# 自编码器

## 编码器和编码器网络

### 自编码器概述

自编码器（AutoEncoder，AE）是一种无监督式学习模型，它可以学习到输入数据的隐含特征，同时用学习到的新特征可以重构出原始输入数据。自编码器的基本思想是利用输入数据本身作为监督，来指导神经网络尝试学习一个映射关系，从而得到一个重构输出，该输出尽可能接近原始输入。自编码器的应用领域包括数据降维、特征提取、生成模型、异常检测、去噪等。

举个例子，有时神经网络需要接收大量的输入信息。例如，当输入信息是高清图片时，输入信息的数量可能达到上千万。直接让神经网络从这上千万个信息源中学习是一项艰巨的任务。因此，可以通过对其进行压缩，提取出原始图片中最具代表性的信息，从而减少输入信息的数量，然后将经过压缩的信息输入到神经网络中进行学习。这样一来，学习过程变得简单而轻松。所以此时就需要自编码在这种情况下发挥作用。

### 编码器网络的结构

**编码器**：是自编码器的前半部分，它负责接收输入，这部分能将输入压缩成潜在空间表征，可以用编码函数表示。

**解码器**：是自编码器的后半部分，负责接收编码y这部分能重构来自潜在空间表征的输入，可以用解码函数表示。

因此，整个自编码器可以用函数来描述，其中输出与原始输入相近。

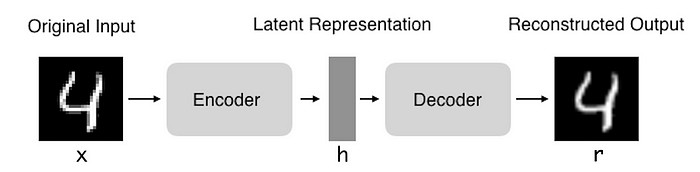


图 6‑1 自编码器结构

### 自编码器的简单实现

1. import numpy as np

使用Keras库中的mnist模块来加载MNIST数据集。MNIST数据集是一个广泛使用的手写数字图像数据集，包含了大量的手写数字图片及其对应的标签。

1. from keras.datasets import mnist
2. (x\_train, \_), (x\_test, \_) = mnist.load\_data()
3. x\_train = (x\_train / 255).reshape(-1, 784)
4. x\_test = (x\_test / 255).reshape(-1, 784)

在这段代码中，我们只关心图像数据而不关心标签数据，所以用一个下划线表示我们不使用标签数据。对训练集和测试集的图像数据进行了归一化处理。将每个像素值除以255，将像素值范围从0到255缩放到0到1之间。

定义激活函数。

1. def sigmoid(z, d=False):
2. return sigmoid(z) \* (1 - sigmoid(z)) + 1e-12 if d else 1 / (1 + np.exp(-z))
4. def relu(z, d=False):
5. return (z > 0)+1e-12 if d else z \* (z > 0)

定义神经网络：第一层使用relu激活函数，输出维度为1024，输入维度为784；第二层使用relu激活函数，输出维度为50，输入维度为1024；第三层使用relu激活函数，输出维度为1024，输入维度为50；第四层使用sigmoid激活函数，输出维度为784，输入维度为1024。

1. layers = [
2. {"act":relu, "shape":(1024,784)},
3. {"act":relu, "shape":(50,1024)},
4. {"act":relu, "shape":(1024,50)},
5. {"act":sigmoid, "shape":(784,1024)}
6. ]

初始化神经网络的参数，包括权重、偏置和一些辅助变量，以及设置优化算法的一些超参数。

1. # 全局变量
2. l, errors, epochs = len(layers), [], 30
4. lr, b1, b2 = 0.002, 0.9, 0.999
5. rw,mw,rb,mb = {},{},{},{}
7. a,w,b,f, = {},{},{},{}
8. for i, layer in zip(range(1,l+1), layers):
9. n\_out, n\_in = layer["shape"]
10. f[i] = layer["act"]
11. # Xavier初始化方法
12. w[i] = np.random.randn(n\_out, n\_in) / n\_in\*\*0.5
13. b[i], rb[i], mb[i] = [np.zeros((n\_out,1)) for i in [1,2,3]]
14. rw[i], mw[i] = [np.zeros((n\_out, n\_in)) for i in [1,2]]

训练和验证过程，通过反向传播和Adam优化算法更新网络参数，同时计算并记录每个训练周期的验证损失：

1. for t in range(1, epochs+1):
2. # 训练
3. for batch in np.split(x\_train, 30):
4. # Forward pass
5. a[0] = batch.T
6. for i in range(1,l+1):
7. a[i] = f[i]((w[i] @ a[i-1]) + b[i])
8. # 反向传播
9. dz,dw,db = {},{},{}
10. for i in range(1,l+1)[::-1]:
11. d = w[i+1].T @ dz[i+1] if l-i else 0.5\*(a[l]-a[0])
12. dz[i] = d \* f[i](a[i],d=1)
13. dw[i] = dz[i] @ a[i-1].T
14. db[i] = np.sum(dz[i], 1, keepdims=True)
16. def adam(m, r, z, dz, i):
17. m[i] = b1 \* m[i] + (1 - b1) \* dz[i]
18. r[i] = b2 \* r[i] + (1 - b2) \* dz[i]\*\*2
19. m\_hat = m[i] / (1. - b1\*\*t)
20. r\_hat = r[i] / (1. - b2\*\*t)
21. z[i] -= lr \* m\_hat / (r\_hat\*\*0.5 + 1e-12)
22. for i in range(1,l+1):
23. adam(mw, rw, w, dw, i)
24. adam(mb, rb, b, db, i)
25. # 验证
26. a[0] = x\_test.T
27. for i in range(1,l+1):
28. a[i] = f[i]((w[i] @ a[i-1]) + b[i])
29. errors += [np.mean((a[l]-a[0])\*\*2)]
30. print("Val loss - ", errors[-1])

将结果输出：

1. import matplotlib.pyplot as plt
2. y\_pred = []
3. a[0] = x\_train[:20].T
4. #forward pass
5. for i in range(1,l+1):
6. a[i] = f[i](w[i] @ a[i-1] + b[i])
7. y\_pred = a[l]
8. plt.figure(figsize=(20,5))
9. for i in range(20):
10. plt.subplot(3, 20, i + 1)
11. plt.imshow(x\_train[i].reshape(28,28), cmap="gray")
12. plt.axis("off")
13. plt.grid(b=False)
14. for i in range(20):
15. plt.subplot(3, 20, i + 1 + 20)
16. plt.imshow(a[l-2].T[i].reshape(5,-1), cmap="gray")
17. plt.axis("off")
18. plt.grid(b=False)
20. for i in range(20):
21. plt.subplot(3, 20, i + 1 + 40)
22. plt.imshow(y\_pred.T[i].reshape(28,28), cmap="gray")
23. plt.axis("off")
24. plt.grid(b=False)
25. plt.show()

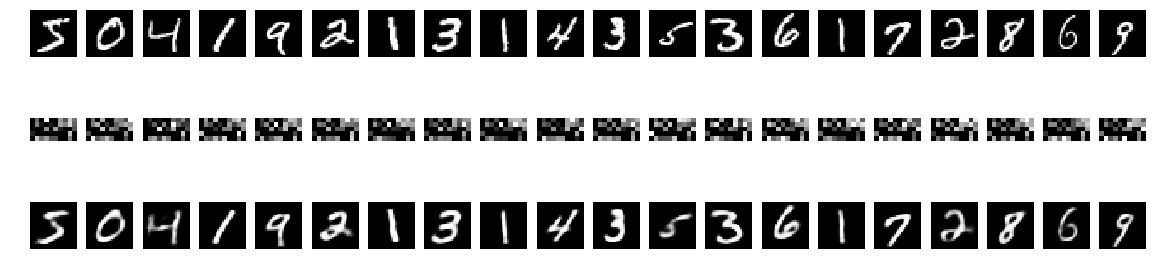


图 6‑2 输出结果

## 6.2去噪自编码器

### 6.2.1 去噪自编码器概述

为了缓解经典AutoEncoder容易过拟合的问题，一个办法是在输入中加入随机噪声；Vincent等人[3]提出了去噪自编码器（Denoising AutoEncoder，DAE）。一个模型，能够从有噪音的原始数据作为输入，而能够恢复出真正的原始数据。这样的模型，更具有鲁棒性。

当采用无监督的方法分层预训练深度网络的权值时，为了学习到较鲁棒的特征，可以在网络的可视层（即数据的输入层）引入随机噪声。

如下图，以经典的MNIST手写数字识别为例，对于输入的数据引入了随机噪点，模型通过训练后可以对有噪音图像更加鲁棒，而这也更符合实际使用的需求。

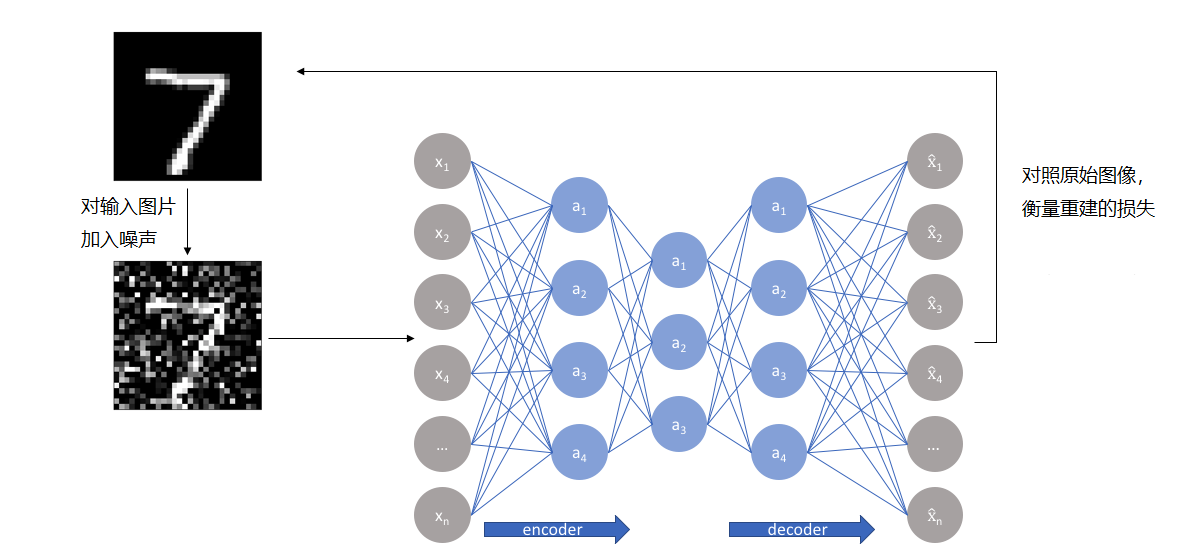


图 6‑3 去噪自编码器示例

### 6.2.2 去噪自编码器的简单实现

下面是用numpy简单实现一个去噪自编码器：

定义sigmoid函数，用于激活隐藏层和重构输入。

1. def sigmoid(x):
2. return 1. / (1 + numpy.exp(-x))

定义dA类，表示去噪自编码器。构造函数用于初始化类的各个属性，包括可见层和隐藏层的单元数量，权重矩阵W，隐藏层偏置hbias，可见层偏置vbias。如果未提供随机数生成器numpy\_rng、权重矩阵W、隐藏层偏置hbias和可见层偏置vbias的值，则使用默认值进行初始化。

1. class dA(object):
2. def \_\_init\_\_(self, input=None, n\_visible=2, n\_hidden=3, \
3. W=None, hbias=None, vbias=None, numpy\_rng=None):
4. self.n\_visible = n\_visible # 可见层（输入层）的单元数量
5. self.n\_hidden = n\_hidden # 隐藏层的单元数量
6. if numpy\_rng is None:
7. numpy\_rng = numpy.random.RandomState(1234)
9. if W is None:
10. a = 1. / n\_visible
11. initial\_W = numpy.array(numpy\_rng.uniform( # 初始化权重矩阵W
12. low=-a,
13. high=a,
14. size=(n\_visible, n\_hidden)))
15. W = initial\_W
16. if hbias is None:
17. hbias = numpy.zeros(n\_hidden) # 初始化隐藏层偏置为0
18. if vbias is None:
19. vbias = numpy.zeros(n\_visible) # 初始化可见层偏置为0
20. self.numpy\_rng = numpy\_rng
21. self.x = input
22. self.W = W
23. self.W\_prime = self.W.T
24. self.hbias = hbias
25. self.vbias = vbias
26. # self.params = [self.W, self.hbias, self.vbias]

定义获取损坏输入的函数。该函数以一定的损坏水平(corruption\_level)对输入进行二值化，返回损坏后的输入。

1. def get\_corrupted\_input(self, input, corruption\_level):
2. assert corruption\_level < 1
3. return self.numpy\_rng.binomial(size=input.shape,
4. n=1,
5. p=1-corruption\_level) \* input

定义获取隐藏层值的函数。通过对输入进行加权求和和偏置项的加法，并通过sigmoid函数将结果映射到[0, 1]的范围内，得到隐藏层的值。

1. def get\_hidden\_values(self, input):
2. return sigmoid(numpy.dot(input, self.W) + self.hbias)

定义获取重构输入的函数。通过对隐藏层进行加权求和和偏置项的加法，并通过sigmoid函数将结果映射到[0, 1]的范围内，得到重构输入。

1. def get\_reconstructed\_input(self, hidden):
2. return sigmoid(numpy.dot(hidden, self.W\_prime) + self.vbias)

定义训练函数。该函数用于对自编码器进行训练。接受学习率(lr)、损坏水平(corruption\_level)和输入(input)作为参数。首先判断是否提供了输入，如果是，则将输入赋值给self.x。然后通过get\_corrupted\_input函数获得损坏后的输入tilde\_x。接下来，通过get\_hidden\_values函数获取隐藏层值y，通过get\_reconstructed\_input函数获取重构输入z。然后计算自编码器的损失值，即L\_h2、L\_h1、L\_vbias和L\_hbias。最后，更新权重矩阵W、隐藏层偏置hbias和可见层偏置vbias。

1. def train(self, lr=0.1, corruption\_level=0.3, input=None):
2. if input is not None:
3. self.x = input
4. x = self.x
5. tilde\_x = self.get\_corrupted\_input(x, corruption\_level)
6. y = self.get\_hidden\_values(tilde\_x)
7. z = self.get\_reconstructed\_input(y)
8. L\_h2 = x - z
9. L\_h1 = numpy.dot(L\_h2, self.W) \* y \* (1 - y)
10. L\_vbias = L\_h2
11. L\_hbias = L\_h1
12. L\_W = numpy.dot(tilde\_x.T, L\_h1) + numpy.dot(L\_h2.T, y)
13. self.W += lr \* L\_W
14. self.hbias += lr \* numpy.mean(L\_hbias, axis=0)
15. self.vbias += lr \* numpy.mean(L\_vbias, axis=0)

定义负对数似然函数。该函数用于计算自编码器的负对数似然损失。首先通过get\_corrupted\_input函数获取损坏后的输入tilde\_x，然后通过get\_hidden\_values函数获取隐藏层值y，通过get\_reconstructed\_input函数获取重构输入z。最后计算交叉熵损失并返回。

1. def negative\_log\_likelihood(self, corruption\_level=0.3):
2. tilde\_x = self.get\_corrupted\_input(self.x, corruption\_level)
3. y = self.get\_hidden\_values(tilde\_x)
4. z = self.get\_reconstructed\_input(y)
5. cross\_entropy = - numpy.mean(
6. numpy.sum(self.x \* numpy.log(z) +
7. (1 - self.x) \* numpy.log(1 - z),
8. axis=1))
9. return cross\_entropy

定义重构函数。该函数用于对给定的输入进行重构。首先通过get\_hidden\_values函数获取隐藏层值y，然后通过get\_reconstructed\_input函数获取重构输入z，并将其返回。

1. def reconstruct(self, x):
2. y = self.get\_hidden\_values(x)
3. z = self.get\_reconstructed\_input(y)
4. return z

## 变分自编码器

在过去的几年里，基于深度学习的生成模型获得了越来越多的关注。依靠大量的数据、精心设计的网络架构和智能训练技术，深度生成模型已经显示出令人难以置信的能力，能够产生高度真实的各种内容，如图像、文本和声音。在这些深度生成模型中，有两个主要的系列脱颖而出，值得特别关注：生成对抗网络（GANs）和变分自编码器（Variational Auto-Encoders，VAEs）。在这里我们主要介绍一下变分自编码器。

### 6.3.1 变分自编码器概述

变分自动编码器（VAE）假定源数据具有某种潜在的概率分布（如高斯），然后试图找到该分布的参数。实现一个变分自动编码器比实现一个自动编码器要有挑战性得多。变分自动编码器是一个生成系统，它的一个主要用途是生成与原始源数据相关的新数据。

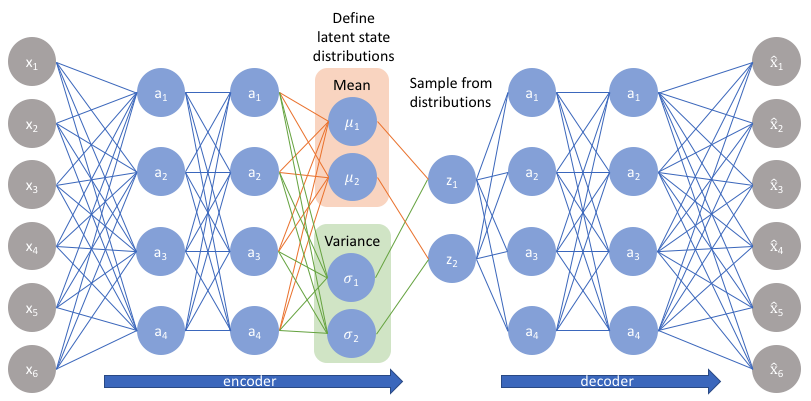


图 6‑3 变分自编码器示例

### 6.3.2 VAE的数学原理

VAE使用KL-divergenc [4]作为其损失函数，其目标是使假定分布与数据集的原始分布之间的差异最小。

假设我们有一个分布z，我们想从它产生观察x。换句话说，我们要计算：

进一步的表示：

但是，的计算可以通过使用积分来完成，因为：

这通常使它成为一个难以处理的分布（需要等于或多于指数时间）。因此，我们需要将近似为，使其成为一个可操作的分布。为了更好地将近似为，我们将最小化KL-分歧损失，它计算两个分布的相似程度：

通过简化，上述最小化问题等同于以下最大化问题：

第一个项代表重建似然，另一个项确保我们学到的分布q与真实的先验分布p相似。因此，我们的总损失由两个项组成，一个是重建误差，另一个是KL-divergenc损失：

### 6.3.3 VAE的实际实现细节

与标准自编码器中直接输出潜状态值不同，变分自编码器（VAE）的编码器模型将输出描述潜空间每个维度分布的参数。由于我们假设先验分布遵循正态分布，我们将输出描述潜态分布均值和方差的两个向量。如果我们要构建一个真正的多变量高斯模型，我们需要定义描述各个维度相关性的协方差矩阵。然而，我们将做出一个简化的假设，即我们的协方差矩阵只在对角线上有非零值，这使得我们可以用一个简单的向量来描述这些信息。

然后，我们的解码器模型将通过从这些定义的分布中进行采样来生成一个潜向量，并开始对原始输入进行重建。

然而，这个采样过程需要额外的注意。在训练模型时，我们需要能够使用反向传播技术计算网络中每个参数与最终输出损失之间的关系。然而，对于随机采样过程，我们无法直接这样做。幸运的是，我们可以利用一个巧妙的想法，即“重参数化技巧”（reparameterization trick）。该技巧建议我们从单位高斯分布中随机采样ε，然后将随机采样的ε通过潜分布的均值μ进行偏移，并乘以潜分布的方差σ进行缩放。

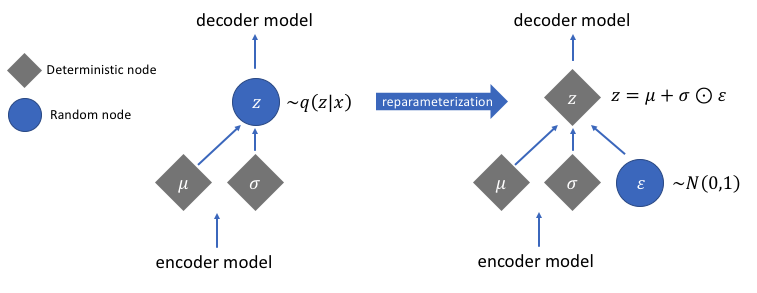


图 6‑4 重参数化示例

通过这种重参数化，我们现在可以优化分布的参数，同时仍然保持从该分布中随机采样的能力。

变分自编码器的主要优势在于我们能够学习到输入数据的平滑潜态表示。对于标准自编码器，我们只需要学习一个编码，使我们能够复制输入。正如您在最左边的图中所看到的，仅关注重建损失确实使我们能够将类别（在本例中是MNIST数字）分开，这使得我们的解码器模型能够复制原始手写数字，但潜空间中的数据分布不均匀。换句话说，潜空间中存在一些不代表任何观察数据的区域。

另一方面，如果我们只关注确保潜分布与先验分布相似（通过KL散度损失项），我们最终会使用相同的单位高斯分布来描述每个观测值，并从中进行采样来描述可视化的潜维度。这实际上将每个观测值都视为具有相同的特征；换句话说，我们没有描述原始数据。

然而，当同时优化这两个项时，我们被鼓励为观测值描述接近先验的潜态分布，但在必要时偏离以描述输入的显著特征。

当构建变分自编码器时，可以检查来自数据的一些样本的潜维度，以了解分布的特征。

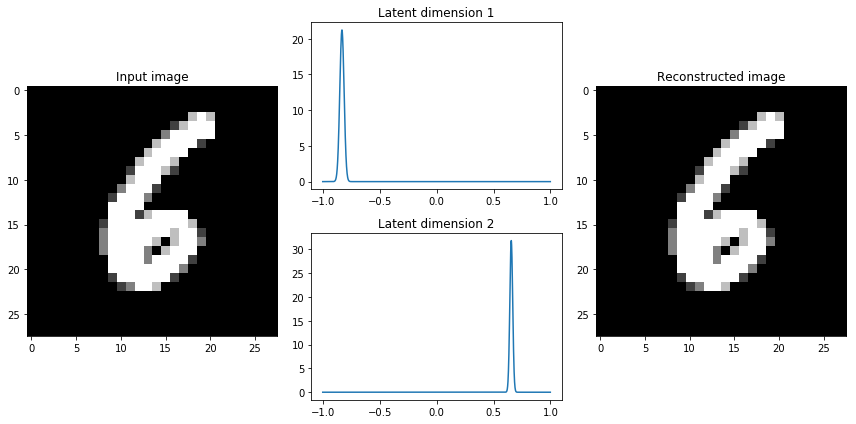


图 6‑5

如果我们观察到潜分布似乎非常紧密，我们可以决定通过设置参数β>1来给予KL散度项更高的权重，鼓励网络学习更广泛的分布。这个简单的洞察力导致了一类新模型的发展 - 解缠变分自编码器。事实证明，通过更大程度地强调KL散度项，我们也在隐含地强制学习到的潜维度是不相关的（通过我们简化假设的对角协方差矩阵）

### 6.3.4 作为生成模型的变分自编码器

通过从潜空间进行采样，我们可以使用解码器网络构建一个生成模型，能够生成类似于训练期间观察到的新数据。具体来说，我们将从我们假设遵循单位高斯分布的先验分布p(z)中进行采样。

下图可视化了在MNIST手写数字数据集上训练的变分自编码器的解码器网络生成的数据。在这里，我们从一个二维高斯分布中采样了一个网格的值，并显示了解码器网络的输出。

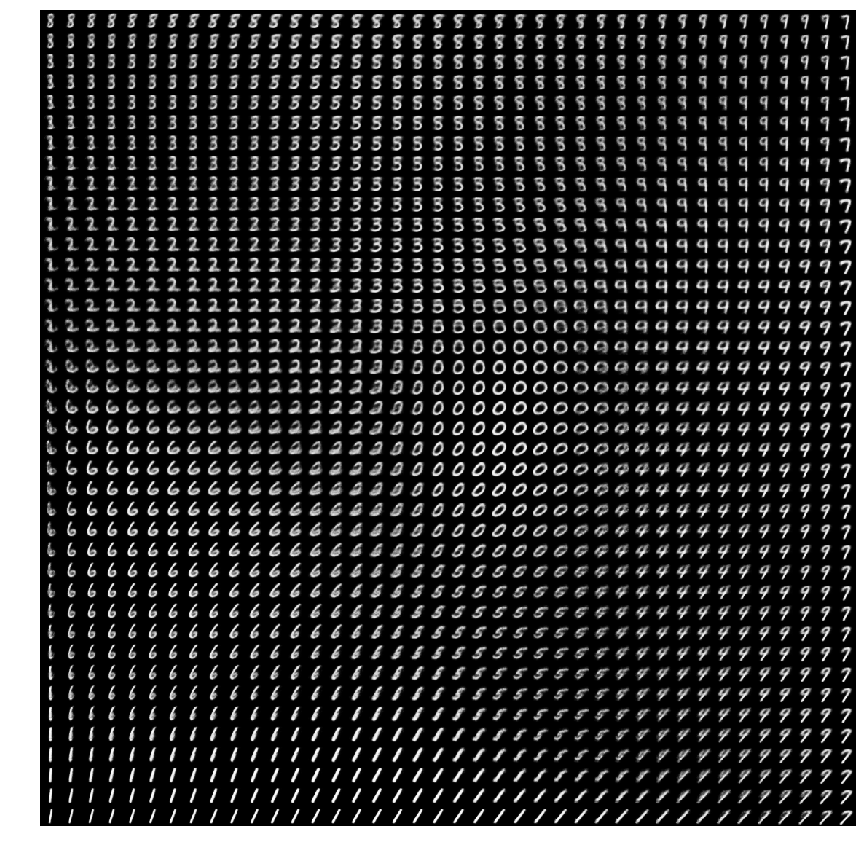


图 6‑6

不同的数字分别存在于潜空间的不同区域，并且它们之间平滑地转换。当我们希望在两个观测之间进行插值时，这种平滑的转换非常有用。

## 6.4 参考文献

使用Zotero工具栏中的Add/Edit Bibliography插入参考文献。

[1] A. Vaswani等, 《Attention is all you need》, *Adv. Neural Inf. Process. Syst.*, 卷 30, 2017.

[2] [Deep inside: Autoencoders. Autoencoders (AE) are neural networks… | by Nathan Hubens | Towards Data Science](https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f)

[3] Extracting and composing robust features with denoising autoencoders, Pascal Vincent etc, 2008

[4] https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained

[5] https://www.jeremyjordan.me/variational-autoencoders/