# TensorFlow Tutorial

# 1、TensorFlow入门

## 到目前为止，我们一直在使用numpy来自己编写神经网络。现在我们将一步步的使用深度学习的框架来很容易的构建属于自己的神经网络。我们将学习TensorFlow这个框架：

## 1. 初始化变量

## 2. 建立一个会话

## 3. 训练的算法

## 4. 实现一个神经网络

使用框架编程不仅可以节省你的写代码时间，还可以让你的优化速度更快。

## 1.1导入TensorFlow库

导入了相关的库之后，我们将引导你完成不同的应用，我们现在看一下下面的计算损失的公式：



y\_hat = tf.constant(36,name="y\_hat") # 定义y\_hat为固定值36

y = tf.constant(39,name="y") # 定义y为固定值39

loss = tf.Variable((y-y\_hat)\*\*2,name="loss" ) # 为损失函数创建一个变量

init = tf.global\_variables\_initializer() # 运行之后的初始化(ession.run(init))

#损失变量将被初始化并准备计算

with tf.Session() as session: #创建一个session并打印输出

session.run(init) #初始化变量

print(session.run(loss)) #打印损失值

(tensorflow2以上版本使用with tf.compat.v1.Session() as session:)

执行结果：

9

对于Tensorflow的代码实现而言，实现代码的结构如下：

1.创建Tensorflow变量（此时，尚未直接计算）；

2.实现Tensorflow变量之间的操作定义；

3.初始化Tensorflow变量；

4.创建Session；

5.运行Session，此时，之前编写操作都会在这一步运行。

因此，当我们为损失函数创建一个变量时，我们简单地将损失定义为其他数量的函数，但没有评估它的价值。为了评估它，我们需要运行init = tf.global\_ variables\_initializer()，初始化损失变量，在最后一行我们最后能够评估损失的值并打印它的值。  
  现在让我们看一个简单的例子：

a = tf.constant(2)

b = tf.constant(10)

c = tf.multiply(a,b)

print(c)

执行结果：

Tensor("Mul:0", shape=(), dtype=int32)

正如预料中一样，我们并没有看到结果20，不过我们得到了一个Tensor类型的变量，没有维度，数字类型为int32。我们之前所做的一切都只是把这些东西放到了一个“计算图(computation graph)”中，而我们还没有开始运行这个计算图，为了实际计算这两个数字，我们需要创建一个会话并运行它：

sess = tf.Session()

print(sess.run(c))

执行结果：

20

总结一下，记得初始化变量，然后创建一个session来运行它。

接下来，我们需要了解一下占位符（placeholders）。占位符是一个对象，它的值只能在稍后指定，要指定占位符的值，可以使用一个feed字典（feed\_dict变量）来传入，接下来我们为x创建一个占位符，这将允许我们在稍后运行会话时传入一个数字。

#利用feed\_dict来改变x的值

x = tf.placeholder(tf.int64,name="x")

print(sess.run(2 \* x,feed\_dict={x:3}))

sess.close()

执行结果：

6

当我们第一次定义x时，我们不必为它指定一个值。 占位符只是一个变量，我们会在运行会话时将数据分配给它。

**1.1.1线性函数**

让我们通过计算以下等式来开始编程：*Y* = *WX* + *b*，*W*和*X*是随机矩阵，*b*是随机向量。我们计算*WX* + *b*，其中*W*、*X*和*b*是从随机正态分布中抽取的。 *W*的维度是（4,3），*X*是（3,1），*b*是（4,1）。 我们开始定义一个shape =（3,1）的常量X：

X = tf.constant(np.random.randn(3,1), name = "X")

**1.1.2计算sigmoid**

我们已经实现了线性函数，TensorFlow提供了多种常用的神经网络的函数比如tf.softmax和 tf.sigmoid。

我们将使用一个占位符变量x来完成这个练习。在运行session时，我们应该使用feed字典来传递输入z。在本练习中，必须(i)创建一个占位符x， (ii)使用tf.sigmoid来定义计算sigmoid所需要的操作符，然后(iii)运行session。

实现下面的sigmoid函数，我们会用到下面的代码：

tf.placeholder(tf.float32, name = “…”)

tf.sigmoid(…)

sess.run(…, feed\_dict = {x: z})

需要注意的是我们可以使用两种方法来创建并使用session。

方法一：

sess = tf.Session()

result = sess.run(...,feed\_dict = {...})

sess.close()

方法二：

with tf.Session as sess:

result = sess.run(...,feed\_dict = {...})

**1.1.3计算成本**

还可以使用内置函数计算神经网络的成本。因此，不需要编写代码来计算成本函数的和

for i=1…m:



实现成本函数，需要用到的是：

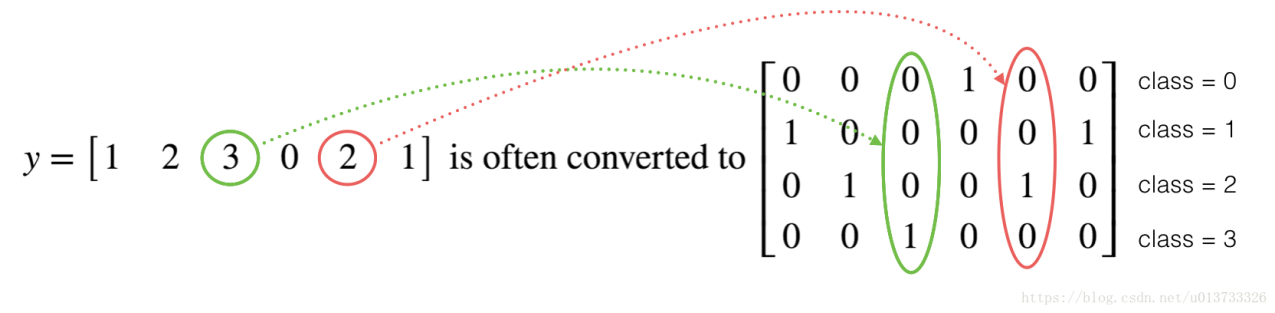
tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits = ..., labels = ...)

  你的代码应该输入z，计算sigmoid（得到 a），然后计算交叉熵成本J

1. 所有的步骤都可以通过一次调用tf.nn.sigmoid\_cross\_entropy\_with\_logits来完成。

**1.1.4使用独热编码（0、1编码）**

  很多时候在深度学习中y向量的维度是从0到C−1的，C是指分类的类别数量，如果C=4，那么对y而言你可能需要有以下的转换方式:



这叫做独热编码（“one hot” encoding），因为在转换后的表示中，每列的一个元素是“hot”（意思是设置为1）。 要在numpy中进行这种转换，您可能需要编写几行代码。在tensorflow中，只需要使用一行代码：

## tf.one\_hot(labels,depth,axis)

如果indices是一个长度为features的向量，当axis==0时输出尺寸为depth \* features；当axis==-1时输出尺寸为features \* depth；当axis = 1时，同axis = -1。

如果indices是一个尺寸为[batch，features]的矩阵，当axis==0时输出尺寸为depth \* batch \* features；当axis==-1时输出尺寸为batch \* features \* depth；当axis==1时输出尺寸为batch \* depth \* features。

## 我们要做的是取一个标签矢量和C类总数，返回一个独热编码。

## **1.1.5初始化为0和1**

## 现在我们将学习如何用0或者1初始化一个向量，我们要用到tf.ones()和tf.zeros()，给定这些函数一个维度值那么它们将会返回全是1或0的满足条件的向量/矩阵。

**2、使用TensorFlow构建你的第一个神经网络**

我们将会使用TensorFlow构建一个神经网络，需要记住的是实现模型需要做以下两个步骤：

1.创建计算图；

2.运行计算图。

我们开始一步步地走一下：

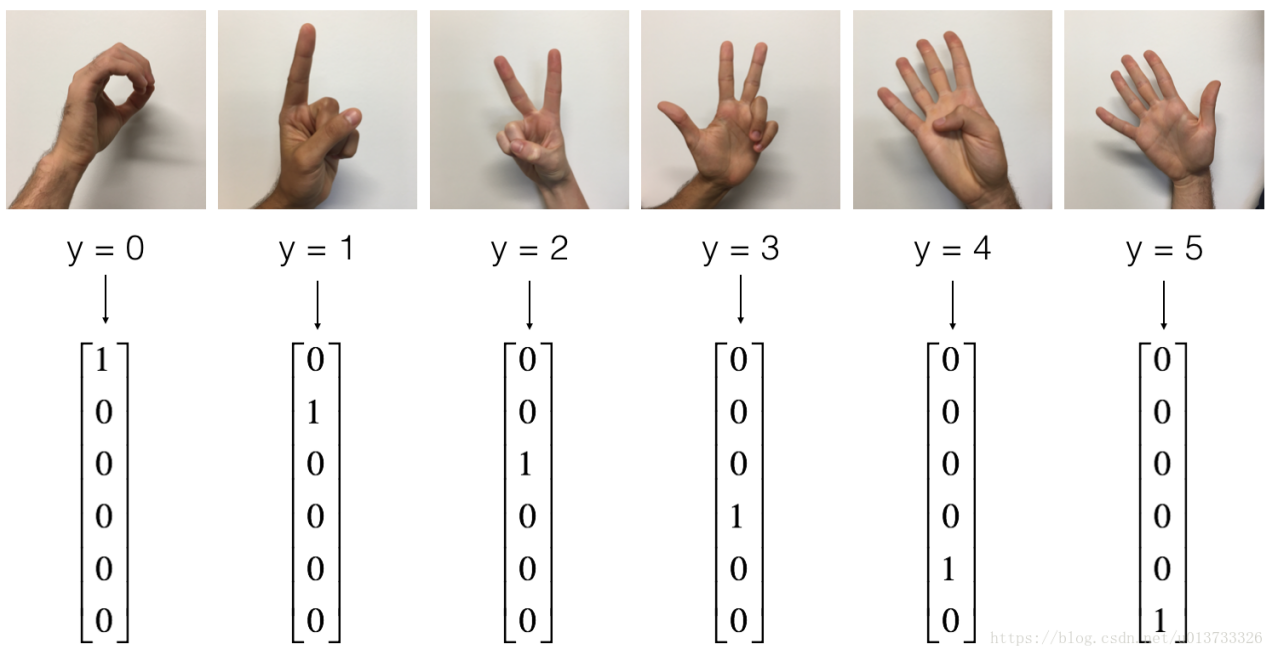
**2.0要解决的问题**

一天下午，我们和一些朋友决定教我们的电脑破译手语。我们花了几个小时在白色的墙壁前拍照，于是就有了了以下数据集。现在，你的任务是建立一个算法，使有语音障碍的人与不懂手语的人交流。

* 训练集：有从0到5的数字的1080张图片(64×64像素)，每个数字拥有180张图片。
* 测试集：有从0到5的数字的120张图片(64×64像素)，每个数字拥有5张图片。

需要注意的是这是完整数据集的一个子集，完整的数据集包含更多的符号。

下面是每个数字的样本，以及我们如何表示标签的解释。这些都是原始图片，我们实际上用的是64×64像素的图片。



首先我们需要加载数据集：

X\_train\_orig,Y\_train\_orig, X\_test\_orig , Y\_test\_orig , classes = tf\_utils.load\_dataset()

# X\_train\_orig.shape=(1080, 64, 64, 3);Y\_train\_orig.shape=(1, 1080);

# X\_test\_orig.shape=(120, 64, 64, 3);Y\_train\_orig.shape=(1, 120);

# classes=[0 1 2 3 4 5]

我们可以看一下数据集里面有什么。

和往常一样，我们要对数据集进行扁平化，然后再除以255以归一化数据，除此之外，我们要需要把每个标签转化为独热向量，像上面的图一样。

我们的目标是构建能够高准确度识别符号的算法。要做到这一点，你要建立一个TensorFlow模型，这个模型几乎和你之前在猫识别中使用的numpy一样（但现在使用softmax输出）。要将您的numpy实现与tensorflow实现进行比较的话这是一个很好的机会。

目前的模型是：LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SOFTMAX，SIGMOID输出层已经转换为SOFTMAX。当有两个以上的类时，一个SOFTMAX层将SIGMOID一般化。

**2.1创建placeholders**

我们的第一项任务是为X和Y创建占位符，这将允许我们稍后在运行会话时传递您的训练数据。

**2.2 初始化参数**

初始化tensorflow中的参数，我们将使用Xavier初始化权重和用零来初始化偏差，比如：

W1=tf.get\_variable("W1",[25,12288],initializer=tf.contrib.layers.xavier\_initializer(seed = 1))

b1 = tf.get\_variable("b1", [25,1], initializer = tf.zeros\_initializer())

tf.Variable() 每次都在创建新对象，对于get\_variable()来说，对于已经创建的变量对象，就把那个对象返回，如果没有创建变量对象的话，就创建一个新的。

**2.3前向传播**

我们将要在TensorFlow中实现前向传播，该函数将接受一个字典参数并完成前向传播，它会用到以下代码：

tf.add(…) ：加法

tf.matmul(… , …) ：矩阵乘法

tf.nn.relu(…) ：Relu激活函数

我们要实现神经网络的前向传播，我们会拿numpy与TensorFlow实现的神经网络的代码作比较。最重要的是前向传播要在Z3处停止，因为在TensorFlow中最后的线性输出层的输出作为计算损失函数的输入，所以不需要A3。

**2.4 计算成本**

如前所述，成本很容易计算：

tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits= ..., labels = ...))

**2.5反向传播&更新参数**

得益于编程框架，所有反向传播和参数更新都在1行代码中处理。计算成本函数后，将创建一个“optimizer”对象。 运行tf.session时，必须将此对象与成本函数一起调用，当被调用时，它将使用所选择的方法和学习速率对给定成本进行优化。

举个例子，对于梯度下降：

optimizer=tf.train.GradientDescentOptimizer(learning\_rate=learning\_rate).minimize(cost)

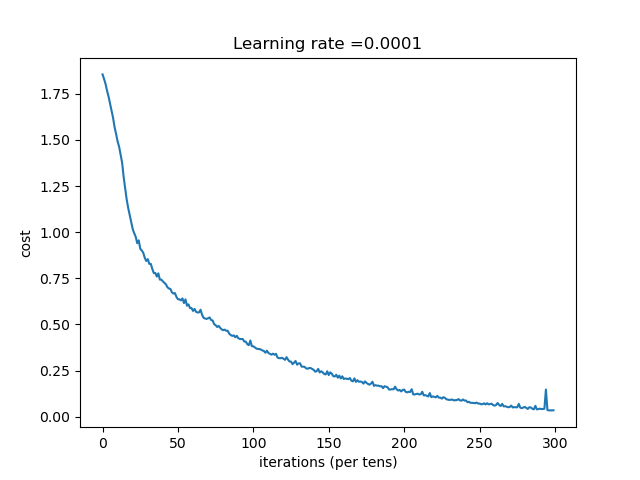
要进行优化，应该这样做：

\_ , c = sess.run([optimizer,cost],feed\_dict={X:mini\_batch\_X,Y:mini\_batch\_Y})

编写代码时，我们经常使用 \_ 作为一次性变量来存储我们稍后不需要使用的值。 这里，\_具有我们不需要的优化器的评估值（并且c取值为成本变量的值）。

**2.6构建模型**

现在我们将实现我们的模型，然后正式运行一下模型，请注意，这次的运行时间大约在5-8分钟左右，如果在epoch = 100的时候，你的epoch\_cost = 1.01645776539的值和我相差过大，那么你就立即停止，回头检查一下哪里出了问题。



现在，我们的算法已经可以识别0-5的手势符号了，准确率在72.5%。

  我们的模型看起来足够大了，可以适应训练集，但是考虑到训练与测试的差异，你也完全可以尝试添加L2或者dropout来减少过拟合。将session视为一组代码来训练模型，在每个minibatch上运行会话时，都会训练我们的参数，总的来说，你已经运行了很多次（1500代），直到你获得训练有素的参数。

**2.7测试你自己的图片（选做）**

  自己拍5张图片，然后将jpg转化为png格式，并将大小剪裁为64×64的图片，因为mpimg只能读取png的图片，最后使用模型进行预测。

自己拍了两组0-5的照片，照片和预测结果如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数字 | 0 | 1 | 2 | 3 | 4 | 5 |
| 第一组  图片 | 0 | 1 | 2 | 3 | 4 | 5 |
| 预测结果 | y = 0 | y = 1 | y = 1 | y = 4 | y = 4 | y = 5 |
| 第一组  图片 | 0 | 1(1) | 2(1) | 3(1) | 4(1) | 5(1) |
| 预测结果 | y = 0 | y = 1 | y = 1 | y = 1 | y = 5 | y = 4 |