**姓名：丁永奇**

**学号：202121090222**

**日期：2021年10月2日**

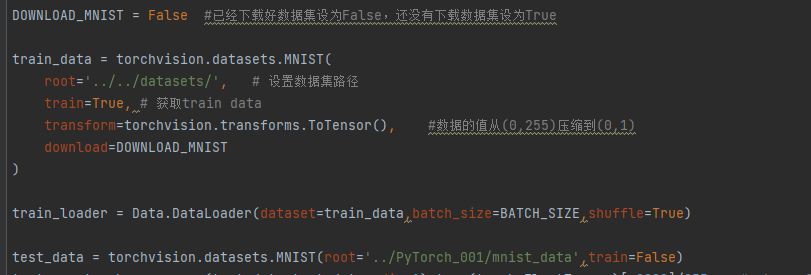
**作业1**

一、概述

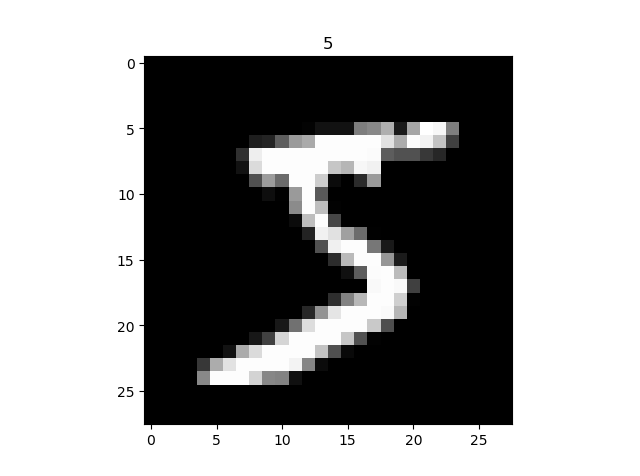
使用PyTorch搭建一个简单的卷积神经网络训练模型来识别MNIST数据集中的图片。

二、数据集

直接使用torchvision中的datasets下载MNIST数据集，这一部分代码如下图：



打印数据集中的一张样本的图像如下：



三、模型设计

模型的结构为卷积层、激活函数、池化层、卷积层、激活函数、池化层和全连接层。

第一层卷积层接收输入为1\*28\*28的MNIST数据集中的图片，kernel\_size设置为5，stride设置为1，padding设置为2，使用16个卷积核，得到16\*28\*28的输出。对得到的结果使用ReLU激活函数和最大池化，之后得到16\*14\*14的结果。

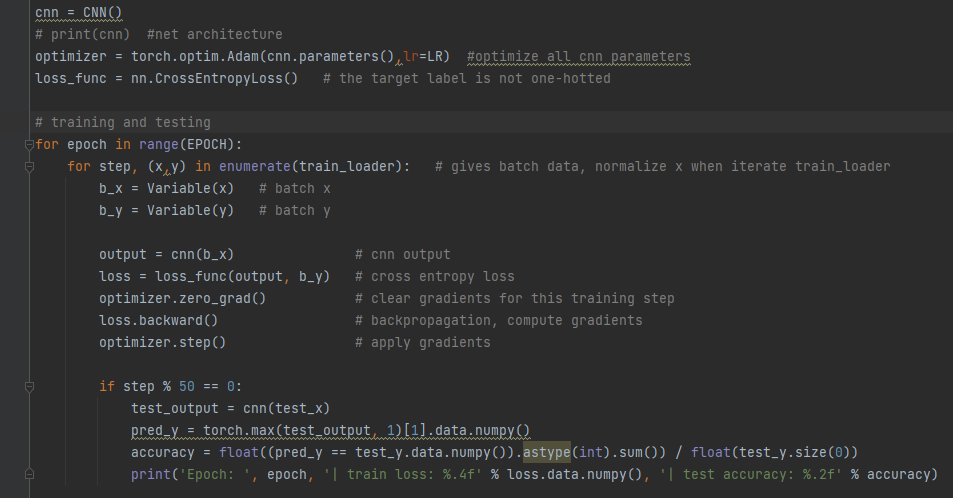
第二层卷积层接收到16\*14\*14的输入，仍然使用大小为5的kernel\_size，stride设置为1，padding设置为2，使用32个卷积核最后得到32\*7\*7的结果。对结果使用ReLU激活函数和最大池化之后得到32\*7\*7的结果。在最后使用全连接层将其分为10类。

四、核心代码

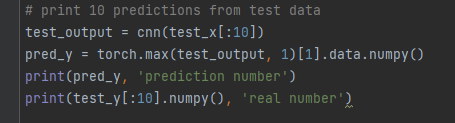
搭建模型部分的代码如下图所示，在代码中多处已经写上了注释。主要是通过nn.Sequential()快速搭建两层卷积层，在最后通过nn.Linear()搭建一个全连接层以将结果分为10类：



训练过程的代码如下图所示，先对之前所搭建的模型创建一个名为cnn的实例对象，然后在多个epoch里面迭代训练（这里简化为1个epoch），损失函数使用交叉熵损失函数来计算，使用Adam进行优化，计算出损失函数之后对PyTroch积累的前一步梯度进行清零再反向传播新的梯度进行优化。每经过50次的优化之后打印出当前的epoch值、loss值以及准确率。

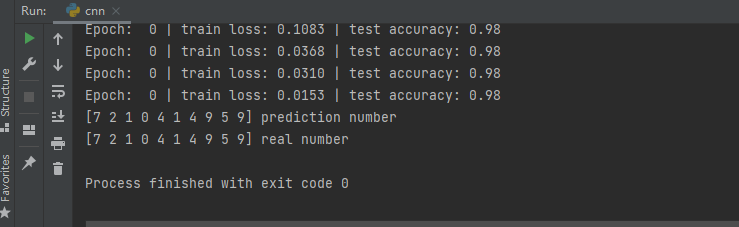


在最后从测试集里拿出十个样本进行测试，看一下模型的效果，这部分代码如下：



五、结果

训练完之后打印最终的训练结果如下图：



可以看到最终的训练误差降到了0.0153，测试的准确度达到了0.98.从测试集中取出10张图像标签分别为[7,2,1,0,4,1,4,9,5,9]，对其进行识别得到的结果并没有出现错误。

**附录**

import torch  
import torch.nn as nn  
from torch.autograd import Variable  
import torch.utils.data as Data  
import torchvision  
import matplotlib.pyplot as plt  
  
*# Hyper Parameters*EPOCH = 1 *# train the training data n times,to save time,we just train 1 epoch*BATCH\_SIZE = 50  
LR = 0.001  
DOWNLOAD\_MNIST = False *#已经下载好数据集设为False，还没有下载数据集设为True*train\_data = torchvision.datasets.MNIST(  
 root=**'../../datasets/'**, *# 设置数据集路径* train=True, *# 获取train data* transform=torchvision.transforms.ToTensor(), *#数据的值从(0,255)压缩到(0,1)* download=DOWNLOAD\_MNIST  
)  
*# plot one example  
# print(train\_data.train\_data.size()) #(60000,28,28)  
# print(train\_data.train\_labels.size()) #(60000)  
# plt.imshow(train\_data.train\_data[0].numpy(),cmap='gray')  
# plt.title('%i' % train\_data.train\_labels[0])  
# plt.show()*

train\_loader = Data.DataLoader(dataset=train\_data,batch\_size=BATCH\_SIZE,shuffle=True)  
  
test\_data = torchvision.datasets.MNIST(root=**'../PyTorch\_001/mnist\_data'**,train=False)  
test\_x = torch.unsqueeze(test\_data.test\_data, dim=1).type(torch.FloatTensor)[:2000]/255. *# shape from (2000, 28, 28) to (2000, 1, 28, 28), value in range(0,1)*test\_y = test\_data.test\_labels[:2000]  
  
class CNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(CNN,self).\_\_init\_\_()  
 self.conv1 = nn.Sequential( *#卷积层* nn.Conv2d( *#(1,28,28)* in\_channels=1,  
 out\_channels=16,  
 kernel\_size=5,*#一个filter的长和宽都是5个像素点* stride=1,*#每扫描一次的跳度范围 pixel* padding=2,  
 *# 在图像的旁边多加了一层为0的数据 if stride=1,如果想要 con2d 出来的图片长宽没有变化,padding=(kernel\_size-1)/2=(5-1)/2* ),*# 卷积层、一个过滤器filter -> (16,28,28)* nn.ReLU(), *# (16,28,28)* nn.MaxPool2d(kernel\_size=2), *# 池化层 -> (16,14,14)* )  
 self.conv2 = nn.Sequential( *# 输入(16,14,14)* nn.Conv2d(16,32,5,1,2),  
 nn.ReLU(),  
 nn.MaxPool2d(2)  
 )  
 self.out = nn.Linear(32\*7\*7,10)  
  
 def forward(self,x):  
 x = self.conv1(x)  
 x = self.conv2(x) *#(batch,32,7,7)* x = x.view(x.size(0),-1) *#(batch,32\*7\*7) -1的作用就是把后面的32 7 7变到一起* output = self.out(x)  
 return output  
  
cnn = CNN()  
*# print(cnn) #net architecture*optimizer = torch.optim.Adam(cnn.parameters(),lr=LR) *#optimize all cnn parameters*loss\_func = nn.CrossEntropyLoss() *# the target label is not one-hotted  
  
# training and testing*for epoch in range(EPOCH):  
 for step, (x,y) in enumerate(train\_loader): *# gives batch data, normalize x when iterate train\_loader* b\_x = Variable(x) *# batch x* b\_y = Variable(y) *# batch y* output = cnn(b\_x) *# cnn output* loss = loss\_func(output, b\_y) *# cross entropy loss* optimizer.zero\_grad() *# clear gradients for this training step* loss.backward() *# backpropagation, compute gradients* optimizer.step() *# apply gradients* if step % 50 == 0:  
 test\_output = cnn(test\_x)  
 pred\_y = torch.max(test\_output, 1)[1].data.numpy()  
 accuracy = float((pred\_y == test\_y.data.numpy()).astype(int).sum()) / float(test\_y.size(0))  
 print(**'Epoch: '**, epoch, **'| train loss: %.4f'** % loss.data.numpy(), **'| test accuracy: %.2f'** % accuracy)  
  
*# print 10 predictions from test data*test\_output = cnn(test\_x[:10])  
pred\_y = torch.max(test\_output, 1)[1].data.numpy()  
print(pred\_y, **'prediction number'**)  
print(test\_y[:10].numpy(), **'real number'**)