Problem 3

Use this notebook to write your code for problem 3.

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

3D - Convolutional network

As in problem 2, we have conveniently provided for your use code that loads and preprocesses the MNIST data.

In [5]:
```
# load MNIST data into PyTorch format
import torch
import torchvision
import torchvision.transforms as transforms

# set batch size
batch_size = 32

# load training data downloaded into data/ folder
mnist_training_data = torchvision.datasets.MNIST('data/', train=True, download=Tru
e,
                                                 transform=transforms.ToTensor())
# transforms.ToTensor() converts batch of images to 4-D tensor and normalizes 0-25
5 to 0-1.0
training_data_loader = torch.utils.data.DataLoader(mnist_training_data,
                                                   batch_size=batch_size,
                                                   shuffle=True)

# load test data
mnist_test_data = torchvision.datasets.MNIST('data/', train=False, download=True,
                                             transform=transforms.ToTensor())
test_data_loader = torch.utils.data.DataLoader(mnist_test_data,
                                               batch_size=batch_size,
                                               shuffle=False)
```

In [6]:
```
# look at the number of batches per epoch for training and validation
print(f'{len(training_data_loader)} training batches')
print(f'{len(training_data_loader) * batch_size} training samples')
print(f'{len(test_data_loader)} validation batches')
```

```
1875 training batches
60000 training samples
313 validation batches
```

In [29]:
```
# sample model
import torch.nn as nn

class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(7 * 7 * 64, 1000)
        self.fc2 = nn.Linear(1000, 10)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.drop_out(out)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

model = ConvNet()
```

In [30]:
```
# why don't we take a look at the shape of the weights for each layer
for p in model.parameters():
    print(p.data.shape)
```

```
torch.Size([32, 1, 5, 5])
torch.Size([32])
torch.Size([64, 32, 5, 5])
torch.Size([64])
torch.Size([1000, 3136])
torch.Size([1000])
torch.Size([10, 1000])
torch.Size([10])
```

In [31]:
```
# our model has some # of parameters:
count = 0
for p in model.parameters():
    n_params = np.prod(list(p.data.shape)).item()
    count += n_params
print(f'total params: {count}')
```

```
total params: 3199106
```

In [32]:
```
# For a multi-class classification problem
#import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=.001)
```

In [33]:
```
# Train the model for 10 epochs, iterating on the data in batches
n_epochs = 10

# store metrics
training_accuracy_history = np.zeros([n_epochs, 1])
training_loss_history = np.zeros([n_epochs, 1])
validation_accuracy_history = np.zeros([n_epochs, 1])
validation_loss_history = np.zeros([n_epochs, 1])

for epoch in range(n_epochs):
    print(f'Epoch {epoch+1}/10:', end='')
    train_total = 0
    train_correct = 0
    # train
    model.train()
    for i, data in enumerate(training_data_loader):
        images, labels = data
        optimizer.zero_grad()
        # forward pass
        output = model(images)
        # calculate categorical cross entropy loss
        loss = criterion(output, labels)
        # backward pass
        loss.backward()
        optimizer.step()

        # track training accuracy
        _, predicted = torch.max(output.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()
        # track training loss
        training_loss_history[epoch] += loss.item()
        # progress update after 180 batches (~1/10 epoch for batch size 32)
        if i % 180 == 0: print('.',end='')
    training_loss_history[epoch] /= len(training_data_loader)
    training_accuracy_history[epoch] = train_correct / train_total
    print(f'\n\tloss: {training_loss_history[epoch,0]:0.4f}, acc: {training_accura
cy_history[epoch,0]:0.4f}',end='')

    # validate
    test_total = 0
    test_correct = 0
    with torch.no_grad():
        model.eval()
        for i, data in enumerate(test_data_loader):
            images, labels = data
            # forward pass
            output = model(images)
            # find accuracy
            _, predicted = torch.max(output.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()
            # find loss
            loss = criterion(output, labels)
            validation_loss_history[epoch] += loss.item()
        validation_loss_history[epoch] /= len(test_data_loader)
        validation_accuracy_history[epoch] = test_correct / test_total
    print(f', val loss: {validation_loss_history[epoch,0]:0.4f}, val acc: {validat
ion_accuracy_history[epoch,0]:0.4f}')
```

```
Epoch 1/10:...........
        loss: 0.1395, acc: 0.9559, val loss: 0.0378, val acc: 0.9877
Epoch 2/10:...........
        loss: 0.0665, acc: 0.9795, val loss: 0.0470, val acc: 0.9838
Epoch 3/10:...........
        loss: 0.0560, acc: 0.9823, val loss: 0.0339, val acc: 0.9888
Epoch 4/10:...........
        loss: 0.0514, acc: 0.9846, val loss: 0.0361, val acc: 0.9881
Epoch 5/10:...........
        loss: 0.0472, acc: 0.9857, val loss: 0.0305, val acc: 0.9919
Epoch 6/10:...........
        loss: 0.0432, acc: 0.9869, val loss: 0.0288, val acc: 0.9902
Epoch 7/10:...........
        loss: 0.0416, acc: 0.9874, val loss: 0.0292, val acc: 0.9910
Epoch 8/10:...........
        loss: 0.0370, acc: 0.9886, val loss: 0.0290, val acc: 0.9908
Epoch 9/10:...........
        loss: 0.0370, acc: 0.9883, val loss: 0.0318, val acc: 0.9906
Epoch 10/10:...........
        loss: 0.0334, acc: 0.9901, val loss: 0.0272, val acc: 0.9920
```

Above, we output the training loss/accuracy as well as the validation loss and accuracy. Not bad! Let's see if you can do better.