第 3 章

TensorFlow 实现神经网络模型训练与测试

本章介绍在 TensorFlow 学习框架下，以 MNIST 手写字体的识别为例，实现 Softmax Regression 简易神经网络及卷积神经网络模型的训练，重点介绍基于 FPGA 实现神经网络中用到的典型 TensorFlow 功能，如数据集格式转换等。

3.1 神经网络训练与测试的基本概念

对于常规的监督机器学习或深度学习，一般分为训练和预测两个阶段。在训练阶段，需要准备原始数据和与之对应的分类标签数据，通过训练得到神经网络模型。在预测阶段，需要使用训练阶段得到的神经网络模型对测试数据进行处理。训练阶段和测试阶段使用的输入数据集合分别称为训练样本集和测试样本集。

在设计人工神经网络模型结构前，需要对输入和输出的数据进行量化。假设输入的数据有 k 个，输出为 n 个分类，那么输入层的神经元节点数应设置为 k，输出层神经元节点数应设置为 n。隐形层的神经元节点数可根据不同的应用需求进行合理设置，隐形层的神经元节点数越多，神经网络的非线性越显著，能够处理的应用越复杂，但会增加计算的复杂度。习惯上，一般第 l 层神经网络的节点数是 $l-1$ 层节点数的 1～1.5 倍。

当人工神经网络模型结构定义好后，输入层、隐形层和输出层的节点数就会确定，剩余没有确定的参数还有权值 W 和偏置 b。

3.1.1 神经网络的训练

训练神经网络模型的目的是确定权值 W 和偏置 b，训练的过程也称神经网络的学习过程。神经网络的训练实际上是通过算法不断调整权值 W 和偏置 b，以尽可能地与真实的模型逼近，从而使整个神经网络的预测效果达到最佳。

首先给所有的权值 W 和偏置 b 赋予随机值，使用这些随机生成的权值和偏置参数预测训练样本集合。假设样本的预测值为 y'，真实值为 y。定义一个损失函数衡量预测值和真实值的逼近程度，损失函数越小，神经网络模型的预测效果越好。因此，训练的过程

可以理解为调整权值 W 和偏置 b 以使损失函数最小。

损失函数的定义有多种形式，求损失函数最小值是一个优化问题。对神经网络的优化就是使损失函数收敛。为解决该优化问题，目前的方法是使用反向传播算法求得网络模型所有参数的梯度，通过梯度下降算法对网络参数进行更新，当损失函数收敛到一定程度时结束训练，并保存训练结束时的神经网络参数。

3.1.2 神经网络的测试

在训练阶段，通过算法修改神经网络的权值 W 和偏置 b，以使损失函数最小，当损失函数收敛到某个阈值或等于 0 时停止训练，就可以得到训练后的模型的权值和偏置。此时，神经网络的所有参数：输入层、输出层、隐形层、权值和偏置都是已知的。

在测试阶段，将神经网络的测试样本集从输入层开始输入，沿数据流动的方向在网络中进行计算，直到数据从输出层输出，输出层的输出即为预测结果。

3.2 基于 TensforFlow 训练神经网络实现 MNIST 数据集识别

3.2.1 MNIST 数据集

MNIST 数据集来自美国国家标准与技术研究所(National Institute of Standards and Technology，NIST)。训练集 (training set) 由 250 个不同人手写的数字构成，其中 50％是高中学生，50％是人口普查局(the Census Bureau)的工作人员。测试集(test set)也是同样比例的手写数字数据。训练集有 55 000 个样本，测试集有 10 000 个样本，同时验证集有 5000 个样本。每个样本都有与之相对应的标注信息，即 label。MNIST 数据集被分成三部分：55 000 行的训练数据集(mnist.train)、10 000 行的测试数据集(mnist.test)和 5000 行的验证数据集(mnist.validation)。这样的划分很重要，在机器学习模型设计时，必须有一个单独的测试数据集不用于训练，而是用来评估这个模型的性能，从而可以更容易地把设计的模型推广到其他数据集上，使其具有更好的泛化性能。

每个 MNIST 数据样本都由两部分组成：一张包含手写数字的图片和一个对应的标签。训练数据集的图片和标签分别是 mnist.train.images 和 mnist.train.labels。测试数据集的图片和标签分别是 mnist.test.images 和 mnist.test.labels。

所有数据集中的样本均为 28×28 像素的灰度图片，即每张图片中有 28×28＝784 个像素点。可以用一个数字数组表示每张图片，如图 3-1 所示。空白部分全部为 0，有笔迹的地方根据颜色深浅在 0～1 之间取值。

把这个 28×28 的矩阵展开成一维数组，数组长度是 28×28＝784。如何展开这个数组不重要，只要保持各张图片采用相同的方式展开即可。

注意：展平图片的数字数组会丢失图片的二维结构信息，这显然是不理想的，最优秀的计算机视觉方法会挖掘并利用这些结构信息。此处不需要建立复杂的模型，所以对问题进行了简化，丢弃了空间结构信息。

在 MNIST 训练数据集中，mnist.train.images 是一个形状为[55000，784] 的张量

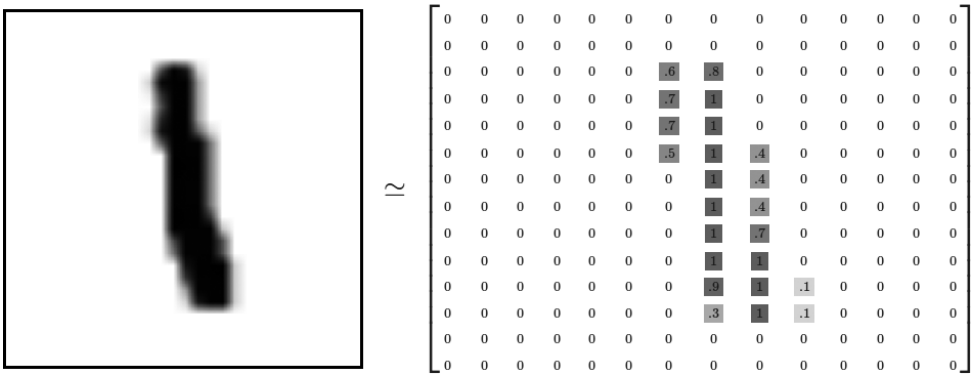


图 3-1　手写数字灰度信息示例

(Tensor),第一个维度是图片的编号,用来索引图片;第二个维度是图片中像素点的编号,用来索引每张图片中的像素点。因此,mnist.train.images 张量中的每个元素都表示某张图片中的某个像素的强度值,值介于 0 和 1。mnist.train.images 张量中的每张图片都可表示为一个 784 维的向量(如图 3-2 所示)。

相对应的 MNIST 数据集的标签 mnist.train.labels 是一个形状为[55000,10]的张量(Tensor),第一个维度是标签的编号,用来索引图片(如图 3-3 所示)。第二个维度是标签的取值,用来表示对应图片代表的数字,是 0～9 的数字。标签数据采用 one-hot 编码,即一个 one-hot 数据除了某一位的数字是 1 以外,其余各维度的数字都是 0。数字 n 将表示成一个只有在第 n 维度(从 0 开始)数字为 1 的 10 维向量。例如,标签 0 将表示成([1,0,0,0,0,0,0,0,0,0])。

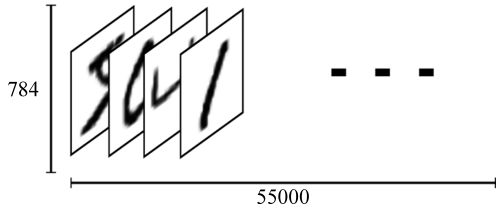


图 3-2　MNIST 训练数据的像素

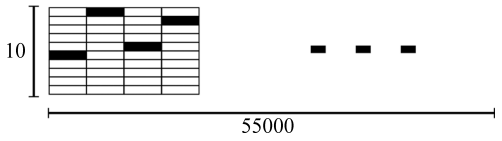


图 3-3　MNIST 训练数据的标签

3.2.2　Softmax Regression 模型

MNIST 的每张图片都表示一个 0～9 的数字。完成图片识别需要根据给定图片代表的每个数字的概率。例如,模型可能推测一张图片代表数字 9 的概率是 80%,代表数字 8 的概率是 5%,代表其他 8 种数字的概率比 5%更小,则模型预测该图片代表数字 9。这是一个使用 Softmax Regression 模型的经典案例。Softmax Regression 模型可以用来给不同的对象分配概率。

为了得到一张给定图片属于某个特定数字类的特征,需要对图片像素进行加权求和。如果这个像素具有很强的特征可以说明这张图片不属于该类,那么相应的权值为负数,相反如果这个像素具有很强的特征支持这张图片属于这个类,那么权值是正数。

同时需要加入一个额外的偏置量(bias),因为输入往往会带有一些无关的干扰量。因此对于给定的输入图片 x,它代表数字 i 的判定依据特征可以表示为

$$\text{feature}_i = \sum_j W_{i,j} x_j + b_i$$

x_j 表示图片 x 中的第 j 个像素,$W_{i,j}$ 代表第 j 个像素对应的权重,b_i 代表数字 i 的偏置。$\sum_j W_{i,j} x_j$ 表示对图片中所有像素与对应权值相乘后求和。使用 softmax()函数可以将这些特征转换成概率:

$$y_i = \text{softmax}(\text{feature}_i)$$

这里的 softmax()可以看成是一个激活(activation)函数,把定义的线性函数的输出转换成关于 10 个数字类的概率分布。因此,对于一张给定的图片,它与 0～9 的吻合度可以用 softmax()函数转换成一个概率值。

对于 MNIST 数据集分类,softmax()函数可以定义为

$$\text{softmax}(x_i) = \text{normlize}(\exp(x_i)) = \frac{\exp(x_i)}{\sum_{k=0}^{9} \exp(x_k)}$$

图 3-4 是 3×3 网络模型结构,可以看出 Softmax Regression 模型是一个没有隐形层的最浅全链接神经网络。根据原理,可以很容易地将该模型扩展为适合 0～9 数字识别的 784×10 网络结构。图 3-5 为图 3-4 网络模型的数学表达,图 3-6 将数学表达以矩阵形式表示。

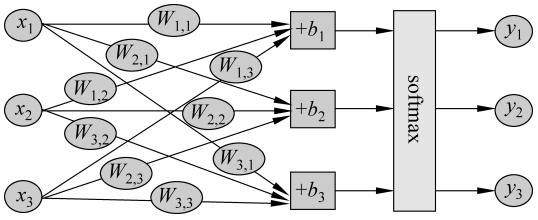


图 3-4　3×3 网络模型结构

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_1 + W_{1,3}x_1 + b_1 \\ W_{2,1}x_2 + W_{2,2}x_2 + W_{2,3}x_2 + b_2 \\ W_{3,1}x_3 + W_{3,2}x_3 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

图 3-5　3×3 网络模型的数学表达

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

图 3-6　3×3 网络模型的数学表达(矩阵形式)

根据图 3-6，可以分别将输入、输出、权值和偏置用向量的形式表示，则 Softmax Regression 模型的矩阵表达式为

$$y = \text{softmax}(Wx + b)$$

3.2.3 MNIST 数据识别的 Softmax Regression 神经网络模型

1. Softmax Regression 神经网络模型训练与测试的 TensorFlow 实现

按照第 2 章的安装步骤完成 TensorFlow 安装后，在路径 /home/tensorflow/lib/python2.7/site-packages/tensorflow/examples/tutorials/mnist/下有 MNIST 模型的例程。本教材不使用该例程，只使用例程中的个别文件。

(1) MNIST 数据集下载。

下载地址为 http://yann.lecun.com/exdb/mnist/(在网页地址栏中应严格按照下载地址格式输入网址)。

```
train-images-idx3-ubyte.gz:training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)
```

(2) 在 Ubuntu 系统下，在/home/tensorflow/文件夹下新建 design 文件夹(若文件夹已存在，则省略该操作)，在 design 文件夹下新建文件夹 my_mnist，在 my_mnist 文件夹下新建文件夹 MNIST_data。将下载的包含 4 个压缩包的 MNIST 数据集复制到 MNIST_data 文件夹中。

(3) 将/home/tensorflow/lib/python2.7/site-packages/tensorflow/examples/tutorials/mnist/中的 input_data.py 复制到/home/tensorflow/design/my_mnist/目录下。

(4) 在/home/tensorflow/design/my_mnist/目录下新建空白文档，并命名为 my_mnist_simple.py，输入以下代码。

```
#coding: utf-8
import tensorflow as tf
import input_data
import pylab

#载入数据集
mnist=input_data.read_data_sets("MNIST_data",one_hot=True)
#每个批次的大小
batch_size=100
#计算一共有多少个批次
n_batch=mnist.train.num_examples // batch_size
#================定义计算图=======================
#定义两个 placeholder
x=tf.placeholder(tf.float32,[None,784])
```

```
y=tf.placeholder(tf.float32,[None,10])
#创建一个简单的神经网络
W=tf.Variable(tf.zeros([784,10]))
b=tf.Variable(tf.zeros([10]))
prediction=tf.nn.softmax(tf.matmul(x,W)+b)
#使用交叉熵,梯度下降更有效
loss=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=prediction))
#使用梯度下降法
train_step=tf.train.GradientDescentOptimizer(0.2).minimize(loss)
#初始化变量
init=tf.global_variables_initializer()
#结果存放在一个布尔型列表中
correct_prediction=tf.equal(tf.argmax(y,1),tf.argmax(prediction,1))
#argmax 返回一维张量中最大值所在的位置
#求准确率
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
#=====================定义计算图结束=========================
#=====================运行计算图=============================
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(101):
          for batch in range(n_batch):
            batch_xs,batch_ys=  mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys})
            train_acc=sess.run(accuracy,feed_dict=
                      {x:mnist.train.images,y:mnist.train.labels})
          print("Iter "+str(epoch)+",Training Accuracy "+str(train_acc))
          test_acc=sess.run(accuracy,feed_dict=
                      {x:mnist.test.images,y:mnist.test.labels})
     print("Testing Accuracy "+str(test_acc))
#====================计算图运行结束=========================
```

(5) 安装 pylab 和 python-tk。

```
python -m pip install matplotlib
apt-get install python-tk
```

(6) 运行 my_mnist_simple.py。

```
(tensorflow)$ python my_mnist_simple.py
```

运行结果如图 3-7 所示。

```
Iter 80,Training Accuracy 0.9323636
Iter 81,Training Accuracy 0.93203634
Iter 82,Training Accuracy 0.9322182
Iter 83,Training Accuracy 0.9324
Iter 84,Training Accuracy 0.9326182
Iter 85,Training Accuracy 0.9326
Iter 86,Training Accuracy 0.93274546
Iter 87,Training Accuracy 0.9325455
Iter 88,Training Accuracy 0.93283635
Iter 89,Training Accuracy 0.93314546
Iter 90,Training Accuracy 0.93307275
Iter 91,Training Accuracy 0.9333636
Iter 92,Training Accuracy 0.9332727
Iter 93,Training Accuracy 0.9336
Iter 94,Training Accuracy 0.93341815
Iter 95,Training Accuracy 0.93347275
Iter 96,Training Accuracy 0.9336182
Iter 97,Training Accuracy 0.9337636
Iter 98,Training Accuracy 0.93381816
Iter 99,Training Accuracy 0.93396366
Iter 100,Training Accuracy 0.93389094
Testing Accuracy 0.9292
```

图 3-7　Softmax Regression 神经网络模型的运行结果

2. 代码解读

(1) ＃coding：utf-8。

当 py 文件中含有中文字符时，需要在 py 文件的开始执行＃coding：utf-8 命令。

(2) import tensorflow as tf。

在 Python 中采用 import tensorflow as tf 的形式载入 TensofFlow，这样可以使用 tf 代替 tensorflow 作为模块名称，使整个程序更加简洁。

(3) import input_data。

导入 input_data 模块，该模块中定义了读取数据的功能。

(4) import pylab。

导入 pylab 模块。

(5) mnist＝input_data.read_data_sets("MNIST_data",one_hot＝True)。

载入 MNIST 数据集，MNIST_data 为 MNIST 数据集所在的路径，one_hot＝True 表示使用 one_hot 编码格式。

(6) batch_size＝100。

定义每批次的训练样本数为 100。

(7) n_batch＝mnist.train.num_examples// batch_size。

根据训练样本集和每批次样本数量计算批次数量。mnist.train.num_examples＝55000，“//”代表除法运算符号。

(8) x＝tf.placeholder(tf.float32,[None,784])。

为样本图片数据创建一个 placeholder，即输入数据的地方，第一个参数代表数据类型，第二个参数代表 Tensor 的 shape，即数据的尺寸，None 代表不限输入样本的数量，784 表示每个样本是一个 784 维的向量。

(9) y＝tf.placeholder(tf.float32,[None,10])。

为样本标签数据创建一个 placeholder。

(10) W=tf.Variable(tf.zeros([784,10]))。

为神经网络的权值创建一个 Variable,尺寸为 784×10,784 对应输入图片的特征维数,10 对应 10 类数字。Variable 在模型训练迭代中一直存在并在每轮迭代中进行更新。

(11) b=tf.Variable(tf.zeros([10]))。

为神经网络的偏置创建一个 Variable,尺寸为 10,10 对应 10 类数字。

(12) prediction=tf.nn.softmax(tf.matmul(x,W)+b)。

tf.nn.softmax 定义了 Softmax Regression 模型,softmax 是 tf.nn 下面的一个函数。tf.nn 包含大量神经网络的组件,包括卷积操作 conv、池化操作 pooling、归一化、loss、分类操作等。tf.matmul 是 TensorFlow 中的矩阵乘法函数。

(13) loss=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=prediction))。

为了训练模型,定义一个 loss function (损失函数)描述模型对问题的分类精度,loss 越小,代表模型的分类效果与真实值的偏差越小,即模型越准确。训练的目的是不断将这个 loss 减小,直至达到一个全局最优解或局部最优解。

对于多分类问题,通常使用 cross-entropy 作为 loss function,定义表达式如下:$H_{y'}(y)=-\sum_i y'_i \log(y_i)$,其中 y 是预测的概率分布,y' 是真实的概率分布(即 label 的 one-hot 编码),通常可以用该 loss function 判断模型对真实概率分布估计的准确程度。注意:在本例程中,y 代表真实的概率分布,prediction 代表预测的概率分布,因此 loss function 的表达式应修改为 $H_y(\text{prediction})=-\sum_i y_i \log(\text{prediction}_i)$。

tf.nn.softmax_cross_entropy_with_logits 函数所做的操作是将每个样本真实标签的 one-hot 向量与其预测概率的对数相乘,将结果按元素取负号,然后按样本将元素逐行相加。

tf.reduce_mean 函数是求均值。

只要定义 loss,训练时就会自动求导并进行梯度下降,完成对 Softmax Regression 模型参数的自动学习。

(14) train_step=tf.train.GradientDescentOptimizer(0.2).minimize(loss)。

采用随机梯度下降 SGD(Stochastic Gradient Descent)优化算法,定义优化算法后,TensorFlow 就可以根据定义的整个计算图(代码中定义的公式可以自动构成计算图)自动求导,并根据反向传播算法进行训练,在每一轮迭代时更新参数以减小 loss。

tf.train.GradientDescentOptimizer 是 TensorFlow 提供的一个封装好的优化器,自动添加许多运算操实现反向传播和梯度下降,用户只需要在每轮迭代时 feed 数据即可。在本例程中,0.2 为学习速率,优化目标设置为 loss,train_step 为定义的训练操作。

(15) init=tf.global_variables_initializer()。

定义初始化操作 init,运行该操作可以对程序中的所有参数进行初始化。tf.global_variables_initializer()为 TensorFlow 的全局参数初始化函数。

(16) correct_prediction=tf.equal(tf.argmax(y,1),tf.argmax(prediction,1))。

tf.argmax 是从一个 tensor 中寻找最大值的序号,tf.argmax(y,1)得到样本真实数字

的类别，tf.argmax(prediction,1) 得到各个预测的数字中概率最大的那一个，tf.equal 判断 tf.argmax(y,1) 和 tf.argmax(prediction,1) 是否相等，当二者相等时，correct_prediction 为真，否则 correct_prediction 为假。

(17) accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))。

定义全部样本预测的准确率。tf.cast 实现类型转换，将 correct_prediction 由 bool 类型转换为 float32 类型。

至此，我们使用 TensorFlow 实现了一个简单的机器学习算法模型 Softmax Regression，(1)～(17)完成了计算图的定义。接下来就是运行计算图，进行迭代训练和测试。

(18) with tf.Session() as sess。

创建一个名为 sess 的会话，并通过 Python 中的上下文管理这个会话。通过 Python 上下文管理机制，只要将所有的计算放在 with 内部即可，当上下文管理器退出时，会自动释放所有的资源，这样既解决了异常退出时资源释放的问题，又解决了忘记调用 Session.close 函数而产生的资源泄露。

(19) sess.run(init)。

使用会话执行 init 操作，实现所有参数的初始化。

(20) for epoch in range(101)。

for 循环控制语句，epoch 为循环变量，取值范围为 0～100。

(21) for batch in range(n_batch)。

for 循环控制语句，batch 为循环变量，取值范围为 0～n_batch－1。

(22) batch_xs,batch_ys=mnist.train.next_batch(batch_size)。

每批次从训练集中一次性提取 batch_size 张图片和对应的标签。将图片赋值给 batch_xs，将标签赋值给 batch_ys。

(23) sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys})。

使用会话执行 train_step 操作，即 train_step 是运行结果的输出张量。输入数据 x 为 batch_xs，y 为 batch_ys。该代码的作用是实现神经网络训练，不断调整网络参数，以使 loss 最小。

(24) train_acc=sess.run(accuracy,feed_dict={x:mnist.train.images,y:mnist.train.labels})。

使用会话执行 accuracy 操作，即 accuracy 是运行结果的输出张量。输入数据 x 为 mnist.train.images，y 为 mnist.train.labels。该代码的作用是用训练样本集验证当前网络参数的识别准确率。

(25) print("Iter"+str(epoch)+",Training Accuracy"+str(train_acc))。

打印每次 epoch 的训练样本的识别结果信息，需要将打印的参数用 str 转换为 string 类型。

(26) test_acc=sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels})。

使用会话执行 accuracy 操作，即 accuracy 是运行结果的输出张量。输入数据 x 为

mnist.test.images，y 为 mnist.test.labels。该代码的作用是用测试样本集验证当前网络参数的识别准确率。

(27) print("Testing Accuracy"+str(test_acc))。

打印测试样本的识别结果信息，需要将打印的参数用 str 转换为 string 类型。

注意：若提示 unexpected indent 错误信息，则说明代码中行之间的缩进不一致。

3.2.4 MNIST 数据识别的卷积神经网络模型

卷积神经网络(convolutional neural networks，CNN)已成为众多科学领域的研究热点，特别是在模式分类领域，由于卷积神经网络避免了对图像复杂的前期预处理，可以直接输入原始图像，因此得到了更为广泛的应用。

1. 卷积神经网络简介

局部感知、权值共享和池化是卷积神经网络的三个重要特征。

1) 局部感知

当图像的尺寸很大时，全连接神经网络的参数将变得很多，从而导致计算量非常大。为了降低计算量，卷积神经网络采用局部感知的方法，即每个神经元只感受局部的图像区域，将这些不同局部的神经元综合起来就可以得到全局信息，如图 3-8 所示。

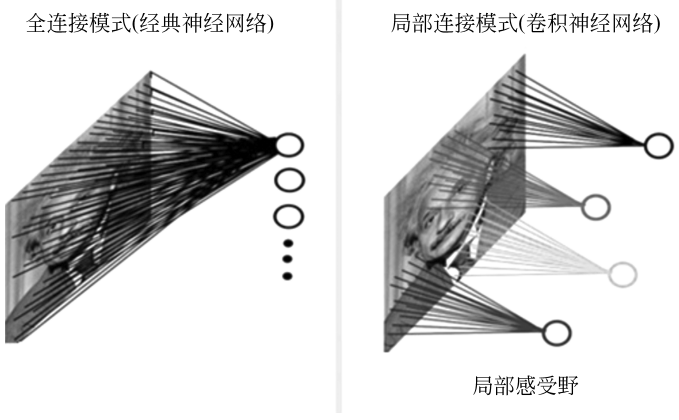


图 3-8 全连接网络全面感知与 CNN 局部感知的对比

感受野(receptive field)是 CNN 每一层输出的特征图(feature map)上每个像素点(神经元)在该层输入图像上映射的区域大小。感受野也可以认为是特征图中的神经元“看到的”输入区域，特征图上某个神经元的计算只受该神经元所看到的输入区域的影响，与区域之外的内容没有关系。图 3-9 展示了卷积神经网络中的输入图像、卷积核、感受野、特征图等基本术语，以及输入图像与卷积和进行卷积操作得到特征图的基本原理。图 3-9 中感受野的尺寸为 3×3，与卷积核尺寸相同，感受野与卷积核对应位置的元素相乘后累加，将得到新的元素构成特征图。

特征图的大小(卷积特征)由 3 个参数控制：深度(depth)、步长(stride)、填充

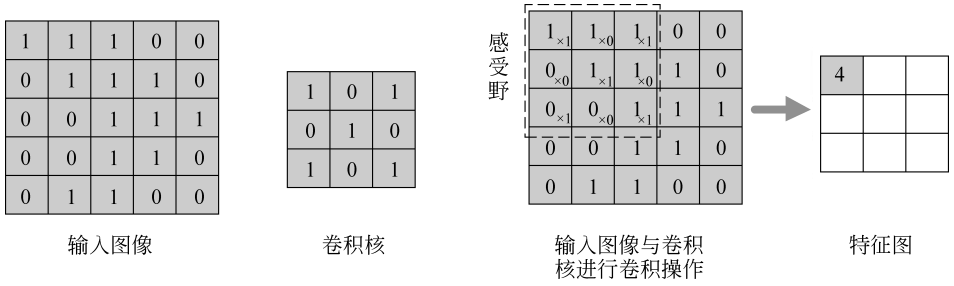


图 3-9　CNN 中的基本术语图示

(padding)。深度是指卷积操作所需的滤波器个数。步长是指每次卷积运算结束后至下一次卷积运算开始前,卷积核在输入图像上滑动的像素数。当步长为 n($n \geqslant 1$ 的整数)时,卷积核每滑动一次,新的感受野与前一个感受野相比位置向右平移 n 个像素单位,若本行结束,则向下平移 n 个像素单位,从最左侧重复滑动操作。填充可以解决图像边界卷积造成图像信息丢失的问题,主要有 3 种方法:Same Padding、Valid Padding 和自定义 Padding。Same Padding 根据卷积核的大小对输入图像进行边界补充,一般填充 0 值,以使卷积后得到的特征图与输入图像的尺寸相同;Valid Padding 保持输入图像尺寸不变,即不进行填充;自定义 Padding 与生成特征图尺寸的计算表达式为

$$\text{output}_h = (\text{input}_h + 2 \times \text{padding}_h - \text{kernel}_h)/\text{stride} + 1 \tag{3-1}$$

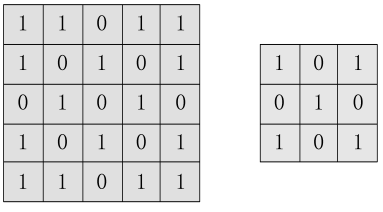
$$\text{output}_w = (\text{input}_w + 2 \times \text{padding}_w - \text{kernel}_w)/\text{stride} + 1 \tag{3-2}$$

式中,input、output 分别为输入图像和特征图的尺寸,kernel 为卷积核的尺寸,stride 为步长,padding 为边界填充尺寸,h、w 分别为输入图像的长和宽。

图 3-10 为 Valid Padding,stride=1、Valid Padding,stride=2 以及 Same Padding,stride=1 的示例。读者可以根据式(3-1)和式(3-2),自行推导 Same Padding,stride=2 时的情况。

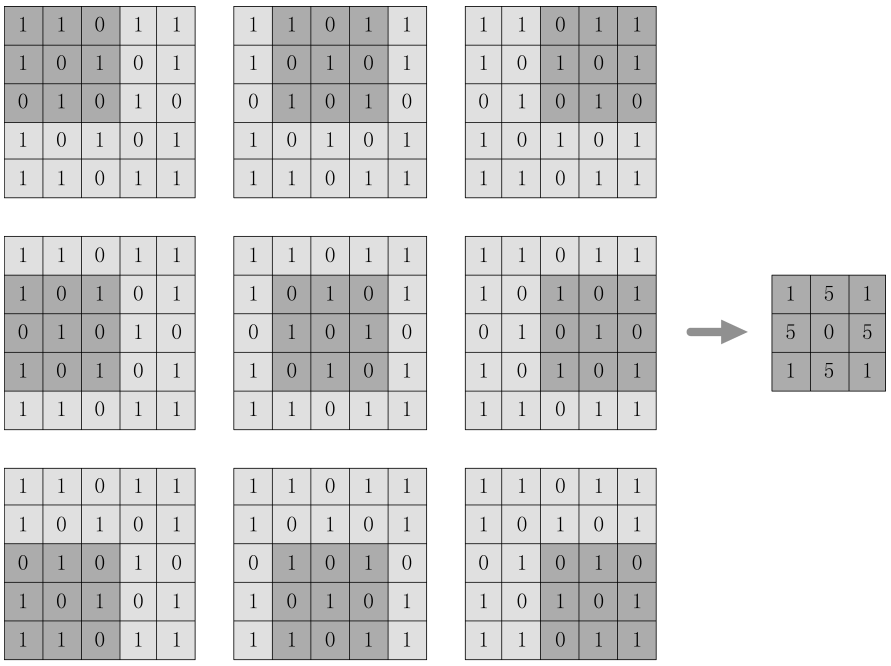
2) 权值共享

卷积核的局部感知机制将输入图像划分为不同区域(不同的感受野),但不同的感受野与具有相同数值的卷积核进行特征图计算,即图中每个感受野会被同样的卷积核处理。卷积核中的数值称作权值(权重),因此称为权值共享。如图 3-11 所示,输入图像的不同区域用同一个卷积核进行卷积运算可以得到特征图输出。

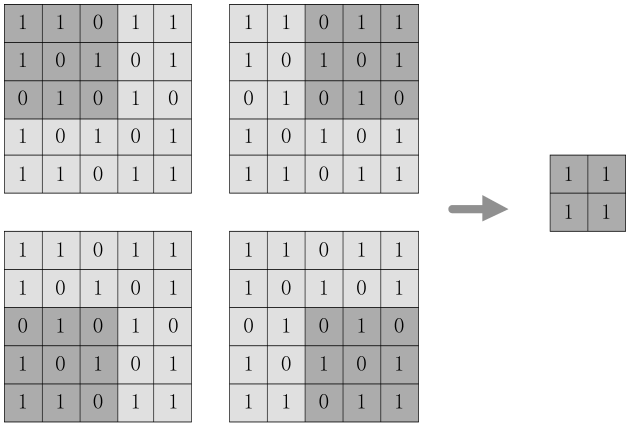


(a) 输入图像和卷积核

图 3-10　padding 模式和步长应用示例

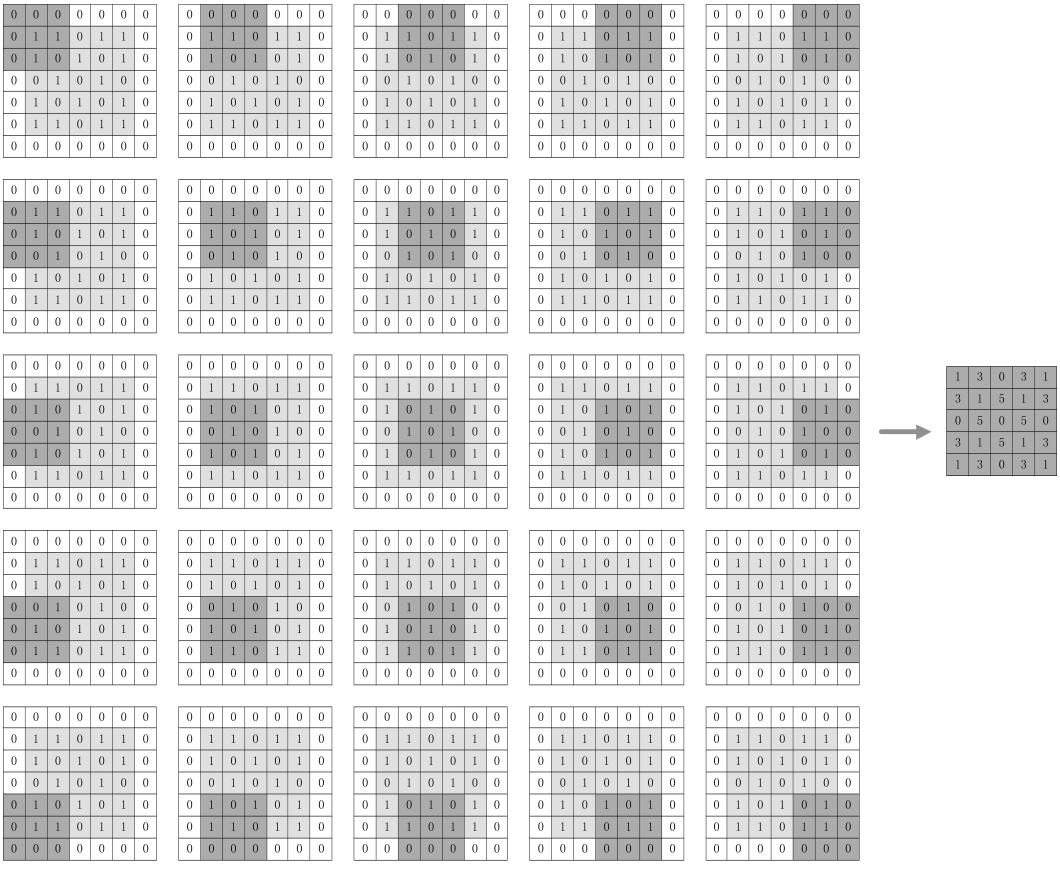


(b) Valid Padding, stride=1



(c) Valid Padding, stride=2

图 3-10 （续）



(d) Same Padding, stride=1

图 3-10　（续）

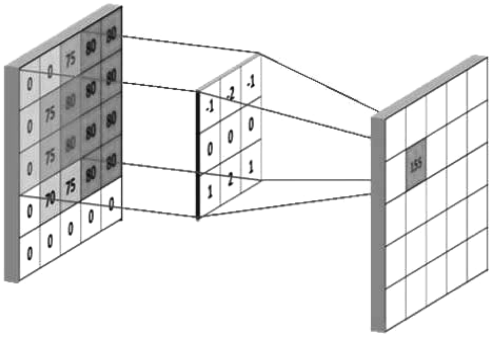


图 3-11　CNN 中的权值共享

3）池化

池化也称下采样，是指将特征图中的若干元素根据池化规则合并为一个元素的过程。池化可以看作是池化规则对特征图的滑动滤波，如图 3-12 所示。基本池化规则主要有两

种：取最大值和求平均值。因此池化方式也有两种：最大池化和均值池化。图 3-13 为最大池化和平均池化的例子。

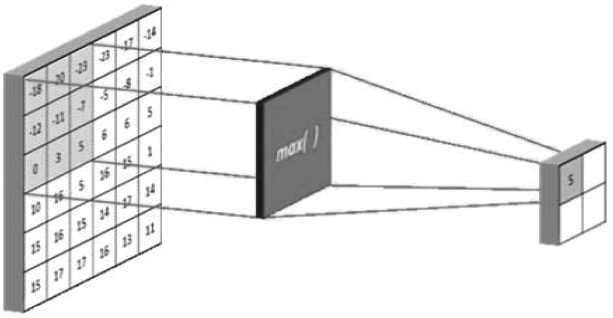
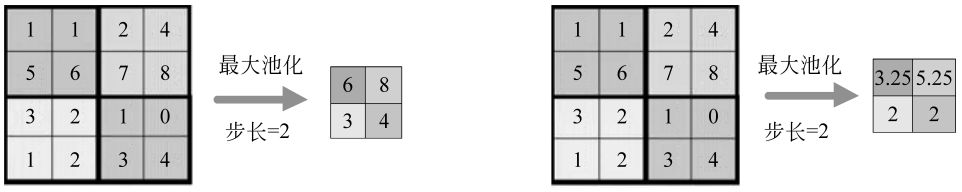


图 3-12　池化操作原理



(a) 最大池化　　(b) 平均池化

图 3-13　池化示例

2. 卷积神经网络的 TensorFlow 实现

1）卷积神经网络结构

两个卷积层＋池化层，最后接上两个全连接层。第一层卷积使用 32 个 3×3×1 的卷积核，步长为 1，边界处理方式为 same padding，即卷积的输入和输出保持相同的尺寸。激活函数为 ReLU（Rectified Linear Unit），后接一个 2×2 的池化层，方式为最大池化。ReLU 的数学表达如式(3-3) 所示，图 3-14 给出了 ReLU 函数的曲线表示。

$$f(x)=\max(x,0) \tag{3-3}$$

第二层卷积使用 50 个 3×3×32 的卷积核，步长为 1，边界处理方式为 same padding，激活函数为 ReLU，后接一个 2×2 的池化层，方式为最大池化。

第一层全连接层使用 1024 个神经元，激活函数依然是 ReLU。

第二层全连接层使用 10 个神经元，激活函数为 softmax，用于输出结果。

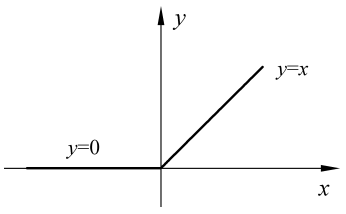


图 3-14　ReLU 激活函数

2）TensorFlow 代码

(1) 在/home/tensorflow/design/my_mnist/目录下新建空白文档，命名为 my_mnist_cnn.py，输入以下代码：

```
#coding=utf-8
import input_data
import tensorflow as tf
import pylab
#读取数据
mnist=input_data.read_data_sets('MNIST_data', one_hot=True)
#每个批次的大小
batch_size=100
#计算一共有多少个批次
n_batch=mnist.train.num_examples // batch_size
#===============构建 CNN 网络结构开始==============
#自定义卷积函数
def conv2d(x,w):
    return tf.nn.conv2d(x,w,strides=[1,1,1,1],padding='SAME')
#自定义池化函数
def max_pool_2_2(x):
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')
#设置占位符,尺寸为样本输入和输出的尺寸
x=tf.placeholder(tf.float32,[None,784])
y_=tf.placeholder(tf.float32,[None,10])
x_img=tf.reshape(x,[-1,28,28,1])
#设置第一个卷积层和池化层
w_conv1=tf.Variable(tf.truncated_normal([3,3,1,32],stddev=0.1))
#32 个 3×3×1 的卷积核
b_conv1=tf.Variable(tf.constant(0.1,shape=[32])) #32 个通道
h_conv1=tf.nn.relu(conv2d(x_img,w_conv1)+b_conv1)
h_pool1=max_pool_2_2(h_conv1)
#设置第二个卷积层和池化层
w_conv2=tf.Variable(tf.truncated_normal([3,3,32,50],stddev=0.1))
#50 个 3×3×32 的卷积核
b_conv2=tf.Variable(tf.constant(0.1,shape=[50])) #50 个通道
h_conv2=tf.nn.relu(conv2d(h_pool1,w_conv2)+b_conv2)
h_pool2=max_pool_2_2(h_conv2)
#设置第一个全连接层
w_fc1=tf.Variable(tf.truncated_normal([7*7*50,1024],stddev=0.1))
b_fc1=tf.Variable(tf.constant(0.1,shape=[1024]))
h_pool2_flat=tf.reshape(h_pool2,[-1,7*7*50])
h_fc1=tf.nn.relu(tf.matmul(h_pool2_flat,w_fc1)+b_fc1)
h_fc1_drop=tf.nn.dropout(h_fc1,0.5)
#设置第二个全连接层
w_fc2=tf.Variable(tf.truncated_normal([1024,10],stddev=0.1))
b_fc2=tf.Variable(tf.constant(0.1,shape=[10]))
y_out=tf.nn.softmax(tf.matmul(h_fc1_drop,w_fc2)+b_fc2)
```

```
#==============构建 CNN 网络结构结束==============
#使用交叉熵
loss=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_,
logits=y_out))
#使用梯度下降法
train_step=tf.train.GradientDescentOptimizer(0.2).minimize(loss)
#定义初始化操作
init=tf.global_variables_initializer()
#建立正确率计算表达式
correct_prediction=tf.equal(tf.argmax(y_out,1),tf.argmax(y_,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
#开始训练
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(101):
        for batch in range(n_batch):
            batch_xs,batch_ys=mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y_:batch_ys})
            train_acc=sess.run(accuracy,feed_dict={x:mnist.train.images,
                      y_:mnist.train.labels})
        print("Iter "+str(epoch)+",Training Accuracy "+str(train_acc))
    test_acc=sess.run(accuracy,feed_dict={x:mnist.test.images,y_:mnist.
    test.labels})
    print("Testing Accuracy "+str(test_acc))
```

(2) 运行 my_mnist_cnn.py,在目录/home/tensorflow/design/my_mnist/下运行 my_mnist_cnn.py。

```
(tensorflow)$ python my_mnist_cnn.py
```

图 3-15 为卷积神经网络模型的运行结果,显示测试结果为 test_accuracy=0.9907。

```
Iter 80,Training Accuracy 0.9976
Iter 81,Training Accuracy 0.99774545
Iter 82,Training Accuracy 0.99756366
Iter 83,Training Accuracy 0.99783635
Iter 84,Training Accuracy 0.99767274
Iter 85,Training Accuracy 0.99783635
Iter 86,Training Accuracy 0.9976909
Iter 87,Training Accuracy 0.9979454
Iter 88,Training Accuracy 0.99785453
Iter 89,Training Accuracy 0.9976364
Iter 90,Training Accuracy 0.9975455
Iter 91,Training Accuracy 0.9980364
Iter 92,Training Accuracy 0.9980909
Iter 93,Training Accuracy 0.99807274
Iter 94,Training Accuracy 0.9978909
Iter 95,Training Accuracy 0.99774545
Iter 96,Training Accuracy 0.9980182
Iter 97,Training Accuracy 0.9982182
Iter 98,Training Accuracy 0.9978182
Iter 99,Training Accuracy 0.9982182
Iter 100,Training Accuracy 0.9981818
Testing Accuracy 0.9907
```

图 3-15　卷积神经网络模型的运行结果

对比图 3-15 和图 3-7 可以发现，与 Softmax Regression 神经网络模型进行比较，采用结构更复杂的卷积神经网络可以实现更高的识别正确率。

3. 代码解读

本部分将对 my_mnist_cnn.py 代码中出现的新的 API 函数进行介绍。未解读的代码请参考 3.2.3 节中的代码解读中对 my_mnist_simple.py 的分析。

(1) tf.nn.conv2d 是 TensorFlow 专门用作卷积运算的 API 函数，该函数的原型为：

```
tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, name=
None)
```

该 API 函数各参数的意义如下。

① input：输入图像或数据，是一个 Tensor，shape=(4,)，shape 格式为 1×4，4 个元素为[batch，in_height，in_width，in_channels]，具体含义是[一个 batch 的图片数量，图片高度，图片宽度，图像通道数]，dtype 为 float32 或 float64。

② filter：CNN 的卷积核，也是一个 Tensor，shape=(4,)，shape 格式为 1×4，4 个元素为[filter_height，filter_width，in_channels，out_channels]，具体含义是[卷积核高度，卷积核宽度，图像通道数，卷积核个数]，dtype 与参数 input 相同。第 3 个元素 in_channels 与 input 的第 4 个元素 in_channels 相同。

③ strides：卷积运算时在图像每一维的步长，是一个一维向量，长度为 4，即含有 4 个元素，格式为[1，横向步长，纵向步长，1]。

④ padding：只能是 SAME 和 VALID 其中之一。

⑤ use_cudnn_on_gpu：bool 类型，指是否使用 cudnn 加速，默认为 true。

⑥ name：指定本次调用 API 函数操作的名字。

该 API 会返回一个 Tensor，即卷积神经网络中的 feature map，shape 格式为[batch，height，width，channels]。

(2) tf.nn.max_pool 是 TensorFlow 专门用作池化运算的 API 函数，该函数的原型为：

```
tf.nn.max_pool(value, ksize, strides, padding, name=None)
```

该 API 函数各参数的意义如下。

① value：池化输入，一般池化层接在卷积层后面，所以输入通常是 feature map，shape 格式依然是[batch，height，width，channels]。

② ksize：池化窗口尺寸，是一个含有 4 个元素的一维向量，格式为[1，height，width，1]。首尾两个元素为 1，代表不在 batch 和 channels 上做池化。本例中 ksize 取值为[1，2，2，1]，代表横向两个和纵向两个共 4 个特征图中数据为一组进行池化。

③ strides：池化窗口在特征图的每一个维度上滑动的步长，格式为[1，stride，stride，1]。

④ padding：和卷积类似，可以取 VALID 或者 SAME。

⑤ name：指定本次调用 API 函数操作的名字。

该 API 返回一个 Tensor，类型与输入相同，shape 格式为[batch，height，width，

channels]。

(3) tf.reshape 是 TensorFlow 中用来改变张量 shape 的 API 函数,该函数的原型为:

```
tf.reshape(tensor,shape,name=None)
```

该 API 函数各参数的意义如下。

tensor:输入张量。

shape:变形后得到的输出张量的形状。

name:指定本次调用 API 函数操作的名字。

该 API 函数的作用是将输入 tensor 变换为参数 shape 形式,参数 shape 为一个列表形式,如本例中的[-1,28,28,1]。-1 代表不用指定这一维的大小,函数会自动进行计算,但是列表中只能存在一个-1。如果 batch 为 1,则每次输入一张图像,输入 x 的 shape 为[1,784],则 x_img 的 shape 为[1,28,28,1]。如果 batch 为 100,则每次输入 100 张图像,输入 x 的 shape 为[100,784],则 x_img 的 shape 为[100,28,28,1]。

(4) tf.truncated_normal 是 TensorFlow 中用来生成截断正态分布随机数据的 API 函数,该函数的原型为:

```
tf.truncated_normal(shape, mean=0.0, stddev=1.0,dtype=tf.float32,seed=
None,name=None)
```

该 API 函数各参数的意义如下。

① shape:表示生成张量的维度。

② mean:均值,默认值为 0.0。

③ stddev:标准差,默认值为 1.0。

④ dtype:数据类型,默认为 tf.float32。

⑤ seed:随机数种子。

⑥ name:指定本次调用 API 函数操作的名字。

该 API 函数根据设定的均值和标准差产生正态分布的随机数据,如果产生正态分布的数值与均值的差值大于标准差的 2 倍,则重新生成,因此称为截断的正态分布生成函数,该函数可保证产生的随机数与均值的差距不会超过标准差的 2 倍。

(5) tf.nn.relu 实现 ReLU 激活函数功能。

(6) tf.nn.dropout 是 TensorFlow 中用来随机屏蔽某些神经元输出的 API 函数,该函数的原型为:

```
tf.nn.dropout(x, keep_prob, noise_shape=None,seed=None,name=None)
```

该 API 函数的各参数意义如下。

① x:输入,输入 tensor。

② keep_prob:float 类型,每个元素被保留下来的概率,设置神经元被选中的概率。

③ noise_shape:一个一维的 int32 张量,代表随机产生“保留/丢弃”标志的 shape。

④ seed:整型变量,随机数种子。

⑤ name：指定本次调用 API 函数操作的名字。

该 API 函数可以在不同的训练过程中按设置的概率随机屏蔽一部分神经元的输出，即让某些神经元的激活值以一定的概率 1-keep_prob 让其停止工作。

各个列表中的取值，与输入图像尺寸、卷积核参数、池化参数等有关，读者可自行推算列表参数中的取值依据。

3.3　MNIST 数据集转换

之所以进行数据集转换，是因为神经网络需要处理的样本集可能是多种多样的，有的是一维时间序列数据，有的是二维图像数据。为了能够实现不同格式样本集的处理，需要将样本集转换为神经网络可以识别的数据格式。通过学习数据集转换，读者可以转换自己采集的样本集的数据格式，然后用相同的方法采用神经网络进行训练和测试。

3.3.1　将数据集转换为以 txt 文件保存的数据

本例将 101 张 MNIST 测试样本和对应的标签转换为 txt 文档。

(1) 在/home/tensorflow/design/my_mnist/目录下新建空白文档，命名为 txt_file_gen.py，输入以下代码。

```
#coding: utf-8
import tensorflow as tf
import input_data
import numpy as np
#载入数据集
mnist=input_data.read_data_sets("MNIST_data",one_hot=True)
with tf.Session() as sess:
    for i in range(101):
        img_in_i=mnist.test.images[i]
        tag_i=np.argmax(mnist.test.labels[i])
    np.savetxt('/home/tensorflow/design/my_mnist/mnist_txt/mnist_img_txt/
img_%d.txt'%i, img_in_i.reshape(-1), fmt="%.31f",delimiter=",")

    np.savetxt('/home/tensorflow/design/my_mnist/mnist_txt/mnist_lab_txt/
img_lab_%d.txt'%i, tag_i.reshape(-1), fmt="%.31f",delimiter=",")
print("text write was sucessful")
```

(2) 在/home/tensorflow/design/my_mnist/目录下新建文件夹 mnist_txt，并在该文件夹下新建两个文件夹 mnist_img_txt 和 mnist_lab_txt。

注意：必须提前建立文件夹，Python 代码不能自动生成文件夹。

(3) 运行 txt_file_gen.py。

```
(tensorflow)$ python txt_file_gen.py
```

若成功运行，则/home/tensorflow/design/my_mnist/mnist_txt/mnist_img_txt 和/home/tensorflow/design/my_mnist/mnist_txt/mnist_lab_txt 分别会有 101 个 txt 文件生成。

3.3.2 将数据集转换为以 bmp 文件保存的图片

1. 训练样本集转换

(1) 在/home/tensorflow/design/目录下新建文件夹 my_mnist_tfrecord。

(2) 将/home/tensorflow/design/my_mnist 目录下的 MNIST_data 和 input_data.py 复制到/home/tensorflow/design/my_mnist_tfrecord 文件夹。

(3) 在/home/tensorflow/design/my_mnist_tfrecord/目录下新建空白文档，命名为 mnist_bmp_train_gen.py，输入以下代码。

```
#coding: utf-8
import os
import tensorflow as tf
import input_data
from PIL import Image
#声明图片的宽和高
rows=28
cols=28
#当前路径下的保存目录
save_dir="./mnist_bmp_train"
#读入 MNIST 数据
mnist=input_data.read_data_sets("MNIST_data/", one_hot=False)
#创建会话
sess=tf.Session()
#获取图片总数
shape=sess.run(tf.shape(mnist.train.images))
images_count=shape[0]
pixels_per_image=shape[1]
#获取标签总数
shape=sess.run(tf.shape(mnist.train.labels))
labels_count=shape[0]
#要提取的图片数量
images_to_extract=shape[0]
#
labels=mnist.train.labels
#检查数据集是否符合预期格式
if (images_count==labels_count) and (shape.size==1):
    print ("数据集总共包含 %s 张图片，和 %s 个标签" %(images_count, labels_count))
    print ("每张图片包含 %s 个像素" %(pixels_per_image))
    print ("数据类型: %s" %(mnist.train.images.dtype))
```