hw3实验报告

52205903005 娄泽华

**hw3要求：**

数据集：MNIST手写识别数据集 <http://yann.lecun.com/exdb/mnist/>

任务：识别字符

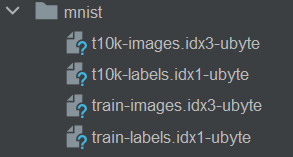
源码+实验报告

交给助教

Deadline: 学期末考试前

# 一、数据集简介

MNIST手写数字数据集有60000个示例的训练集和10000个示例的测试集。它是NIST提供的更大集合的子集。图像中的数字已经过大小标准化，并在固定大小的图像中居中：



数据集说明：

train-images-idx3-ubyte: training set images

train-labels-idx1-ubyte: training set labels

t10k-images-idx3-ubyte: test set images

t10k-labels-idx1-ubyte: test set labels

数据集格式：

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset] [type] [value] [description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 60000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

........

xxxx unsigned byte ?? label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset] [type] [value] [description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 60000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

........

xxxx unsigned byte ?? pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset] [type] [value] [description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 10000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

........

xxxx unsigned byte ?? label

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

[offset] [type] [value] [description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 10000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

........

xxxx unsigned byte ?? pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

# 二、实验思路

使用CNN实现手写数字的预测。先读入数据并解析、转换为标准的DataLoader对象，设计MNIST\_CNN网络，输出log softmax值作为损失。

# 三、代码实现

## 3.1 预处理

读入数据并解析

def decode\_idx3\_ubyte(idx3\_ubyte\_file):  
 with open(idx3\_ubyte\_file, 'rb') as f:  
 fb\_data = f.read()  
  
 offset = 0  
 fmt\_header = '>iiii'  
 magic\_number, num\_images, num\_rows, num\_cols = struct.unpack\_from(fmt\_header, fb\_data, offset)  
 offset += struct.calcsize(fmt\_header)  
 fmt\_image = '>' + str(num\_rows \* num\_cols) + 'B'  
  
 images = np.empty((num\_images, num\_rows, num\_cols))  
 for i in range(num\_images):  
 im = struct.unpack\_from(fmt\_image, fb\_data, offset)  
 images[i] = np.array(im).reshape((num\_rows, num\_cols))  
 offset += struct.calcsize(fmt\_image)  
 return images

def decode\_idx1\_ubyte(idx1\_ubyte\_file):  
 with open(idx1\_ubyte\_file, 'rb') as f:  
 fb\_data = f.read()  
 offset = 0  
 fmt\_header = '>ii'  
 magic\_number, label\_num = struct.unpack\_from(fmt\_header, fb\_data, offset)  
 offset += struct.calcsize(fmt\_header)  
 labels = []  
 fmt\_label = '>B'  
 for i in range(label\_num):  
 labels.append(struct.unpack\_from(fmt\_label, fb\_data, offset)[0])  
 offset += struct.calcsize(fmt\_label)  
 return np.array(labels)

# read data  
X\_train = decode\_idx3\_ubyte(os.path.join(mnist\_dir, 'train-images.idx3-ubyte'))  
Y\_train = decode\_idx1\_ubyte(os.path.join(mnist\_dir, 'train-labels.idx1-ubyte'))  
X\_test = decode\_idx3\_ubyte(os.path.join(mnist\_dir, 't10k-images.idx3-ubyte'))  
Y\_test = decode\_idx1\_ubyte(os.path.join(mnist\_dir, 't10k-labels.idx1-ubyte'))

转换为dataloader对象，并打乱顺序：

# format data  
train\_dataset = MnistDataset(X\_train, Y\_train)  
train\_dataloader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True)  
test\_dataset = MnistDataset(X\_test, Y\_test)  
test\_dataloader = DataLoader(test\_dataset, batch\_size=BATCH\_SIZE, shuffle=True)

选用adam优化器

optimizer = optim.Adam(model.parameters())

## 3.2 设计CNN

class MNIST\_CNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(MNIST\_CNN, self).\_\_init\_\_()  
 # m \* init\_channels \* 28 \* 28:  
 # m = BATCH\_SIZE or (N mod BATCH\_SIZE): number of samples in each batch ,  
 # init\_channels = 1 for gray image,  
 # img\_shape = 28 \* 28;  
 # loops: N = 60000, n = ceil(N / m) times loop for each epoch.  
 # in\_channels = 1，out\_channels = 8，kernel\_size = 5, stride = 1,  
 # image\_shape = 24 \* 24 # valid padding: floor((image\_shape[i] - kernel\_size) / stride) + 1 for i in [0, 1]  
 self.conv1 = nn.Conv2d(1, 8, 5, 1)  
 # image\_shape = 12 \* 12  
 self.pool1 = nn.MaxPool2d(2, 2)  
 # in\_channels = 8，out\_channels = 16，kernel\_size = 3, stride = 1, image\_shape = 10 \* 10  
 self.conv2 = nn.Conv2d(8, 16, 3, 1)  
 # in\_channels = out\_channels \* row \* column  
 self.fc1 = nn.Linear(16 \* 10 \* 10, 120)  
 self.fc2 = nn.Linear(120, 10)  
  
 def forward(self, x):  
 x = torch.relu(self.conv1(x)) # m \* 1 \* 28 \* 28 -> m \* 8 \* 24 \* 24  
 x = self.pool1(x) # m \* 8 \* 24 \* 24 -> m \* 8 \* 12 \* 12  
  
 x = torch.relu(self.conv2(x)) # m \* 8 \* 12 \* 12 -> m \* 16 \* 10 \* 10  
  
 x = x.view(x.size(0), -1) # m \* 16 \* 10 \* 10 -> m \* 1600  
 x = torch.relu(self.fc1(x)) # m \* 1600 -> m \* 120  
 x = self.fc2(x) # m \* 120 -> m \* 10  
  
 out = torch.log\_softmax(x, dim=1) # log(softmax(x))  
 return out

设计层次如下：

1)输入：m \* 1 \* 28 \* 28，表示m个样本，每个样本的数据为1个灰度通道，高度和宽度形状为28\*28。

2)卷积层1：conv1设置为1,8,5,1表示输入1通道，输出8通道，卷积核为5\*5，步长为1。本层输出为m \* 8 \* 24 \* 24。宽和高的计算公式：。

3)Relu层1：不影响张量形状。

4)pool层1：聚合参数为2,2，max聚合，宽和高减半，即输出形状为m \* 8 \* 12 \* 12。

5)卷积层2：conv2设置为8,16,3,1表示输入8通道，输出16通道，卷积核为3\*3，步长为1。本层输出为m \* 16 \* 10 \* 10。宽和高的计算公式：。

6)Relu层2：不影响张量形状。

7)全连接层1：参数为16\*10\*10,120，即输出形状为m个120维的列向量。

8) Relu层3：不影响张量形状。

9)全连接层2：参数为120,10，即输出形状为m个10维的列向量。

10)输出层：对m个列向量的每一个向量内的分量之间计算log-softmax作为log似然损失输出，不影响张量形状。

## 3.3训练、预测

训练，输出训练的epoch信息，包括训练的进度信息；并输出每个batch后的总体精度，以及该batch完成后的样本平均log似然损失：

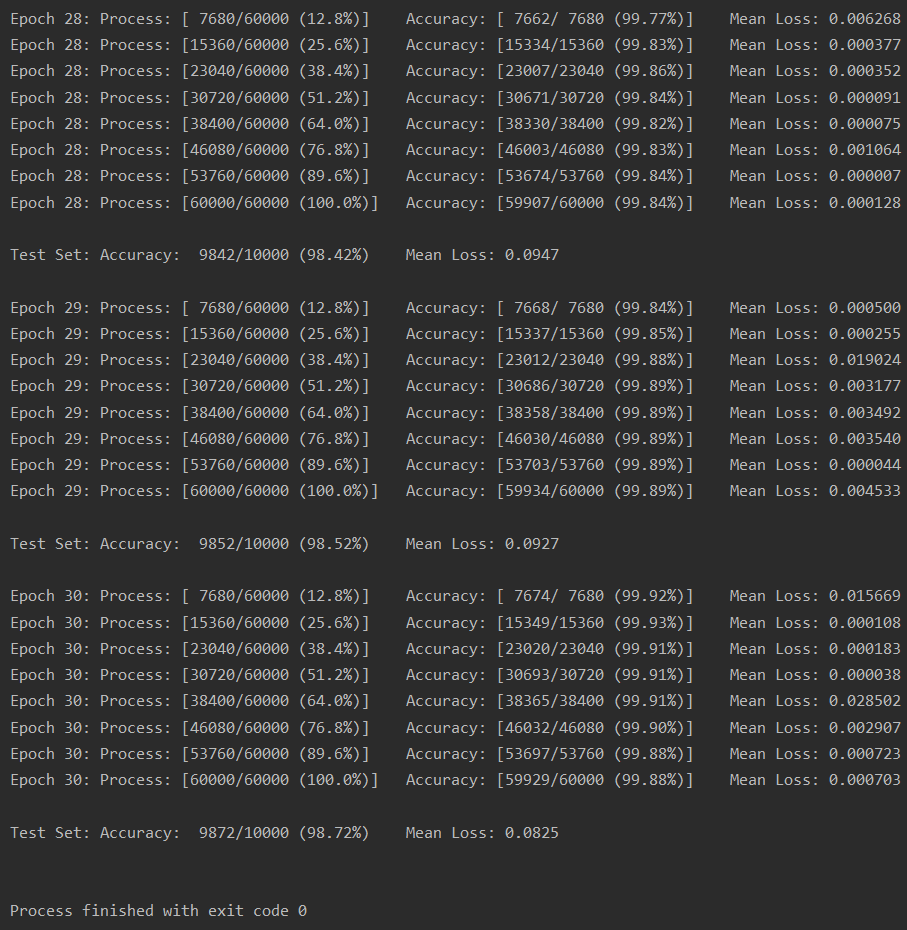
def m\_train(model, train\_loader, optimizer, epoch, device=DEVICE):  
 model.train()  
 train\_correct = 0  
 train\_total = 0  
 for batch\_idx, (data, target) in enumerate(train\_loader):  
 data, target = data.to(device), target.to(device)  
 optimizer.zero\_grad()  
 output = model(data)  
 loss = F.nll\_loss(output, target)  
 loss.backward()  
 optimizer.step()  
 pred = output.max(1, keepdim=True)[1] # 概率最大的索引  
 train\_correct += pred.eq(target.view\_as(pred)).sum().item()  
 train\_total += target.size(0)  
 if (batch\_idx + 1) % 30 == 0 or batch\_idx + 1 >= len(train\_loader):  
 print(  
 'Epoch {:2d}: Process: [{:5d}/{:5d} ({:.1f}%)]\tAccuracy: [{:5d}/{:5d} ({:.2f}%)]\tMean Loss: {:.6f}'.format(  
 epoch, batch\_idx \* BATCH\_SIZE + len(data), len(train\_loader.dataset),  
 100.0 \* (batch\_idx \* BATCH\_SIZE + len(data)) / len(train\_loader.dataset), train\_correct,  
 train\_total, 100.0 \* train\_correct / train\_total, loss.item()))

测试，每个epoch训练完成后，在测试集上测试模型，输出每个batch后的总体精度，以及该batch完成后的样本平均log似然损失：

def m\_test(model, test\_loader, device=DEVICE):  
 model.eval()  
 test\_loss = 0  
 correct = 0  
 with torch.no\_grad():  
 for data, target in test\_loader:  
 data, target = data.to(device), target.to(device)  
 output = model(data)  
 test\_loss += F.nll\_loss(output, target, reduction='sum').item()  
 pred = output.max(1, keepdim=True)[1] # 概率最大的索引  
 correct += pred.eq(target.view\_as(pred)).sum().item()  
  
 test\_loss /= len(test\_loader.dataset)  
 print('\nTest Set: Accuracy: {:5d}/{:5d} ({:.2f}%)\tMean Loss: {:.4f}\n'.format(  
 correct, len(test\_loader.dataset), 100.0 \* correct / len(test\_loader.dataset), test\_loss))

# 四、实验结果

30个epoch后，准确率范围稳定在98.400%-98.800%。



# 五、实验心得

本次实验中，我在编程过程中了解了使用pytorch构建CNN网络的方法，并熟悉了CNN的流程及内部数据张量的转换过程。此次实验的关键点主要在于对CNN原理的理解上，此外，参数的设置以及层次结构的选取需要视实际数据不同而合理设置，如用于降维的pool层只用了1次，卷积核随着数据的宽和高度降低也相应地变小了。