## 使用自编码器对图片去噪

|  |  |
| --- | --- |
| 【实验名称】 | 使用自编码器对图片去噪 |
| 【知识点准备】 | |
| Python基础，keras，科学计算包基础（pandas，numpy） | |
| 【实验目的】 | |
| 1. 增强对卷积神经网络的理解  2. 学会使用自编码器处理实际问题 | |
| 【实验内容】 | |
| 构建带有噪声的数据  编写神经网络实现自编码器  构建训练过程  查看模型效果 | |
| 【实验要求】 | |
| 对带噪声图像数据实现有效降噪 | |
| 【实验步骤】 | |
| * **单机模式**  1. 打开jupyter，引入必要模块   # -\*- coding: utf-8 -\*-  from keras.datasets import mnist import numpy as np   1. 加载训练数据并进行归一化及变形处理   f = np.load('mnist.npz')  x\_train, y\_train = f['x\_train'], f['y\_train']  x\_test, y\_test = f['x\_test'], f['y\_test']  x\_train = x\_train.astype('float32') / 255. x\_test = x\_test.astype('float32') / 255. x\_train = np.reshape(x\_train, (len(x\_train), 28, 28, 1)) x\_test = np.reshape(x\_test, (len(x\_test), 28, 28, 1))   1. 为图像添加噪声构建输入数据   noise\_factor = 0.5  # 均值为0方差为1 x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)  x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)  x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.) x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)   1. 引入绘图包并展示加上噪声之后的图像   import matplotlib.pyplot as plt  %matplotlib inline  n = 10 plt.figure(figsize=(20, 2)) for i in range(n):  ax = plt.subplot(1, 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) plt.show()     1. 构建网络层并训练保存模型   from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D from keras.models import Model, load\_model  input\_img = Input(shape=(28, 28, 1,)) # N \* 28 \* 28 \* 1  # 卷积  x = Conv2D(32, (3, 3), padding='same', activation='relu')(input\_img) # 28 \* 28 \* 32 x = MaxPooling2D((2, 2), padding='same')(x) # 14 \* 14 \* 32 x = Conv2D(32, (3, 3), padding='same', activation='relu')(x) # 14 \* 14 \* 32 encoded = MaxPooling2D((2, 2), padding='same')(x) # 7 \* 7 \* 32  # 开始反卷积  # 7 \* 7 \* 32 x = Conv2D(32, (3, 3), padding='same', activation='relu')(encoded) # 7 \* 7 \* 32 x = UpSampling2D((2, 2))(x) # 14 \* 14 \* 32 x = Conv2D(32, (3, 3), padding='same', activation='relu')(x) # 14 \* 14 \* 32 x = UpSampling2D((2, 2))(x) # 28 \* 28 \* 32 decoded = Conv2D(1, (3, 3), padding='same', activation='sigmoid')(x) # 28 \* 28 \* 1  # 编译网络  autoencoder = Model(input\_img, decoded) autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')  # 训练并保存模型  autoencoder.fit(x\_train\_noisy, x\_train,  epochs=4,  batch\_size=128,  shuffle=True,  validation\_data=(x\_test\_noisy, x\_test))  autoencoder.save('autoencoder.h5')   1. 加载模型并验证模型效果   autoencoder = load\_model('autoencoder.h5')  decoded\_imgs = autoencoder.predict(x\_test\_noisy)  n = 10 plt.figure(figsize=(20, 4)) for i in range(n):  # display original  ax = plt.subplot(2, 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)    # display reconstruction  ax = plt.subplot(2, n, i + 1 + n)  plt.imshow(decoded\_imgs[i].reshape(28, 28))  plt.gray()  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False) plt.show() | |
|  | |