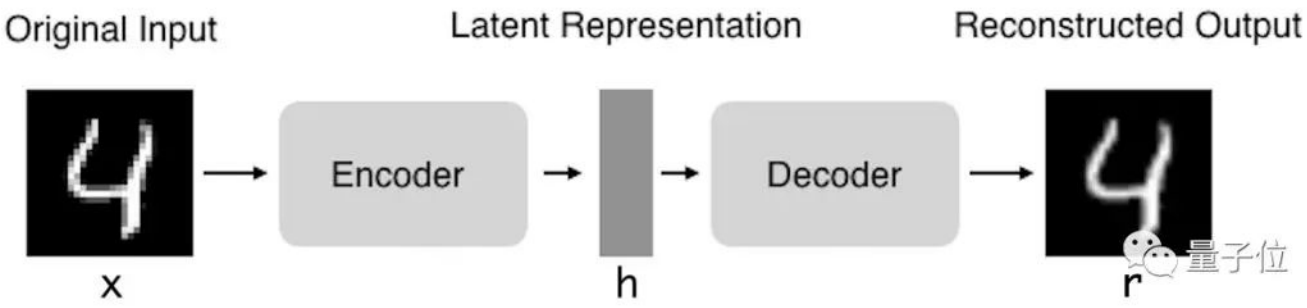
自编码(AutoEncoder)模型及几种扩展之一

自编码器（Autoencoder，AE），是一种利用反向传播算法使得输出值等于输入值的神经网络，它先将输入压缩成潜在空间表征，然后通过这种表征来重构输出。

自编码器由两部分组成：

编码器：这部分能将输入压缩成潜在空间表征，可以用编码函数h=f(x)表示。

解码器：这部分能重构来自潜在空间表征的输入，可以用解码函数r=g(h)表示。



自编码器结构

因此，整个自编码器可以用函数g(f(x)) = r来描述，其中输出r与原始输入x相近。

为何要用输入来重构输出?

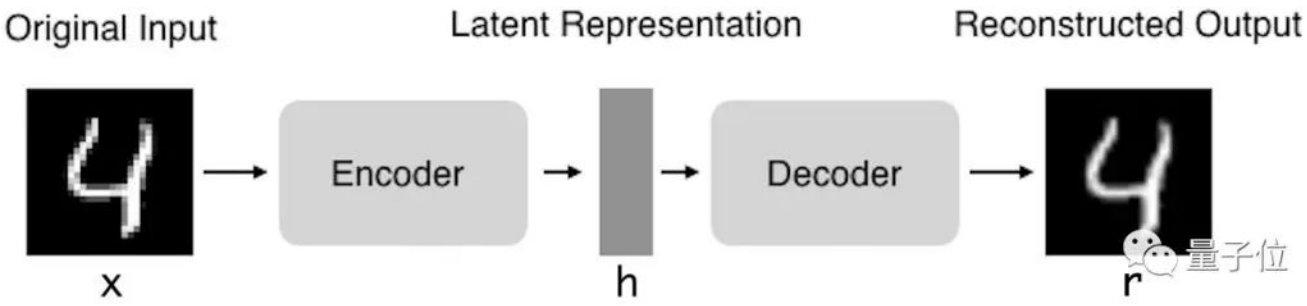
如果自编码器的唯一目的是让输出值等于输入值，那这个算法将毫无用处。事实上，我们希望通过训练输出值等于输入值的自编码器，**让潜在表征h将具有价值属性**。

自编码器（Autoencoder，AE），是一种利用反向传播算法使得输出值等于输入值的神经网络，它先将输入压缩成潜在空间表征，然后通过这种表征来重构输出。

自编码器由两部分组成：

编码器：这部分能将输入压缩成潜在空间表征，可以用编码函数h=f(x)表示。

解码器：这部分能重构来自潜在空间表征的输入，可以用解码函数r=g(h)表示。



自编码器结构

因此，整个自编码器可以用函数g(f(x)) = r来描述，其中输出r与原始输入x相近。

为何要用输入来重构输出?

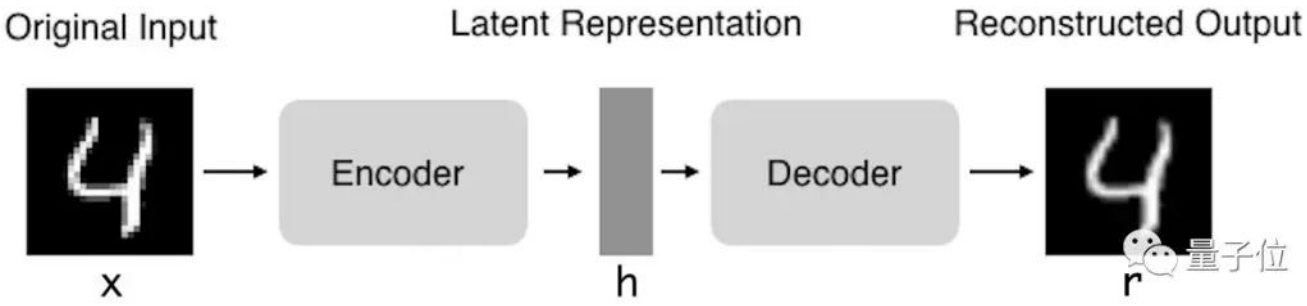
如果自编码器的唯一目的是让输出值等于输入值，那这个算法将毫无用处。事实上，我们希望通过训练输出值等于输入值的自编码器，**让潜在表征h将具有价值属性。**

自编码器（Autoencoder，AE），是一种利用反向传播算法使得输出值等于输入值的神经网络，它先将输入压缩成潜在空间表征，然后通过这种表征来重构输出。

自编码器由两部分组成：

编码器：这部分能将输入压缩成潜在空间表征，可以用编码函数h=f(x)表示。

解码器：这部分能重构来自潜在空间表征的输入，可以用解码函数r=g(h)表示。



自编码器结构

因此，整个自编码器可以用函数g(f(x)) = r来描述，其中输出r与原始输入x相近。

为何要用输入来重构输出?

如果自编码器的唯一目的是让输出值等于输入值，那这个算法将毫无用处。事实上，我们希望通过训练输出值等于输入值的自编码器，**让潜在表征h将具有价值属性。**

对于自编码器，我们往往并不关系输出是啥（反正只是复现输入），**我们真正关心的是中间层的编码，或者说是从输入到编码的映射。**可以这么想，在我们强迫编码y和输入x不同的情况下，系统还

能够去复原原始信号x，那么说明编码y已经承载了原始数据的所有信息，但以一种不同的形式！这就是特征提取啊，而且是自动学出来的！实际上，自动学习原始数据的特征表达也是神经网络和深度学习的核心目的之一

自编码器有如下三个特点：

数据相关性。就是指自编码器只能压缩与自己此前训练数据类似的数据，比如说我们使用mnist训练出来的自编码器用来压缩人脸图片，效果肯定会很差。
数据有损性。自编码器在解压时得到的输出与原始输入相比会有信息损失，所以自编码器是一种数据有损的压缩算法。
自动学习性。自动编码器是从数据样本中自动学习的，这意味着很容易对指定类的输入训练出一种特定的编码器，而不需要完成任何新工作。

自编码器用来干什么?

目前，自编码器的应用主要有两个方面，第一是**数据去噪**，第二是**为进行可视化而降维**。设置合适的维度和稀疏约束，自编码器可以学习到比PCA等技术更有意思的数据投影。

自编码器能从数据样本中进行无监督学习，这意味着可将这个算法应用到某个数据集中，来取得良好的性能，且不需要任何新的特征工程，只需要适当地训练数据。

但是，**自编码器在图像压缩方面表现得不好**。由于在某个给定数据集上训练自编码器，因此它在处理与训练集相类似的数据时可达到合理的压缩结果，但是在压缩差异较大的其他图像时效果不佳。这里，像JPEG这样的压缩技术在通用图像压缩方面会表现得更好。

训练自编码器，可以使输入通过编码器和解码器后，保留尽可能多的信息，但也可以训练自编码器来使新表征具有多种不同的属性。不同类型的自编码器旨在实现不同类型的属性

几种的不同的自编码器

Vanilla autoencoders （基础)

在这种自编码器的最简单结构中，只有三个网络层，即只有一个隐藏层的神经网络。它的输入和输出是相同的，可通过使用Adam优化器和均方误差损失函数，来学习如何重构输入。

在这里，如果隐含层维数（64）小于输入维数（784），则称这个编码器是有损的。通过这个约束，来迫使神经网络来学习数据的压缩表征。

minst数据准备

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Model
from keras.layers import Input, add
from keras.layers.core import Layer, Dense, Dropout, Activation, Flatten, Reshape



#Load the data
# 不需要y
#
(X_train, _), (X_test, _) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)



#Normalize the data
#We want the pixels values between 0 and 1 instead of between 0 and 255
X_train = X_train.astype("float32")/255.
X_test = X_test.astype("float32")/255.

#print('X_train shape:', X_train.shape)
#print(X_train.shape[0], 'train samples')
#print(X_test.shape[0], 'test samples')

#Flatten the images for the Fully-Connected Networks
X_train = X_train.reshape((len(X_train), np.prod(X_train.shape[1:])))
X_test = X_test.reshape((len(X_test), np.prod(X_test.shape[1:])))
```

AE模型构建

```
#The first network is the most simple autoencoder.
# It has three layers : Input - encoded - decoded

input_size = 784
hidden_size = 64
output_size = 784

x = Input(shape=(input_size,))
```

```
h = Dense(hidden_size, activation='relu')(x)
r = Dense(output_size, activation='sigmoid')(h)

#自编码模型
autoencoder = Model(inputs=x, outputs=r)
autoencoder.compile(optimizer='adam', loss='mse')

#输出模型结构
print(autoencoder.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                50240
_________________________________________________________________
dense_2 (Dense)              (None, 784)               50960
=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
```

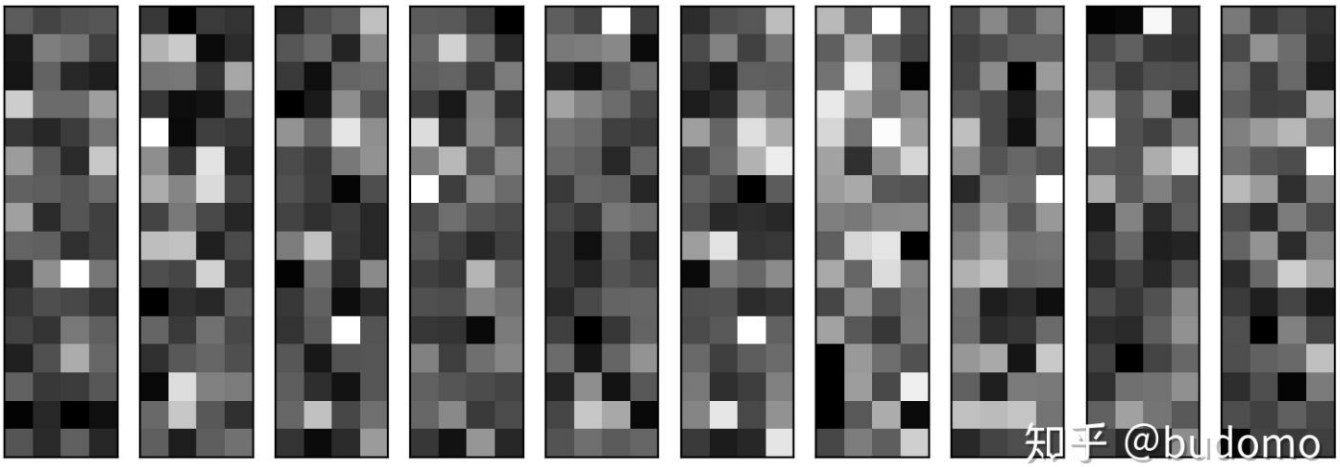

训练模型

```
#训练模型
epochs = 5
batch_size = 128

#Adam优化器和均方误差RMSE损失函数
history = autoencoder.fit(X_train, X_train, batch_size=batch_size, epochs=epochs, verb


#编码过程： 取出编码部分
conv_encoder = Model(x, h)
```

测试集上输出编码器的结果：64维， 只输出前10个样本的编码结果

主要用这个conv_encoder进行编码(!)

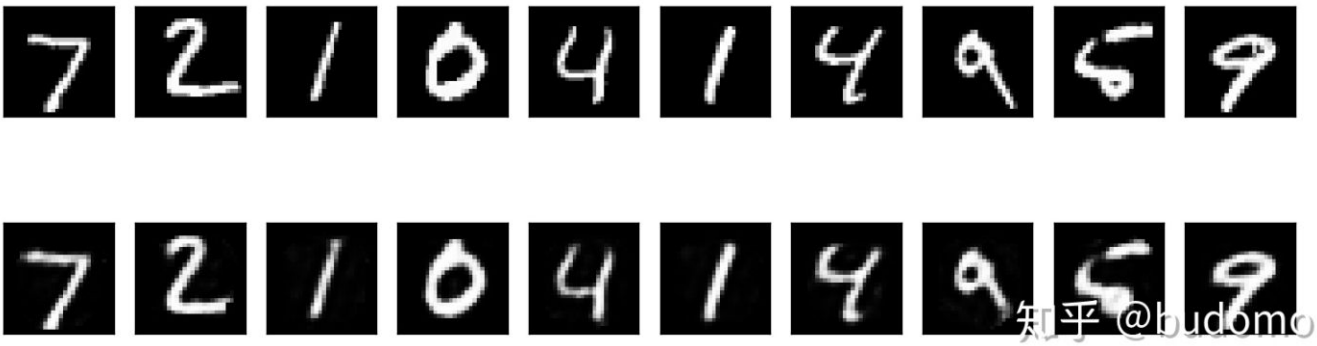
```
#在测试集上进行编码输出，输出图形的编码形式（64维度)
encoded_imgs = conv_encoder.predict(X_test)

n = 10
plt.figure(figsize=(20, 8))
for i in range(n):
    ax = plt.subplot(1, n, i+1)
    plt.imshow(encoded_imgs[i].reshape(4, 16).T)
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



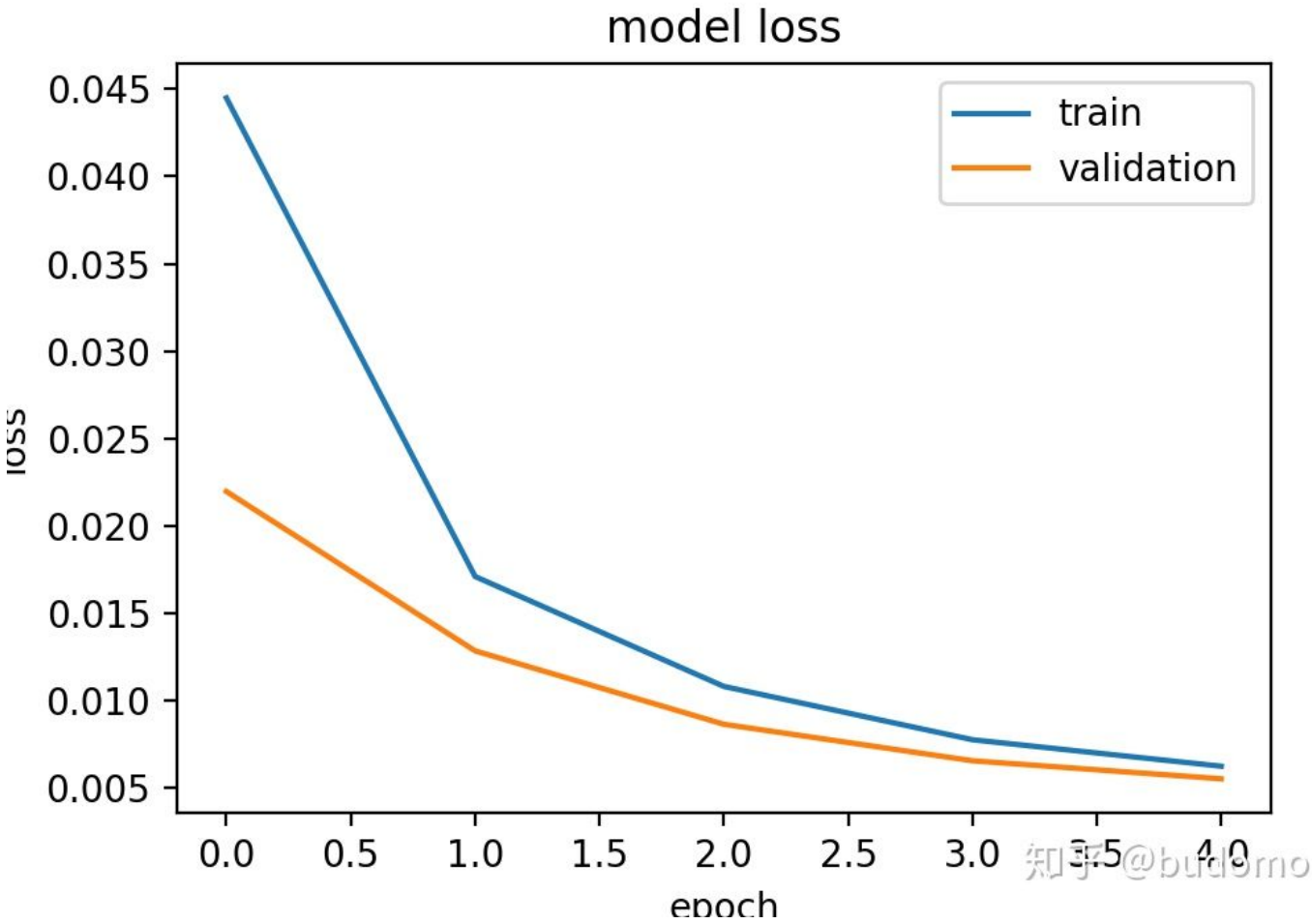
解码结果：

```
#解码过程：
#Predict on the test set
decoded_imgs = autoencoder.predict(X_test)

n = 10
plt.figure(figsize=(20, 6))
for i in range(n):
    # display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(X_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```
    # 解码后的图形：display reconstruction
    ax = plt.subplot(3, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```



损失曲线(MSE损失)：

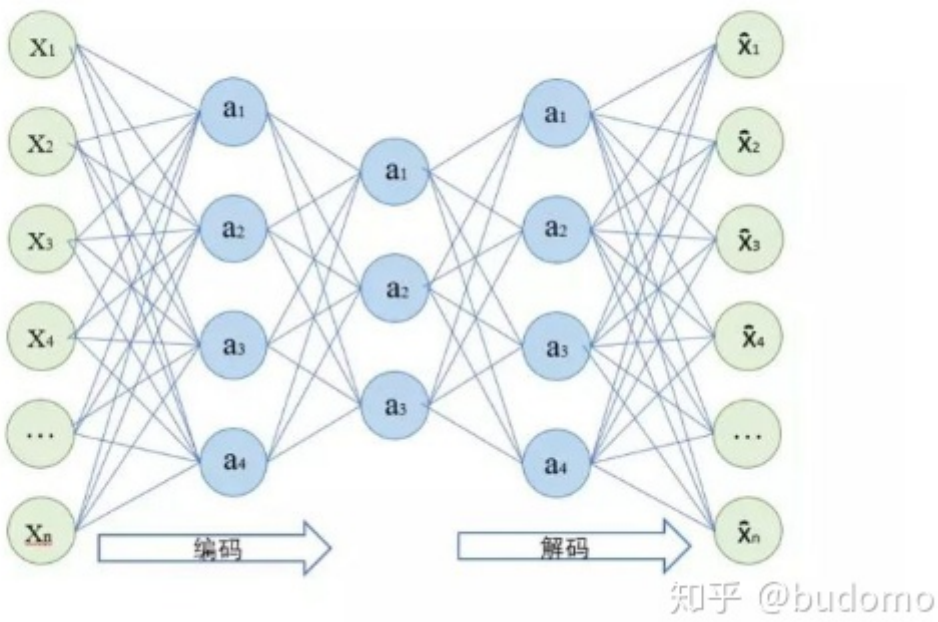


2 多层AutoEncoder

如果一个隐含层还不够，显然可以将自动编码器的隐含层数目进一步提高。

在这里，实现中使用了3个隐含层，而不是只有一个。任意一个隐含层都可以作为特征表征，但是为了使网络对称，我们使用了最中间的网络层。

区别于栈自编码器(Stack AE)



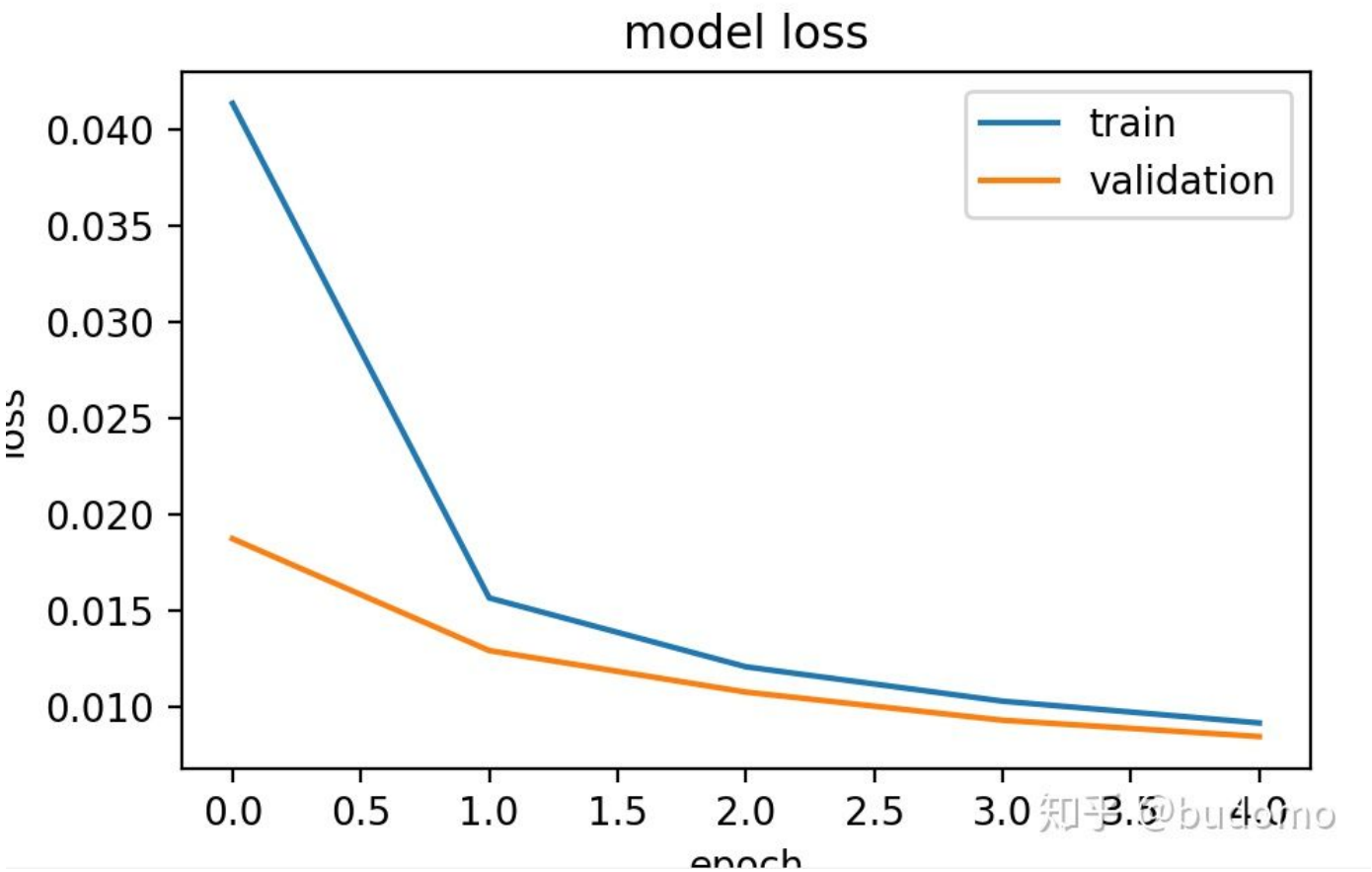
模型构建

```
多层 Autoencoder
#Create the network
# We extend the idea of the first network to more layer

input_size = 784
hidden_size = 128
code_size = 64
output_size = 784

x = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(x)
h = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(h)
r = Dense(output_size, activation='sigmoid')(hidden_2)

#自编码模型
autoencoder = Model(inputs=x, outputs=r)
autoencoder.compile(optimizer='adam', loss='mse')
```

```
#输出模型结构
print(autoencoder.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_4 (Dense)              (None, 784)               101136
=================================================================
Total params: 218,192
Trainable params: 218,192
Non-trainable params: 0
```


其它步骤与前面差不多，不在叙述



似乎MSE还不如基本AE模型好。

3 去噪AutoEncoder

去噪自编码器在自编码器的基础上，在输入中加入**随机噪声再传递给自编码器，通过自编码器来重建出无噪声的输入**。加入随机噪声的方式有很多种。该过程随机的把输入的一些位（最多一半位）设置为0，这样去噪自编码器就需要通过没有被污染的位来猜测被置为零的位。能够从数据的抽样部分预测整体数据的任何子集是在该抽样中能够找到变量联合分布的充分条件(Gibbs抽样的理论依据)，这说明去噪自编码器能够从理论上证明潜在表示能够获取到输入的所有有效特征。

意义在于：在训练集上建立从X_noise到X的模型A，然后A可以对其他噪声数据提取干净的特征。

降噪自动编码器就是在自动编码器的基础之上，**为了防止过拟合问题**而对输入层的输入数据加入噪音，使学习得到的编码器具有鲁棒性而改进的，是Bengio在08年论文：Extracting and composing robust features with denoising autoencoders提出的。

```
#去噪自编码 Autoencoder

x = Input(shape=(28, 28, 1))

# Encoder
```

```
conv1_1 = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
pool1 = MaxPooling2D((2, 2), padding='same')(conv1_1)
conv1_2 = Conv2D(32, (3, 3), activation='relu', padding='same')(pool1)
h = MaxPooling2D((2, 2), padding='same')(conv1_2)



# Decoder
conv2_1 = Conv2D(32, (3, 3), activation='relu', padding='same')(h)
up1 = UpSampling2D((2, 2))(conv2_1)
conv2_2 = Conv2D(32, (3, 3), activation='relu', padding='same')(up1)
up2 = UpSampling2D((2, 2))(conv2_2)
r = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2)



#自编码模型
autoencoder = Model(inputs=x, outputs=r)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')

#autoencoder.compile(optimizer='adam', loss='mse')

#输出模型结构
print(autoencoder.summary())



#训练模型
epochs = 5
batch_size = 128

#Adam优化器和均方误差RMSE损失函数
# 注意：输出是干净的X_train
history = autoencoder.fit(X_train_noisy, X_train, batch_size=batch_size, epochs=epochs

#编码过程： 取出编码部分
conv_encoder = Model(x, h)
```

模型解码：

```
#解码过程：
#Predict on the test set
decoded_imgs = autoencoder.predict(X_test)

n = 10
```

```
plt.figure(figsize=(20, 6))
for i in range(n):
    # display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(X_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # 解码后的图形：display reconstruction
    ax = plt.subplot(3, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

#损失函数绘制Plot the losses
print(history.history.keys())

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```