# 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。

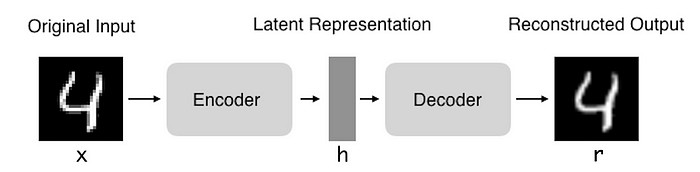


图 6‑1 自编码器结构

### 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()

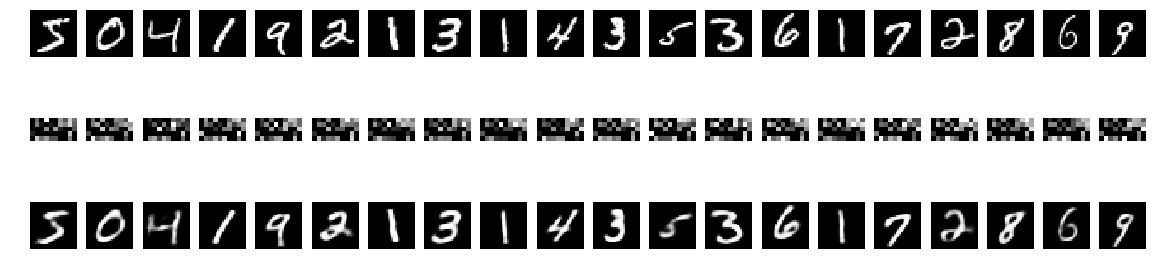


图 6‑2 输出结果

## 6.2去噪自编码器

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。

到目前为止，我已经讨论了训练神经网络的概念，其中输入和输出是相同的，我们的模型的任务是尽可能精确地复制输入，同时通过某种信息瓶颈。回想一下，我提到我们希望我们的自编码器足够敏感以重新创建原始观察结果，但对于训练数据足够不敏感，以便模型学习到可推广的编码和解码。另一种开发可推广模型的方法是略微破坏输入数据，但仍保持未破坏的数据作为我们的目标输出。

通过这种方法，我们的模型无法简单地开发一个记忆训练数据的映射，因为我们的输入和目标输出不再相同。相反，模型学习一个向量场，将输入数据映射到一个低维流形（回想一下我之前的图形，流形描述了输入数据聚集的高密度区域）；如果这个流形准确描述了自然数据，我们实际上已经“抵消掉”了添加的噪声。

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

## 6.3 变分自编码器

## 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