《人工智能导论》大作业

任务名称： 带 OOD 检测的 Mnist 分类器

完成组号：

小组人员： 曹越淇、周祎晨、孙靖轩、曲展衡、周体豪

完成时间： 2023.6.17

1．任务目标

配置相关环境，基于 **Mnist** 数据集和非数字图像数据集构建一个分类模型。该模型可以对 **mnist** 数据集的图像正确识别代表的数字，同时将非数字的图像识别为 **OOD** 类。要求：

模型是 **11** 分类（**0-9** 代表数字的十个分类和一个 **OOD** 类）；

在 **CPU** 上有合理的运行时间。

实现接口类文件 **oodcls.py**，提供接口函数加载模型文件。并完成相应报告说明等。

2．具体内容

（1）实施方案

1)配置实验环境

本次实验中选用 **jupyter** 中添加的虚拟环境执行。具体步骤如下：

• 以 **pytorch 1.8** 为例。先激活想要添加的虚拟环境并进入，执行命令 **pip install ipykernel ipython** 开始安装。

• 执行 **ipython kernel install (--user --name pytorch1.8)**添加虚拟环境，括号内为自己的名。打开 **jupyter notebok** 即可使用添加的虚拟环境。

• 如过程中缺少环境也执行 **pip install**，注意要安装对应版本。

• 引入相关库并下载数据集。

```
#库的引入
%matplotlib inline
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision

from torchvision import transforms
import torchvision.transforms as transforms
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import CIFAR10
from torch.autograd import Variable
```

2) 具体方案

定义 **OodDataset** 作为数据集类用于加载 **Cifar** 数据集作为带 **OOD** 的数据集。使用 **PyTorch** 的数据加载器将 **Mnist** 和 **Cifar** 作为训练集，并初始化模型和优化器。训练过程即常规的使用交叉熵损失函数计算损失后用 **Adam** 优化器更新模型参数，最后评估模型在不同测试集上的准确率并输出。每一部分的具体过程见下文“核心代码分析”。

（2）核心代码分析

1)准备工作

```python
# 定义OOD数据集类
class OodDataset(Dataset):
    def __init__(self, ood_data, transform=None):
        self.ood_data = ood_data
        self.transform = transform

    def __len__(self):
        return len(self.ood_data)

    def __getitem__(self, idx):
        image, _ = self.ood_data[idx]
        if self.transform:
            image = self.transform(image)
        return image, 10  # OOD类的标签为10
```

定义一个 **OOD** 数据集类，用于加载 **Cifar** 数据集以更改其初始化标签为 **10.**

```python
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1), # 彩色图像转灰度图像num_output_channels默认1
    transforms.RandomCrop((28, 28)),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,)),
])
```

该函数将彩色图像转化为灰度图像，进行裁剪后再将图像数据类型展为 **Pytorch** 张量并归一化，最后进行标准化，提高模型的收敛速度和准确率。

```python
#下载并加载MNIST数据集
mnist_trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
mnist_testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
mnist_testloader = DataLoader(mnist_testset, batch_size=64, shuffle=False)

# 加载CIFAR10数据集作为OOD数据集
cifar10_trainset_1 = CIFAR10(root='./data', train=True, download=True, transform=transform)
cifar10_trainset=OodDataset(cifar10_trainset_1)
cifar10_testset_1 = CIFAR10(root='./data', train=False, download=True, transform=transform)
cifar10_testset=OodDataset(cifar10_testset_1)
ood_testloader = DataLoader(cifar10_testset, batch_size=64, shuffle=False)

#将MNIST数据集和CIFAR10数据集合并为训练集
trainset=torch.utils.data.ConcatDataset([mnist_trainset,cifar10_trainset])
trainloader=DataLoader(trainset,batch_size=64,shuffle=True)
```

加载 **MNIST** 和 **CIFAR** 数据集并将其合并为一个训练集。对数据进行预处理后将训练集和测试集加载到内存中，分批次进行训练和测试。

2)分类器模型的建立

```
#分类器的模型
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.fc1 = nn.Linear(784, 100)
        self.fc2 = nn.Linear(100, 50)
        self.fc3 = nn.Linear(50, 11)


    def forward(self, x):
        x = x.view((-1, 784))
        h = self.fc1(x)
        #h = self.bc1(h)
        h = F.relu(h)
        #h = F.dropout(h, p=0.5, training=self.training)

        h = self.fc2(h)
        #h = self.bc2(h)
        h = F.relu(h)
        #h = F.dropout(h, p=0.2, training=self.training)

        h = self.fc3(h)
        out = F.softmax(h,1)
        return out
```

建立一个分类器模型，输入为一个一维向量，输出是 11 个分类中的一个，最后用 softmax 函数进行归一化处理，以进行在训练时的损失计算和推理预测。

```
#定义优化器
model = Classifier()
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

用前述的 Classifier 类建立模型并随机初始化模型参数，然后用 Adam 优化器 optimizer 采用梯度下降法优化参数，最后定义了交叉熵损失函数用以计算 11 分类的损失。

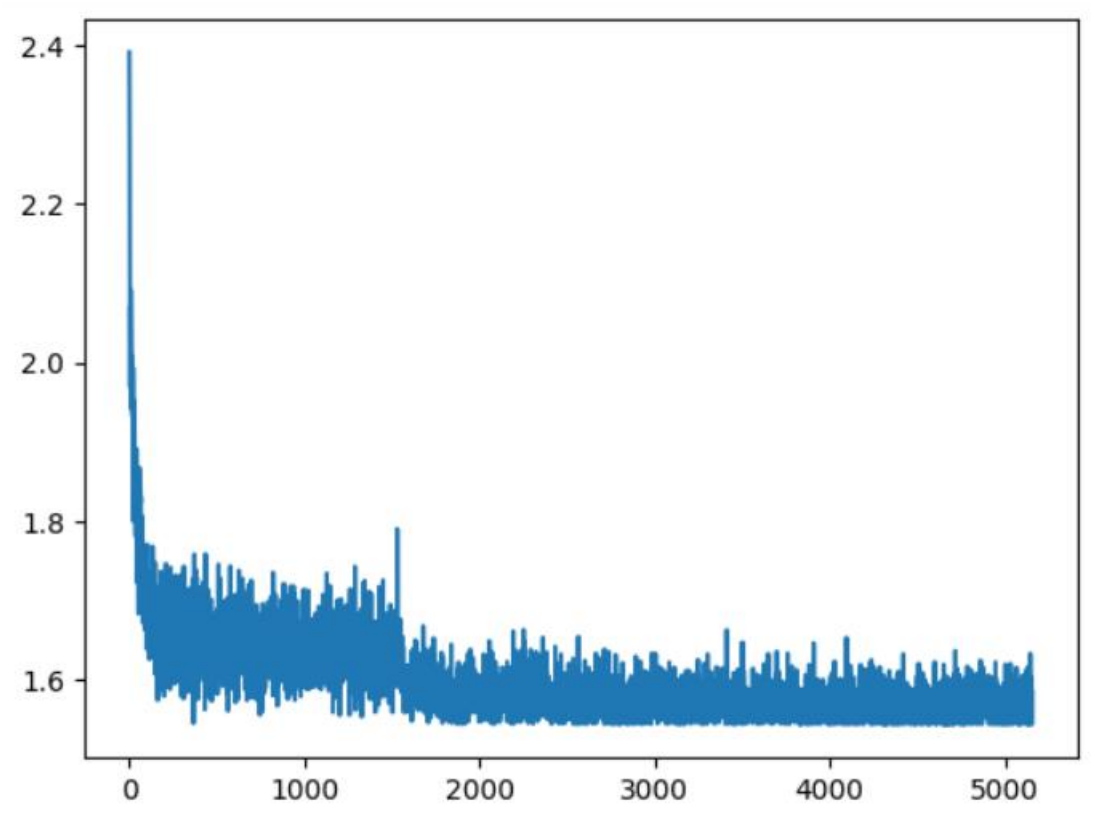
3)训练和测试过程

```python
#模型的训练
model.train()
losses=[]
num_epochs = 3
for epoch in range(num_epochs):
    for batch_idx, (images, labels) in enumerate(trainloader):
        images = Variable(images)
        labels = Variable(labels)

        optimizer.zero_grad()
        outputs = model(images)

        loss = F.cross_entropy(outputs, labels)
        losses.append(loss)
        # Backpropagation
        loss.backward()
        optimizer.step()

        # 训练进度的监测和错误率展示
        if batch_idx % 100 == 1:
            print('\r Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch,
                batch_idx * len(images),
                len(trainloader.dataset),
                100. * batch_idx / len(trainloader),
                loss),
                end='')

    print()
```

训练过程中，先遍历每一批数据，操作为将数据封装为 **Variable** 对象后计算模型输出 **outputs** 和对应的交叉熵损失 **loss**，将 **loss** 记录在 **losses** 数组中以便后续观察 **loss** 的变化。计算导数后更新梯度，最后用函数 **optimizer.step()**更新每个参数的值。

```
Train Epoch: 0 [108864/110000 (99%)]   Loss: 1.548694
Train Epoch: 1 [108864/110000 (99%)]   Loss: 1.588872
Train Epoch: 2 [108864/110000 (99%)]   Loss: 1.577991
```

训练中的错误率变化如下图：




```
#测试minist数据集
model.eval()
num_epochs = 10  # Number of epochs to run
for epoch in range(num_epochs):
    correct = 0
    total = 0
    for images, labels in mnist_testloader:
        images = Variable(images)
        labels = Variable(labels)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)


        correct += (predicted == labels).sum()

    accuracy = 100 * correct / total
    print('Epoch [{}/{}], Accuracy on Mnist Test Set: {:.2f}%'.format(epoch+1, num_epochs, accuracy))
```

测试过程中，将模型转换至评估模式后从测试集中读取数据样本，先进前向传播得到 **outputs**，然后使用 **torch.max** 函数获取预测的类别，最后统计正确的样本数并计算 **accuracy**。

```python
#测试cifar数据集
model.eval()
num_epochs = 10  # Number of epochs to run
for epoch in range(num_epochs):
    correct = 0
    total = 0
    for images, labels in ood_testloader:

        images = Variable(images)
        labels = Variable(labels)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)

        correct += (predicted == labels).sum()



    accuracy = 100 * correct / total
    print('Epoch [{}/{}], Accuracy on OOD Test Set: {:.2f}%'.format(epoch+1, num_epochs, accuracy))
```

在 **OOD** 数据集上测试，遍历中的操作与后续计算和上一段代码相同，最终计算得出准确率。

测试结果的分析过程如下：

```
Epoch [1/10], Accuracy on OOD Test Set: 99.91%
Epoch [2/10], Accuracy on OOD Test Set: 99.89%
Epoch [3/10], Accuracy on OOD Test Set: 99.92%
Epoch [4/10], Accuracy on OOD Test Set: 99.87%
Epoch [5/10], Accuracy on OOD Test Set: 99.92%
Epoch [6/10], Accuracy on OOD Test Set: 99.89%
Epoch [7/10], Accuracy on OOD Test Set: 99.89%
Epoch [8/10], Accuracy on OOD Test Set: 99.88%
Epoch [9/10], Accuracy on OOD Test Set: 99.91%
Epoch [10/10], Accuracy on OOD Test Set: 99.89%
```

4)接口类文件

接口类文件 **oodcl.py** 中先定义了一个和分类器模型一样的 **classifier** 类用于接收 **model.pt** 中训练得到的模型的参数。

```python
class OodCls:
    def __init__(self, model_path='model.pt'):

        self.model = Classifier()
        self.model.load_state_dict(torch.load(model_path))
        self.model.eval()

        self.CIFAR10_MEAN = (0.4914, 0.4822, 0.4465)
        self.CIFAR10_STD = (0.2023, 0.1994, 0.2010)

    def classify(self, testloader):


        self.transform = transforms.Compose([
            transforms.Grayscale(num_output_channels=1), # 彩色图像转灰度图像num_output_channels默认1
            transforms.RandomCrop((28, 28)),
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,)),
        ])

        images=testloader
        images = Variable(images)
        #images = self.transform(images)
        images = images.unsqueeze(0)

        outputs = self.model(images)
        _, predicted = torch.max(outputs.data, 1)

        return predicted
```

定义了一个接口类 OodCls，实现了两个函数__init__和 classify。__init__用于初始化模型，classify 用于接收一组维度为 n*1*28*28 的 tensor 作为参数，输出一个整数型 n 维 tensor 代表分类结果。

3．工作总结

（1）收获、心得

通过本次实验，本小组成员掌握了数据集的下载和使用、模型的构建、训练和完善的一般过程，学习了 Mnist 数据集中的数字图像识别和带 OOD 分类的方法，完成了接口类文件的编写和初始化等任务。了解了深度学习和模型训练的一般过程，能够训练解决较简单问题的相关模型。

（2）遇到问题及解决思路

1)配置环境时已安装了最新版本环境，仍对 import torchvision 等语句报错如下：

以及 DLL load failed while importing _imaging: 找不到指定的模块等错误。

解决方法：把虚拟环境导入 jupyter notebook 并安装 matplotlib、pillow 等。

2)cifar 数据集中的图片的大小不统一且为彩色，不能直接导入训练过程，强行导入会导致 OOD 的测试集准确率过低。

解决方法：在 transforms.Compose 函数加入其它处理函数，先将彩色图像转为灰度图像，然后对图像进行随机裁剪以进行归一化和标准化处理。

3)在以混合数据集作为训练集进行训练时，由于 cifar 数据集的 label 为随机数，导致其损失函数不下降以至于带 OOD 的分类效果不理想。

解决方法：定义一个 OOD 数据类，加载 cifar 数据集时引用这个类将其 label 改为 10 代表 OOD。

4．课程建议

通过本次课程的学习，同学们都受益良多。课程介绍了人工智能的概念、发展历程、重要方法和理论等，帮助了我们了解人工智能的相关知识和未来发展方向。由于课程的大作业是真正实现并训练一个模型而不是论文等考核方式，希望在以后的课程教学过程中，在介绍理论的同时展示一些实际的代码实现等具体过程，这样能将课程内容和大作业的形式更加紧密地结合起来。从对理论知识和方法的了解到具体实操还有一段距离，如果不进行具体实现方面的相关引导，基础普通的同学在完成作业时会略显吃力。