第 5 章 基于 TensorFlow 2 的无监督学习

CHAPTER 5

本章将研究如何使用 TensorFlow 2 进行无监督学习，无监督学习的目的是发现无标签数据间的模式或关系。这与监督学习不同，监督学习同时提供了数据特征及标签，并希望依此预测出未见过的新特征的标签。而无监督学习的核心是挖掘数据间的潜在结构或规律。换句话说，在没有任何数据间结构先验知识的情况下，以某种方式对数据进行分组或组织，这就是常说的聚类。举个例子，亚马逊公司在其个性化推荐系统中就用了无监督学习，通过识别用户之前购买书籍的类群，对用户买书时的喜好做出推测并提出建议。

无监督学习还可以用于数据压缩。得益于无监督学习，该数据的模式占用更少内存，且不会损害到数据的结构和完整性。本章将介绍两个自动编码器，并利用它们进行数据压缩和图像去噪。

下面开始深入探讨自动编码器。

5.1 自动编码器

自动编码是一种使用人工神经网络(ANN)实现的数据压缩和解压缩算法。它是一种无监督形式的学习算法，只需给它提供未标记的数据即可。该算法的工作方式为强制输入通过瓶颈(即宽度小于原始输入的一层或多层神经网络)来生成输入的压缩版本。为了重构输入(即解压缩)，要将过程逆转，使用反向传播在中间层创建输入的表征，然后将这种表征重构为输出。

自动编码是有损的压缩算法。与无损压缩算法不同，自动编码解压缩的输出与原来的输入相比是退化的，MP3、JPEG 等压缩算法也是如此。

自动编码与数据相关，这意味着自动编码器只能压缩与训练数据类似

的数据。例如，使用汽车图片训练的自动编码器，在街道标志图片上就会表现很差，因为它学习的是汽车独有的特征。

5.2　一个简单的自动编码器

编写一个非常简单的自动编码器，该编码器仅使用了一层人工神经网络。同之前一样，从导入开始，代码如下。

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras import regularizers
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

5.2.1　数据预处理

第一步，加载数据。本示例使用 fashion_mnist 数据集，该数据集创建初衷是替代著名的 MNIST 数据集，本节末尾有时装图像的示例。每个数据项（图像像素）都是 0～255 的无符号整数，因此首先将其转换为 float32，然后归一化到 0～1 范围内，以使其适合后续的学习过程。

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()      #无须标签
x_train = x_train.astype('float32') / 255.                 #归一化
x_test = x_test.astype('float32') / 255.
print(x_train.shape)                                       #输入的形状
print(x_test.shape)
```

形状如下：

```
(60000, 28, 28)
(10000, 28, 28)
```

第二步，为将图像传入一个一维的全连接层，将图像展平。

```
x_train = x_train.reshape(( x_train.shape[0], np.prod(x_train.shape[1:])))
#展平
x_test = x_test.reshape((x_test.shape[0], np.prod(x_test.shape[1:])))
```

```
print(x_train.shape)
print(x_test.shape)
```

形状如下：

```
(60000, 784)
(10000, 784)
```

指定所需的尺寸，代码如下。

```
image_dim = 784          #输入图像的维度=784
encoding_dim = 32        #编码的维度.压缩系数=784/32=24.5
```

第三步，创建单层编码器和自动编码器模型，代码如下。

```
input_image = Input(shape=(image_dim, ))      #输入占位符
encoded_image = Dense(encoding_dim, activation='relu',
activity_regularizer=regularizers.l1(10e-5))(input_image)
                                              #编码是输入的表征
encoder = Model(input_image, encoded_image)
decoded_image = Dense(image_dim, activation='sigmoid')(encoded_image)
#解码是输入的有损重构
autoencoder = Model(input_image, decoded_image)
#创建自动编码器模型,该模型将输入映射到输入的重构
```

第四步，创建解码器模型，代码如下：

```
encoded_input = Input(shape=(encoding_dim,))
                                              #为编码(32维)输入创建占位符
decoder_layer = autoencoder.layers[-1]        #自动编码器模型的最后一层
decoder = Model(encoded_input, decoder_layer(encoded_input))
                                              #创建解码器模型
```

第五步，编译自动编码器。数据几乎均为二进制，所以选择二进制交叉熵损失函数，用以最小化每个像素的二进制交叉熵。

```
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

定义两个实用的检查点。第一个检查点在每轮之后保存模型。如果save_best_only=True，则根据监测值(验证损失)，保留最新的最佳模型。

签名如下：

```
keras.callbacks.ModelCheckpoint(filepath, monitor = 'val_loss', verbose = 0,
save_best_only = False, save_weights_only = False, mode = 'auto', period = 1)
```

声明如下：

```
checkpointer1 = ModelCheckpoint(filepath = 'model.weights.best.hdf5',
verbose = 2, save_best_only = True)
```

当监测值(验证损失)的变化小于最小增量 min_delta 时，第二个检查点停止训练。也就是说，小于 min_delta 的变化被视为没有改善。需要注意的是，必须要进行 patience 轮之后才可以停止训练，签名如下：

```
EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 0, verbose = 0,
mode = 'auto', baseline = None)
```

声明如下：

```
checkpointer2 = EarlyStopping(monitor = 'val_loss', min_delta = 0.0005,
patience = 2, verbose = 2, mode = 'auto')
```

5.2.2　训练

运行训练使用.fit 方法，签名如下：

```
autoencoder.fit(x = None, y = None, batch_size = None, epochs = 1, verbose = 1,
callbacks = None, validation_split = 0.0, validation_data = None, shuffle =
True,
class_weight = None, sample_weight = None, initial_epoch = 0,
steps_per_epoch = None, validation_steps = None, max_queue_size = 10, workers = 1,
use_multiprocessing = False, **kwargs)
```

下面是一个常规的训练。本例将 x 作为输入，并欲在输出中复现(y=x)，所以需注意 x_train 是如何同时赋给 x 和 y 的，代码如下：

```
epochs = 50
autoencoder.fit(x_train, x_train, epochs = epochs, batch_size = 256, verbose = 2,
shuffle = True, validation_data = (x_test, x_test))
```

接下来对测试数据进行压缩和解压缩(编码和解码)。因为 encoder 和 decoder 都是模型，所以可直接调用该方法，并用 predict 方法生成输出。

```
encoded_images = encoder.predict(x_test)            #压缩
decoded_images = decoder.predict(encoded_images)    #解压缩
```

亦可用检查点 ModelCheckpoint。.fit 调用代码如下：

```
epochs = 50
autoencoder.fit(x_train, x_train, epochs = epochs, batch_size = 256, verbose = 2,
callbacks = [checkpointer1], shuffle = True, validation_data = (x_test, x_
test))
```

另外，需按如下方式加载保存的权重，以获取最佳模型。

```
autoencoder.load_weights('model.weights.best.hdf5')
encoded_images = encoder.predict(x_test)
decoded_images = decoder.predict(encoded_images)
```

同样地，可以使用检查点 EarlyStopping。这种情况下的.fit 调用代码为：

```
epochs = 50
autoencoder.fit(x_train, x_train, epochs = epochs, batch_size = 256, verbose = 2,
callbacks = [checkpointer2], shuffle = True, validation_data = (x_test, x_
test))
```

5.2.3 结果显示

下面的代码可将压缩前和解压缩后的图片显示到屏幕上。

```
plt.subplot(nrows, ncols, index, ** kwargs)
```

将子图绘制在具有 nrows 行和 ncols 列的网格上，索引位置从左上角的 1 开始，向右增加来放置时装图像。

```
number_of_items = 12      #要显示的图像数目
plt.figure(figsize = (20, 4))
for i in range(number_of_items):
    #压缩前显示
    graph = plt.subplot(2, number_of_items, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    graph.get_xaxis().set_visible(False)
    graph.get_yaxis().set_visible(False)
    #解压缩后显示
```

```
        graph = plt.subplot(2, number_of_items, i + 1 + number_of_items)
        plt.imshow(decoded_images[i].reshape(28, 28))
        plt.gray()
        graph.get_xaxis().set_visible(False)
        graph.get_yaxis().set_visible(False)
    plt.show()
```

压缩前的图像显示结果如图 5-1 所示。

图 5-1　压缩前的图像

解压缩后的图像显示结果如图 5-2 所示。

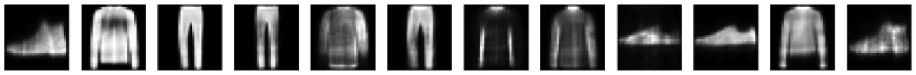

图 5-2　解压缩后的图像

显而易见，压缩/解压缩是有损的。为了做合理性检验，设置 encoding_dim＝768(隐藏层节点数与输入相同)得到如图 5-3 所示的结果。

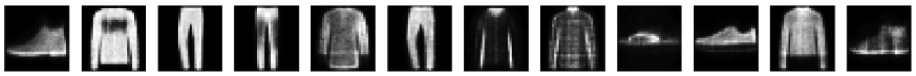

图 5-3　改变编码维度后的图像

可见，图 5-3 更接近原始图片。5.3 节将研究自动编码的另一个应用。

5.3　自动编码器的应用——去噪

自动编码器还有另一个很实用的应用——去噪，即去除图像中的小随机伪影(噪声)。下面会用多层卷积编码器取代简单的单层自动编码器来实现去噪。

首先在时装图像中加入人工噪声，然后使用自动编码器将其消除。同时本节还运用 TensorBoard 来检查网络训练指标。

5.3.1　设置

导入所有用于卷积网络的模块。

需要注意的是，不必显式地使用 Keras，因为它是 TensorFlow 本身的一个模块。

```
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D,
UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.callbacks import TensorBoard
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

5.3.2 数据预处理

首先，加载图像数据，本例不需要标签，只关注图像本身。

```
(train_x, _), (test_x, _) = fashion_mnist.load_data()
```

如前所述，将图像数据转换为 0～1 范围内的 float32 类型。

```
train_x = train_x.astype('float32') / 255.
test_x = test_x.astype('float32') / 255.
```

检查训练数据和测试数据的形状，代码如下：

```
print(train_x.shape)
print(test_x.shape)
```

结果如下：

```
(60000, 28, 28)
(10000, 28, 28)
```

卷积层所需的输入形状，由以下代码给出。

```
train_x = np.reshape(train_x, (len(train_x), 28, 28, 1))
test_x = np.reshape(test_x, (len(test_x), 28, 28, 1))
```

其中，形状中的“1”为灰度通道，下面是形状的合理性检验。

```
print(train_x.shape)
print(test_x.shape)
```

结果如下：

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
```

为了给图像引入随机噪声，在训练集和测试集中加入一个由 np.random.normal 产生的高斯数组，签名如下：

```
numpy.random.normal(loc = 0.0, scale = 1.0, size = None)
```

此处，loc 是分布的中心，scale 是标准差，size 是输出形状，代码如下：

```
noise = 0.5
train_x_noisy = train_x + noise * np.random.normal(loc = 0.0, scale = 1.0,
size = train_x.shape)
test_x_noisy = test_x + noise * np.random.normal(loc = 0.0, scale = 1.0,
size = test_x.shape)
```

结果值可能会超出 0～1 的范围，所以需要将这些值进行剪裁，令其落入该范围内。

```
train_x_noisy = np.clip(train_x_noisy, 0., 1.)
test_x_noisy = np.clip(test_x_noisy, 0., 1.)
```

5.3.3　带噪声的图像

从测试集中打印一些有噪声的图像，注意调整图像的形状以便显示，代码如下：

```
plt.figure(figsize = (20, 2))
for i in range(number_of_items):
  display = plt.subplot(1, number_of_items,i + 1)
  plt.imshow(test_x_noisy[i].reshape(28, 28))
  plt.gray()
  display.get_xaxis().set_visible(False)
  display.get_yaxis().set_visible(False)
plt.show()
```

结果如图 5-4 所示。

很明显，从噪声中分辨出原始图像比较困难。

图 5-4　带噪声的图像

5.3.4　创建编码层

接下来创建编码和解码层。使用 Keras 的函数 API 来定义模型，从一个输入占位符开始，并以（下一个）卷积层所要求的格式定义。

```
input_image = Input(shape = (28, 28, 1))
```

接着是一个卷积层，回忆卷积层的签名。

```
Conv2D(filters, kernel_size, strides = (1, 1), padding = 'valid',
data_format = None, dilation_rate = (1, 1), activation = None, use_bias = True,
kernel_initializer = 'glorot_uniform', bias_initializer = 'zeros',
kernel_regularizer = None, bias_regularizer = None, activity_regularizer =
None,
kernel_constraint = None, bias_constraint = None, * * kwargs)
```

以上参数基本使用默认值。该卷积层 Conv2D 卷积核大小为（3，3），同时也是 Keras 用于输入图像的滑动窗口的大小。padding＝'same'表示图像周围用 0 填充，该填充方式保证了卷积的输出和输入大小相同。默认步长（1，1），表示滑动窗口每次以一个像素点从左至右水平移动到图像的另一端，之后向下移动一个像素，以此类推。接下来，可用以下代码查看每一个神经层的形状。

```
im = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu',
padding = 'same')(input_image)
print(x.shape)
```

结果如下：

```
(?, 28, 28, 32)
```

其中，？表示输入项的数量。

接着是 MaxPooling2D 层。该池化层以一个大小为（2，2）的滑动窗口在图像上移动，取它在每个窗口中找到的最大值作为池化结果。签名如下。

```
MaxPooling2D(pool_size = (2, 2), strides = None, padding = 'valid', data_format =
None, ** kwargs)
```

这是一个下采样的示例，因为生成的图像尺寸减小了。

```
im = MaxPooling2D((2, 2), padding = 'same')(im)
print(im.shape)
```

结果如下：

```
(?, 14, 14, 32)
```

编码层的其余部分如下：

```
im = Conv2D(32, (3, 3), activation = 'relu', padding = 'same')(im)
print(im.shape)
encoded = MaxPooling2D((2, 2), padding = 'same')(im)
print(encoded.shape)
```

以上就是所有的编码实现。

5.3.5　创建解码层

为了构建解码层对图像进行解码，需要反转编码过程，并使用上采样层(UpSampling2D)代替最大池化层。上采样层分别通过 size[0]和 size[1]复制数据的行和列。

此种情况虽然损失了细粒度，但消除了最大池化层的影响。签名如下：

```
UpSampling2D(size = (2, 2), data_format = None, ** kwargs)
```

使用以下方法：

```
im = UpSampling2D((2, 2))(im)
```

下面是解码层。

```
im = Conv2D(32, (3, 3), activation = 'relu', padding = 'same')(encoded)
print(im.shape)
im = UpSampling2D((2, 2))(im)
print(im.shape)
```

```
im = Conv2D(32, (3, 3), activation = 'relu', padding = 'same')(im)
print(im.shape)
im = UpSampling2D((2, 2))(im)
print(im.shape)
decoded = Conv2D(1, (3, 3), activation = 'sigmoid', padding = 'same')(im)
print(decoded.shape)
```

结果如下：

```
(?, 7, 7, 32)
(?, 14, 14, 32)
(?, 14, 14, 32)
(?, 28, 28, 32)
(?, 28, 28, 1)
```

以上就是解码层反转编码层的过程。

5.3.6 模型概要

图 5-5 为模型概要。

层(类型)	输出形状	参数
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 32)	9248
up_sampling2d (UpSampling2D)	(None, 14, 14, 32)	0
conv2d_3 (Conv2D)	(None, 14, 14, 32)	9248
up_sampling2d_1 (UpSampling2	(None, 28, 28, 32)	0
conv2d_4 (Conv2D)	(None, 28, 28, 1)	289

总参数：28 353
可训练参数：28 353
不可训练参数：0

图 5-5 模型概要

如何得出参数数量很重要，计算公式为

卷积核数量×核大小×上一层深度＋卷积核数量(用于偏置)

- **input_1**：这是一个占位符，没有可训练参数。
- **conv2d**：卷积核数量＝32，核大小＝3×3＝9，上一层深度＝1，所以有32×9＋32＝320。

- **max_pooling2d**：最大池化层，没有可训练参数。
- **The conv2d_1**：卷积核数量＝32，核大小＝3×3＝9，上一层深度＝14，所以有 32×9×32＋32＝9248。
- **conv_2d_2**，**conv2d_3**：同 **conv2d_1**。
- **conv2d_4**：1×9×32＋1＝289。

5.3.7　模型实例化、编译和训练

下面，用输入层和输出层将模型实例化，并用.compile 方法设置模型以进行训练。

```
autoencoder = Model(inputs = input_img, outputs = decoded)
autoencoder.compile(optimizer = 'adadelta', loss = 'binary_crossentropy')
```

准备训练模型，并尝试恢复时装物品的图像。为了查看某些训练指标，为 TensorBoard 提供一个回调。Keras TensorBoard 签名如下：

```
keras.callbacks.TensorBoard( ["log_dir = './logs'", 'histogram_freq = 0',
'batch_size = 32',
'write_graph = True', 'write_grads = False', 'write_images = False',
'embeddings_freq = 0', 'embeddings_layer_names = None',
'embeddings_metadata = None', 'embeddings_data = None', "update_freq =
'epoch'"], )
```

参数基本上都使用默认值，如下所示：

```
tb = [TensorBoard(log_dir = './tmp/tb', write_graph = True)]
```

接下来，使用.fit()方法训练自动编码器。方法签名如下：

```
fit(x = None, y = None, batch_size = None, epochs = 1, verbose = 1, callbacks =
None,
validation_split = 0.0, validation_data = None, shuffle = True,
class_weight = None, sample_weight = None, initial_epoch = 0,
steps_per_epoch = None, validation_steps = None, validation_freq = 1)
```

注意观察如何将 x_train_noise 作为特征（输入）以及将 x_train 作为标签（输出）。

```
epochs = 100
batch_size = 128
autoencoder.fit(x_train_noisy, x_train,
epochs = epochs,batch_size = batch_size, shuffle = True,
validation_data = (x_test_noisy, x_test), callbacks = tb)
```

5.3.8 图像去噪

接着，对测试集中的带噪声图像去噪。按照下面第一行代码的形式，对所有测试集进行解码，然后以一个固定值（number_of_items）循环并显示图像。注意，每个图像（im）在显示之前都需要重塑形状。

```
decoded_images = autoencoder.predict(test_noisy_x)
number_of_items = 10
plt.figure(figsize = (20, 2))
for item in range(number_of_items):
  display = plt.subplot(1, number_of_items,item + 1)
  im = decoded_images[item].reshape(28, 28)
plt.imshow(im, cmap = "gray")
  display.get_xaxis().set_visible(False)
  display.get_yaxis().set_visible(False)
plt.show()
```

得到如图 5-6 所示的结果。

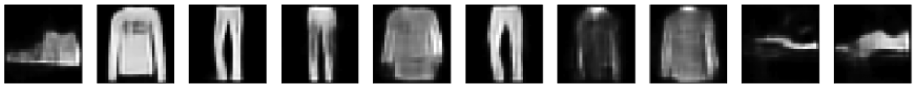

图 5-6 去噪后的图像

考虑到图像最初的模糊程度，在恢复图像上，去噪器已经尽力给出了一个很好的结果。

5.3.9 TensorBoard 输出

要查看 tensorboard 的输出，需在命令行中使用如下命令：

```
tensorboard -- logdir = ./tmp/tb
```

还需访问 http://localhost:6006。

下面两张图分别展示训练和验证时期，损失函数（y 轴）随着更新轮次

(x 轴)的变化。

训练损失如图 5-7 所示。

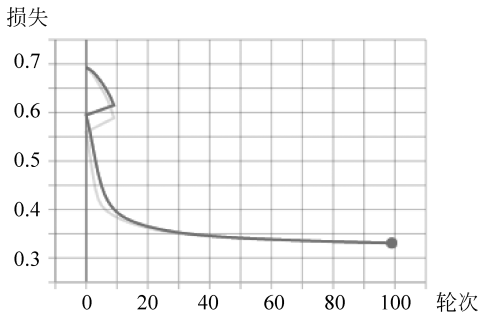

图 5-7　训练损失

验证损失如图 5-8 所示。

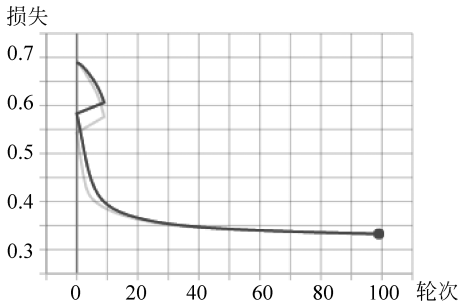

图 5-8　验证损失

以上就是对自动编码器的研究。

5.4　小结

本章研究了自动编码器在无监督学习中的两个应用：第一个是压缩数据，第二个是图像去噪——从图像中去除噪声。

第 6 章将讨论神经网络在图像处理和识别中的应用。