实验二　机器学习分类问题与交叉熵 Cross Entropy 实现

学号：21030031009 姓名：惠欣宇 电话：18149045867

本实验要求独立完成。实验报告正文要求采用小四号字，中文使用宋体，英文使用 Times New Roman。对于实验中出现的代码文字，可使用 Consolas 字体。实验报告要求提交 pdf 电子版。电子版文件命名格式要求为：学号-姓名.pdf。

1. 实验目的

1) 掌握 Cross Entropy 相关概念和计算。
2) 实现在 pytorch 环境下以 Cross Entropy 为损失函数的机器学习分类问题。
3) 熟悉 pyotrch 环境下机器学习框架。

2. 实验预备知识及实验要求

2.1. 实验预备知识

1) Cross Entropy 相关知识
2) pyotrch 环境下机器学习框架
3) 参考资料:
   pyotrch 中文文档
   https://pytorch-cn.readthedocs.io/zh/latest/

2.2. 实验要求：

1) 算法思想（可用伪代码表述），须与上交的代码保持一致。

```
import numpy as np

def cross_entropy(y, p):

Y = np.float_(y)

P = np.float_(p)

return -np.sum(y * np.log(p) + (1 - y) *
np.log(1 - p))
```

2) 展示整个实验过程，可适当截图，并标注说明。
首先导入相关的包：

```
import os
import torch
import torch.nn as nn
```

以上三个包用于模型的定义，我们使用 pytorch 的框架，用于定义卷积神经网络

```
import torch.utils.data as Data
import torchvision
import torchvision.transforms as transforms
```

以上三个包主要用于数据集的准备，用于从 Mnist 数据集中提取训练集与测试集。

```
import matplotlib.pyplot as plt
```

上面这个包主要用于绘制图片，更加清晰地看到结果。

相关数据集的准备：

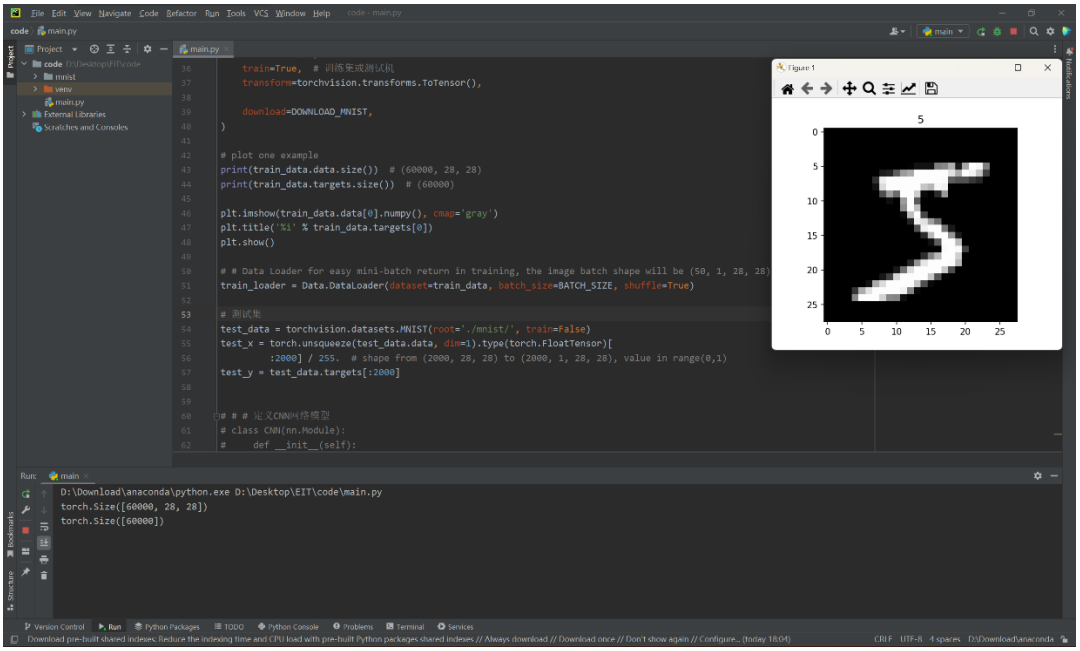
```
# Mnist 数据集
if not (os.path.exists('mnist/')) or not os.listdir('mnist/'):
    DOWNLOAD_MNIST = True

train_data = torchvision.datasets.MNIST(
    root='mnist/',
    train=True,  # 训练集或测试机
    transform=torchvision.transforms.ToTensor(),

    download=DOWNLOAD_MNIST,
)
```

train 为 true 加载训练集，为 false 加载测试集。transform 将加载出来的的数据转换为 ToTensor 格式，将数据维度改变为 chw 的格式(c 为通道数，h 与 w 表示高和宽)。download 表示数据集是否需要下载。

下面通过 print 函数打印一下训练集的数据维度和标签的维度，再通过 matplotlib 模块打印一张图片。

```
# plot one example
print(train_data.data.size())  # (60000, 28, 28)
print(train_data.targets.size())  # (60000)

plt.imshow(train_data.data[0].numpy(), cmap='gray')
plt.title('%i' % train_data.targets[0])
plt.show()
```

注释掉训练过程后运行代码得到 Mnist 训练结果有 60000 张图片，每个图片的大小为 28x28，同时也存在 60000 个标签。右侧图片为使用 matplotlib 绘制的一张图形。说明数据集读取正常。



将训练集转化为迭代器的形式。

```
# # Data Loader for easy mini-batch return in training, the image batch shape will be (50, 1, 28, 28)
train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE, shuffle=True)
```

使用相同的方法加载 Mnist 数据集的测试集，并取出前 2000 张用于测试。

```
# 测试集
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)
test_x = torch.unsqueeze(test_data.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.targets[:2000]
```

卷积神经网络的模型定义：

卷积神经网络主要是通过提升通道数到 16 来降低图片的大小，以此在不同的通道上压缩提取特征。在使用完卷积神经网络之后使用 ReLU 激活函数增加非线性性，来解决线性模型不能解决的问题。在 ReLU 函数之后使用最大池化层函数继续压缩图片的大小。

```
# # 定义CNN网络模型
2 usages
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(  # input shape (1, 28, 28)    channel  h   w  1,28,28
            nn.Conv2d(
                in_channels=1,  # input height
                out_channels=16,  # n_filters
                kernel_size=5,  # filter size
                stride=1,  # filter movement/step
                padding=2,
                # if want same width and length of this image after Conv2d, padding=(kernel_size-1)/2 if stride=1
            ),  # output shape (16, 28, 28)
            nn.ReLU(),  # activation
            nn.MaxPool2d(kernel_size=2),  # choose max value in 2x2 area, output shape (16, 14, 14)
        )
```

然后定义第二个卷积神经网络，将通道数增加到 32，与上述过程相同，添加激活函数与最大池化层函数。

```
self.conv2 = nn.Sequential(  # input shape (16, 14, 14)
    nn.Conv2d(16, 32, 5, 1, 2),  # output shape (32, 14, 14)
    nn.ReLU(),  # activation
    nn.MaxPool2d(2),  # output shape (32, 7, 7)   32 7 7
)
self.out = nn.Linear(32 * 7 * 7, 10)  # fully connected layer, output 10 classes
```

最后使用全连接层进行连接然后输出。

```
def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = x.view(x.size(0), -1)  # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
    output = self.out(x)
    return output, x  # return x for visualization
```

下面打印一下整个网络结构。

```
83      def forward(self, x):
84          x = self.conv1(x)
85          x = self.conv2(x)

un:  main
D:\Download\anaconda\python.exe D:\Desktop\EIT\code\main.py
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)

Process finished with exit code 0
```

使用 Adam 优化器。

```
optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)  # optimize all cnn parameters
```

交叉熵损失函数。

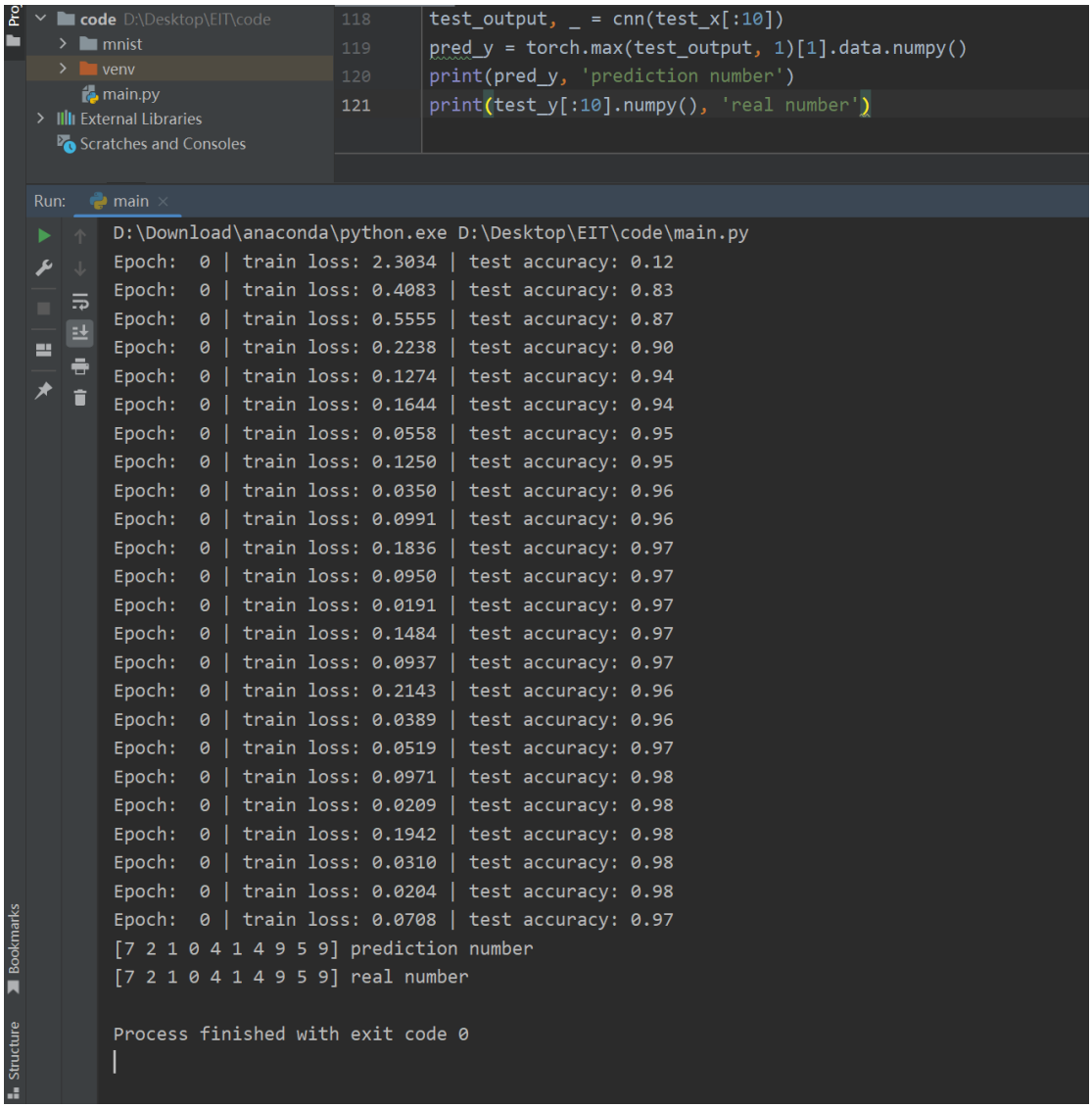
```
loss_func = nn.CrossEntropyLoss()  # the target label is not one-hotted
```

模型训练与测试。

```
# # training and testing
for epoch in range(EPOCH):
    for step, (b_x, b_y) in enumerate(train_loader):  # gives batch data, normalize x when iterate train_loader

        output = cnn(b_x)[0]  # 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, last_layer = 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)

plt.ioff()

# 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')
```

3) 实验结果。

运行代码进行训练，我们可以得到每一轮训练得到的精度，最后精度达到0.98，说明训练效果良好。我们最后发现预测标签与真实标签的十个数据相同，说明模型没有问题。



4) 总结分析。总结实验过程所遇到的困难和问题(给出你的解决办法)。总结关键步需要的注意事项。

我们往往不能只看训练数据上的误分率和交叉熵，还是要关注测试数据上的表现。如果在测试集上的表现也不错，才能保证这不是一个过拟合或者欠拟合的模型。交叉熵比照误分率还有更多的优势，因为它可以和很多概率模型完美的结合。

所以逻辑思路是，为了让学到的模型分布更贴近真实数据分布，我们最小化模型数据分布与训练数据之间的 KL 散度，而因为训练数据的分布是固定的，因此最小化 KL 散度等价于最小化交叉熵。