第 3 章

CHAPTER 3

Keras 深度学习

本章将介绍不同的神经网络架构，将创建 Keras 顺序模型——构建单层和多层模型，并评估训练模型的性能。不同架构的网络将帮助读者了解过拟合和欠拟合。此外本章还将探索可对抗训练数据过拟合的早停法。

3.1 简介

在第 2 章学习了神经网络的数学知识，包括标量、向量、矩阵和张量的线性变换。然后，使用 Keras 创建了我们的第一个神经网络，通过构建一个逻辑回归模型将网站用户分类为准备购买的用户和不准备购买的用户。

本章将继续学习用 Keras 搭建神经网络。本章涵盖了深度学习的基础知识，并提供必要的基础知识以便构建高度复杂的神经网络架构。首先将逻辑回归模型扩展到一个简单的单层神经网络，然后继续扩展到具有多个隐藏层的更复杂的神经网络。

在此过程中，可以了解到神经网络的基本概念，包括用于预测的前向传播、损失计算、用于计算模型参数损失函数的反向传播，以及最后用于模型学习最佳参数的梯度下降模型。我们还将了解各种可用的选择，方便以后根据激活函数、损失函数和优化器来构建和训练神经网络。

此外，还将进一步学习如何评估模型性能，了解如何识别过拟合和欠拟合等问题，同时了解它们是如何影响模型的性能的。我们还将了解用与训练相同的数据集评估模型时的缺点，以及保留部分可用数据集以进行评估的替代方法。然后，学习如何比较这两个数据集的子集模型错误率，以检测模型是否存在高偏差和高方差的问题。了解早停这种可减少过拟合的技术。早停技术基于对数据集两个子集的模型错误率进行比较。

3.2 搭建第一个神经网络

本节将了解深度学习的表示和概念，例如前向传播——数据在神经网络中的传播，将输入值乘以连接的每个节点的权重；反向传播——计算损失函数相对于矩阵权重的梯度；梯度下降——用于寻找损失函数最小值的优化算法。

本书不会深入研究这些概念，因为对于本书而言这不是必须要了解的。但是，这个概念将从本质上帮助任何想要将深度学习应用于实际问题的人们。

然后，将继续使用 Keras 搭建神经网络。可以先从最基础的开始学习，即只有单个隐藏层的神经网络。学习如何用 Keras 定义模型，选择超参数（在训练模型之前设置的模型参数），然后训练模型。在本节结尾，将用 Keras 搭建神经网络来巩固所学的知识，以对数据集进行分类，并观察神经网络如何优于逻辑回归等更简单的模型。

3.2.1 从逻辑回归到深度神经网络

在第 1 章中，我们了解了逻辑回归模型。在第 2 章中，我们学习了如何用 Keras 实现一个顺序模型。从技术上讲，逻辑回归涉及一个非常简单的神经网络，只有一个隐藏层，且隐藏层中只有一个节点。

图 3.1 是具有二维输入的逻辑回归模型。图 3.1 中圆圈表示的是深度学习中的一个节点或单元。逻辑回归术语和深度学习术语之间存在一些差异。在逻辑回归中，把模型的参数称为系数和截距。在深度学习模型中，参数被称为权重（w）和偏差（b）。

在每个节点/单元，输入乘以一些权重，然后将偏置项添加到这些输入权重的总和中。这可以在图 3.1 中节点上方的计算中看到：输入为 X1 和 X2，权重为 w1 和 w2，偏差为 b。接下来，将一个非线性函数（如逻辑回归模型中的 sigmoid 函数）应用于加权输入的总和，并使用偏置项来计算节点的最终输出。σ 代表图 3.1 所示的计算。在深度学习中，非线性函数称为激活函数，节点的输出称为该节点的激活。

如图 3.2 所示，通过将逻辑回归节点/单元堆叠在一层中来构建单层神经网络是可行的。输入层 X1 和 X2 的每个值都被传递到隐藏层的所有节点。

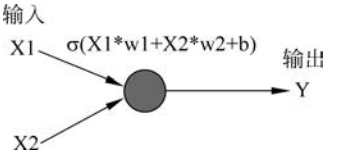

图 3.1 具有二维输入的逻辑回归模型

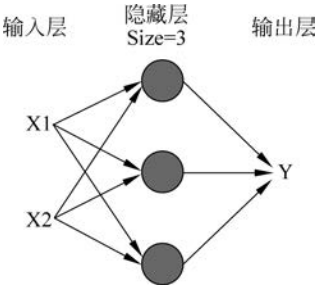

图 3.2 具有二维输入和大小为 3 的隐藏层的单层神经网络

也可通过堆叠多层节点并依次处理来构建多层神经网络。图 3.3 显示了一个二维输入的两层神经网络。

图 3.2 和图 3.3 显示了神经网络最常见的表示方式。每个神经网络都由一个输入层、一个输出层和一个或多个隐藏层组成。如果只有一个隐藏层，则该网络称为浅层神经网络。具有许多隐藏层的神经网络称为深度神经网络，训练它们的过程称为深度学习。

图 3.2 显示了只有一个隐藏层的神经网络，所以这是一个浅层神经网络。而图 3.3 中的神经网络有两个隐藏层，所以它是一个深度神经网络。输入层通常在左侧。对于图 3.3 来说，特征是 X1 和 X2，它们被输入到具有三个节点的第一个隐藏层，箭头表示应用于输入的权重值。在第二个隐藏层，第一个隐藏层的结果成为第二个隐藏层的输入。第一和第二隐藏层之

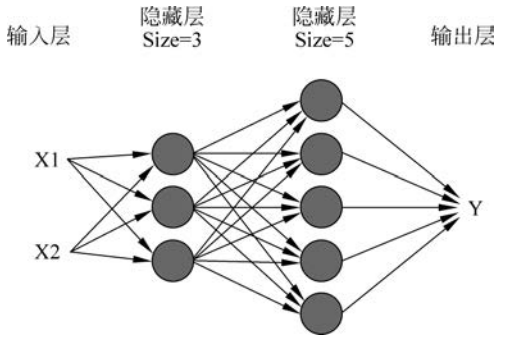

图 3.3 具有二维输入的两层神经网络

间的箭头表示权重。输出通常在最右边的层，在图 3.3 中由标记为 Y 的层表示。

说明：在某些资料中，可能会看到一个网络（如图 3.3 所示的网络）被称为四层网络，这是因为输入和输出层也被计算在内了。然而更常见的是只计算隐藏层，因此图 3.3 的网络被称为两层网络。

在深度学习设置中，输入层的节点数等于输入数据的特征数，输出层的节点数等于输出数据的维数。但是，需要自己设置隐藏层中节点数或隐藏层的大小。如果选择规模更大的层，模型会变得更灵活，也能够使用更复杂的功能进行建模。这种灵活性的提高是需要以更多的训练数据和计算来训练模型的。开发者需要选择的参数为超参数，包括层数、每层的节点数等参数。常见超参数包括训练的轮数和损失函数。

下一节将介绍在每个隐藏层之后应用的激活函数。

3.2.2 激活函数

除了层的大小之外，还需要为模型中添加的每个隐藏层选择一个激活函数，并对输出层执行相同的操作。在使用 Keras 构建神经网络时，了解了逻辑回归模型中的 sigmoid 激活函数，但是也可以选择其他的激活函数。例如，sigmoid 激活函数作为二元分类任务输出层的激活函数是一个不错的选择，因为 sigmoid 函数的结果介于 0 和 1。一些常用的深度学习激活函数有 sigmoid/logistic、tanh（双曲正切）和整流线性单元（ReLU）。

图 3.4 显示了一个 sigmoid 激活函数。

图 3.5 显示了 tanh 激活函数。

图 3.6 显示了 ReLU 激活函数。

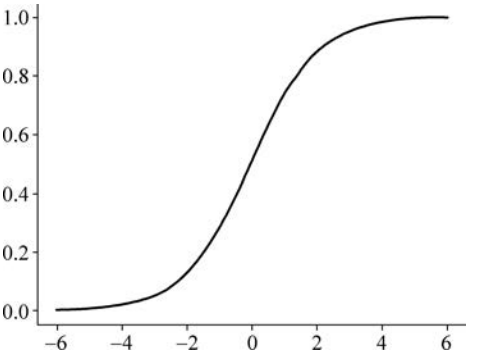

图 3.4 sigmoid 激活函数

如图 3.4 和图 3.5 所示，sigmoid 函数的输出总是在 0～1 范围内，而 tanh 的输出是在 −1～1 范围内，这使得 tanh 激活函数成为更好的选择，因为它保持每一层的输出平均值接近于零。事实上，在构建二元分类器输出层时，sigmoid 是一个不错的激活函数选项，因为它的输出可以解释为给定输入属于类的概率。

tanh 和 ReLU 是隐藏层最常见的激活函数选项。事实证明，使用 ReLU 激活函数时学习过程更快，因为它对大于 0 的输入有一个固定的导数（或斜率），而在其他地方斜率都为 0。

说明：可在 https://keras.io/activations/ 阅读其他 Keras 激活函数的信息。

3.2.3 用于预测的前向传播

神经网络通过执行前向传播对输出进行预测，前向传播需要在神经网络的每一层对输入进

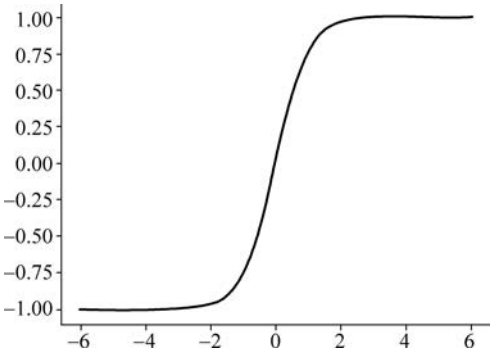

图 3.5 tanh 激活函数

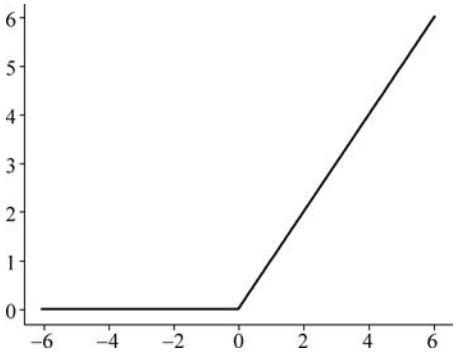

图 3.6 ReLU 激活函数

行计算并继续向前传播直到到达输出层。这里通过一个例子来理解前向传播。

输入数据为二维、输出数据为一维的二进制类标签的两层神经网络如图 3.7 所示。第 1 层和第 2 层的激活函数为 tanh，输出层的激活函数为 sigmoid。

图 3.7 将每一层的权重和偏差显示为具有适当索引的矩阵和向量。对于每一层，权重矩阵中的行数等于前一层的节点数，列数等于该层的节点数。

例如，W1 有 2 行 3 列，因为第 1 层的输入来自列数为 2 的输入层 X，然后第 1 层有 3 个节点。同样 W2 有 3 行 5 列，因为有 3 个节点的第 1 层是第 2 层的输入，所以第 2 层有 5 个节点。然而，偏差始终是一个大小等于该层节点数的向量。深度学习模型中的参数总数等于所有权重矩阵和偏差向量中的元素总数。

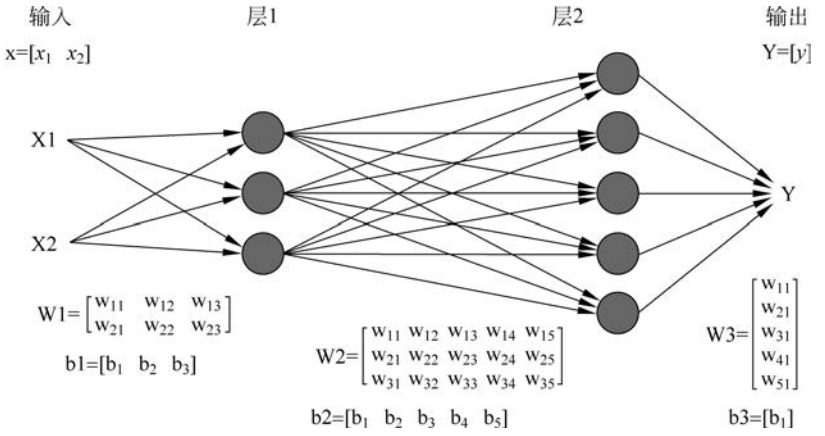

图 3.7 一个两层神经网络

对图 3.7 概述的神经网络执行前向传播的所有步骤如下。

执行前向传播的步骤，

(1) X 是图 3.7 网络的网络输入，是第一个隐藏层的输入。首先，输入矩阵 X 是 X 乘以第 1 层的权重矩阵 W1，并加上偏置 b1。

```
z1 = X*W1 + b1
```

(2) 接下来，第 1 层的输出是通过对上一步的输出 z1 应用激活函数来计算的。

```
a1 = tanh(z1)
```

(3) a1 是第一层的输出,称之为第一层的激活,也是第二层的输入。输入第二层后,a1 将再次被乘以权重矩阵 W2,并加入偏置 b2。

```
z2 = a1 * W2 + b2
```

(4) 对第二层输出 z2 应用激活函数。

```
a2 = tanh(z2)
```

(5) 第二层的输出实际上是下一层(此处为网络最后输出层)的输入。在此之后,第二层的激活是矩阵乘以输出层的权重矩阵 W3,并添加偏置 b3。

```
z3 = a2 * W3 + b3
```

(6) 最后,通过将 sigmoid 激活函数应用于 z3 来计算网络输出 Y。

```
Y = sigmoid(z3)
```

此模型中的参数总数等于 W1、W2、W3、b1、b2 和 b3 中的元素数之和。因此,参数个数可以通过将权重矩阵和偏差中的每个参数相加来计算,等于 6+15+5+3+5+1=35。这些是深度学习过程中需要的学习参数。

现在了解了前向传播的步骤,我们必须评估模型并将其与实际目标值进行比较。下一节将介绍如何使用损失函数,并了解一些可用于分类和回归任务的常见损失函数。

3.2.4 损失函数

模型在学习最佳参数(权重和偏差)时,需要定义一个函数来测量误差,该函数称为损失函数,它为我们提供了衡量网络预测输出与数据集实际结果差异程度的方法。

针对具体的问题和目标,可以使用不同的方式来定义损失函数。例如对于分类问题,损失通常被定义为数据集中错误分类输入的比例,并将其用作模型错误率。对于回归问题,损失函数通常被定义为预测输出与对应的实际输出之间的距离,然后对数据集中的所有样本求平均。

Keras 中常用的损失函数如下。

- mean_squared_error 是回归问题的一种损失函数,对数据集中每个样本计算(真实输出-预测输出),然后返回它们的平均值。
- mean_absolute_error 是回归问题的一种损失函数,对数据集每个样本计算 abs (真实输出-预测输出)并返回平均值。
- mean_absolute_percentage_error 是回归问题的一种损失函数,对数据集每个样本计算 abs [(真实输出-预测输出)/真实输出]并返回平均值再乘以 100%。
- binary_crossentropy 是一个两类/二元分类问题的损失函数。一般来说,交叉熵损失用于计算模型的损失,输出是一个 0~1 的概率数。
- categorical_crossentropy 是多类(多于两个类)分类问题的一种损失函数。

说明:可在 https://keras.io/losses/ 阅读更多有关 Keras 中损失函数所有可用选项的信息。

在训练过程中,不断改变模型参数直到达到模型预测值和真实值之差最小。这个过程被

称为优化，将在后面的部分中详细解释了它的工作原理。对于神经网络，我们使用反向传播来计算损失函数对于权重的导数。

3.2.5 反向传播计算损失函数的导数

反向传播是指从神经网络的输出层到输入层执行微积分链规则的过程，以便计算损失函数相对于每一层的模型参数的导数。函数的导数就是该函数的斜率，我们对损失函数的斜率较为感兴趣，因为它为我们提供了模型参数需要改变的方向，以使损失值达到最小。

微积分的链式法则表明，如果 z 是 y 的函数，y 是 x 的函数，则 z 对 x 的导数可以通过将 z 对 y 的导数乘以 y 对 x 的导数得到。这可以写成如下形式。

$$dz/dx = dz/dy \times dy/dx$$

在深度神经网络中，损失函数是预测输出的函数。可以通过下面给出的等式来证明这一点。

```
loss = L(y_predicted)
```

另一方面，根据前向传播方程，模型预测的输出是模型参数的函数结果，即每一层的权重和偏差。因此根据微积分的链式法则，可以通过计算损失对预测输出的导数乘以预测输出对模型参数的导数来得到损失对模型参数的导数。

下一节将学习在知道损失函数对于权重的导数的情况下，如何最优化权重参数。

3.2.6 通过梯度下降法学习参数

本节将介绍深度学习模型如何学习最优参数。目标是更新权重参数使损失函数最小化，这是一个迭代过程。这个过程被称为学习参数，它是通过使用优化算法来完成的。在机器学习中用于学习参数的一种常见优化算法是梯度下降。

如果把数据集中所有可能的模型参数值的损失平均值绘制出来，它通常是一个凸的形状，如图 3.8 所示。在梯度下降中，我们的目标是找到图中的最小点(Pt)。该算法首先用一些随机值(P1)初始化模型参数。然后，它计算损失和损失相对于该点的参数的导数。正如之前提到的，一个函数的导数实际上就是该函数的斜率。在计算了初始点的斜率之后，就有了需要更新参数的方向。

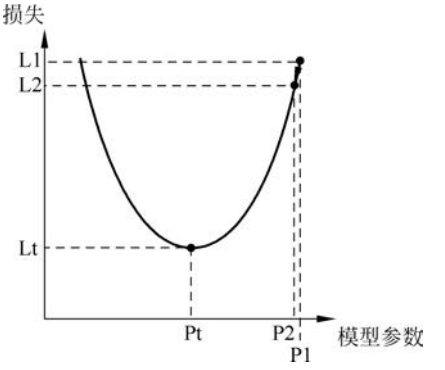

图 3.8 梯度下降算法寻找最小化损失的参数集

超参数，又称为学习率(alpha)，决定了算法从初始点开始能够走多远。选择合适的 alpha 值后，算法更新参数初始值为新值(如图 3.8 P2 点所示)。如图 3.8 所示，P2 离目标点更近。如果一直朝那个方向移动，最终会到达目标点 Pt。该算法在 P2 处再次计算函数的斜率并进行下一步骤。

这里提供了梯度下降算法的伪代码。

```
Initialize all the weights (w) and biases (b) arbitrarily
Repeat Until converge {
Compute loss given w and b
Compute derivatives of loss with respect to w (dw), and with respect to b
(db) using backpropagation
Update w to w - alpha * dw
Update b to b - alpha * db
}
```

总而言之，在训练深度神经网络时重复以下步骤（将参数初始化为一些随机值之后）。

（1）使用前向传播和当前参数来预测整个数据集的输出。

（2）使用预测输出来计算所有算例的损失。

（3）使用反向传播来计算损失相对每一层的权重和偏差的导数。

（4）使用导数值和学习率更新权重和偏差。

在这里讨论的是标准梯度下降算法，它使用整个数据集计算损失和导数以更新参数。梯度下降还有另一种版本，称为随机梯度下降（SGD），每次仅使用一个子集或一批数据示例来计算损失和导数，因此它的学习速度比标准梯度下降更快。

说明：Adam 是另一个常见的优化算法。Adam 在训练深度学习模型时通常优于 SGD。正如已经了解到的，SGD 使用单个超参数（称为学习率）来更新参数。然而，Adam 通过使用学习率、梯度的加权平均值和平方梯度的加权平均值来改进这个过程，在每次迭代时更新参数。

通常在构建神经网络时，需要为优化过程选择两个超参数（称为 batch_size 和 epochs）。batch_size 参数确定了优化算法每次迭代所包含的数据范例的数量。batch_size＝None 相当于梯度下降的标准版本，即每次迭代都使用整个数据集。epochs 参数确定优化算法在停止之前模型训练整个数据集的次数。

例如，假设有一个大小为 n＝400 的数据集，选择 batch_size＝5 和 epochs＝20。在这种情况下，优化器将在一次遍历整个数据集时进行 400/5＝80 次迭代。由于它应该遍历整个数据集 20 次，因此总共将有 80×20 次迭代。

说明：在 Keras 中构建模型时，需要选择训练模型时要使用的优化器类型。除了 SGD 和 Adam 在 Keras 中还有其他一些选项。可以在 https://keras.io/optimizers/ 阅读 Keras 中优化器的所有可能选项的更多信息。

本章中的所有实践和训练都将在 Jupyter Notebook 中开发。从 https://packt.live/39pOUMT 下载本书的 GitHub 存储库以及所有准备好的模板。

训练 3.01　使用Keras实现神经网络

本训练将学习使用 Keras 分步实现神经网络。我们的模拟数据集代表了在森林中发现的各种树木的测量值，如高度、树枝数量、树干基部的周长等。我们的目标是根据所给定的测量结果将数据分类为落叶树和针叶树。首先，执行以下代码块来加载一个由两个类组成的包含 10000 条记录的模拟数据集，这些记录包含两个树种，其中每个数据示例有 10 个特征值。

```
import numpy as np
import pandas as pd
X = pd.read_csv('../data/tree_class_feats.csv')
y = pd.read_csv('../data/tree_class_target.csv')

# Print the sizes of the dataset
print("Number of Examples in the Dataset = ", X.shape[0])
print("Number of Features for each example = ", X.shape[1])
print("Possible Output Classes = ", np.unique(y))
```

期望输出的结果如下。

```
Number of Examples in the Dataset = 10000
Number of Features for each example = 10
Possible Output Classes = [0 1]
```

由于该数据集中的每个数据实例只能属于这两个类中的一个，因此这是一个二元分类问题。二元分类问题在现实生活中非常重要且普遍。例如，假设此数据集中的实例代表森林中10000 棵树的测量结果。目标是使用此数据集构建模型，以预测所测量的每棵树的物种是落叶树种还是针叶树种。树木的 10 个特征可以包括诸如高度、树枝数量和树干周长等预测变量。

输出类 0 表示树是针叶树种，而输出类 1 表示树是落叶树种。

现在，按如下步骤构建和训练 Keras 模型以执行分类。

(1) 在 NumPy 和 TensorFlow 中设置种子，并将模型定义为 Keras 顺序模型。Sequential 模型实际上是层的堆叠。定义模型后，可以根据需要添加任意多的层。

```
from keras.models import Sequential
from tensorflow import random
np.random.seed(42)
random.set_seed(42)
model = Sequential()
```

(2) 向模型中添加一个大小为 10 且激活函数为 tanh 的隐藏层(记住，输入维度必须等于10)。Keras 中有不同类型的图层可供选择。在这里，将只使用最简单的类型层，称为 Dense 层。Dense 层等价于在其他示例中看到的全连接层。

```
from keras.layers import Dense, Activation
model.add(Dense(10, activation='tanh', input_dim=10))
```

(3) 向模型添加大小为 5 的另一个隐藏层和激活函数 tanh。注意，输入维度参数仅提供给第一层，因为下一层的输入维度是已知的。

```
model.add(Dense(5, activation='tanh'))
```

(4) 使用 sigmoid 激活函数添加输出层。注意，输出层的单元数等于输出维度。

```
model.add(Dense(1, activation='sigmoid'))
```

(5) 确保损失函数是二元交叉熵，优化器是 SGD，使用 compile()方法训练模型并打印模型摘要以查看其架构。

```
model.compile(optimizer='sgd', loss='binary_crossentropy', \
              metrics=['accuracy'])
model.summary()
```

代码输出如图 3.9 所示。

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                110
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 55
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 6
=================================================================
Total params: 171
Trainable params: 171
Non-trainable params: 0
_________________________________________________________________
```

图 3.9　已创建模型的摘要

(6) 训练模型 100 轮并将 batch_size 设置为 5，validation_split 设置为 0.2，然后使用 fit() 方法将 shuffle 设置为 false。需要将输入数据 x 及其相应的输出 y 传递给 fit()方法来训练模型。此外，训练网络可能需要很长时间，具体取决于数据集的大小、网络的大小、轮次的数量以及可用的 CPU 或 GPU 的数量。将结果保存到名为 history 的变量中。

```
history = model.fit(X, y, epochs=100, batch_size=5, \
                    verbose=1, validation_split=0.2, \
                    shuffe=False)
```

verbose 参数可以采用以下三个值中的任何一个：0、1 或 2。如果设置 verbose＝0，训练期间不会输出任何信息；verbose＝1 将在每次迭代时打印一个完整的进度条；verbose＝2 将仅打印轮次编号。400 个轮次中最后 5 个轮次的损失细节如图 3.10 所示。

```
Epoch 96/100
8000/8000 [==============================] - 2s 227us/step - loss: 0.1372 - accuracy: 0.9469 - val_loss: 0.1488 - val
_accuracy: 0.9405
Epoch 97/100
8000/8000 [==============================] - 2s 220us/step - loss: 0.1397 - accuracy: 0.9481 - val_loss: 0.1457 - val
_accuracy: 0.9385
Epoch 98/100
8000/8000 [==============================] - 2s 258us/step - loss: 0.1381 - accuracy: 0.9473 - val_loss: 0.1495 - val
_accuracy: 0.9405
Epoch 99/100
8000/8000 [==============================] - 2s 217us/step - loss: 0.1377 - accuracy: 0.9484 - val_loss: 0.1503 - val
_accuracy: 0.9385
Epoch 100/100
8000/8000 [==============================] - 2s 236us/step - loss: 0.1386 - accuracy: 0.9467 - val_loss: 0.1422 - val
_accuracy: 0.9440
```

图 3.10　400 个轮次中最后 5 个轮次的损失细节

(7) 输出模型在训练和验证数据上的正确率和损失作为轮次的函数。

```
import matplotlib.pyplot as plt
%matplotlib inline

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
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 left')
plt.show()
```

图 3.11 显示了上述代码的输出。

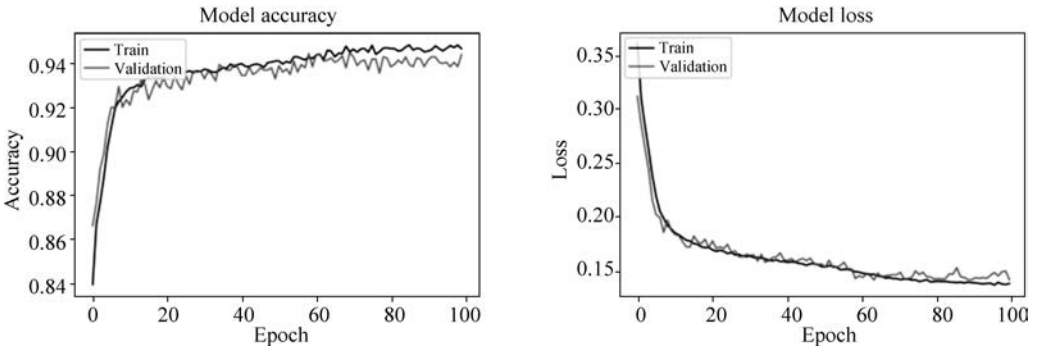

图 3.11 模型的正确率和损失对于一段时期的训练过程的函数

(8) 使用训练好的模型来预测输出类的前 10 个输入数据实例(X.iloc[0:10,:])。

```
y_predicted = model.predict(X.iloc[0:10,:])
```

可以使用以下代码块输出预测类。

```
 # print the predicted classes
print("Predicted probability for each of the "\
print(y_predicted)
print("Predicted class label for each of the examples: "),
print(np.round(y_predicted))
```

期望输出如下。

```
Predicted probability for each of the examples belonging to class 1:
[[0.00354007]
 [0.8302744 ]
 [0.00316998]
 [0.95335543]
 [0.99479216]
 [0.00334176]
 [0.43222323]
 [0.00391936]
 [0.00332899]
 [0.99759173]
Predicted class label for each of the examples:
[[0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]
```

```
[0.]
[0.]
[1.]]
```

使用训练好的模型来预测输出前 10 棵树的物种。第二、四、五和十棵树被预测为 1 类的树种，落叶类。

说明：源代码网址为 https://packt.live/2YX3fxX，在线运行代码网址为 https://packt.live/38pztVR。

可以通过向网络添加更多隐藏层来扩展这些步骤。在添加输出层之前可向模型添加任意数量的层。但是，输入维度参数仅为了提供给第一层，因为下一层的输入维度是由第一层决定的。现在已经学习了如何用 Keras 实现神经网络，可以通过一个用于分类的神经网络来进一步训练。

实践 3.01 构建单层神经网络进行二进制分类

在本实践中，使用 Keras 顺序模型来构建二进制分类器。提供的模拟数据集表示生产的飞机螺旋桨的测试结果。我们的目标变量是手动检查螺旋桨得到的结果："通过"（表示为值 1）或"失败"（表示为值 0）。

我们的目标是将测试结果分类为"通过"或"失败"类别，以匹配手动检查。可使用不同架构的模型，并观察不同模型性能的可视化。这能更好地了解一个处理单元到一层处理单元是如何改变模型的灵活性和性能的。

假设该数据集包含的两个特征表示 3000 多个飞机螺旋桨在两种不同测试中的测试结果（这两个特征被归一化为均值为零）。输出是螺旋桨通过测试的可能性，1 表示通过，0 表示失败。该公司希望减少对耗时、容易出错的手动螺旋桨检查的依赖，转移资源以开发自动化的测试方法更快地评估螺旋桨。因此，我们的目标是建立一个模型，在给出两种测试结果时，可以预测飞机螺旋桨是否可以通过人工检查。本实践先构建一个逻辑回归模型，再构建一个具有三个单元的单层神经网络，最后构建一个具有六个单元的单层神经网络来执行分类。按照以下步骤完成此实践。

（1）导入必需的包。

```
# import required packages from Keras
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
import pandas as pd
from tensorflow import random
from sklearn.model_selection import train_test_split
# import required packages for plotting
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
import matplotlib.patches as mpatches
# import the function for plotting decision boundary
from utils import plot_decision_boundary
```

说明：需要从 GitHub 下载 utils.py 文件并将其保存到实践文件夹中，以便 utils 导入

语句正常工作，网址为 https://packt.live/31EumPY。

(2) 设置一个生成随机参数的种子，以便结果能够重复产生。

```
"""
define a seed for random number generator so the result will be
reproducible
"""
seed = 1
```

说明：上面代码片段中显示的三引号("""")是多行代码注释的起点和终点。注释用于解释代码中的特定逻辑。

(3) 使用 Pandas 库中的 read_csv 函数加载数据集。使用 feats. shape、target. shape 和 feats. shape[0]输出训练数据集中的 X 和 Y 大小以及示例数。

```
feats = pd.read_csv('outlier_feats.csv')
target = pd.read_csv('outlier_target.csv')
print("X size = ", feats.shape)
print("Y size = ", target.shape)
print("Number of examples = ", feats.shape[0])
```

(4) 使用下列代码对数据集画图。

```
plt.scatter(feats[:,0], feats[:,1], \
            s=40, c=Y, cmap=plt.cm.Spectral)
```

(5) 在 Keras 中将逻辑回归模型实现为顺序模型。记住，二元分类的激活函数是 sigmoid。

(6) 设置 optimizer='sgd'，loss='binary_crossentropy'，batch_size = 5，epochs = 100 和 shuffle=False 训练模型。设置 verbose=1 和 validation_split=0.2 观察每次迭代中的损失值。

(7) 使用以下代码绘制训练模型的决策边界。

```
plot_decision_boundary(lambda x: model.predict(x), \
                       X_train, y_train)
```

(8) 实现一个在隐藏层中具有三个节点、200 个轮次和 ReLU 激活函数的单层神经网络。重点是，输出层的激活函数仍然需要是 sigmoid，因为该问题是一个二元分类问题。如果为输出层选择 ReLU 或不选择激活函数的话，模型将不会产生可以解释为“类标签”的输出。使用 verbose=1 训练模型并观察每次迭代中的损失。模型训练完成后，绘制决策边界并评估测试数据集的损失和正确率。

(9) 对大小为 6 和 400 个轮次的隐藏层重复步骤(8)，并比较最终损失和决策边界图。

(10) 使用隐藏层的 tanh 激活函数重复步骤(8)和(9)，并将结果与使用 ReLU 激活函数的模型进行比较。哪种激活函数更适合解决这个问题？

说明：此实践的答案见附录 A 的实践 3.01 构建单层神经网络进行二进制分类。

本实践能够解释为何在模型的一个层中堆叠多个处理单元比单个处理单元更强大。这就是神经网络模型如此强大的基本原因。增加层中的单元数量会增加模型的灵活性，这意味着可以更精确地估计非线性分离决策边界。

但是，具有更多处理单元的模型需要更长的时间来学习，需要更多的轮次进行训练，并且可能会过拟合训练数据。因此，神经网络模型的计算量非常大。而且与使用 ReLU 激活函数

相比使用 tanh 激活函数会导致训练过程变慢。

本节创建了各种模型并根据数据对其进行了训练。可以观察到，通过对训练数据进行评估，有些模型比其他模型表现更好。下一节将了解一些可用于模型评估的替代方法，这些方法提供了无偏见的评估。

3.3 模型评估

本节继续学习多层/深度神经网络，同时还将学习到评估模型性能的一些技术。通过上面的学习您可能已经意识到，构建深度神经网络时还需要多次地选择超参数。

一些应用深度学习的挑战包括如何正确地设置隐藏层层数、每个隐藏层中的单元数量、每层使用的激活函数类型以及优化器和损失的类型。做这些决定前先进行模型评估是必不可少的。通过模型评估，可以判断一个特定的深度架构或一组特定的超参数在特定数据集上是否运行良好，从而决定是否进行更改。

此外，本节还将学习过拟合和欠拟合。在解决特定问题时需要寻找合适的深度神经网络并尽可能提高其性能，所以了解过拟合和欠拟合的概念以及它们何时会发生是必要的。

3.3.1 用 Keras 进行模型评估

在之前的实践中，通过预测每个可能输入值的输出来绘制模型的决策边界。因为处理的是二维输入数据，所以模型的性能是可以可视化的。但输入空间中的特征或测量值的数量总是超过两个，因此不适合通过 2D 绘图进行可视化。一种判断模型在特定数据集上的表现的方法是在预测示例输出时计算总体损失，可以通过 Keras 中的 evaluate()方法来完成。该方法接收一组输入(X)输出(y)，计算并返回模型在输入 X 上的整体损失。

例如，考虑构建一个神经网络。该网络具有两个维度分别为 8 和 4 的隐藏层以执行二分类或二元分类。可用存储在 x，y 数组中的数据点和其对应的类标签来构建和训练上述模型。

```
model = Sequential()
model.add(Dense(8, activation='tanh', input_dim=2))
model.add(Dense(4, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='sgd', loss='binary_crossentropy')
model.fit(X, y, epochs=epochs, batch_size=batch_size)
```

可通过计算整个数据集的损失来评估模型的整体性能，而不是使用 model.predict()来预测给定输入集的输出。

```
model.evaluate(X, y, batch_size=None, verbose=0)
```

如果评估其他指标，例如正确率，则在为模型定义 compile()方法时，调用 evaluate()方法可同时返回损失和这些指标。例如在下面的代码中的 compile() 参数中添加这些指标，调用 evaluate()方法将返回训练模型在整个数据集上的整体损失和整体正确率。

```
model.compile(optimizer='sgd', loss='binary_crossentropy', \
              metrics=['accuracy'])
model.evaluate(X, y, batch_size=None, verbose=0)
```

说明：可以在 https://keras.io/metrics/中查看 Keras 的所有可选 metrics 参数。

下一节将学习如何拆分数据集为训练集和测试集。然后就像在第 1 章中所做的那样，对单独的数据进行训练和评估，这样可以无偏差地评估模型性能。

3.3.2 将数据集拆分为训练集和测试集

一般来说，在用于训练模型的同一数据集上评估模型是一种方法性的错误。由于训练集已被模型训练过以减少错误，用训练集进行评估将导致模型性能的评估出现偏差。换句话说，训练集的错误率总是低估了模型的真实错误率。

构建机器学习模型的目的不是为了在训练数据集上取得良好的性能，而是为了让模型在未见过的示例上依然表现良好。这就是为什么要用非训练数据集来评估模型的性能。

实现此目的的一种方法是将可用数据集分成训练集和测试集。训练集用于训练模型，而测试集用于模型评估。训练集的作用是为模型提供足够的示例使其能够学习数据之间的关系和规律；而测试集的作用是作为没见过的例子对模型性能进行无偏估计。机器学习中通常对相对较小的数据测试集执行 70%：30%或 80%：20%比例的拆分。在处理具有数百万个示例且目标是训练大型深度神经网络的数据集时，可以使用 98%：2%或 99%：1%的比例进行训练-测试拆分。

图 3.12 显示了将数据集划分为训练集和测试集。注意，训练集和测试集之间没有重叠。

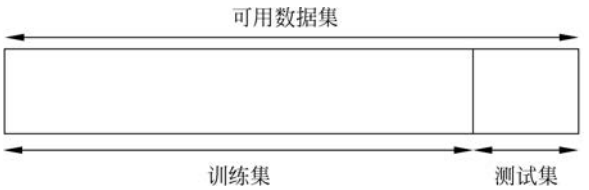

图 3.12 将数据集拆分为训练集和测试集

可以使用 scikit-learn 的 train_test_split 函数轻松地对数据集执行拆分。以下代码执行了 70%：30%比例拆分。

```
from sklearn.model_selection import train_test_split
X_train, X_test, \
y_train, y_test = train_test_split(X, y, test_size=0.3, \
                                   random_state=None)
```

test_size 参数代表测试集中保留数据集的比例，因此它应该在 0 和 1 之间。通过给 random_state 指定 int 参数，可以选择生成随机分割训练集和测试集的种子。

将数据拆分为训练集和测试集后，更改上节代码，仅提供训练集作为 fit()的参数。

```
model = Sequential()
model.add(Dense(8, activation='tanh', input_dim=2))
model.add(Dense(4, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='sgd', \
              loss='binary_crossentropy')
model.fit(X_train, y_train, epochs=epochs, \
          batch_size=batch_size)
```

分别计算训练集和测试集上的模型错误率。

```
model.evaluate(X_train, y_train, batch_size=None, \
               verbose=0)
model.evaluate(X_test, y_test, batch_size=None, \
               verbose=0)
```

拆分的另一种方法是给 Keras 的 fit() 方法加入 validation_split 参数。例如，将上节代码中的 model.fit(X, y) 更改为 model.fit(X, y, validation_split=0.3)。模型会将最后 30%的数据示例保留在一个单独的测试集中。它只会在其他 70%的样本上训练模型，并在每个轮次结束时在训练集和测试集上评估模型。这样做可以观察到训练集错误率和随着训练进行的测试集错误率的变化。

对模型进行公正评估的原因是希望可以看到模型有哪些改进的空间。由于神经网络要学习的参数太多也可以学习复杂的函数，因此通常会最终过拟合训练数据并学习训练数据中的噪声。这会导致模型无法在新的、未知的数据上表现良好。下一节将详细探讨这些概念。

3.3.3 过拟合和欠拟合

本节将学习在构建机器学习模型拟合数据集时可能面临的两个问题——过拟合和欠拟合，这类似于模型的偏差和方差的概念。

一方面，如果模型不够灵活，无法学习数据集中数据的关系和模式，模型将会得到一个很高的训练集错误率。可以称这样的模型为高偏差模型。另一方面，如果模型对于给定的数据集过于灵活，它将学习训练数据中的噪声，以及数据中的关系和模式，这样的系统将导致测试误差与训练误差相比大幅增加。之前提到过，测试误差总是比训练误差略高。

但是，测试集错误率和训练集错误率之间差距很大，代表系统具有一个高的方差。在数据分析中，这些情况（高偏差和高方差）都不是可取的。找到同时具有尽可能低的偏差和方差的模型是构建模型的目标。

例如，一个数据集代表了两种蝴蝶目击的归一化位置，如图 3.13 所示。我们的目标是找到一个模型，当给定这两种蝴蝶的目击地点时，能够将它们分开，显然，这两个类别之间的分界线不是线性的。因此，如果选择一个简单的模型，如逻辑回归（具有一个大小的隐藏层的神经网络）来对这个数据集进行分类，将得到一个两类之间的线性分离线/决策边界，但这无法捕捉到数据集的真实规律。

图 3.14 说明了此类模型的决策边界。通过评估这个模型可以观察到，训练错误率很高，测试错误率略高于训练错误率。具有高训练错误率表示模型具有高偏差，而训练错误和测试错误之间的微小差异表示模型具有低方差。这是一个明显的欠拟合情况——模型无法拟合两个类之间真正的分割线。

如果通过添加更多层来增加神经网络的灵活性并增加每层中的单元数量，可以训练更好的模型并成功捕捉非线性的决策边界，如图 3.15 所示。这是一个训练错误率低、测试误率低的模型（同样，测试错误率略高于训练错误率）。具有低训练错误率和测试错误率，且两者之间差异较小表明模型具有低偏差和低方差。模型具有低偏差和低方差代表给定数据集正确拟合。

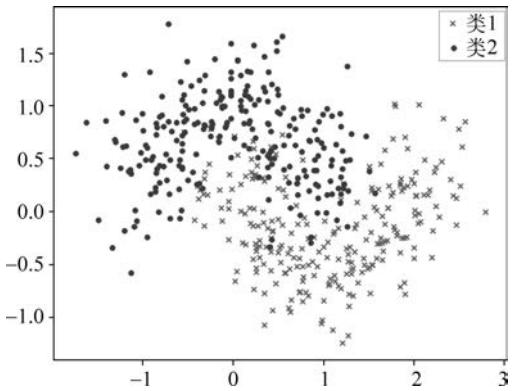

图 3.13 两个不同类别的二维数据点

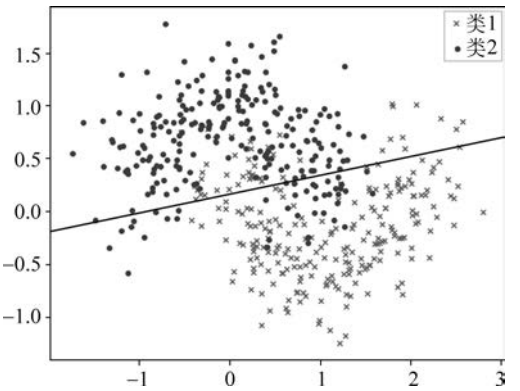

图 3.14 欠拟合

但是,如果进一步增加神经网络的灵活性会发生什么?模型增加太多的灵活性后不仅会训练数据中的模式和关系,还会学习其中的噪声。换句话说,模型将更拟合每个单独的训练示例,而不是拟合它们的整体趋势和关系,图 3.16 显示了这种情况的系统。评估这个模型会得到非常低的训练错误率和很高的测试错误率(训练错误率和测试错误率之间的差异很大)。这是一个低偏差高方差的模型,这种情况称为过拟合。

在训练集和测试集上评估模型并比较它们的错误率,可以提供当前模型是否适合给定数据集的有价值信息。此外,在当前模型无法正确拟合数据集的情况下,也可以通过判定数据是过拟合还是欠拟合来相应地更改模型以找到正确的模型。例如,如果模型欠拟合,可以加深网络。如果模型过拟合,可以通过缩小网络或为其提供更多训练数据来减少过拟合。在实践中,有很多方法可以用来防止欠拟合或过拟合,将在下一节中探讨其中的一种。

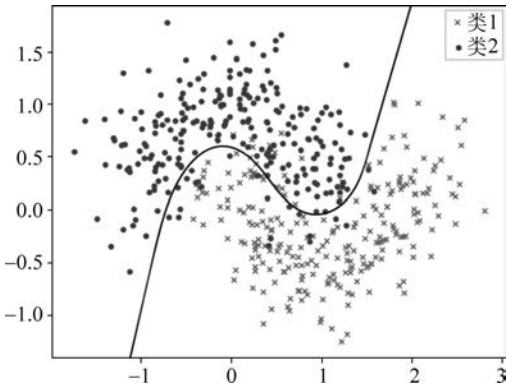

图 3.15 正确拟合

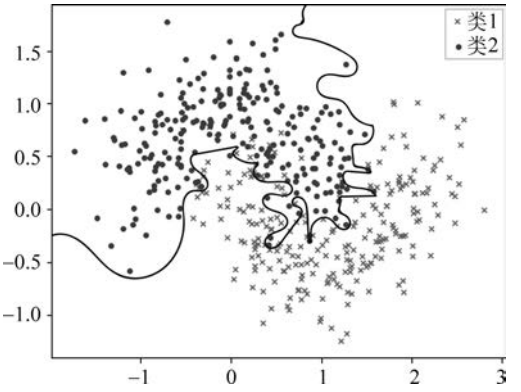

图 3.16 过拟合

3.3.4 早停

有时,即使模型的灵活性适合数据集但过拟合或欠拟合仍然会发生,这是因为模型训练的迭代太多或者太少。当使用迭代优化器(例如梯度下降)进行优化时,优化器会尝试在每次迭代中越来越好地拟合训练数据。因此,如果在学习数据后不断地更新参数,模型将开始拟合各

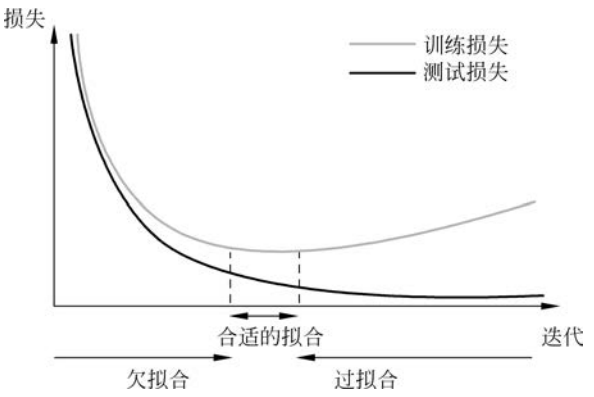

图 3.17　训练模型时的训练错误率和测试错误率图

个数据示例。

通过观察每次迭代的训练和测试错误率，可以确定网络从何时开始过拟合训练数据并在此之前终止训练过程。与欠拟合和过拟合相关的区域已在图 3.17 中标出。训练模型的正确迭代次数可以从测试错误率具有最低值的区域确定。将该区域标记为正确拟合区域，可以看出该区域中的训练错误率和测试错误率都很低。

在使用 Keras 进行训练时，可以轻松地存储每个轮次中的训练损失和测试损失的值。为此，需要在为模型定义 fit()方法时提供测试集作为 validation_data 参数，并将其存储在 history 字典中。

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test))
```

稍后绘制存储在 history 记录中的值以找到正确的数量迭代来训练模型。

```
import matplotlib.pyplot as plt
import matplotlib

# plot training loss
plt.plot(history.history['loss'])
# plot test loss
plt.plot(history.history['val_loss'])
```

一般来说，由于深度神经网络是高度灵活的模型，故发生过拟合的可能性非常高。有一整组技术，称为正则化技术，已被开发用于减少机器学习模型中的过拟合，特别是深度神经网络。第 5 章将更深入地了解有关这些技术的信息。在下一个实践中，把理解付诸实践，并尝试找到训练的最佳轮次数，以防止过拟合。

实践 3.02　神经网络与高级纤维化诊断

本实践将使用真实数据集的年龄、性别和 BMI 等测量值来预测患者是否患有晚期纤维化。该数据集包含 1385 名接受丙型肝炎治疗的患者信息。每个患者有 28 个不同的属性，以及一个只能取两个值的类别标签：1 表示患有晚期纤维化；0 表示没有晚期纤维化的迹象。这是一个输入维度等于 28 的二元/二分类问题，图 3.18 是用于诊断丙型肝炎的二元分类器。

图 3.18　诊断糖尿病的二元分类器

本实践将通过不同的深度神经网络来执行此分类，并绘制训练错误率和测试错误率的趋势确定最终分类器需要训练多少轮数。

说明：此实践中使用的数据集网址为 https://packt.live/39pOUMT。

按照下列步骤完成实践。

(1) 从 GitHub 的 Chapter03 文件夹的数据子文件夹加载数据集，导入所有必要的依

赖项。

```
X = pd.read_csv('../data/HCV_feats.csv')
y = pd.read_csv('../data/HCV_target.csv')
```

(2) 输出数据集中的示例数量、可用的特征数量以及类标签的可能值。

(3) 使用 scikit-learn 中的 StandardScalar 函数缩放数据。预处理并将数据集以 80：20 的比例拆分为训练集和测试集，然后输出拆分后每个集合中的示例数。

(4) 实现一个具有大小为 3 的隐藏层和一个 tanh 激活函数的浅层神经网络执行分类。编译模型以下超参数：optimizer='sgd'，loss='binary_crossentropy'，metrics=['accuracy']。

(5) 使用以下超参数拟合模型，并在训练过程中存储训练错误率和测试错误率的值：batch_size=20，epochs=100，validation_split=0.1，shuffle=False。

(6) 绘制每个训练时期的训练错误率和测试错误率。使用该图来确定网络在哪个时期开始过拟合数据集。此外，输出在训练集和测试集上达到的最佳准确度值，以及在测试数据集上评估的损失和正确率。

(7) 对具有两个隐藏层(大小为 4 的第一层和大小为 3 的第二层)和两层 tanh 激活函数的深度神经网络重复步骤(4)和步骤(5)，以执行分类。

说明：此实践的答案见附录 A 的实践 3.02 神经网络与高级纤维化诊断。

注意与测试集相比，这两个模型都在训练集或验证集上有更好的正确率，并且在训练大量轮数时训练错误率不断下降。然而如果验证错误率在训练过程中下降到一定值后开始增加，表明训练数据已经过拟合。最大验证正确率对应于图上验证损失最低的点，并且表示模型稍后在独立示例上的真正表现。

从结果可以看出，与两层模型相比具有一层隐藏层的模型能够达到较低的验证和测试错误率。由此可以得出，该模型是上述特定问题的最佳选择。具有一个隐藏层的模型显示出大量的偏差，这表明训练和验证错误之间的差距很大。两者仍在下降，表明该模型可以训练更多的轮数。最后，从图中可以确定在验证错误率开始增加的区域停止训练，可以防止模型过拟合数据点。

3.4 总结

本章扩展了深度学习的知识，从理解常见的表示和术语到通过训练和活动在实践中实现它们。我们一起学习了神经网络中前向传播的工作原理以及如何用于预测输出，损失函数，模型性能的衡量标准，以及如何使用反向传播来计算损失函数相对于模型参数的导数。

同时了解了梯度下降，使用反向传播计算的梯度来逐步更新模型参数。除了基本理论和概念之外，还使用 Keras 实现和训练了浅层和深层神经网络，并利用它们对给定输入预测输出。

为了适当地评估模型，将数据集拆分为训练集和测试集，作为改进网络评估的替代方法，并了解了在训练示例上评估模型可能会产生误导的原因。这有助于进一步了解训练模型时可能发生的过拟合和欠拟合。最后，利用训练错误率和测试错误率来检测网络中的过拟合和欠拟合，并实施早停以减少网络中的过拟合。

下一章将介绍带有 scikit-learn 的 Keras 包，以及如何通过使用交叉验证等重采样方法进一步改进模型评估，从而学习如何为深度神经网络找到最佳的超参数集。