tensorflow提供了API文档

参考：<http://www.tensorfly.cn/tfdoc/api_docs/python/nn.html>

下面列出一些常用的ops，比如array ops ,math , nn ,train等。

具体可以?tf.add查看详细帮助

<http://www.cnblogs.com/wuzhitj/p/6431381.html>

# Building Graphs

## Core graph data structures

**class tf.Graph**

tf.Graph.\_\_init\_\_()

tf.Graph.as\_default()

tf.Graph.device(device\_name\_or\_function)

tf.Graph.name\_scope(name)

tf.Graph.add\_to\_collection(name, value)

tf.Graph.add\_to\_collections(names, value)

tf.Graph.get\_collection(name, scope=None)

# get\_collection 其实就是get a list of values by name , 也就是 get\_values

tf.Graph.get\_operation\_by\_name(name)

tf.Graph.get\_operations()

tf.Graph.seed

tf.Graph.unique\_name(name, mark\_as\_used=True)

tf.Graph.version

tf.Graph.create\_op(op\_type, inputs, dtypes, input\_types=None, name=None, attrs=None, op\_def=None, compute\_shapes=True, compute\_device=True)

**class tf.Operation**

**class tf.Tensor**

tf.Tensor.dtype

tf.Tensor.name

tf.Tensor.value\_index

tf.Tensor.graph

tf.Tensor.op

tf.Tensor.consumers()

tf.Tensor.eval(feed\_dict=None, session=None)

tf.Tensor.get\_shape()

tf.Tensor.set\_shape(shape)

tf.Tensor.\_\_abs\_\_(x, name=None)

tf.Tensor.\_\_add\_\_(x, y)

tf.Tensor.\_\_and\_\_(x, y)

tf.Tensor.\_\_bool\_\_()

tf.Tensor.\_\_div\_\_(x, y)

## Tensor types

**class tf.DType**

The following DType objects are defined:

* tf.float16: 16-bit half-precision floating-point.
* tf.float32: 32-bit single-precision floating-point.
* tf.float64: 64-bit double-precision floating-point.
* tf.bfloat16: 16-bit truncated floating-point.
* tf.complex64: 64-bit single-precision complex.
* tf.complex128: 128-bit double-precision complex.
* tf.int8: 8-bit signed integer.
* tf.uint8: 8-bit unsigned integer.
* tf.uint16: 16-bit unsigned integer.
* tf.int16: 16-bit signed integer.
* tf.int32: 32-bit signed integer.
* tf.int64: 64-bit signed integer.
* tf.bool: Boolean.
* tf.string: String.
* tf.qint8: Quantized 8-bit signed integer.
* tf.quint8: Quantized 8-bit unsigned integer.
* tf.qint16: Quantized 16-bit signed integer.
* tf.quint16: Quantized 16-bit unsigned integer.
* tf.qint32: Quantized 32-bit signed integer.
* tf.resource: Handle to a mutable resource.

**tf.as\_dtype(type\_value)**

converts numpy types and string type names to a DType object

# Constants, Sequences, and Random Values

## Constant Value Tensors

tf.zeros(shape, dtype=tf.float32, name=None)

tf.zeros\_like(tensor, dtype=None, name=None)

tf.ones(shape, dtype=tf.float32, name=None)

tf.ones\_like(tensor, dtype=None, name=None)

tf.fill(dims, value, name=None)

tf.constant(value, dtype=None, shape=None, name='Const')

## Sequences

tf.linspace(start, stop, num, name=None)

tf.range(start, limit, delta=1, name='range')

## Random Tensors

#正太分布

tf.random\_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

#截断的正太分布

#如果生成的值大于平均值2个标准偏差的值则丢弃重新选择。

tf.truncated\_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)#

#均匀分布

tf.random\_uniform(shape, minval=0.0, maxval=1.0, dtype=tf.float32, seed=None, name=None)

#随机打散shuffle

tf.random\_shuffle(value, seed=None, name=None)

tf.set\_random\_seed(seed)

tf.stack(values, axis=0, name="stack"):

# 'x' is [1, 4]

# 'y' is [2, 5]

# 'z' is [3, 6]

tf.stack([x, y, z]) = np.asarray([x, y, z])

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

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

# Variables

## Variables

class tf.Variable

## Variable helper functions

tf.global\_variables()

tf.local\_variables()

tf.model\_variables()

tf.trainable\_variables()

tf.moving\_average\_variables()

tf.global\_variables\_initializer()

tf.local\_variables\_initializer()

tf.variables\_initializer(var\_list, name=init)

tf.is\_variable\_initialized(variable)

tf.report\_uninitialized\_variables(var\_list=None, name=report\_uninitialized\_variables)

tf.assert\_variables\_initialized(var\_list=None)

tf.assign(ref, value, validate\_shape=None, use\_locking=None, name=None)

tf.assign\_add(ref, value, use\_locking=None, name=None)

tf.assign\_sub(ref, value, use\_locking=None, name=None)

## Saving and Restoring Variables（变量保存）

class tf.train.Saver

#保存

saver=tf.train.Saver()

with tf.Session() as sess:

saver.save(sess,"path/model/model.ckpt")

目录下会生成图结构的model.ckpt.meta和变量取值的model.ckpt

# load

saver = tf.train.Saver()

# 若只加载图

saver2 = tf.train.import\_meta\_graph("path/model/model.ckpt/model.ckpt.meta")

with tf.Session() as sess:

saver.restore(sess,"/model/model.ckpt")

saver2.restore(sess,"/model/model.ckpt")

tf.train.latest\_checkpoint(checkpoint\_dir, latest\_filename=None)

tf.train.get\_checkpoint\_state(checkpoint\_dir, latest\_filename=None)

tf.train.update\_checkpoint\_state(save\_dir, model\_checkpoint\_path, all\_model\_checkpoint\_paths=None, latest\_filename=None)

## Sharing Variables

tf.get\_variable(name, shape=None, dtype=None, initializer=None, regularizer=None, trainable=True, collections=None, caching\_device=None, partitioner=None, validate\_shape=True, custom\_getter=None)

tf.get\_variable可以创建变量和获取变量。

#创建变量时，下面2个定义是等价的

v=tf.get\_variable("v",shape=[1],initializer=tf.constant\_initializer(1))

v=tf.Variable(tf.constant(1.0,shape=[1]),name="v")

class tf.VariableScope

tf.variable\_scope(name\_or\_scope, default\_name=None, values=None, initializer=None, regularizer=None, caching\_device=None, partitioner=None, custom\_getter=None, reuse=None, dtype=None)

tf.variable\_scope会创建一个namespace,所以提供了一个变量命名空间的方式。

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

v2=tf.get\_variable("v",[1])

print(v2.name)

# foo/v:0

tf.variable\_op\_scope(values, name\_or\_scope, default\_name=None, initializer=None, regularizer=None, caching\_device=None, partitioner=None, custom\_getter=None, reuse=None, dtype=None)

tf.get\_variable\_scope()

tf.make\_template(name\_, func\_, create\_scope\_now\_=False, unique\_name\_=None, \*\*kwargs)

tf.no\_regularizer(\_)

## initializer

#将变量初始化为给定常量

tf.constant\_initializer(value=0, dtype=tf.float32)

#将变量初始化为满足正太分布的随即值。

tf.random\_normal\_initializer(mean=0.0, stddev=1.0, seed=None, dtype=tf.float32)

tf.truncated\_normal\_initializer(mean=0.0, stddev=1.0, seed=None, dtype=tf.float32)

tf.random\_uniform\_initializer(minval=0, maxval=None, seed=None, dtype=tf.float32)

#均匀分布，只是这个初始化方法不需要指定最小最大值，自动计算出来

tf.uniform\_unit\_scaling\_initializer(factor=1.0, seed=None, dtype=tf.float32)

#将变量设置为0,

tf.zeros\_initializer(shape, dtype=tf.float32, partition\_info=None)

tf.ones\_initializer(dtype=tf.float32, partition\_info=None)

tf.orthogonal\_initializer(gain=1.0, dtype=tf.float32, seed=None)

**# xaiver**

tf.contrib.layers.xavier\_initializer()

limit : x = sqrt(6. / (in + out)); [-x, x]

tf.contrib.layers.xavier\_initializer\_conv2d()

xavier\_initializer\_conv2d = xavier\_initializer

def xavier\_init(fan\_in, fan\_out):

n = (fan\_in + fan\_out) / 2.0

factor=1

limit = np.sqrt(3.0 \* factor / n)

return tf.random\_uniform((fan\_in, fan\_out),

-limit, limit,

dtype = tf.float32)

## Variable Partitioners for Sharding

tf.fixed\_size\_partitioner(num\_shards, axis=0)

tf.variable\_axis\_size\_partitioner(max\_shard\_bytes, axis=0, bytes\_per\_string\_element=16, max\_shards=None)

tf.min\_max\_variable\_partitioner(max\_partitions=1, axis=0, min\_slice\_size=262144, bytes\_per\_string\_element=16)

## Sparse Variable Updates

tf.scatter\_update(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_add(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_sub(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_mul(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_div(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_nd\_update(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_nd\_add(ref, indices, updates, use\_locking=None, name=None)

tf.scatter\_nd\_sub(ref, indices, updates, use\_locking=None, name=None)

tf.sparse\_mask(a, mask\_indices, name=None)

class tf.IndexedSlices

## Read-only Lookup Tables

tf.initialize\_all\_tables(name=init\_all\_tables)

## Exporting and Importing Meta Graphs

tf.train.export\_meta\_graph(filename=None, meta\_info\_def=None, graph\_def=None, saver\_def=None, collection\_list=None, as\_text=False, graph=None, export\_scope=None, clear\_devices=False, \*\*kwargs)

tf.train.import\_meta\_graph(meta\_graph\_or\_file, clear\_devices=False, import\_scope=None, \*\*kwargs)

Deprecated functions (removed after 2017-03-02). Please don't use them.

tf.all\_variables(\*args, \*\*kwargs)

tf.initialize\_all\_variables(\*args, \*\*kwargs)

tf.initialize\_local\_variables(\*args, \*\*kwargs)

tf.initialize\_variables(\*args, \*\*kwargs)

# Tensor Transformations(array ops)

## Cast

tf.string\_to\_number(string\_tensor, out\_type=None, name=None)

tf.as\_string(input, precision=None)

tf.to\_double(x, name='ToDouble')

tf.to\_float(x, name='ToFloat')

tf.to\_bfloat16(x, name='ToBFloat16')

tf.to\_int32(x, name='ToInt32')

tf.to\_int64(x, name='ToInt64')

tf.cast(x, dtype, name=None)

## Shapes and Shaping

tf.shape(input, name=None)

tf.size(input, name=None)

tf.rank(input, name=None)

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

tf.squeeze(input, squeeze\_dims=None, name=None)

tf.expand\_dims(input, dim, name=None)

增加一维，位置是dim

```python

# '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]

# 't2' is a tensor of shape [2, 3, 5]

shape(expand\_dims(t2, 0)) ==> [1, 2, 3, 5]

shape(expand\_dims(t2, 2)) ==> [2, 3, 1, 5]

shape(expand\_dims(t2, 3)) ==> [2, 3, 5, 1]

```

## Slicing and Joining

tf.slice(input\_, begin, size, name=None)

tf.split(split\_dim, num\_split, value, name='split')

tf.tile(input, multiples, name=None)

tf.pad(input, paddings, name=None)

tf.concat(concat\_dim, values, name='concat')

tf.stack(values, axis=0, name='stack')

if axis == 0 then the output tensor will have the shape (N, A, B, C).

if axis == 1 then the output tensor will have the shape (A, N, B, C). Etc.

# 'x' is [1, 4]

# 'y' is [2, 5]

# 'z' is [3, 6]

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

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

tf.pack(values, name='pack')

tf.unpack(value, num=None, name='unpack')

tf.reverse\_sequence(input, seq\_lengths, seq\_dim, name=None)

tf.reverse(tensor, dims, name=None)

tf.transpose(a, perm=None, name='transpose')

tf.gather(params, indices, name=None)

参数说明：params是一个tensor， indices是个值为int的tensor用来指定要从params取得元素的第0维的index。该函数可看成是tf.nn.embedding\_lookup()的特殊形式，所以功能与其类是，即将其看成是embedding\_lookup函数的params参数内只有一个tensor时的情形。

tf.dynamic\_partition(data, partitions, num\_partitions, name=None)

tf.dynamic\_stitch(indices, data, name=None)

Args:

indices: A list of at least 1 Tensor objects with type int32.

data: A list with the same length as indices of Tensor objects with the same type.

name: A name for the operation (optional).

example:

x\_shuffle = tf.dynamic\_stitch([ind], [x1])

#x1是个tensor , 如果x是个araay ,可以这样： x\_ shuffle =x[ind]

x\_shuffle = tf.dynamic\_stitch([ind1,ind2], [x1,x2])

#相当于

x=merge(x1,x2)

ind=merge(ind1,ind2)

x\_shuffle = x[ind]

# Math ops

## Arithmetic Operators

tf.add(x, y, name=None) 求和

tf.sub(x, y, name=None) 减法

tf.mul(x, y, name=None) 乘法

tf.div(x, y, name=None) 除法

tf.mod(x, y, name=None) 取模

## Basic Math Functions

tf.abs(x, name=None) 求绝对值

tf.neg(x, name=None) 取负 (y = -x).

tf.sign(x, name=None) 返回符号 y = sign(x) = -1 if x < 0; 0 if x == 0; 1 if x > 0.

tf.inv(x, name=None) 取反

tf.square(x, name=None) 计算平方 (y = x \* x = x^2).

tf.round(x, name=None) 舍入最接近的整数

# ‘a’ is [0.9, 2.5, 2.3, -4.4]

tf.round(a) ==> [ 1.0, 3.0, 2.0, -4.0 ]

tf.sqrt(x, name=None) 开根号 (y = \sqrt{x} = x^{1/2}).

tf.pow(x, y, name=None) 幂次方

# tensor ‘x’ is [[2, 2], [3, 3]]

# tensor ‘y’ is [[8, 16], [2, 3]]

tf.pow(x, y) ==> [[256, 65536], [9, 27]]

tf.exp(x, name=None) 计算e的次方

tf.log(x, name=None) 计算log，一个输入计算e的ln，两输入以第二输入为底

tf.maximum(x, y, name=None) 返回最大值 (x > y ? x : y)

tf.minimum(x, y, name=None) 返回最小值 (x < y ? x : y)

tf.cos(x, name=None) 三角函数cosine

tf.sin(x, name=None) 三角函数sine

tf.tan(x, name=None) 三角函数tan

tf.atan(x, name=None) 三角函数ctan

## Matrix Math Functions

tf.diag(diagonal, name=None)

返回一个给定对角值的对角tensor

# ‘diagonal’ is [1, 2, 3, 4]

tf.diag(diagonal) ==>

[[1, 0, 0, 0]

[0, 2, 0, 0]

[0, 0, 3, 0]

[0, 0, 0, 4]]

tf.diag\_part(input, name=None) 功能与上面相反

tf.transpose(a, perm=None, name='transpose')

tf.matmul(a, b, transpose\_a=False, transpose\_b=False, a\_is\_sparse=False, b\_is\_sparse=False, name=None)

tf.matrix\_determinant(input, name=None) 返回方阵的行列式

tf.matrix\_inverse(input, name=None) 求方阵的逆矩阵，adjoint为True时，计算输入共轭矩阵的逆矩阵

tf.cholesky(input, name=None) 对输入方阵cholesky分解，

即把一个对称正定的矩阵表示成一个下三角矩阵L和其转置的乘积的分解A=LL^T

tf.matrix\_solve(matrix, rhs, adjoint=None, name=None)

求解tf.matrix\_solve(matrix, rhs, adjoint=None, name=None)

matrix为方阵shape为[M,M],rhs的shape为[M,K]，output为[M,K]

## Reduction

tf.reduce\_sum(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None)

tf.reduce\_prod(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None)

tf.reduce\_min(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None)

tf.reduce\_max(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None)

tf.reduce\_mean(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None)

tf.reduce\_all(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None) 对tensor中各个元素求逻辑’与’

tf.reduce\_any(input\_tensor, reduction\_indices=None, keep\_dims=False, name=None) 对tensor中各个元素求逻辑’或’

tf.accumulate\_n(inputs, shape=None, tensor\_dtype=None, name=None)

计算一系列tensor的和

# tensor ‘a’ is [[1, 2], [3, 4]]

# tensor b is [[5, 0], [0, 6]]

tf.accumulate\_n([a, b, a]) ==> [[7, 4], [6, 14]]

tf.cumsum(x, axis=0, exclusive=False, reverse=False, name=None)

求累积和

tf.cumsum([a, b, c]) ==> [a, a + b, a + b + c]

tf.cumsum([a, b, c], exclusive=True) ==> [0, a, a + b]

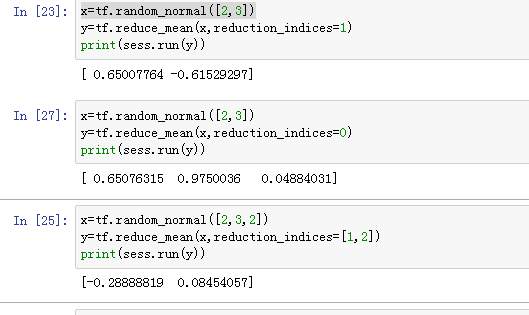
tf.cumsum([a, b, c], reverse=True) ==> [a + b + c, b + c, c]

tf.cumsum([a, b, c], exclusive=True, reverse=True) ==> [b + c, c, 0]

\*这里说明下参数：reduction\_indices

假设input的shape=[2,3] ,即2行3列的矩阵。

若reduction\_indices=1,则按shape(1)的维度做reduce，做完reduce后，shape(1)=1，即output的shape=[2,1] 。简单记忆：reduction\_indices的维度最后的shape会=1.



## Segmentation

tf.segment\_sum(data, segment\_ids, name=None)

根据segment\_ids的分段计算各个片段的和

其中segment\_ids为一个size与data第一维相同的tensor

其中id为int型数据，最大id不大于size

c = tf.constant([[1,2,3,4], [-1,-2,-3,-4], [5,6,7,8]])

tf.segment\_sum(c, tf.constant([0, 0, 1]))

==>[[0 0 0 0]

[5 6 7 8]]

上面例子分为[0,1]两id,对相同id的data相应数据进行求和,

并放入结果的相应id中，

且segment\_ids只升不降

tf.segment\_prod(data, segment\_ids, name=None)

根据segment\_ids的分段计算各个片段的积

tf.segment\_min(data, segment\_ids, name=None)

tf.segment\_max(data, segment\_ids, name=None)

tf.segment\_mean(data, segment\_ids, name=None)

tf.unsorted\_segment\_sum(data, segment\_ids,num\_segments, name=None)

与tf.segment\_sum函数类似，不同在于segment\_ids中id顺序可以是无序的

tf.sparse\_segment\_sum(data, indices, segment\_ids, name=None)

输入进行稀疏分割求和

c = tf.constant([[1,2,3,4], [-1,-2,-3,-4], [5,6,7,8]])

# Select two rows, one segment.

tf.sparse\_segment\_sum(c, tf.constant([0, 1]), tf.constant([0, 0]))

==> [[0 0 0 0]]

对原data的indices为[0,1]位置的进行分割，

并按照segment\_ids的分组进行求和

## Sequence Comparison and Indexing

tf.argmin(input, dimension, name=None) 返回input最小值的索引index

tf.argmax(input, dimension, name=None)

tf.listdiff(x, y, name=None) 返回x，y中不同值的索引

tf.where(input, name=None)

返回bool型tensor中为True的位置

# ‘input’ tensor is

#[[True, False]

#[True, False]]

# ‘input’ 有两个’True’,那么输出两个坐标值.

# ‘input’的rank为2, 所以每个坐标为具有两个维度.

where(input) ==>

[[0, 0],

[1, 0]]

tf.unique(x, name=None)

返回一个元组tuple(y,idx)，y为x的列表的唯一化数据列表，

idx为x数据对应y元素的index

# tensor ‘x’ is [1, 1, 2, 4, 4, 4, 7, 8, 8]

y, idx = unique(x)

y ==> [1, 2, 4, 7, 8]

idx ==> [0, 0, 1, 2, 2, 2, 3, 4, 4]

tf.edit\_distance(hypothesis, truth, normalize=True, name='edit\_distance')

tf.invert\_permutation(x, name=None)

置换x数据与索引的关系

# tensor x is [3, 4, 0, 2, 1]

invert\_permutation(x) ==> [2, 4, 3, 0, 1]

# NN

## Activation Functions

tf.nn.relu(features, name=None)

tf.nn.relu6(features, name=None)

tf.nn.softplus(features, name=None)

tf.nn.dropout(x, keep\_prob, noise\_shape=None, seed=None, name=None)

tf.nn.bias\_add(value, bias, name=None)

tf.sigmoid(x, name=None)

tf.tanh(x, name=None)

tf.nn.softmax(logits, name=None)

tf.nn.log\_softmax(logits, dim=-1, name=None)

## Convolution

tf.nn.convolution(input, filter, padding, strides=None, dilation\_rate=None, name=None, data\_format=None)

tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)

-----------------------------------------------------------

input: [batch, in\_height, in\_width, in\_channels]

filter: [filter\_height, filter\_width, in\_channels, out\_channels]

strides: [1, stride, stride, 1]

padding: SAME,VALID. SAME表示input不需要填充

output: [batch,停留次数，停留次数，out\_channels]

eg:

x\_image = tf.reshape(x, [-1,28,28,1])

# x\_image的batch=n,根据计算得到, 28\*28像素，1通道。如果是RGB,则channels=3.

W\_conv1 = tf.truncated\_normal ([5, 5, 1, 32] , stddev=0.1)

# W\_conv1的out\_channels=32,其实就是说conv层的depth=32,即32个神经元

y\_conv1 = tf.nn.conv2d(x\_image, W\_conv1, strides=[1, 1, 1, 1], padding='SAME')

# strides=[1, 1, 1, 1] 表示步长为1

b\_conv1 = bias\_variable([32])

h\_conv1 = tf.nn.relu(y\_conv1 + b\_conv1)

-----------------------------------------------------------

tf.nn.conv1d(value, filters, stride, padding, use\_cudnn\_on\_gpu=None, data\_format=None, name=None)

tf.nn.conv3d(input, filter, strides, padding, name=None)

tf.nn.depthwise\_conv2d(input, filter, strides, padding, name=None)

tf.nn.separable\_conv2d(input, depthwise\_filter, pointwise\_filter, strides, padding, name=None)

## Pooling

tf.nn.avg\_pool(value, ksize, strides, padding, name=None)

tf.nn.max\_pool(value, ksize, strides, padding, name=None)

tf.nn.max\_pool\_with\_argmax(input, ksize, strides, padding, Targmax=None, name=None)

## Normalization

tf.nn.l2\_normalize(x, dim, epsilon=1e-12, name=None)

tf.nn.local\_response\_normalization(input, depth\_radius=None, bias=None, alpha=None, beta=None, name=None)

tf.nn.moments(x, axes, name=None) #返回mean and variance

## Losses

tf.nn.l2\_loss(t, name=None) #平方损失

tf.nn.log\_poisson\_loss(log\_input, targets, compute\_full\_loss=False, name=None)

## Classification loss

tf.nn.softmax(logits, name=None)

计算softmax

softmax[i, j] = exp(logits[i, j]) / sum\_j(exp(logits[i, j]))

tf.nn.log\_softmax(logits, name=None)

logsoftmax[i, j] = logits[i, j] - log(sum(exp(logits[i])))

tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits, targets, name=None)

tf.nn.softmax\_cross\_entropy\_with\_logits(logits, labels, name=None)

tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits, labels, name=None)

tf.nn.weighted\_cross\_entropy\_with\_logits(logits, targets, pos\_weight, name=None)

与sigmoid\_cross\_entropy\_with\_logits()相似，但给正向样本损失加了权重pos\_weight

#可以自己定义。

cross\_entropy = -tf.reduce\_sum(y\*tf.log(y\_))

## Embeddings

tf.nn.embedding\_lookup(params, ids, name=None)

根据索引ids查询embedding列表params中的tensor值

如果len(params) > 1，id将会安照partition\_strategy策略进行分割

1、如果partition\_strategy为”mod”，

id所分配到的位置为p = id % len(params)

比如有13个ids，分为5个位置，那么分配方案为：

[[0, 5, 10], [1, 6, 11], [2, 7, 12], [3, 8], [4, 9]]

2、如果partition\_strategy为”div”,那么分配方案为：

[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10], [11, 12]]

tf.nn.embedding\_lookup\_sparse(params, sp\_ids, sp\_weights, partition\_strategy='mod', name=None, combiner=None, max\_norm=None)

对给定的ids和权重查询embedding

1、sp\_ids为一个N x M的稀疏tensor，

N为batch大小，M为任意，数据类型int64

2、sp\_weights的shape与sp\_ids的稀疏tensor权重，

浮点类型，若为None，则权重为全’1’

## RNN

tf.nn.dynamic\_rnn(cell, inputs, sequence\_length=None, initial\_state=None, dtype=None, parallel\_iterations=None, swap\_memory=False, time\_major=False, scope=None)

tf.nn.rnn(cell, inputs, initial\_state=None, dtype=None, sequence\_length=None, scope=None)

tf.nn.state\_saving\_rnn(cell, inputs, state\_saver, state\_name, sequence\_length=None, scope=None)

tf.nn.bidirectional\_dynamic\_rnn(cell\_fw, cell\_bw, inputs, sequence\_length=None, initial\_state\_fw=None, initial\_state\_bw=None, dtype=None, parallel\_iterations=None, swap\_memory=False, time\_major=False, scope=None)

tf.nn.bidirectional\_rnn(cell\_fw, cell\_bw, inputs, initial\_state\_fw=None, initial\_state\_bw=None, dtype=None, sequence\_length=None, scope=None)

tf.nn.raw\_rnn(cell, loop\_fn, parallel\_iterations=None, swap\_memory=False, scope=None)

## Evaluation

tf.nn.top\_k(input, k, name=None) 返回前k大的值及其对应的索引

tf.nn.in\_top\_k(predictions, targets, k, name=None)

该函数就判断targets所代表的类别在predictions中是否排在前k个，如果是，那就返回true

correct = tf.nn.in\_top\_k(logits, labels, 1)

#bool转变成int32，并求和

return tf.reduce\_sum(tf.cast(correct, tf.int32)

## Sampled Loss Functions

为了更快的训练，tensorflow提供了采样的 Loss Functions

# 多分类

tf.nn.sampled\_softmax\_loss(weights, biases, inputs, labels, num\_sampled, num\_classes, num\_true=1, sampled\_values=None, remove\_accidental\_hits=True, name='sampled\_softmax\_loss')

# 噪声对比估计(NCE)

tf.nn.nce\_loss(weights, biases, inputs, labels, num\_sampled, num\_classes, num\_true=1, sampled\_values=None, remove\_accidental\_hits=False, name='nce\_loss')

## Candidate Samplers(候选采样)

NCE\_LOSS的负采样，就用了log\_uniform\_candidate\_sampler方法。

tf.nn.uniform\_candidate\_sampler(true\_classes, num\_true,

num\_sampled, unique, range\_max, seed=None, name=None)

通过均匀分布的采样集合

返回三元tuple

1、sampled\_candidates 候选集合。

2、期望的true\_classes个数，为浮点值

3、期望的sampled\_candidates个数，为浮点值

tf.nn.log\_uniform\_candidate\_sampler(true\_classes, num\_true,num\_sampled, unique, range\_max, seed=None, name=None)

通过log均匀分布的采样集合，返回三元tuple

tf.nn.learned\_unigram\_candidate\_sampler(true\_classes, num\_true, num\_sampled, unique, range\_max, seed=None, name=None)

根据在训练过程中学习到的分布状况进行采样,返回三元tuple

tf.nn.fixed\_unigram\_candidate\_sampler(true\_classes, num\_true,num\_sampled, unique, range\_max, vocab\_file=”, distortion=1.0, num\_reserved\_ids=0, num\_shards=1, shard=0, unigrams=(), seed=None, name=None)

基于所提供的基本分布进行采样

# train

## Optimizers

class tf.train.Optimizer

tf.train.Optimizer.\_\_init\_\_(use\_locking, name)

tf.train.Optimizer.minimize(loss, global\_step=None, var\_list=None, gate\_gradients=1, aggregation\_method=None, colocate\_gradients\_with\_ops=False, name=None, grad\_loss=None)

tf.train.Optimizer.compute\_gradients(loss, var\_list=None, gate\_gradients=1, aggregation\_method=None, colocate\_gradients\_with\_ops=False, grad\_loss=None)

tf.train.Optimizer.apply\_gradients(grads\_and\_vars, global\_step=None, name=None)

opt = GradientDescentOptimizer(learning\_rate=0.1)

opt\_op = opt.minimize(cost, var\_list=<list of variables>)

opt\_op.run()

# Compute the gradients for a list of variables.

grads\_and\_vars = opt.compute\_gradients(loss, <list of variables>)

# grads\_and\_vars is a list of tuples (gradient, variable). Do whatever you

# need to the 'gradient' part, for example cap them, etc.

capped\_grads\_and\_vars = [(MyCapper(gv[0]), gv[1]) for gv in grads\_and\_vars]

# Ask the optimizer to apply the capped gradients.

opt.apply\_gradients(capped\_grads\_and\_vars)

子类，subclass

class tf.train.GradientDescentOptimizer

tf.train.GradientDescentOptimizer.\_\_init\_\_(learning\_rate, use\_locking=False, name='GradientDescent')

class tf.train.AdadeltaOptimizer

tf.train.AdadeltaOptimizer.\_\_init\_\_(learning\_rate=0.001, rho=0.95, epsilon=1e-08, use\_locking=False, name='Adadelta')

class tf.train.AdagradOptimizer

tf.train.AdagradOptimizer.\_\_init\_\_(learning\_rate, initial\_accumulator\_value=0.1, use\_locking=False, name='Adagrad')

class tf.train.MomentumOptimizer

tf.train.MomentumOptimizer.\_\_init\_\_(learning\_rate, momentum, use\_locking=False, name='Momentum', use\_nesterov=False)

class tf.train.AdamOptimizer

tf.train.AdamOptimizer.\_\_init\_\_(learning\_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08, use\_locking=False, name='Adam')

class tf.train.FtrlOptimizer

tf.train.FtrlOptimizer.\_\_init\_\_(learning\_rate, learning\_rate\_power=-0.5, initial\_accumulator\_value=0.1, l1\_regularization\_strength=0.0, l2\_regularization\_strength=0.0, use\_locking=False, name='Ftrl')

class tf.train.RMSPropOptimizer

tf.train.RMSPropOptimizer.\_\_init\_\_(learning\_rate, decay=0.9, momentum=0.0, epsilon=1e-10, use\_locking=False, centered=False, name='RMSProp')

例如：

#生成一个object

tf.train.AdamOptimizer(leanring\_rate)

#调用其中的minimize方法。

tf.train.AdamOptimizer(leanring\_rate).minimize(loss)

## Gradient Computation

tf.gradients(ys, xs, grad\_ys=None, name='gradients', colocate\_gradients\_with\_ops=False, gate\_gradients=False, aggregation\_method=None)

class tf.AggregationMethod

tf.stop\_gradient(input, name=None)

tf.hessians(ys, xs, name='hessians', colocate\_gradients\_with\_ops=False, gate\_gradients=False, aggregation\_method=None)

## Clipping（裁剪）

#按指定的范围[clip\_value\_min, clip\_value\_max]裁剪，一般避免出现log(0)

tf.clip\_by\_value(t, clip\_value\_min, clip\_value\_max, name=None)

tf.clip\_by\_norm(t, clip\_norm, name=None)

tf.clip\_by\_average\_norm(t, clip\_norm, name=None)

tf.clip\_by\_global\_norm(t\_list, clip\_norm, use\_norm=None, name=None)

tf.global\_norm(t\_list, name=None)

## Decaying the learning rate(衰减学习率)

tf.train.exponential\_decay(learning\_rate, global\_step, decay\_steps, decay\_rate, staircase=False, name=None)

指数衰减,

公式是：

decayed\_learning\_rate=learning\_rate \* decay\_rate ^( global\_step/decay\_steps)

staircase=True时，( global\_step/decay\_steps)会被转成整数。

## Moving Averages（移动平均）

class tf.train.ExponentialMovingAverage

## Coordinator and QueueRunner（协调和队列执行）

class tf.train.Coordinator

class tf.train.QueueRunner

tf.train.add\_queue\_runner(qr, collection='queue\_runners')

tf.train.start\_queue\_runners(sess=None, coord=None, daemon=True, start=True, collection='queue\_runners')

## Distributed execution

class tf.train.Server

tf.train.Server.\_\_init\_\_(server\_or\_cluster\_def, job\_name=None, task\_index=None, protocol=None, config=None, start=True)

tf.train.Server.create\_local\_server(config=None, start=True)

tf.train.Server.target

tf.train.Server.server\_def

tf.train.Server.start()

tf.train.Server.join()

class tf.train.Supervisor

class tf.train.SessionManager

## Summary Operations

tf.scalar\_summary(tags, values, collections=None, name=None)

tf.image\_summary(tag, tensor, max\_images=None, collections=None, name=None)

tf.histogram\_summary(tag, values, collections=None, name=None)

tf.nn.zero\_fraction(value, name=None)

tf.merge\_summary(inputs, collections=None, name=None)

tf.merge\_all\_summaries(key='summaries')

收集当前图的所有信息。

Adding Summaries to Event Files

class tf.train.SummaryWriter

tf.train.SummaryWriter.add\_summary(summary, global\_step=None)

把收集到的summary和step信息添加到缓冲区。

tf.train.SummaryWriter.flush()

把把缓冲区的内容更新记录到之前创立的硬盘文件中。

## Training utilities（公共功能）

tf.train.global\_step(sess, global\_step\_tensor)

tf.train.write\_graph(graph\_def, logdir, name, as\_text=True)

# Sparse Tensors

## Sparse Tensor Representation表示

class tf.SparseTensor

class tf.SparseTensorValue

## Sparse to Dense Conversion

tf.sparse\_to\_dense(sparse\_indices, output\_shape, sparse\_values, default\_value, name=None)

tf.sparse\_tensor\_to\_dense(sp\_input, default\_value, name=None)

tf.sparse\_to\_indicator(sp\_input, vocab\_size, name=None)

## Manipulation操控

tf.sparse\_concat(concat\_dim, sp\_inputs, name=None)

tf.sparse\_reorder(sp\_input, name=None)

tf.sparse\_retain(sp\_input, to\_retain)

tf.sparse\_fill\_empty\_rows(sp\_input, default\_value, name=None)

# control flow

## Control Flow Operations

tf.identity(input, name=None)

tf.tuple(tensors, name=None, control\_inputs=None)

tf.group(\*inputs, \*\*kwargs)

#在训练神经网络模型时，既要更新w，又要更新它的滑动平均值，这时就可以用tf.group

tf.no\_op(name=None)

tf.count\_up\_to(ref, limit, name=None)

tf.cond(pred, fn1, fn2, name=None)

# Return either fn1() or fn2() based on the boolean predicate `pred`.

tf.case(pred\_fn\_pairs, default, exclusive=False, name='case')

tf.while\_loop(cond, body, loop\_vars, shape\_invariants=None, parallel\_iterations=10, back\_prop=True, swap\_memory=False, name=None)

## Logical Operators

tf.logical\_and(x, y, name=None)

tf.logical\_not(x, name=None)

tf.logical\_or(x, y, name=None)

tf.logical\_xor(x, y, name='LogicalXor')

## Comparison Operators

tf.equal(x, y, name=None)

tf.not\_equal(x, y, name=None)

tf.less(x, y, name=None)

tf.less\_equal(x, y, name=None)

tf.greater(x, y, name=None) # max(x,y)

tf.greater\_equal(x, y, name=None)

tf.select(condition, t, e, name=None)

tf.where(condition, x=None, y=None, name=None)

# Inputs and Readers

## Placeholders

tf.placeholder(dtype, shape=None, name=None)

tf.placeholder\_with\_default(input, shape, name=None)

tf.sparse\_placeholder(dtype, shape=None, name=None)

## Readers

class tf.ReaderBase

#

class tf.TextLineReader

tf.TextLineReader.\_\_init\_\_(skip\_header\_lines=None, name=None)

tf.TextLineReader.num\_records\_produced(name=None)

tf.TextLineReader.read(queue, name=None) # return A tuple of Tensors (key, value).

tf.TextLineReader.read\_up\_to(queue, num\_records, name=None) #读取最多num条记录

#

class tf.WholeFileReader

class tf.IdentityReader

class tf.TFRecordReader

class tf.FixedLengthRecordReader

## Converting

转化为tensor，

# decode\_csv将文本转化为tensor

tf.decode\_csv(records, record\_defaults, field\_delim=None, name=None)

# decode将pb转化为tensor

tf.decode\_raw(bytes, out\_type, little\_endian=None, name=None)

## Example protocol buffer

class tf.VarLenFeature

class tf.FixedLenFeature

class tf.FixedLenSequenceFeature

tf.parse\_example(serialized, features, name=None, example\_names=None)

tf.parse\_single\_example(serialized, features, name=None, example\_names=None)

tf.parse\_tensor(serialized, out\_type, name=None)

tf.decode\_json\_example(json\_examples, name=None)

## Queues

class tf.QueueBase

基本的操作方法（子类继承了这些方法）

tf.QueueBase.\_\_init\_\_(dtypes, shapes, names, queue\_ref)

#初始化队列的元素

tf.QueueBase.enqueue\_many(vals, name=None)

#入队

tf.QueueBase.enqueue(vals, name=None)

#出对

tf.QueueBase.dequeue(name=None)

class tf.FIFOQueue

class tf.PaddingFIFOQueue

class tf.RandomShuffleQueue

class tf.PriorityQueue

example:

#创建一个先进先出的队列，指定队列中做多可以保存2个元素，元素类型为int32

q=tf.FIFOQueue(2,"int32")

#

init=q.enqueue\_many(([0,10],))

#

x=q.dequeue()

print(x)

y=x+1

q\_inc=q.enqueue([y])

with tf.Session() as sess:

init.run()

for i in range(5):

v,i=sess.run([x,q\_inc])

print(v)

## Conditional Accumulators

class tf.ConditionalAccumulatorBase

class tf.ConditionalAccumulator

class tf.SparseConditionalAccumulator

## Dealing with the filesystem

tf.matching\_files(pattern, name=None)

tf.read\_file(filename, name=None)

tf.write\_file(filename, contents, name=None)

## Input pipeline

tf.train.match\_filenames\_once(pattern, name=None)

tf.train.string\_input\_producer(string\_tensor, num\_epochs=None, shuffle=True, seed=None, capacity=32, shared\_name=None, name=None, cancel\_op=None) # capacity翻译为容量

tf.train.batch(tensors, batch\_size, num\_threads=1, capacity=32, enqueue\_many=False, shapes=None, dynamic\_pad=False, allow\_smaller\_final\_batch=False, shared\_name=None, name=None)

tf.train.shuffle\_batch(tensors, batch\_size, capacity, min\_after\_dequeue, num\_threads=1, seed=None, enqueue\_many=False, shapes=None, allow\_smaller\_final\_batch=False, shared\_name=None, name=None)

tf.train.batch\_join(tensors\_list, batch\_size, capacity=32, enqueue\_many=False, shapes=None, dynamic\_pad=False, allow\_smaller\_final\_batch=False, shared\_name=None, name=None)

tf.train.shuffle\_batch\_join(tensors\_list, batch\_size, capacity, min\_after\_dequeue, seed=None, enqueue\_many=False, shapes=None, allow\_smaller\_final\_batch=False, shared\_name=None, name=None)