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	人工智能（陈川）	专业（方向）	计算机科学与技术（人工智能与大数据方向）
学号	20337025	姓名	崔璨明

目录

一、 实验题目

设计 Auto-Encoder 的网络结构，并完成分类任务：

- 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
- 给出利用该特征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

二、 实验内容

1. 算法原理

1.1 AutoEncoder

AutoEncoder 是一种无监督的学习算法，主要用于数据的降维或者特征的抽取，在深度学习中，AutoEncoder 可用于在训练阶段开始前，确定权重矩阵的初始值。

神经网络中的权重矩阵可看作是对输入的数据进行特征转换，即先将数据编码为另一种形式，然后在此基础上进行一系列学习。然而，在对权重初始化时，我们并不知道初始的权重值在训练时会起到怎样的作用，也不知道在训练过程中权重会怎样的变化。因此一种较好的思路是，利用初始化生成的权重矩阵进行编码时，我们希望编码后的数据能够较好的保留原始数据的主要特征。如果编码后的数据能够较为容易地通过解码恢复成原始数据，我们则认为权重矩阵较好的保留了数据信息。

简单来说，AutoEncoder（自编码）是一种无监督学习的算法，他利用反向传播算法，让目标值等于输入值。输入一张图片，通过一个 Encoder 神经网络，输出一个降维（压缩）后的特征。之后将这个特征通过一个 Decoder 网络，又可以将这张图片还原。

1.2 Variational AutoEncoders

Variational AutoEncoders（VAE）提供了一种概率分布的描述形式，VAE 中 Encoder 描述的是每个潜在属性的概率分布，而不是直接输出一个值。除此之外，VAE 会在输入的数据中添加一些噪音，使得在噪音范围内的图片可以被还原，这样 VAE 可以产生了输入数据中不包含的数据，（可以认为产生了含有某种特定信息的新的数据），而 AE 只能产生尽可能接近或者就是以前的数据（当数据简单时，编码解码损耗少时）。

2. 伪代码

设计 AutoEncoder 网络结构的伪代码如下：

```
输入：原始图片a
输出：降维后提取的特征f，重构的图片x
def Encoder(a):{
    a=Linear(784, 256)
    ReLU(a)
    a=Linear(256, 64)
    ReLU(a)
    a=Linear(64, 20)
    ReLU(a)
    return a
}

def Decoder(f):{
    f=Linear(20, 64)
    ReLU(f)
    f=Linear(64, 256)
    ReLU(f)
    f=Linear(256, 784)
    Sigmoid(f)
    return f
}
def AE(a):
    f=Encoder(a)
    x=Decoder(f)
    return f,x
```

3. 关键代码展示（带注释）

3.1 数据集的加载

数据集采用 mnist 数据集，直接调用 torchvision 库中的 datasets，下载 mnist 数据集，其关键代码如下：

```
#下载mnist数据集
mnist_train = datasets.MNIST('mnist', train=True, transform=transforms.Compose([
    transforms.ToTensor()
]), download=True)
mnist_train = DataLoader(mnist_train, batch_size=32, shuffle=True)


mnist_test = datasets.MNIST('mnist', train=False, transform=transforms.Compose([
    transforms.ToTensor()
]), download=True)
mnist_test = DataLoader(mnist_test, batch_size=32)
```

3.2 自编码器的设计

调用 torch，依照实验给出的网络架构构建 AutoEncode 神经网络，神经网络包括编码网络和译码网络，在编码网络中对图像进行特征提取，该网络包括三个全连接层，每个全连接层后面跟有激活函数层，最后提取出的特征向量的维数为 20。在译码网络中根据图像特征进行图像的复原，该网络包括三个全连接层，同样每个全连接层后面跟有激活函数层。该部分的关键代码如下：

```python
class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()
        #编码器的网络结构
        self.encoder = nn.Sequential(
            # [b, 784] => [b, 256]
            nn.Linear(784, 256),
            nn.ReLU(),
            # [b, 256] => [b, 64]
            nn.Linear(256, 64),
            nn.ReLU(),
            # [b, 64] => [b, 20]
            nn.Linear(64, 20),
            nn.ReLU()
        )
        #译码器的网络结构
        self.decoder = nn.Sequential(
            # [b, 20] => [b, 64]
            nn.Linear(20, 64),
            nn.ReLU(),
            # [b, 64] => [b, 256]
            nn.Linear(64, 256),
            nn.ReLU(),
            # [b, 256] => [b, 784]
            nn.Linear(256, 784),
            nn.Sigmoid()
        )
```

```python
def forward(self, x):
    batchsz = x.size(0)
    #展平
    x = x.view(batchsz, -1)
    #编码
    x = self.encoder(x)
    #译码
    x = self.decoder(x)
    #将其恢复原来的格式
    x = x.view(batchsz, 1, 28, 28)
    return x
```

3.3 自编码器训练过程

用 mnist 数据集中的训练集数据对 AutoEncoder 进行训练，关键代码如下：

```python
#训练过程
def Train(epochs_num,_model,_criterion,_optim,_exp_lr_scheduler,_trian_loader):
    _model.train()
    _exp_lr_scheduler.step()
    loss_set=[]
    for ep in range(epochs_num):
        for i,(img,label) in enumerate(_trian_loader):
            output=_model(img)
            loss=_criterion(output,img)

            _optim.zero_grad()
            #反向传播
            loss.backward()
            _optim.step()
            print("Epoch:{}/{}, step:{}, loss:{:.4f}".format(ep + 1, epochs_num, i + 1, loss.item()))
            loss_set.append(loss.item())
    return loss_set
```

3.4 自编码器测试过程

用 mnist 数据集中的测试集数据对 AutoEncoder 进行测试，即进行图像的复原，并和原来图像进行对比，关键代码如下：

```python
#测试过程
def Test():
    model.eval()
    N=4
    M=8
    #迭代器
    dataiter = iter(mnist_test)
    images, labels = dataiter.next()
    with torch.no_grad():
        _images = model(images)
    p1=plt.figure(1)
    for i in range(32):
        plt.subplot(N,M,i+1)#表示第i张图片，下标只能从1开始，不能从0
        plt.imshow(images[i].numpy().squeeze(), cmap='gray_r')
        plt.xticks([])
        plt.yticks([])
    p2=plt.figure(2)
    for i in range(32):
        plt.subplot(N,M,i+1)#表示第i张图片，下标只能从1开始，不能从0
        plt.imshow(_images[i].numpy().squeeze(), cmap='gray_r')
        plt.xticks([])
        plt.yticks([])
    plt.show()
```

3.5 程序主函数

```python
if __name__=='__main__':
    device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model=AE().to(device)
    epochs_num=1

    #损失函数
    criterion=nn.MSELoss()
    #学习率
    learn_rate=1e-3
    optim=torch.optim.Adam(model.parameters(),lr=learn_rate)


    # 定义学习率调度器：输入包装的模型，定义学习率衰减周期step_size，gamma为衰减的乘法因子
    exp_lr_scheduler = lr_scheduler.StepLR(optim, step_size=6, gamma=0.1)
    #进行训练，返回损失函数的变化过程
    loss_set=Train(epochs_num,model,criterion,optim,exp_lr_scheduler,mnist_train)
    index = [i for i in range(len(loss_set))]
    fig1 = plt.figure(1)
    plt.plot(index,loss_set)
    plt.xlabel("Train times")
    plt.ylabel("Loss")
    plt.show()

    Test2()
```

4. 创新点&优化

除了实现 AutoEncoder 之外，我还对 Variational AutoEncoders（VAE），并进行了简单的实现，关键代码如下：

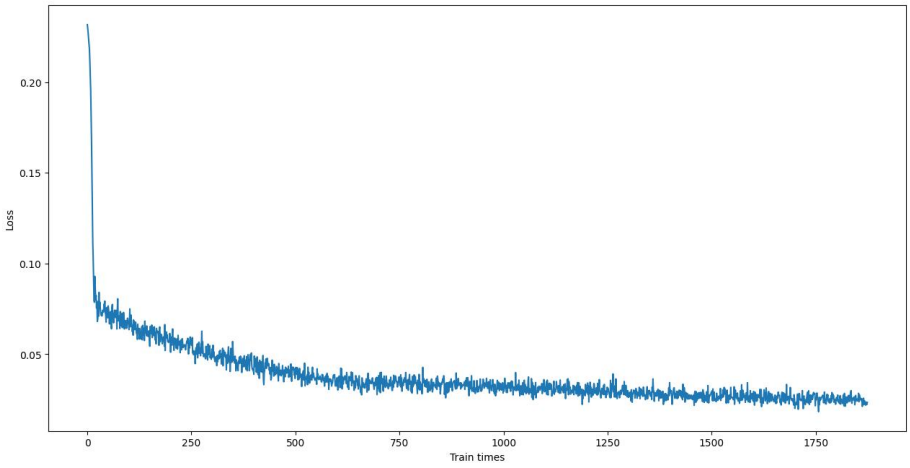
```python
#vae的网络结构
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            # [b, 784] => [b, 256]
            nn.Linear(784, 256),
            nn.ReLU(),
            # [b, 256] => [b, 64]
            nn.Linear(256, 64),
            nn.ReLU(),
            # [b, 64] => [b, 20]
            nn.Linear(64, 20),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            # [b, 10] => [b, 64]
            nn.Linear(10, 64),
            nn.ReLU(),
            # [b, 64] => [b, 256]
            nn.Linear(64, 256),
            nn.ReLU(),
            # [b, 256] => [b, 784]
            nn.Linear(256, 784),
            nn.Sigmoid()
        )
```

```python
def forward(self, x):
    batchsz = x.size(0)
    # flatten
    x = x.view(batchsz, -1)
    # encoder
    q = self.encoder(x)
    mu, sigma = q.chunk(2, dim=1)
    q = mu + sigma * torch.randn_like(sigma)
    # decoder
    x_hat = self.decoder(q)
    # reshape
    x_hat = x_hat.view(batchsz, 1, 28, 28)
    kld = 0.5 * torch.sum(
        torch.pow(mu, 2) +
        torch.pow(sigma, 2) -
        torch.log(1e-8 + torch.pow(sigma, 2)) - 1
    ) / (batchsz*28*28)
    return x_hat, kld
```

三、 实验结果及分析

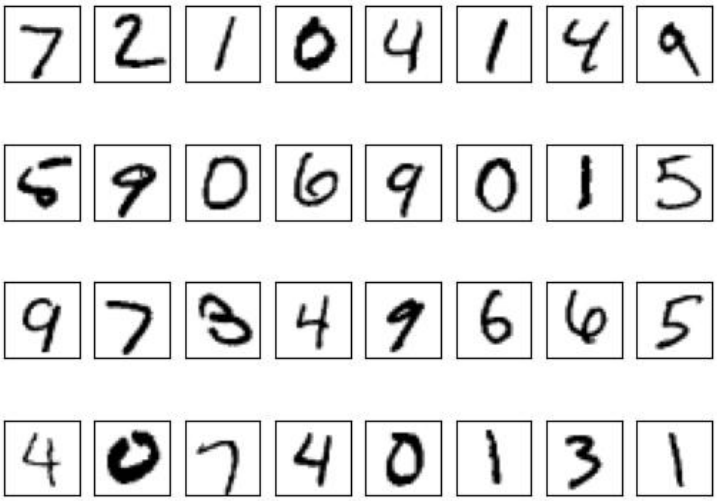
1. 实验结果展示示例（可图可表可文字，尽量可视化）

1.1 损失函数变化图：

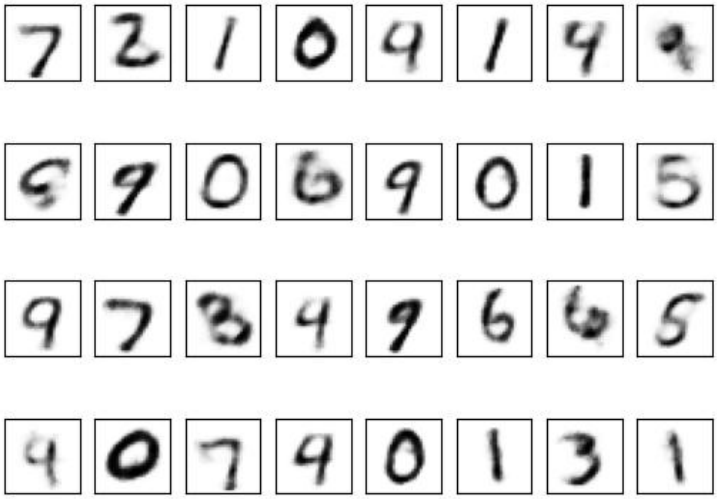


1.2 重建图片展示

原始手写数字图片：



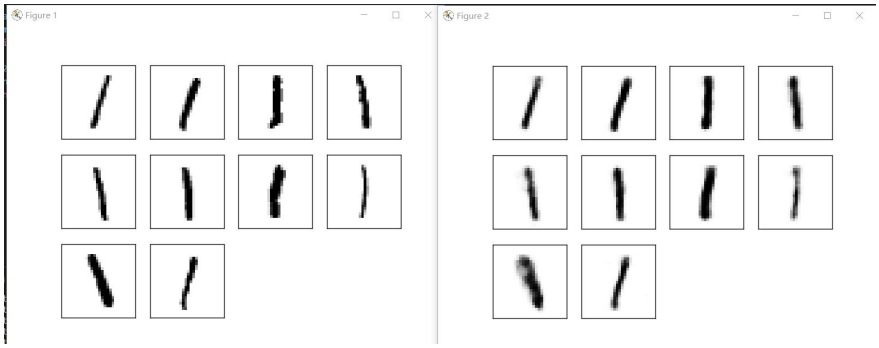
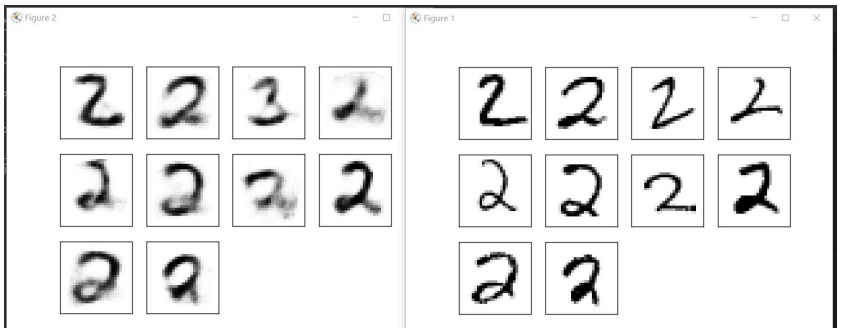
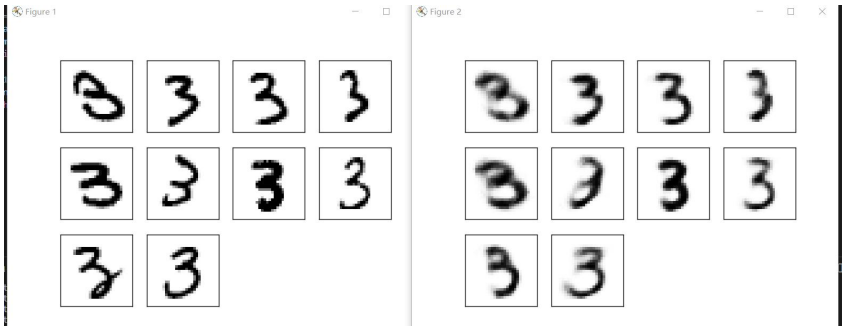
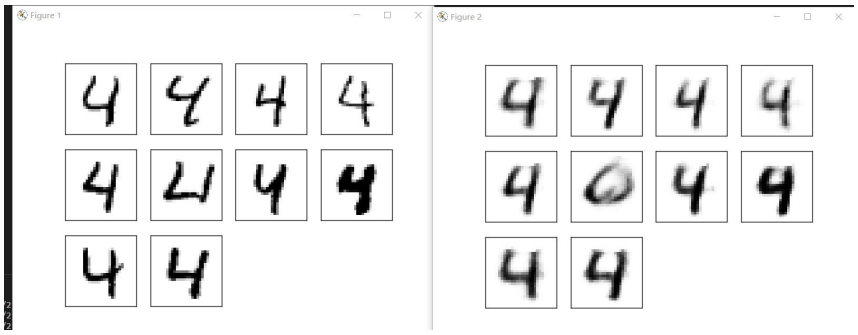
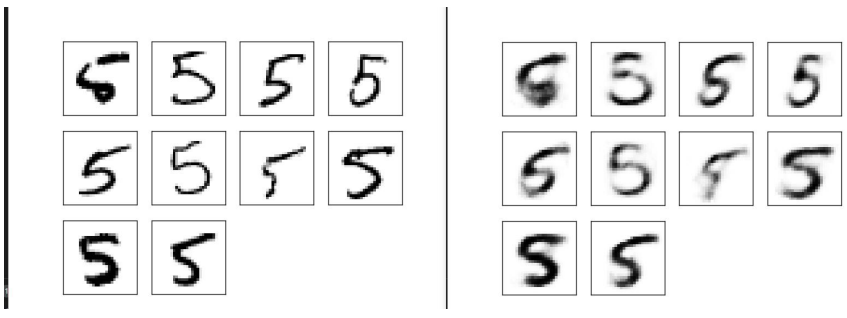
经过自编码器后的重建图片：



可见自编码器能够很好地提取图片的特征，这从复原的图片和原图片非常相近可以看出。

1.3 每个数字的重建展示

按照实验要求，每个标签的数据随机选取 10 张图片展示重构后的图像，结果展示如下：

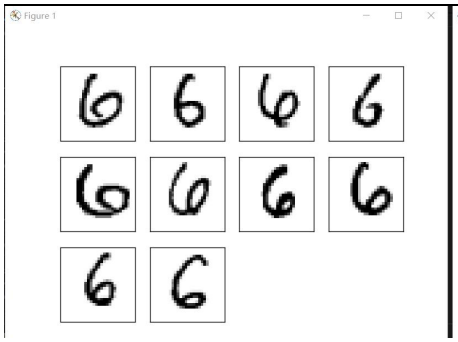


Figure 1

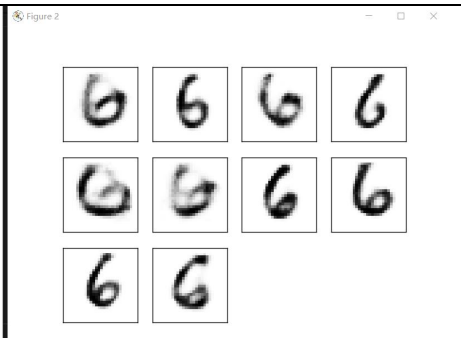


Figure 2

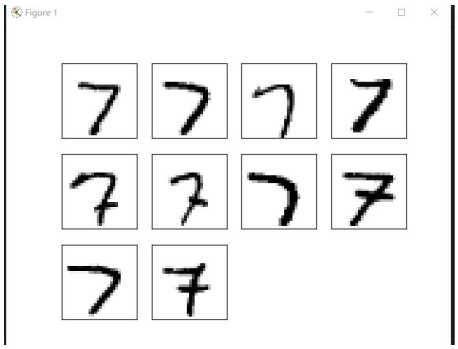


Figure 1

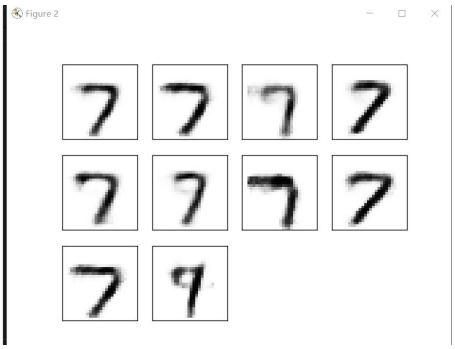


Figure 2

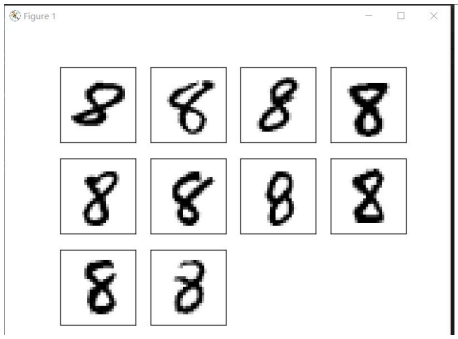


Figure 1

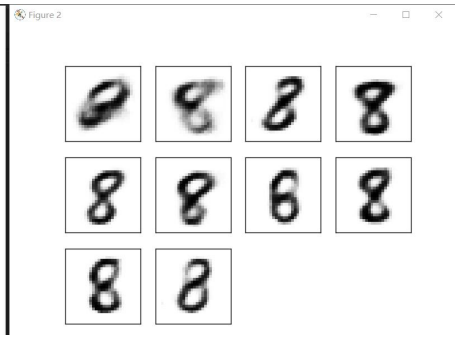


Figure 2

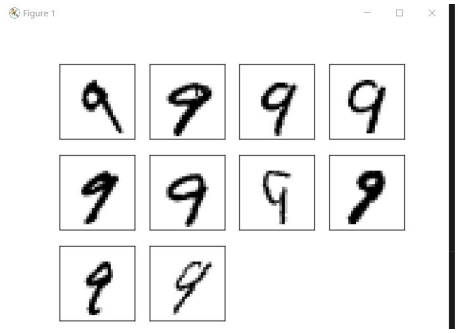


Figure 1

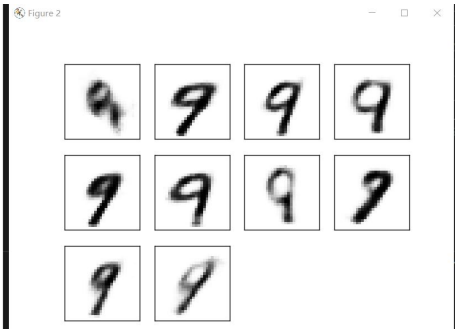


Figure 2

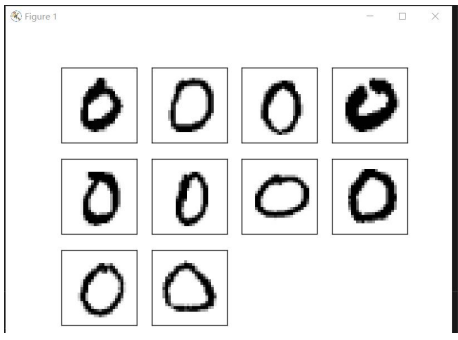


Figure 1

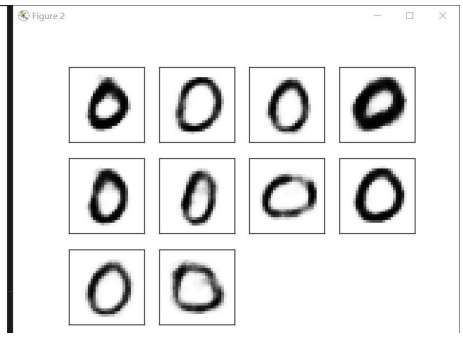
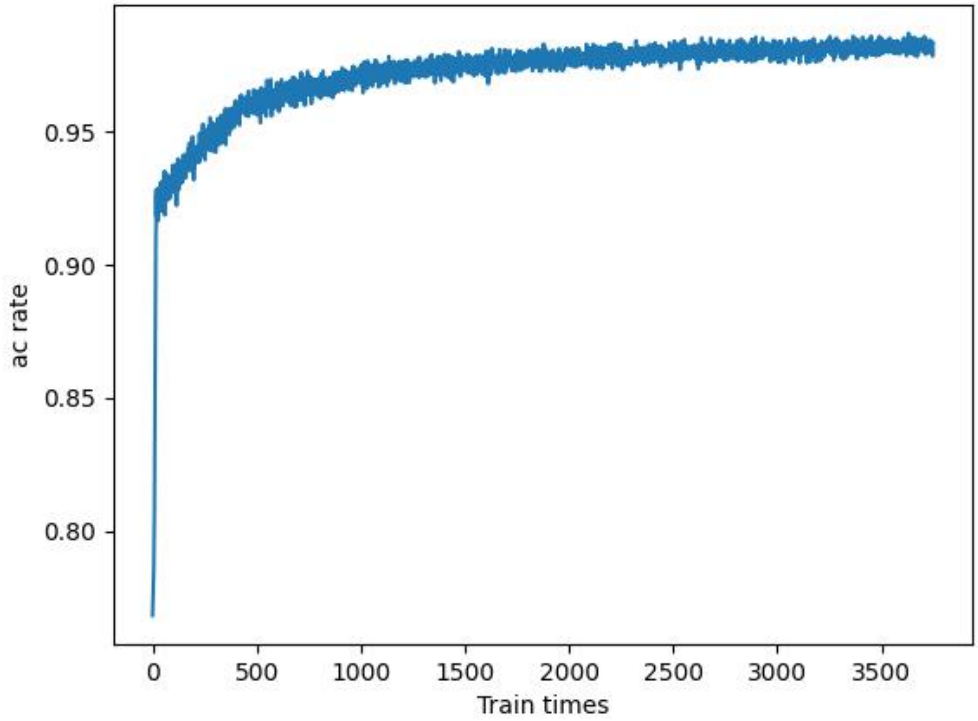


Figure 2

1.5 分类准确率变化图



四、 参考资料

1.(131 条消息) VAE 模型基本原理简单介绍_smile-yan 的博客-CSDN 博客_vae 模型