Problem 1

Comment on your hyperparameter choices. These include the learning rate and the number of epochs for training. (6 points)

For this problem I will go through my hyperparameter choices for each one of the three models.

a. **One Layer** - For this model, I set my batch-size to 32, the number of epochs to 32, and my learning rate I kept at 0.1. The reason why I went with these hyperparameter choices is because whenever I increased the number of epochs for example, my testing loss would increase dramatically. For instance, whenever I increased the number of epochs from 32 to 64, I would end up getting a testing loss of around 0.77. Additionally, if I set the number of epochs to be very low (i.e., around 2), the testing loss would also increase to be around 0.78. Similar results were found for the batch size and the learning rate parameters. Having too small of a value could lead to underfitting to the training data while having too large of a value could lead to overfitting which means that it is important to make sure that we reach a value that is not too high but also not too low at the same time.

b. **Two Layer** - For this model, I set my batch-size to 10, the number of epochs to 32, and the learning rate I also kept at 0.1. The reason why I went with these hyperparameters is a similar reason to the explanation in the previous model. However, another interesting pattern that I found is that whenever I made the hyperparameters greater, the time it took for the model to train dramatically increased. For instance, the time it took to train with a number of epochs at 32 versus when the number of epochs was 128 for example was more than a couple of seconds. Therefore, I felt like these hyperparameters was a good mix between convergence speed as well as minimizing the testing loss.

c. **CNN** - For this model, I kept the batch size at 10, the number of epochs at 7, and I kept the learning rate at 0.1. Initially, I went with a number of epochs to be around 32 like I did with my previous models, but the reason why I went with these hyperparameters is that I realized that my accuracy was reaching around 100% for my training dataset around iteration 7; therefore, it didn't make sense to keep on training on the dataset, so I was able to lower the number of epochs from 32 to 7 to avoid needlessly training on the dataset when the accuracy was already 100%. Therefore, that is why I chose the hyperparameters for this specific model.

Problem 2

For the convolutional neural network, try different settings and discuss your findings. You may try different optimizers, batch size, number of layers, kernel size, etc. (6 points)

The first parameter that I changed was the number of epochs like I mentioned previously. For the epochs, I realized that when the batch size was 10, after 7 epochs, the accuracy would cap at 100%, essentially meaning that I couldn't get an accuracy higher even with additional epochs. Therefore, in interest in decreasing the time it took to train my model, it would be best to not increase the number of epochs beyond this number because that would just be wasted resources. Additionally, another parameter that I changed was the batch size. When the batch size was too small (i.e., I set the batch size to be equal to 2), the accuracy went up and down. Specifically, when I set the batch size to 2 and the number of epochs to 7, the accuracy would jump from $73\%, 93\%, 87\%, 71\%, 52\%$ until it finally reached 19% which obviously didn't lead to the greatest accuracy. Additionally, when I made the batch size extremely high (i.e., I set it equal to 100), while I did end up getting a high accuracy, it was lower than the accuracy when I set the batch size to a reasonable number. Therefore, once again, it seems important the batch size is not too large but also not too small as well for the convolutional neural network. When it came to optimizers, I tried two different optimizers: Adam and Adadelta (these were the first ones that I saw when I typed in the letter "A"). For Adadelta, it didn't make too much difference when it came to the accuracy, but when I used the Adam optimizer, it dramatically decreased the level of my accuracy to around 13% without me changing a single hyperparameter. Therefore, through the different settings, hyperparameters, and optimizers, it seems that tuning these values have a dramatic effect on the result of the model.

Problem 3

Zhang et al. find that deep neural networks are powerful enough to easily fit random labels. Here is the conference talk given by Zhang in ICLR2017. In this paper, the authors question the measure of deep model complexity by only considering properties of the model itself. We have provided functions for you to reimplement one experiment mentioned in the paper, which is fitting randomized training labels. We will do this experiment with our CNN model and the MNIST dataset.

In test cnn() of main.py, set shuffle train label = True. Because it takes longer for the model to fit the random label, you also need to increase num epoch. If you are using the recommended CNN architecture, you can set num epoch = 280. Based on the training and testing accuracy, answer the following questions.

1. What is one possible difference between decision boundaries of datasets with shuffled

training labels and true training labels? How does shuffling training labels affect the generalization ability of the CNN model? (6 points)

First, when looking at the result of training and testing accuracy, there is a big difference when comparing the two accuracies. When I set the shuffle train label to True, it change the training accuracy to 80%, but the testing accuracy fell dramatically to 43% using the architecture set in the handout. Therefore, it seems that randomly shuffling the training labels has a negative effect on the testing accuracy specifically. The reason is that one main possible difference between the decision boundaries is that the decision boundaries of the shuffled data will be more be complex since there is just a mapping with random labeling which would lead to random decision boundaries. Therefore, due to this increased complexity with this random labeling, this would lead to a loss of generality to new datasets as the model too closely trains off the training dataset which leads to overfitting and a lower accuracy on the testing dataset. By having random decision boundaries, this is why the training accuracy is high (since the model too closely memorizes the training dataset without taking into account the real patterns that is happening the data) and the testing accuracy is low (since the model is losing generalization so it doesn't apply well to other training datasets).

2. In the bias-complexity tradeoff lecture, we learned that when the model complexity increases, the generalization ability tends to decrease. However, in this experiment with shuffled training labels, the model structure remains the same, but we notice a huge divergence between model performance on the training set and test set. Do you think the number of parameters is the only metric for measuring a deep model's complexity? If not, what other factors may contribute to model complexity? Will the 'complexity' in the bias-complexity tradeoff theory change when the dataset (or task) changes? Justify your answer. (6 points)

   I feel that the parameters are not the only metric when measuring a deep model's complexity. The reason for this has to do with my responses to the previous report questions. Not only did the accuracies change when I changed the parameters, but also when I changed the optimizer for example as well as different settings. Therefore, there are a lot of things that could affect a model's complexity which include optimizers, loss functions, the number of layers, the kernel size, etc. Therefore, it doesn't just depend on the parameters when determining the complexity of the system. With that said, I feel that the complexity of the bias-complexity tradeoff will change even when it comes to dataset as well. For instance, it doesn't make sense to have a complicated, complex model when we are dealing with a simple dataset, and it doesn't make sense to have a simple model when trying to train a dataset that is complex and has a lot of features to take into account. This means that the complexity of the model has to somewhat match the complexity of the dataset as well or else the accuracy would decrease as well.