CHAPTER 4

深度学习概论

本章将向您介绍深度学习，包括多层感知器(MLP)，卷积神经网络(CNN)。其他的深度学习架构，如循环神经网络(RNN)和长短期记忆网络(LSTM)。

本章中大部分材料是描述性内容，以及一些基于keras的代码示例。这一章是对一系列不同主题的粗略介绍。

如果你不熟悉深度学习，对于本章中的一些内容你还需要额外的学习才能理解。

本章的第一部分简要讨论了深度学习、深度学习可以解决的问题以及未来的挑战。第二部分简要介绍了神经网络的构建。人工神经网络（ANN)，多层感知器(MLP)，循环神经网络(RNN)，长短期记忆网络(LSTM)，变分编码器（VAE）都是基于多个感知器多层网络的，以及其他的附加步骤。

本章的第三部分介绍了卷积神经网络（CNN），然后给出了一个使用MNIST数据及训练基于keras的CNN的示例。如果你阅读了第五章激活函数的部分，这部分代码会更有帮助。

Keras 及异或函数

异或函数是一个广为人知的函数，它在平面上是线性不可分的。异或函数的真值表是直观的：给定两个二进制的输入，如果最多有一个输入是1，则输出为1；否则输出为0。可以定义一个XOR函数表示异或函数，它有两个二进制的输入，则输出如下：

```
XOR(0,0) = 0
XOR(1,0) = 1
XOR(0,1) = 1
XOR(1,1) = 0
```

我们可以将输出值视为与输入值关联的标签。具体来说，点（0,0）和（1,1）属于0类，而点（0,1）和点（1,0）属于1类。在平面上画这些点，你将得到一个单位正方形的四个顶点，其左下角的顶点是原点。才外，每对对角线元素都属于同一类，并且不能用欧几里得平面上的直线将类0中的点与类1中的点分开。因此，异或函数在平面上线性不可分。

例4.1 展示了tf2_keras_xor.py，说明了如何创建一个基于keras的神经网络来训练异或函数。

Listing 4.1: tf2_keras_xor.py

```
import tensorflow as tf
import numpy as np

# Logical XOR operator and "truth" values:
x = np.array([[0., 0.],[0., 1.],[1., 0.],[1.,
   1.]])
y = np.array([[0.], [1.], [1.], [0.]])

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(2, input_dim=2,
   activation='relu'))
model.add(tf.keras.layers.Dense(1))
```

```
print("compiling model...")
model.compile(loss='mean_squared_error',
   optimizer='adam')
print("fitting model...")
model.fit(x,y,verbose=0,epochs=1000)
pred = model.predict(x)

# Test final prediction
print("Testing XOR operator")
p1 = np.array([[0., 0.]])
p2 = np.array([[0., 1.]])
p3 = np.array([[1., 0.]])
p4 = np.array([[1., 1.]])

print(p1,":", model.predict(p1))
print(p2,":", model.predict(p2))
print(p3,":", model.predict(p3))
print(p4,":", model.predict(p4))
```

例4.1初始化了Numpy数组x，为四对0,1组成的数组，numpy数组y为异或操作的结果。

下一部分定义了一个基于keras的模型，其中包含两个全连接层。之后对模型进行编译训练，然后在变量pred中填充基于训练模型的预测。

下一段代码初始化点p1,p2,p2,p4,然后显示这些点的预测值。4.1的输出如下：

```
compiling model...
fitting model...
Testing XOR operator
[[0. 0.]] : [[0.36438465]]
[[0. 1.]] : [[1.0067574]]
[[1. 0.]] : [[0.36437267]]
[[1. 1.]] : [[0.15084022]]
```

测试不同的时间值，看看这对预测的影响。使用例4.1的代码作为其他逻辑函数的模板。

NOR函数:

```
y = np.array([[1.], [0.], [0.], [1.]])
```

OR函数:

```
y = np.array([[0.], [1.], [1.], [1.]])
```

XOR函数:

```
y = np.array([[0.], [1.], [1.], [0.]])
```

ANDR函数：

```
y = np.array([[0.], [0.], [0.], [1.]])
```

```
mnist = tf.keras.datasets.mnist
```

什么是深度学习？

深度学习是机器学习的一个子集，主要研究神经网络和训练神经网络的算法。深度学习包括很多类型的神经网络，如CNN，RNN，LSTM，GRU、VAE、GAN等。一个深度学习模型需要一个神经网络中至少两个隐藏层（非常深入的学习至少要有十个隐藏层）。

从高层次的角度来看，有监督学习的深度学习包括定义模型（神经网络）以及

- 估算数据点
- 计算每个估算的损失或误差
- 通过梯度下降减小误差

在第三章中，你在机器学习背景下学习了线性回归：

```
m = tf.Variable(0.)
```

```
b = tf.Variable(0.)
```

训练过程集中在下面的等式：

```
y = m*x + b
```

我们要计算因变量y，给定独立变量x的值。在这种情况下，计算由以下python函数执行：

```
def predict(x):
  y = m*x + b
  return y
```

损失就是当前估计的错误，可以通过以下函数确定MSE的值：

```
def squared_error(y_pred, y_actual):
  return tf.reduce_mean(tf.square(y_pred-y_actual))
```

我们还需要为训练数据和测试相关数据初始化变量通常分隔的比例为：80/20或75/25.训练过程按以下方式调用python函数：

```
loss = squared_error(predict(x_train), y_train)
print("Loss:", loss.numpy())
```

什么是超参数？

深度学习涉及到超参数，像是旋钮和刻度盘，它们的值在实际训练过程之前已经被初始化。例如：隐藏层的数量和隐藏层神经元的数量就是超参数的例子。你将在深度学习模型中遇到许多超参数，其中一些参数列在下面：

- 隐藏层数量
- 隐藏层神经元数量
- 权重初始化
- 激活函数。
- 成本函数
- 优化器
- 学习率
- 丢弃率

上面列出的前三个超参数是神经网络必须的。前向传播需要第四个超参数。之后三个（成本函数、优化器、学习率）是在监督学习任务期间执行反向传播需要的。这个步骤计算已选数组，用于更新神经网络中的权值，提高了神经网络的精度。最后一个超参数在你需要减少模型的过拟合时使用。一般来说，成本函数是所有超参数中最复杂的。

在反向传播过程中，可能会出现梯度消失的情况，之后一些权重就不会再更新，在这种情况下，神经网络本质上是懒惰的。另一个考虑因素：决定一个局部最小值是否足够好，并且是否愿花费时间和精力来找到这个绝对的最小值。

深度学习体系结构

如前所述，深度学习支持各种体系结构，包括MLP，CNN，RNN，LSTM等。尽管在结构可以解决的问题存在重叠，但是还是有差异的。随着从MLP到LSTM的发展，体系结构变得更加复杂。有时这些结构的组合非常适合解决任务。例如，捕获视频并进行预测通常使用CNN（用于处理视频序列中的每个输入图像）和LSTM来预测视频流中对象的位置。

此外，用于NLP的神经网络可以包含一个或多个CNN，RNN，LSTM，双向LSTM。特别是，强化学习与这些结构的结合为深度强化学习。

虽然MLP已经流行了很长一段时间，但是它们有两个缺点 ：它们不能拓展到计算机视觉领域，它们训练起来也比较困难。另一方面，CNN不要求相邻层完全连接。CNN的另一个优点是平移不变性，这意味着图像（比如数字、猫、狗等）不管出现在图片的哪个位置，都能够被识别。

深度学习可以解决的问题

我们知道，反向传播涉及到更新连续层之间的边的权重，这是以从输出层到输入层的方向执行的。更新包括链规则（计算导数的规则）和参数和梯度值的算数乘积。会出现两种反常结果：项的乘积接近零（称为梯度消失问题）或项的乘积变得任意大（称为梯度爆炸问题）。这些问题是由sigmoid激活函数引起的。

深度学习可以通过LSTM缓解这两个问题。深度学习模型通常用ReLU激活函数代替sigmoid激活函数。ReLU是一个非常简单的连续函数，除了原点，它在任何地方都是可微的（y轴右侧的值为1，y轴左侧的值为-1）。因此，有必要进行一些调整，使模型在原点处运行良好。

深度学习中的挑战

虽然深度学习是强大的，并在很多领域取得了令人印象深刻的成果。但仍有一些重要的挑战正在探索中，其中包括：

- 算法中的偏置
- 易受敌对攻击
- 概括能力有限
- 缺乏可解释性
- 为相关关系而不是因果关系

算法可能包括无意的偏置，即使消除了偏置，数据中心也可能存在偏差。例如：一个神经网络被训练在一个白人男性和女性照片的数据集上，训练的结果决定男性是医生，女性是家庭主妇，这样做的可能性更大。原因很简单：数据集几乎只描述了男性和女性的这两个角色。

深度学习侧重于在数据集中发现一些模式，而归纳这些结果则是更困难的工作。有一些任务尝试为神经网络的结果提供解释，但是这项工作仍然处于初级阶段。深度学习可以找到模式并确定相关性，但是无法确定因果关系。

接下来我们要了解深度学习的重要组成部分：感知器。

什么是感知器

第三章讨论的是神经元输出的线性回归模型，第四章讨论的是一个神经元输出层。DNN网络不仅可以解决多层的问题，而且至少可以解决两层的问题。事实上，分类问题模型的输出层由一组概率组成，它们的和为1.

图4.1显示了一个具有数字权重传入边的感知器。

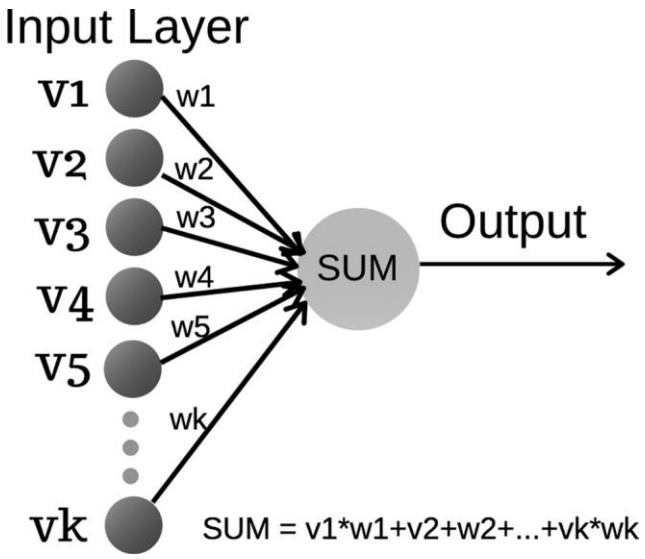


FIGURE 4.1 感知器的一个例子

下一节将深入研究嘎吱器的细节，以及它们如何构成MLP的主体部分。

感知器函数的定义

感知器涉及一个函数f(x)如下所示：

```
f(x) = 1 if w*x + b > 0 (otherwise f(x) = 0)
```

w表示权重向量，x是输入向量，b是偏置向量。乘积w*x是向量w和x的内积，激活感知器是一个全有或全无的决定（例如：灯是开的还是关的，没有中间状态）。

请注意，函数f(x)检查线性项w*x+b的值，这也是在逻辑回归的sigmoid函数中指定的。在计算sigmoid值时会出现相同的项，如下所示：

```
1/[1 + e^(w*x+b)]
```

给定w*x+b的值，前面的表达式可以生成一个数值。然而，在一般情况下，W是权重矩阵，x和b是向量。

下一节讨论人工神经网络，之后再讨论MLP。

感知器的详细描述

神经元本质上是神经网络的组成部分。一般来说，每个神经元接受多个输入，每个输入来自神经网络中属于前一层的神经元。计算输入的加权平均并分配给神经元。

具体地说，假设一个神经元N’接受权重在集合{w1,w2,w3,…,wn}，其中这些数字指定连接到神经元N’的边的全中国。由于前向传播为从左到右的数据流，这意味着边缘的左端点连接到神经元{n1,n2,…,nk}这些边的右端点是N’。加权和计算如下：

```
x1*w1 + x2*w2 + . . . + xn*wn
```

在计算出加权和之后，它被输入到计算第二个值的激活函数中。这一步是人工神经网络所必需的，在本章后面会解释。对给定层中的每个神经元重复计算加权和的过程，然后对神经网络下一层的神经元重复相同的过程。

整个过程称为正向传播，由反向误差传播完成。在误差反向传播阶段，为整个神经网络计算新的权值。对于每个数据点（例如，CSV文件中的每一行数据）重复前向属性和后向属性的计算。目标是完成这一训练过程，使最终化的神经网络（模型）准确地表示数据集中的数据，并能准确地预测测试数据的值。当然，神经网络的准确性取决于所讨论的数据集，有时准确度可以高于99%。

人工神经网络

人工神经网络由输入层、输出层和一个或多个隐藏层组成。对于ANN中的每一对相邻层，左层的神经元与右层的神经元通过一条具有数值权重的边相连。如果左侧层的所有神经元都与右侧所有神经元相连，则成为MLP。

ANN中的感知器是“无状态的”：它们不保留关于先前处理的数据的任何信息。此外，ANN不包含循环。相比之下，RNN和LSTM是保留状态的，而且它们确实具有类似循环的行为，后面会介绍。

ANN也可以看做一组连续的二分图，其中的数据从输入层流向输出层。

你可以把ANN的结构看做下面的超参数的组合:

- 隐藏层的数量
- 每个隐藏层的神经元数量
- 连接神经元的边的初始化权重
- 激活函数
- 损失函数
- 优化器（与损失函数一起用）
- 学习率（一个较小的值）
- 丢失率（可选）

Figure4.2 展示了ANN的内容。

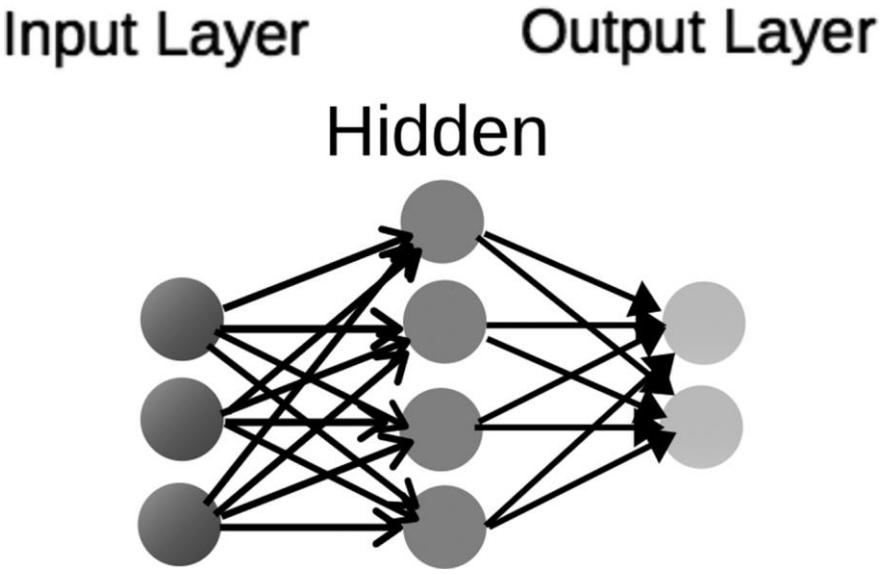


FIGURE 4.2ANN的一个例子.

上面的ANN的输出层包含不止一个神经元，它是一个分类任务的模型。

初始化模型的超参数

上一节中列出的超参数中的前三个是初始化神经网络必须的。隐藏层是中间计算层，每一层都由神经元组成。每队相邻层之间的边数是你自己决定的。

每对相邻层（包括输入输出层）中连接神经元的边具有数值权重。这些权重的初始值通常介于0和1之间的小的随机数。相邻层之间的连接可能会影响模型的复杂性。训练过程的目的是微调边缘权重，以提高模型的精度。

人工神经网络不一定是完全联通的，就是说相邻层神经元之间的某些边可能是缺失的。相比之下：CNN这样的神经网络共享边和它们的权重可以使得计算上更加可行。可以利用Keras `tf.keras.layers.Dense()`类解决完全连接的两个相邻层的任务。正如后面讨论的，MLP是完全连接的，这可以大大增加神经网络的训练时间。

激活超参数

第四个参数是应用于每对连接层之间的权重的激活函数。具有许多层的神经网络通常包含不同的激活函数。例如：CNN在特征映射上使用ReLU激活函数（通过对图像应用过滤器创建），而倒数第二层通过softmax函数（是sigmoid函数的泛化）连接到输出层。

损失函数超参数

第五个第六个第七个超参数是从输出层开始向输入层从右向左移动的反向误差传播。这些超参数完成了机器学习框架的工作：用于计算神经网络中权值的更新。

损失函数是多维欧式空间中的函数。例如：MSE损失函数是具有全局最小值的碗装损失函数。一般来说，目标是最小化MSE函数以最小化损失，同时也最大化了模型的精度。但是有时需要的是局部最小值，而不是全局最小值，所以需要进行权衡。对于较大的数据集，损失函数往往比较复杂。还有一个损失函数是交叉熵函数，它涉及到最大化似然函数。

优化器超参数

优化器是一种结合损失函数选择的算法，其目的是在训练阶段收敛到成本函数的最小值。在训练过程中，不同的优化器对计算新的近似值的方式作出了不同的假设。一些优化器只涉及最新的近似值，而其他优化器则考虑使用了先前近似值的滚动平均值。

有几个著名的优化器，包括SGD、RMSprop、Adagrad、Adadelta和Adam。在线查看有关这些优化器的优势和权衡的详细信息。

学习率超参数

学习率是一个很小的数字，通常在0.001到0.05之间，这会影响添加到边的当前权重的数量大小，以便使用这些更新的权重训练模型。学习率有一种节流效应。如果该值太大，新方法可能会超出最佳点；如果太小，则训练时间会显著增加。举个例子，假设你坐在一架客机上，离机场100英里远。当你接近机场时，飞机的速度会降低，这相当于降低神经网络的学习速度。

丢弃率超参数

丢弃率是第八个超参数，它是一个介于0和1之间的十进制值，通常在0.2到0.5之间。将这个十进制值乘以100，以确定在训练过程中每次向前传播时要忽略的随机选择的neu-rons的百分比。例如，如果丢失率为0.2，则在前向传播的每个步骤中随机选择20%的神经元忽略不计。当神经网络处理一个新的数据点时，随机选择一组不同的神经元。注意神经元并没有从神经网络中移除：它们仍然存在，在前向传播过程中忽略它们会导致神经网络变薄。在TF 2中`tf.keras.layers.Dropout`类执行细化神经网络的任务

什么是反向误差传播

ANN通常是以从左到右的方式传播的，其中最左边的层是输入层。每个层的输出会成为下一层的输入。术语向前传播是指向输入层提供值并通过隐藏层向输出层前进。输出层包含正向通过模型的结果（模型估计的数值）。

这里有一个关键点：反向误差传播涉及计算用于更新神经网络中边缘的权重的数字。更新过程通过损失函数（以及优化器和学习速率）来执行，从输出层开始，然后以从右向左的方式移动，以更新连续的层之间的边的权重。这个过程训练神经网络，包括减少输出和真实值之间的误差。这样的一次处理为一轮，神经网络的训练通常需要多轮。

上一段没有解释损失函数是什么以及是如何选择的：这是因为损失函数、优化器和学习率是前面几节讨论的超参数。两个常用的损失函数是MSE和交叉熵；一个常用的优化器是Adam优化器（以及SGD和RMSprop等）；学习率的一个常用值是0.01。

什么是多层感知器 (MLP)?

多层感知器（MLP）是由至少三层结点组成的前馈人工神经网络：输入层、隐藏层和输出层。一个MLP是完全联通的：给定一对相邻层，左层的每个节点都连接到右层的每个节点。除了输入层的结点外，每个节点都是一个神经元，每一层神经元都包含一个非线性激活函数。此外，MLP使用一种称为反向误差传播的技术进行训练，对于CNN（卷积神经网络）也是这样。

图4.3 展示了具有两个隐藏层的MLP的内容。

需要记住的一点是：MLP的非线性激活函数将MLP与线性感知器区分开。实际上，MLP可以处理不可线性分离的数据。例如，OR函数和AND函数涉及到线性可分的数据，因此它们可以通过线性感知器来表示。另一方面，异或函数设计的数据不是线性可分的，因此需要一个神经网络，如MLP。

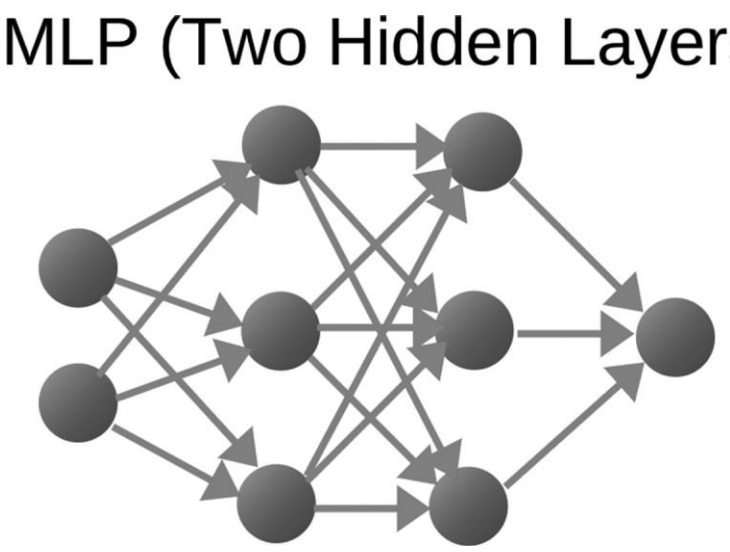


FIGURE 4.3

激活函数

为了防止神经网络层的减少，MLP必须包含相邻层之间的非线性激活函数（对于任何其他深层神经网络也是如此）。非线性激活函数的选择通常是sigmoid、tanh（双曲正切函数）或ReLU（整流线性单元）。sigmoid函数的输出范围从0到1，这具有“压缩”数据值的效果。类似地，tanh函数的输出范围为-1到1。然而，ReLU激活函数（或它的一个变体）是ann和CNNs的首选，而sigmoid和tanh用于LSTMs。

接下来的几节将介绍构造MLP的细节，例如如何初始化MLP的权重、存储权重和偏差以及如何通过反向误差传播训练神经网络。

如何正确分类数据点？

作为参考点，数据点是指数据集中的一行数据可以是房地产数据集，缩略图图像数据集或其他类型的数据集。假设我们要为包含4个类的数据集训练MLP。输出层需要包括四个神经元，其中神经元的索引值为0,1,2,3。由于从倒数第二层转换到输出层使用的softmax函激活函数，输出层中的概率综合始终等于1.

将概率最大的索引值与来自数据集地当前数据点标签的索引值进行比较。如果索引值相等，那么NN已经正确的对当前数据点进行了分裂。

例如:MNIST数据集包含从0到9的手绘数字图像，这意味着MNIST数据集的NN在最后一层有10个输出，每个数字对应一个输出。假设包含数字3的图像当前正在通过NN。3的一次热编码是[0,0,0,1,0,0,0,0,0,0],一次热编码中最大值的索引值的3.现在假设处理数字3后的神经网络输出层是以下值向量：[0.05,0.05,0.2,0.6,0.2,0.2,0.1,0.1,0.238]。对应的，最大值（0.6）的索引值也是3.在这种情况下，神经网络已经正确的识别了输入的图像。

二元分类器包括两个处理任务的结果，如确定垃圾邮件/非垃圾邮件，欺诈/非欺诈，股票增减（或温度气压等的升高或降低）等等。预测股票价格的未来价值是一项回归任务，而预测股票价格的上涨或下跌是一项分类任务。

在机器学习中，多层感知器是一种用于二值分类器监督学习的神经网络（是一种线性分类器）。然而，单层感知器只能学习线性可分模式。事实上，马文·明斯基和西摩·帕普特（1969年写）的一本名著《感知器》（Perceptrons）表明，这些网络类学习异或函数是不可能的。然而，异或函数可以由两层感知器“学习”。

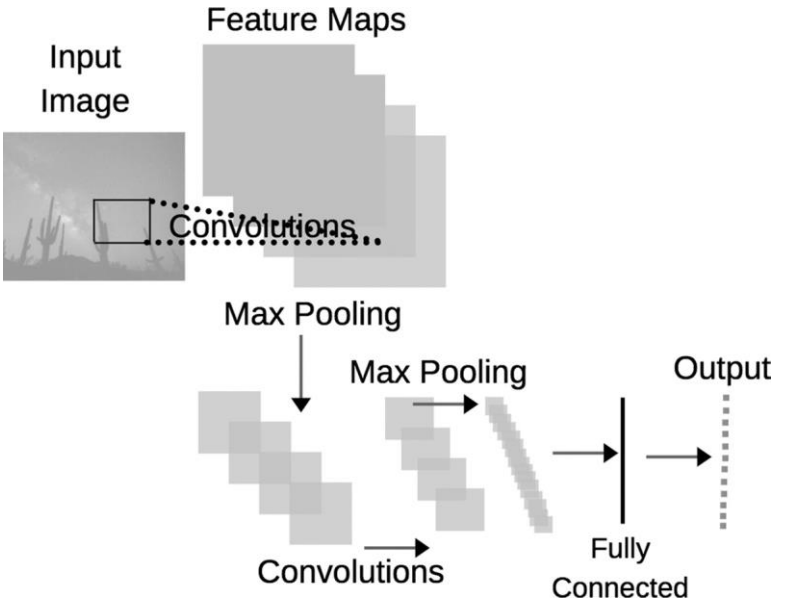
CNN的概述

CNN是深度NN（具有一个或多个卷积层），非常适合图像分类、音频处理、NLP等等领域。

虽然MLP已经成功地应用于图像识别，但是由于相邻层的每一对都是完全连通的，因此它们的计算比较复杂。对于大型图像或其他大型输入，复杂性变得显著，对性能产生不良影响。

Figure 4.4 展示了CNN的内容。

FIGURE 4.4



一个简单的CNN

一个实际应用的CNN可能是非常复杂的，包括许多隐藏层。我们先介绍一个简单的CNN，由以下几层构成：

- Conv2D（卷积层）
- ReLU (激活函数)
- 池化层 (还原技术)
- 全连接层
- Softmax激活功能

下面将作详细解释。

卷积层 (Conv2D)

在python代码中，卷积层通常被标记为Conv2D. Conv2D层涉及一组滤波器，这些滤波器是小方阵，尺寸通常是3x3，但是也可以是5x5, 7x7甚至1x1. 每一个框都扫描在一张图像上，在每一步中，都会使用过滤器和当前位于过滤器下面图像部分计算一个内积。这个扫描的过程的结果称为特征图。

Figure 4.5展示了7x7的数字网络，记忆一个3x3子区域的3x3过滤器的内积，结果是出现在特征图中的数字4.

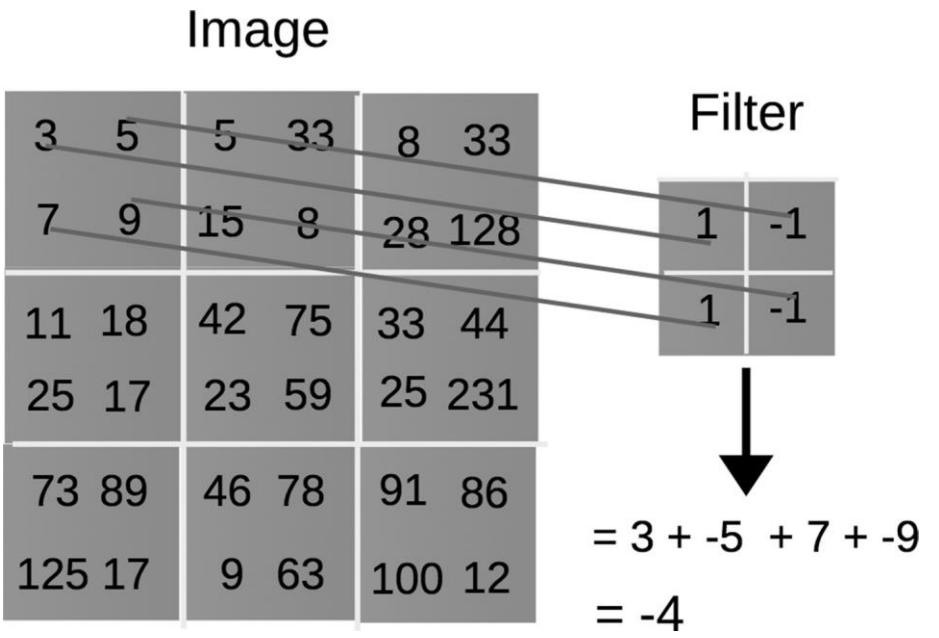


FIGURE 4.5

The ReLU 激活函数

在创建每个特征图之后，特征图上的某些值可能是负值。ReLU激活函数的功能是将这些负值替换为0.ReLU的定义如下：

```
ReLU(x) = x if x >=0 and ReLU(x) = 0 if x < 0
```

如果绘制一个二维的ReLU图，它由两部分组成：x小于0的水平轴和x大于等于0的第一象限中的单位函数（直线）。

最大池化层

第三步是最大池化操作，执行起来很简单：再上一步中ReLU激活函数处理特征映射后，将更新后的特征映射分成2x2个矩形，并从每个矩形中选择最大值。结果是一个更小的数组，包含25%的特征映射（75%的数据被丢弃）。有几种算法可用于执行此提取操作：计算每个平方数字的平均值；每个平方数字的平方和的平方根；或求每个平方和中的最大数。

对于CNN，最大池算法从每个2x2矩形中选择最大的数目。图4.6显示了CNN中最大池的结果。

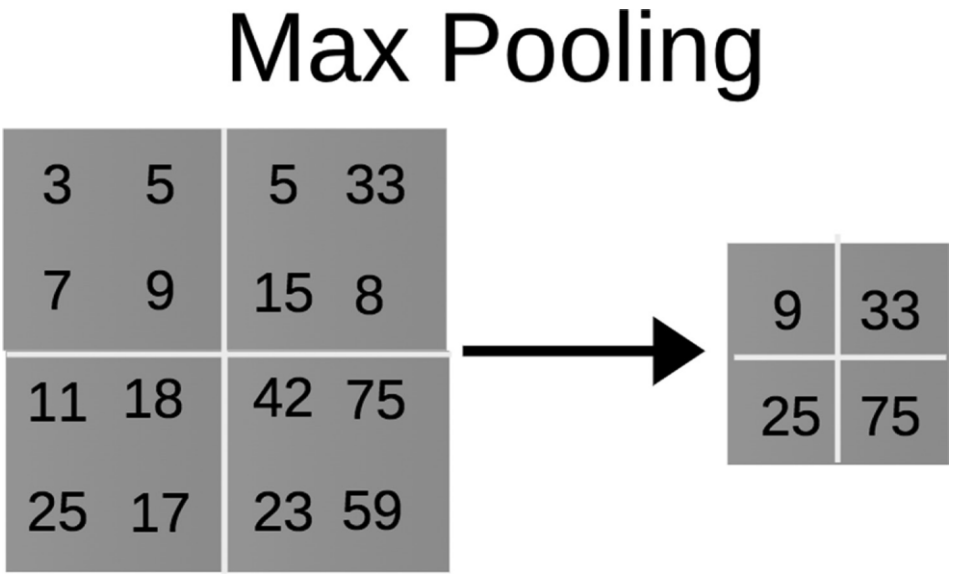


FIGURE 4.6

如你所见，结果是一个小的方形数组，其大小仅为之前的特征映射的25%。对在卷积层中选择的滤波器组中的每个滤波器执行次序列。此集合可以是8个、16个、32个或更多个滤波器。

如果您对这项技术感到困惑，可以考虑压缩算法，压缩算法可分为两类：有损和无损。JPEG是一种有损算法（即，数据在压缩过程中丢失），但它对压缩图像的效果很好。可以把max pooling看作是有损压缩算法的对应，可以说明算法的有效性。

事实上，GeoffreyHinton（经常被称为深度学习的教父）提出了一种称为胶囊网络的的最大池化的替代方案。这种体系结构更复杂，更难训练，而且超出了本书的范围（您可以找到详细讨论胶囊网络的在线教程）。然而，胶囊网络更倾向于抵抗GAN（生成性对抗网络）。

重复前面的一系列步骤（如LeNet），然后形成一个相当非直观的操作：将所有这些小数组展平，使它们成为一维向量，并将这些向量连接成一个（非常长的）向量。结果向量与输出层完全连接，输出层由10个“桶”组成。对于MNIST，这些占位符代表0到9之间的数字（包括0到9）。注意Keras类tf.keras.层。

softmax激活函数应用于长向量的数字，以填充输出层的10个桶。结果是：10个bucket被一组非零（非负）的数字填充，这些数字的总和等于1。找到包含最大数字的存储桶的索引，并将此数字与与刚刚处理的图像相关联的一个热编码标签的索引进行比较。如果索引值相等，则图像被成功地完全识别。

更复杂的cnn涉及多个Conv2D层、多个FC（完全连接）层、不同的滤波器大小以及组合前一层（如ResNet）以增强数据值的当前层的技术。

现在您已经对CNNs有了一个高层次的了解，让我们看一个代码示例，它演示了MNIST数据集中的一个图像（以及该图像的像素值），然后是两个使用Keras在MNIST数据集上训练模型的代码示例。

利用MNIST数据集显示图像

例4.2 展示了如何在TensorFlow中创建一个处理MNIST数据集地神经网络。

Listing 4.2: tf2_keras_mnist_digit.py

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(X_train, y_train), (X_test, y_test) = mnist.
   load_data()

print("X_train.shape:",X_train.shape)
print("X_test.shape: ",X_test.shape)

first_img = X_train[0]

# uncomment this line to see the pixel values
#print(first_img)

import matplotlib.pyplot as plt
plt.imshow(first_img, cmap='gray')
plt.show()
```

4.2从import语句开始，然后从MNIST数据集填充训练数据和测试数据。变量first_img初始化为X_train数组中的第一个条目，该数组是训练数据集中的第一个图像。清单中的最后一个代码块显示第一个图像的像素值。输出如下：

```
X_train.shape: (60000, 28, 28)
X_test.shape:  (10000, 28, 28)
```

Figure4.7展示了MNIST数据集的第一张图像。

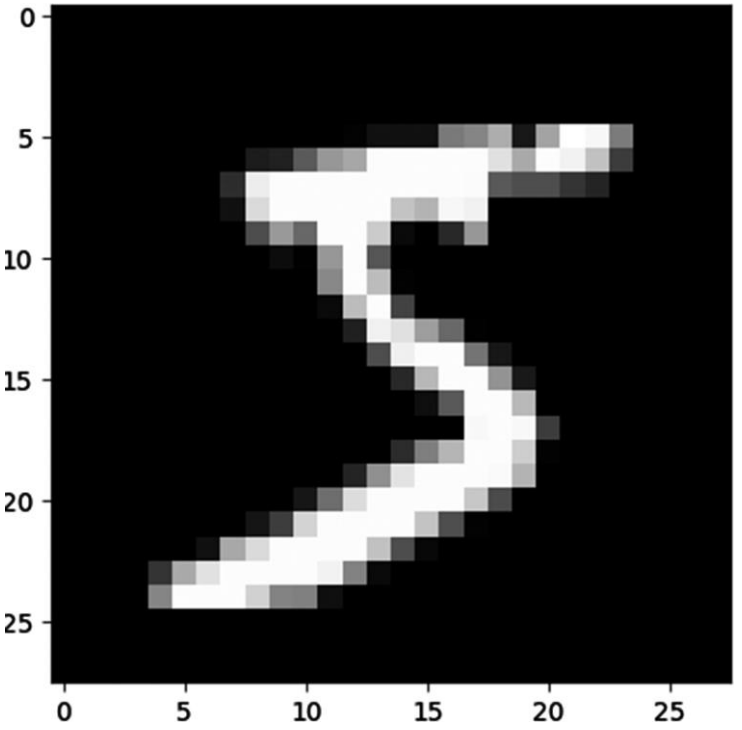


FIGURE 4.7

Keras和MNIST数据集

Listing 4.3 展示了如何在TensorFlow中创建一个基于Keras的神经网络来处理MNIST数据集。

Listing 4.3: keras_mnist.py

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) =mnist.load_
   data()

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.
   nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.
   nn.softmax)
])

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_
   crossentropy',
              metrics=[‘accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

例4.3首先将变量mnist初始化为对内置mnist数据集地引用。接下来，使用mnist数据集地相应不等初始化相关变量，然后对X_train和X-test进行缩放转换。

例4.3的下一部分定义了一个非常简单的基于Keras的模型，它有四个层，这些层从tf.keras.layers中包装得到。下一个代码段显示模型定义的摘要：

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130

```
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```

模型的其余部分为编译、拟合和评估模型，该模型生成以下输出：

```
Epoch 1/5
60000/60000 [==============================] - 14s
   225us/step - loss: 0.2186 - acc: 0.9360
Epoch 2/5
60000/60000 [==============================] - 14s
   225us/step - loss: 0.0958 - acc: 0.9704
Epoch 3/5
60000/60000 [==============================] - 14s
   232us/step - loss: 0.0685 - acc: 0.9783
Epoch 4/5
60000/60000 [==============================] - 14s
   227us/step - loss: 0.0527 - acc: 0.9832
Epoch 5/5
60000/60000 [==============================] - 14s
   225us/step - loss: 0.0426 - acc: 0.9861
10000/10000 [==============================] - 1s
   59us/step
```

模型的最终准确率为 98.6%,是很不错的结果。

Keras,CNNs,和MNIST数据集

4.4 展示了keras_cnn_mnist.py的代码描述了如何基于Keras创建一个神经网络并通过TensorFlow操作该网络。

Listing 4.4: keras_cnn_mnist.py

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images,test_
   labels) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape((60000, 28,28,
   1))
test_images  = test_images.reshape((10000, 28, 28,1))

# Normalize pixel values: from the range 0-255 to
   the range 0-1
train_images, test_images =train_images/255.0,
   test_images/255.0

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3,3),
    activation='relu', input_shape=(28, 28,1)))
 model.add(tf.keras.layers.MaxPooling2D((2,2)))
model.add(tf.keras.layers.Conv2D(64, (3,3),
   activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3,3),
   activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,
   activation='relu'))
```

```
model.add(tf.keras.layers.Dense(10,
   activation='softmax'))

model.summary()

model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
             metrics=[‘accuracy'])

model.fit(train_images, train_labels, epochs=1)
test_loss, test_acc = model.evaluate(test_images,
   test_labels)
print(test_acc)

# predict the label of one image
test_image = np.expand_dims(test_images[300],
   axis = 0)
plt.imshow(test_image.reshape(28,28))
plt.show()

result = model.predict(test_image)
print("result:", result)
print("result.argmax():", result.argmax())
```

例4.4通过load_data()函数初始化训练数据和标签，以及测试数据和标签。接下来，对图像进行整形，使图像的尺寸为28x28，然后将像素值从范围0-255重新缩放为0-1（十进制的）。

例4.4的下一部分使用Keras Sequential() API定义一个名为model的基于keras的模型，它包含两对Conv2D和Maxpooling2D层，后面是faltten层，之后是两个连续的全连接层。

之后通过compile()编译、训练和计算模型，通过fit()和evaluate()方法预测标签为4的图像，然后通过matplotlib显示该图像，输出如下：

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650

```
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

60000/60000 [==============================] - 54s
   907us/sample - loss: 0.1452 - accuracy: 0.9563
10000/10000 [==============================] - 3s
   297us/sample - loss: 0.0408 - accuracy: 0.9868
0.9868
Using TensorFlow backend.
result: [[6.2746993e-05 1.7837329e-03 3.8957372e-04
   4.6143982e-06 9.9723744e-01
  1.5522403e-06 1.9182076e-04 3.0044283e-04
   2.2602901e-05 5.3929521e-06]]
result.argmax(): 4
```

图4.8展示了运行4. 4代码显示的图像。

你可能会觉得当每个输入图像被展平成一维向量时，4.4中的模型为什么会达到这么高的精度，一维向量会丢失二维向量中可用的邻接信息。在CNN变得流行之前，一种技术使用MLP，另一种技术使用SVM作为图像处理的模型。事实上，如果你没有足够的图像来训练一个模型，你仍然可以使用支持向量机建模或者使用GAN生成合成数据。

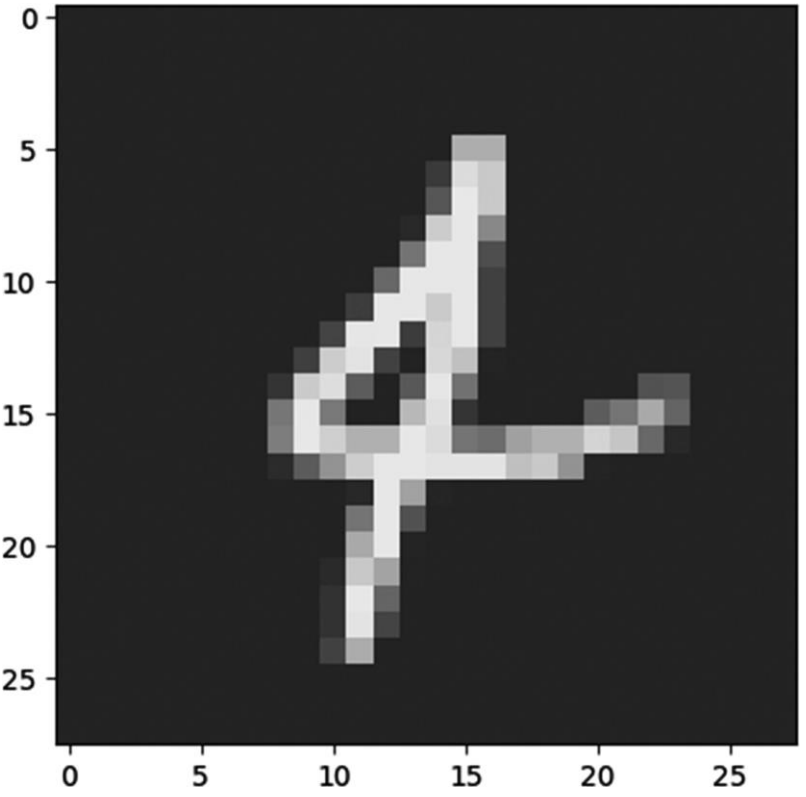


FIGURE 4.8

使用CNN分析音频信号

除了图像分类，你还可以用音频信号训练CNN，音频信号可以从模拟信号转换为数字信号。音频信号具有各种数字参数（包括分贝值、压力值等）如下所示：

https://en.wikipedia.org/wiki/Audio_signal

如果你有一组音频信号，它们相关参数的数值将成为CNN的数据集。CNN是不理解数字输入的：不管数值的来源是什么，数值都是以相同的方式处理的。

其中一个用例涉及到建筑物外的麦克风检测和识别各种声音。很明显，区分车辆和声音和枪声是很重要的。遇到枪声警方应该被告知潜在的犯罪行为。有些公司使用CNN来识别不同类型的声音；还有公司正在探索使用RNN和LSTM来代替CNN。

总结

在本章中，你了解到深度学习的简单介绍、它和机器学习的区别以及它可以解决的问题。你了解了深度学习领域存在的挑战，包括算法的偏差、对抗性攻击的敏感性、泛化能力有限、神经网络缺乏可解释性以及缺乏因果关系。

接下来，我们学习了XOR函数，它是平面上四个点的非线性可分集的一个例子。尽管XOR函数在二维情况下很简单，但是它不能用单层的网络来求解：相反，它需要两个隐藏层。之后你还学习了感知器，它本质上是基于keras神经网络的核心组成部分。

你也可以把基于keras的神经网络的样本作为寻来拿数据。此外，你还了解到CNN是如何构建的，以及一个用于MNIST数据集训练CNN的keras代码示例。