2022机器学习期末大作业

基于VAE的手写数字生成

- 姓名：刘伟　　张朝阳　　边笛
- 学号：2013029　2011746　2012668

实验内容

1. 基本要求：利用 pytorch 神经网络框架编程实现变分自编码器 (VAE)；分别采用交叉熵损失 (Cross Entropy Loss) 和L2损失 (MSE Loss) 构建损失函数生成新的手写数字,分析结果并展示生成的手写数字
2. 中级要求：实现VAE的变分推导，描述VAE的由来以及优缺点
3. 高级要求：实现对VAE进行改进或创新，有VAE的变种——CVAE、β-VAE的实现，可视化方法创新——隐变量输出的实现

成员贡献

- 共同部分：学习了解VAE变分自编码器的框架结构，掌握基本的理论前提。
- 刘伟：pytorch构建基本的Dense全连接层VAE；可视化的创新——隐变量层的输出展示；变种——CVAE条件变分自编码器
- 张朝阳：pytorch构建结合Conv卷积层的VAE结构；深入学习VAE的理论推导——复现VAE的变分推导
- 边笛：VAE的变种创新——β-VAE，全面分析β对整个VAE结构的影响并结合隐变量层的学习成果深入分析

初级要求

利用 pytorch 神经网络框架编程实现变分自编码器 (VAE)

库函数与训练数据的加载

In [ ]:
```
# 导入实验所需的相关库函数
import torch
import torch.utils.data as Data
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torchvision.datasets import MNIST
from torchvision.utils import save_image
from torchsummary import summary
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# 下载实验所需的MNIST数据集
train_dataset = MNIST(
    root='MNIST',
    train=True,
    transform=transforms.ToTensor(),
    download=True
)

# 查看mnist数据
print(train_dataset)

# 查看是否支持cuda GPU训练
print(torch.cuda.is_available())

# 数据集切分
batch_size = 128
dataIter = Data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: MNIST
    Split: Train
    StandardTransform
Transform: ToTensor()
True
```

代价损失函数：通过俩种不同的方式构建损失函数

In [ ]:
```
# 展平：多维数据展平成低维数据
def flatten(x):
    N = x.shape[0]  # read in N, C, H, W
    return x.view(N, -1)  # torch.Size([128, 784])


# L2损失 (MSE Loss) 构建损失函数
def vae_loss_mse(x, gen_x, mean, log_var):
    # 重构项损失
    mse_loss = torch.nn.MSELoss(reduction='sum')
    loss1 = mse_loss(gen_x, x)

    # 最小化 q(z|x)  和 p(z) 的距离
    KL_loss = 0.5 * torch.sum(torch.exp(log_var) + torch.pow(mean, 2) - log_var - 1

    return loss1 + KL_loss


# 交叉熵损失 (Cross Entropy Loss) 构建损失函数
def vae_loss_cross(x, gen_x, mean, log_var):
    # 重构项损失
    loss1 = F.binary_cross_entropy(gen_x, x, reduction='sum')

    # 最小化 q(z|x)  和 p(z) 的距离
    KL_loss = 0.5 * torch.sum(torch.exp(log_var) + torch.pow(mean, 2) - log_var - 1

    return loss1 + KL_loss
```

构造网络模型

- 采用全连接Dense(Linear)设计网络架构
- 采用卷积层Conv+Dense设计网络架构

全连接层——VAE网络

In [ ]:
```
class VariationalAutoEncoder_Dense(nn.Module):
    def __init__(self,
                 image_shape=(1, 28, 28),
                 dense_size=(784, 256, 128),
                 z_dim=20):
        super(VariationalAutoEncoder_Dense, self).__init__()

        # 激活函数
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

        # encoder全连接层
        self.enDense0 = nn.Linear(dense_size[0], dense_size[1])
        self.enDense1 = nn.Linear(dense_size[1], dense_size[2])

        # 隐变量层
        self.latent_mean = nn.Linear(dense_size[-1], z_dim)
        self.latent_log_var = nn.Linear(dense_size[-1], z_dim)

        # decoder全连接层
        self.deDense0 = nn.Linear(z_dim, dense_size[-1])
        self.deDense1 = nn.Linear(dense_size[-1], dense_size[-2])
        self.deDense2 = nn.Linear(dense_size[-2], dense_size[-3])

        # decoder reshape
        self.dec_reshape = transforms.Lambda(
            lambda x: torch.reshape(x, (x.shape[0], 1, int(image_shape[1]), int(ima

    # 编码
    def encode(self, x):
        # 展平
        out = flatten(x)
        # 全连接层
        out = self.relu(self.enDense0(out))
        out = self.relu(self.enDense1(out))

        # 均值 mean
        latent_mean = self.latent_mean(out)
        # log方差 log_var
        latent_log_var = self.latent_log_var(out)

        return latent_mean, latent_log_var

    # 重参数化生成隐变量
    def re_parameterize(self, mu, log_var):
        var = torch.exp(0.5 * log_var)
        epsilon = torch.randn_like(var)
        return mu + torch.mul(var, epsilon)

    # 解码
    def decode(self, z):
        # 全连接层
        out = self.relu(self.deDense0(z))
        out = self.relu(self.deDense1(out))
        out = self.sigmoid(self.deDense2(out))

        # 重构shape
```

```
        out = self.dec_reshape(out)

        return out

    # 整个前向传播过程：编码 --> 解码
    def forward(self, x):
        latent_mean, latent_log_var = self.encode(x)
        sampled_Z = self.re_parameterize(latent_mean, latent_log_var)
        Gen_X = self.decode(sampled_Z)
        return latent_mean, latent_log_var, Gen_X
```

卷积层——VAE网络

In [ ]:

```
class VariationalAutoEncoder_Conv(nn.Module):
    def __init__(self,
                 image_shape=(1, 28, 28),
                 conv_size=(1, 6, 16, 4),
                 dense_size=(784, 128),
                 kernel=3,
                 z_dim=20):
        super(VariationalAutoEncoder_Conv, self).__init__()

        # 激活函数
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

        # encoder卷积层
        # 输入 (128,1,28,28) 输出 (128,6,28,28)
        self.enConv0 = nn.Conv2d(conv_size[0], conv_size[1], kernel, padding=1, padd
        # 输入 (128,6,28,28) 输出 (128,16,14,14)
        self.enConv1 = nn.Conv2d(conv_size[1], conv_size[2], kernel, stride=2, paddi
        # 输入 (128,16,14,14) 输出 (128,4,14,14)
        self.enConv2 = nn.Conv2d(conv_size[2], conv_size[3], kernel, padding=1, padd

        # encoder全连接层
        self.enDense0 = nn.Linear(dense_size[0], dense_size[1])

        # 隐变量层
        self.latent_mean = nn.Linear(dense_size[-1], z_dim)
        self.latent_log_var = nn.Linear(dense_size[-1], z_dim)

        # decoder全连接层
        self.deDense0 = nn.Linear(z_dim, dense_size[-1])
        self.deDense1 = nn.Linear(dense_size[-1], dense_size[-2])
        self.deDense2 = nn.Linear(dense_size[-2], int(np.prod(image_shape)))

        # 需要对 全连接层输出的一维结果数据 进行 reshape
        self.dec_reshape = transforms.Lambda(
            lambda x: torch.reshape(x, (x.shape[0], 1, int(image_shape[1]), int(ima

    # 编码
    def encode(self, x):
        # 卷积层
        out = self.relu(self.enConv0(x))
        out = self.relu(self.enConv1(out))
        out = self.relu(self.enConv2(out))

        # 展平
        out = flatten(out)

        # 全连接层
        out = self.relu(self.enDense0(out))
```

```
        # 均值 mean
        latent_mean = self.latent_mean(out)
        # log方差 log_var
        latent_log_var = self.latent_log_var(out)

        return latent_mean, latent_log_var

    # 重参数化生成隐变量
    def re_parameterize(self, mu, log_var):
        var = torch.exp(0.5 * log_var)
        epsilon = torch.randn_like(var)
        return mu + torch.mul(var, epsilon)

    # 解码
    def decode(self, z):
        # 全连接层
        out = self.relu(self.deDense0(z))
        out = self.relu(self.deDense1(out))
        out = self.sigmoid(self.deDense2(out))

        # 重构shape
        out = self.dec_reshape(out)

        return out

    # 整个前向传播过程：编码 --> 解码
    def forward(self, x):
        latent_mean, latent_log_var = self.encode(x)
        sampled_Z = self.re_parameterize(latent_mean, latent_log_var)
        Gen_X = self.decode(sampled_Z)
        return latent_mean, latent_log_var, Gen_X
```

查看网络结构

In [ ]:

```
# 网络训练设备：建议使用cuda
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

# 利用summary函数查看网络的层次结构
print("==========================VAE(Dense)网络结构==========================")
model_Dense = VariationalAutoEncoder_Dense()
model_Dense.to(device)
summary(model_Dense, input_size=(1, 28, 28))
print("==========================VAE(Conv)网络结构==========================")
model_Conv = VariationalAutoEncoder_Conv()
model_Conv.to(device)
summary(model_Conv, input_size=(1, 28, 28))
```

```
cuda
==========VAE(Dense)网络结构==========
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 256]         200,960
              ReLU-2                  [-1, 256]               0
            Linear-3                  [-1, 128]          32,896
              ReLU-4                  [-1, 128]               0
            Linear-5                   [-1, 20]           2,580
            Linear-6                   [-1, 20]           2,580
            Linear-7                  [-1, 128]           2,688
              ReLU-8                  [-1, 128]               0
            Linear-9                  [-1, 256]          33,024
             ReLU-10                  [-1, 256]               0
           Linear-11                  [-1, 784]         201,488
          Sigmoid-12                  [-1, 784]               0
================================================================
Total params: 476,216
Trainable params: 476,216
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 1.82
Estimated Total Size (MB): 1.84
----------------------------------------------------------------
==========VAE(Conv)网络结构==========
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]              60
              ReLU-2            [-1, 6, 28, 28]               0
            Conv2d-3           [-1, 16, 14, 14]             880
              ReLU-4           [-1, 16, 14, 14]               0
            Conv2d-5            [-1, 4, 14, 14]             580
              ReLU-6            [-1, 4, 14, 14]               0
            Linear-7                  [-1, 128]         100,480
              ReLU-8                  [-1, 128]               0
            Linear-9                   [-1, 20]           2,580
           Linear-10                   [-1, 20]           2,580
           Linear-11                  [-1, 128]           2,688
             ReLU-12                  [-1, 128]               0
           Linear-13                  [-1, 784]         101,136
             ReLU-14                  [-1, 784]               0
           Linear-15                  [-1, 784]         615,440
          Sigmoid-16                  [-1, 784]               0
================================================================
Total params: 826,424
Trainable params: 826,424
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.16
Params size (MB): 3.15
Estimated Total Size (MB): 3.32
----------------------------------------------------------------
```

设定模型参数

In [ ]:
```
# 训练模型参数
learning_rate = 1e-3  # 学习率
```

```
epoches = 20  # 迭代次数

# 实例化模型 二选一 即可
model = model_Conv
# model = model_Dense

# 创建优化器
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

模型训练

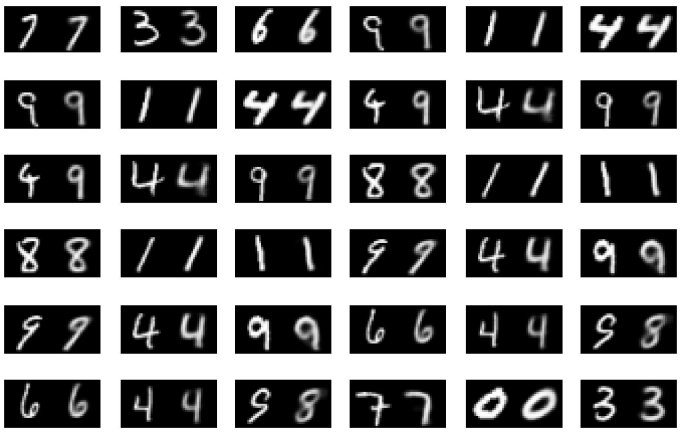
In [ ]:

```
train_loss = []  # 保存每个epoch的训练误差
number = len(dataIter.dataset)
result_dir = './VAEResult'  # 保存生成图片的目录
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练

        # 前向传播
        mean, log_var, gen_x = model(x)

        # 计算损失函数 二选一 即可
        loss = vae_loss_mse(x, gen_x, mean, log_var)
        # loss = vae_loss_cross(x, gen_x, mean, log_var)
        batch_loss.append(loss.item())

        # 反向传播和优化
        optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 梯度下降

        # 输出batch信息
        # if i % 100 == 0 and i > 0:
        #     print("epoch : {0} | batch : {1} | batch average loss: {2}"
        #           .format(epoch + 1, i, loss.item() / x.shape[0]))

        # 保存 各个epoch下 VAE的生成效果图
        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
                                   dim=3)  #torch.Size([128, 1, 28, 56])
        #     save_image(x_concatD, './%s/reconstructed-%d.png' % (result_dir, epoch

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))
```

```
epoch[1/20] | loss:52.37612461751302
epoch[2/20] | loss:38.15426111653646
epoch[3/20] | loss:33.8532580485026
epoch[4/20] | loss:32.30478760986328
epoch[5/20] | loss:31.240277282714842
epoch[6/20] | loss:30.621502152506512
epoch[7/20] | loss:30.20562481689453
epoch[8/20] | loss:29.88453271077474
epoch[9/20] | loss:29.626651607259113
epoch[10/20] | loss:29.438368599446616
epoch[11/20] | loss:29.267931953938803
epoch[12/20] | loss:29.087553869628906
epoch[13/20] | loss:28.97938673502604
epoch[14/20] | loss:28.863276033528646
epoch[15/20] | loss:28.741947403971356
epoch[16/20] | loss:28.638105383300783
epoch[17/20] | loss:28.515713920084636
epoch[18/20] | loss:28.441164237467447
epoch[19/20] | loss:28.351834224446616
epoch[20/20] | loss:28.28068007405599
```

展示最终迭代训练后：VAE生成图片与原始图片的对比

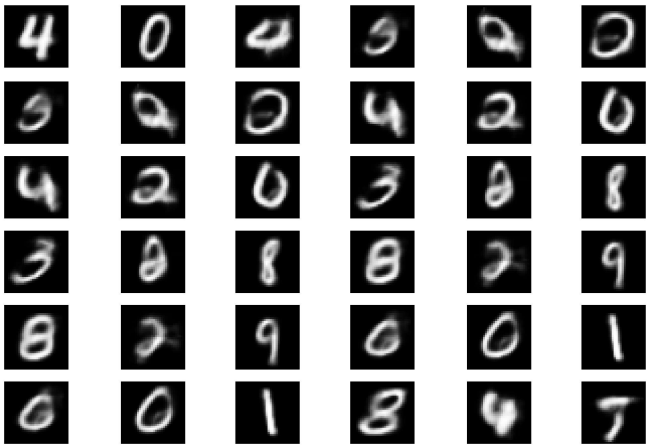
In [ ]:
```
# 展示最终迭代训练后：VAE生成图片与原始图片的对比
x_s = x_concatD.view(128, 28, 56).detach().cpu().numpy() * 255.
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_s[i * 3 + j], cmap='gray')
plt.show()
```



evaluation 测试生成效果

In [ ]:
```
# 从正态分布随机采样z
z = torch.randn((batch_size, 20)).to(device)
logits = model.decode(z)  # 仅通过解码器生成图片
x_hat = torch.sigmoid(logits)  # 转换为像素范围
x_hat = x_hat.view(128, 28, 28).detach().cpu().numpy() * 255.
# 展示图片
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
```

```
        axes[i][j].imshow(x_hat[i * 3 + j], cmap='gray')
plt.show()
```



中级要求

实现VAE的变分推导

有一批数据样本 $\{x_1, x_2, \cdots x_n\}$，x 的分布 $p(x)$ 未知

z：隐变量　　q：隐变量 z 的近似分布

散度定义：　$KL(a\|b) = \sum_x a(x) \log \frac{a(x)}{b(x)}$，KL越大说明 a, b 两分布差异越大，且 $KL \geq 0$

我们的目的是最大化 $p(x)$，采用最大似然的方法，即最大化 $\sum_x \log p(x)$.

[标红处为用到的公式或定理]

解：已知　$\int_z q(z|x)\, dz = 1$，

两边同乘 $\log p(x)$ 得：　$\log p(x) = \int_z q(z|x) \cdot \log p(x)\, dz$

将 $p(x) = \frac{p(x,z)}{p(z|x)} = \frac{p(x,z)}{q(z|x)} \cdot \frac{q(z|x)}{p(z|x)}$ 代入等式右边：[$p(x,y) = p(x|y) \cdot p(y)$]

$$\log p(x) = \int_z q(z|x) \cdot \log \left[\frac{p(x,z)}{q(z|x)} \cdot \frac{q(z|x)}{p(z|x)} \right] dz$$

$$= \int_z q(z|x) \cdot \log \frac{p(x,z)}{q(z|x)}\, dz + \int_z q(z|x) \cdot \log \frac{q(z|x)}{p(z|x)}\, dz$$

可以看出第二项为 $q(z|x)$ 和 $p(z|x)$ 的散度，即：

$$\log p(x) = \int_z q(z|x) \cdot \log \frac{p(x,z)}{q(z|x)}\, dz + KL(q(z|x)\|p(z|x)) \quad \cdots\cdots ①$$

接下来对①式分析。①式共两项，我们的目的是让①式整体尽可能大，但第二项，即 q 和 p 之间的散度应尽可能小，即 $q(z|x)$ 和 $p(z|x)$ 两分布应尽可能接近。在 $p(x)$ 值不变的情况下，第一项和第二项之间的大小关系呈反相关，因此我们下边的工作是让第一项尽可能大。

记第一项为 L，代入 [$p(x,z) = p(x|z) \cdot p(z)$]

$$L = \int_z q(z|x) \cdot \log \frac{p(z)}{q(z|x)}\, dz + \int_z q(z|x) \cdot \log p(x|z)\, dz \quad \cdots\cdots ②$$

分析可得第一项为 $q(z|x)$ 和 $p(z)$ 散度的相反数，第二项可看作是 $\log p(x|z)$ 在 $q(z|x)$ 分布上的期望

[对函数 $Y = g(x)$，在分布 $f(x)$ 上，有 $E(g(x)) = \int_x f(x) \cdot g(x)\, dx$]

即　$ELBO = L = E_{q(z|x)} \log p(x|z) - KL(q(z|x)\|p(z))$

第一项称为重构损失项，是要最大化的对数似然函数；第二项则是要最小化 $q(z|x)$ 和 $p(z)$ 的距离

$$Loss = -ELBO = KL(q(z|x)\|p(z)) - E_{q(z|x)} \log p(x|z)$$

下面对损失函数进行化简：

第一项：已知 $q(z|x)\sim N(\mu,\sigma^2)$　　$p(z)\sim N(0,1)$，

在一元正态分布的情况下，

$$KL(q(z|x)\|p(z))=\int_z q(z|x)\cdot\log\frac{q(z|x)}{p(z)}dz$$

$$=\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\log\frac{\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}}dx$$

$$=\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\log\left\{\frac{1}{\sigma}\cdot e^{\left[\frac{x^2}{2}-\frac{(x-\mu)^2}{2\sigma^2}\right]}\right\}dx$$

$$=\frac{1}{2}\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\left(x^2-\frac{(x-\mu)^2}{\sigma^2}-\log\sigma^2\right)dx$$

拆开算三项：

(1) $\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot x^2\,dx=E(x^2)=D(x)+E(x)^2=\sigma^2+\mu^2$

[对函数 $Y=g(x)$，在分布 $f(x)$ 上，有 $E(g(x))=\int_x f(x)\cdot g(x)\,dx$

(2) $\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\frac{(x-\mu)^2}{\sigma^2}dx=E\left[\frac{(x-\mu)^2}{\sigma^2}\right]$

$$=\frac{1}{\sigma^2}[E(x^2)-2\mu E(x)+\mu^2]$$

$$=\frac{1}{\sigma^2}[\sigma^2+\mu^2-2\mu^2+\mu^2]=1$$

(3) $\int_z\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\cdot\log\sigma^2dx=\log\sigma^2\cdot\int_z\frac{1}{\sqrt{2\pi}\sigma}\cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx=\log\sigma^2$

∴第一项为 $\frac{1}{2}(\sigma^2+\mu^2-1-\log\sigma^2)$

当是多元正态分布时（d为维度）为，

$$\frac{1}{2}\sum_{i=1}^{d}(\sigma^2+\mu^2-1-\log\sigma^2)$$

第二项：当样本数为 n 时，　$[E(x)=\frac{1}{n}\sum_{i=1}^{n}x_i]$

$$E_{q(z,x)}\log p(x|z)=\frac{1}{n}\sum_{i=1}^{n}\log p(x_i|z_i)$$

综上：$Loss=\frac{1}{2}\sum_{i=1}^{d}(\sigma^2+\mu^2-1-\log\sigma^2)-\frac{1}{n}\sum_{i=1}^{n}\log p(x_i|z_i)$

VAE的由来以及优缺点

自编码器（Auto-Encoder，AE）是一种无监督的神经网络，一般由编码器和解码器两部分构成。其过程为：编码器处理输入的数据进行编码，解码器根据编码内容来解码，从而还原出原始输入数据。简单来说，就是输入即为输出。这样的过程当然存在一定的问题：

- 容易受噪声数据干扰
- 对于没见过的输入，会给出意料之外的输出
- 以原输入再生成出原输入，在实际操作中，并无太大用处。

为了克服上述缺点，人们提出了一些AE模型的变种方案，VAE就是一种AE的变种。

变分自编码器（Variational AutoEncoder，VAE），是一个根据输入的数据的分布，类型，来模拟生成类似于输入数据的数据生成模型。VAE结合贝叶斯理论，不再认为整个样本空间只服从一个可观测的概率分布，而认为存在一个隐变量 `z`，控制着每个样本的概率分布。在隐变量 `z` 的作用下，VAE的输出不再与输入相同，而是成为了输入的一个近似，存在更多的不确定性。这样的结果是一把双刃剑：

- 优点：VAE采用分布的形式使得输出更加多样化，且不同的输出结果之间连接比较平滑。可以处理各种类型的数据，序列的和非序列的，连续的或离散的，甚至有标签的或无标签的，这使得它成为非常强大的生成工具。
- 缺点：生成的输出比较模糊，甚至可能偏离输入内容较远。不适用于对结果精确度有较高要求的情景。

高级要求

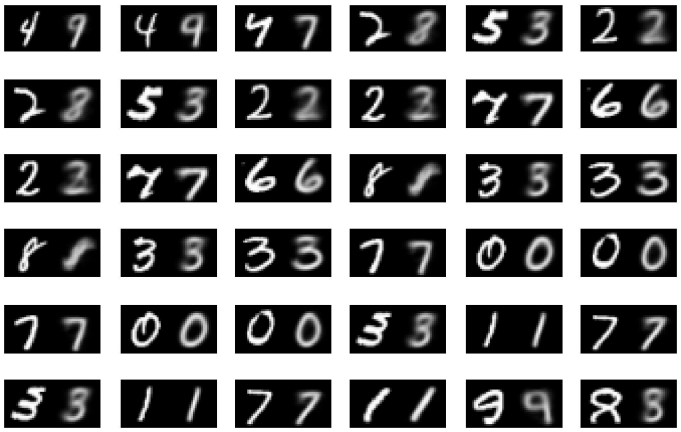
隐变量输出展示

为了便于得知隐变量的学习内容，将z_dim改成2维，进行训练学习，观察隐变量采样

In [ ]:
```
import os
from scipy.stats import norm

# 生成隐空间内的采样
# 训练时Z_dim设定为20，这里仅观察隐空间前俩维度表示特征
model = VariationalAutoEncoder_Conv(z_dim=2)
model.to(device)
epoches = 20
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_loss = []  # 保存每个epoch的训练误差
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练

        # 前向传播
        mean, log_var, gen_x = model(x)

        # 计算损失函数 二选一 即可
        loss = vae_loss_mse(x, gen_x, mean, log_var)
        batch_loss.append(loss.item())

        # 反向传播和优化
        optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 梯度下降

        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
                                   dim=3)  #torch.Size([128, 1, 28, 56])

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))
```

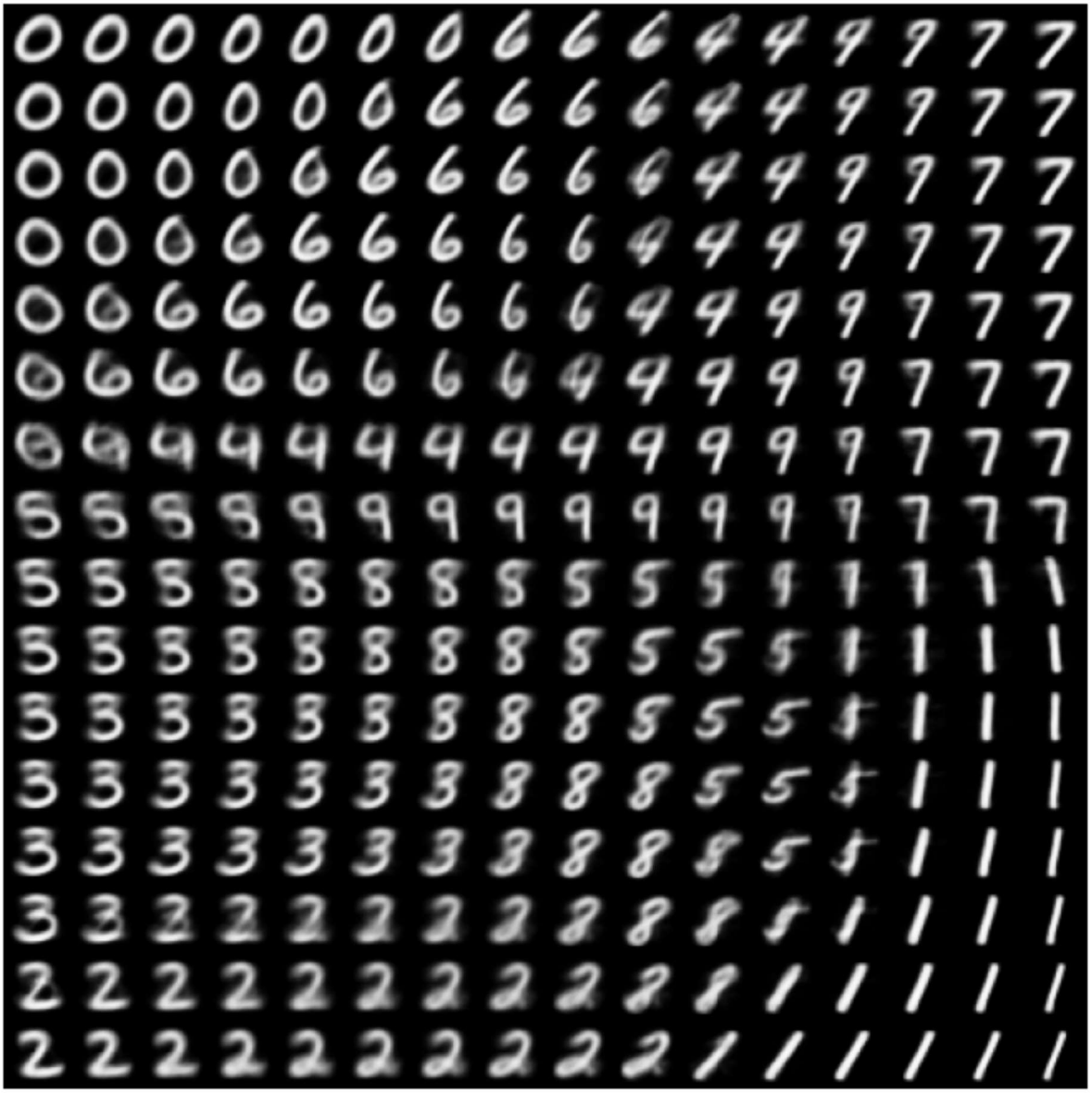
```
# 展示最终迭代训练后：VAE生成图片与原始图片的对比
x_s = x_concatD.view(128, 28, 56).detach().cpu().numpy() * 255.
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_s[i * 3 + j], cmap='gray')
plt.show()
```

```
epoch[1/20] | loss:54.73777405598958
epoch[2/20] | loss:48.60477434082031
epoch[3/20] | loss:41.396358451334635
epoch[4/20] | loss:39.77705034586589
epoch[5/20] | loss:38.7182927734375
epoch[6/20] | loss:38.03185735270182
epoch[7/20] | loss:37.55494031575521
epoch[8/20] | loss:37.23315347086589
epoch[9/20] | loss:36.958701883951825
epoch[10/20] | loss:36.75901299641927
epoch[11/20] | loss:36.58999401041667
epoch[12/20] | loss:36.450571240234375
epoch[13/20] | loss:36.328932267252604
epoch[14/20] | loss:36.19714766438802
epoch[15/20] | loss:36.11057159423828
epoch[16/20] | loss:35.986125130208336
epoch[17/20] | loss:35.932440087890626
epoch[18/20] | loss:35.844634948730466
epoch[19/20] | loss:35.799882779947914
epoch[20/20] | loss:35.727352176920576
```



In [ ]:

```
num_per_row, num_per_col = 16, 16
digit_size = 28
figure = np.zeros((digit_size * num_per_row, digit_size * num_per_col))
grid_x = norm.ppf(np.linspace(0.05, 0.95, num_per_row))
grid_y = norm.ppf(np.linspace(0.05, 0.95, num_per_col))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = torch.tensor(np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = model.decode(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        digit = torch.sigmoid(digit)
        figure[i * digit_size: (i + 1) * digit_size,
        j * digit_size: (j + 1) * digit_size] = digit.detach().cpu().numpy() * 255.

plt.figure(figsize=(12, 12))
plt.axis("off")
plt.imshow(figure, cmap='Greys_r')
plt.show()
```



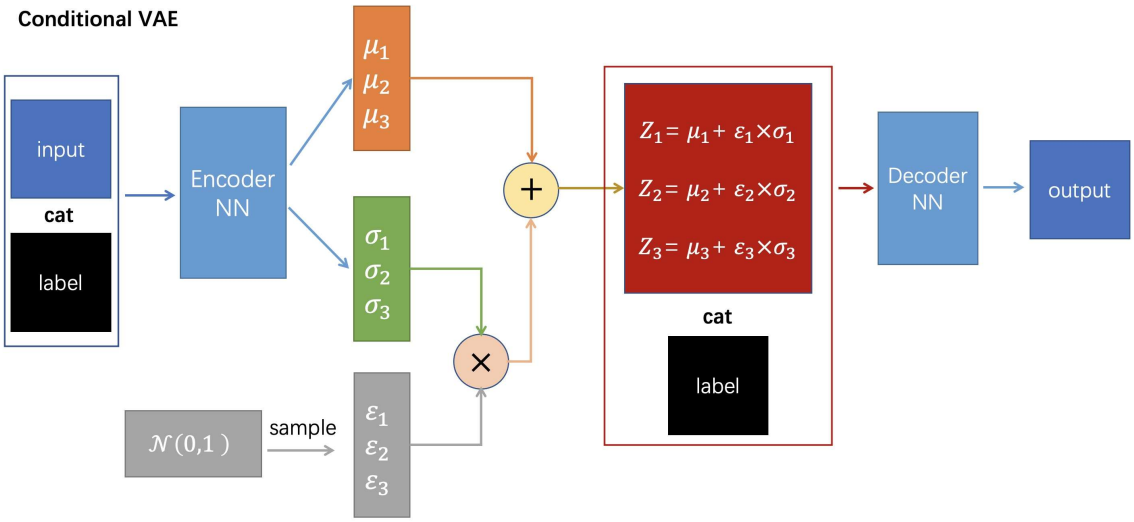
从上图中可以看出，VAE学到的二维隐变量Z:

- 第一维度（水平维度）主要控制 `数字的形状` ，字体的斜度改变幅度不大
- 第二维度（竖直维度）主要控制数字的 `倾斜` 和 `形状（低于第一维度的影响）`

CVAE

先介绍AE和VAE的优缺点：

- AE适合数据压缩与还原，不适合生成未见过的数据
- VAE适合生成未见过的数据，但不能控制生成内容。 ### CVAE（Conditional VAE）可以在生成数据时通过指定其标签来生成想生成的数据



In [ ]:
```python
# 条件变分自编码器
class CVAE(nn.Module):

    def __init__(self, feature_size, class_size, latent_size):
        super(CVAE, self).__init__()

        self.fc1 = nn.Linear(feature_size + class_size, 200)
        self.fc2_mu = nn.Linear(200, latent_size)
        self.fc2_log_std = nn.Linear(200, latent_size)
        self.fc3 = nn.Linear(latent_size + class_size, 200)
        self.fc4 = nn.Linear(200, feature_size)

    def encode(self, x, y):
        h1 = F.relu(self.fc1(torch.cat([x, y], dim=1)))  # 连接特征和标签
        mu = self.fc2_mu(h1)
        log_std = self.fc2_log_std(h1)
        return mu, log_std

    def decode(self, z, y):
        h3 = F.relu(self.fc3(torch.cat([z, y], dim=1)))  # 连接标签和latent隐变量
        recon = torch.sigmoid(self.fc4(h3))
        return recon

    def reparametrize(self, mu, log_std):
        std = torch.exp(0.5 * log_std)
        eps = torch.randn_like(std)
        z = mu + eps * std
        return z

    def forward(self, x, y):
        mu, log_std = self.encode(x, y)
        z = self.reparametrize(mu, log_std)
        recon = self.decode(z, y)
        return recon, mu, log_std

    def loss_function(self, recon, x, mu, log_std) -> torch.Tensor:
        recon_loss = F.mse_loss(recon, x, reduction="sum")
        kl_loss = -0.5 * (1 + 2 * log_std - mu.pow(2) - torch.exp(2 * log_std))
        kl_loss = torch.sum(kl_loss)
        loss = recon_loss + kl_loss
        return loss
```

In [ ]:
```python
cvae = CVAE(feature_size=784, class_size=10, latent_size=20)
optimizer = torch.optim.Adam(cvae.parameters(), lr=1e-3)
result_dir = '.\\VAEResult\\CVAE'
```

```
train_loss = []
epoches = 40
for epoch in range(epoches):

    batch_loss = []
    for i, data in enumerate(dataIter):
        img, label = data
        img.to(device)
        inputs = img.view(img.shape[0], -1)
        y = torch.nn.functional.one_hot(label, num_classes=10)
        # print(inputs.shape)
        # print(y.shape)
        y.to(device)

        recon, mu, log_std = cvae(inputs, y)
        loss = cvae.loss_function(recon, inputs, mu, log_std)
        batch_loss.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # 输出batch信息
        if i % 100 == 0 and i > 0:
            print("epoch : {0} | batch : {1} | batch average loss: {2}"
                  .format(epoch + 1, i, loss.item() / img.shape[0]))
        if i == 0:
            x_concatD = torch.cat([img.view(-1, 1, 28, 28), recon.view(-1, 1, 28, 28
            save_image(x_concatD, '%s\\reconstructed-%d.png' % (result_dir, epoch +

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))
```

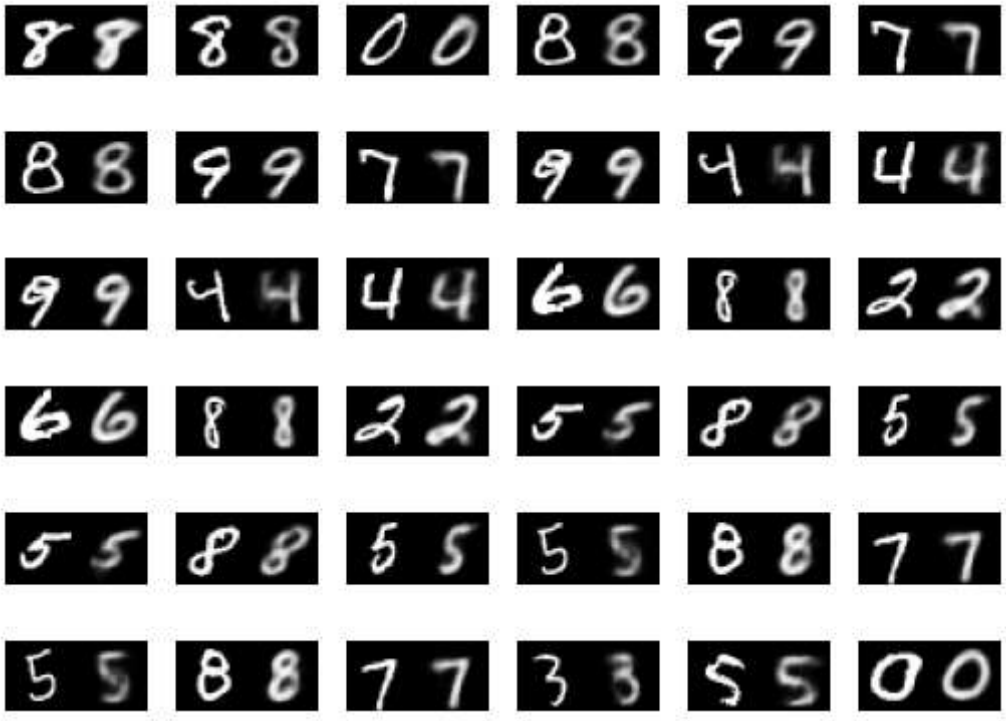
```
epoch : 1 | batch : 100 | batch average loss: 51.963008880615234
epoch : 1 | batch : 200 | batch average loss: 45.13134002685547
epoch : 1 | batch : 300 | batch average loss: 40.94240188598633
epoch : 1 | batch : 400 | batch average loss: 38.570213317871094
epoch[1/40] | loss:48.95314457194011
epoch : 2 | batch : 100 | batch average loss: 38.26136016845703
epoch : 2 | batch : 200 | batch average loss: 37.370811462402344
epoch : 2 | batch : 300 | batch average loss: 35.45708084106445
epoch : 2 | batch : 400 | batch average loss: 35.08346176147461
epoch[2/40] | loss:36.59286827799479
epoch : 3 | batch : 100 | batch average loss: 36.25738525390625
epoch : 3 | batch : 200 | batch average loss: 33.35258865356445
epoch : 3 | batch : 300 | batch average loss: 32.806697845458984
epoch : 3 | batch : 400 | batch average loss: 33.62622833251953
epoch[3/40] | loss:34.4992719523112
epoch : 4 | batch : 100 | batch average loss: 34.40482711791992
epoch : 4 | batch : 200 | batch average loss: 34.585147857666016
epoch : 4 | batch : 300 | batch average loss: 32.33565139770508
epoch : 4 | batch : 400 | batch average loss: 31.76677131652832
epoch[4/40] | loss:33.41451225992839
epoch : 5 | batch : 100 | batch average loss: 33.434326171875
epoch : 5 | batch : 200 | batch average loss: 34.187599182128906
epoch : 5 | batch : 300 | batch average loss: 32.80463790893555
epoch : 5 | batch : 400 | batch average loss: 31.56391143798828
epoch[5/40] | loss:32.67819062093099
epoch : 6 | batch : 100 | batch average loss: 33.56522750854492
epoch : 6 | batch : 200 | batch average loss: 31.77747917175293
epoch : 6 | batch : 300 | batch average loss: 31.043067932128906
epoch : 6 | batch : 400 | batch average loss: 33.22935485839844
epoch[6/40] | loss:32.212308577473955
epoch : 7 | batch : 100 | batch average loss: 33.25436019897461
epoch : 7 | batch : 200 | batch average loss: 30.396987915039062
epoch : 7 | batch : 300 | batch average loss: 29.83347511291504
epoch : 7 | batch : 400 | batch average loss: 30.063520431518555
epoch[7/40] | loss:31.85576857096354
epoch : 8 | batch : 100 | batch average loss: 31.95095443725586
epoch : 8 | batch : 200 | batch average loss: 30.38056182861328
epoch : 8 | batch : 300 | batch average loss: 33.28582763671875
epoch : 8 | batch : 400 | batch average loss: 30.283403396606445
epoch[8/40] | loss:31.6026166015625
epoch : 9 | batch : 100 | batch average loss: 29.716054916381836
epoch : 9 | batch : 200 | batch average loss: 31.252925872802734
epoch : 9 | batch : 300 | batch average loss: 30.440357208251953
epoch : 9 | batch : 400 | batch average loss: 34.86772537231445
epoch[9/40] | loss:31.36993547363281
epoch : 10 | batch : 100 | batch average loss: 33.55823516845703
epoch : 10 | batch : 200 | batch average loss: 29.30370330810547
epoch : 10 | batch : 300 | batch average loss: 31.13372230529785
epoch : 10 | batch : 400 | batch average loss: 30.678951263427734
epoch[10/40] | loss:31.19885840657552
epoch : 11 | batch : 100 | batch average loss: 31.896329879760742
epoch : 11 | batch : 200 | batch average loss: 31.10787010192871
epoch : 11 | batch : 300 | batch average loss: 30.818161010742188
epoch : 11 | batch : 400 | batch average loss: 30.791887283325195
epoch[11/40] | loss:31.03942110188802
epoch : 12 | batch : 100 | batch average loss: 32.36382293701172
epoch : 12 | batch : 200 | batch average loss: 31.722469329833984
epoch : 12 | batch : 300 | batch average loss: 30.01117515563965
epoch : 12 | batch : 400 | batch average loss: 30.601226806640625
epoch[12/40] | loss:30.90047344156901
epoch : 13 | batch : 100 | batch average loss: 29.797889709472656
epoch : 13 | batch : 200 | batch average loss: 30.809833526611328
epoch : 13 | batch : 300 | batch average loss: 31.6361083984375
epoch : 13 | batch : 400 | batch average loss: 30.772850036621094
```

```
epoch[13/40] | loss:30.824939388020834
epoch : 14 | batch : 100 | batch average loss: 29.731353759765625
epoch : 14 | batch : 200 | batch average loss: 31.06955337524414
epoch : 14 | batch : 300 | batch average loss: 31.559471130371094
epoch : 14 | batch : 400 | batch average loss: 30.52155303955078
epoch[14/40] | loss:30.73059589029948
epoch : 15 | batch : 100 | batch average loss: 30.45916748046875
epoch : 15 | batch : 200 | batch average loss: 30.16378402709961
epoch : 15 | batch : 300 | batch average loss: 31.088815689086914
epoch : 15 | batch : 400 | batch average loss: 32.014007568359375
epoch[15/40] | loss:30.675125561523437
epoch : 16 | batch : 100 | batch average loss: 31.838071823120117
epoch : 16 | batch : 200 | batch average loss: 30.342453002929688
epoch : 16 | batch : 300 | batch average loss: 30.87862777709961
epoch : 16 | batch : 400 | batch average loss: 29.179012298583984
epoch[16/40] | loss:30.59822233072917
epoch : 17 | batch : 100 | batch average loss: 29.511093139648438
epoch : 17 | batch : 200 | batch average loss: 30.699844360351562
epoch : 17 | batch : 300 | batch average loss: 32.082977294921875
epoch : 17 | batch : 400 | batch average loss: 30.24424934387207
epoch[17/40] | loss:30.530924995930988
epoch : 18 | batch : 100 | batch average loss: 30.295574188232422
epoch : 18 | batch : 200 | batch average loss: 30.765518188476562
epoch : 18 | batch : 300 | batch average loss: 30.426048278808594
epoch : 18 | batch : 400 | batch average loss: 30.835126876831055
epoch[18/40] | loss:30.472196333821614
epoch : 19 | batch : 100 | batch average loss: 30.74138832092285
epoch : 19 | batch : 200 | batch average loss: 30.207359313964844
epoch : 19 | batch : 300 | batch average loss: 30.918550491333008
epoch : 19 | batch : 400 | batch average loss: 28.2825870513916
epoch[19/40] | loss:30.369287532552082
epoch : 20 | batch : 100 | batch average loss: 30.907062530517578
epoch : 20 | batch : 200 | batch average loss: 29.95197296142578
epoch : 20 | batch : 300 | batch average loss: 30.51449966430664
epoch : 20 | batch : 400 | batch average loss: 30.24428939819336
epoch[20/40] | loss:30.36614881591797
epoch : 21 | batch : 100 | batch average loss: 29.085006713867188
epoch : 21 | batch : 200 | batch average loss: 28.42890739440918
epoch : 21 | batch : 300 | batch average loss: 28.805042266845703
epoch : 21 | batch : 400 | batch average loss: 28.745967864990234
epoch[21/40] | loss:30.30871149088542
epoch : 22 | batch : 100 | batch average loss: 31.5228214263916
epoch : 22 | batch : 200 | batch average loss: 30.416906356811523
epoch : 22 | batch : 300 | batch average loss: 29.9090576171875
epoch : 22 | batch : 400 | batch average loss: 30.389057159423828
epoch[22/40] | loss:30.278405017089845
epoch : 23 | batch : 100 | batch average loss: 29.83504867553711
epoch : 23 | batch : 200 | batch average loss: 28.420379638671875
epoch : 23 | batch : 300 | batch average loss: 29.209976196289062
epoch : 23 | batch : 400 | batch average loss: 32.416481018066406
epoch[23/40] | loss:30.226935026041666
epoch : 24 | batch : 100 | batch average loss: 29.19626235961914
epoch : 24 | batch : 200 | batch average loss: 29.717859268188477
epoch : 24 | batch : 300 | batch average loss: 29.214136123657227
epoch : 24 | batch : 400 | batch average loss: 30.004039764404297
epoch[24/40] | loss:30.199798457845052
epoch : 25 | batch : 100 | batch average loss: 31.328393936157227
epoch : 25 | batch : 200 | batch average loss: 30.36971664428711
epoch : 25 | batch : 300 | batch average loss: 30.22542381286621
epoch : 25 | batch : 400 | batch average loss: 30.190940856933594
epoch[25/40] | loss:30.12611864827474
epoch : 26 | batch : 100 | batch average loss: 28.886539459228516
epoch : 26 | batch : 200 | batch average loss: 28.849742889404297
epoch : 26 | batch : 300 | batch average loss: 30.608139038085938
```

```
epoch : 26 | batch : 400 | batch average loss: 30.700090408325195
epoch[26/40] | loss:30.143408459472656
epoch : 27 | batch : 100 | batch average loss: 30.710426330566406
epoch : 27 | batch : 200 | batch average loss: 29.411388397216797
epoch : 27 | batch : 300 | batch average loss: 28.671241760253906
epoch : 27 | batch : 400 | batch average loss: 32.32023239135742
epoch[27/40] | loss:30.078115478515624
epoch : 28 | batch : 100 | batch average loss: 29.604829788208008
epoch : 28 | batch : 200 | batch average loss: 30.363327026367188
epoch : 28 | batch : 300 | batch average loss: 30.298006057739258
epoch : 28 | batch : 400 | batch average loss: 28.877004623413086
epoch[28/40] | loss:30.047482482910155
epoch : 29 | batch : 100 | batch average loss: 29.092262268066406
epoch : 29 | batch : 200 | batch average loss: 30.497058868408203
epoch : 29 | batch : 300 | batch average loss: 29.675079345703125
epoch : 29 | batch : 400 | batch average loss: 29.023897171020508
epoch[29/40] | loss:30.00537834065755
epoch : 30 | batch : 100 | batch average loss: 30.67474937438965
epoch : 30 | batch : 200 | batch average loss: 30.208881378173828
epoch : 30 | batch : 300 | batch average loss: 28.00241470336914
epoch : 30 | batch : 400 | batch average loss: 28.121036529541016
epoch[30/40] | loss:30.008169958496094
epoch : 31 | batch : 100 | batch average loss: 30.705406188964844
epoch : 31 | batch : 200 | batch average loss: 30.130264282226562
epoch : 31 | batch : 300 | batch average loss: 30.544179916381836
epoch : 31 | batch : 400 | batch average loss: 29.273479461669922
epoch[31/40] | loss:29.990759016927083
epoch : 32 | batch : 100 | batch average loss: 31.866615295410156
epoch : 32 | batch : 200 | batch average loss: 28.473804473876953
epoch : 32 | batch : 300 | batch average loss: 30.587295532226562
epoch : 32 | batch : 400 | batch average loss: 31.260520935058594
epoch[32/40] | loss:29.96353927001953
epoch : 33 | batch : 100 | batch average loss: 28.99738883972168
epoch : 33 | batch : 200 | batch average loss: 28.271421432495117
epoch : 33 | batch : 300 | batch average loss: 28.6134033203125
epoch : 33 | batch : 400 | batch average loss: 28.58405303955078
epoch[33/40] | loss:29.90408229166667
epoch : 34 | batch : 100 | batch average loss: 29.95927619934082
epoch : 34 | batch : 200 | batch average loss: 29.22492790222168
epoch : 34 | batch : 300 | batch average loss: 28.661806106567383
epoch : 34 | batch : 400 | batch average loss: 28.675168991088867
epoch[34/40] | loss:29.940753666178384
epoch : 35 | batch : 100 | batch average loss: 28.52299690246582
epoch : 35 | batch : 200 | batch average loss: 31.263370513916016
epoch : 35 | batch : 300 | batch average loss: 30.374156951904297
epoch : 35 | batch : 400 | batch average loss: 29.15502166748047
epoch[35/40] | loss:29.90136706542969
epoch : 36 | batch : 100 | batch average loss: 29.628070831298828
epoch : 36 | batch : 200 | batch average loss: 29.055295944213867
epoch : 36 | batch : 300 | batch average loss: 30.4556941986084
epoch : 36 | batch : 400 | batch average loss: 31.521160125732422
epoch[36/40] | loss:29.854595438639322
epoch : 37 | batch : 100 | batch average loss: 29.48159408569336
epoch : 37 | batch : 200 | batch average loss: 28.464279174804688
epoch : 37 | batch : 300 | batch average loss: 28.795665740966797
epoch : 37 | batch : 400 | batch average loss: 30.136043548583984
epoch[37/40] | loss:29.827265291341146
epoch : 38 | batch : 100 | batch average loss: 29.85831069946289
epoch : 38 | batch : 200 | batch average loss: 27.91811752319336
epoch : 38 | batch : 300 | batch average loss: 30.69519805908203
epoch : 38 | batch : 400 | batch average loss: 29.865501403808594
epoch[38/40] | loss:29.818939029947916
epoch : 39 | batch : 100 | batch average loss: 29.2919921875
epoch : 39 | batch : 200 | batch average loss: 32.69929504394531
```

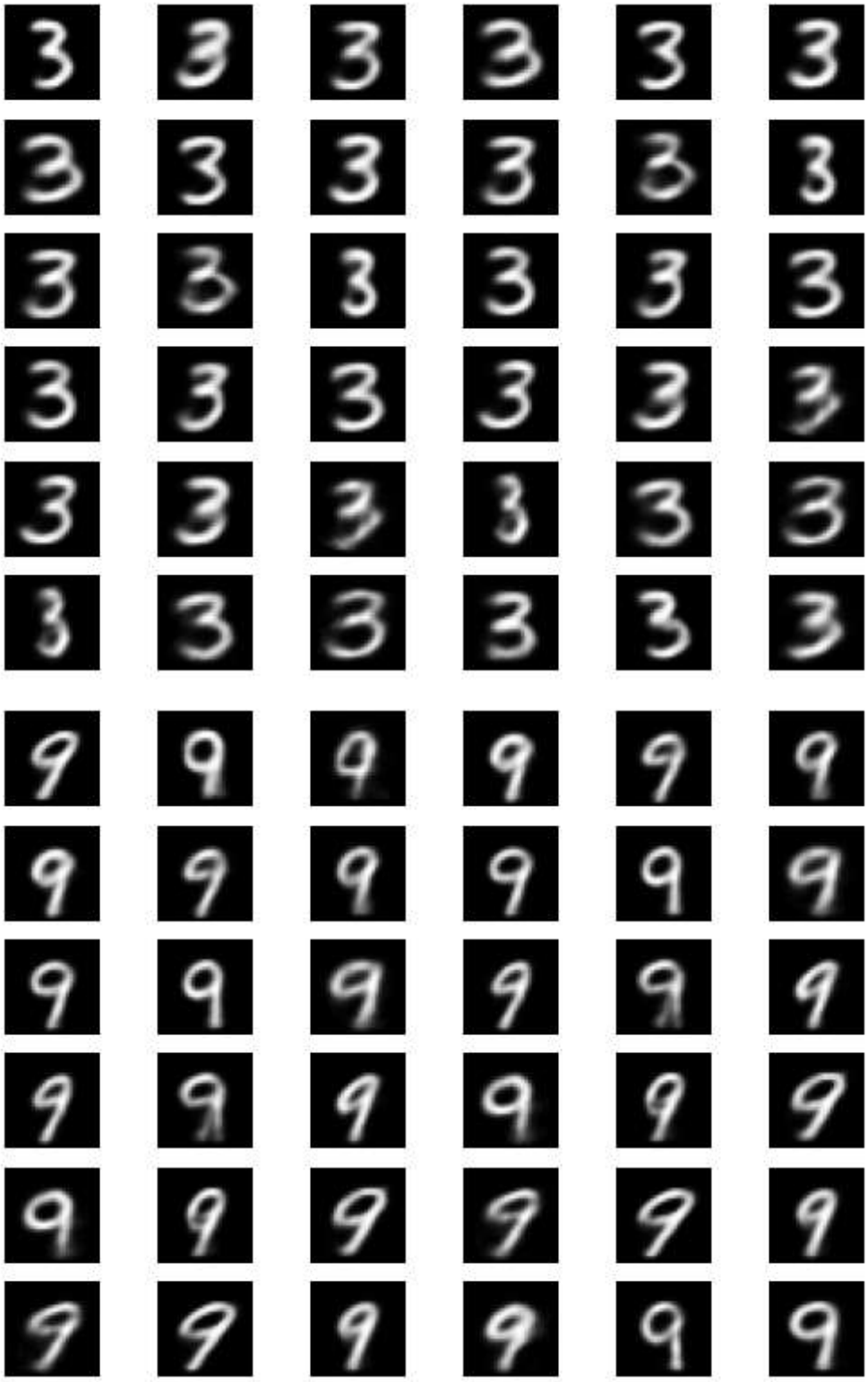
```
epoch : 39 | batch : 300 | batch average loss: 29.28822135925293
epoch : 39 | batch : 400 | batch average loss: 31.737045288085938
epoch[39/40] | loss:29.8075778523763
epoch : 40 | batch : 100 | batch average loss: 30.906143188476562
epoch : 40 | batch : 200 | batch average loss: 29.92847442626953
epoch : 40 | batch : 300 | batch average loss: 30.740585327148438
epoch : 40 | batch : 400 | batch average loss: 29.669231414794922
epoch[40/40] | loss:29.782788846842447
```

In [ ]:
```python
# 展示最终迭代训练后：VAE生成图片与原始图片的对比
x_s = x_concatD.view(128, 28, 56).detach().cpu().numpy() * 255.
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_s[i * 3 + j], cmap='gray')
plt.show()
```



In [ ]:
```python
# 利用CVAE生成特定label下的图片
label = 3
z = torch.randn(784, 20)
labels = torch.full(size=(784,), fill_value=label, dtype=torch.int64)
y = torch.nn.functional.one_hot(labels, num_classes=10)
recon = cvae.decode(z, y)

# 显示
img = recon.view(recon.shape[0],  28, 28).detach().cpu().numpy() * 255.
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(img[i * 3 + j], cmap='gray')
plt.show()

label = 9
z = torch.randn(784, 20)
labels = torch.full(size=(784,), fill_value=label, dtype=torch.int64)
y = torch.nn.functional.one_hot(labels, num_classes=10)
recon = cvae.decode(z, y)
```

```
# 显示
img = recon.view(recon.shape[0],  28, 28).detach().cpu().numpy() * 255.
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(img[i * 3 + j], cmap='gray')
plt.show()
```



β-VAE

β-VAE是在传统VAE基础上对损失函数进行修改，为损失函数的

KL散度项加上一个超参数β

VAE的损失函数由两部分组成，一部分是重构项，一部分是KL散度。VAE的整体过程可以粗略看成两步，第一步是编码，第二步是解码，两步由编码生成的隐变量连结。β的引入相当于为隐变量的信息增加了一个信息瓶颈。

重构项反映了输入与隐变量的互信息大小，KL部分反映了隐变量含有的信息量大小，β的引入约束了隐变量的传递的信息量，在信息保存（重建成本作为正则化）和潜在信道容量限制（β>1）之间寻找平衡，增强了模型的解耦能力。

损失函数引入超参数β

In [ ]:
```
# L2损失 (MSE Loss) 构建损失函数
def beta_vae_loss_mse(x, gen_x, mean, log_var, beta):
    # 重构项损失
    mse_loss = torch.nn.MSELoss(reduction='sum')
    loss1 = mse_loss(gen_x, x)

    # 最小化 q(z|x)  和 p(z) 的距离
    KL_loss = 0.5 * torch.sum(torch.exp(log_var) + torch.pow(mean, 2) - log_var - 1

    return loss1 + beta*KL_loss


# 交叉熵损失 (Cross Entropy Loss) 构建损失函数
def beta_vae_loss_cross(x, gen_x, mean, log_var, beta):
    # 重构项损失
    loss1 = F.binary_cross_entropy(gen_x, x, reduction='sum')

    # 最小化 q(z|x)  和 p(z) 的距离
    KL_loss = 0.5 * torch.sum(torch.exp(log_var) + torch.pow(mean, 2) - log_var - 1

    return loss1 + beta*KL_loss
```

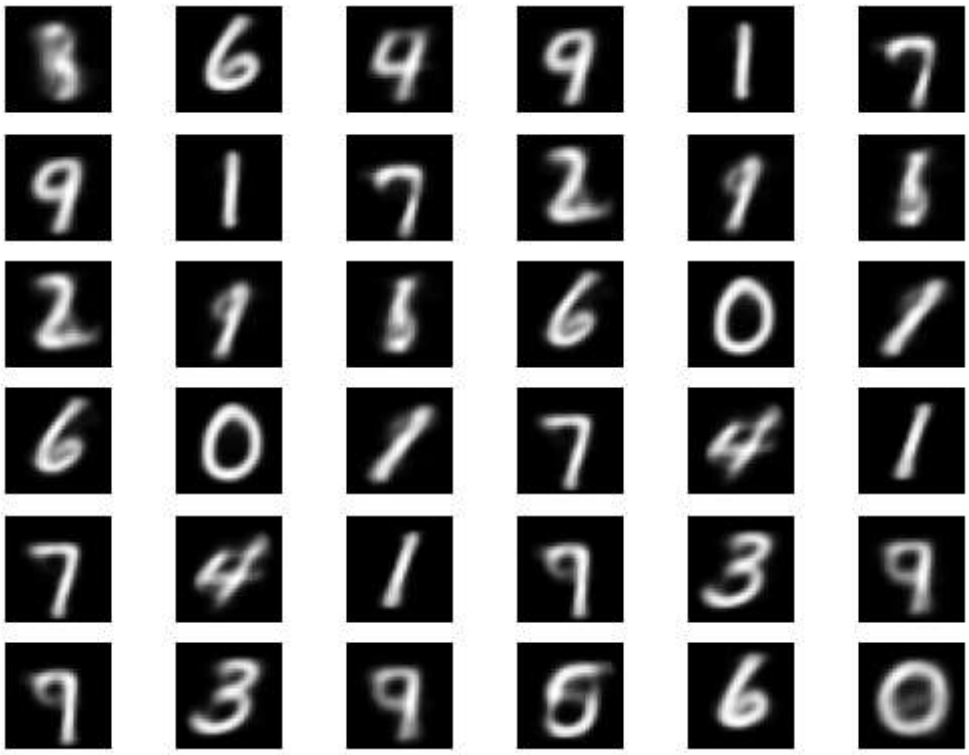
In [ ]:
```
test_z = torch.randn((batch_size, 20)).to(device)
```

经测试 β = 3 时有较好的生成效果

In [ ]:
```
# 训练模型参数
learning_rate = 1e-3  # 学习率
epoches = 10  # 迭代次数

# 实例化模型
beta_vae = VariationalAutoEncoder_Conv()
beta_vae.to(device)
beta_model = beta_vae

#超参数
beta = 3

# 创建优化器
beta_optimizer = torch.optim.Adam(beta_model.parameters(), lr=learning_rate)

train_loss = []  # 保存每个epoch的训练误差
result_dir = '.\\VAEResult\\bata-VAE'  # 保存生成图片的目录
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
```

```
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练


        mean, log_var, gen_x = beta_model(x)
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, beta)
        batch_loss.append(loss.item())

        # 反向传播和优化
        beta_optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        beta_optimizer.step()  # 梯度下降

        # 保存 各个epoch下 VAE的生成效果图
        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

logits = beta_model.decode(test_z)  # 仅通过解码器生成图片
x_hat = torch.sigmoid(logits)  # 转换为像素范围
x_hat = x_hat.view(128, 28, 28).detach().cpu().numpy() * 255.

# 展示图片
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_hat[i * 3 + j], cmap='gray')
plt.show()
```

```
epoch[1/10] | loss:53.801485278320314
epoch[2/10] | loss:48.48496276855469
epoch[3/10] | loss:46.122619482421875
epoch[4/10] | loss:44.91053746744792
epoch[5/10] | loss:44.41830204264323
epoch[6/10] | loss:44.05171790364583
epoch[7/10] | loss:43.800046158854165
epoch[8/10] | loss:43.617858894856774
epoch[9/10] | loss:43.45289364420573
epoch[10/10] | loss:43.37656237792969
```



为进一步探究验证超参数β的作用，测试不同β值下的生成结果

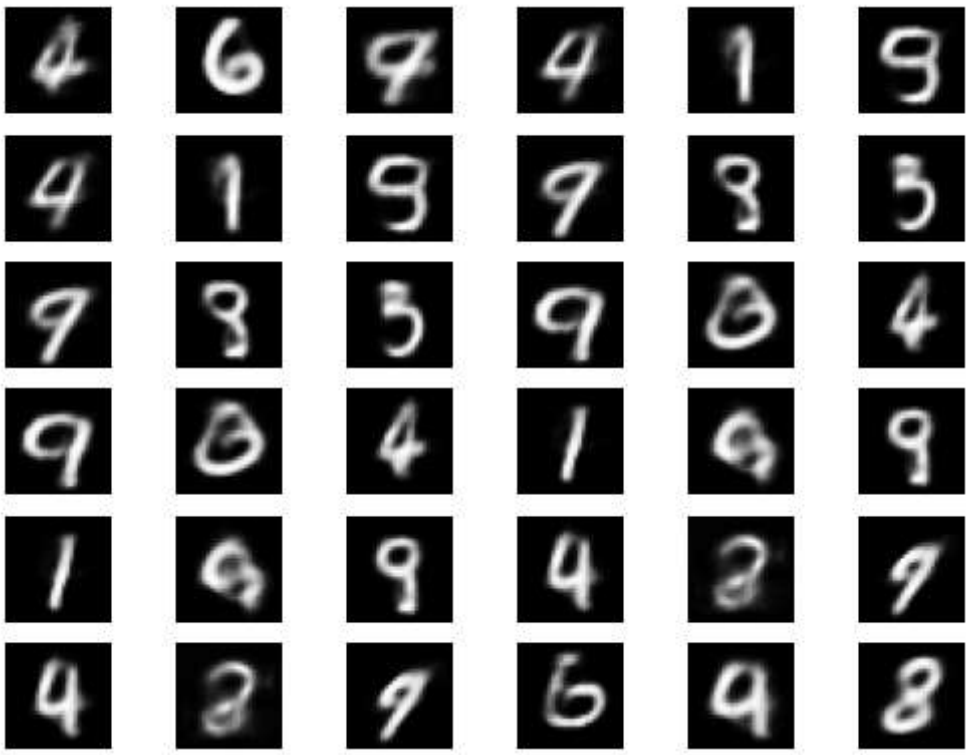
β = 1 (VAE)

In [ ]:

```
# 训练模型参数
learning_rate = 1e-3  # 学习率
epoches = 10  # 迭代次数

# 实例化模型
beta_vae1 = VariationalAutoEncoder_Conv()
beta_vae1.to(device)
beta_model = beta_vae1

#超参数
beta = 1

# 创建优化器
beta_optimizer = torch.optim.Adam(beta_model.parameters(), lr=learning_rate)

train_loss = []  # 保存每个epoch的训练误差
result_dir = '.\\VAEResult\\bata-VAE'  # 保存生成图片的目录
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练


        mean, log_var, gen_x = beta_model(x)
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, beta)
        batch_loss.append(loss.item())

        # 反向传播和优化
        beta_optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        beta_optimizer.step()  # 梯度下降
```

```
        # 保存 各个epoch下 VAE的生成效果图
        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))


logits = beta_model.decode(test_z)  # 仅通过解码器生成图片
x_hat = torch.sigmoid(logits)  # 转换为像素范围
x_hat = x_hat.view(128, 28, 28).detach().cpu().numpy() * 255.

# 展示图片
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_hat[i * 3 + j], cmap='gray')
plt.show()
```

```
epoch[1/10] | loss:49.908536385091146
epoch[2/10] | loss:36.818755301920575
epoch[3/10] | loss:33.74037689208984
epoch[4/10] | loss:32.548841548665365
epoch[5/10] | loss:31.865619213867188
epoch[6/10] | loss:31.306879956054686
epoch[7/10] | loss:30.82028231608073
epoch[8/10] | loss:30.462920483398438
epoch[9/10] | loss:30.15816045735677
epoch[10/10] | loss:29.915357299804686
```



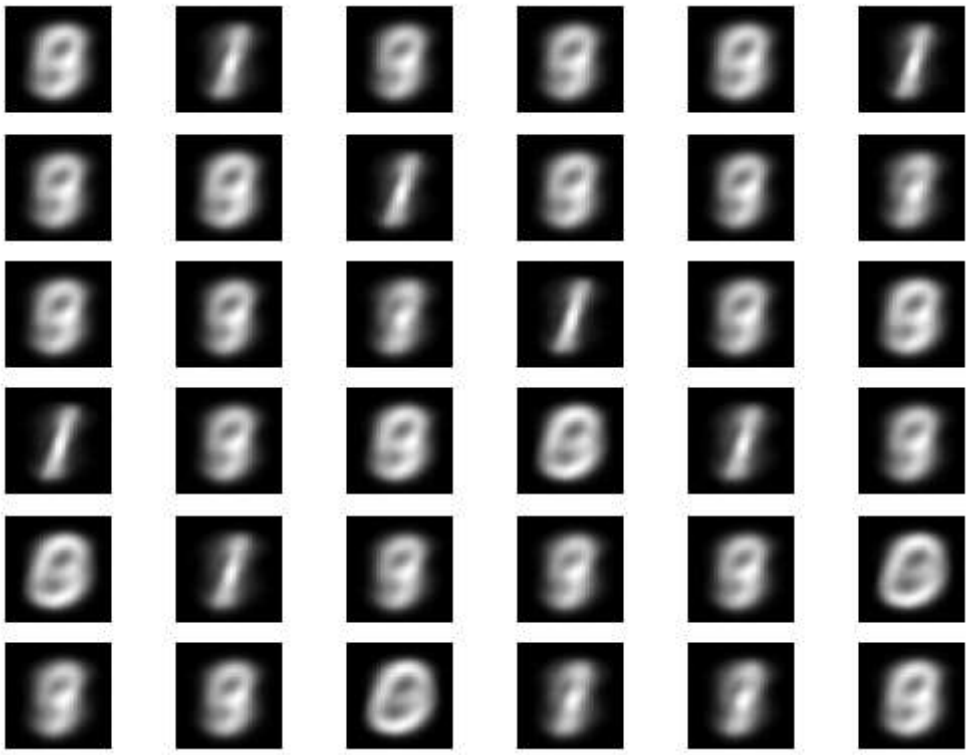
β = 8

In [ ]:

```
# 训练模型参数
learning_rate = 1e-3  # 学习率
epoches = 10  # 迭代次数
```

```
# 实例化模型
beta_vae2 = VariationalAutoEncoder_Conv()
beta_vae2.to(device)
beta_model = beta_vae2

#超参数
beta = 8

# 创建优化器
beta_optimizer = torch.optim.Adam(beta_model.parameters(), lr=learning_rate)

train_loss = []  # 保存每个epoch的训练误差
result_dir = '.\\VAEResult\\bata-VAE'  # 保存生成图片的目录
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练


        mean, log_var, gen_x = beta_model(x)
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, beta)
        batch_loss.append(loss.item())

        # 反向传播和优化
        beta_optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        beta_optimizer.step()  # 梯度下降

        # 保存 各个epoch下 VAE的生成效果图
        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

logits = beta_model.decode(test_z)  # 仅通过解码器生成图片
x_hat = torch.sigmoid(logits)  # 转换为像素范围
x_hat = x_hat.view(128, 28, 28).detach().cpu().numpy() * 255.

# 展示图片
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_hat[i * 3 + j], cmap='gray')
plt.show()
```

```
epoch[1/10] | loss:54.8489836344401
epoch[2/10] | loss:52.94638594563802
epoch[3/10] | loss:52.85733248697917
epoch[4/10] | loss:52.76279864095052
epoch[5/10] | loss:52.70004223632812
epoch[6/10] | loss:52.665559692382814
epoch[7/10] | loss:52.62776123046875
epoch[8/10] | loss:52.4991755859375
epoch[9/10] | loss:52.44217868652344
epoch[10/10] | loss:52.42447747395833
```

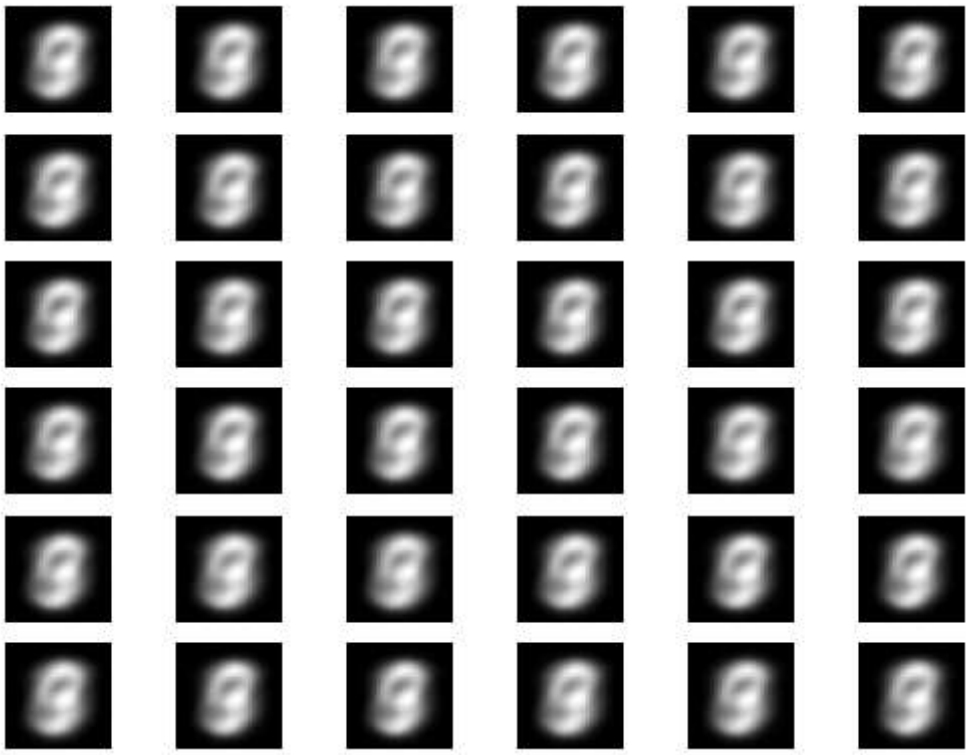


β = 16

In [ ]:
```
# 训练模型参数
learning_rate = 1e-3  # 学习率
epoches = 10  # 迭代次数

# 实例化模型
beta_vae3 = VariationalAutoEncoder_Conv()
beta_vae3.to(device)
beta_model = beta_vae3

#超参数
beta = 16

# 创建优化器
beta_optimizer = torch.optim.Adam(beta_model.parameters(), lr=learning_rate)

train_loss = []  # 保存每个epoch的训练误差
result_dir = '.\\VAEResult\\bata-VAE'  # 保存生成图片的目录
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练


        mean, log_var, gen_x = beta_model(x)
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, beta)
        batch_loss.append(loss.item())

        # 反向传播和优化
        beta_optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        beta_optimizer.step()  # 梯度下降

        # 保存 各个epoch下 VAE的生成效果图
        if i == 0:
```

```
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

logits = beta_model.decode(test_z)  # 仅通过解码器生成图片
x_hat = torch.sigmoid(logits)  # 转换为像素范围
x_hat = x_hat.view(128, 28, 28).detach().cpu().numpy() * 255.

# 展示图片
_, axes = plt.subplots(6, 6)
for i in range(6):
    for j in range(6):
        axes[i][j].axis('off')
        axes[i][j].imshow(x_hat[i * 3 + j], cmap='gray')
plt.show()
```

```
epoch[1/10] | loss:54.87018264160156
epoch[2/10] | loss:52.99105546875
epoch[3/10] | loss:52.91904027506511
epoch[4/10] | loss:52.876090885416666
epoch[5/10] | loss:52.853292586263024
epoch[6/10] | loss:52.82928025716146
epoch[7/10] | loss:52.80811359049479
epoch[8/10] | loss:52.7967451578776
epoch[9/10] | loss:52.78326774902344
epoch[10/10] | loss:52.77977409667969
```



经过测试发现，随着β的增大生成的手写数字特征逐渐单一，可以推断出是由于隐变量信息保留的过少导致的。

为进一步验证，对于不同β下学习得到的隐变量进行观察

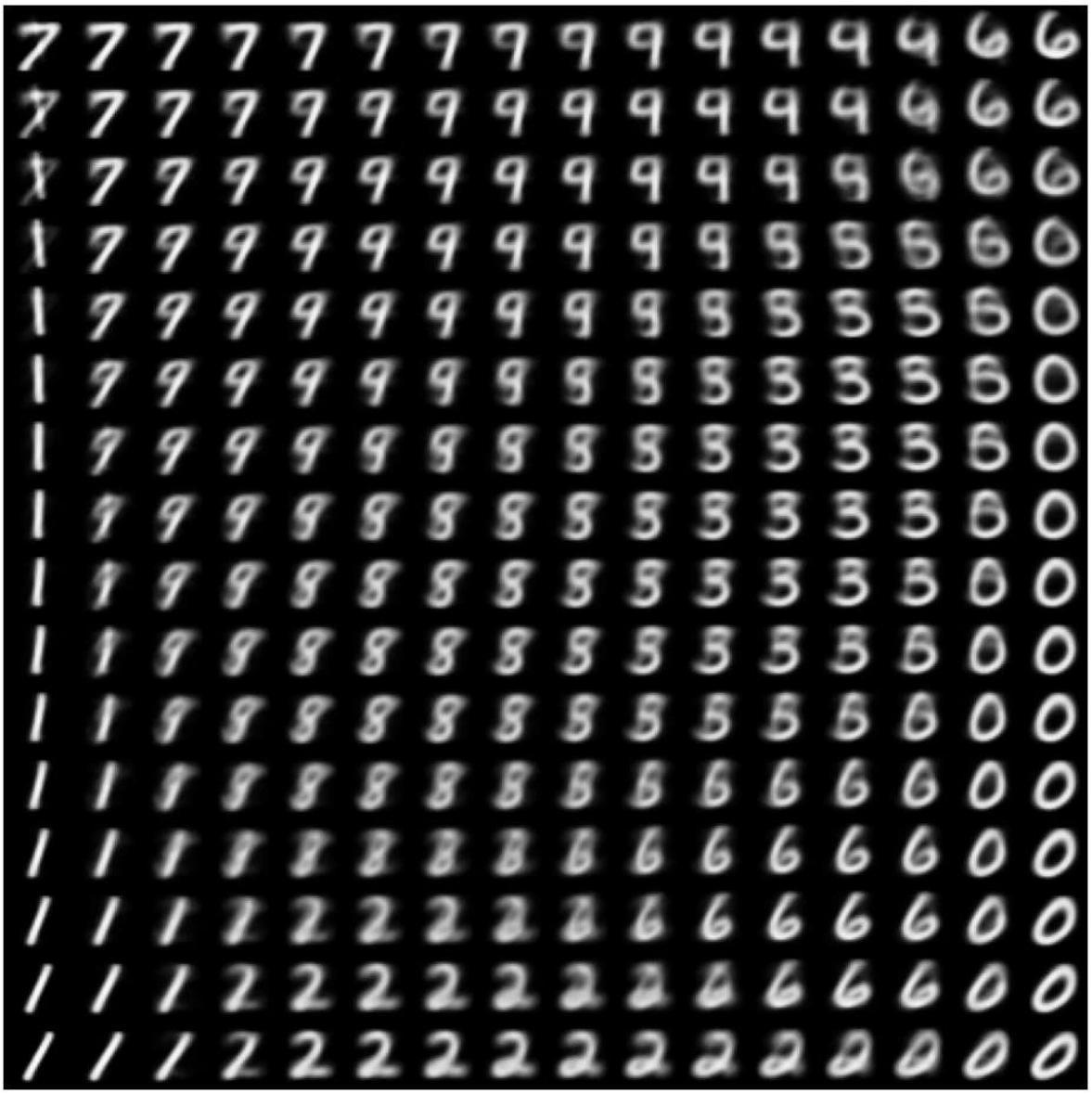
β = 3

In [ ]:

```
import os
from scipy.stats import norm
```

```
# 生成隐空间内的采样
# 训练时Z_dim设定为20，这里仅观察隐空间前俩维度表示特征
model = VariationalAutoEncoder_Conv(z_dim=2)
model.to(device)
epoches = 15
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
train_loss = []  # 保存每个epoch的训练误差
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练

        # 前向传播
        mean, log_var, gen_x = model(x)

        # 计算损失函数 二选一 即可
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, 3)
        batch_loss.append(loss.item())

        # 反向传播和优化
        optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 梯度下降

        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
                                   dim=3)  #torch.Size([128, 1, 28, 56])

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

num_per_row, num_per_col = 16, 16
digit_size = 28
figure = np.zeros((digit_size * num_per_row, digit_size * num_per_col))
grid_x = norm.ppf(np.linspace(0.05, 0.95, num_per_row))
grid_y = norm.ppf(np.linspace(0.05, 0.95, num_per_col))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = torch.tensor(np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = model.decode(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        digit = torch.sigmoid(digit)
        figure[i * digit_size: (i + 1) * digit_size,
        j * digit_size: (j + 1) * digit_size] = digit.detach().cpu().numpy() * 255.

plt.figure(figsize=(12, 12))
plt.axis("off")
plt.imshow(figure, cmap='Greys_r')
plt.show()
```

```
epoch[1/15] | loss:53.993572607421875
epoch[2/15] | loss:48.99080161946615
epoch[3/15] | loss:47.26605036621094
epoch[4/15] | loss:46.69642425130208
epoch[5/15] | loss:46.279779052734376
epoch[6/15] | loss:45.92567962239583
epoch[7/15] | loss:45.71086240234375
epoch[8/15] | loss:45.59258842773438
epoch[9/15] | loss:45.46363501790365
epoch[10/15] | loss:45.325813842773435
epoch[11/15] | loss:45.29882972819011
epoch[12/15] | loss:45.17237499186198
epoch[13/15] | loss:45.110591959635414
epoch[14/15] | loss:45.02343063964844
epoch[15/15] | loss:45.03124555664063
```



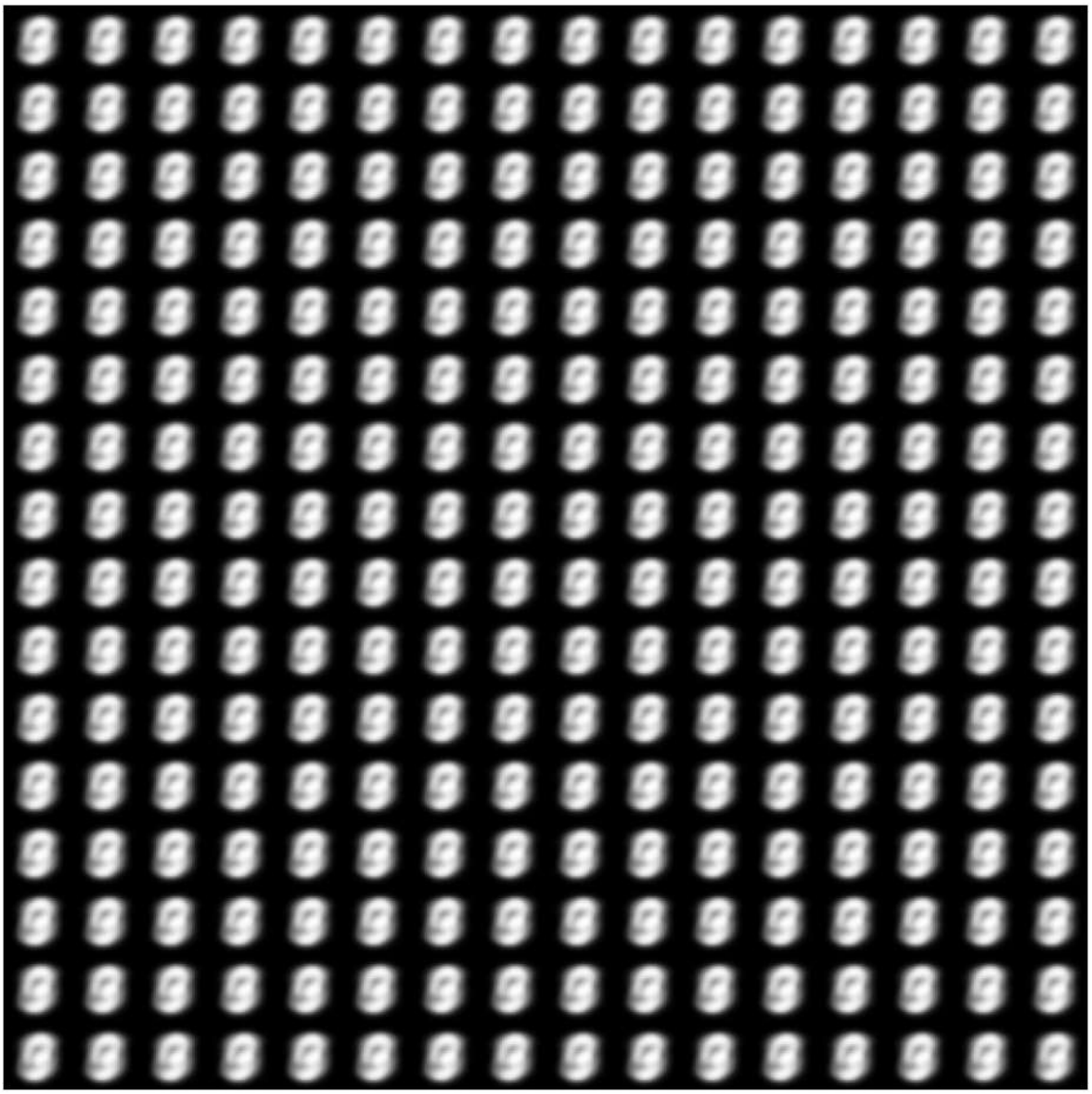
β = 8

In [ ]:

```
import os
from scipy.stats import norm

# 生成隐空间内的采样
# 训练时Z_dim设定为20，这里仅观察隐空间前俩维度表示特征
model = VariationalAutoEncoder_Conv(z_dim=2)
model.to(device)
epoches = 15
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
train_loss = []  # 保存每个epoch的训练误差
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练

        # 前向传播
        mean, log_var, gen_x = model(x)

        # 计算损失函数 二选一 即可
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, 8)
        batch_loss.append(loss.item())

        # 反向传播和优化
        optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 梯度下降

        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
                                   dim=3)  #torch.Size([128, 1, 28, 56])

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

num_per_row, num_per_col = 16, 16
digit_size = 28
figure = np.zeros((digit_size * num_per_row, digit_size * num_per_col))
grid_x = norm.ppf(np.linspace(0.05, 0.95, num_per_row))
grid_y = norm.ppf(np.linspace(0.05, 0.95, num_per_col))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = torch.tensor(np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = model.decode(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        digit = torch.sigmoid(digit)
        figure[i * digit_size: (i + 1) * digit_size,
        j * digit_size: (j + 1) * digit_size] = digit.detach().cpu().numpy() * 255.

plt.figure(figsize=(12, 12))
plt.axis("off")
plt.imshow(figure, cmap='Greys_r')
plt.show()
```

```
epoch[1/15] | loss:54.59007465820312
epoch[2/15] | loss:52.874010213216145
epoch[3/15] | loss:52.84616063639323
epoch[4/15] | loss:52.82879040527344
epoch[5/15] | loss:52.81733885091146
epoch[6/15] | loss:52.79720810546875
epoch[7/15] | loss:52.792586328125
epoch[8/15] | loss:52.78499647623698
epoch[9/15] | loss:52.777526326497394
epoch[10/15] | loss:52.77128342285156
epoch[11/15] | loss:52.76522055664063
epoch[12/15] | loss:52.76194845377604
epoch[13/15] | loss:52.75534208984375
epoch[14/15] | loss:52.75855578613281
epoch[15/15] | loss:52.755916861979166
```



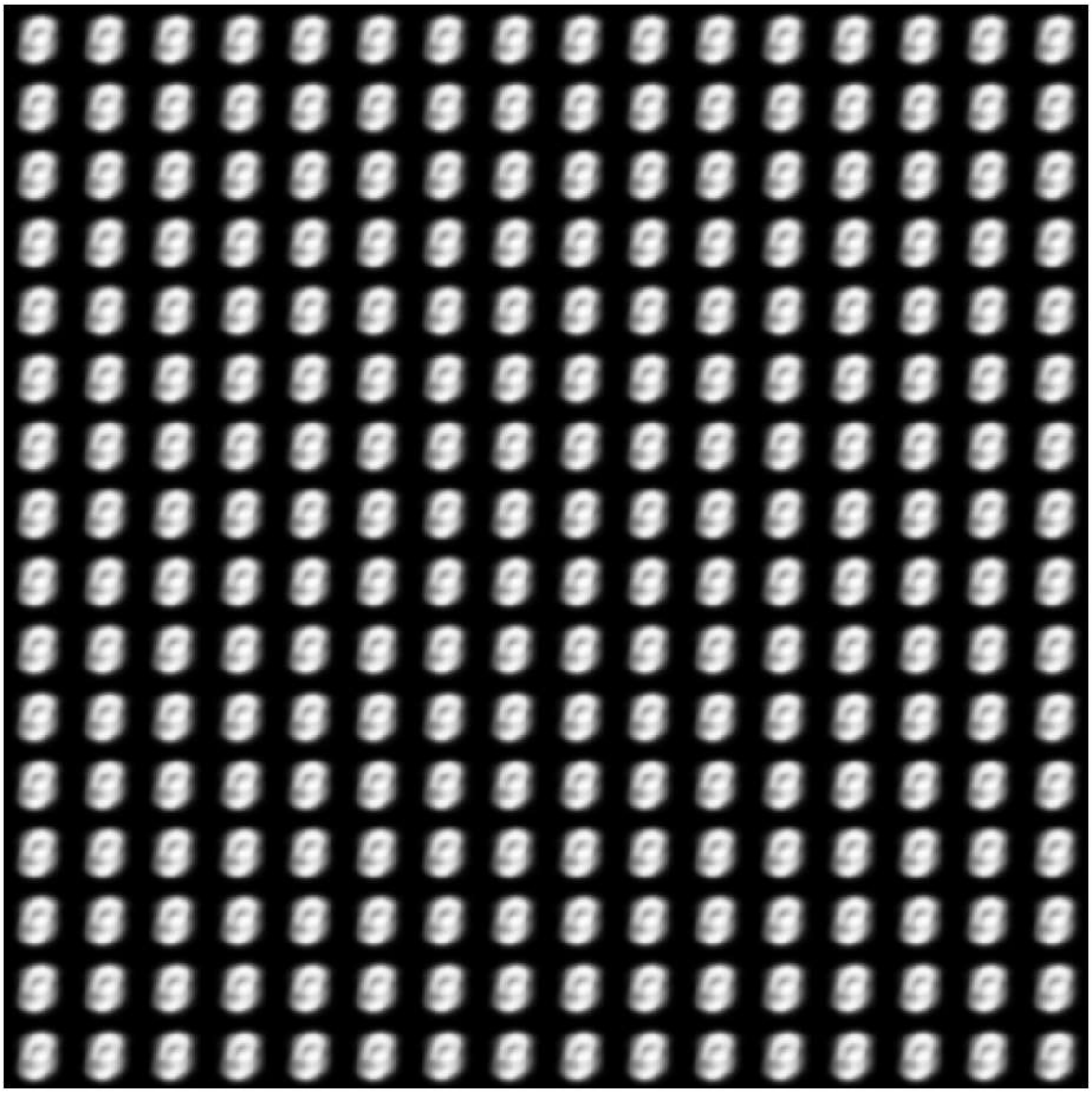
β = 16

In [ ]:

```
import os
from scipy.stats import norm

# 生成隐空间内的采样
# 训练时Z_dim设定为20，这里仅观察隐空间前俩维度表示特征
model = VariationalAutoEncoder_Conv(z_dim=2)
model.to(device)
epoches = 15
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
train_loss = []  # 保存每个epoch的训练误差
x_concatD = torch.empty((128, 1, 28, 56))

for epoch in range(epoches):
    batch_loss = []
    for i, (x, y) in enumerate(dataIter):
        x = x.to(device)  # gpu训练

        # 前向传播
        mean, log_var, gen_x = model(x)

        # 计算损失函数 二选一 即可
        loss = beta_vae_loss_mse(x, gen_x, mean, log_var, 16)
        batch_loss.append(loss.item())

        # 反向传播和优化
        optimizer.zero_grad()  # 每一次循环之前，将梯度清零
        loss.backward()  # 反向传播
        optimizer.step()  # 梯度下降

        if i == 0:
            x_concatD = torch.cat([x.view(-1, 1, 28, 28), gen_x.view(-1, 1, 28, 28)
                                   dim=3)  #torch.Size([128, 1, 28, 56])

    # 输出epoch信息
    train_loss.append(np.sum(batch_loss) / number)
    print("epoch[{}/{}] | loss:{}"
          .format(epoch + 1, epoches, train_loss[epoch]))

num_per_row, num_per_col = 16, 16
digit_size = 28
figure = np.zeros((digit_size * num_per_row, digit_size * num_per_col))
grid_x = norm.ppf(np.linspace(0.05, 0.95, num_per_row))
grid_y = norm.ppf(np.linspace(0.05, 0.95, num_per_col))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = torch.tensor(np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = model.decode(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        digit = torch.sigmoid(digit)
        figure[i * digit_size: (i + 1) * digit_size,
        j * digit_size: (j + 1) * digit_size] = digit.detach().cpu().numpy() * 255.

plt.figure(figsize=(12, 12))
plt.axis("off")
plt.imshow(figure, cmap='Greys_r')
plt.show()
```

```
epoch[1/15] | loss:54.63142722167969
epoch[2/15] | loss:52.87429535319011
epoch[3/15] | loss:52.84610010579427
epoch[4/15] | loss:52.8118392171224
epoch[5/15] | loss:52.809633618164064
epoch[6/15] | loss:52.79538573404948
epoch[7/15] | loss:52.79316895345052
epoch[8/15] | loss:52.78191050618489
epoch[9/15] | loss:52.766322029622394
epoch[10/15] | loss:52.767954125976566
epoch[11/15] | loss:52.767016292317706
epoch[12/15] | loss:52.75978400065104
epoch[13/15] | loss:52.756106754557294
epoch[14/15] | loss:52.75557743326823
epoch[15/15] | loss:52.749213810221356
```



很明显可以看到随着β的增大，隐变量所蕴含信息的减少，更加印证了β引入的作用。

将标准的VAE框架重新设计为 具有强大潜在容量约束和独立先验压力的约束优化问题。通过用可调节此类压力强度的β系数增加下界公式，并因此提高模型所体现的定性性质，β-VAE能够一致而稳健地发现数据中更多的变异因素，并且它学习的表示形式涵盖了更广泛的因子值范围，并且比其他基准测试更清晰地进行了纠缠。。