Name: Ipsa Mishra
Student ID: 1001759616

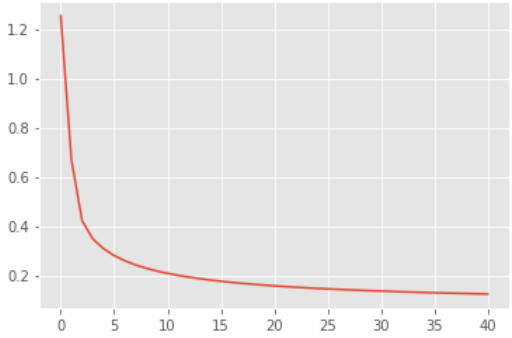
Data Mining Assignment

We have developed python program for executing both the codes.

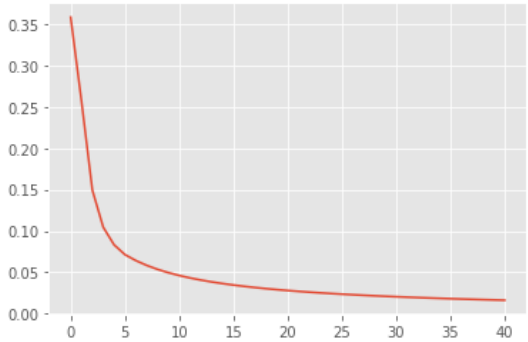
Problem1

1. Perform batch training using gradient descent. Divide the derivative with the total number of training dataset as you go through iteration (it is very likely that you will get NaN if you don't do this.). Change your learning rate as η = {1, 0.1, 0.01, 0.001}. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training loss (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number of iterations it took for training and the accuracy that you have.

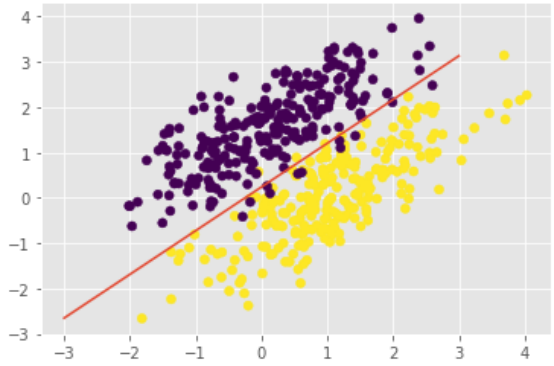
Step_Size = 1, L1-norm(gradiant) threshold = 0.001
Number of epochs: 40
Accuracy: 97.2
Loss Plot:



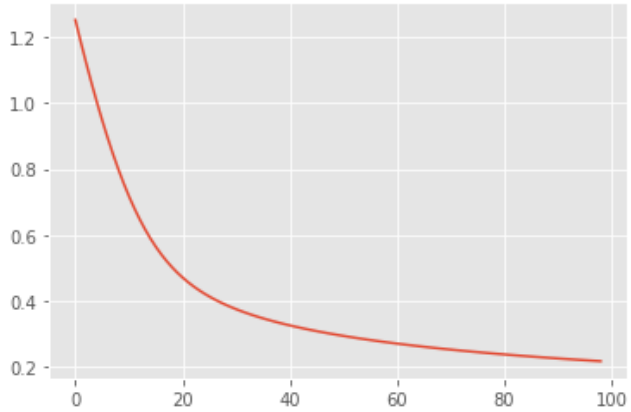
Gradient Plot:



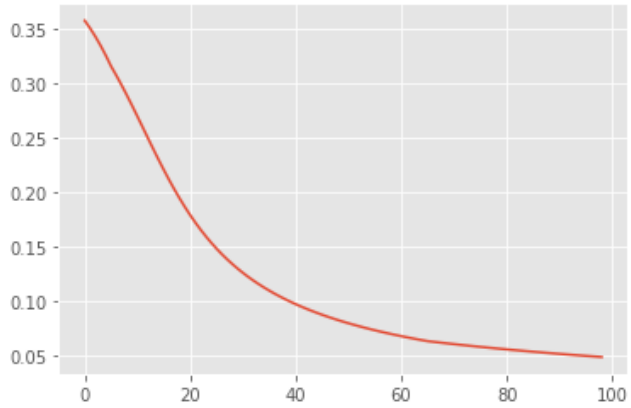
Scatter Plot:



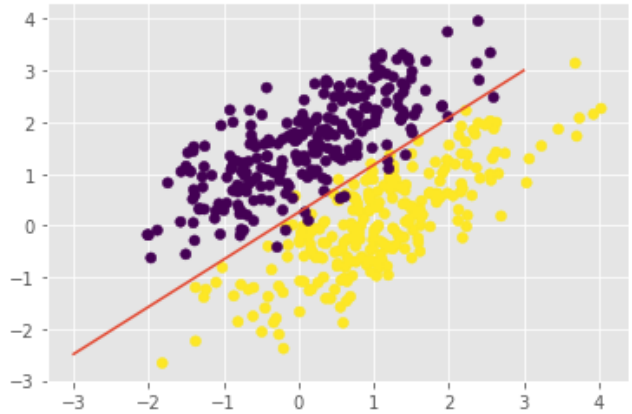
Step_Size = 0.1, L1-norm(gradiant) threshold = 0.001
Number of epochs: 98
Accuracy: 97.4
Loss Plot:



Gradient Plot:

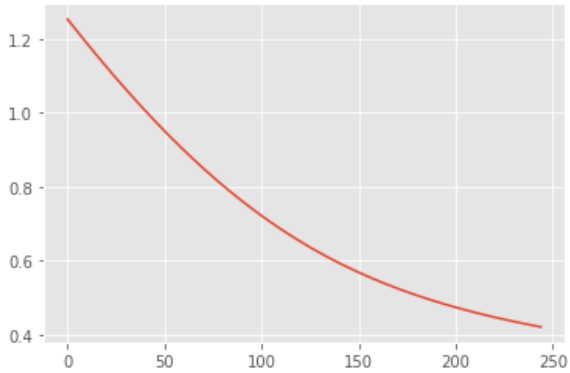


Scatter Plot:

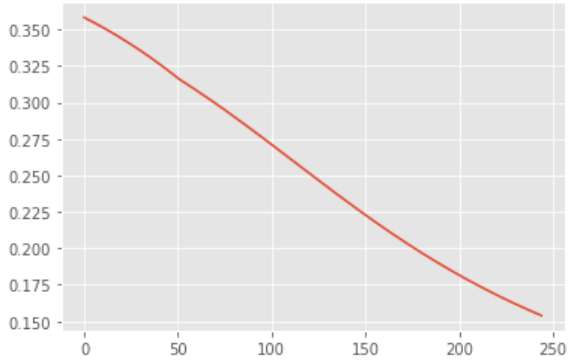


Step_Size = 0.01, L1-norm(gradiant) threshold = 0.001
Number of epochs: 244
Accuracy: 88.8

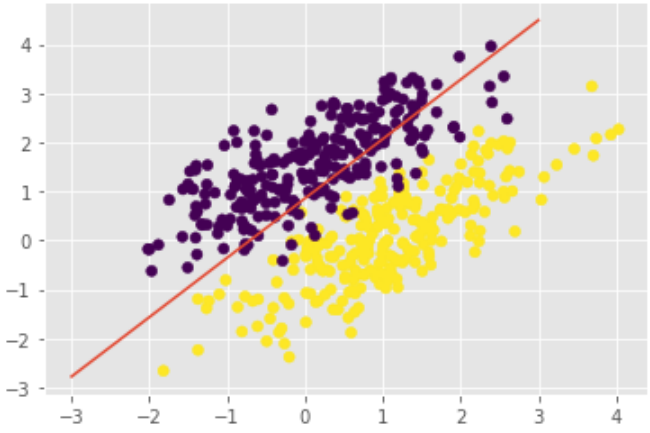
Loss Plot:



Gradient Plot:

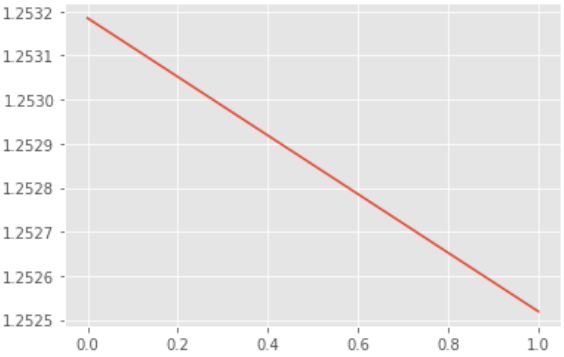


Scatter Plot:

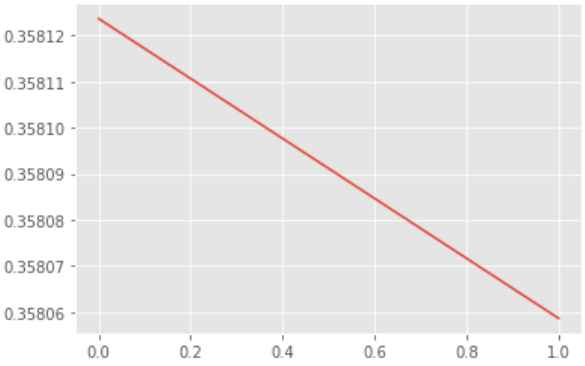


Step_Size = 0.001, L1-norm(gradiant) threshold = 0.001
Number of epochs: 1
Accuracy: 46.4
The training stops in one epoch due to the comparable values of step_size and the threshold which causes training to stop quickly.

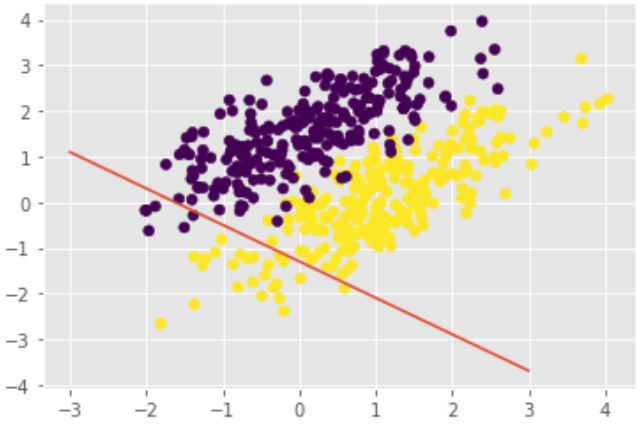
Loss Plot:



Gradient Plot:



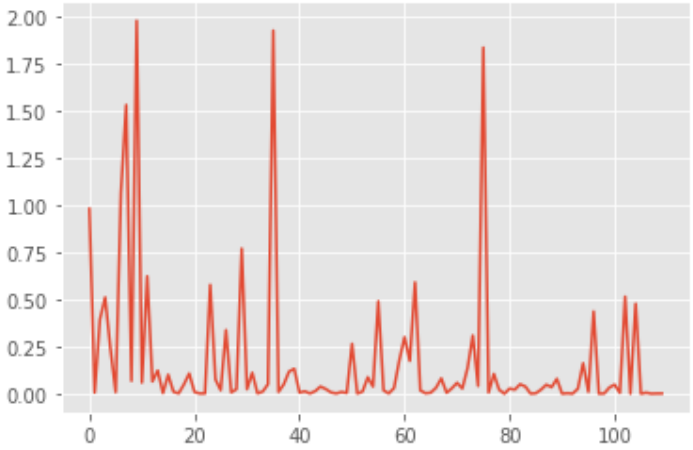
Scatter Plot:



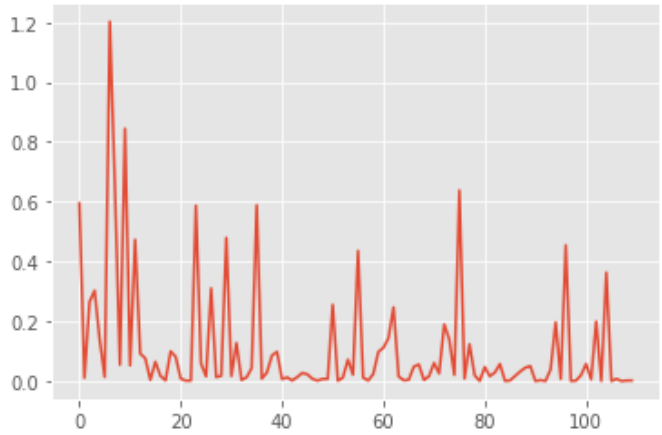
2. Perform online training using gradient descent. Here, you do not need to normalize the gradient with the training dataset size since each iteration takes one training sample at a time. Try various learning rate as η = {1,0.1,0.01,0.001}. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training loss (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number of iterations it took for training and the accuracy that you have. Write your brief observation comparing this result from the result from batch training.

Step_Size = 1, L1-norm(gradiant) threshold = 0.0001
Number of epochs: 109
Accuracy: 97.4

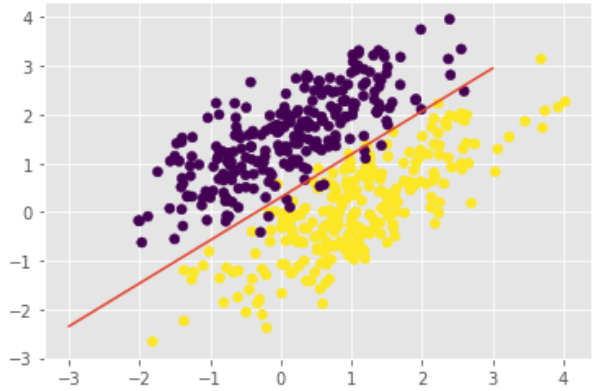
Loss Plot:



Gradient Plot:

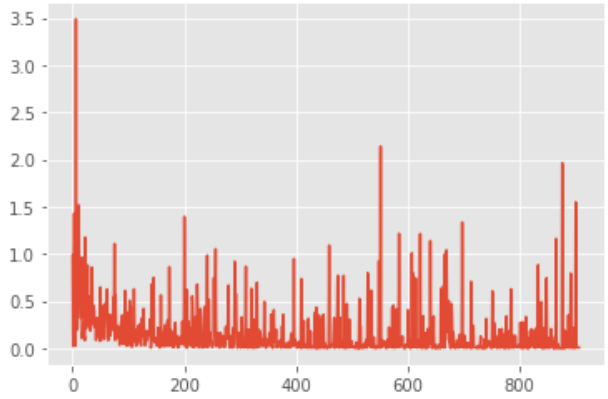


Scatter Plot:

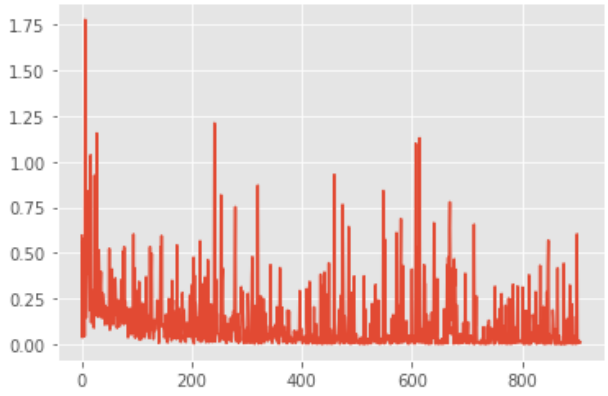


Step_Size = 0.1, L1-norm(gradiant) threshold = 0.0001
Number of epochs: 906
Accuracy: 97.4

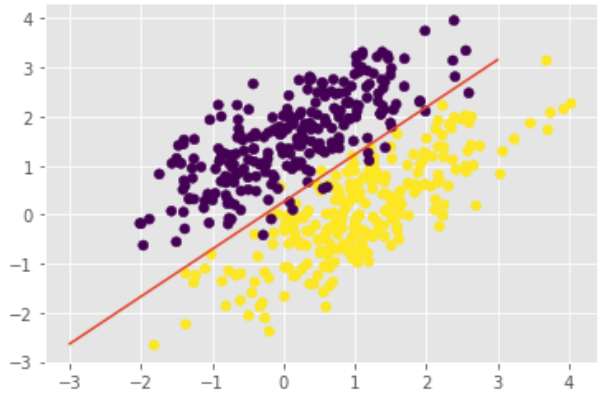
Loss Plot:



Gradient Plot:



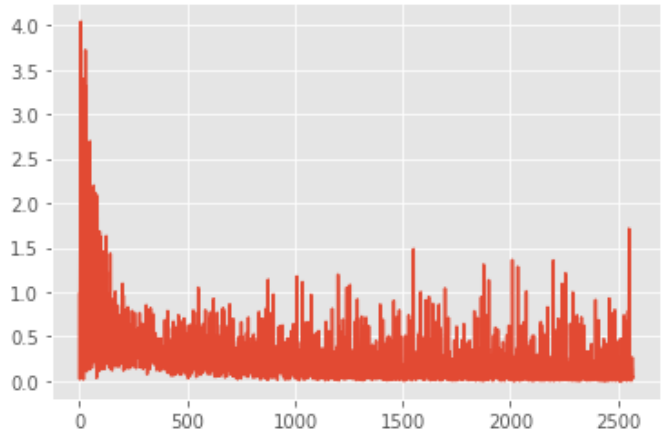
Scatter Plot:



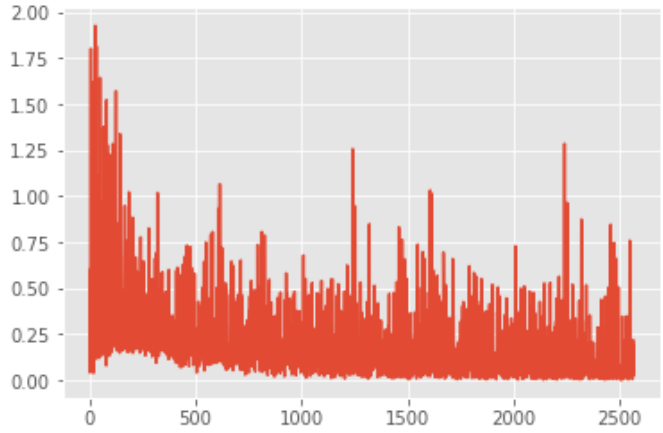
Step_Size = 0.01, L1-norm(gradiant) threshold = 0.0001
Number of epochs: 2568

Accuracy: 97.2

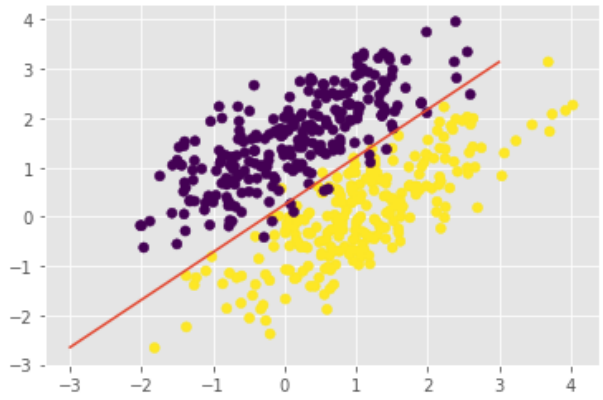
Loss Plot:



Gradient Plot:



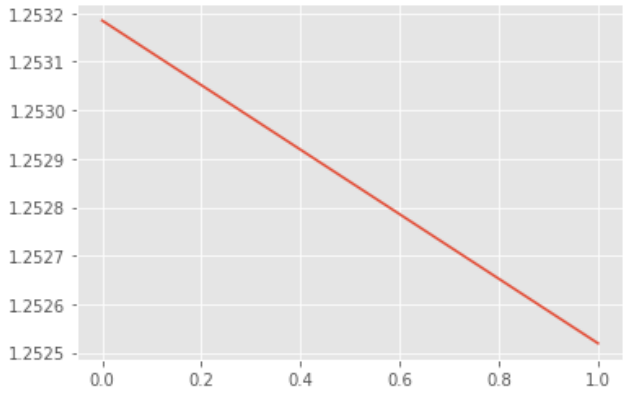
Scatter Plot:



Step_Size = 0.001, L1-norm(gradiant) threshold = 0.0001
Number of epochs: 1214

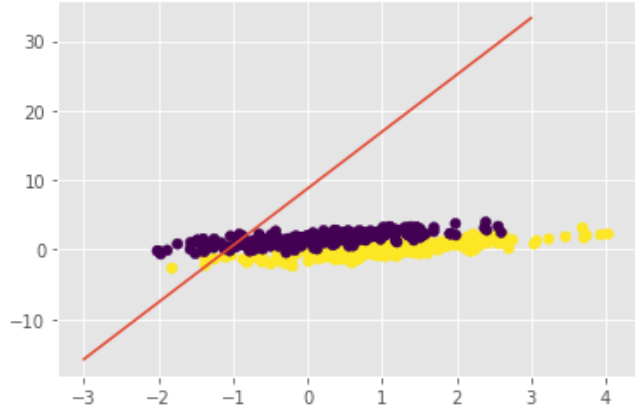
Accuracy: 55.8
The accuracy is low as the step_size is only one magnitude higher than the stopping threshold and online training is inherently noisy.

Loss Plot:



Gradient Plot:

Scatter Plot:

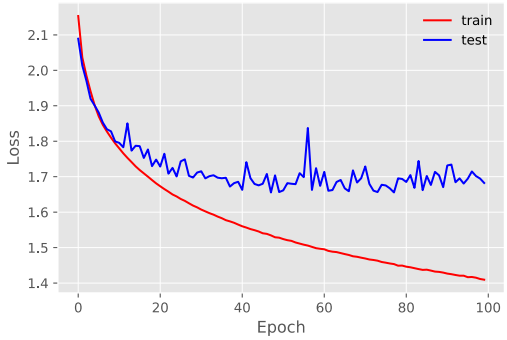


Problem 2:

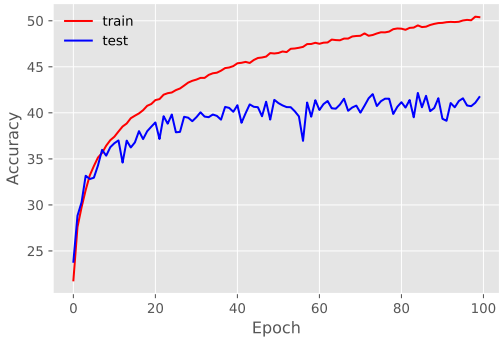
Use PyTorch to design your own Neural Network for Image Classification on CIFAR-10 dataset. Convert color images to gray scale and make sure that the pixel intensities are normalized to stay in [0,1]. We are providing a sample code (template.zip) for you to complete. Please read readme.md carefully. First, start with only one hidden layer between input and output layers in your design. Use ReLu function as your activation function and cross entropy for your objective function to minimize. Set your (mini) batch size to 32, and number of neurons in the hidden layer to 64. Use soft-max function at the output layer for pseudo probability for multiple classes to predict. Train your model until convergence (i.e., when the gradient is sufficiently small or decrease in objective function is small) or when the number of epoch reaches 100 (max epoch is there so that your ANN doesn't run forever). The criteria for convergences is up to you.
For each of the following questions, you should include:
• figure of changes of training loss, training accuracy and testing accuracy w.r.t. epoch. and your observation/comparisons/analysis.

1. Train your model on training dataset and test the trained model on testing dataset

The Loss curve:



The Accuracy curve:

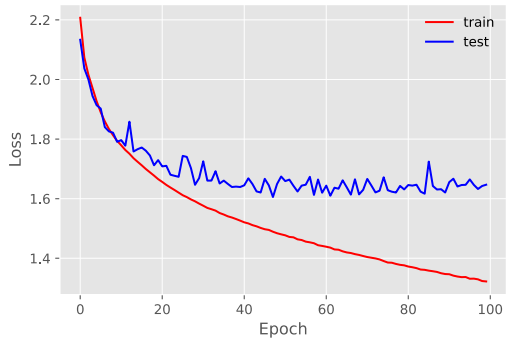


:

Number of epochs: 99
Train set: Average Loss: 1.409560 Accuracy: 50.38%
Test set: Average loss: 1.6824, Accuracy: 4171/10000 (42%)

2. Add one more hidden layer (with 64 nodes) to your network and try out your model. What has changed from the previous result by adding one more layer?

The Loss Curve:



The Accuracy curve:

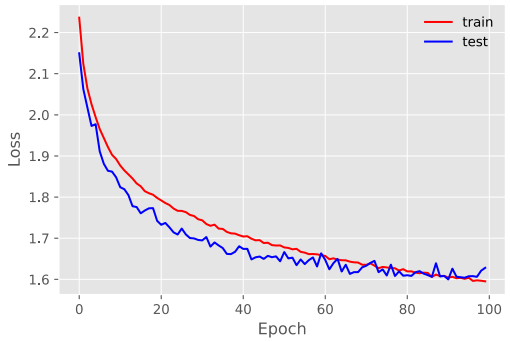
Number of epochs: 99
Train set: Loss: 1.322372 Accuracy: 53.00%
Test set: Average loss: 1.6469, Accuracy: 4334/10000 (43%)

The train set loss and test set loss has decreased, the train accuracy and test accuracy has increased.

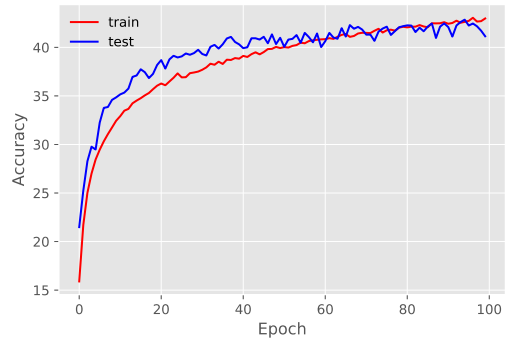
3. Try dropout at the rate of 0.2 for the hidden layers and apply your model on the dataset. What does the dropout do? What has changed compared to the previous training/results?

For Dropout= 0.2 and other parameters same as question 2:
Number of epochs: 99
Train set: Loss: 1.595257 Accuracy: 42.98%
Test set: Average loss: 1.6281, Accuracy: 4114/10000 (41%)

Loss Curve:



Accuracy Curve:



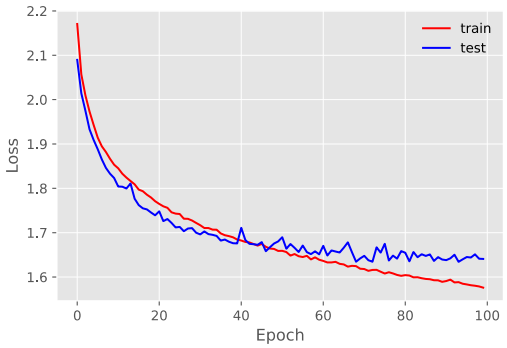
For Droput= 0.2 and parameters same as question 1:

Number of epochs: 99

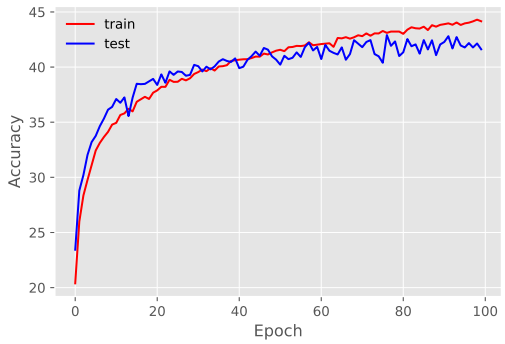
Train set: Loss: 1.576137 Accuracy: 44.15%

Test set: Average loss: 1.6407, Accuracy: 4162/10000 (42%)

Loss Curve:



Accuracy Curve:

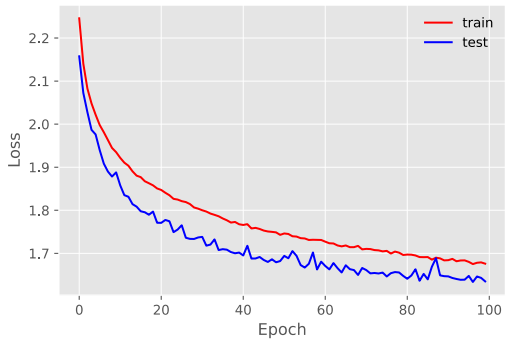


The train set loss and test set loss have increased, train set accuracy and test set accuracy have decreased with the addition of dropout.

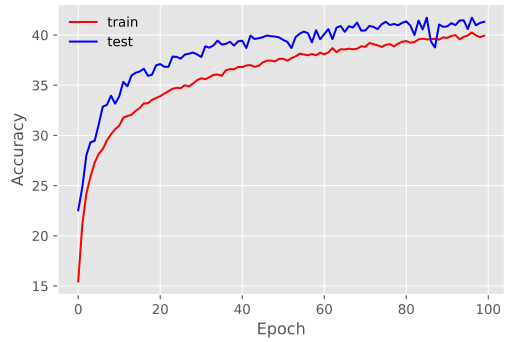
4. Now, you may freely change the architecture of the ANN (e.g., number of hidden layers, number of hidden nodes, activation function, dropout rate and other regularizers) to obtain better testing accuracy than the previous results. Try your best.

   With Dropout= 0.3 and other parameters same as question 2:
   Number of epochs: 99
   Train Set: Loss: 1.676022 Accuracy: 39.94%
   Test Set: Average loss: 1.6349, Accuracy: 4131/10000 (41%)

   Loss Curve:

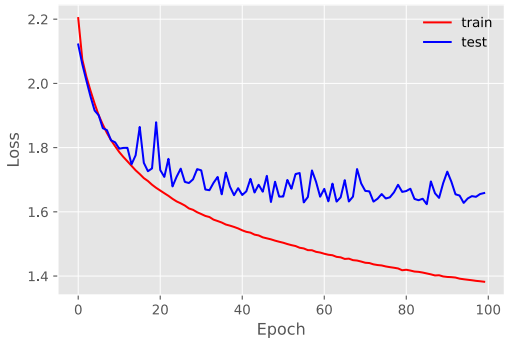


Accuracy Curve:

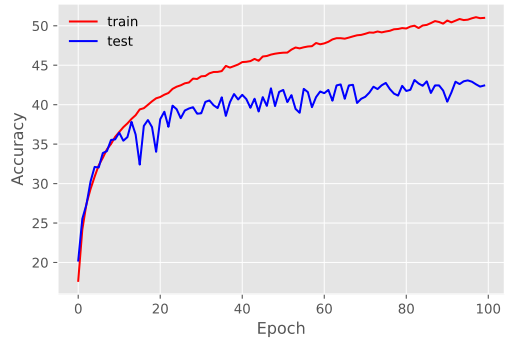


<u>With number of nodes= 50 and other parameters same as question 2:</u>
Number of epochs: 99
Train Set: Loss: 1.382377 Accuracy: 51.00%
Test Set: Average loss: 1.6585, Accuracy: 4243/10000 (42%)

Loss Curve:



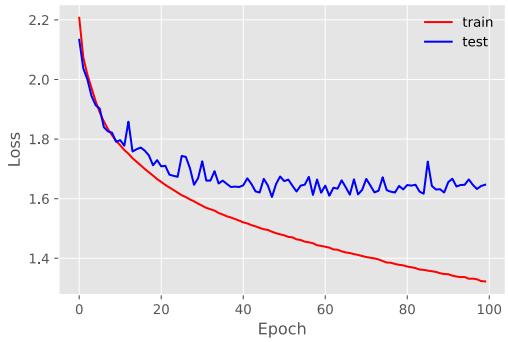
Accuracy Curve:



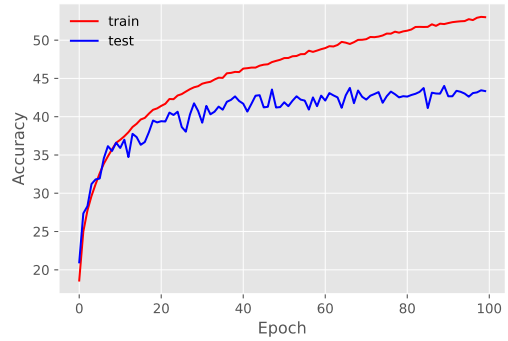
<u>When number of hidden layers = 5 and other parameters same as question 2:</u>
Number of epochs: 99
Train Set: Loss: 1.322372 Accuracy: 53.00%

Test Set: Average loss: 1.6469, Accuracy: 4334/10000 (43%)

Loss Curve:

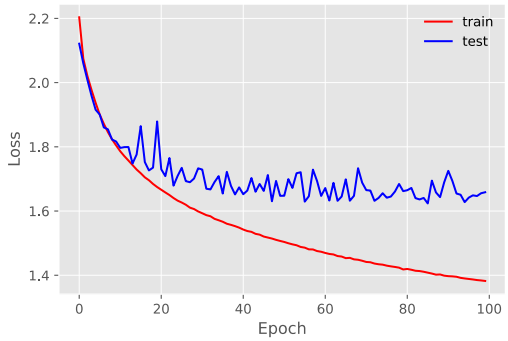


Accuracy Curve:



<u>When number of hidden layers = 5, number of nodes in hidden layer = 50 and other parameters same as question 2:</u>
Number of epochs: 99
Train Set: Loss: 1.382377 Accuracy: 51.00%
Test Set: Average loss: 1.6585, Accuracy: 4243/10000 (42%)

Loss Curve:



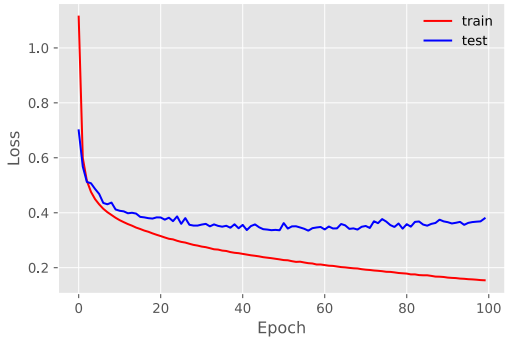
Accuracy Curve:

Thus the testing accuracy is best (43%) same as question 2 when number of hidden layers= 5

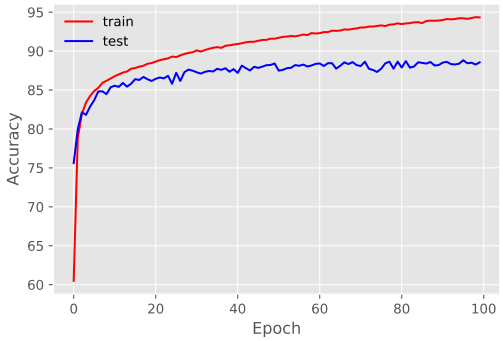
5. Let's move on to the fashion MNIST dataset. The data loader should be given in the template. Apply your ANN from P2.4 to see if it can correctly classify different classes.

Number of epochs: 99
Train Set: Loss: 0.154063 Accuracy: 94.33%
Test Set: Average loss: 0.3796, Accuracy: 8855/10000 (89%)

Loss Curve:



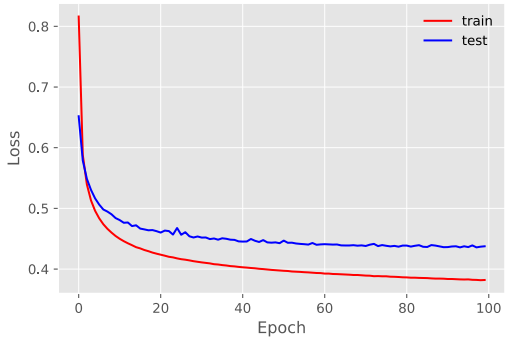
Accuracy Curve:



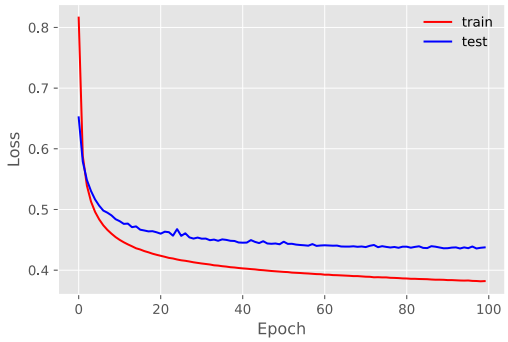
6. Now, remove all the hidden layers and train the model. The trained model contains the weights directly from input to output nodes that it has learned from training. Plot the "new representation" that it has learned for each output node. That is, reshape the learned weights (i.e., vector) to the image dimension (in 2D, i.e., 28x28) and show them as an image. Can you see any interesting shapes?

   Number of epochs: 99
   Train Set: Loss: 0.381941 Accuracy: 86.78%
   Test Set: Average loss: 0.4375, Accuracy: 8452/10000 (85%)

   Loss Curve:

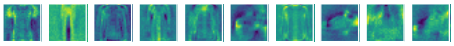


   Accuracy Curve:

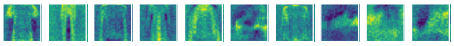


Representation from each output node:

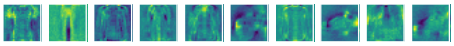
Node 0



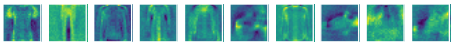
Node 5



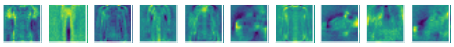
Node 10



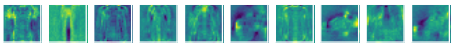
Node 15



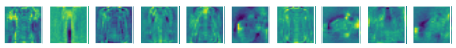
Node 20



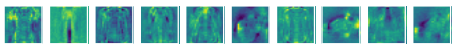
Node 25



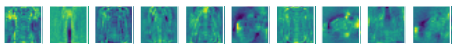
Node 30

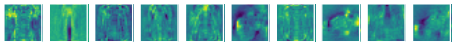


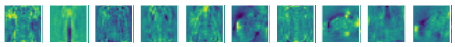
Node 35

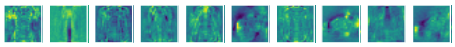


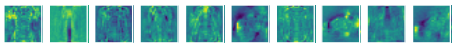
Then node 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95 in order

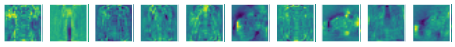


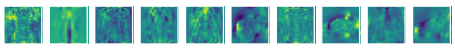


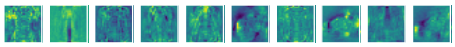


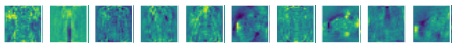


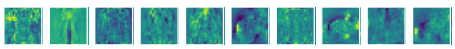


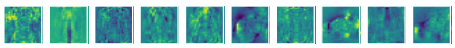


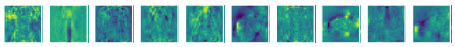












Yes. If we see closely, we can see figures of dresses in each of the small square representations.