实验四实验报告

1.实验介绍：

本实验使用pytorch完成MNIST任务，采用MLP和CNN两种网络结构完成.本次实验采用kaggle上的MNIST数据集并且提交了结果。

2.代码分析：

两种网络主要不同在于网络模型架构，其余数据载入等部分相同，因此放在一起说明：

- 载入必要的库

```
import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import torch
from torch import nn,functional
import torchvision
from torchvision import datasets,transforms
device = torch.device("cuda:4" if torch.cuda.is_available else "cpu")

from sklearn.model_selection import train_test_split

%matplotlib inline
```

- batch size=100,载入测试数据集并且以8:2的比例划分训练集和验证集

```
train_batch_size =100
validation_batch_size =100

#Loading data
train = pd.read_csv("./input/train.csv")
X = train.loc[:,train.columns != "label"].values/255   #Normalizing the values
Y = train.label.values

features_train, features_test, targets_train, targets_test =
train_test_split(X,Y,test_size=0.2,random_state=42)
X_train = torch.from_numpy(features_train).to(device)
X_validation = torch.from_numpy(features_test).to(device)

Y_train = torch.from_numpy(targets_train).type(torch.LongTensor).to(device)
Y_validation = torch.from_numpy(targets_test).type(torch.LongTensor).to(device)

train = torch.utils.data.TensorDataset(X_train,Y_train)
```

```
validation = torch.utils.data.TensorDataset(X_validation,Y_validation)

train_loader = torch.utils.data.DataLoader(train, batch_size = train_batch_size,
shuffle = False)
validation_loader = torch.utils.data.DataLoader(validation, batch_size =
validation_batch_size, shuffle = False)
```

- 定义网络并实例化模型：

  MLP版本： 采用了两个全连接层，使用softmax进行分类，并使用了drop层 和relu激活函数，采用SGD优化

```
#define network

class MLP(nn.Module):
    def __init__(self):
        super(MLP,self).__init__()

        self.fc1=nn.Linear(28*28,256)
        self.drop1=nn.Dropout(p=0.3)
        self.relu1=nn.ReLU()

        self.fc2=nn.Linear(256,64)
        self.drop2=nn.Dropout(p=0.3)
        self.relu2=nn.ReLU()

        self.fc3 = nn.Linear(64,10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self,x):

        x = x.view(x.size(0), -1)

        out = self.fc1(x)
        out = self.drop1(out)
        out = self.relu1(out)

        out = self.fc2(out)
        out = self.drop2(out)
        out = self.relu2(out)

        out = self.fc3(out)
        out = self.softmax(out)
        return out


#create network criterion and optimizer

model = MLP()
model = model.double()
model = model.to(device)
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),lr=0.1, momentum=0.8)
```

CNN版本：构建了两个卷基层，并采用了maxpool和relu激活函数 为了防止过拟合采用了dropout，优化器采用了SGD

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.cnn_1 = nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 5,
stride=1, padding=0)
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.cnn_2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 5,
stride=1, padding=0)
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

        self.fc1 = nn.Linear(32*4*4,10)
        self.drop=nn.Dropout(p=0.3)
        self.softmax = nn.Softmax(dim=1)

    def forward(self,x):

        out = self.cnn_1(x)
        out = self.relu1(out)
        out = self.maxpool1(out)

        out = self.cnn_2(out)
        out = self.relu2(out)
        out = self.maxpool2(out)

        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.drop(out)
        out = self.softmax(out)
        return out

model = CNN()
model = model.double()
model = model.to(device)
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),lr=0.1, momentum=0.8)
```

- 训练和验证 采用15个epoch

```
#define network

class MLP(nn.Module):
```

```
    def __init__(self):
        super(MLP,self).__init__()

        self.fc1=nn.Linear(28*28,256)
        self.drop1=nn.Dropout(p=0.3)
        self.relu1=nn.ReLU()

        self.fc2=nn.Linear(256,64)
        self.drop2=nn.Dropout(p=0.3)
        self.relu2=nn.ReLU()

        self.fc3 = nn.Linear(64,10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self,x):

        x = x.view(x.size(0), -1)

        out = self.fc1(x)
        out = self.drop1(out)
        out = self.relu1(out)

        out = self.fc2(out)
        out = self.drop2(out)
        out = self.relu2(out)

        out = self.fc3(out)
        out = self.softmax(out)
        return out
```

- 绘图

```
plt.plot(train_losses, label='Training loss')
plt.plot(validation_losses, label='Validation loss')
plt.plot(validation_accs,label='Validation Accuracy')
plt.legend(frameon=False)
```

- 测试集数据载入以及测试

```
test_images = pd.read_csv("./input/test.csv")
test_image = test_images.loc[:,test_images.columns != "label"].values/255
test_dataset = torch.from_numpy(test_image).to(device)
new_test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 1, shuffle =
False)

top_classes = []
with torch.no_grad():
    model.eval()
```

```
    for images in new_test_loader:
        test = images.view(-1,1,28,28).to(device)
        output = model(test)
        ps = torch.exp(output)
        top_p, top_class = ps.topk(1, dim = 1)
        top_classes.append(int(top_class))

predlabel = top_classes
predictions = np.array(predlabel)

submissions=pd.DataFrame({"ImageId": list(range(1,len(predictions)+1)),
                          "Label": predictions})
submissions.to_csv("./output/my_submissions_cnn.csv", index=False, header=True)
```

3.结果分析：

本实验中超参数的选择如下：

batch_size = 100 总epoch次数15次

SGD学习率为 0.1 动量0.8

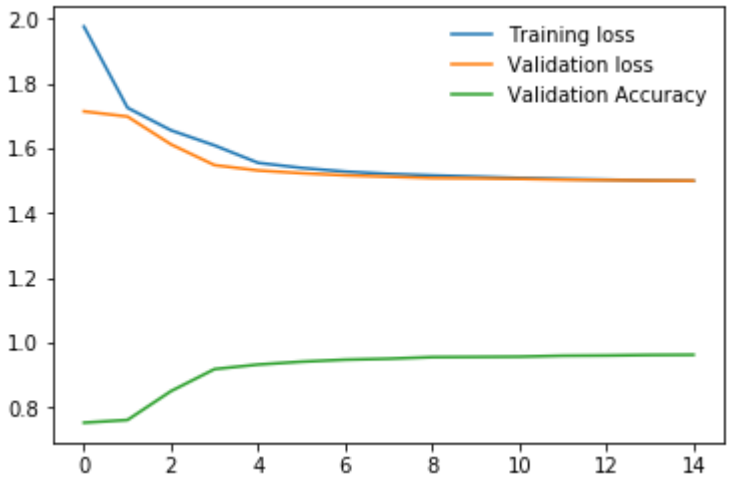
以下为两个网络模型的结果：

MLP:

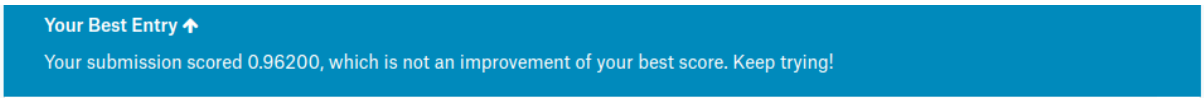
```
Epoch: 1/15..  Training Loss: 1.975..  Validation Loss: 1.714..  Validation Accuracy:
0.753
Epoch: 2/15..  Training Loss: 1.725..  Validation Loss: 1.698..  Validation Accuracy:
0.762
Epoch: 3/15..  Training Loss: 1.655..  Validation Loss: 1.612..  Validation Accuracy:
0.851
Epoch: 4/15..  Training Loss: 1.609..  Validation Loss: 1.548..  Validation Accuracy:
0.919
Epoch: 5/15..  Training Loss: 1.555..  Validation Loss: 1.531..  Validation Accuracy:
0.933
Epoch: 6/15..  Training Loss: 1.539..  Validation Loss: 1.523..  Validation Accuracy:
0.941
Epoch: 7/15..  Training Loss: 1.528..  Validation Loss: 1.516..  Validation Accuracy:
0.948
Epoch: 8/15..  Training Loss: 1.521..  Validation Loss: 1.512..  Validation Accuracy:
0.951
Epoch: 9/15..  Training Loss: 1.516..  Validation Loss: 1.508..  Validation Accuracy:
0.956
Epoch: 10/15..  Training Loss: 1.512..  Validation Loss: 1.507..  Validation Accuracy:
0.956
Epoch: 11/15..  Training Loss: 1.508..  Validation Loss: 1.505..  Validation Accuracy:
0.957
Epoch: 12/15..  Training Loss: 1.505..  Validation Loss: 1.503..  Validation Accuracy:
0.960
```

```
Epoch: 13/15..  Training Loss: 1.503..  Validation Loss: 1.502..  Validation Accuracy:
0.961
Epoch: 14/15..  Training Loss: 1.500..  Validation Loss: 1.500..  Validation Accuracy:
0.962
Epoch: 15/15..  Training Loss: 1.499..  Validation Loss: 1.500..  Validation Accuracy:
0.963
```

Loss和acc曲线:



在kaggle平台上提交了预测结果后 test_acc约为0.96

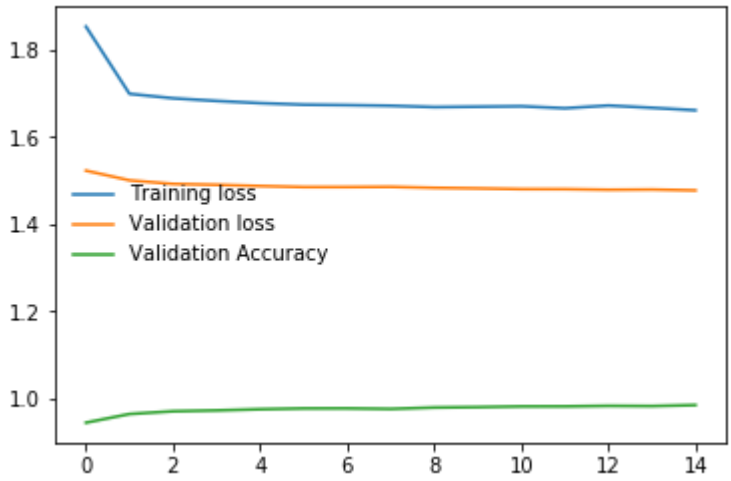


CNN:

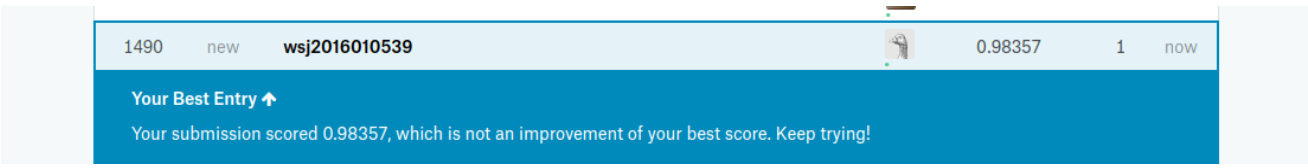
```
Epoch: 1/15..  Training Loss: 1.853..  Validation Loss: 1.522..  Validation Accuracy:
0.944
Epoch: 2/15..  Training Loss: 1.698..  Validation Loss: 1.500..  Validation Accuracy:
0.964
Epoch: 3/15..  Training Loss: 1.688..  Validation Loss: 1.492..  Validation Accuracy:
0.971
Epoch: 4/15..  Training Loss: 1.682..  Validation Loss: 1.490..  Validation Accuracy:
0.973
Epoch: 5/15..  Training Loss: 1.677..  Validation Loss: 1.487..  Validation Accuracy:
0.975
Epoch: 6/15..  Training Loss: 1.674..  Validation Loss: 1.485..  Validation Accuracy:
0.977
Epoch: 7/15..  Training Loss: 1.673..  Validation Loss: 1.485..  Validation Accuracy:
0.977
Epoch: 8/15..  Training Loss: 1.671..  Validation Loss: 1.485..  Validation Accuracy:
0.976
Epoch: 9/15..  Training Loss: 1.668..  Validation Loss: 1.483..  Validation Accuracy:
0.979
```

```
Epoch: 10/15..  Training Loss: 1.669..  Validation Loss: 1.482..  Validation Accuracy:
0.980
Epoch: 11/15..  Training Loss: 1.670..  Validation Loss: 1.480..  Validation Accuracy:
0.982
Epoch: 12/15..  Training Loss: 1.665..  Validation Loss: 1.480..  Validation Accuracy:
0.982
Epoch: 13/15..  Training Loss: 1.672..  Validation Loss: 1.478..  Validation Accuracy:
0.983
Epoch: 14/15..  Training Loss: 1.666..  Validation Loss: 1.479..  Validation Accuracy:
0.982
Epoch: 15/15..  Training Loss: 1.661..  Validation Loss: 1.477..  Validation Accuracy:
0.985
```

Loss和acc曲线：



在kaggle平台上提交了预测结果后 test_acc约为0.983



经过分析可以看见，MLP和CNN都能在MNIST数据集上取得比较好的效果。在验证集上MLP能够取得0.96+的准确率，CNN在验证集上能够取得0.98+的准确率。从网络训练速度上看，MLP相比较CNN架构速度更快一些，而收敛速度都比较快，大约在1第十个epoch下达到收敛。

通过图片我们可以看见，validation loss始终低于training loss，这主要是因为在validation的过程中采用了eval()模式，不再dropout，因此所有的神经元都处于激活状态，所以loss更低。这一现象在cnn上比较明显。

通过不同的网络架构对比，可以发现，CNN在图像分类识别的任务上取得的效果更好.通过validation和test的准确率比较，发现相差不大，未出现过拟合的情况。

本次试验使用pytorch库，pytorch封装的一些结构和函数大大增加了搭建神经网络的方便性，并且支持GPU运算，比较简单易学，相比较前几次的作业，我深深地感到轮子的伟大。