University of Texas Arlington

Homework 3

Sundesh Raj, 1001633297

Due Date: 08 May 2020

Inferences: Following points answered

- Describing what you have solved
- What implementation has been done
- Interpretation of the results

Problem 1:

Implement logistic regression. For your training data, generate 1000 training instances in two sets of random data points (500 in each) from multi-variate normal distribution with

$$\mu_1 = [1, 0],\ \mu_2 = [0, 1.5],\ \Sigma_1 = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$$

and label them 0 and 1. Generate testing data in the same manner but include 250 instances for each class, i.e., 500 in total. You will implement a logistic regression from scratch. Use sigmoid function for your activation function and cross entropy for your objective function. Stop your training when the l1-norm of your gradient is less than 0:001 (or close to 0 based on your own threshold) or the number of iterations reaches 100000. Do not forget to include bias term!

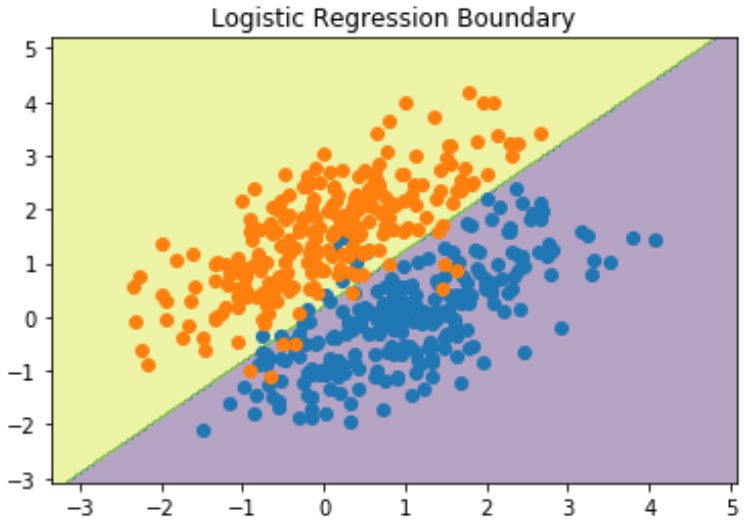
1. Perform batch training using gradient descent. Divide the derivative with the total number of training dataset as you go through iteration (it is very likely that you will get NaN if you do not do this.). Change your learning rate as n= {1; 0:1; 0:01; 0:001}. Your report should include: 1) scatter 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.

For learning rate = 1 and mode=batch

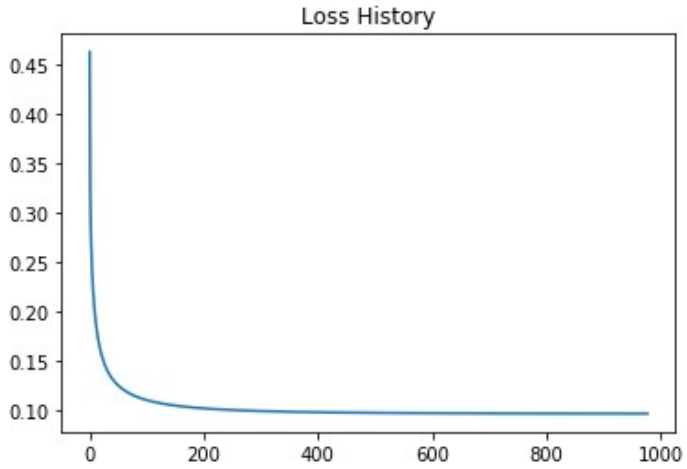
Iteration: 978/100000, Loss: 0.09655, Norm: 0.00100/0.00100
Mode: batch, LearningRate: 1, Iteration: 978, Accuracy: 0.95600

As we can see the model training stops at iteration 978. This is the point where we reach the maximum accuracy for learning rate 1, the loss here is 0.09655. The accuracy is 95.6%

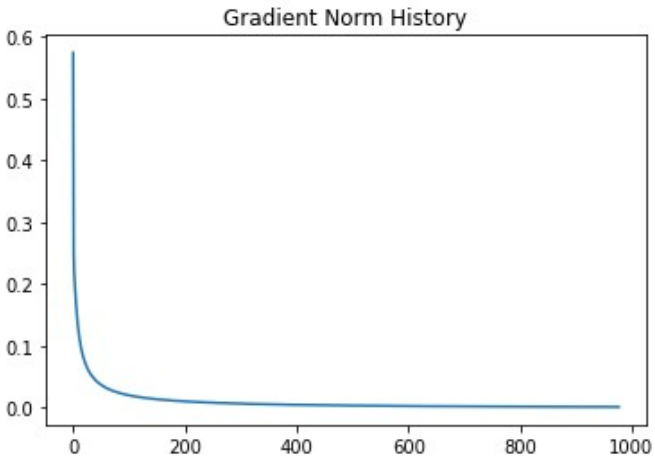
Regression Boundary :



Loss History:



Gradient Norm History:

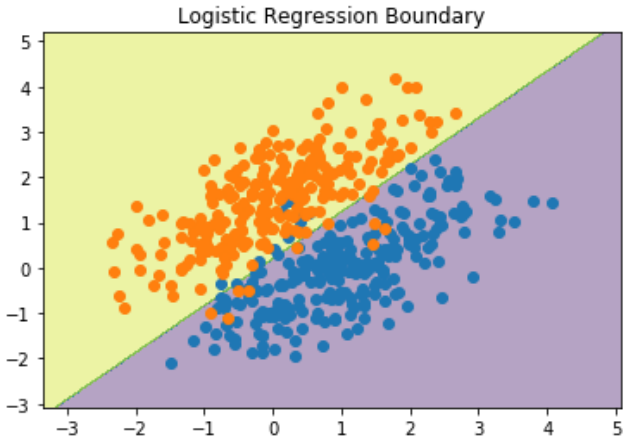


For learning rate = 0.1 and mode=batch

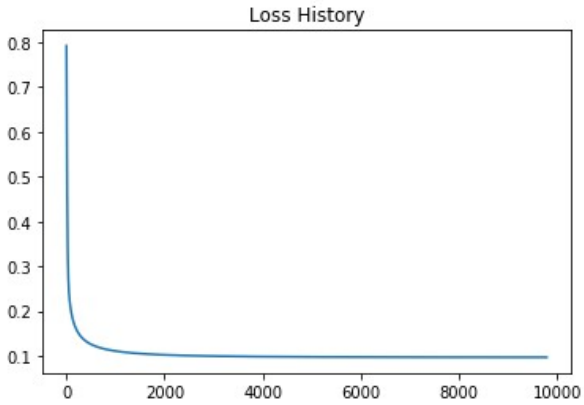
Iteration: 9787/100000, Loss: 0.09655, Norm: 0.00100/0.00100
Mode: batch, LearningRate: 0.1, Iteration: 9787, Accuracy: 0.95600

As we can see the model training stops at iteration 9787. This is the point where we reach the maximum accuracy for learning rate 0.1, the loss here is 0.09655. The accuracy is 95.6%

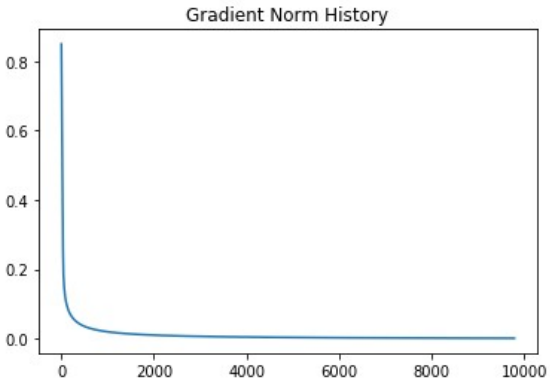
Regression Boundary:



Loss History:



Gradient Norm History:



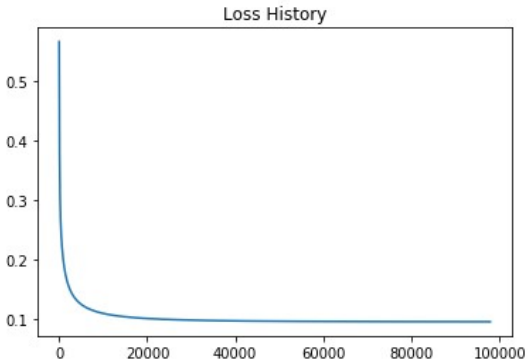
For learning rate = 0.01 and mode = batch

Iteration: 97899/100000, Loss: 0.09655, Norm: 0.00100/0.00100
Mode: batch, LR: 0.01, Iteration: 97899, Accuracy: 0.95600

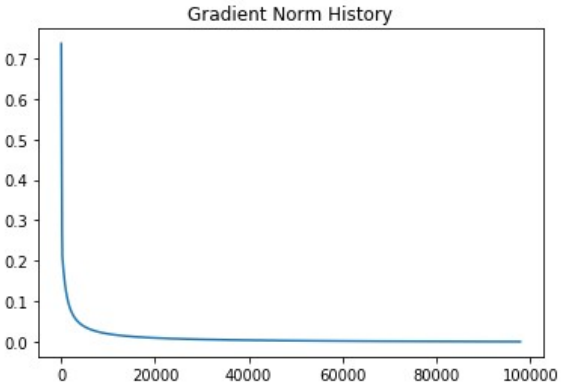
As we can see the model training stops at iteration 97899. This is the point where we reach the maximum accuracy for learning rate 0.01, the loss here is 0.09655. The accuracy is 95.6%

Regression Boundary:

Loss History:



Gradient Norm History:

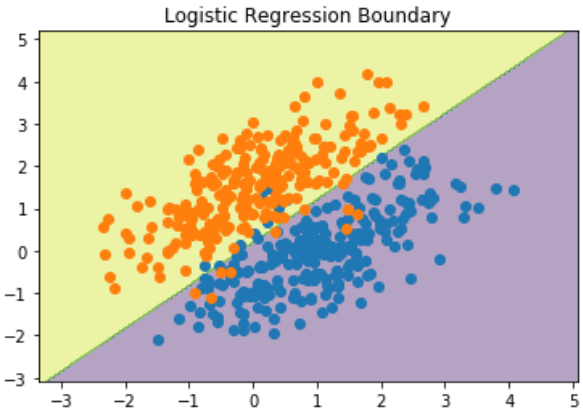


For learning rate = 0.001 and mode = batch,

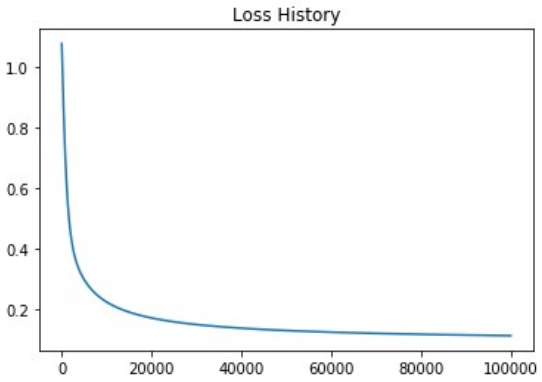
Iteration: 100000/100000, Loss: 0.11115, Norm: 0.02076/0.00100
Mode: Batch, LearningRate: 0.001, Iteration: 100000, Accuracy: 0.95600

As we can see the model training stops at iteration 100000. This is the point where we reach the maximum accuracy for learning rate 0.001, the loss here is 0.11115. The accuracy is 95.6%

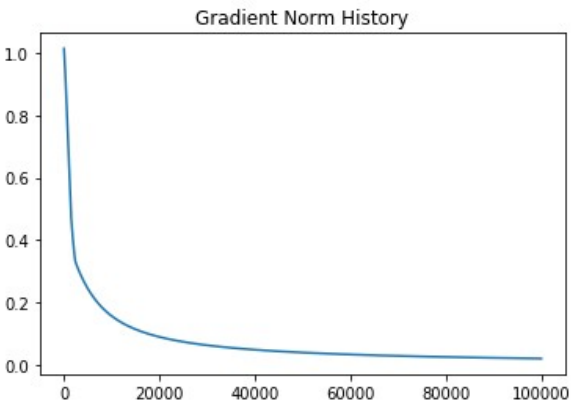
Regression Boundary:



Loss History:



Gradient Norm History:



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 n= {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.

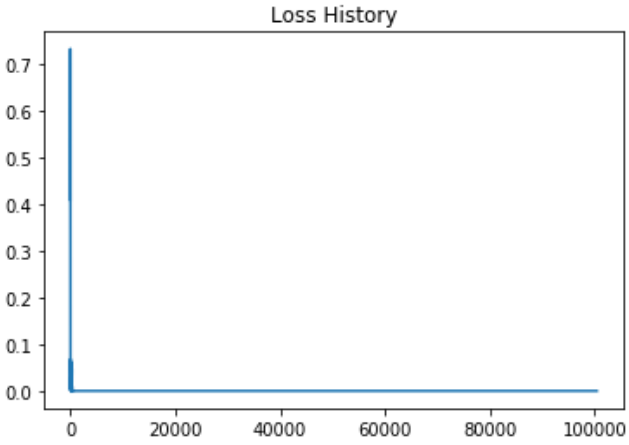
For learning rate = 1 and mode = online

Iteration: 100000/100000, Loss: 0.00033, Norm: 0.00100/0.00100
Mode: Online, LearningRate: 1, Iteration: 100000, Accuracy: 0.50600

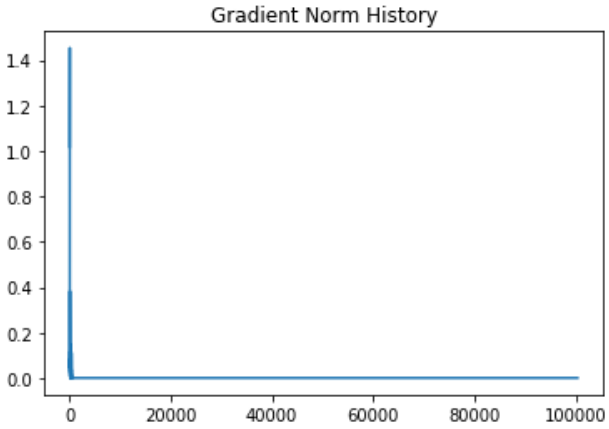
As we can see the model training stops at iteration 100000. This is the point where we reach the maximum accuracy for learning rate 1, the loss here is 0.00033. The accuracy is 50.6%

Regression Boundary:

Loss History:



Gradient Norm History:

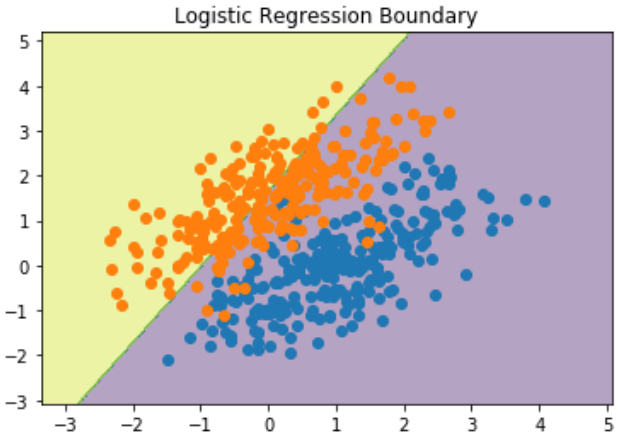


For learning rate = 0.1 and mode = online

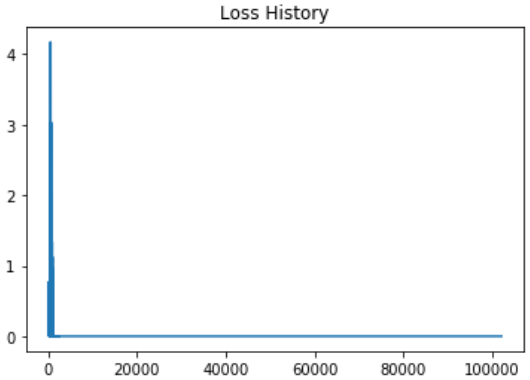
Iteration: 100000/100000, Loss: 0.00033, Norm: 0.00100/0.00100
Mode: Online, LearningRate: 0.1, Iteration: 100000, Accuracy: 0.72000

As we can see the model training stops at iteration 100000. This is the point where we reach the maximum accuracy for learning rate 0.1, the loss here is 0.00033. The accuracy is 72.0%

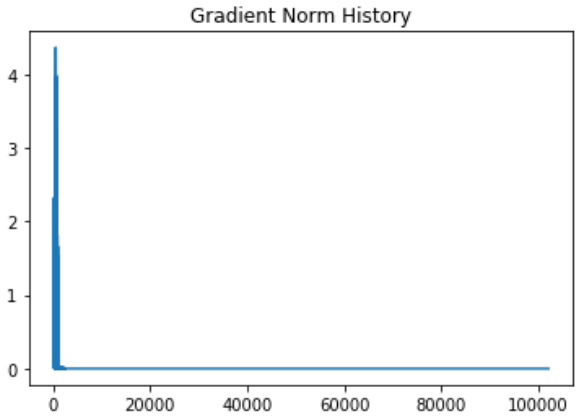
Regression Boundary:



Loss History:



Gradient Norm History:

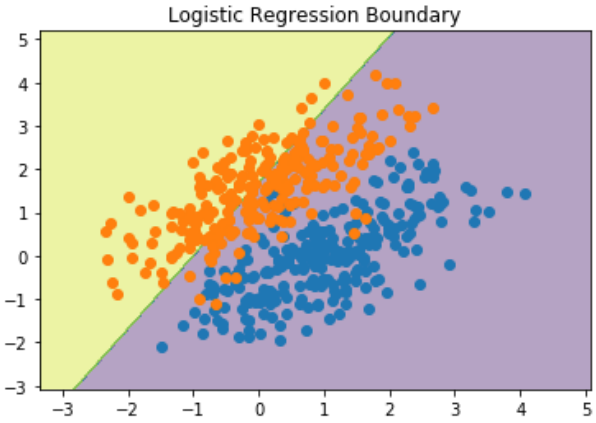


For learning rate = 0.01 and mode = online

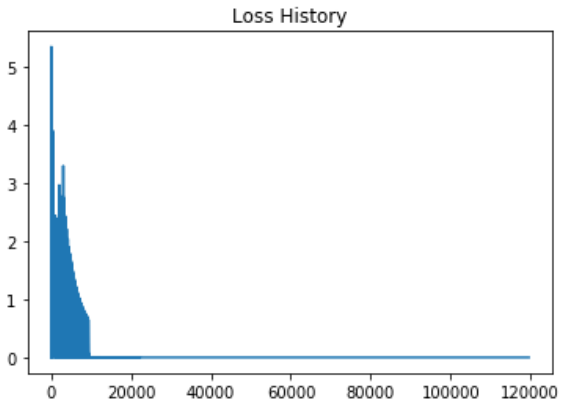
Iteration: 100000/100000, Loss: 0.00033, Norm: 0.00100/0.00100
Mode: Online, LearningRate: 0.01, Iteration: 100000, Accuracy: 0.71600

As we can see the model training stops at iteration 100000. This is the point where we reach the maximum accuracy for learning rate 0.01, the loss here is 0.00033. The accuracy is 71.6%

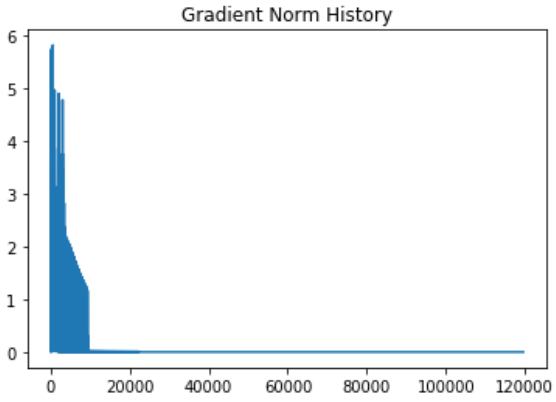
Regression Boundary:



Loss History:



Gradient Norm History:

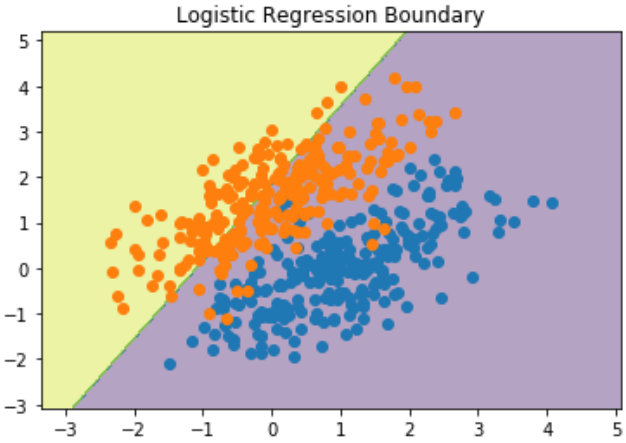


For learning rate = 0.001 and mode = online

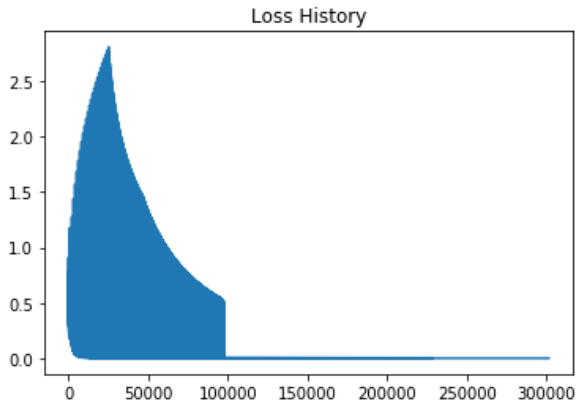
Iteration: 100000/100000, Loss: 0.00033, Norm: 0.00100/0.00100
Mode: Online, LearningRate: 0.001, Iteration: 100000, Accuracy: 0.70000

As we can see the model training stops at iteration 100000. This is the point where we reach the maximum accuracy for learning rate 0.001, the loss here is 0.00033. The accuracy is 70.0%

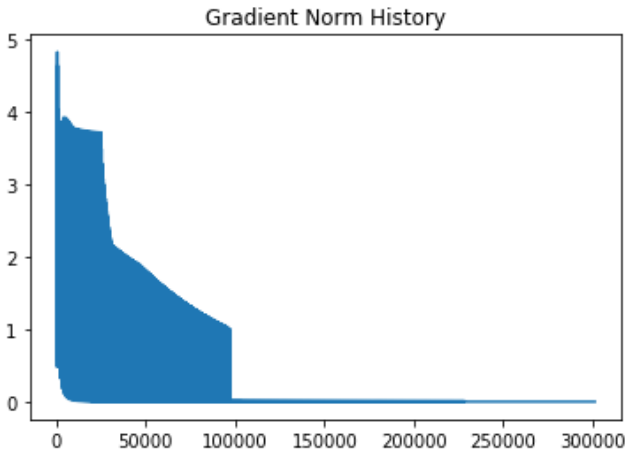
Regression Boundary:



Loss History:



Gradient Norm History:



Problem 2:

(Artificial Neural Network, 70pts) Use PyTorch to design your own Neural Network for Image Classification on CIFAR-10 dataset. Convert colour images to grey 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 epochs reaches 100 (max epoch is there so that your ANN doesn't run forever). The criteria for convergences are up to you.

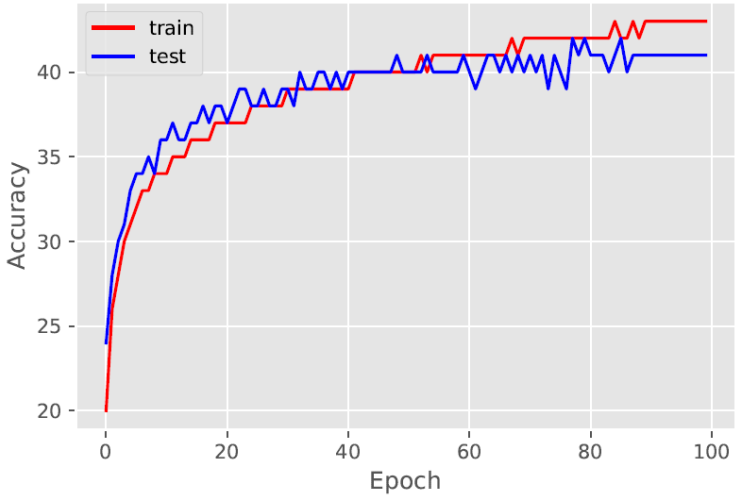
1. Train your model on training dataset and test the trained model on testing dataset.

Initial model = MLP --- multi layer perceptron
One input layer with 1024 input features
One output layer with 64 input channels and 10 output channels

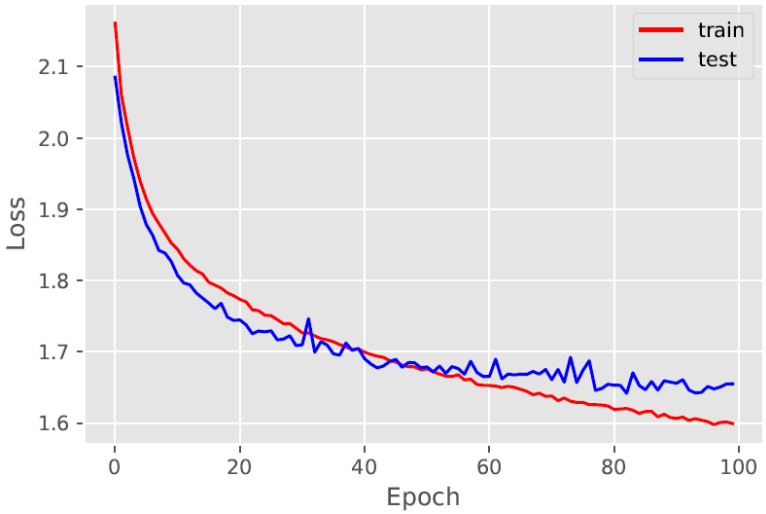
Best Outcome:

Epoch = 85
Loss = 1.6467
Accuracy = 42%

Accuracy vs Epochs:



Loss vs Epochs:



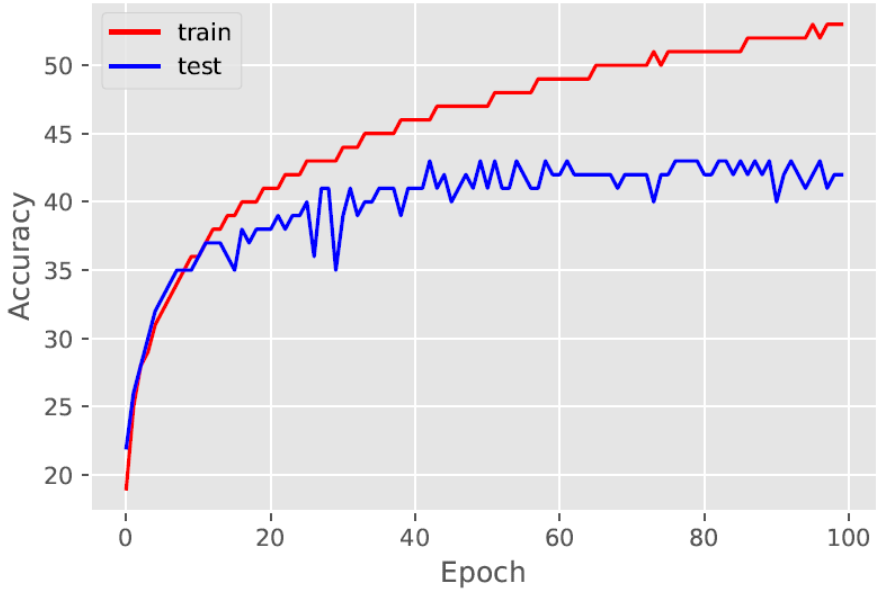
2. Add one more hidden layer (with 64 nodes) to your network and try out your model. What has changed from the previous result by adding one more layer?

Model = MLP
One input layer with 1024 input features
One hidden layer with 64 input channels and 64 output channels
One output layer with 64 input channels and 10 output channels
No Dropout

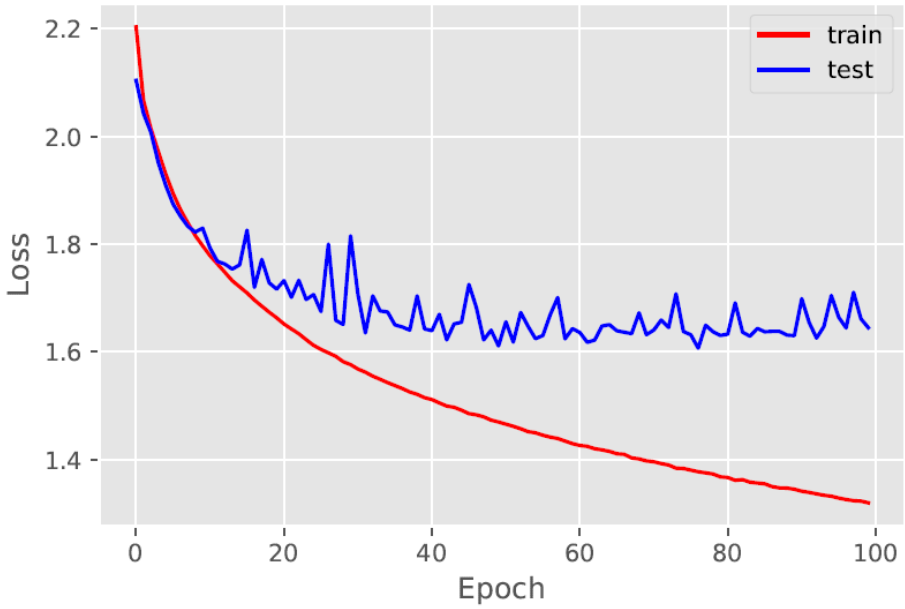
Best Outcome:

Epoch = 49
Loss = 1.6110
Accuracy = 43%

Accuracy vs Epochs:



Loss vs Epochs:



3. Try dropout at the rate of 0.2 for the hidden layers and apply your model on the dataset. What

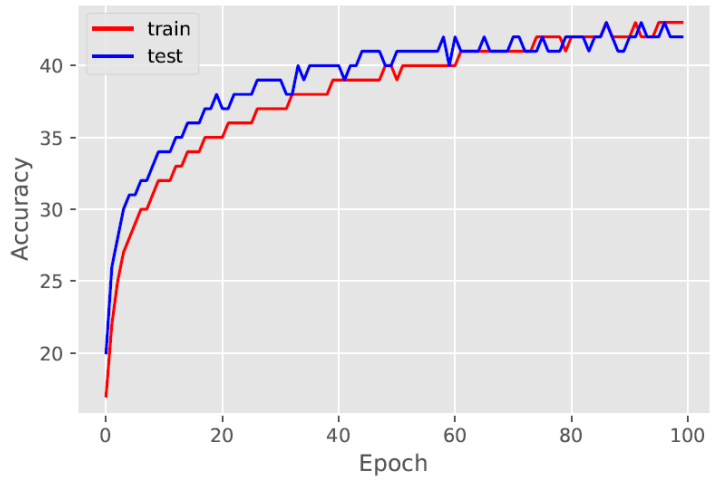
does the dropout do? What has changed compared to the previous training/results?

Model = MLP
One input layer with 1024 input features
First dropout with dp = 0.2
One hidden layer with 64 input channels and 64 output channels
Second dropout with dp = 0.2
One output layer with 64 input channels and 10 output channels

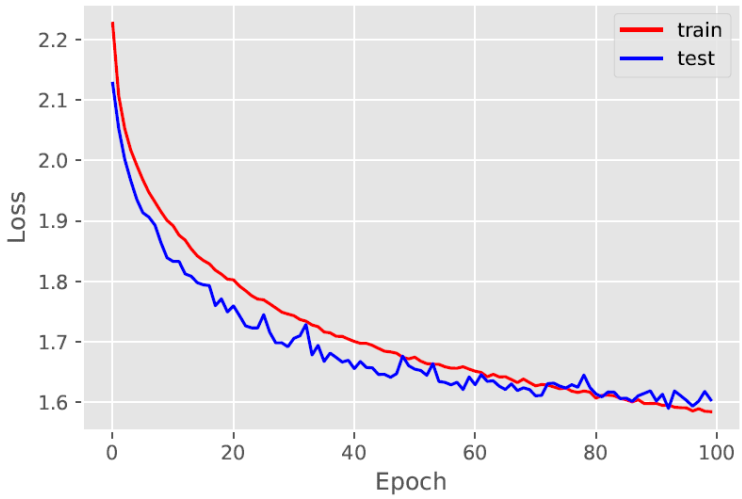
Best Outcome:

Epoch = 86
Loss = 1.6015
Accuracy = 43%

Accuracy vs Epochs:



Loss vs Epochs:



4. Now, you may freely change the architecture of the ANN (e.g., number of hidden layers,

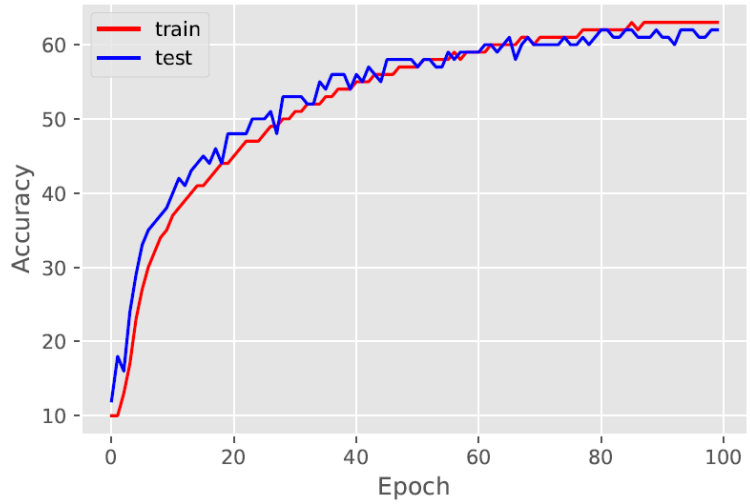
number of hidden nodes, activation function, dropout rate and other regularizers) to obtain better testing accuracy than the previous results. Try your best.

Model : CNN – Convolutional Neural Network
First layer with 1 input channel and 6 output channels
1 max pooling layer, 2nd Conv layer, one fc layer
1 dropout layer with dp = 0.5 and 2nd fc layer
2nd dropout layer and 3rd fc layer

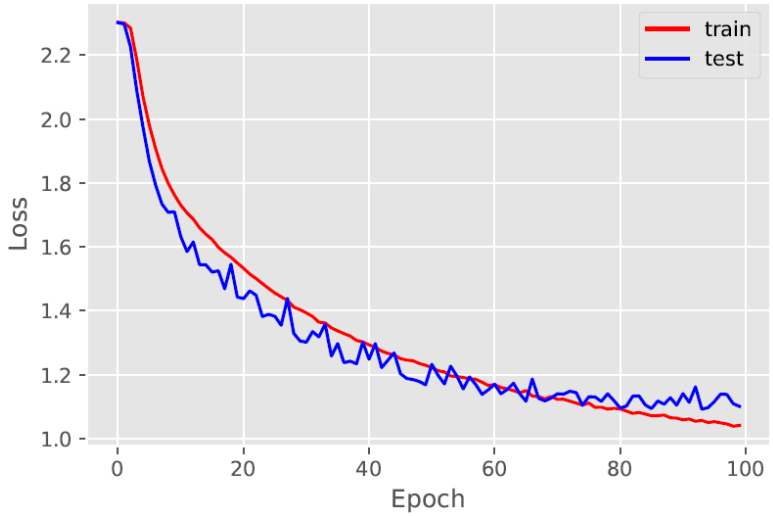
Best Outcome:

Epochs = 93
Loss = 1.0916
Accuracy = 62%

Accuracy vs Epochs:



Loss vs Epochs:



5. Let's move on to the fashion MNIST dataset. The data loader should be given in the template.

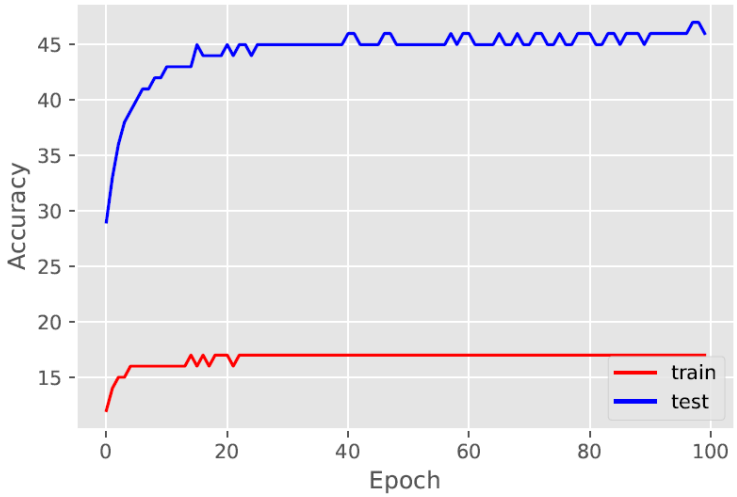
Apply your ANN from P2.4 to see if it can correctly classify different classes.

Model : CNN – Convolutional Neural Network
First layer with 1 input channel and 6 output channels
1 max pooling layer, 2nd Conv layer, one fc layer
1 dropout layer with dp = 0.5 and 2nd fc layer
2nd dropout layer and 3rd fc layer

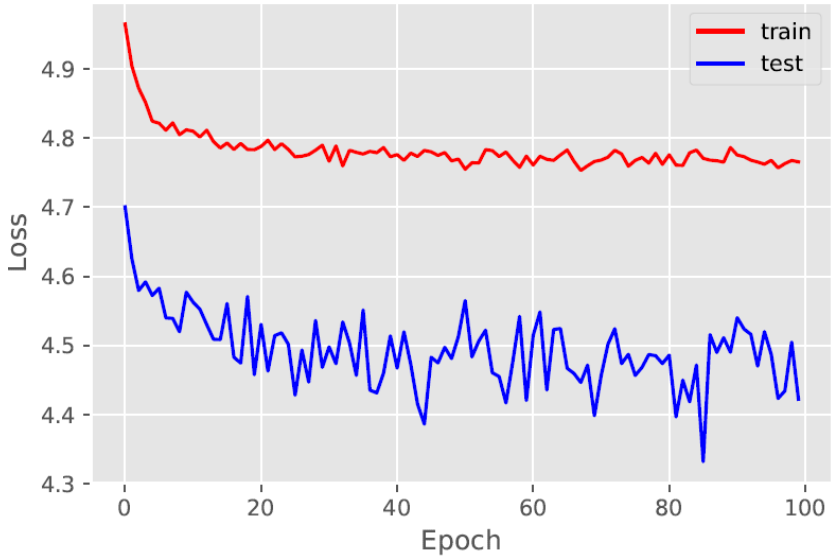
Best Outcome:

Epochs = 97
Loss = 4.4338
Accuracy = 47%

Accuracy vs Epochs:



Loss vs Epochs:

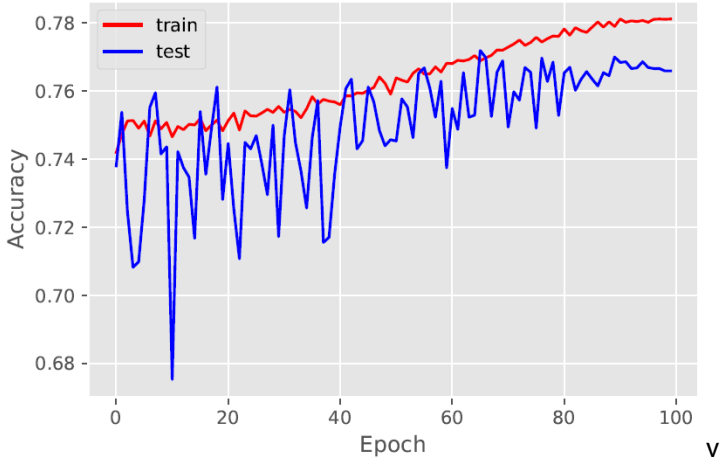


6. Now, remove all the hidden layers and train the model. The trained model contains the weights

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?

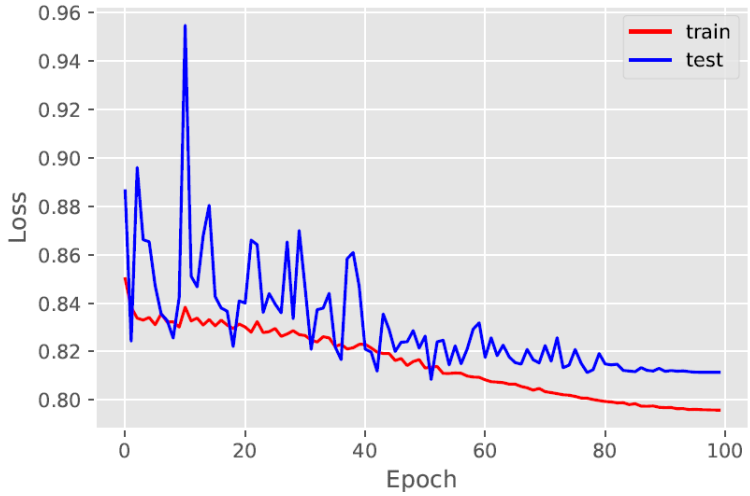
Let us see the accuracy vs epochs and loss vs epochs for this problem

Accuracy vs Epochs:

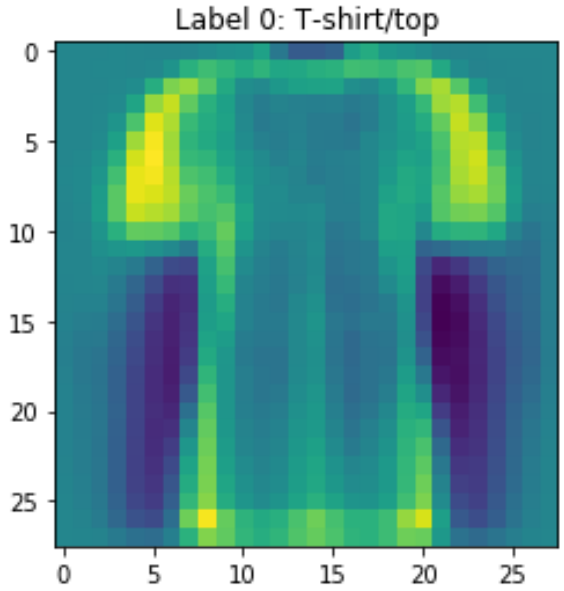


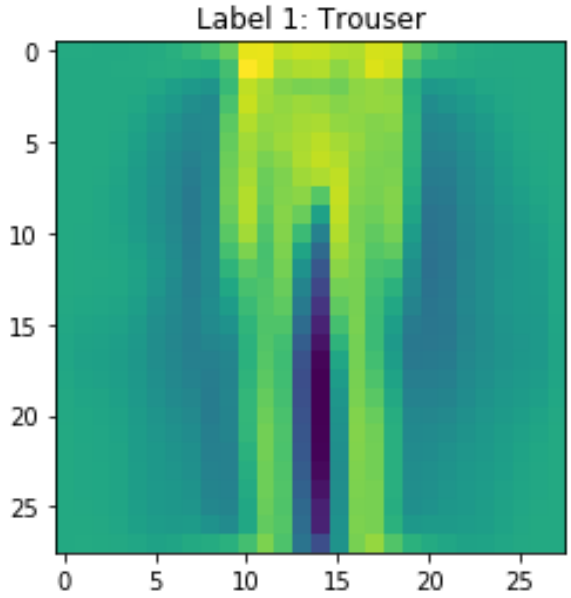
v

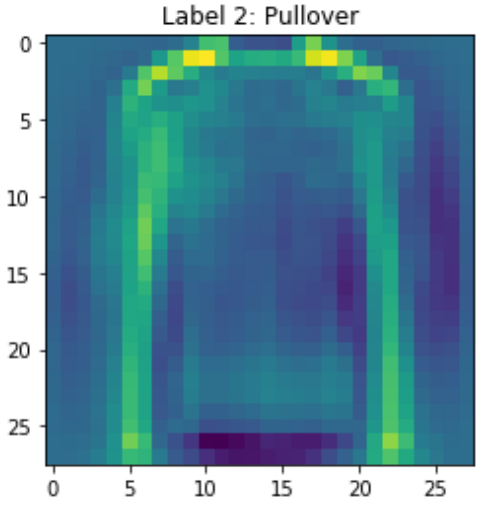
Loss vs Epochs:

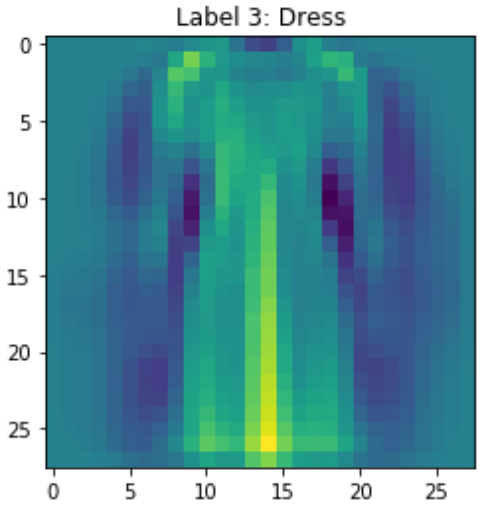


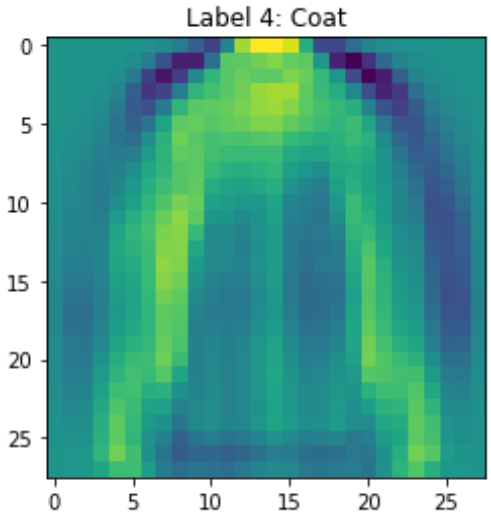
Below is the image formed from the vector of the weights acquired from the One-layer model using the fashion mnist data set

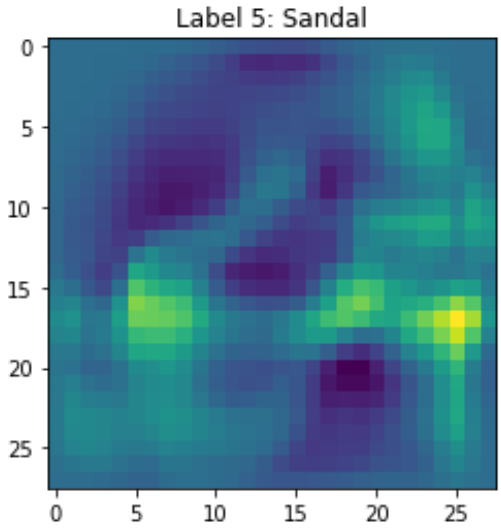

Label 0: T-shirt/top
0
5
10
15
20
25
0
5
10
15
20
25

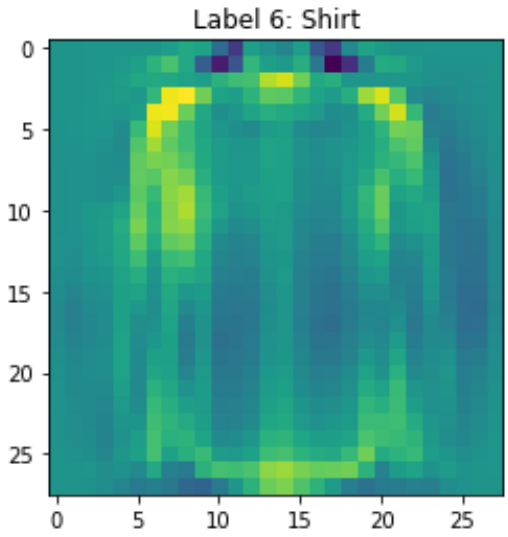

Label 1: Trouser
0
5
10
15
20
25
0
5
10
15
20
25

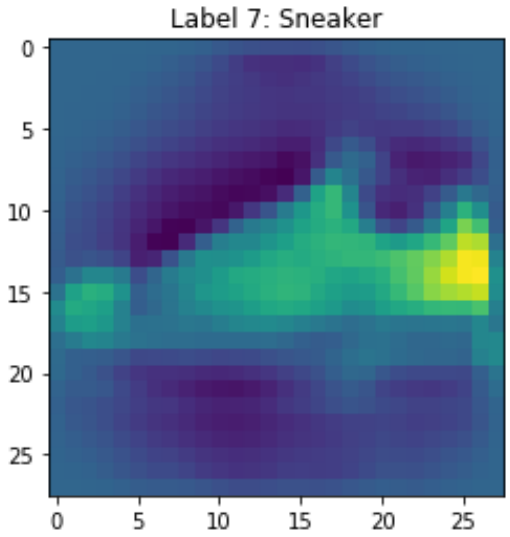

Label 2: Pullover
0
5
10
15
20
25
0
5
10
15
20
25

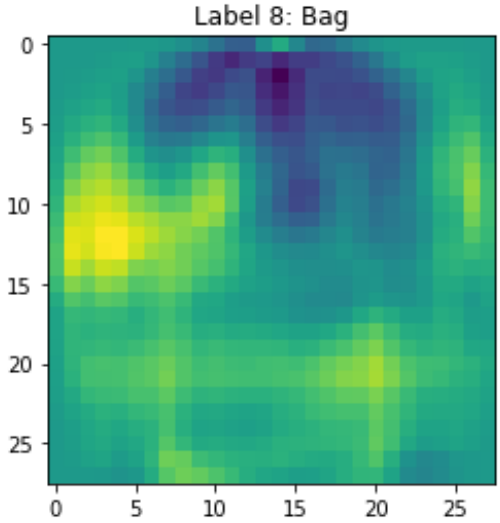

Label 3: Dress
0
5
10
15
20
25
0
5
10
15
20
25

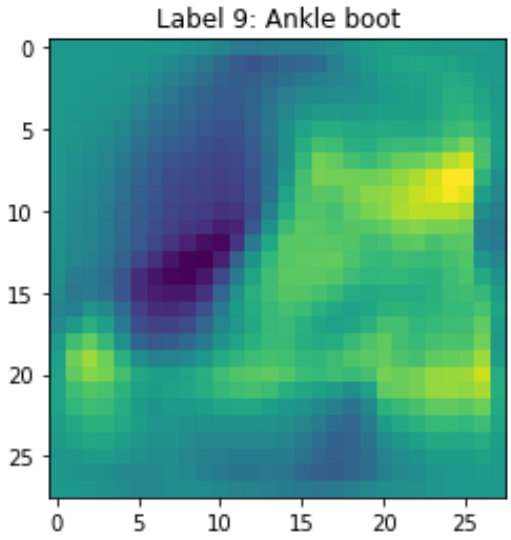

Label 4: Coat
0
5
10
15
20
25
0
5
10
15
20
25


Label 5: Sandal
0
5
10
15
20
25
0
5
10
15
20
25


Label 6: Shirt
0
5
10
15
20
25
0
5
10
15
20
25


Label 7: Sneaker
0
5
10
15
20
25
0
5
10
15
20
25


Label 8: Bag
0
5
10
15
20
25
0
5
10
15
20
25

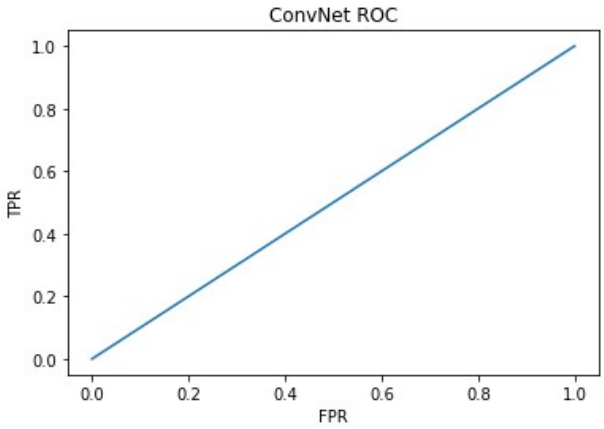


7. Select any two classes from the dataset and apply your ANN and logistic regression model from P1. Report ROC curve (you may use built-in functions) along with other results. Compare the training process / results between your ANN and Logistic regression.

We can run the plotROC.py file to get the ROC curves for Conv Net predictions and Logistic Regression predictions,

Below are the ROC curves for Conv Net and LR, respectively.

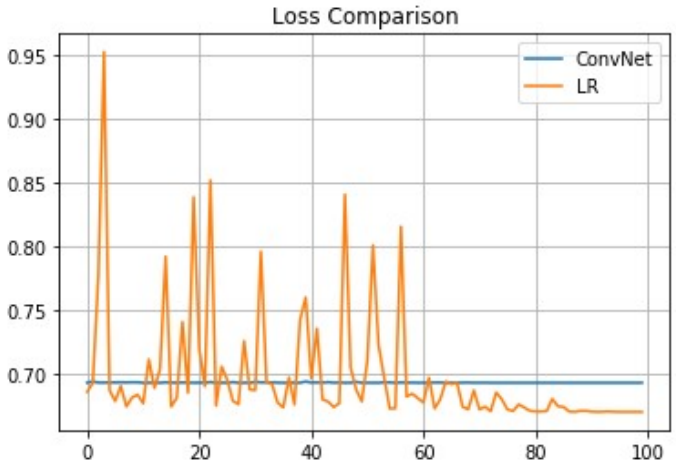
ConvNet ROC:



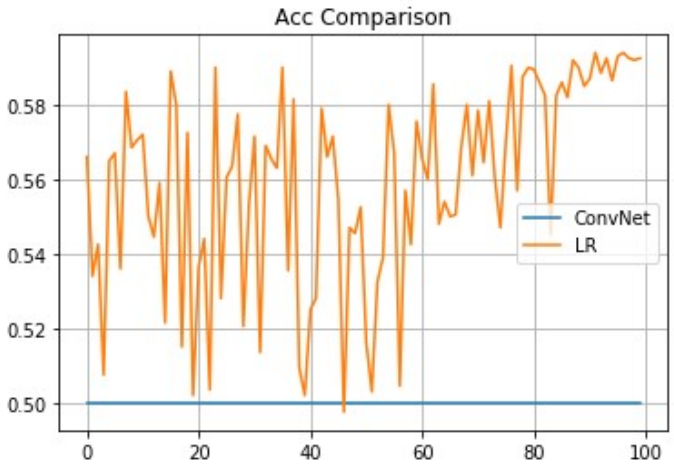
LR ROC:

With batch size 32 we perform training on LR and Conv net models and compare accuracies and loss. The compareTwoClass.py fill will provide us with the required results.

Loss Comparison:



Accuracy Comparison:



External References:

- https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
- https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
- http://parneetk.github.io/blog/cnn-cifar10/
- https://blog.tensorflow.org/2018/04/fashion-mnist-with-tfkeras.html
- https://github.com/linbo0518/CSE-5334-Data-Mining/tree/master/Assignments/Homework%203
- https://github.com/linbo0518/CSE-5334-Data-Mining/blob/master/Assignments/Homework%203/q1/logistic_regression.py
- https://github.com/linbo0518/CSE-5334-Data-Mining/blob/master/Assignments/Homework%203/q1/main.py
- https://krzysztofarendt.github.io/2019/01/29/cifar-10.html
- https://github.com/heitorrapela/fashion-mnist-mlp
- https://www.tensorflow.org/tutorials/keras/classification
- https://towardsdatascience.com/fashion-mnist-classification-with-tensorflow-featuring-deepmind-sonnet-aeb3987d826e
- https://medium.com/@ipylypenko/exploring-neural-networks-with-fashion-mnist-b0a8214b7b7b

==========

END OF FILE