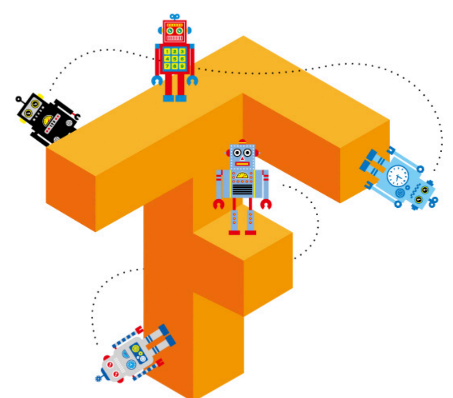
零基础入门，有趣新颖，实用有效



21个项目玩转深度学习

基于TensorFlow的实践详解

何之源◎编著

第 1 章 MNIST 机器学习入门

当我们学习编程语言时，第一课通常会学习一个简单的“Hello World”程序，而 MNIST 手写字符识别可以算得上是机器学习界的“Hello World”。MNIST 是由 Yann LeCun 等人建立的一个手写字符数据集。它简单易用，是一个很好的入门范例。本章会以 MNIST 数据库为例，用 TensorFlow 读取数据集中的数据，并建立一个简单的图像识别模型，同时介绍 TensorFlow 的几个核心概念。

1.1 MNIST 数据集

1.1.1 简介

首先介绍 MNIST 数据集。如图 1-1 所示，MNIST 数据集主要由一些手写数字的图片和相应的标签组成，图片一共有 10 类，分别对应从 0~9，共 10 个阿拉伯数字。

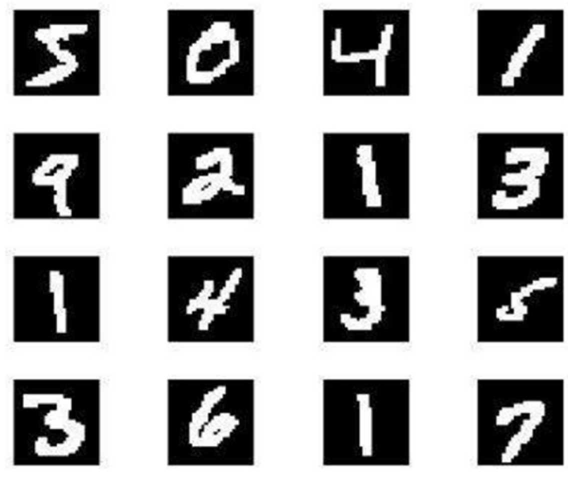


图 1-1　MNIST 数据集图片示例

原始的 MNIST 数据库一共包含下面 4 个文件，见表 1-1。

表 1-1　原始的 MNIST 数据集包含的文件

文件名	大　　小	用　　途
train-images-idx3-ubyte.gz	≈9.45 MB	训练图像数据
train-labels-idx1-ubyte.gz	≈0.03 MB	训练图像的标签
t10k-images-idx3-ubyte.gz	≈1.57MB	测试图像数据
t10k-labels-idx1-ubyte.gz	≈4.4 KB	测试图像的标签

在表 1-1 中，图像数据是指很多张手写字符的图像，图像的标签是指每一张图像实际对应的数字是几，也就是说，在 MNIST 数据集中的每一张图像都事先标明了对应的数字。

在 MNIST 数据集中有两类图像：一类是训练图像（对应文件 train-images-idx3-ubyte.gz 和 train-labels-idx1-ubyte.gz），另一类是测试图像（对应文件 t10k-images-idx3-ubyte.gz 和 t10k-labels-idx1-ubyte.gz）。训练图像一共有 60000 张，供研究人员训练出合适的模型。测试图像一共有 10000 张，供研究人员测试训练的模型的性能。在 TensorFlow 中，可以使用下面的 Python 代码下载 MNIST 数据（在随书附赠的代码中，该代码对应的文件是

donwload.py)。

```
# coding:utf-8
# 从tensorflow.examples.tutorials.mnist引入模块
# 这是TensorFlow为了教学MNIST而提前编制的程序
from tensorflow.examples.tutorials.mnist import input_data
# 从MNIST_data/中读取MNIST数据。这条语句在数据不存在时，会自动执行下载
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

在执行语句 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)时，TensorFlow 会检测数据是否存在。当数据不存在时，系统会自动将数据下载到 MNIST_data/文件夹中。当执行完语句后，读者可以自行前往 MNIST_data/文件夹下查看上述 4 个文件是否已经被正确地下载[1]。

成功加载 MNIST 数据集后，得到了一个 mnist 对象，可以通过 mnist 对象的属性访问到 MNIST 数据集，见表 1-2。

表 1-2　mnist 对象中各个属性的含义和大小

属性名	内　容	大　小
mnist.train.images	训练图像	(55000, 784)
mnist.train.labels	训练标签	(55000, 10)
mnist.validation.images	验证图像	(5000, 784)
mnist.validation.labels	验证标签	(5000, 10)
mnist.test.images	测试图像	(10000, 784)
mnist.test.labels	测试标签	(10000, 10)

运行下列代码可以查看各个变量的形状大小：

```
# 查看训练数据的大小
print(mnist.train.images.shape)  # (55000, 784)
print(mnist.train.labels.shape)  # (55000, 10)

# 查看验证数据的大小
```

1　若因网络问题无法正常下载，可以前往 MNIST 官网 http://yann.lecun.com/exdb/mnist/使用下载工具下载上述 4 个文件，并将它们复制到 MNIST_data/文件夹中。

```
print(mnist.validation.images.shape)  # (5000, 784)
print(mnist.validation.labels.shape)  # (5000, 10)

# 查看测试数据的大小
print(mnist.test.images.shape)  # (10000, 784)
print(mnist.test.labels.shape)  # (10000, 10)
```

原始的 MNIST 数据集中包含了 60000 张训练图片和 10000 张测试图片。而在 TensorFlow 中，又将原先的 60000 张训练图片重新划分成了新的 55000 张训练图片和 5000 张验证图片。所以在 mnist 对象中，数据一共分为三部分:mnist.train 是训练图片数据，mnist.validation 是验证图片数据，mnist.test 是测试图片数据，这正好对应了机器学习中的训练集、验证集和测试集。一般来说，会在训练集上训练模型，通过模型在验证集上的表现调整参数，最后通过测试集确定模型的性能。

1.1.2 实验：将 MNIST 数据集保存为图片

在原始的 MNIST 数据集中，每张图片都由一个 28×28 的矩阵表示，如图 1-2 所示。

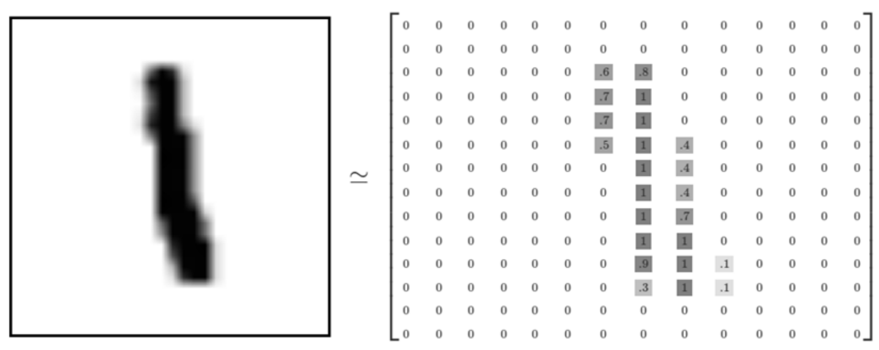


图 1-2 单张图片样本的矩阵表示

在 TensorFlow 中，变量 mnist.train.images 是训练样本，它的形状为(55000, 784)。其中，5000 是训练图像的个数，而 784 实际为单个样本的维数，即每张图片都由一个 784 维的向量表示（784 正好等于 28×28）。可以使用以下代

码打印出第 0 张训练图片对应的向量表示：

```
# 打印出第 0 张图片的向量表示
print(mnist.train.images[0, :])
```

为了加深对这种表示的理解，下面完成一个简单的程序：将 MNIST 数据集读取出来，并保存为图片文件。对应的代码文件为 save_pic.py。

```
#coding: utf-8
from tensorflow.examples.tutorials.mnist import input_data
import scipy.misc
import os

# 读取 MNIST 数据集。如果不存在会事先下载
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# 把原始图片保存在 MNIST_data/raw/文件夹下
# 如果没有这个文件夹，会自动创建
save_dir = 'MNIST_data/raw/'
if os.path.exists(save_dir) is False:
   os.makedirs(save_dir)

# 保存前 20 张图片
for i in range(20):
   # 请注意，mnist.train.images[i, :]就表示第 i 张图片（序号从 0 开始）
   image_array = mnist.train.images[i, :]
   # TensorFlow 中的 MNIST 图片是一个 784 维的向量，我们重新把它还原为 28×28 维的图像
   image_array = image_array.reshape(28, 28)
   # 保存文件的格式为:
   # mnist_train_0.jpg, mnist_train_1.jpg, ... ,mnist_train_19.jpg
   filename = save_dir + 'mnist_train_%d.jpg' % i
   # 将 image_array 保存为图片
   # 先用 scipy.misc.toimage 转换为图像，再调用 save 直接保存
   scipy.misc.toimage(image_array, cmin=0.0, cmax=1.0).save(filename)
```

运行此程序后，在 MNIST_data/raw/文件夹下就可以看到 MNIST 数据集中训练集的前 20 张图片。读者可以修改上述程序打印更多的图片。

1.1.3 图像标签的独热表示

变量 mnist.train.labels 表示训练图像的标签，它的形状是(55000, 10)。原始的图像标签是数字 0~9，我们完全可以用一个数字来存储图像标签，但为什么这里每个训练标签是一个 10 维的向量呢？其实，这个 10 维的向量是原先类别号的独热（one-hot）表示。

所谓独热表示，就是“一位有效编码”。我们用 *N* 维的向量来表示 *N* 个类别，每个类别占据独立的一位，任何时候独热表示中只有一位是 1，其他都为 0。读者可以直接从表 1-3 中理解独热表示。

表 1-3　类别的原始表示和独热表示

原始表示 （0~9，共 10 个类别）	独热表示 （10 维向量，每一维对应一个类别）
0	(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
1	(0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
2	(0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
3	(0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
……	……
9	(0, 0, 0, 0, 0, 0, 0, 0, 0, 1)

运行下面的代码可以打印出第 0 张训练图片的标签：

```
# 打印出第 0 张训练图片的标签
print(mnist.train.labels[0, :])
```

代码运行的结果是[0.　0.　0.　0.　0.　0.　0.　1.　0.　0.]，也就是说第 0 张图片对应的标签为数字“7”。

此外，我们可以打印出前 20 张图片的标签（对应程序 label.py），读者可以尝试与第 1.1.2 节中保存的图片对照，查看图像与图像的标签是否正确地对应上了。

```
# coding: utf-8
from tensorflow.examples.tutorials.mnist import input_data
```

```
import numpy as np
# 读取MNIST数据集。如果不存在会事先下载
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# 看前20张训练图片的label
for i in range(20):
    # 得到独热表示，形如(0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
    one_hot_label = mnist.train.labels[i, :]
    # 通过np.argmax，可以直接获得原始的label
    # 因为只有1位为1，其他都是0
    label = np.argmax(one_hot_label)
    print('mnist_train_%d.jpg label: %d' % (i, label))
```

至此，读者应当对变量 mnist.train.images 和 mnist.train.labels 很熟悉了。剩下的 mnist.validation.images、mnist.validation.labels、mnist.test.images、mnist.test.labels 四个变量与它们非常类似，唯一的区别只是图像的个数不同，本章就不再做更详细的解释了。

1.2 利用 TensorFlow 识别 MNIST

在第 1.1 节中，我们已经对 MNIST 数据集和 TensorFlow 中 MNIST 数据集的载入有了基本的了解。本节将真正以 TensorFlow 为工具，写一个手写体数字识别程序，使用的机器学习方法是 Softmax 回归。

1.2.1 Softmax 回归

1．Softmax 回归的原理

Softmax 回归是一个线性的多类分类模型，实际上它是直接从 Logistic 回归模型转化而来的。区别在于 Logistic 回归模型为两类分类模型，而 Softmax 模型为多类分类模型。

在手写体识别问题中，一共有 10 个类别（0~9），我们希望对输入的图像计算它属于每个类别的概率。如属于 9 的概率为 70%，属于 1 的概率为 10%等。最后模型预测的结果就是概率最大的那个类别。

先来了解什么是 Softmax 函数。Softmax 函数的主要功能是将各个类别的“打分”转化成合理的概率值。例如，一个样本可能属于三个类别：第一个类别的打分为 a，第二个类别的打分为 b，第三个类别的打分为 c。打分越高代表属于这个类别的概率越高，但是打分本身不代表概率，因为打分的值可以是负数，也可以很大，但概率要求值必须在 0~1，并且三类的概率加起来应该等于 1。那么，如何将(a, b, c)转换成合理的概率值呢？方法就是使用 Softmax 函数。例如，对$(a,\ b,\ c)$使用 Softmax 函数后，相应的值会变成$\left(\frac{\mathrm{e}^a}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}, \frac{\mathrm{e}^b}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}, \frac{\mathrm{e}^c}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}\right)$，也就是说，第一类的概率可以用$\frac{\mathrm{e}^a}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}$表示，第二类的概率可以用$\frac{\mathrm{e}^b}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}$表示，第三类的概率可以用$\frac{\mathrm{e}^c}{\mathrm{e}^a+\mathrm{e}^b+\mathrm{e}^c}$表示。显然，这三个数值都在 0~1 之间，并且加起来正好等于 1，是合理的概率表示。

假设 $\boldsymbol{x}$ 是单个样本的特征，$\boldsymbol{W}$、$\boldsymbol{b}$ 是 Softmax 模型的参数。在 MNIST 数据集中，$\boldsymbol{x}$ 就代表输入图片，它是一个 784 维的向量，而 $\boldsymbol{W}$ 是一个矩阵，它的形状为(784, 10)，$\boldsymbol{b}$ 是一个 10 维的向量，10 代表的是类别数。Softmax 模型的第一步是通过下面的公式计算各个类别的 Logit：

$$\mathrm{Logit} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}$$

Logit 同样是一个 10 维的向量，它实际上可以看成样本对应于各个类别的“打分”。接下来使用 Softmax 函数将它转换成各个类别的概率值：

$$y = \mathrm{Softmax}(\mathrm{Logit})$$

Softmax 模型输出的 y 代表各个类别的概率，还可以直接用下面的式子来表示整个 Softmax 模型:

$$y = \mathrm{Softmax}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b})$$

2．Softmax 回归在 TensorFlow 中的实现

本节对应的程序为 softmax_regression.py，在该程序中，使用 TensorFlow 定义了一个 Softmax 模型，实现了 MNIST 数据集的分类。首先导入 TensorFlow 模块：

```
# 导入TensorFlow
```

```
# 这句 import tensorflow as tf 是导入 TensorFlow 约定俗成的做法，请大家记住
import tensorflow as tf
```

导入 TensorFlow 的语句一般写作：import tensorflow as tf。这是一种约定俗成的写法。请记住这条语句，它将在后面的每一章中重复出现。

接下来和之前一样，导入 MNIST 数据库：

```
# 导入 MNIST 教学的模块
from tensorflow.examples.tutorials.mnist import input_data
# 与之前一样，读入 MNIST 数据
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

下面的步骤是非常关键的几步，先来看代码：

```
# 创建 x，x 是一个占位符（placeholder），代表待识别的图片
x = tf.placeholder(tf.float32, [None, 784])

# W 是 Softmax 模型的参数，将一个 784 维的输入转换为一个 10 维的输出
# 在 TensorFlow 中，变量的参数用 tf.Variable 表示
W = tf.Variable(tf.zeros([784, 10]))
# b 是又一个 Softmax 模型的参数，一般叫作“偏置项”（bias）
b = tf.Variable(tf.zeros([10]))

# y 表示模型的输出
y = tf.nn.softmax(tf.matmul(x, W) + b)

# y_ 是实际的图像标签，同样以占位符表示
y_ = tf.placeholder(tf.float32, [None, 10])
```

这里定义了一些**占位符和变量（Variable）**。在 TensorFlow 中，无论是占位符还是变量，它们实际上都是“Tensor”。从 TensorFlow 的名字中，就可以看出 Tensor 在整个系统中处于核心地位。TensorFlow 中的 Tensor 并不是具体的数值，它只是一些我们“希望”TensorFlow 系统计算的“节点”。

这里的占位符和变量是不同类型的 Tensor。先来讲解占位符。占位符不依赖于其他的 Tensor，它的值由用户自行传递给 TensorFlow，通常用来存储样本数据和标签。如在这里定义了 x = tf.placeholder(tf.float32, [None, 784])，它是用来存储训练图片数据的占位符。它的形状为[None, 784]，None 表示这一维的大小可以是任意的，也就是说可以传递任意张训练图片给这个占位

符，每张图片用一个 784 维的向量表示。同样的，y_ = tf.placeholder(tf.float32, [None, 10])也是一个占位符，它存储训练图片的实际标签。

再来看什么是变量。变量是指在计算过程中可以改变的值，每次计算后变量的值会被保存下来，通常用变量来存储模型的参数。如这里创建了两个变量：W = tf.Variable(tf.zeros([784, 10]))、b = tf.Variable(tf.zeros([10]))。它们都是 Softmax 模型的参数。创建变量时通常需要指定某些初始值。这里 W 的初始值是一个 784×10 的全零矩阵，b 的初始值是一个 10 维的 0 向量。

除了变量和占位符之外，还创建了一个 y = tf.nn.softmax(tf.matmul(x, W) + b)。这个 y 就是一个依赖 x、W、b 的 Tensor。如果要求 TensorFlow 计算 y 的值，那么系统首先会获取 x、W、b 的值，再去计算 y 的值。

$\boldsymbol{y}$ 实际上定义了一个 Softmax 回归模型，在此可以尝试写出 $\boldsymbol{y}$ 的形状。假设输入 $\boldsymbol{x}$ 的形状为$(N, 784)$，其中 $\boldsymbol{N}$ 表示输入的训练图像的数目。$\boldsymbol{W}$ 的形状为(784, 10)，$\boldsymbol{b}$ 的形状为(10,)[1]。那么，$\boldsymbol{Wx}+\boldsymbol{b}$ 的形状是$(\boldsymbol{N}, 10)$。Softmax 函数不改变结果的形状，所以得到 $\boldsymbol{y}$ 的形状为$(\boldsymbol{N}, 10)$。也就是说，$\boldsymbol{y}$ 的每一行是一个 10 维的向量，表示模型预测的样本对应到各个类别的概率。

模型的输出是 y，而实际的标签为 y_，它们应当越相似越好。在 Softmax 回归模型中，通常使用“交叉熵”损失来衡量这种相似性。损失越小，模型的输出就和实际标签越接近，模型的预测也就越准确。

在 TensorFlow 中，这样定义交叉熵损失：

```
# 至此，得到了两个重要的Tensor：y和y_
# y是模型的输出，y_是实际的图像标签，注意y_是独热表示的
# 下面会根据y和y_构造损失

# 根据y和 y_构造交叉熵损失
cross_entropy = \
   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y)))
```

1　形状（784.10）表示一个 784 行 10 列的矩阵，形状（10，）表示一个 10 维的列向量，下同。

构造完损失之后，下面一步是如何优化损失，让损失减小。这里使用梯度下降法优化损失，定义为

```
# 有了损失，就可以用梯度下降法针对模型的参数（W和b）进行优化
train_step = tf.train.GradientDescentOptimizer(0.01).minimize
(cross_entropy)
```

TensorFlow 默认会对所有变量计算梯度。在这里只定义了两个变量 W 和 b，因此程序将使用梯度下降法对 W、b 计算梯度并更新它们的值。tf.train.GradientDescentOptimizer(0.01)中的 0.01 是梯度下降优化器使用的学习率（Learning Rate）。

在优化前，必须要创建一个会话（Session），并在会话中对变量进行初始化操作：

```
# 创建一个Session。只有在Session中才能运行优化步骤train_step
sess = tf.InteractiveSession()
# 运行之前必须要初始化所有变量，分配内存
tf.global_variables_initializer().run()
```

会话是 TensorFlow 的又一个核心概念。前面提到 Tensor 是“希望”TensorFlow 进行计算的结点。而会话就可以看成对这些结点进行计算的上下文。之前还提到过，变量是在计算过程中可以改变值的 Tensor，同时变量的值会被保存下来。事实上，变量的值就是被保存在会话中的。在对变量进行操作前必须对变量进行初始化，实际上是在会话中保存变量的初始值。初始化所有变量的语句是 tf.global_variables_initializer().run()。

有了会话，就可以对变量 W、b 进行优化了，优化的程序如下：

```
# 进行1000步梯度下降
for _ in range(1000):
    # 在mnist.train中取100个训练数据
    # batch_xs是形状为(100, 784)的图像数据，batch_ys是形如(100, 10)的实际标签
    # batch_xs, batch_ys对应着两个占位符x和y_
    batch_xs, batch_ys = mnist.train.next_batch(100)
    # 在Session中运行train_step，运行时要传入占位符的值
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

每次不使用全部训练数据，而是每次提取 100 个数据进行训练，共训练 1000 次。batch_xs, batch_ys 分别是 100 个训练图像及其对应的标签。在训练时，需要把它们放入对应的占位符 x，y_中，对应的语句是 feed_dict={x: batch_xs, y_: batch_ys}。

在会话中，不需要系统计算占位符的值，而是直接把占位符的值传递给会话。与变量不同的是，占位符的值不会被保存，每次可以给占位符传递不同的值。

运行完梯度下降后，可以检测模型训练的结果，对应的代码如下：

```
# 正确的预测结果
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
# 计算预测准确率，它们都是Tensor
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# 在Session 中运行Tensor 可以得到Tensor 的值
# 这里是获取最终模型的准确率
print(sess.run(accuracy,feed_dict={x:mnist.test.images,y_:mnist.test.lab
els}))# 0.9185
```

模型预测 y 的形状是$(N, 10)$，而实际标签 y_的形状是$(N, 10)$，其中 N 为输入模型的样本个数。tf.argmax(y, 1)、tf.argmax(y_, 1)的功能是取出数组中最大值的下标，可以用来将独热表示以及模型输出转换为数字标签。假设传入四个样本，它们的独热表示 y_为（需要通过 sess.run(y_)才能获取此 Tensor 的值，下同）：

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

tf.argmax(y_, 1)就是：

```
[0, 2, 9, 0]
```

也就是说，取出每一行最大值对应的下标位置，它们是输入样本的实际标签。假设此时模型的预测输出 y 为：

```
[[0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
```

```
[0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
[0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
[0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]]
```

tf.argmax(y_, 1)就是：

```
[0, 0, 0, 0]
```

得到了预测的标签和实际标签，接下来通过 tf.equal 函数来比较它们是否相等，并将结果保存到 correct_prediction 中。在上述例子中，correct_prediction 就是：

```
[True, False, False, True]
```

即第一个样本和最后一个样本预测是正确的，另外两个样本预测错误。可以用 tf.cast(correct_prediction, tf.float32)将比较值转换成 float32 型的变量，此时 True 会被转换成 1，False 会被转换成 0。在上述例子中，tf.cast(correct_prediction, tf.float32)的结果为：

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

最后，用 tf.reduce_mean 可以计算数组中的所有元素的平均值，相当于得到了模型的预测准确率，如[1., 0., 0., 1.]的平均值为 0.5，即 50%的分类准确率。

在程序 softmax_regression.py 中，传入占位符的值是 feed_dict={x: mnist.test.images, y_: mnist.test.labels}。也就是说，使用全体测试样本进行测试。测试图片一共有 10000 张，运行的结果为 0.9185，即 91.85%的准确率。因为 Softmax 回归是一个比较简单的模型，这里预测的准确率并不高，在下一节将学习如何使用卷积神经网络将预测的准确率提高到 99%。

1.2.2 两层卷积网络分类

本节对应的程序文件是 convolutional.py，将建立一个卷积神经网络，它可以把 MNIST 手写字符的识别准确率提高到 99%，读者可能需要一些卷积神经网络的基础知识才能更好地理解本节的内容。

程序的开头依旧是导入 TensorFlow：

```
# coding: utf-8
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

接下来载入 MNIST 数据，并建立占位符。占位符 x 的含义为训练图像，y_为对应训练图像的标签，这与上文是一样的。

```
# 读入数据
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# x 为训练图像的占位符、y_为训练图像标签的占位符
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
```

由于使用的是卷积网络对图像进行分类，所以不能再使用 784 维的向量表示输入的 x，而是将其还原为 28×28 的图片形式。[-1, 28, 28, 1]中的-1 表示形状第一维的大小是根据 x 自动确定的。

```
# 将单张图片从 784 维向量重新还原为 28×28 的矩阵图片
x_image = tf.reshape(x, [-1, 28, 28, 1])
```

x_image 就是输入的训练图像，接下来，我们对训练图像进行卷积计算，第一层卷积的代码如下：

```
def weight_variable(shape):
   initial = tf.truncated_normal(shape, stddev=0.1)
   return tf.Variable(initial)


def bias_variable(shape):
   initial = tf.constant(0.1, shape=shape)
   return tf.Variable(initial)


def conv2d(x, W):
   return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x2(x):
   return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
```

```
# 第一层卷积层
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

先定义了四个函数，函数 weight_variable 可以返回一个给定形状的变量并自动以截断正态分布初始化，bias_variabale 同样返回一个给定形状的变量，初始化时所有值是 0.1，可分别用这两个函数创建卷积的核(kernel)与偏置(bias)。h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)是真正进行卷积计算，卷积计算后选用 ReLU 作为激活函数。h_pool1 = max_pool_2x2(h_conv1)是调用函数 max_pool_2x2 进行一次池化操作。卷积、激活函数、池化，可以说是一个卷积层的“标配”，通常一个卷积层都会包含这三个步骤，有时也会去掉最后的池化操作。

对第一次卷积操作后产生的 h_pool1 再做一次卷积计算，使用的代码与上面类似。

```
# 第二层卷积层
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

两层卷积层之后是全连接层：

```
# 全连接层，输出为 1024 维的向量
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# 使用 Dropout，keep_prob 是一个占位符，训练时为 0.5，测试时为 1
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

在全连接层中加入了 Dropout，它是防止神经网络过拟合的一种手段。在每一步训练时，以一定概率“去掉”网络中的某些连接，但这种去除不是永久性的，只是在当前步骤中去除，并且每一步去除的连接都是随机选择的。

在这个程序中，选择的 Dropout 概率是 0.5，也就是说训练时每一个连接都有 50%的概率被去除。在测试时保留所有连接。

最后，再加入一层全连接，把上一步得到的 h_fc1_drop 转换为 10 个类别的打分。

```
# 把 1024 维的向量转换成 10 维，对应 10 个类别
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

y_conv 相当于 Softmax 模型中的 Logit，当然可以使用 Softmax 函数将其转换为 10 个类别的概率，再定义交叉熵损失。但其实 TensorFlow 提供了一个更直接的 tf.nn.softmax_cross_entropy_with_logits 函数，它可以直接对 Logit 定义交叉熵损失，写法为

```
# 不采用先 Softmax 再计算交叉熵的方法
# 而是用 tf.nn.softmax_cross_entropy_with_logits 直接计算
cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
# 同样定义 train_step
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

定义测试的准确率（和第 1.2.1 节类似）:

```
# 定义测试的准确率
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

训练过程同样与第 1.2.1 节类似，不同点在于这次会额外在验证集上计算模型的准确度并输出，方便监控训练的进度，也可以据此来调整模型的参数。

```
# 创建 Session，对变量初始化
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())

# 训练 20000 步
for i in range(20000):
   batch = mnist.train.next_batch(50)
   # 每 100 步报告一次在验证集上的准确率
```

```
  if i % 100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
      x: batch[0], y_: batch[1], keep_prob: 1.0})
    print("step %d, training accuracy %g" % (i, train_accuracy))
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
```

训练结束后，打印在全体测试集上的准确率：

```
# 训练结束后报告在测试集上的准确率
print("test accuracy %g" % accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

得到的准确率结果应该在 99%左右。与 Softmax 回归模型相比，使用两层卷积的神经网络模型借助了卷积的威力，准确率有非常大的提升。本节的程序同第 1.2.1 节在流程上非常相似，都是先读入 MNIST 数据集，再定义训练数据的占位符（x 和 y_），以 x 为输入定义模型，最后定义损失，进行训练。

1.3 总结

本章中首先介绍了 MNIST 数据集，以及如何使用 TensorFlow 把它读到内存中。接着通过一个简单的 Softmax 回归模型例子，学习如何使用 TensorFlow 建立简单的图像识别模型。虽然 Softmax 回归模型只能达到 92%左右的准确率，但它让我们了解了使用 TensorFlow 处理问题的基本流程。最后，我们使用 TensorFlow 建立了有两层卷积层的神经网络，将 MNIST 的识别准确率提高到 99%。

拓展阅读

- 本章介绍的 MNIST 数据集经常被用来检验机器学习模型的性能，在它的官网（地址：http://yann.lecun.com/exdb/mnist/）中，可以找到多达 68 种模型在该数据集上的准确率数据，包括相应的论文出处。这些模型包括线性分类器、K 近邻方法、普通的神经网络、卷积神经网络等。

- 本章的两个 MNIST 程序实际上来自于 TensorFlow 官方的两个新手教程，地址为 https://www.tensorflow.org/get_started/mnist/beginners 和 https://www. tensorflow.org/get_started/mnist/pros。读者可以将本书的内容和官方的教程对照起来进行阅读。这两个新手教程的中文版地址为 http://www.tensorfly.cn/tfdoc/tutorials/mnist_beginners.html 和 http://www. tensorfly.cn/tfdoc/tutorials/mnist_pros.html。

- 本章简要介绍了 TensorFlow 的 tf.Tensor 类。tf.Tensor 类是 TensorFlow 的核心类，常用的占位符（tf.placeholder）、变量（tf.Variable）都可以看作特殊的 Tensor。读者可以参阅 https://www.tensorflow.org/programmers_guide/tensors 来更深入地学习它的原理。

- 常用 tf.Variable 类来存储模型的参数，读者可以参阅 https://www.tensorflow.org/programmers_guide/variables 详细了解它的运行机制，文档的中文版地址为 http://www.tensorfly.cn/tfdoc/how_tos/ variables.html。

- 只有通过会话（Session）才能计算出 tf.Tensor 的值。强烈建议读者在学习完 tf.Tensor 和 tf.Variable 后，阅读 https://www.tensorflow.org/programmers_ guide/graphs 中的内容，该文档描述了 TensorFlow 中计算图和会话的基本运行原理，对理解 TensorFlow 的底层原理有很大帮助。

第 2 章 CIFAR-10 与 ImageNet 图像识别

本章的主要任务还是图像识别，使用的数据集是 CIFAR-10——这是一个更接近普适物体的彩色图像数据集。主要通过 CIFAR-10 学习两方面的内容：一是 TensorFlow 中的数据读取原理，二是深度学习中数据增强的原理。最后还会介绍更加通用且复杂的 ImageNet 数据集和相应的图像识别模型。

2.1 CIFAR-10 数据集

2.1.1 CIFAR-10 数据集简介

CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。它一共包含 10 个类别的 RGB 彩色图片：飞机（airplane）、汽车（automobile）、鸟类（bird）、猫（cat）、鹿（deer）、

狗（dog）、蛙类（frog）、马（horse）、船（ship）和卡车（truck）。图片的尺寸为 32×32，数据集中一共有 50000 张训练图片和 10000 张测试图片。CIFAR-10 的图片样例如图 2-1 所示。

图 2-1　CIFAR-10 数据集的图片样例

与 MNIST 数据集相比，CIFAR-10 具有以下不同点：

- CIFAR-10 是 3 通道的彩色 RGB 图像，而 MNIST 是灰度图像。
- CIFAR-10 的图片尺寸为 32×32，而 MNIST 的图片尺寸为 28×28，比 MNIST 稍大。
- 相比于手写字符，CIFAR-10 含有的是现实世界中真实的物体，不仅噪声很大，而且物体的比例、特征都不尽相同，这为识别带来很大困难。直接的线性模型如 Softmax 在 CIFAR-10 上表现得很差。

本章将以 CIFAR-10 为例，介绍深度图像识别的基本方法。本章代码中的一部分来自于 TensorFlow 的官方示例，如表 2-1 所示。