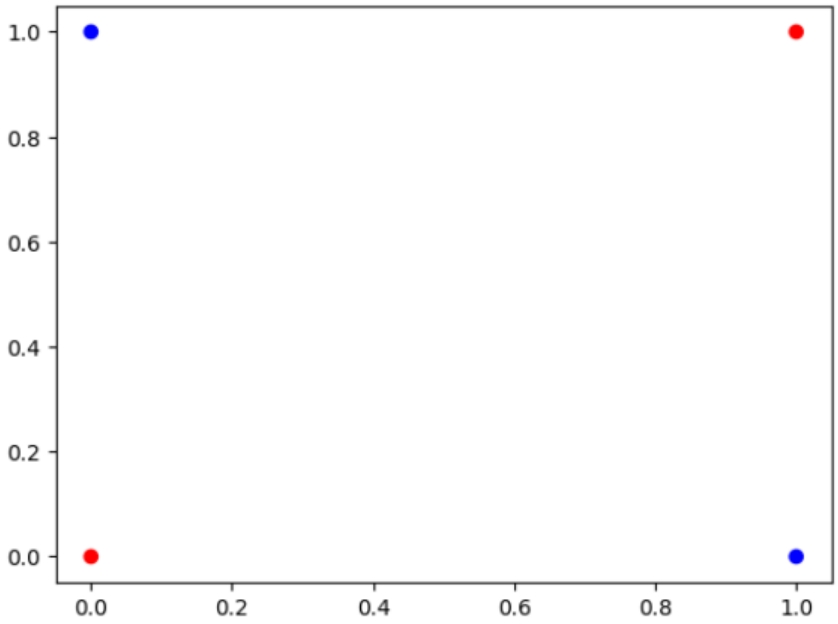
Iteration 4: w0, w1, w2 ~ 1, 0, 0



Iteration 5: w0, w1, w2 ~ -3, -2, -2



Iteration 6: w0, w1, w2 ~ 1, 0, 0

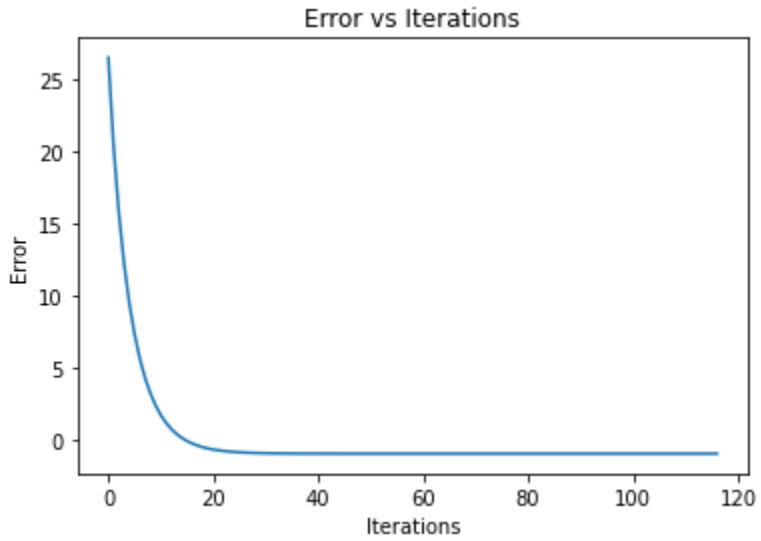


We ran our PTA for 10 iterations but from these 6 iterations, it's pretty clear that the weights are only looping around two values and will continue to do so for the remainder of the iterations. Hence we can say that the XOR problem cannot be solved using a single perceptron.

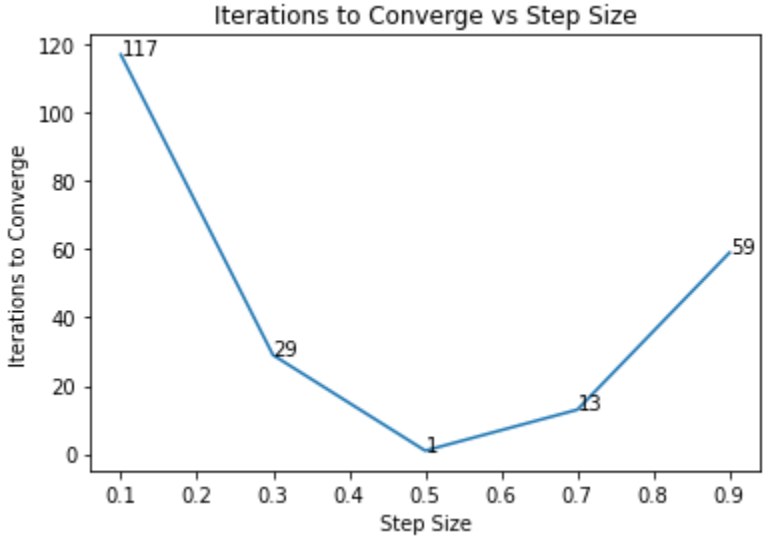
Part-2

b)
x1_initial: 6.888437030500963
x2_initial: 5.159088058806049
Steps: 117

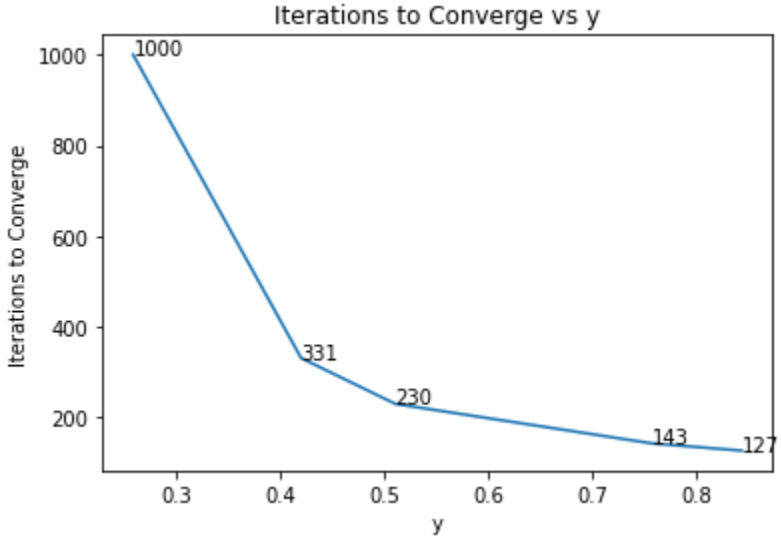
c)



d)

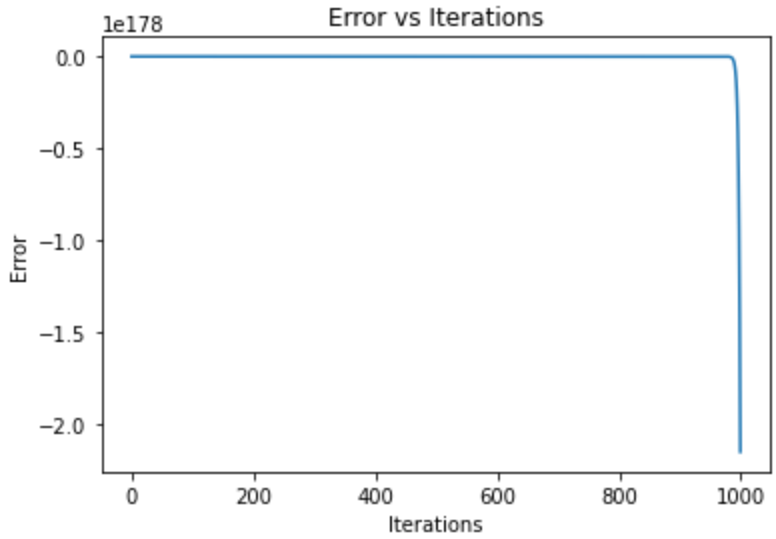


e)
Y-values: [0.25891675029296335, 0.420571580830845, 0.5112747213686085, 0.7579544029403025, 0.8444218515250481]



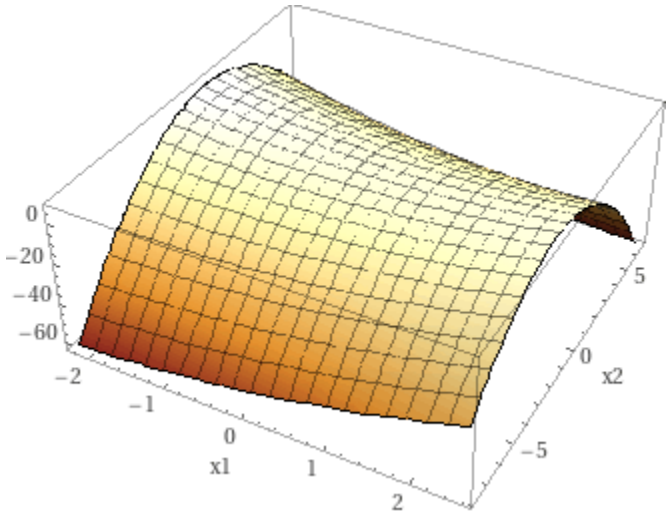
Hence for some values of y convergence is better than others. From the trend it seems the closer the value is to 1 the quicker the convergence.

f)
We can see there is no convergence for 1000 iterations -



Also the error keeps increasing in magnitude

If we try plotting the function we see that the function does not have any minima-



Hence the gradient descent algorithm which can be thought of as descending down a mountain to a valley keeps descending in hopes of reaching a minimum value but at each point it can go further down indefinitely. This is why gradient descent fails to work thereby explaining the observed behavior.

Part - 3

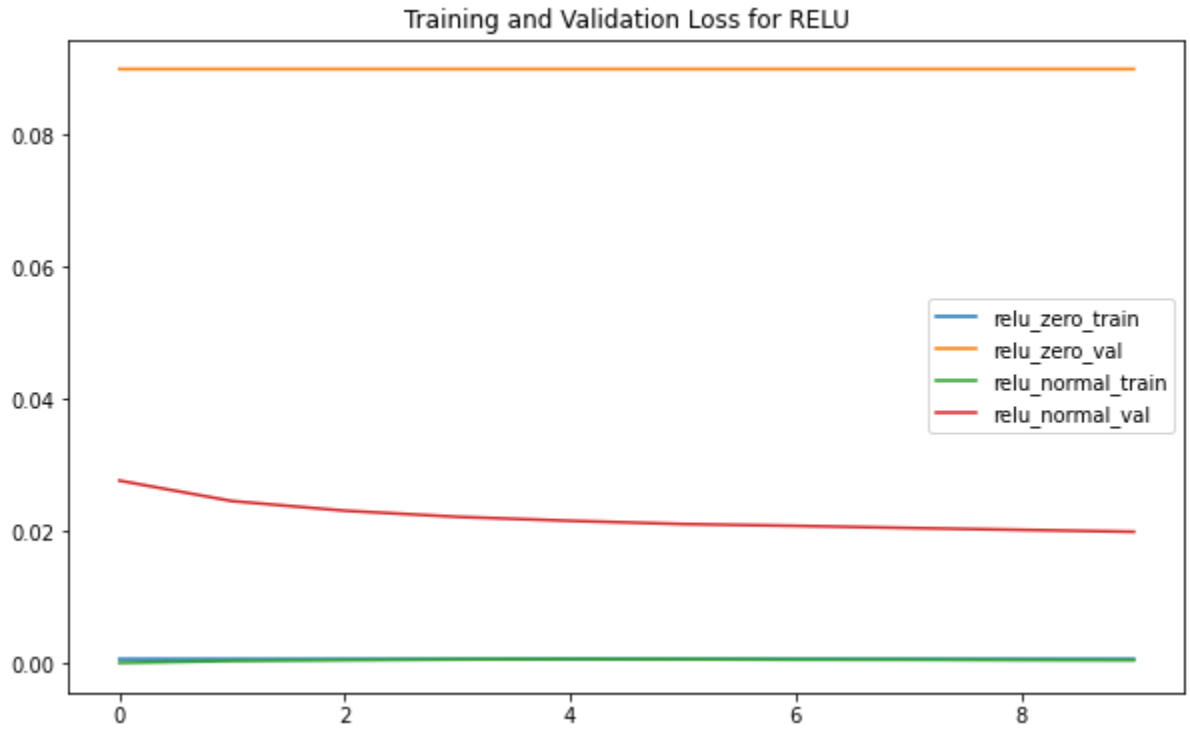
Hyperparameters Used:

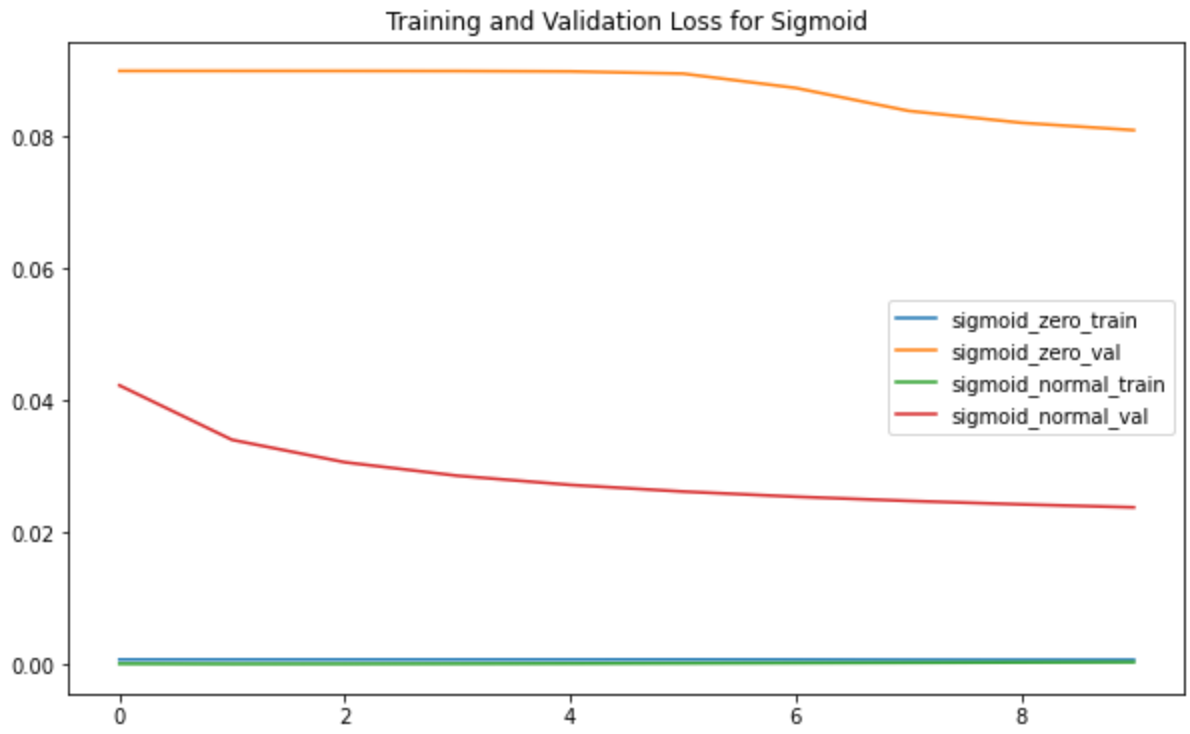
- 1 Hidden Layer of Size 128
- Learning Rate 0.01
- Batch Size 128
- Early Stopping with Patience 5
- Epochs 10

c)

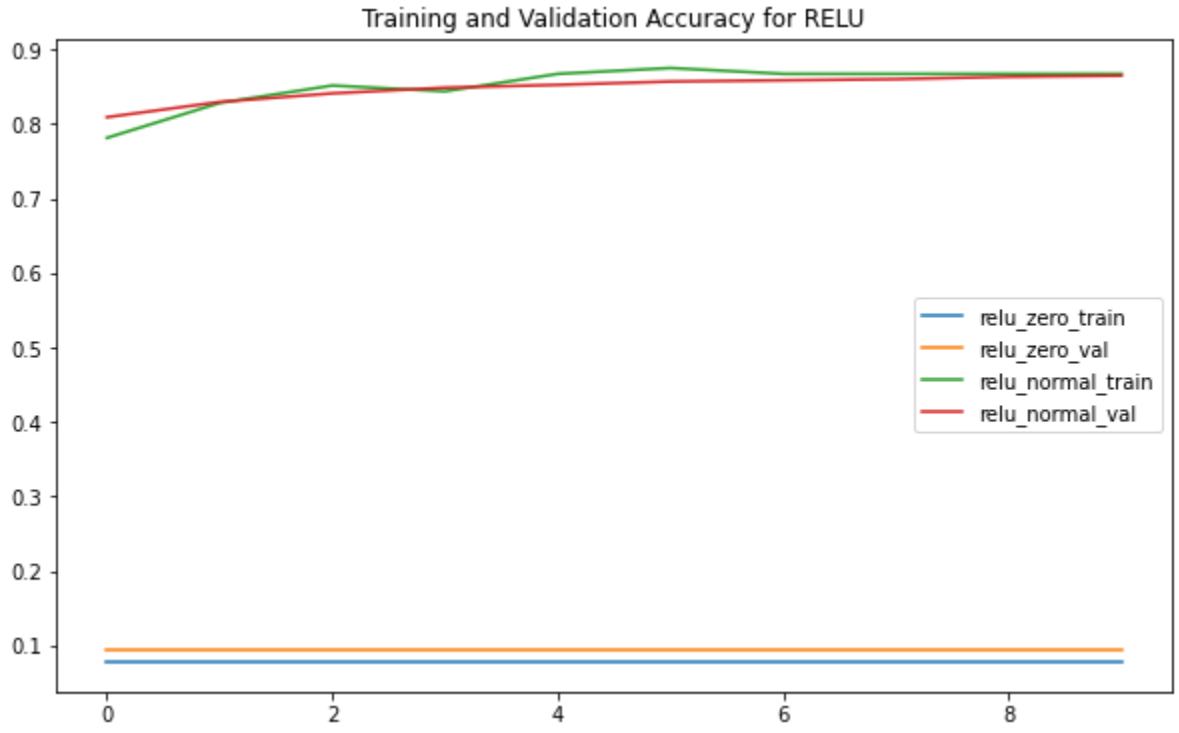
Activation Function: relu, Initialization: normal_init - Accuracy: 0.8661
Activation Function: relu, Initialization: zeros - Accuracy: 0.1
Activation Function: sigmoid, Initialization: normal_init - Accuracy: 0.8416
Activation Function: sigmoid, Initialization: zeros - Accuracy: 0.2489

Loss plot - Training Loss and Validation Loss V/s Epochs.





Accuracy plot - Training Accuracy, Validation Accuracy V/s Epochs)



Analysis:

- Whenever the weight initialization is 0 both the loss and accuracy suffer as the model is unable to learn. In the case of relu this is because relu(0) = 0 and hence the gradients are always 0. For sigmoid this is not the case as it is not 0 at input 0 but it causes the weights to move in a similar fashion thereby reducing the effectiveness of backpropagation.
- For normal initialization these problems are not present and the model works well and converges to a good accuracy within just 10 epochs.
- The loss plots and accuracy plots represent these above described trends.
- Relu seems to be better than Sigmoid in the 10 epoch run but different random initializations might tip the favor and robust analysis across seeds would be needed to ascertain the trend.
- The growth is smaller than expected across epochs as we observe even after 1 epoch the model has learnt quite a lot and gains near about 85% Accuracy. This validates our implementation as the 2019 SOTA stood at 91.4% Accuracy according to PapersWithCode.