**中山大学计算机学院**

**人工智能**

**本科生实验报告**

**（2022学年春季学期）**

课程名称：Artificial Intelligence

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 实验题目

**设计实现Auto-Encoder的网络结构，并完成如下的分类任务。**

* 利用Auto-Encoder学习MNIST数据集的表征，并用得到的表征进行分类器训练
* 有兴趣的同学可以去了解较复杂的自编码器结构，比如VAE
* 需要在实验报告中画出损失函数自编码器随训练过程的变化曲线
* 每个标签的数据随机选取10张图片展示重构后的图像
* 给出利用该必争进行分类的准确率变化曲线
* 如果采用了新的自编码器结构，需要在报告中给出说明

# 实验内容

1. 算法原理

在本实验中，我选用的自编码器是AE编码器（auto encoder）

Autoencoder 是一种无监督学习算法，通常由编码器和解码器构成，它可以用于数据的降维、特征提取和数据重构等任务。Autoencoder 的基本思想是将输入数据压缩到一个低维空间中（encoder），并通过解码器将其重构回原始空间（decoder）。

Autoencoder 的训练过程通常采用反向传播算法和梯度下降算法。在训练过程中，首先将输入数据传递给编码器，将其映射到低维空间中。然后，将低维空间的表示传递给解码器，将其映射回原始空间。最后，计算重构误差，并使用反向传播算法和梯度下降算法来更新编码器和解码器的参数，以最小化重构误差。

Autoencoder 的一个重要应用是数据降维。通过将输入数据压缩到一个低维空间中，可以发现数据中的主要特征，并减少数据的维度。另一个重要应用是数据重构。通过将输入数据压缩到一个低维空间中，并通过解码器将其重构回原始空间。本实验在识别mnist数据集并进行压缩和重构时用到了这两种应用；而在提取数据特征，并用于训练卷积神经网络识别数据并分类时用到了降维的应用。

1. 伪代码

此为autoencoder的模型代码，在实现过程中，可以在类中另外定义一个encoder函数与decoder函数来指定具体的压缩、解压要求。

输入：一张图片数据化后的二维数组

过程：输出->压缩->解压->输出

输出：重构后的图片的二维数组

class AutoEncoder(nn.Module):

    def \_\_init\_\_(self, dims):

        super().\_\_init\_\_()

        encoders = []

        for i in range(len(dims) - 1):

            encoders.append(nn.Linear(dims[i], dims[i + 1]))

            encoders.append(nn.ReLU())

        self.encoders = nn.Sequential(\*encoders)

        decoders = []

        for i in range(len(dims) - 1, 0, -1):

            decoders.append(nn.Linear(dims[i], dims[i - 1]))

            decoders.append(nn.ReLU())

        self.decoders = nn.Sequential(\*decoders)

    def forward(self, X):

        h = self.encoders(X)

        X\_hat = self.decoders(h)

        return h, X\_hat

下面为神经网络训练的伪代码，训练部分在学习特征并重构与分类器训练中都有用到。

输入：网络net，损失函数criterion，网络优化器optimizer，训练集train——loader

输出：经过训练后的网络

for epoch in range(epochs):

        for step,(x,y) in enumerate(train\_loader):

            output=net(x)

            loss=criterion(output,y)

            optimizer.zero\_grad()

            loss.backward()

            optimizer.step()

1. 关键代码展示（带注释）

自编码器：

class AutoEncoder(torch.nn.Module):

    def \_\_init\_\_(self):

        super(AutoEncoder,self).\_\_init\_\_()

        # 编码器 学习特征

        self.encoder=torch.nn.Sequential(

            torch.nn.Linear(784,256),

            torch.nn.ReLU(),

            torch.nn.Linear(256,128),

            torch.nn.ReLU(),

            torch.nn.Linear(128,36),

            torch.nn.ReLU()

        )

        # 解码器 生成

        self.decoder=torch.nn.Sequential(

            torch.nn.Linear(36,128),

            torch.nn.ReLU(),

            torch.nn.Linear(128,256),

            torch.nn.ReLU(),

            torch.nn.Linear(256,784),

            torch.nn.Sigmoid()

        )

    def forward(self, x):

        batch\_size=x.size(0) # x:[b,1,28,28]

        x=x.view(batch\_size,-1) # flatten

        x=self.encoder(x) # encoder

        x=self.decoder(x) # decoder

        x=x.view(batch\_size,1,28,28) # reshape

        return x

    def forward\_(self, x):

        batch\_size=x.size(0) # x:[b,1,28,28]

        x=x.view(batch\_size,-1) # flatten

        x=self.encoder(x) # encoder

        x=x.view(batch\_size,1,6,6) # reshape

        return x

其中，forward是图像压缩重构中所用函数，forward是在训练好网络后的提取特征函数。

获取数据集，初始化网络，定义损失函数以及网络优化器以后，开始训练：

    train\_list, test\_list=download\_data()

    epochs=2 #epochs次数

    device=torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    net=AutoEncoder().to(device)

    criterion=torch.nn.MSELoss() #定义损失函数

    optim=torch.optim.Adam(net.parameters(),lr=1e-3) #定义网络优化器

    train1(net,criterion,optim,epochs)

    test1(3)

训练函数1（针对图像压缩及解压）：

def train1(net,criterion,optim,epochs):

    loss=[]

    for epoch in range(epochs):

        for batch\_id,(input,\_) in enumerate(train\_list):

            net.train()

            output=net(input)

            loss\_one=criterion(output,input)

            optim.zero\_grad() #在更新前清除上一步的梯度

            loss\_one.backward() #loss反向传播

            optim.step() #优化器更新

            loss.append(loss\_one.item()) #损失值记录

            print("TRAIN 1 epoch:",epoch,"data id:",batch\_id+1,"loss:",loss\_one.item())

    x=[i for i in range(0,len(loss))]

    plt.plot(x,loss)

    plt.title("loss graph")

    plt.show()

在训练完成后，开始测试：

while 1:

        for i in range(32): # label维度是32

            if labels[i]==goal:

                plt.subplot(2,10,id+1)

                plt.imshow(input[i].numpy().squeeze(), cmap='gray\_r')

                plt.xticks([])

                plt.yticks([])

                plt.subplot(2,10,id+11)

                plt.imshow(output[i].numpy().squeeze(), cmap='gray\_r')

                plt.xticks([])

                plt.yticks([])

                id+=1

            if id==10: break

        if id==10: break

        else:

            input,labels=next(data)

            with torch.no\_grad(): output=net(input)

此函数可以得到原图像以及重构后的图像对比。

在训练完autoencoder以后，可以利用AE提取mnist数据集的特征，并用于cnn神经网络的训练中，以下是分类器训练。先定义好各参数，网络、损失函数、网络优化器同1

train\_loader,test\_loader=data\_processing()

    cnn=CNN().to(device)

    criterion=torch.nn.CrossEntropyLoss()

    optimizer=torch.optim.Adam(cnn.parameters(),lr=1e-3)

    train2(cnn,criterion,optimizer,epochs)

其中，data\_processing()是数据预处理函数，其将mnist数据集中的内容压缩后打包。内容比较重复，因此这里仅展示训练集部分的压缩，测试集部分是同理的。

def data\_processing(): #用AE提取特征 并压缩为loader

    train\_features = []

    train\_labels = []

    ……

    with torch.no\_grad():

        for data in train\_list:

            inputs, labels = data

            encoded = net.forward\_(inputs.view(inputs.size(0), -1))

            train\_features.append(encoded)

            train\_labels.append(labels)

        for data in test\_list:

           ……

    # 合并成一维

    train\_features = torch.cat(train\_features, dim=0).numpy()

    train\_labels = torch.cat(train\_labels, dim=0).numpy()

    ……

    train\_features = torch.from\_numpy(train\_features).float()

    train\_labels = torch.from\_numpy(train\_labels).long()

    ……

    train\_dataset = TensorDataset(train\_features, train\_labels)

    train\_loader = DataLoader(train\_dataset, batch\_size=10, shuffle=True)

    ……

    return train\_loader,test\_loader

然后是分类器的训练和测试函数。为了测试准确率随训练次数的变化，这里直接将测试函数放在了训练函数中。

def test2(net):

    correct=0

    total=0

    with torch.no\_grad():

        for \_,(x,y) in enumerate(test\_loader):

            outputs=net(x)

            \_,pred=torch.max(outputs.data,dim=1)

            total+=y.size(0)

            correct+=(pred==y).sum().item()

    return correct/total

def train2(cnn,criterion,optimizer,epochs):

    accuracy=[]

    for epoch in range(epochs):

        for step,(x,y) in enumerate(train\_loader):

            output=cnn(x)

            loss=criterion(output,y)

            optimizer.zero\_grad()

            loss.backward()

            optimizer.step()

            print("TRAIN 2,epoch:",epoch,"data id:",step+1,"loss:",loss.item())

            if step%50==0:

                rate=test2(cnn)

                accuracy.append(rate)

    x=[i for i in range(0,len(accuracy))]

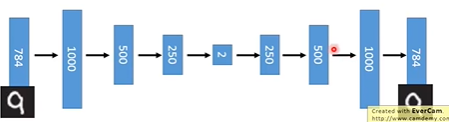
    plt.plot(x,accuracy)

    plt.title("accuracy graph")

    plt.show()

1. 创新点&优化（如果有）

在训练AE模型时，我采用了先扩张图片大小再进行压缩的操作，这一能使图像特征更易被提取。思路如下所示：



代码：

self.encoder=torch.nn.Sequential(

            torch.nn.Linear(784,1000),

            torch.nn.ReLU(),

            torch.nn.Linear(1000,500),

            torch.nn.ReLU(),

            torch.nn.Linear(500,256),

            torch.nn.ReLU(),

            torch.nn.Linear(256,128),

            torch.nn.ReLU(),

            torch.nn.Linear(128,36),

            torch.nn.ReLU()

        )

        self.decoder=torch.nn.Sequential(

            torch.nn.Linear(36,128),

            torch.nn.ReLU(),

            torch.nn.Linear(128,256),

            torch.nn.ReLU(),

            torch.nn.Linear(256,500),

            torch.nn.ReLU(),

            torch.nn.Linear(500,1000),

            torch.nn.ReLU(),

            torch.nn.Linear(1000,784),

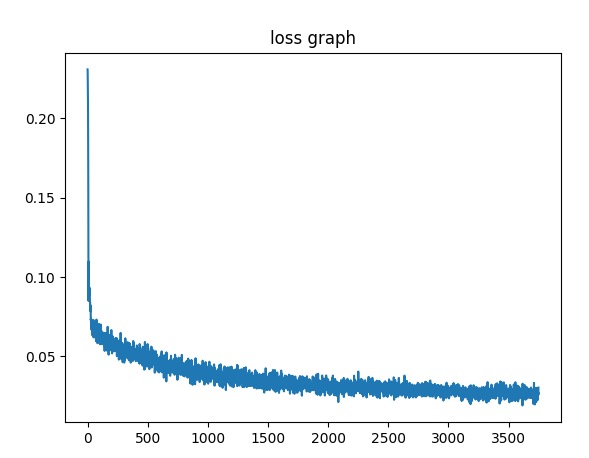
            torch.nn.Sigmoid()

# 实验结果及分析

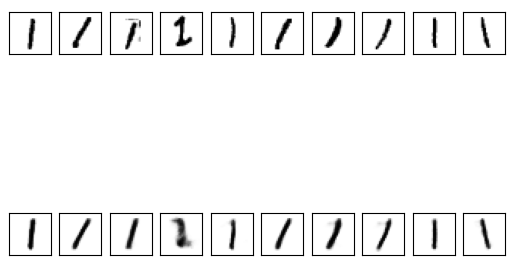
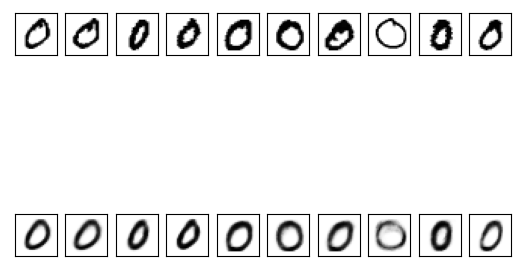
1. 实验结果展示示例（可图可表可文字，尽量可视化）

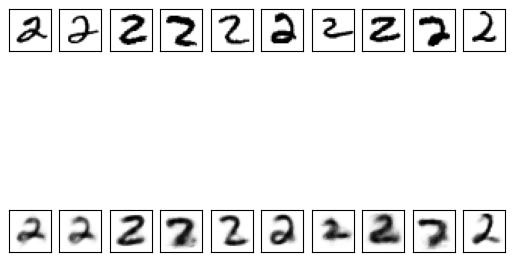
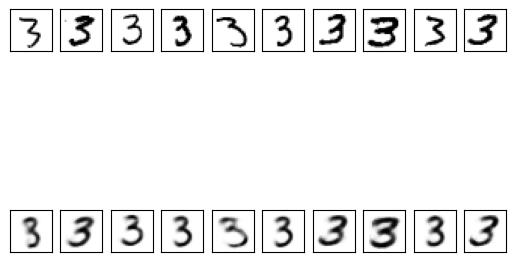
先展示训练AE的过程。

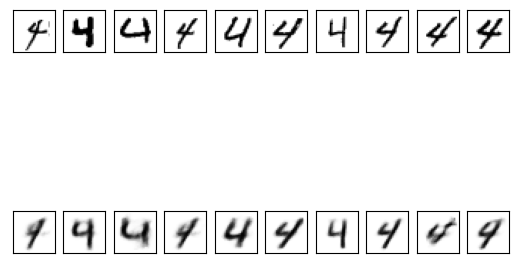
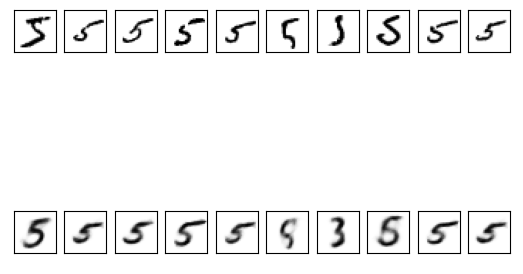
**以下是其中一次训练时的loss损失函数：**

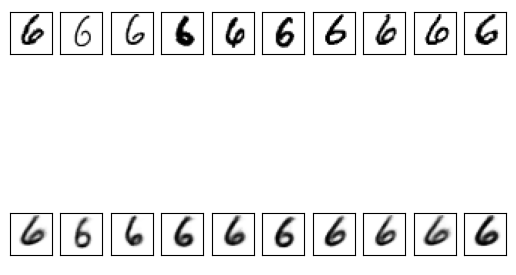
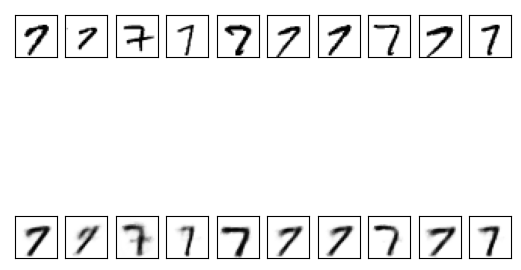


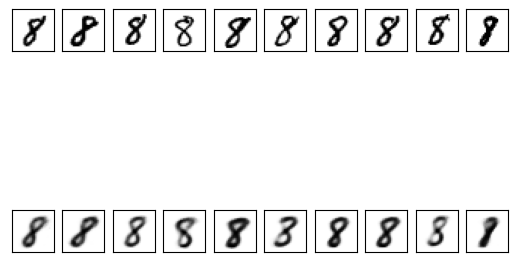
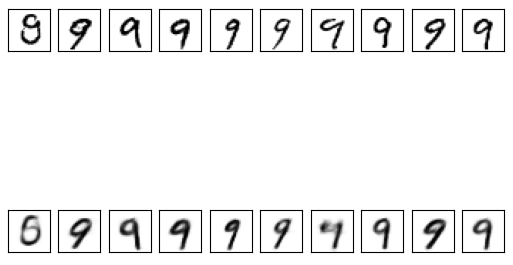
**以下是0~9的数字图像重构，第一行为原始图像，第二行为重构图像：**



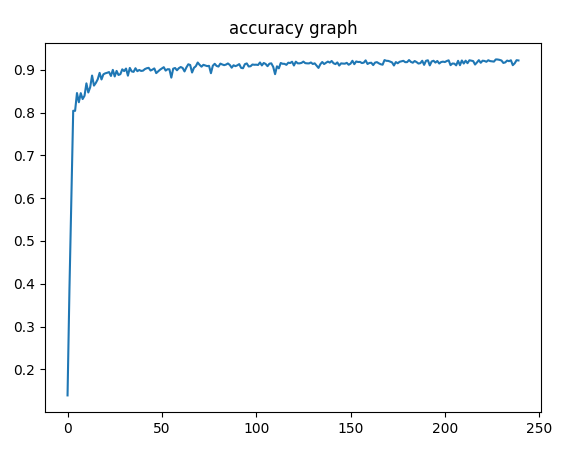
 

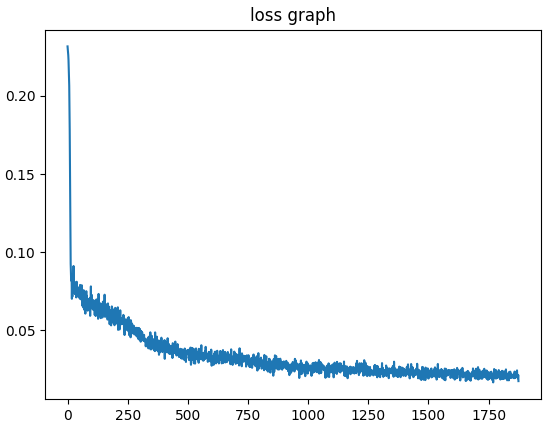
 

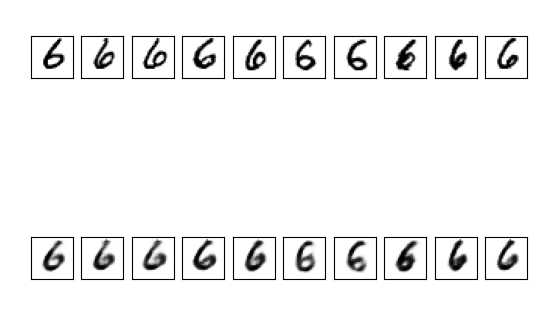
**以下是正确率随训练次数变化曲线：**

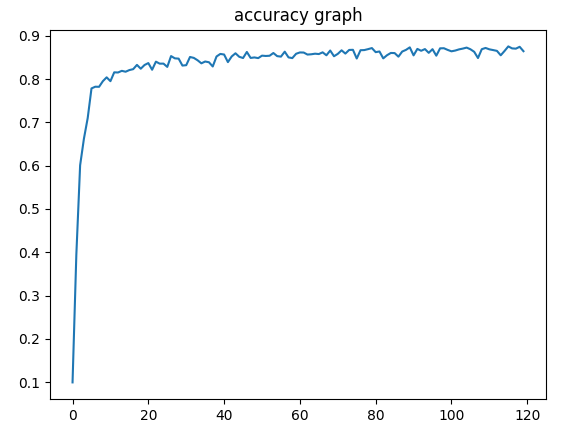


2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

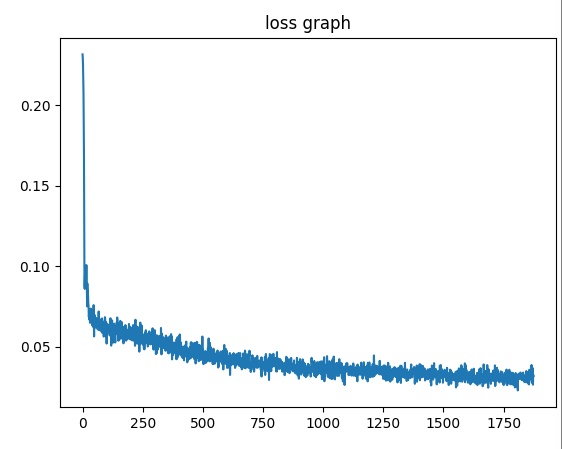
当AE中并未扩展图像时，运行结果如下：

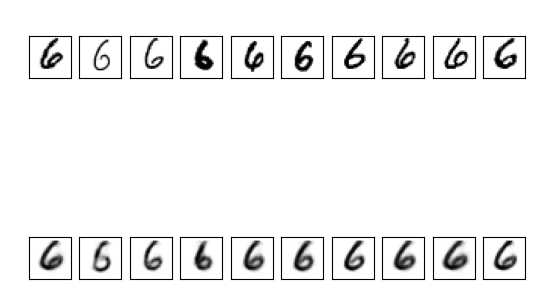


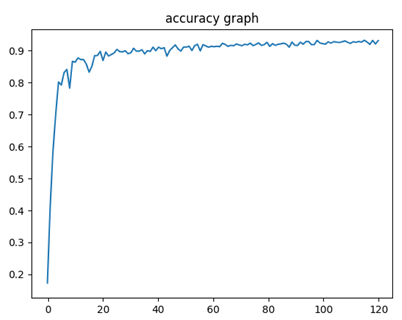




当AE中扩展图像时，运行结果如下：







**可见改进过后正确率很快就能突破90%**

# 参考资料

[使用AutoEncoder对MNIST手写数据集进行表征学习（pytorch）](https://blog.csdn.net/weixin_46321755/article/details/127578301)

[自编码器AutoEncoder处理Mnist数据集](http://edu.ichenhua.cn/read/319)

超算习堂-课时30-第十六讲实验课