



import torch  
import torch.nn as nn  
import torchvision.datasets as dsets  
import torchvision.transforms as transforms  
from torch.autograd import Variable  
  
# Hyper Parameters 配置参数  
torch.manual\_seed(1) # 设置随机数种子，确保结果可重复  
input\_size = 784  
hidden\_size = 500  
num\_classes = 10  
num\_epochs = 5 # 训练次数  
batch\_size = 100 # 批处理大小  
learning\_rate = 0.001 # 学习率  
  
# MNIST Dataset 下载训练集 MNIST 手写数字训练集  
train\_dataset = dsets.MNIST(root="D:\桌面\python程序\机器学习\MNIST数据集", # 数据保持的位置  
 train=True, # 训练集  
 transform=transforms.ToTensor(),  
# 一个取值范围是[0,255]的PIL.Image  
# 转化为取值范围是[0,1.0]的torch.FloadTensor  
 download=True) # 下载数据  
test\_dataset = dsets.MNIST(root="D:\桌面\python程序\机器学习\MNIST数据集",  
 train=False, # 测试集  
 transform=transforms.ToTensor())  
  
# Data Loader (Input Pipeline)  
# 数据的批处理，尺寸大小为batch\_size,  
# 在训练集中，shuffle 必须设置为True, 表示次序是随机的  
train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset,  
 batch\_size=batch\_size,  
 shuffle=True)  
  
test\_loader = torch.utils.data.DataLoader(dataset=test\_dataset,  
 batch\_size=batch\_size,  
 shuffle=False)  
  
  
# Neural Network Model (1 hidden layer) 定义神经网络模型  
class Net(nn.Module):  
 def \_\_init\_\_(self, input\_size, hidden\_size, num\_classes):  
 super(Net, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(input\_size, hidden\_size)  
 self.relu = nn.ReLU()  
 self.fc2 = nn.Linear(hidden\_size, num\_classes)  
  
 def forward(self, x):  
 out = self.fc1(x)  
 out = self.relu(out)  
 out = self.fc2(out)  
 return out  
  
  
net = Net(input\_size, hidden\_size, num\_classes)  
  
print(net)  
  
# Loss and Optimizer 定义loss和optimizer  
criterion = nn.CrossEntropyLoss()  
  
optimizer = torch.optim.Adam(net.parameters(), lr=learning\_rate)  
# Train the Model 开始训练  
  
for epoch in range(num\_epochs):  
  
 for i, (images, labels) in enumerate(train\_loader): # 批处理  
  
 # Convert torch tensor to Variable  
  
 images = Variable(images.view(-1, 28 \* 28))  
  
 labels = Variable(labels)  
  
 # Forward + Backward + Optimize  
  
 optimizer.zero\_grad() # zero the gradient buffer #梯度清零，以免影响其他batch  
  
 outputs = net(images) # 前向传播  
  
 # import pdb  
  
 # pdb.set\_trace()  
  
 loss = criterion(outputs, labels) # loss  
  
 loss.backward() # 后向传播，计算梯度  
  
 optimizer.step() # 梯度更新  
  
 if (i + 1) % 100 == 0:  
 print('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'  
  
 % (epoch + 1, num\_epochs, i + 1, len(train\_dataset) // batch\_size, loss.item()))  
  
# Test the Model  
correct = 0  
total = 0  
for images, labels in test\_loader: # test set 批处理  
 images = Variable(images.view(-1, 28 \* 28))  
 outputs = net(images)  
 \_, predicted = torch.max(outputs.data, 1) # 预测结果  
 total += labels.size(0) # 正确结果  
 correct += (predicted == labels).sum() # 正确结果总数  
  
print('Accuracy of the network on the 10000 test images: %d %%' % (100 \* correct / total))