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	信息与计算科学班	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪

目录

一、实验题目

设计实现 Auto-Encoder 的网络结构，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

二、实验内容

1. 算法原理

AutoEncoder:

AutoEncoder（自动编码器）是一种**无监督学习算法**，用于**数据降维**、**特征提取**和**生成新的数据样本**。它由一个编码器和一个解码器组成，通过将输入数据压缩成低维表示（编码）并将其解码回原始形式来工作。**编码器**将输入数据转换为低维表示，通常称为编码。这个过程涉及到将输入数据传递给一系列的隐藏层，每个隐藏层都有一组学习参数（权重和偏置），用于将输入数据转换为更低维度的表示。编码器的目标是捕获输入数据中的关键特征，并且尽可能地减少信息丢失。**解码器**接收编码器的输出，并试图将其解码为原始输入数据的重构。解码器也由一系列的隐藏层组成，每个隐藏层也有一组学习参数，用于将编码的表示映射回原始输入的空间。解码器的目标是尽可能准确地重建输入数据，并且与原始输入保持一致。AutoEncoder 使用**损失函数**来度量重构的准确度。常见的损失函数是均方误差，它度量重构样本与原始样本之间的差异。训练过程中，AutoEncoder 的目标是最小化损失函数，从而使解码器能够生成与输入数据尽可能相似的重构。**训练过程**包括两个阶段：编码阶段和解码阶段。在编码阶段，输入数据通过编码器，并产生编码表示。在解码阶段，编码表示通过解码器，并生成重构数据。通过比较重构数据和原始输入数据，计算损失函数，并使用反向传播算法来更新编码器和解码器的参数。

AutoEncoder 可以用于多个任务，其中最常见的任务之一是降维。通过训练 AutoEncoder，可以学习到输入数据的紧凑表示，其中包含原始数据的关键特征。这种低维表示可以用于数据可视化、特征提取或后续任务的输入。

Variational AutoEncoder:

Variational Autoencoder（VAE）与传统的 **Autoencoder（AE）**相比，引入了概率分布的描述形式，使得编码器输出的是每个潜在属性的概率分布，而不仅仅是一个确定性的数值。这样的概率分布表示了潜在属性的不确定性，使得模型更加灵活和表达力强。

此外，VAE 在输入数据中添加一些噪音，这样可以使得模型能够还原在噪音范围内的图片。这种噪音的引入有两个作用：一是增加了模型的鲁棒性，使得模型能够处理噪声数据；二是允许 VAE 生成与输入数据有一定差异但仍符合数据分布的新数据样本。这样，VAE 能够产生具有某种特定信息的新数据，而传统的 AE 则更倾向于生成与原始数据相似的重构数据。

2. 伪代码

AutoEncoder：

输入：数据集 $x^{(1)},\ldots,x^{(N)}$

输出：编码器 f_ϕ，解码器 g_θ

初始化参数 ϕ,θ

repeat

计算重构误差：$E=\sum_{i=1}^{N}\left\|x^{(i)}-g_\theta\left(f_\phi\left(x^{(i)}\right)\right)\right\|$

梯度下降法(批量梯度下降法)，更像参数 ϕ,θ

until 参数 ϕ,θ 收敛

3. 关键代码展示（带注释）

AutoEncoder 结构：

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

    def forward(self, x):
        batchsz = x.size(0)
```

```
        x = x.view(batchsz, -1)  # 将输入数据展平为一维向量
        x = self.encoder(x)  # 经过编码器得到特征表示
        features = x.clone()  # 保存特征表示
        x = self.decoder(x)  # 经过译码器重构数据
        x = x.view(batchsz, 1, 28, 28)  # 将数据重新转换为图片的形状
        return x, features
```

获取数据集：

```
mnist_train = datasets.MNIST('mnist', train=True, transform=transforms.ToTensor(),
download=True)
mnist_train = DataLoader(mnist_train, batch_size=64, shuffle=True, num_workers=4)

mnist_test = datasets.MNIST('mnist', train=False, transform=transforms.ToTensor(),
download=True)
mnist_test = DataLoader(mnist_test, batch_size=64, num_workers=4)
```

训练过程：

```
# 训练
def train(epochs, model, criterion, optimizer, scheduler, train_loader):
    model.train()  # 将模型设置为训练模式
    loss_set = []

    for epoch in range(epochs):
        scheduler.step()  # 调整学习率
        total_loss = 0.0  # 用于累计每个 epoch 的损失值

        for i, (img, label) in enumerate(train_loader):
            img = img.to(device)  # 将输入数据移动到 GPU
            label = label.to(device)  # 将标签数据移动到 GPU
            optimizer.zero_grad()  # 梯度归零，清除之前的梯度信息
            output, features = model(img)  # 通过模型进行前向传播，得到输出和特征表示
            loss = criterion(output, img)  # 计算损失函数值
            loss.backward()  # 反向传播，计算梯度
            optimizer.step()  # 更新模型参数
            total_loss += loss.item()  # 累加损失函数值
            _, predicted = torch.max(features.data, 1)  # 获取预测结果中概率最高的类别

            # 打印当前的训练信息
            print("Epoch: {}/{}, Step: {}, Loss: {:.4f}".format(epoch + 1, epochs, i +
1, loss.item()))
            loss_set.append(loss.item())
    return loss_set
```

展示重构后的图像：

```
# 自编码
def Image_all(model, test_loader):
    model.eval()  # 将模型设置为评估模式
    N = 8  # 子图的行数
    M = 8  # 子图的列数

    with torch.no_grad():  # 不需要计算梯度
        images, _ = next(iter(test_loader))  # 获取测试数据集的一个 batch
        images = images.to(device)  # 将测试数据移动到指定设备 GPU
        _images, _ = model(images)  # 通过模型进行前向传播，得到重构图像

    p1 = plt.figure(1)  # 创建第一个图形窗口

    # 绘制原始图像
    for i in range(N * M):
        plt.subplot(N, M, i + 1)
        plt.imshow(images[i].cpu().numpy().squeeze(), cmap='gray_r')
        plt.xticks([])
        plt.yticks([])

    p2 = plt.figure(2)  # 创建第二个图形窗口

    # 绘制重构图像
    for i in range(N * M):
        plt.subplot(N, M, i + 1)
        plt.imshow(_images[i].cpu().numpy().squeeze(), cmap='gray_r')
        plt.xticks([])
        plt.yticks([])

    plt.show()  # 显示图像
```

每个标签随机取 10 张重构后的图像：

```
# 取样输出
def Image_sep(num):
    model.eval()  # 将模型设置为评估模式
    N = 2  # 子图的行数
    M = 5  # 子图的列数
    num_images = 10  # 要显示的图像数量
    cnt = 0  # 计数器，用于控制显示的图像数量

    p1 = plt.figure(1)
    p2 = plt.figure(2)
```

```
    dataiter = iter(mnist_test)  # 创建测试数据集的迭代器
    images, labels = next(dataiter)  # 获取测试数据集的一个 batch 的图像和标签

    while cnt < num_images:
        for i, (image, label) in enumerate(zip(images, labels)):
            if label == num:
                plt.figure(1)
                plt.subplot(N, M, cnt + 1)
                plt.imshow(image.cpu().numpy().squeeze(), cmap='gray_r')
                plt.xticks([])
                plt.yticks([])

                with torch.no_grad():
                    image = image.unsqueeze(0).to(device)
                    reconstructed_image, _ = model(image)

                plt.figure(2)
                plt.subplot(N, M, cnt + 1)
                plt.imshow(reconstructed_image.squeeze().cpu().numpy(),cmap='gray_r')
                plt.xticks([])
                plt.yticks([])

                cnt += 1
                if cnt == num_images:
                    Break

        try:
            images, labels = next(dataiter)  # 获取下一个 batch 的图像和标签
        except StopIteration:
            Break

    plt.show()
```

分类测试：

```
def test(model, dataloader, criterion):
    model.eval()  # 设置为评估模式
    running_loss = 0.0  # 用于累计每个 batch 的损失值
    correct = 0
    total = 0

    with torch.no_grad():  # 不需要计算梯度
        for images, labels in dataloader:
            images = images.to(device)
            labels = labels.to(device)
```

```
            reconstructions, features = model(images)  # 通过模型进行前向传播，得到预测结果
            labels = labels.unsqueeze(1)  # 保持标签维度为 [batch_size, 1]
            loss = criterion(features, images)  # 计算损失函数值
            running_loss += loss.item()  # 累加损失函数值
            _, predicted = features.max(1)  # 获取预测结果中概率最高的类别
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    accuracy = correct / total  # 计算准确率
    return accuracy
```

主函数：

```
if __name__ == '__main__':
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = AutoEncoder().to(device)  # 创建自编码器模型，并将其移动到指定设备上
    epochs_num = 2  # 训练的总轮数
    num_epochs = 50  # 测试的总轮数
    test_accuracy_set = []  # 存储测试准确率的列表
    criterion = nn.MSELoss()  # 定义损失函数
    learn_rate = 1e-3  # 学习率
    optimizer = torch.optim.Adam(model.parameters(), lr=learn_rate)  # 创建优化器
    scheduler = lr_scheduler.StepLR(optimizer, step_size=6, gamma=0.1)  # 设置学习率衰减策略
    loss_set = train(epochs_num, model, criterion, optimizer, scheduler, mnist_train)  # 进行训练并返回损失值列表
    index_train = [i for i in range(len(loss_set))]  # 创建用于绘制训练损失曲线的横坐标列表
    fig1 = plt.figure(1)
    plt.plot(index_train, loss_set)
    plt.xlabel("Steps")
    plt.ylabel("Loss")
    plt.show()

    Image_all(model, mnist_test)  # 显示所有测试图像和对应的重构图像

    for _num in range(0, 10):
        Image_sep(_num)  # 分别显示每个数字类别的原始图像和对应的重构图像

    for epoch in range(num_epochs):
        test_accuracy = test(model, mnist_test, criterion)  # 计算测试准确率
        test_accuracy_set.append(test_accuracy)  # 将测试准确率添加到列表中
```

```
    index_test = [i for i in range(len(test_accuracy_set))]  # 创建用于绘制测试准确率曲线的横坐标列表
    fig2 = plt.figure(2)
    plt.plot(index_test, test_accuracy_set)  # 绘制测试准确率曲线
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.show()
```

4. 创新点&优化（如果有）

学习并尝试实现 VAE。

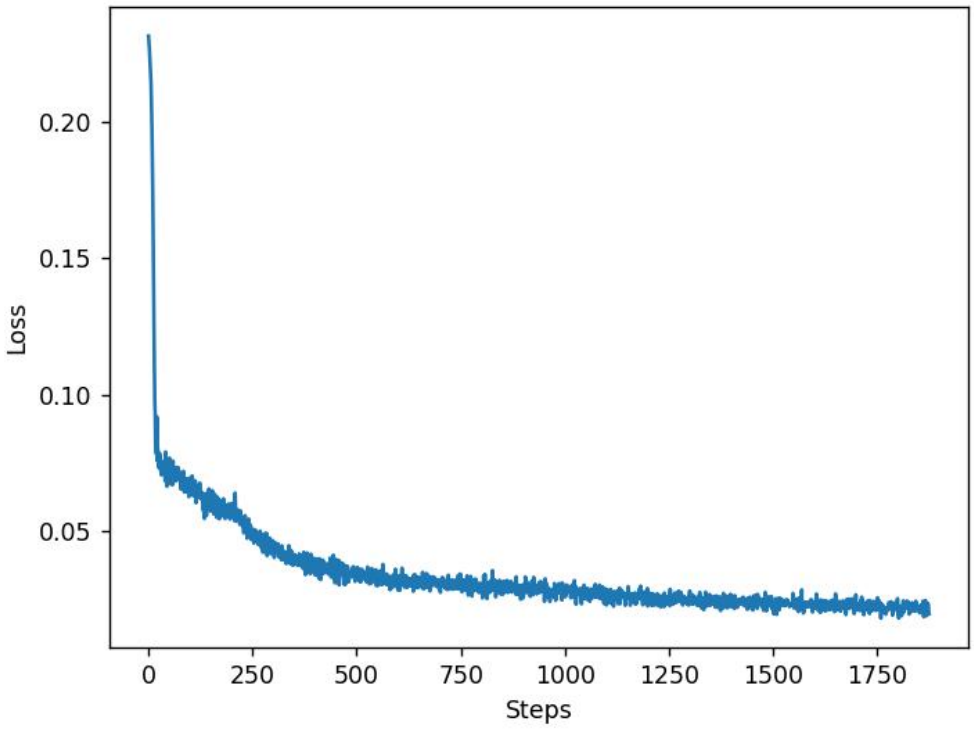
```
# 变分自编码器的网络结构
class VariationalAutoencoder(nn.Module):
    def __init__(self):
        super(VariationalAutoencoder, self).__init__()
        # 编码器的网络结构
        self.encoder = nn.Sequential(
            nn.Linear(784, 256),
            nn.ReLU(),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Linear(64, 20),
            nn.ReLU()
        )
        # 解码器的网络结构
        self.decoder = nn.Sequential(
            nn.Linear(10, 64),
            nn.ReLU(),
            nn.Linear(64, 256),
            nn.ReLU(),
            nn.Linear(256, 784),
            nn.Sigmoid()
        )
        # 均值和方差的线性层
        self.fc_mu = nn.Linear(20, 10)
        self.fc_logvar = nn.Linear(20, 10)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        z = mu + eps * std
        return z
```

```
def forward(self, x):
    batchsz = x.size(0)
    x = x.view(batchsz, -1)
    x = self.encoder(x)
    mu = self.fc_mu(x)
    logvar = self.fc_logvar(x)
    z = self.reparameterize(mu, logvar)
    features = z.clone()
    x = self.decoder(z)
    x = x.view(batchsz, 1, 28, 28)
    return x, mu, logvar, features
```

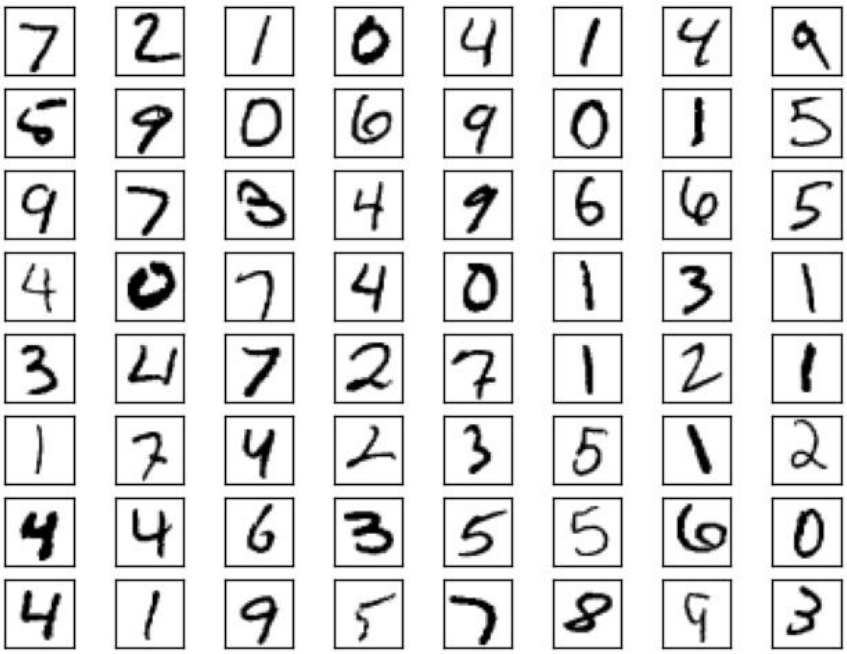
三、实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

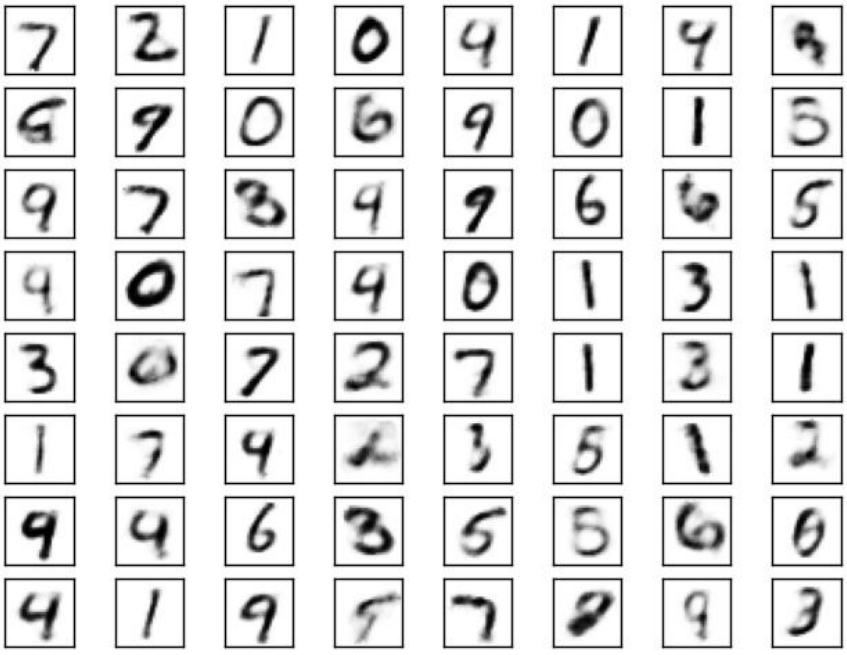
损失函数变化曲线：



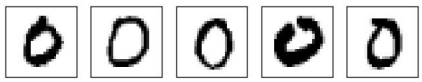
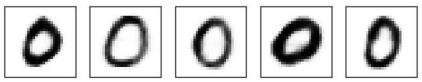
初始一批图像：

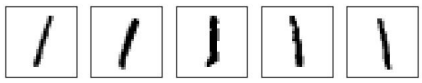


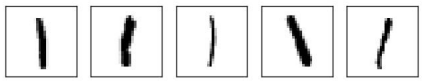
重构后一批图像：



每个数字的展示：（左边为原图像，右边为重构图像）

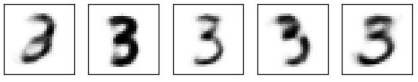



















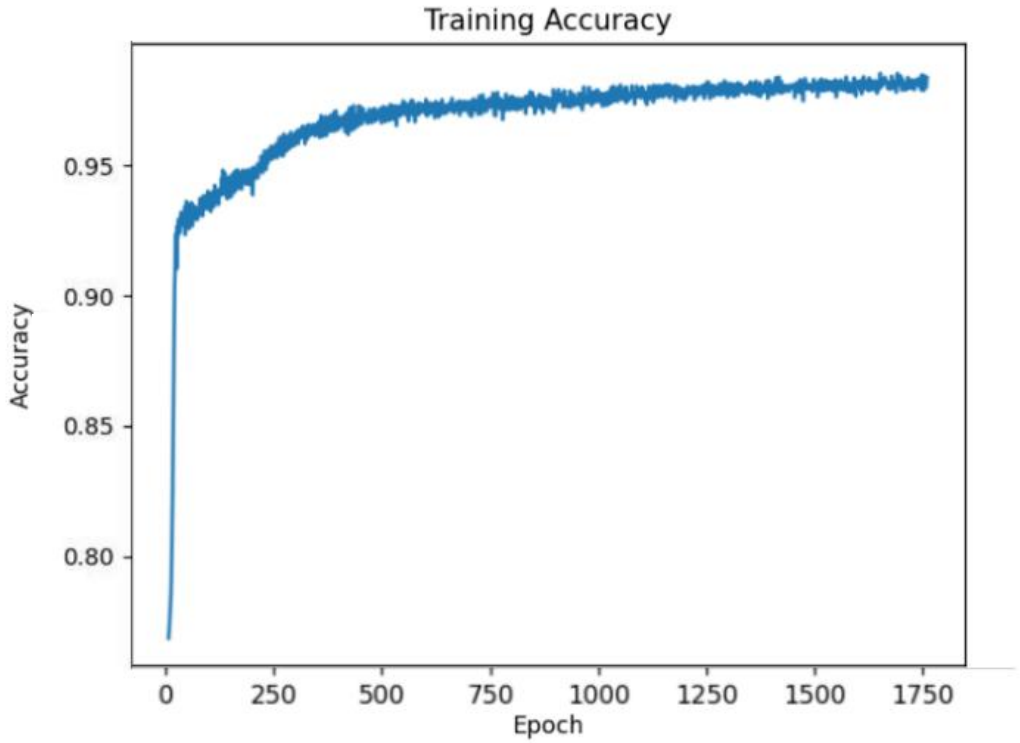






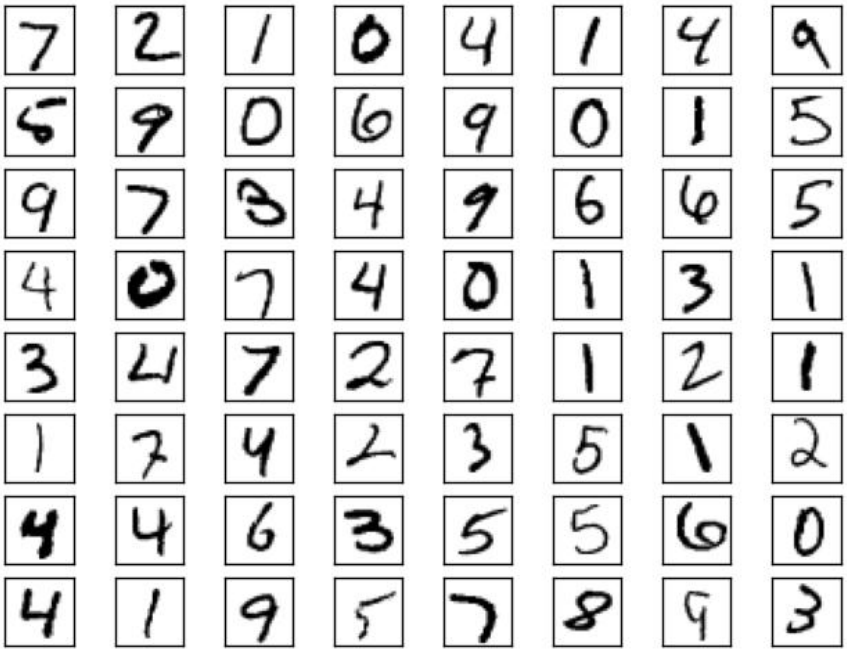

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

分类准确率变化曲线：



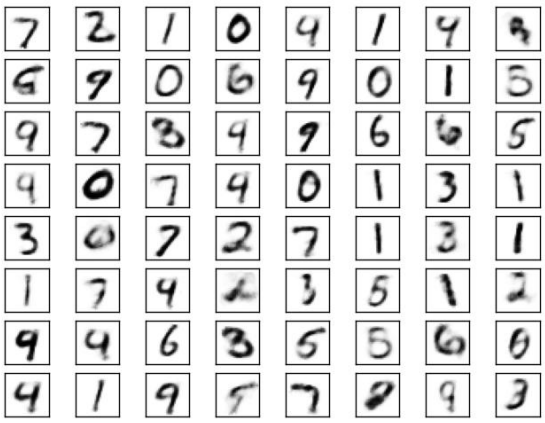
关于 AE 和 VAE 的重构对比实验：

原图像：

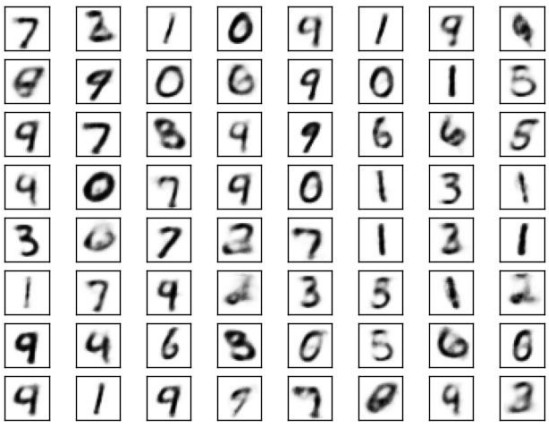


重构图像：

AE:



VAE:



肉眼可见，用 VAE 重构的图像比用 AE 重构的图像保留信息更多，且更清晰。并且 VAE 重构的图像出现了原图像中不包含的部分。

四、参考资料

1、
http://t.csdn.cn/YF1Hy

2、
https://www.bilibili.com/video/BV1Yx411976B/?spm_id_from=333.337.search-card.all.click&vd_source=fe8a61bc42546259ff5da93db830dc1a

3、
http://t.csdn.cn/8dgVR