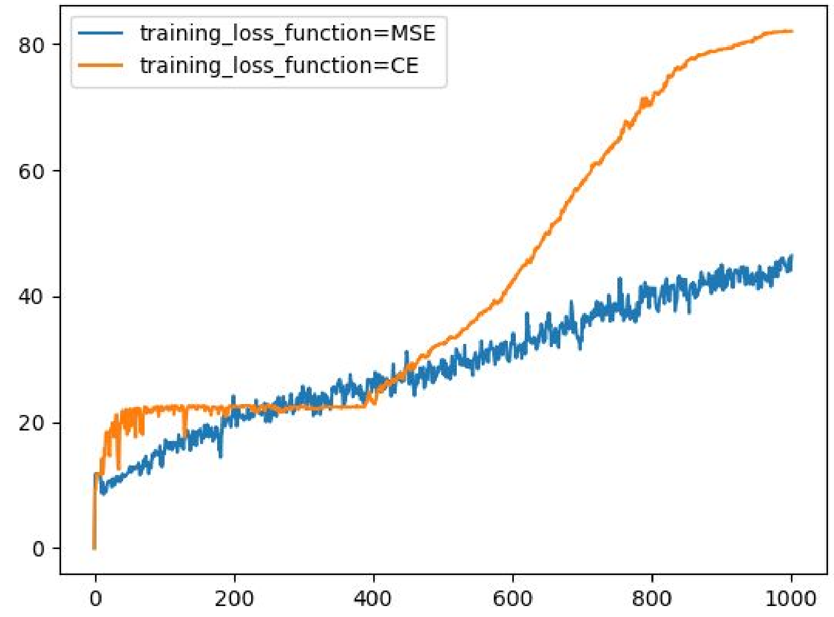
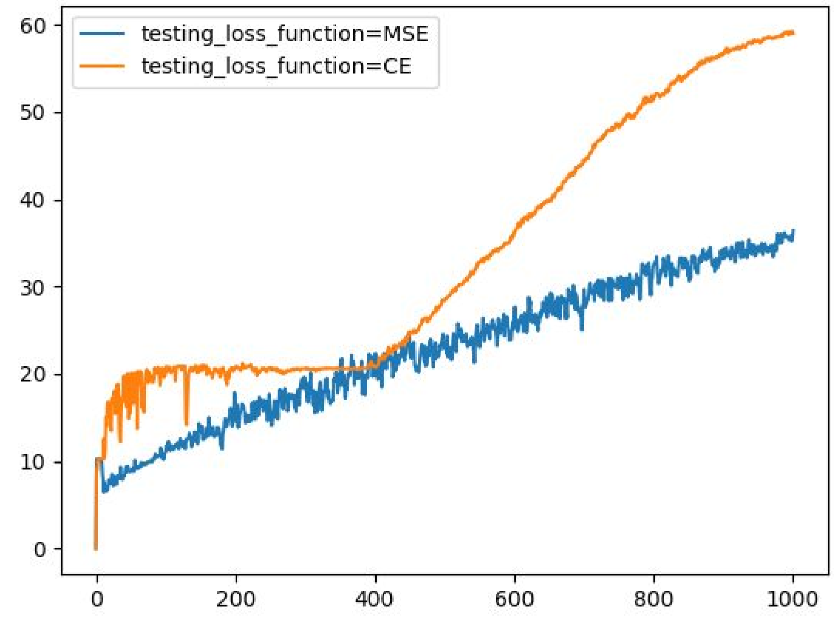
Problem 4

1. Download the python template prob4.py and read through the code which implements a neural network with PyTorch based on MNIST data. Within the provided python file is a basic scaffold of a model training workflow. You may use the existing functions in the script for both 4.1 and 4.2. Compare the squared loss and cross entropy loss. To do this,

- Finish the provided script to train the model with each of the above loss functions. (See prob4.py)
- Create a plot of the training accuracy vs epoch for each loss function (2 lines, 1 plot)



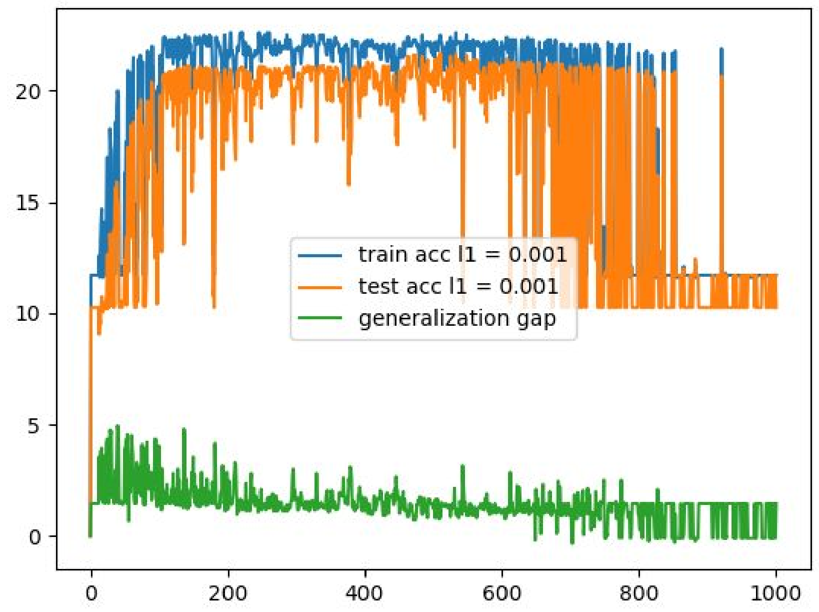
-
- Create a plot of the test accuracy vs epoch for each loss function (2 lines, 1 plot)



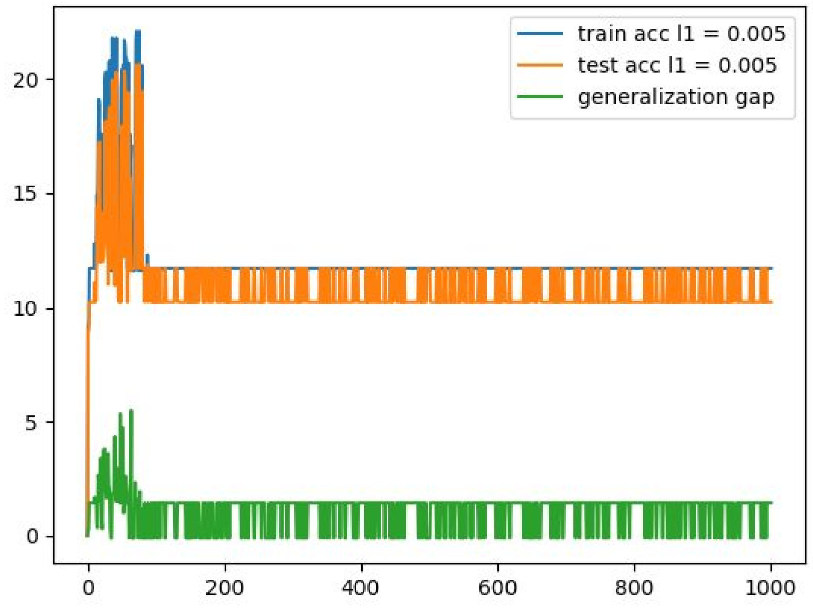
-
- Which loss function converges fastest? Which achieves the highest test accuracy? Provide some rational as to the observed differences.

Cross entropy converges faster and achieves the highest test accuracy score. This is most likely because it treats the labels as discrete as opposed to continuous like mean squared error does. Mean squared error gets better gradually and it probably would get to a good accuracy eventually, but cross entropy is able to adjust to errors much quicker, around 400 epochs is when it starts to significantly outperform mse.

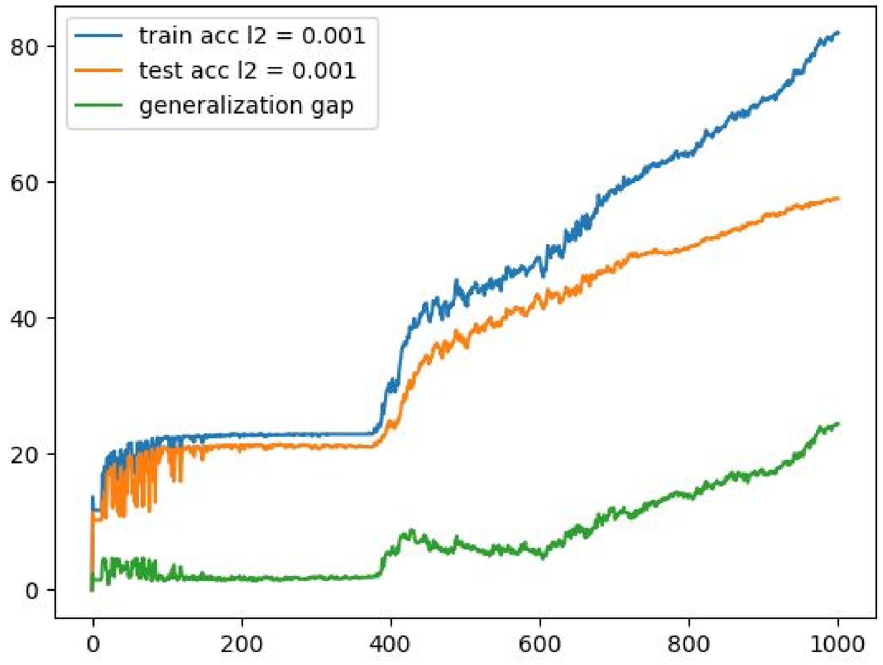
2. Using the same set-up from prob 4.1, let's now add regularization to the previous network. For the following experiments, you may use the best performing loss function from prob 4.1. Generalization gap below refers to the (train accuracy - test accuracy).
    - Implement L1 regularization and train the model using λ ∈ {0.001, 0.005}. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ (3 plots, 2 lines each).



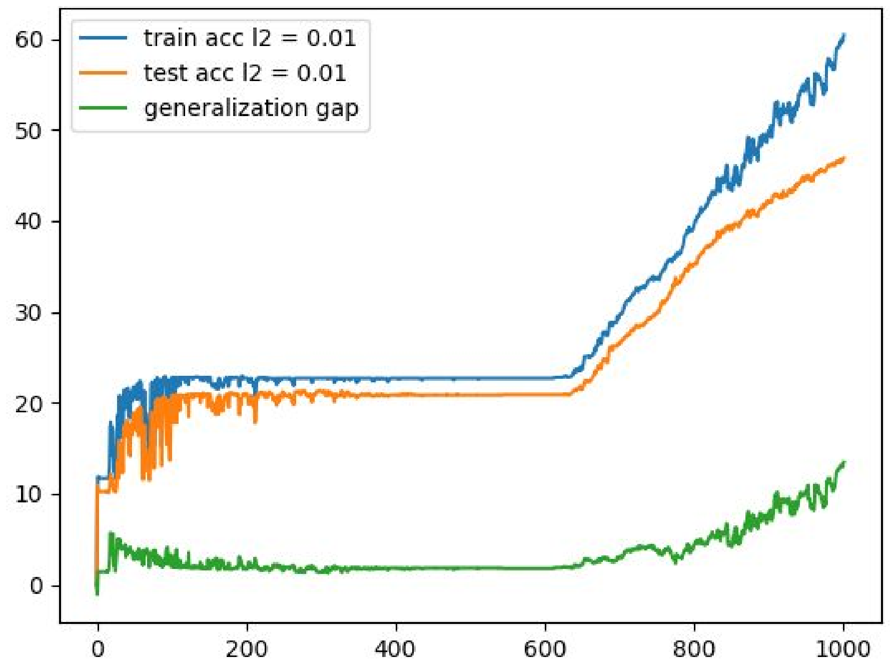
    -



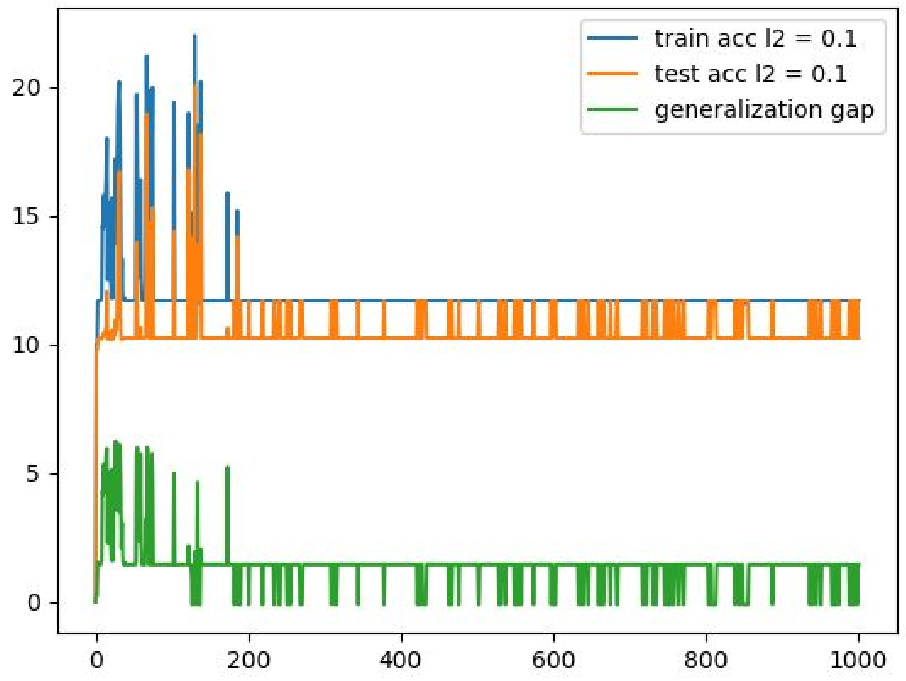
-
- Implement L2 regularization and train the model using λ ∈ {0.001, 0.01, 0.1}. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ (3 plots, 3 lines each).



-

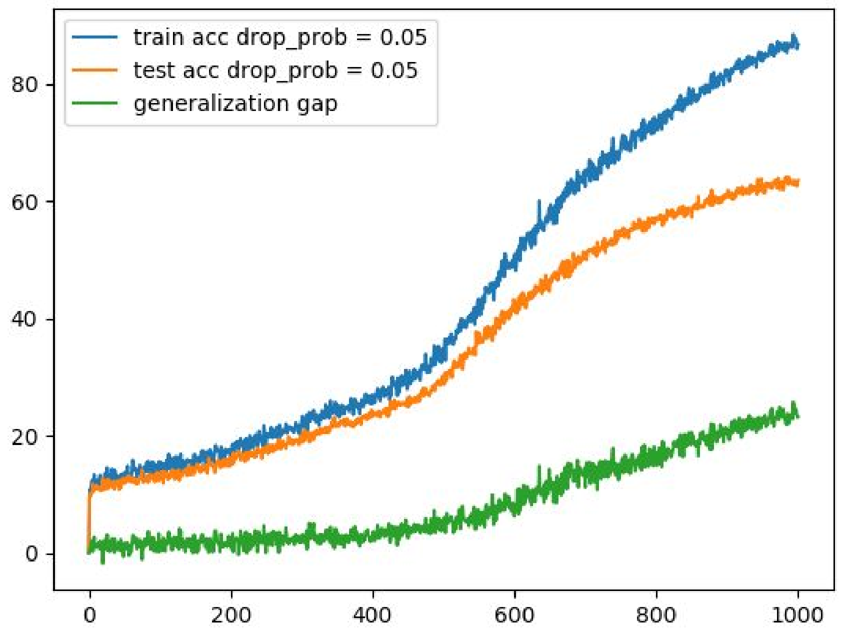


-

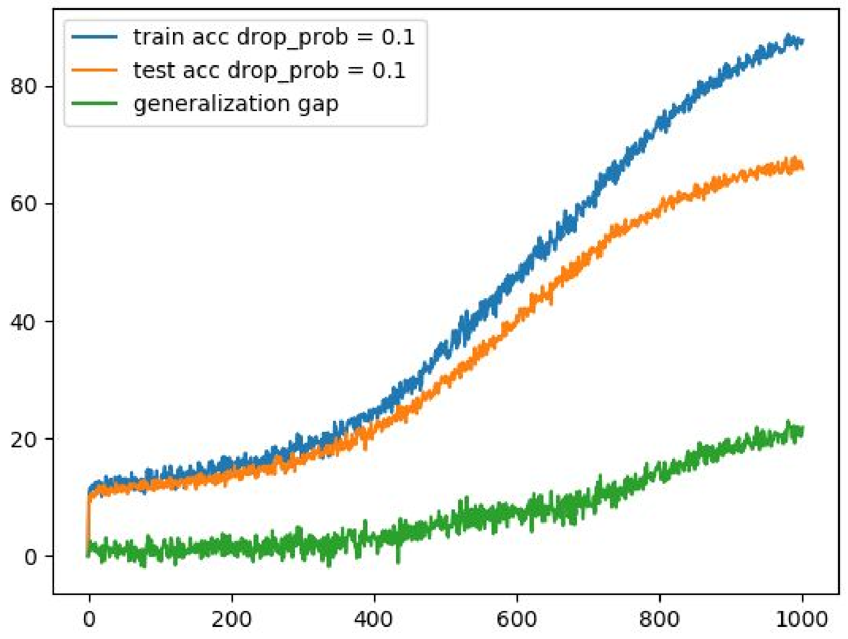


-

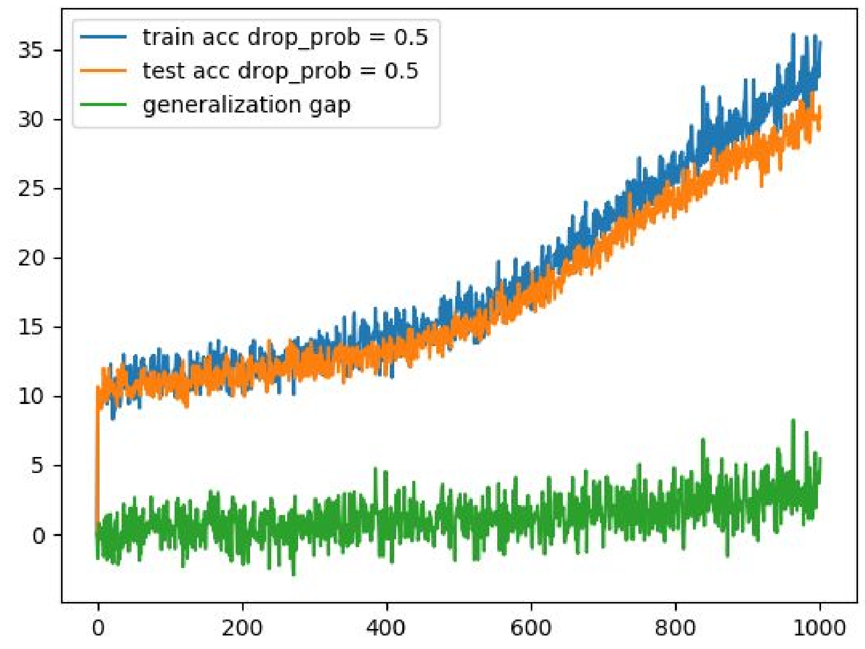
- Apply dropout to both hidden layers and train the model using p ∈ {0.05,0.1,0.5}. Hint: To implement dropout, you can use a special type of PyTorch layer included in torch.nn [2]. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each p value (3 plots, 3 lines each)


train acc drop_prob = 0.05
test acc drop_prob = 0.05
generalization gap
0
20
40
60
80
0
200
400
600
800
1000

-

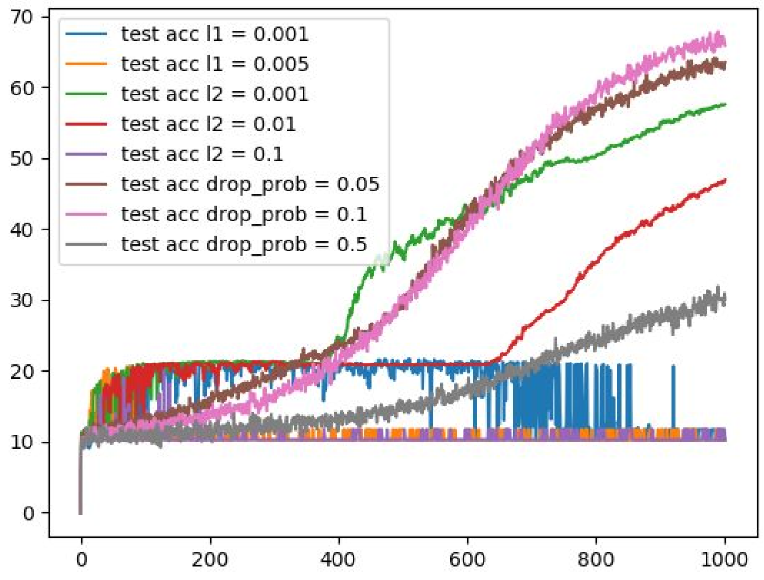

train acc drop_prob = 0.1
test acc drop_prob = 0.1
generalization gap
0
20
40
60
80
0
200
400
600
800
1000

-



-

- Using the loss data you've collected so far, create a plot of the test accuracy vs epoch for each of the experiments performed for prob 4.2. (8 lines, 1 plot)



-

- Are the final results sensitive to each parameter? Is there any regularization method which performs best?
  Dropout seems to perform consistently high, as long as the drop probability is not too high. L2 regularization also performs well, as long as the lambda isn't too high. L1 regularization performs the worst, possibly because cross entropy is being used. Even with low lambda values in the case of L1 regularization, the model seems to have a difficult time learning anything. So the results are very sensitive to each parameter, as can be seen in the plot above with very differing accuracies.