Homework 3 [**100 pts**]

Due Wednesday, November 20 at 5pm

In this assignment, you will train two neural networks to classify the MNIST digits. I have provided starter code that loads the MNIST data and visualizes the first 9 examples in order to give you a sense of what they look like. You may do the training on your own computer or you can use Google Colab, which should have all of the python packages preinstalled, and may be faster depending on your machine.

Concretely, each digit is a 28x28 grayscale image (so there is only one color channel so the shape of an input is just 28x28). There are 60000 training examples provided, and 10000 validation/test images.

Please use Pytorch to implement your neural networks. There are a million tutorials out there on the internet that can help you learn how to use it, feel free to use any of them, but **do not** just copy-paste code you found on the internet into your assignment (and don't use LLMs to write the code either).

1 A fully connected network [55 pts]

First, you'll train a fully connected network to classify the MNIST digits. The network will be 3 layers deep, with 64 hidden units in the first layer, 32 in the second layer, and then 10 output units corresponding to the 10 digits. You can use any activation function you want. You can feel free to implement the training however you'd like, but I'll provide some general guidance that you can choose to follow:

1. As you train, you will need to generate minibatches consisting of B training image / label pairs (where B is a hyperparameter of your choice). Pytorch provides the `torch.utils.data.DataLoader` object which makes this relatively easy, but you can roll your own minibatch generator if you prefer.
2. You'll need the cross entropy loss function, which you can construct simply using the `torch.nn.CrossEntropyLoss` object.
3. You'll need to build a fully connected network model. Usually, you would implement your own Network class that inherits from the `torch.nn.Module` class, and involves `torch.nn.Linear` layers plus the activation function you choose. You'll need to implement the `__init__()` method which instantiates the `torch.nn.Linear` layers, and then you'll need to define a `forward(self, x)` method that accepts an input x and returns the output of the network.

4. You'll need to use an optimizer object to train the model. You can use any optimizer you'd like, but typical choices are `optim.SGD` or `optim.Adam`, which each require some hyperparameters like the stepsize and momentum.

5. Finally, you'll need to write a training loop, which repeatedly gets a training minibatch, clears the gradient information leftover from the previous iteration by calling `optimizer.zero_grad()`, evaluates the loss of the model output on that minibatch, calls `loss.backward()` to calculate the gradient, and then calls `optimizer.step()` to take an SGD/Adam step. (You can get more details about this by googling e.g. "pytorch training loop")

In your submission, include the following:

1. Your python code [**20 pts**]
2. Describe in about a paragraph how you trained your model. How did you choose the hyperparameters? How long did you train for and why? Did you need to change anything from your initial implementation to make it work better? Etc. [**20 pts**]
3. Report the accuracy of your model on the train and test datasets. [**10 pts**]
4. How many parameters are in your model? [**5 pts**]

2 A convolutional neural network [45 pts]

Now, we will train a convolutional neural network to classify the MNIST digits. Train a 6 layer network consisting of 4 convolutional layers followed by 2 fully-connected layers. You may choose whatever specific architecture you'd like, which means choosing the following details:

1. The number and size of the filters in each convolutional layer, and the stride and padding used
2. Whether to use max-/average-pooling between layers, and what activation function to use
3. The number of hidden units in the final 2 fully connected layers (note: there needs to be exactly 10 output units, one for each possible digit)

If you did things correctly in the first problem, you should be able to reuse almost all of your code here. The only thing that needs to change is your model definition. This is the benefit of Pytorch, you can swap out one model for another very easily!

In your submission, include the following:

1. Your python code [**10 pts**]
2. Describe the model architecture you used in about a paragraph. [**10 pts**]
3. Describe in about a paragraph how you trained your model. How did you choose the hyperparameters? How long did you train for and why? Did you need to change anything from your initial implementation to make it work better? Etc. [**10 pts**]

4. Report the accuracy of your model on the train and test datasets. [**10 pts**]
5. How many parameters are in your model? [**5 pts**]