本文章内容比较繁杂，主要是一些比较常用的函数的用法，结合了网上的资料和源码，还有我自己写的示例代码。**建议照着目录来看**。

1.矩阵操作

1.1矩阵生成

这部分主要将如何生成矩阵，包括全０矩阵，全１矩阵，随机数矩阵，常数矩阵等

tf.ones | tf.zeros

**tf.ones(shape,type=tf.float32,name=None)**   
**tf.zeros([2, 3], int32)**   
用法类似，都是产生尺寸为shape的张量(tensor)

sess = tf.InteractiveSession()

x = tf.ones([2, 3], int32)

print(sess.run(x))

#[[1 1 1],

# [1 1 1]]

tf.ones\_like | tf.zeros\_like

**tf.ones\_like(tensor,dype=None,name=None)**   
**tf.zeros\_like(tensor,dype=None,name=None)**   
新建一个与给定的tensor类型大小一致的tensor，其所有元素为1和0

tensor=[[1, 2, 3], [4, 5, 6]]

x = tf.ones\_like(tensor)

print(sess.run(x))

#[[1 1 1],

# [1 1 1]]

tf.fill

**tf.fill(shape,value,name=None)**   
创建一个形状大小为shape的tensor，其初始值为value

print(sess.run(tf.fill([2,3],2)))

#[[2 2 2],

# [2 2 2]]

tf.constant

**tf.constant(value,dtype=None,shape=None,name=’Const’)**   
创建一个常量tensor，按照给出value来赋值，可以用shape来指定其形状。value可以是一个数，也可以是一个list。   
如果是一个数，那么这个常亮中所有值的按该数来赋值。   
如果是list,那么len(value)一定要小于等于shape展开后的长度。赋值时，先将value中的值逐个存入。不够的部分，则全部存入value的最后一个值。

a = tf.constant(2,shape=[2])

b = tf.constant(2,shape=[2,2])

c = tf.constant([1,2,3],shape=[6])

d = tf.constant([1,2,3],shape=[3,2])

sess = tf.InteractiveSession()

print(sess.run(a))

#[2 2]

print(sess.run(b))

#[[2 2]

# [2 2]]

print(sess.run(c))

#[1 2 3 3 3 3]

print(sess.run(d))

#[[1 2]

# [3 3]

# [3 3]]

tf.random\_normal | tf.truncated\_normal | tf.random\_uniform

**tf.random\_normal(shape,mean=0.0,stddev=1.0,dtype=tf.float32,seed=None,name=None)**   
**tf.truncated\_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)**   
**tf.random\_uniform(shape,minval=0,maxval=None,dtype=tf.float32,seed=None,name=None)**   
这几个都是用于生成随机数tensor的。尺寸是shape   
random\_normal: 正太分布随机数，均值mean,标准差stddev   
truncated\_normal:截断正态分布随机数，均值mean,标准差stddev,不**过只保留[mean-2\*stddev,mean+2\*stddev]范围内的随机数**   
random\_uniform:均匀分布随机数，范围为[minval,maxval]

sess = tf.InteractiveSession()

x = tf.random\_normal(shape=[1,5],mean=0.0,stddev=1.0,dtype=tf.float32,seed=None,name=None)

print(sess.run(x))

#===>[[-0.36128798 0.58550537 -0.88363433 -0.2677258 1.05080092]]

* 1
* 2
* 3
* 4

tf.get\_variable

get\_variable(name, shape=None, dtype=dtypes.float32, initializer=None,

regularizer=None, trainable=True, collections=None,

caching\_device=None, partitioner=None, validate\_shape=True,

custom\_getter=None):

* 1
* 2
* 3
* 4

如果在该命名域中之前已经有名字=name的变量，则调用那个变量；如果没有，则根据输入的参数重新创建一个名字为name的变量。在众多的输入参数中，有几个是我已经比较了解的，下面来一一讲一下

name: 这个不用说了，变量的名字   
shape: 变量的形状，[]表示一个数，[3]表示长为3的向量，[2,3]表示矩阵或者张量(tensor)   
dtype: 变量的数据格式，主要有tf.int32, tf.float32, tf.float64等等   
**initializer: 初始化工具，有tf.zero\_initializer, tf.ones\_initializer, tf.constant\_initializer, tf.random\_uniform\_initializer, tf.random\_normal\_initializer, tf.truncated\_normal\_initializer等**

1.2 矩阵变换

tf.shape

**tf.shape(Tensor)**   
Returns the shape of a tensor.返回张量的形状。但是注意，**tf.shape函数本身也是返回一个张量**。而在tf中，张量是需要用sess.run(Tensor)来得到具体的值的。

labels = [1,2,3]

shape = tf.shape(labels)

print(shape)

sess = tf.InteractiveSession()

print(sess.run(shape))

# >>>Tensor("Shape:0", shape=(1,), dtype=int32)

# >>>[3]

tf.expand\_dims

**tf.expand\_dims(Tensor, dim)**   
为张量+1维。官网的例子：’t’ is a tensor of shape [2]   
shape(expand\_dims(t, 0)) ==> [1, 2]   
shape(expand\_dims(t, 1)) ==> [2, 1]   
shape(expand\_dims(t, -1)) ==> [2, 1]

sess = tf.InteractiveSession()

labels = [1,2,3]

x = tf.expand\_dims(labels, 0)

print(sess.run(x))

x = tf.expand\_dims(labels, 1)

print(sess.run(x))

#>>>[[1 2 3]]

#>>>[[1]

# [2]

# [3]]

tf.pack

**tf.pack(values, axis=0, name=”pack”)**   
Packs a list of rank-R tensors into one rank-(R+1) tensor   
将一个R维张量列表沿着axis轴组合成一个R+1维的张量。

# 'x' is [1, 4]

# 'y' is [2, 5]

# 'z' is [3, 6]

pack([x, y, z]) => [[1, 4], [2, 5], [3, 6]] # Pack along first dim.

pack([x, y, z], axis=1) => [[1, 2, 3], [4, 5, 6]]

tf.concat

**tf.concat(concat\_dim, values, name=”concat”)**   
Concatenates tensors along one dimension.   
将张量沿着指定维数拼接起来。个人感觉跟前面的pack用法类似

t1 = [[1, 2, 3], [4, 5, 6]]

t2 = [[7, 8, 9], [10, 11, 12]]

tf.concat(0, [t1, t2])

#==> [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]

tf.concat(1, [t1, t2])

#==> [[1, 2, 3, 7, 8, 9], [4, 5, 6, 10, 11, 12]]

tf.sparse\_to\_dense

稀疏矩阵转密集矩阵   
定义为：

def sparse\_to\_dense(sparse\_indices,

output\_shape,

sparse\_values,

default\_value=0,

validate\_indices=True,

name=None):

几个参数的含义：   
sparse\_indices: 元素的坐标[[0,0],[1,2]] 表示(0,0)，和(1,2)处有值   
output\_shape: 得到的密集矩阵的shape   
sparse\_values: sparse\_indices坐标表示的点的值，可以是0D或者1D张量。若0D，则所有稀疏值都一样。若是1D，则len(sparse\_values)应该等于len(sparse\_indices)   
default\_values: 缺省点的默认值

tf.random\_shuffle

**tf.random\_shuffle(value,seed=None,name=None)**   
沿着value的第一维进行随机重新排列

sess = tf.InteractiveSession()

a=[[1,2],[3,4],[5,6]]

x = tf.random\_shuffle(a)

print(sess.run(x))

#===>[[3 4],[5 6],[1 2]]

tf.argmax | tf.argmin

tf.argmax(input=tensor,dimention=axis)   
找到给定的张量tensor中在指定轴axis上的最大值/最小值的位置。

a=tf.get\_variable(name='a',

shape=[3,4],

dtype=tf.float32,

initializer=tf.random\_uniform\_initializer(minval=-1,maxval=1))

b=tf.argmax(input=a,dimension=0)

c=tf.argmax(input=a,dimension=1)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

print(sess.run(a))

#[[ 0.04261756 -0.34297419 -0.87816691 -0.15430689]

# [ 0.18663144 0.86972666 -0.06103253 0.38307118]

# [ 0.84588599 -0.45432305 -0.39736366 0.38526249]]

print(sess.run(b))

#[2 1 1 2]

print(sess.run(c))

#[0 1 0]

tf.equal

tf.equal(x, y, name=None):   
判断两个tensor是否每个元素都相等。返回一个格式为bool的tensor

tf.cast

cast(x, dtype, name=None)   
将x的数据格式转化成dtype.例如，原来x的数据格式是bool，那么将其转化成float以后，就能够将其转化成0和1的序列。反之也可以

a = tf.Variable([1,0,0,1,1])

b = tf.cast(a,dtype=tf.bool)

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

print(sess.run(b))

#[ True False False True True]

tf.matmul

用来做矩阵乘法。若a为l\*m的矩阵，b为m\*n的矩阵，那么通过tf.matmul(a,b) 结果就会得到一个l\*n的矩阵   
不过这个函数还提供了很多额外的功能。我们来看下函数的定义：

matmul(a, b,

transpose\_a=False, transpose\_b=False,

a\_is\_sparse=False, b\_is\_sparse=False,

name=None):

可以看到还提供了transpose和is\_sparse的选项。   
如果对应的transpose项为True，例如transpose\_a=True,那么a在参与运算之前就会先转置一下。   
而如果a\_is\_sparse=True,那么a会被当做稀疏矩阵来参与运算。

tf.reshape

reshape(tensor, shape, name=None)   
顾名思义，就是将tensor按照新的shape重新排列。一般来说，shape有三种用法：   
如果 shape=[-1], 表示要将tensor展开成一个list   
如果 shape=[a,b,c,…] 其中每个a,b,c,..均>0，那么就是常规用法   
如果 shape=[a,-1,c,…] 此时b=-1，a,c,..依然>0。这表示tf会根据tensor的原尺寸，自动计算b的值。   
官方给的例子已经很详细了，我就不写示例代码了

# tensor 't' is [1, 2, 3, 4, 5, 6, 7, 8, 9]

# tensor 't' has shape [9]

reshape(t, [3, 3]) ==> [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

# tensor 't' is [[[1, 1], [2, 2]],

# [[3, 3], [4, 4]]]

# tensor 't' has shape [2, 2, 2]

reshape(t, [2, 4]) ==> [[1, 1, 2, 2],

[3, 3, 4, 4]]

# tensor 't' is [[[1, 1, 1],

# [2, 2, 2]],

# [[3, 3, 3],

# [4, 4, 4]],

# [[5, 5, 5],

# [6, 6, 6]]]

# tensor 't' has shape [3, 2, 3]

# pass '[-1]' to flatten 't'

reshape(t, [-1]) ==> [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6]

# -1 can also be used to infer the shape

# -1 is inferred to be 9:

reshape(t, [2, -1]) ==> [[1, 1, 1, 2, 2, 2, 3, 3, 3],

[4, 4, 4, 5, 5, 5, 6, 6, 6]]

# -1 is inferred to be 2:

reshape(t, [-1, 9]) ==> [[1, 1, 1, 2, 2, 2, 3, 3, 3],

[4, 4, 4, 5, 5, 5, 6, 6, 6]]

# -1 is inferred to be 3:

reshape(t, [ 2, -1, 3]) ==> [[[1, 1, 1],

[2, 2, 2],

[3, 3, 3]],

[[4, 4, 4],

[5, 5, 5],

[6, 6, 6]]]

2. 神经网络相关操作

tf.nn.embedding\_lookup

embedding\_lookup(params, ids, partition\_strategy=”mod”, name=None,   
validate\_indices=True):

简单的来讲，就是将一个数字序列ids转化成embedding序列表示。   
假设params.shape=[v,h], ids.shape=[m], 那么该函数会返回一个shape=[m,h]的张量。用数学来表示，就是 

ids=[i1,i2,…,im]params=⎡⎣⎢⎢⎢⎢⎢w11,w21,⋯,wh1w12,w22,⋯,wh2⋮w1v,w2v,⋯,whv⎤⎦⎥⎥⎥⎥⎥res=⎡⎣⎢⎢⎢⎢⎢⎢w1i1,w2i1,…,whi1w1i2,w2i2,…,whi2⋮w1im,w2im,…,whim⎤⎦⎥⎥⎥⎥⎥⎥

那么这个有什么用呢？如果你了解[word2vec](http://blog.csdn.net/u014595019/article/details/51884529)的话，就知道我们可以根据文档来对每个单词生成向量。单词向量可以进一步用来测量单词的相似度等等。那么假设我们现在已经获得了每个单词的向量，都存在param中。那么根据单词id序列ids,就可以通过embedding\_lookup来获得embedding表示的序列。

tf.trainable\_variables

**返回所有可训练的变量**。   
在创造变量(tf.Variable, tf.get\_variable 等操作)时，都会有一个trainable的选项，表示该变量是否可训练。这个函数会返回图中所有trainable=True的变量。   
**tf.get\_variable(…), tf.Variable(…)的默认选项是True, 而 tf.constant(…)只能是False**

import tensorflow as tf

from pprint import pprint

a = tf.get\_variable('a',shape=[5,2]) # 默认 trainable=True

b = tf.get\_variable('b',shape=[2,5],trainable=False)

c = tf.constant([1,2,3],dtype=tf.int32,shape=[8],name='c') # 因为是常量，所以trainable=False

d = tf.Variable(tf.random\_uniform(shape=[3,3]),name='d')

tvar = tf.trainable\_variables()

tvar\_name = [x.name for x in tvar]

print(tvar)

# [<tensorflow.python.ops.variables.Variable object at 0x7f9c8db8ca20>, <tensorflow.python.ops.variables.Variable object at 0x7f9c8db8c9b0>]

print(tvar\_name)

# ['a:0', 'd:0']

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

pprint(sess.run(tvar))

#[array([[ 0.27307487, -0.66074866],

# [ 0.56380701, 0.62759042],

# [ 0.50012994, 0.42331111],

# [ 0.29258847, -0.09185416],

# [-0.35913971, 0.3228929 ]], dtype=float32),

# array([[ 0.85308731, 0.73948073, 0.63190091],

# [ 0.5821209 , 0.74533939, 0.69830012],

# [ 0.61058474, 0.76497936, 0.10329771]], dtype=float32)]

tf.gradients

用来计算导数。该函数的定义如下所示

def gradients(ys,

xs,

grad\_ys=None,

name="gradients",

colocate\_gradients\_with\_ops=False,

gate\_gradients=False,

aggregation\_method=None):

虽然可选参数很多，但是最常使用的还是ys和xs。根据说明得知，ys和xs都可以是一个tensor或者tensor列表。而计算完成以后，该函数会返回一个长为len(xs)的tensor列表，列表中的每个tensor是ys中每个值对xs[i]求导之和。如果用数学公式表示的话，那么 g = tf.gradients(y,x)可以表示成 

gi=∑j=0len(y)∂yj∂xig=[g0,g1,...,glen(x)]

tf.clip\_by\_global\_norm

修正梯度值，用于**控制梯度爆炸的问题**。梯度爆炸和梯度弥散的原因一样，都是因为链式法则求导的关系，导致梯度的指数级衰减。为了避免梯度爆炸，需要对梯度进行修剪。   
先来看这个函数的定义：

def clip\_by\_global\_norm(t\_list, clip\_norm, use\_norm=None, name=None):

* 1

**输入参数**中：t\_list为待修剪的张量, clip\_norm 表示修剪比例(clipping ratio).

函数**返回2个参数**： list\_clipped，修剪后的张量，以及global\_norm，一个中间计算量。当然如果你之前已经计算出了global\_norm值，你可以在use\_norm选项直接指定global\_norm的值。

那么具体**如何计算**呢？根据源码中的说明，可以得到   
list\_clipped[i]=t\_list[i] \* clip\_norm / max(global\_norm, clip\_norm),其中   
global\_norm = sqrt(sum([l2norm(t)\*\*2 for t in t\_list]))

如果你更熟悉数学公式，则可以写作 

Lic=Lit∗Ncmax(Nc,Ng)Ng=∑i(Lit)2−−−−−−−√

其中，   
Lic和Lig代表t\_list[i]和list\_clipped[i]，   
Nc和Ng代表clip\_norm 和 global\_norm的值。   
其实也可以看到其实Ng就是t\_list的L2模。上式也可以进一步写作 

Lic={Lit,(Ng<=Nc)Lit∗NcNg,(Ng>Nc)Ng=∑i(Lit)2−−−−−−−√

也就是说，当t\_list的L2模大于指定的Nc时，就会对t\_list做等比例缩放

tf.nn.dropout

dropout(x, keep\_prob, noise\_shape=None, seed=None, name=None)   
**按概率来将x中的一些元素值置零，并将其他的值放大**。用于进行dropout操作，一定程度上可以防止过拟合   
x是一个张量，而keep\_prob是一个（0,1]之间的值。x中的各个元素清零的概率互相独立，为1-keep\_prob,而没有清零的元素，则会统一乘以1/keep\_prob, 目的是为了保持x的整体期望值不变。

sess = tf.InteractiveSession()

a = tf.get\_variable('a',shape=[2,5])

b = a

a\_drop = tf.nn.dropout(a,0.8)

sess.run(tf.initialize\_all\_variables())

print(sess.run(b))

#[[ 0.28667903 -0.66874665 -1.14635754 0.88610041 -0.55590457]

# [-0.29704338 -0.01958954 0.80359757 0.75945008 0.74934876]]

print(sess.run(a\_drop))

#[[ 0.35834879 -0.83593333 -1.43294692 1.10762548 -0. ]

# [-0.37130421 -0. 0. 0.94931257 0.93668592]]

3.普通操作

tf.linspace | tf.range

**tf.linspace(start,stop,num,name=None)**   
**tf.range(start,limit=None,delta=1,name=’range’)**   
这两个放到一起说，是因为他们都用于产生等差数列，不过具体用法不太一样。   
tf.linspace在[start,stop]范围内产生num个数的等差数列。不过注意，**start和stop要用浮点数表示，不然会报错**   
tf.range在[start,limit)范围内以步进值delta产生等差数列。注意是不包括limit在内的。

sess = tf.InteractiveSession()

x = tf.linspace(start=1.0,stop=5.0,num=5,name=None) # 注意1.0和5.0

y = tf.range(start=1,limit=5,delta=1)

print(sess.run(x))

print(sess.run(y))

#===>[ 1. 2. 3. 4. 5.]

#===>[1 2 3 4]

tf.assign

**assign(ref, value, validate\_shape=None, use\_locking=None, name=None)**   
tf.assign是用来更新模型中变量的值的。ref是待赋值的变量，value是要更新的值。即效果等同于 ref = value   
简单的实例代码见下

sess = tf.InteractiveSession()

a = tf.Variable(0.0)

b = tf.placeholder(dtype=tf.float32,shape=[])

op = tf.assign(a,b)

sess.run(tf.initialize\_all\_variables())

print(sess.run(a))

# 0.0

sess.run(op,feed\_dict={b:5.})

print(sess.run(a))

# 5.0

4.规范化

tf.variable\_scope

简单的来讲，就是为变量添加命名域

with tf.variable\_scope("foo"):

with tf.variable\_scope("bar"):

v = tf.get\_variable("v", [1])

assert v.name == "foo/bar/v:0"

函数的定义为

def variable\_scope(name\_or\_scope, reuse=None, initializer=None,

regularizer=None, caching\_device=None, partitioner=None,

custom\_getter=None):

各变量的含义如下：   
name\_or\_scope: string or VariableScope: the scope to open.   
reuse: True or None; if True, we go into reuse mode for this scope as well as all sub-scopes; if None, we just inherit the parent scope reuse. 如果reuse=True, 那么就是使用之前定义过的name\_scope和其中的变量，   
initializer: default initializer for variables within this scope.   
regularizer: default regularizer for variables within this scope.   
caching\_device: default caching device for variables within this scope.   
partitioner: default partitioner for variables within this scope.   
custom\_getter: default custom getter for variables within this scope.

tf.get\_variable\_scope

返回当前变量的命名域，返回一个tensorflow.python.ops.variable\_scope.VariableScope变量。