Họ và tên: Võ Hồng Quân

MSSV: 22134012

Lớp: 22134

import numpy as np  
import scipy.io  
import torch  
import torch.nn as nn  
from torch.utils.data import Dataset, DataLoader  
import torchvision  
from sklearn.model\_selection import train\_test\_split  
import torch.optim.lr\_scheduler as lr\_scheduler  
import matplotlib.pyplot as plt  
from torch.utils.tensorboard import SummaryWriter

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
writer = SummaryWriter()

# SplitDataset class to load and split the data  
class SplitDataset:  
 def \_\_init\_\_(self, path='', train\_test\_split\_size=[]):  
 self.data = scipy.io.loadmat(path)  
 self.inputs = self.data['X']  
 self.targets = np.ravel(self.data['y'])  
 self.train\_test\_split\_size = train\_test\_split\_size  
  
 # Modify label 1-10 into 0-9  
 self.targets[self.targets == 10] = 0  
  
 def \_\_call\_\_(self):  
 if self.train\_test\_split\_size:  
 train\_size, test\_size, valid\_size = self.train\_test\_split\_size  
 self.X\_new, self.X\_test, self.y\_new, self.y\_test = train\_test\_split(  
 self.inputs, self.targets, train\_size=train\_size, test\_size=test\_size)  
 self.X\_train, self.X\_valid, self.y\_train, self.y\_valid = train\_test\_split(  
 self.X\_new, self.y\_new, test\_size=valid\_size / (train\_size + valid\_size))  
  
 self.train = [self.X\_train, self.y\_train]  
 self.test = [self.X\_test, self.y\_test]  
 self.valid = [self.X\_valid, self.y\_valid]  
   
 return self.train, self.test, self.valid

# Custom Dataset class  
class GetDataset(Dataset):  
 def \_\_init\_\_(self, transforms=None, dataset=None):  
 super().\_\_init\_\_()  
 self.transform = transforms  
 self.inputs, self.targets = dataset  
  
 def \_\_getitem\_\_(self, index):  
 sample = self.inputs[index], self.targets[index]  
  
 if self.transform:  
 sample = self.transform(sample)  
  
 return sample  
   
 def \_\_len\_\_(self):  
 return len(self.inputs)

# Transformation class to convert data to tensors  
class Transform:  
 def \_\_call\_\_(self, sample):  
 inputs, targets = sample  
 inputs = inputs.astype(np.float32).reshape(1, 20, 20) # Reshape to (1, 20, 20)  
 targets = torch.tensor(targets, dtype=torch.long)  
 return torch.from\_numpy(inputs), targets

# Load and split the dataset  
train\_array, test\_array, valid\_array = SplitDataset( path='ex7data.mat', train\_test\_split\_size=[0.7, 0.2, 0.1]).\_\_call\_\_()  
train\_tensor = GetDataset(transforms=Transform(), dataset=train\_array)  
test\_tensor = GetDataset(transforms=Transform(), dataset=test\_array)  
valid\_tensor = GetDataset(transforms=Transform(), dataset=valid\_array)

# DataLoaders  
batch\_size = 200  
train\_loader = DataLoader(dataset=train\_tensor, batch\_size=batch\_size, shuffle=True)  
test\_loader = DataLoader(dataset=test\_tensor, batch\_size=batch\_size, shuffle=False)  
valid\_loader = DataLoader(dataset=valid\_tensor, batch\_size=batch\_size, shuffle=False)

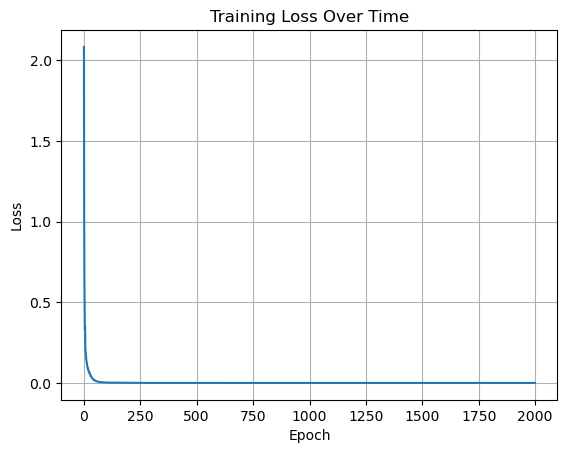
# Updated ConvNet architecture  
class ConvNet(nn.Module):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.conv = nn.Sequential(  
 nn.Conv2d(in\_channels=1, out\_channels=32, kernel\_size=3), # Output: [32, 18, 18]  
 nn.ReLU(),  
 nn.MaxPool2d(kernel\_size=2, stride=2), # Output: [32, 9, 9]  
  
 nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3), # Output: [64, 7, 7]  
 nn.ReLU(),  
 nn.MaxPool2d(kernel\_size=2, stride=2), # Output: [64, 3, 3]  
  
 nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3),# Output: [128, 1, 1]  
 nn.ReLU()  
 )  
  
 self.fc = nn.Sequential(  
 nn.Flatten(),  
 nn.Linear(128, 64),  
 nn.Linear(64, 32),  
 nn.Linear(32, 10) # Output logits for 10 classes  
 )  
  
 def forward(self, x):  
 x = self.conv(x)  
 x = self.fc(x)  
 return x

# Initialize the model  
model = ConvNet().to(device)  
criterion = nn.CrossEntropyLoss()  
# optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  
optimizer = torch.optim.Adagrad(params= model.parameters(), lr= 0.01)  
scheduler = lr\_scheduler.StepLR(optimizer= optimizer, step\_size= 350, gamma= 0.1)

# Training Loop  
num\_epochs = 2000  
loss\_ = []  
for epoch in range(num\_epochs):  
 model.train()  
 running\_loss = 0.0  
 for inputs, targets in train\_loader:  
 inputs, targets = inputs.to(device), targets.to(device)  
  
 # Forward pass  
 outputs = model(inputs) # Outputs of shape [batch\_size, 10]  
  
 # print(targets.shape)  
 # print(outputs.shape)  
 # Compute loss  
 loss = criterion(outputs, targets)  
  
 # Backward pass and optimization  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item()  
 scheduler.step()  
 avg\_loss = running\_loss / len(train\_loader)  
 loss\_.append(avg\_loss)  
 if (epoch + 1) % 100 == 0:  
 print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {avg\_loss:.4f}")

Epoch [100/2000], Loss: 0.0026  
Epoch [200/2000], Loss: 0.0007  
Epoch [300/2000], Loss: 0.0004  
Epoch [400/2000], Loss: 0.0003  
Epoch [500/2000], Loss: 0.0003  
Epoch [600/2000], Loss: 0.0002  
Epoch [700/2000], Loss: 0.0002  
Epoch [800/2000], Loss: 0.0002  
Epoch [900/2000], Loss: 0.0002  
Epoch [1000/2000], Loss: 0.0002  
Epoch [1100/2000], Loss: 0.0002  
Epoch [1200/2000], Loss: 0.0002  
Epoch [1300/2000], Loss: 0.0002  
Epoch [1400/2000], Loss: 0.0002  
Epoch [1500/2000], Loss: 0.0002  
Epoch [1600/2000], Loss: 0.0002  
Epoch [1700/2000], Loss: 0.0002  
Epoch [1800/2000], Loss: 0.0002  
Epoch [1900/2000], Loss: 0.0002  
Epoch [2000/2000], Loss: 0.0002

import matplotlib.pyplot as plt  
  
plt.plot(loss\_)  
plt.title('Training Loss Over Time')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.grid()  
plt.show()



# Testing Loop  
classes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']  
model.eval()  
with torch.no\_grad():  
 n\_correct = 0  
 n\_samples = 0  
 n\_class\_correct = [0 for \_ in range(10)]  
 n\_class\_samples = [0 for \_ in range(10)]  
 for images, labels in test\_loader:  
 images, labels = images.to(device), labels.to(device)  
 outputs = model(images) # Outputs of shape [batch\_size, 10]  
 \_, predicted = torch.max(outputs, 1)  
 n\_samples += labels.size(0)  
 n\_correct += (predicted == labels).sum().item()  
  
 for i in range(len(labels)):  
 label = labels[i]  
 pred = predicted[i]  
 if label == pred:  
 n\_class\_correct[label] += 1  
 n\_class\_samples[label] += 1  
  
 acc = 100.0 \* n\_correct / n\_samples  
 print(f'Overall Accuracy: {acc:.2f}%')  
  
 for i in range(10):  
 if n\_class\_samples[i] > 0:  
 acc = 100.0 \* n\_class\_correct[i] / n\_class\_samples[i]  
 print(f'Accuracy of class {classes[i]}: {acc:.4f}%')  
 else:  
 print(f'Accuracy of class {classes[i]}: N/A (no samples)')

Overall Accuracy: 97.00%  
Accuracy of class 0: 97.9167%  
Accuracy of class 1: 95.8333%  
Accuracy of class 2: 99.0566%  
Accuracy of class 3: 94.5455%  
Accuracy of class 4: 99.0385%  
Accuracy of class 5: 99.0000%  
Accuracy of class 6: 97.1154%  
Accuracy of class 7: 95.7447%  
Accuracy of class 8: 96.9388%  
Accuracy of class 9: 94.5652%