Kenneth Le

UID:804953883

CS M146

PSET 3 Write Up

Question 4.)

a.)

### ========== TODO : START ========== ###

### part a: print out three training images with different labels

int1 = np.random.randint(1,300)

int2 = np.random.randint(1,300)

int3 = np.random.randint(1,300)

plot\_img(X\_train[int1])

plot\_img(X\_train[int2])

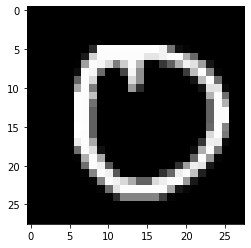
plot\_img(X\_train[int3])

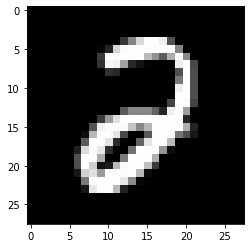
print(int1)

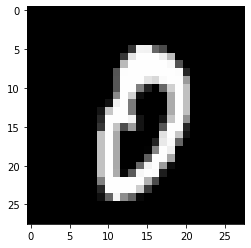
print(int2)

print(int3)

### ========== TODO : END ========== ###







b.)

### ========== TODO : START ========== ###

### part b: convert numpy arrays to tensors

X\_train\_t = torch.from\_numpy(X\_train)

X\_test\_t = torch.from\_numpy(X\_test)

y\_train\_t = torch.from\_numpy(y\_train)

y\_test\_t = torch.from\_numpy(y\_test)

X\_valid\_t = torch.from\_numpy(X\_valid)

y\_valid\_t = torch.from\_numpy(y\_valid)

### ========== TODO : END ========== ###

c.)

### ========== TODO : START ========== ###

### part c: prepare dataloaders for training, validation, and testing

### we expect to get a batch of pairs (x\_n, y\_n) from the dataloader

### train\_loader = ...

### valid\_loader = ...

### test\_loader = ...

train\_loader = DataLoader(TensorDataset(X\_train\_t,y\_train\_t),batch\_size = 10)

test\_loader = DataLoader(TensorDataset(X\_test\_t,y\_test\_t),batch\_size = 10)

valid\_loader = DataLoader(TensorDataset(X\_valid\_t,y\_valid\_t),batch\_size = 10)

### ========== TODO : END ========== ###

d.)

### ========== TODO : START ========== ###

### part d: implement OneLayerNetwork with torch.nn.Linear

self.one\_layer = torch.nn.Linear(784,3)

### ========== TODO : END ========== ###

def forward(self, x):

# x.shape = (n\_batch, n\_features)

### ========== TODO : START ========== ###

### part d: implement the foward function

outputs = self.one\_layer(x)

### ========== TODO : END ========== ###

return outputs

e.)

### ========== TODO : START ========== ###

### part e: prepare OneLayerNetwork, criterion, and optimizer

model\_one = OneLayerNetwork()

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model\_one.parameters(),lr = 0.0005)

### ========== TODO : END ========== ###

f.)

### ========== TODO : START ========== ###

### part f: implement the training process

y\_prediction = model.forward(batch\_X)

model.zero\_grad()

loss = criterion(y\_prediction,batch\_y)

loss.backward()

optimizer.step()

### ========== TODO : END ========== ###

g.)

### ========== TODO : START ========== ###

### part g: implement TwoLayerNetwork with torch.nn.Linear

self.two\_layer\_1 = torch.nn.Linear(784,400)

self.two\_layer\_2 = torch.nn.Linear(400,3)

### ========== TODO : END ========== ###

def forward(self, x):

# x.shape = (n\_batch, n\_features)

### ========== TODO : START ========== ###

### part g: implement the foward function

sigmoid = torch.nn.Sigmoid()

first\_layer = self.two\_layer\_1(x)

first\_layer = sigmoid(first\_layer)

second\_layer = self.two\_layer\_2(first\_layer)

outputs = second\_layer

### ========== TODO : END ========== ###

return outputs

h.)

# ### ========== TODO : START ========== ###

# ### part h: prepare TwoLayerNetwork, criterion, and optimizer

model\_two = TwoLayerNetwork()

criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model\_two.parameters(),lr = 0.0005)

# ### ========== TODO : END ========== ###

i.)

### ========== TODO : START ========== ###

### part i: generate a plot to comare one\_train\_loss, one\_valid\_loss, two\_train\_loss, two\_valid\_loss

epoch\_range = np.arange(1,31)

plt.figure(figsize =[9,9])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.plot(epoch\_range,one\_train\_loss,color = "red",label = "One Layer Training Loss",marker = 'o')

plt.plot(epoch\_range,one\_valid\_loss,color = "green",label = "One Layer Valid Loss",marker = 'o')

plt.plot(epoch\_range,two\_train\_loss,color = "blue",label = "Two Layer Training Loss",marker = 'o')

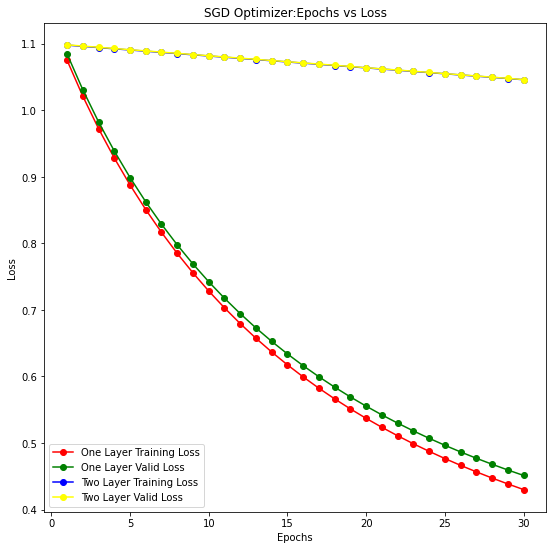
plt.plot(epoch\_range,two\_valid\_loss,color = "yellow",label = "Two Layer Valid Loss",marker = 'o')

plt.title("SGD Optimizer:Epochs vs Loss")

plt.legend()

plt.show()

### ========== TODO : END ========== ###



From this plot we can see that for the 1 layer network the training and validation loss decreases at a much quicker rate than the 2 layer network does. This is mostly due to the fact that the SGD optimizer we are using uses a small number of training samples for each iteration. As a result, the loss for the 2 layer network will take longer to decrease vs. the 1 layer. Another thing to note is that our training and validation curves are extremely close to one another suggesting that the data is not overfitted.

j.)

### part j: generate a plot to comare one\_train\_acc, one\_valid\_acc, two\_train\_acc, two\_valid\_acc

plt.figure(figsize =[9,9])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.plot(epoch\_range,one\_train\_acc,color = "red",label = "One Layer Training Accuracy",marker = 'o')

plt.plot(epoch\_range,one\_valid\_acc,color = "green",label = "One Layer Valid Accuracy",marker = 'o')

plt.plot(epoch\_range,two\_train\_acc,color = "blue",label = "Two Layer Training Accuracy",marker = 'o')

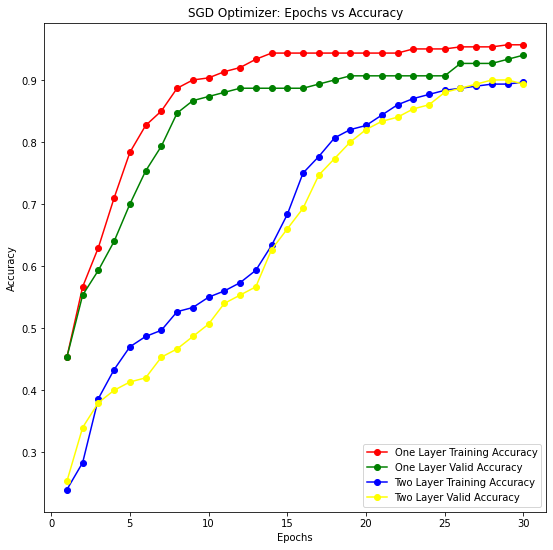
plt.plot(epoch\_range,two\_valid\_acc,color = "yellow",label = "Two Layer Valid Accuracy",marker = 'o')

plt.title("SGD Optimizer: Epochs vs Accuracy")

plt.legend()

plt.show()

### ========== TODO : END ========== ##



From this plot we can see that the accuracy for the 1 layer network also increases at a more rapid rate than the 2 layer network does but begin to reach the same level of accuracy over more epochs. Still, the 1 layer network achieves higher accuracy by a small margin at the end of the range of epochs.

k.)

### ========== TODO : START ========== ###

### part k: calculate the test accuracy

one\_layer\_test\_acc = evaluate\_acc(model\_one,test\_loader)

two\_layer\_test\_acc = evaluate\_acc(model\_two,test\_loader)

print("One layer Test Accuracy = ",one\_layer\_test\_acc)

print("Two Layer Test Accuracy = ",two\_layer\_test\_acc)

### ========== TODO : END ========== ###

One layer Test Accuracy = tensor(0.9600)

Two Layer Test Accuracy = tensor(0.9000)

It is natural to see that the 1 layer network in this scenario has achieved a higher test accuracy than the 2 layer as the 1 layer network can tune itself quicker with less parameters than the 2 layer network over less number of epochs. In order for the two layer network to reach the accuracy of the one layer it will need more epochs.

l.)

### ========== TODO : START ========== ###

### part l: replace the SGD optimizer with the Adam optimizer and do the experiments again

model\_one = OneLayerNetwork()

model\_two = TwoLayerNetwork()

criterion = torch.nn.CrossEntropyLoss()

optimizer1 = torch.optim.Adam(model\_one.parameters(),lr = 0.0005)

results\_one = train(model\_one, criterion, optimizer1, train\_loader, valid\_loader, epochs=31)

optimizer2 = torch.optim.Adam(model\_two.parameters(),lr = 0.0005)

results\_two = train(model\_two, criterion, optimizer2, train\_loader, valid\_loader, epochs=31)

one\_train\_loss, one\_valid\_loss, one\_train\_acc, one\_valid\_acc = results\_one

two\_train\_loss, two\_valid\_loss, two\_train\_acc, two\_valid\_acc = results\_two

plt.figure(figsize =[9,9])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.plot(epoch\_range,one\_train\_loss,color = "red",label = "One Layer Training Loss",marker = 'o')

plt.plot(epoch\_range,one\_valid\_loss,color = "green",label = "One Layer Valid Loss",marker = 'o')

plt.plot(epoch\_range,two\_train\_loss,color = "blue",label = "Two Layer Training Loss",marker = 'o')

plt.plot(epoch\_range,two\_valid\_loss,color = "yellow",label = "Two Layer Valid Loss",marker = 'o')

plt.title("Adam Optimizer: Epochs vs Loss")

plt.legend()

plt.show()

plt.figure(figsize =[9,9])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.plot(epoch\_range,one\_train\_acc,color = "red",label = "One Layer Training Accuracy",marker = 'o')

plt.plot(epoch\_range,one\_valid\_acc,color = "green",label = "One Layer Valid Accuracy",marker = 'o')

plt.plot(epoch\_range,two\_train\_acc,color = "blue",label = "Two Layer Training Accuracy",marker = 'o')

plt.plot(epoch\_range,two\_valid\_acc,color = "yellow",label = "Two Layer Valid Accuracy",marker = 'o')

plt.title("Adam Optimizer: Epochs vs Accuracy")

plt.legend()

plt.show()

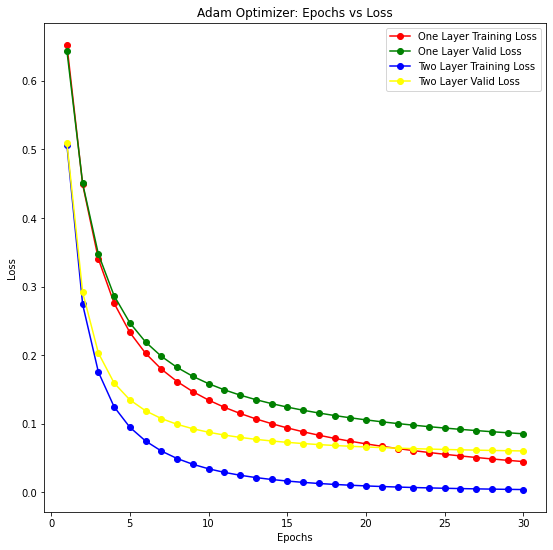
one\_layer\_test\_acc = evaluate\_acc(model\_one,test\_loader)

two\_layer\_test\_acc = evaluate\_acc(model\_two,test\_loader)

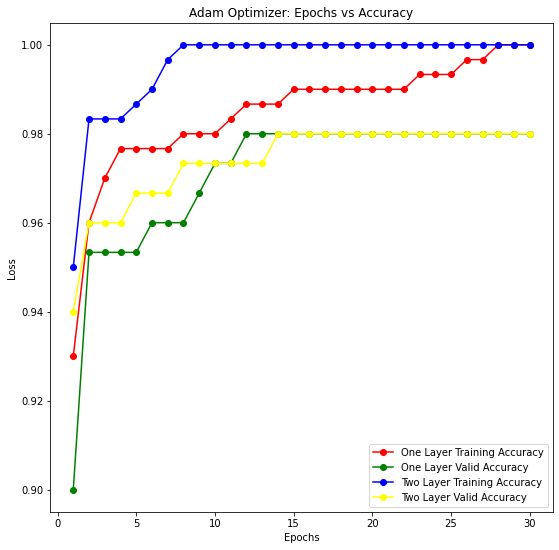
print("Adam Optimizer: One layer Test Accuracy = ",one\_layer\_test\_acc)

print("Adam Optimizer: Two Layer Test Accuracy = ",two\_layer\_test\_acc)

### ========== TODO : END ========== ###



From this plot we can see the Adam optimizer is able to converge towards 0 at a rate much faster than the SGD optimizer can. This is due to the fact that the adam optimizer to tune the learning rate depending on whether it is at the beginning of training or at the end of training. As such it learns at a much faster rate in the beginning and much slower at the end which explains the speed at which it was able to decrease the loss.



From the accuracy plot here we see the same effect as before when observing the loss plot as we see that the accuracy of both 1 layer and 2 layer network converge towards 100 percent for the training and 98 percent for the validation set at a much faster rate and is able to converge to these values in 30 epochs.

Adam Optimizer: One layer Test Accuracy = tensor(0.9733)

Adam Optimizer: Two Layer Test Accuracy = tensor(0.9667)

With the Adam optimizer now we see that both the one layer and two layer test accuracies are much closer to each other than before.