# Tensorflow基础

## 1. 张量和图

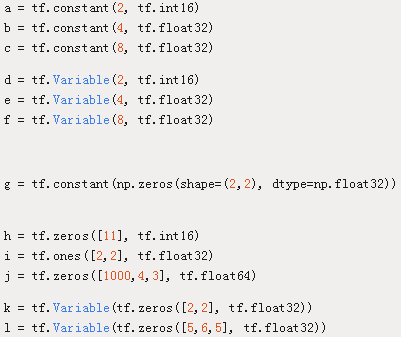
TensorFlow 是一种采用数据流图（data flow graphs），用于数值计算的开源软件库。其中 Tensor 代表传递的数据为张量（多维数组），Flow 代表使用计算图进行运算。数据流图用「结点」（nodes）和「边」（edges）组成的有向图来描述数学运算。「结点」一般用来表示施加的数学操作，但也可以表示数据输入的起点和输出的终点，或者是读取/写入持久变量（persistent variable）的终点。边表示结点之间的输入/输出关系。这些数据边可以传送维度可动态调整的多维数据数组，即张量（tensor）。

### 1.1 session()：运行图运算

在Tensorflow中，所有不同的变量和运算都是储存在计算图。所以在我们构建完模型所需要的图之后，还需要打开一个会话（Session）来运行整个计算图。

### 1.2 常量constant和变量Variable

TensorFlow 中最基本的单位是常量（Constant）、变量（Variable）和占位符（Placeholder）。常量定义后值和维度不可变，变量定义后值可变而维度不可变。在神经网络中，变量一般可作为储存权重和其他信息的矩阵，而常量可作为储存超参数或其他结构信息的变量。

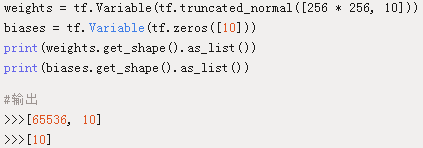


以下语句声明一个2行3列的变量矩阵，该变量的值服从标准差为1的正态分布，并随机生成。



TensorFlow还有tf.truncated\_normal()函数，即截断正态分布随机数，它只保留 [mean-2\*stddev, mean+2\*stddev] 范围内的随机数。

利用变量可以定义一层神经元个数为10，每个神经元输入为256\*256的神经网络的权重矩阵和偏置向量：



### 1.3 占位符placeholder和feed\_dict

占位符placeholder并没有初始值，它只会分配必要的内存。在会话中，占位符可以使用 feed\_dict 馈送数据。

Placeholder需要两个输入参数，第一个是数据类型，第二个为tensor的shape，如果shape中包含None，则代表该维度不限制维数。

feed\_dict 是一个字典，在字典中需要给出每一个用到的占位符的取值。在训练神经网络时需要每次提供一个批量的训练样本，如果每次迭代选取的数据要通过常量表示，那么 TensorFlow 的计算图会非常大。因为每增加一个常量，TensorFlow 都会在计算图中增加一个结点。所以说拥有几百万次迭代的神经网络会拥有极其庞大的计算图，而占位符却可以解决这一点，它只会拥有占位符这一个结点。

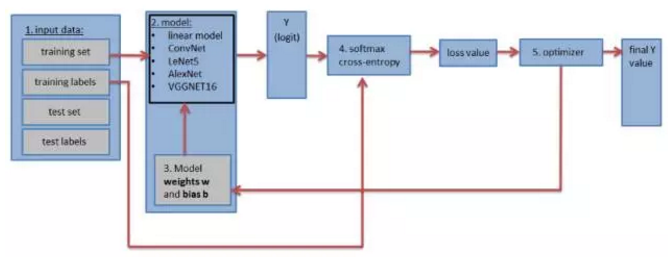
下面搭建一个三层的全连接神经网络：



其中需要注意的几个函数如 tf.nn.relu() 代表调用 ReLU 激活函数，tf.matmul() 为矩阵乘法等。tf.clip\_by\_value(yhat,1e-10,1.0) 这一语句代表的是截断 yhat 的值，因为这一语句是嵌套在 tf.log() 函数内的，所以我们需要确保 yhat 的取值不会导致对数无穷大。

tf.train.AdamOptimizer(learning\_rate).minimize(cost\_function) 是进行训练的函数，其中我们采用的是 Adam 优化算法更新权重，并且需要提供学习速率和损失函数这两个参数。

## 2. TensorFlow中的神经网络



上图所描述的图像识别流程需要包含以下几步：

* 1）输入数据集，数据集分为训练数据集和标注、测试数据集和标注（包括验证数据集和标注）。测试和验证集能赋值到 tf.constant() 中，而训练集可以导入 tf.placeholder() 中，训练集只有导入占位符我们才能在随机梯度下降中成批量地进行训练。
* 2）确定神经网络模型，该模型可以是简单的一层全连接网络或 9 层、16 层的复杂卷积网络组成。
* 3）网络定义的权重矩阵和偏置向量后需要执行初始化，每一层需要一个权重矩阵和一个偏置向量。
* 4）构建损失函数，并计算训练损失。模型会输出一个预测向量，我们可以比较预测标签和真实标签并使用交叉熵函数和 softmax 回归来确定损失值。训练损失衡量预测值和真实值之间差距，并用于更新权重矩阵。
* 5）优化器，优化器将使用计算的损失值和反向传播算法更新权重和偏置项参数。

### 2.1 创建网络的函数

在使用 TensorFlow 构建卷积网络之前，我们需要了解一下 TensorFlow 中的函数。

TensorFlow 包含很多操作和函数，很多我们需要花费大量精力完成的过程可以直接调用已封装的函数，比如说「logits = tf.matmul(tf\_train\_dataset, weights) + biases」可以由函数「logits = tf.nn.xw\_plus\_b(train\_dataset, weights, biases)」代替。

还有很多函数可以让构建不同层级的神经网络变得十分简单。例如 conv\_2d() 和 fully\_connected() 函数分别构建了卷积层和全连接层。通过这些函数，层级的数量、滤波器的大小/深度、激活函数的类型等都可以明确地作为一个参数。权重矩阵和偏置向量能自动创建，附加激活函数和 dropout 正则化层同样也能轻松构建。

如下两种方式构建卷积层+池化层，都是等价的：

|  |
| --- |
| import tensorflow as tf    w1 = tf.Variable(tf.truncated\_normal([filter\_size, filter\_size, image\_depth, filter\_depth], stddev=0.1))  b1 = tf.Variable(tf.zeros([filter\_depth]))  layer1\_conv = tf.nn.conv2d(data, w1, [1, 1, 1, 1], padding='SAME')  layer1\_relu = tf.nn.relu(layer1\_conv + b1)  layer1\_pool = tf.nn.max\_pool(layer1\_pool, [1, 2, 2, 1], [1, 2, 2, 1], padding='SAME') |
| from tflearn.layers.conv import conv\_2d, max\_pool\_2d  layer1\_conv = conv\_2d(data, filter\_depth, filter\_size, activation='relu')  layer1\_pool = max\_pool\_2d(layer1\_conv\_relu, 2, strides=2) |

2.1.1 conv2d()函数和max\_pool函数

conv2d()函数有四个参数：

1）输入图像，即一个四维张量 [batch size, image\_width, image\_height, image\_depth]

2）卷积核权重矩阵，即一个四维张量 [filter\_size, filter\_size, image\_depth, filter\_depth]

3）卷积操作时每一个维度的步幅数

4）Padding (= 'SAME' / 'VALID')

前面两个参数都是四维张量，其包括了批量输入图像的信息和卷积滤波器的权值。

第三个参数为卷积的步幅（stride），即卷积滤波器在 4 个维度中的每一次移动的距离。四个中间的

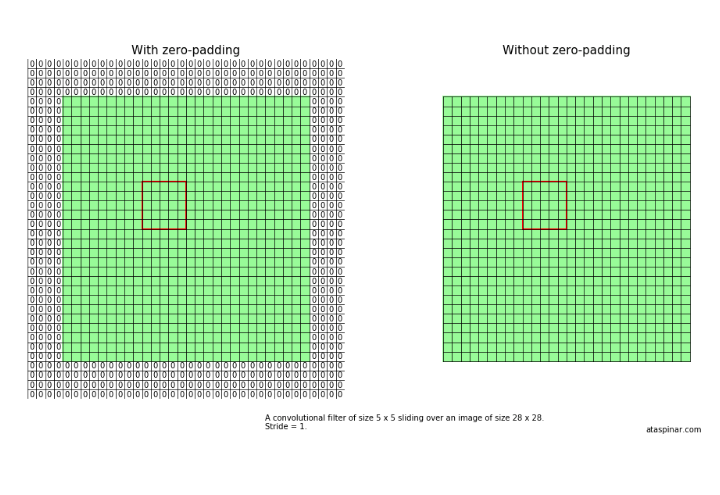
第一个维度代表着图像的批量数，这个维度肯定每次只能移动一张图片。

最后一个维度为图片深度（即色彩通道数，1 代表灰度图片，而 3 代表 RGB 图片），因为我们通常并不想跳过任何一个通道，所以这一个值也通常为 1。

第二个和第三个维度代表 X 和 Y 方向（图片宽度和高度）的步幅。如果我们希望能应用步幅参数，我们需要设定每个维度的移动步幅。例如设定步幅为 1，那么步幅参数就需要设定为 [1, 1, 1, 1]，如果我们希望在图像上移动的步幅设定为 2，步幅参数为 [1, 2, 2, 1]。

第四个参数表明 TensorFlow 是否需要使用 0 来填补图像周边，这样以确保图像输出尺寸在步幅参数设定为 1 的情况下保持不变。通过设置 padding = 'SAME'，图像会只使用 0 来填补周边（输出尺寸不变），而 padding = 'VALID'则不会使用 0。

在下图中，我们将看到两个使用卷积滤波器在图像上扫描的案例，其中滤波器的大小为 5 x 5、图像的大小为 28 x 28。左边的 Padding 参数设置为'SAME'，并且最后四行/列的信息也会包含在输出图像中。而右边 padding 设置为 'VALID'，最后四行/列是不包括在输出图像内的。



对于任意给定的步幅 S、滤波器尺寸 K、图像尺寸 W、padding 尺寸 P，输出的图像尺寸可以总结上表的规则如下：

IMG_256

Max\_pool()函数有四个参数：

第一个是输入的特征图；

第二个是池化操作的尺寸大小；

第三个是池化操作的步幅大小；

第四个是padding的方式。

其中第二个和第三个参数都是四维矩阵，其中第一维表示特征图的个数，所以一般都取1；第二维和第三维表示每个特征图的宽和高的维度，第四维表示每个特征图的通道数，一般也取1。

2.1.2 relu激活函数和偏置项

a=tf.nn.relu(tf.matmul(x,w1)+biases1)

2.1.3 softmax交叉熵函数

cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits(yhat,y)

2.1.4 均方误差损失函数mse

mse=tf.reduce\_mean(tf.square(y-yhat))

2.1.5 带L1/L2正则化的损失函数

w=tf.Variable(tf.random\_normal([2,1],stddev=1,seed=1))

yhat=tf.matmul(x,w)

Loss\_L1=tf.reduce\_mean(tf.square(y-yhat))+tf.contrib.layers.l1\_regularizer(lambda)(w)

Loss\_L2=tf.reduce\_mean(tf.square(y-yhat))+tf.contrib.layers.l2\_regularizer(lambda)(w)

2.1.6 逐步衰减学习率

#decayed\_learning\_rate=learning\_rate\*decay\_rate^(global\_steps/decay\_steps)，指数衰减函数的定义

global\_step=tf.Variable(0)

#使用exponential\_decay生成学习速率，因为staircase=tire，每100次迭代，学习率×0.96

learning\_rate=tf.train.exponential\_decay(0.1,global\_step,100,0.96,staircase=True)

#在minimize中导入global\_step将自动更新

#learning\_step=tf.train.GtadientDescentOptimizer(learning\_rate).minimize(loss\_function,global\_step=global\_step)

2.1.7 训练优化方法 SGD / Adam

（1）SGD

train\_step=tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss,global\_step=global\_step)

（2）Adam

train\_step=tf.train.AdamOptimizer(0.001).minimize(cross\_entropy)

### 2.2 其他函数

tf.variable\_scope('layer1-conv1', reuse = True)

tf.name\_scope('layer2-pool1')

tf.greater()的输入是两个张量，比较两个张量中的每一个元素，并返回比较结果（true或false的向量）。

tf.select()有三个参数，第一个参数条件为真时选择第二个参数中的值，否则选择第三个参数的值。

TensorFlow内建的运算操作

|  |  |
| --- | --- |
| 类型 | 示例（tf.\*） |
| 标量运算 | Add、Sub、Mul、Div、Exp、Log、Greater、Less、Equal |
| 向量运算 | Concat、Slice、Split、Constant、Rank、Shape、Shuffle |
| 矩阵运算 | MatMul、MatrixInverse、MatrixDeterminant |
| 带状态的运算 | Variable、Assign、AssignAdd |
| 神经网络组件 | SoftMax、Sigmoid、RELU、Convolution2D、MaxPooling |
| 存储、恢复 | Save、Restore |
| 队列及同步运算 | Enqueue、Dequeue、MutexAcquire、MutexRelease |
| 控制流 | Merge、Switch、Enter、Leave、NextIteration |

Tensorflow（简称tf）下的函数介绍

|  |  |  |
| --- | --- | --- |
| tf.nn | tf.nn.softmax | 求softmax函数 |
|  |  |
|  |  |
|  |  |
|  |  |
| tf.matmul | 矩阵乘法 | |
| tf.reduce\_mean | 求平均的作用 | |
| tf.reduce\_sum | 求和符号的作用 | |
| tf.train | tf.train.GradientDescentOptimizer().minimize() | 梯度下降求解器、学习率、优化目标 |
| tf.global\_variables | 全局参数初始化器 | |
| tf.argmax | 从一个tensor中寻找最大值的序号 | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

### 2.3 设备选择与多GPU并行

2.3.1 指定运算使用的设备

With tf.device(“/gpu:0，2”) #使用第一块GPU和第三块GPU

With tf.device(“/cpu:0”) #使用cpu运算

由于有些操作只能在CPU上进行，在Session中的allow\_soft\_placement=True，可以避免错误。

2.3.2 使用多个GPU并行

在一个循环内可以循环使用每个GPU进行运算，调用tf.get\_variable\_scope().reuse\_variables()重用参数，让所有GPU共用一个模型以及完全相同的参数。

将单个GPU产生的梯度，添加到梯度列表，最后计算平均梯度，然后更新模型参数。

这样就完成多GPU的同步训练和更新。

## 3. Keras--TensorFlow外层模块化神经网络库

崇尚极简、高度模块化的神经网络库，基于python实现，可以运行在TensorFlow和theano上。用户只需要将高级模块拼装在一起，大大降低了编程开销。

它同时支持卷积网络和循环网络，支持级联模型或任意图结构模型。Keras训练模型相比TensorFlow和theano基本没有性能损耗，只是简化了编程复杂度。可以说模型越复杂，使用keras收益越大。尤其是在高度依赖权值共享、多模型组合、多任务学习模型上。