第三次上机作业

注意事项

1. 上机作业所需上传的文件，应打包并命名为"学号-姓名-上机#序号"，如"18000xxxxx-张三-上机3"，并在截止时间前上传到教学网。截止时间后仍开放提交，但会酌情扣分。
2. 上机作业为半自动评分。上传的压缩包，会根据压缩包名称自动分类，并提取压缩包名称中的有效信息。请尽可能保持压缩包命名符合规范。压缩包名中的短横线'-'可替换为下划线'_'、加号'+'，但不能是空格' '、长横线'—'。
3. 上机作业中的代码会自动测试。要求代码中读入的外部数据文件存放在代码文件上一级目录下的data文件夹下，生成的图表、数据文件等应在代码文件所在目录下。

```
root
  ├─workdir (Compress this directory and upload it)
  |     ├─mnist_cnn.py (Required)
  |     ├─mnist_test_prediction.csv (Required)
  |     ├─checkpoints (Required)
  |     |     └─model.pt (Required)
  |     ├─handwritten (Optional)
  |     |     ├─num_0.png
  |     |     ├─num_1.png
  |     |     ├─ ...
  |     |     └─num_9.png
  |     └─report.pdf (Required)
  └─data (DO NOT upload this directory)
        └─MNIST
            ├─processed
            |     ├─test.pt
            |     └─training.pt
            └─raw
                  ├─t10k-images-idx3-ubyte
                  ├─t10k-labels-idx1-ubyte
                  ├─train-images-idx3-ubyte
                  └─train-labels-idx1-ubyte
```

4. 题目中要求生成的图表和计算的数据均列在报告中。
5. 题目中要求计算的数据，代码运算得出结果后，应输出到控制台。

手写数字识别是一个经典的多分类任务。卷积神经网络（Convolutional Neural Network, CNN）在解决此类问题上一直有优秀的表现。基于给定的CNN模型，了解手写数字体特征提取、分析、识别的整个流程，完成下列任务：

（1）描述代码中CNN模型的网络结构（可绘制示意图），并简要解释网络中不同层发挥的功能（Linear, Conv2d, MaxPool2d, ReLU, Dropout, LogSoftmax）。

（2）给出输入、输出网络每一层数据的维度。

提示：如一张手写数字图片的尺寸为28x28（单位为像素），是灰度图，如果是以channel first方式存储，其数据维度应为[1, 28, 28]，如果是以channel last方式存储，其数据维度应为[28, 28, 1]

Layer	Input	Output
layer1		
conv1		
pool1		
layer2		
conv2		
pool2		
out_layer		
fc1		
fc2		

（3）简要解释下列名词的含义及其在网络训练中发挥的功能：

- Batch size
- Adam Optimizer
- Learning Rate
- Weight Decay
- NLL Loss与Cross Entropy Loss的联系与区别
- Epoch

（4）用混淆矩阵、精度和召回率评价CNN模型处理多分类问题的效果，并指出这一模型容易将哪些数字错误指认为其他数字？（对于多分类问题，可以将每一类视为正，同时其余所有类视为负，转化为二分类问题计算precision、recall等指标）

（5）从训练集中划分出验证集，在验证集上优化模型的超参数，给出一组最优的超参数，并给出使用这组超参数的CNN模型在测试集上的表现。

（6）手绘一组0 ~ 9的数字，用使用最优超参数的CNN模型进行分类，分类效果如何？可以尝试手绘数字样本的背景和数字颜色、线条粗细、不同的书写方式、加入噪点等会对分类效果产生怎样的影响？你认为可以如何改进模型的分类能力？

（7）将文件打包上传，压缩包中应有文件

- Python源代码`cnn_mnist.py`

  （源代码中应包含（4）-（6）中的功能，参考模板内容）
- 手绘数字样本，可命名为`num_0.png/.../num_9.png`，置于`handwritten`文件夹下
- 对MNIST测试集（10000个样本）的预测结果`mnist_test_prediction.csv`
- 训练好的模型checkpoint，`checkpoints/model.pt`
- 分析报告`*.doc/*.docx/*.pdf`

（如果使用Markdown写报告，请将报告转为pdf格式）

cnn_mnist.py模板

```
import torch
import torchvision
import pandas as pd
import numpy as np
import os
from torch import nn
import torch.nn.functional as F
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader

# custom Dataset wrapper for new handwritten digits organization
class mySet(Dataset):
    def __init__(self, images):
        super(mySet, self).__init__()
        self.data = images
    def __getitem__(self, x):
        return self.data[x]
    def __len__(self):
        return len(self.data)

class CNN(nn.Module):
    def __init__(self): # Parameters are not required. If you introduce any
parameters for convenience, their default values should be specified.
        super(CNN, self).__init__()
        # put your layers here

    def forward(self, x):
        # x: [batch_size, 1, 28, 28]
        # out = YOUR_LAYERS(x)
        return out

    def fit(self, trainloader, valloader, lr=0.001, weight_decay=1e-5,
max_epoch=1, checkpoints_path="./checkpoints"):
        ############################
        # Your Training Procedure #
        ############################

    @torch.no_grad()
    def evaluation(self, evalloader):
        ##############################
        # Your Evaluation Procedure #
        ##############################
        return accuracy, confusion_matrix

    def save_checkpoint(self, path, epoch, loss):
        try:
            optim_state = optimizer.state_dict()
        except:
            optim_state = None
        checkpoint = {
            "model_state_dict": self.state_dict(),
            "epoch": epoch,
            "loss": loss,
            "optimizer_state_dict": optim_state
        }
```

```
        torch.save(checkpoint, path)

    def load_checkpoint(self, path, optimizer = Adam):
        checkpoint = torch.load(path)
        self.load_state_dict(checkpoint['model_state_dict'])
        if checkpoint['optimizer_state_dict'] is not None:
            self.optimizer = optimizer(self.parameters())
            self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        epoch = checkpoint['epoch']
        loss = checkpoint['loss']
        return epoch, loss

    def eval_handwritten_digits(self, directory):
        images_path = os.listdir(directory)
        images = []
        # read images
        for i in images_path:
            # ... #
            images.append((image, label))
        # Calculate mean and std
        # ... #
        # Data normalization
        mytransform = torchvision.transforms.Normalize((_mean), (_sd))
        for i in range(len(images)):
            img = mytransform(images[i][0])
            images[i] = (img, images[i][1])

        # build custom Dataset
        myevalset = mySet(images)
        # build DataLoader
        myloader = DataLoader(mySet(images), shuffle=False, drop_last = False,
batch_size = 1)
        # Evaluation
        myacc, mycm = self.evaluation(myloader)
        return myacc
```