# Tensorflow Python API 翻译（math_ops）（第一部分）

**算术运算符**

TensorFlow提供了一些操作，你可以使用基本的算术运算符添加到你的图表。

---

`tf.add(x, y, name = None)`

解释：这个函数返回x与y逐元素相加的结果。

注意：tf.add操作支持广播形式，但是tf.add_n操作不支持广播形式。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant(2)
b = tf.constant(3)
c = tf.add(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是必须是以下之一：`float32`，`float64`，`int8`，`int16`，`int32`，`complex64`，`int64`。
- `y`: 一个`Tensor`，数据类型必须和`x`相同。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同。

---

`tf.sub(x, y, name = None)`

解释：这个函数返回x与y逐元素相减的结果。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,2])
b = tf.constant(2)
c = tf.sub(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是必须是以下之一：`float32`，`float64`，`int8`，`int16`，`int32`，`complex64`，`int64`。

- y: 一个Tensor，数据类型必须和x相同。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.mul(x, y, name = None)**

解释：这个函数返回x与y逐元素相乘的结果。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,2])
b = tf.constant(2)
c = tf.mul(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之

一：float32，float64，int8，int16，int32，complex64，int64。

- y: 一个Tensor，数据类型必须和x相同。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.div(x, y, name = None)**

解释：这个函数返回x与y逐元素相除的结果。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,2])
b = tf.constant(2)
c = tf.div(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之

一：float32，float64，int8，int16，int32，complex64，int64。

- y: 一个Tensor，数据类型必须和x相同。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

`tf.mod(x, y, name = None)`

解释：这个函数返回x与y逐元素取余的结果。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,2])
b = tf.constant(2)
c = tf.mod(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是必须是以下之一：`int16`，`int32`，`float32`，`float64`。
- `y`: 一个`Tensor`，数据类型必须和`x`相同。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同。

---

**基础的数学函数**

TensorFlow提供了一些操作，你可以使用基本的数学函数，将它们添加到你的图表。

---

`tf.add_n(inputs, name = None)`

解释：这个函数的作用是对inputs列表中的元素相应位置累加。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,2], tf.int32)
b = tf.constant([3,4], tf.int32)
c = tf.add_n([a,b])
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `inputs`: 一个列表，其中至少有一个`Tensor`，数据类型是必须是以下之一：`float32`，`float64`，`int64`，`int32`，`uint8`，`int16`，`int8`，`complex64`，`qint8`，`quint8`，`quint32`。并且列表中的每个`Tensor`必须有相同的数据维度。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`inputs`相同。

---

`tf.abs(x, name = None)`

解释：这个函数的作用返回`x`的绝对值。

给定`x`，它是一个实数`Tensor`。这个操作返回一个`tensor`，这个`tensor`中的每个值是对应于`x`中的每个值得绝对值。

如果，你需要处理复数的绝对值，那么可以使用`tf.complex_abs()`函数。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,-2])
c = tf.abs(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之一：float，double，int64或者int32。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和数据维度都和x相同。

---

**tf.neg(x, name = None)**

解释：这个函数的作用是得到x中每个值得负数，即y = -x

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,-2])
c = tf.neg(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.sign(x, name = None)**

解释：这个函数是一个符号函数，按照如下规则转换x中的每一个值。

如果 x < 0，y = sign(x) = -1；

如果 x == 0，y = sign(x) = 0；

如果 x > 0，y = sign(x) = 1；

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([1,-2,0])
c = tf.sign(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之一：float32，float64，int32，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.inv(x, name = None)**

解释：这个函数是计算x中每个元素的倒数，即y = 1/x。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant(7.0)
c = tf.inv(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

译者注：

我测试了一下这个API，但好像x的数据类型只有是float类型时才能成功。

---

**tf.square(x, name = None)**

解释：这个函数是计算x中每个元素的平方，即y = x*x = x^2。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.0,7.0])
c = tf.square(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.round(x, name = None)**

解释：这个函数是得到x中每个元素离它最接近的整数。

比如：

```
# 'a' is [0.9, 2.5, 2.3, -4.4]
tf.round(a) ==> [ 1.0, 3.0, 2.0, -4.0 ]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.9, 0.0, -2.1, 2.0, 7.2])
c = tf.round(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是float或者double。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同，数据维度和x相同。

---

**tf.sqrt(x, name = None)**

解释：这个函数是得到x中每个元素的开平方值，即y = x^{1/2}。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2, 3], tf.float32)
c = tf.sqrt(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.rsqrt(x, name = None)**

解释：这个函数是得到x中每个元素的开平方值的导数，即y = 1/x^{1/2}。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2, 3], tf.float32)
c = tf.rsqrt(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。

- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.pow(x, y, name = None)**

解释：这个函数是计算幂运算。

给定一个x和y，对应x和y中的每一个值，计算x^y。

比如：

```
# tensor 'x' is [[2, 2]], [3, 3]]
# tensor 'y' is [[8, 16], [2, 3]]
tf.pow(x, y) ==> [[256, 65536], [9, 27]]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2, 3])
b = tf.constant([2, 3])
c = tf.pow(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float，double，int32，complex64，int64。
- y: 一个Tensor，数据类型必须是以下之一：float，double，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor。

---

**tf.exp(x, name = None)**

解释：这个函数是计算x中每个元素的指数，即y = e^x。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.0, 1], tf.float32)
c = tf.exp(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.log(x, name = None)**

解释：这个函数是计算x中每个元素的自然对数，即y = log(x)。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.0, 1], tf.float32)
c = tf.log(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.ceil(x, name = None)**

解释：这个函数是返回不小于x中每个元素的最小整数。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8], tf.float32)
c = tf.ceil(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.floor(x, name = None)**

解释：这个函数是返回不大于x中每个元素的最大整数。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8], tf.float32)
c = tf.floor(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64。

- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同。

---

**`tf.maximum(x, y, name = None)`**

解释：这个函数是逐个比较`x`和`y`中的值，求得最大值，即`x > y ? x : y`。该函数支持广播形式。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8, 0.0])
b = tf.constant(1.0)
c = tf.maximum(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型必须是以下之一：`float32`，`float64`，`int32`，`int64`。
- `y`: 一个`Tensor`，数据类型和`x`相同。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同。

---

**`tf.minimum(x, y, name = None)`**

解释：这个函数是逐个比较`x`和`y`中的值，求得最小值，即`x < y ? x : y`。该函数支持广播形式。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8, 0.0])
b = tf.constant(1.0)
c = tf.minimum(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型必须是以下之一：`float32`，`float64`，`int32`，`int64`。
- `y`: 一个`Tensor`，数据类型和`x`相同。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同。

---

**`tf.cos(x, name = None)`**

解释：这个函数是计算`x`中每个元素的余弦值。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8, 0.0])
c = tf.cos(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**tf.sin(x, name = None)**

解释：这个函数是计算x中每个元素的正弦值。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8, 0.0])
c = tf.sin(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型必须是以下之一：float32，float64，int32，complex64，int64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和x相同。

---

**矩阵数学函数**

TensorFlow提供了一些操作，你可以使用添加基本的矩阵数学函数在你的图中。

---

**tf.diag(diagonal, name = None)**

解释：这个函数是给定一个对角值diagonal，然后返回一个对角tensor。

给定一个对角值diagonal，这个操作返回一个对角tensor，对角线上面的值是diagonal，其余值都用0来填充。

假设diagonal的维度为[D1, D2, ..., Dk]，那么输出tensor的秩为2k，维度是[D1, D2, ..., Dk, D1, D2, ..., Dk]，如下：

```
output[i1, i2, ..., ik, i1, i2, ..., ik] = diagonal[i1, .., ik]，其余值都是0。
```

比如：

```
# '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]]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

a = tf.constant([2.2, -1.8, 1.0])
c = tf.diag(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `diagonal`: 一个`Tensor`，数据类型必须是以下之一：`float32`，`float64`，`int32`，`int64`。它的秩最大为3。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`diagonal`相同。

---

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

解释：将`a`进行转置，并且根据`perm`参数重新排列输出维度。

输出数据`tensor`的第`i`维将根据`perm[i]`指定。比如，如果`perm`没有给定，那么默认是`perm = [n-1, n-2, ..., 0]`，其中`rank(a) = n`。默认情况下，对于二维输入数据，其实就是常规的矩阵转置操作。

比如：

```
input_data.dims = (1, 4, 3)
perm = [1, 2, 0]

# 因为 output_data.dims[0] = input_data.dims[ perm[0] ]
# 因为 output_data.dims[1] = input_data.dims[ perm[1] ]
# 因为 output_data.dims[2] = input_data.dims[ perm[2] ]
# 所以得到 output_data.dims = (4, 3, 1)
output_data.dims = (4, 3, 1)
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

sess = tf.Session()
input_data = tf.constant([[1,2,3],[4,5,6]])
print sess.run(tf.transpose(input_data))
print sess.run(input_data)
print sess.run(tf.transpose(input_data, perm=[1,0]))
input_data = tf.constant([[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]])
print 'input_data shape: ', sess.run(tf.shape(input_data))
output_data = tf.transpose(input_data, perm=[1, 2, 0])
print 'output_data shape: ', sess.run(tf.shape(output_data))
print sess.run(output_data)
sess.close()
```

输入参数：

- `a`: 一个`Tensor`。
- `perm`: 一个对于`a`的维度的重排列组合。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个经过翻转的`Tensor`。

---

**`tf.matmul(a, b, transpose_a = False, transpose_b = False, a_is_sparse = False, b_is_sparse = False, name = None)`**

解释：将矩阵`a`和矩阵`b`进行相乘，得到矩阵`a*b`。

输入数据必须是一个二维的矩阵，经过转置或者不转置，内部维度必须相匹配。

输入矩阵必须有相同的数据类型，数据类型为：`float`，`double`，`int32`，`complex64`。

矩阵可以被设置为转置操作，即`transpose_a = True, transpose_b = True`。默认情况下，该标记都是被设置为`False`。

如果矩阵中存在很多的`0`，那么我们可以使用`sparse`标记，即`a_is_sparse = True, b_is_sparse = True`。默认情况下，该标记都是被设置为`False`。

比如：

```
# 2-D tensor `a`
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3]) => [[1. 2. 3.]
                                                      [4. 5. 6.]]
# 2-D tensor `b`
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2]) => [[7. 8.]
                                                         [9. 10.]
                                                         [11. 12.]]
c = tf.matmul(a, b) => [[58 64]
                        [139 154]]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(2,3))
b = tf.constant(np.random.rand(1,3))
c = tf.matmul(a, b, transpose_b = True)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `a`: 一个`Tensor`，数据类型是`float`，`double`，`int32`或者`complex64`。
- `b`: 一个`Tensor`，数据类型和`a`相同。
- `transpose_a`: 如果该值维`True`，那么在矩阵计算之前，先将`a`进行转置。
- `transpose_b`: 如果该值维`True`，那么在矩阵计算之前，先将`b`进行转置。
- `a_is_sparse`: 如果该值维`True`，那么在矩阵计算的时候，将`a`当做一个`sparse`矩阵考虑。
- `b_is_sparse`: 如果该值维`True`，那么在矩阵计算的时候，将`b`当做一个`sparse`矩阵考虑。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`a`相同。

---

**`tf.batch_matmul(x, y, adj_x = None, adj_y = None, name = None)`**

解释：这个函数的作用是将两个张量按批切片进行相乘。

将张量`x`和`y`进行切片（每个切片就是一个批的元素），然后将对应的`x`和`y`的每个切片进行相乘，将得到的结果按照原来批的大小进行重新安排。如果我们把`adj_x`或者`adj_y`设置成`True`，在做乘法之前，每个独立的切片可以组成它的共轭（其实相当于转置）。

输入的`x`和`y`是三维`tensor`，或者更高维度的`[..., r_x, c_x]`和`[..., r_y, c_y]`。

输出`tensor`是一个三维的，或者更高维度的`[..., r_o, c_o]`，其中：

```
r_o = c_x if adj_x else r_x
c_o = r_y if adj_y else c_y
```

计算过程如下：

```
out[..., :, :] = matrix(x[..., :, :]) * matrix(y[..., :, :])
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(2, 2, 3))
b = tf.constant(np.random.rand(3, 3, 1))
c = tf.batch_matmul(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(3, 2, 3, 1))
b = tf.constant(np.random.rand(3, 2, 3, 1))
c = tf.batch_matmul(a, b, adj_x = False, adj_y = True )
sess = tf.Session()
print sess.run(c)
print sess.run(tf.shape(c))
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是`float32`，`float64`，`int32`或者`complex64`，数据维度是三维的，或者更高维度`[..., r_x, c_x]`。
- `y`: 一个`Tensor`，数据类型和`x`相同，数据维度是三维的，或者更高维度`[..., r_y, c_y]`。
- `adj_x`: 这是一个可选的布尔类型的值，默认情况下是`False`。如果我们设置为`True`，`x`的每个切片将进行转置。
- `adj_y`: 这是一个可选的布尔类型的值，默认情况下是`False`。如果我们设置为`True`，`y`的每个切片将进行转置。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`x`相同，数据维度是三维的，或者更高维度`[..., r_o, c_o]`。

---

**`tf.matrix_determinant(input, name=None)`**

解释：这个函数的作用是计算`n`阶矩阵的行列式。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(3, 3))
c = tf.matrix_determinant(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `input`: 一个`Tensor`，数据类型是`float32`或者`float64`，数据维度是`[M, M]`。

- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor的标量，数据类型和input相同。

---

**tf.batch_matrix_determinant(input, name=None)**

解释：这个函数的作用是计算每个批（切片）的n阶矩阵的行列式。

输入tensor的数据维度必须是[..., M, M]，其中内部必须是一个二维的方阵，对于所有的子矩阵，输出结果是一个一维的tensor。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(4, 2, 3, 3))
c = tf.batch_matrix_determinant(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- input: 一个Tensor，数据类型是float32或者float64，数据维度是[..., M, M]。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和input相同，数据维度是[...]。

---

**tf.matrix_inverse(input, name=None)**

解释：这个函数的作用是计算n阶矩阵的逆矩阵，并且检查可逆性。

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(3, 3))
c = tf.matrix_inverse(a)
sess = tf.Session()
print sess.run(c)
d = tf.matmul(a, c)
print sess.run(d)
e = tf.matrix_determinant(d)
print sess.run(e)
sess.close()
```

输入参数：

- input: 一个Tensor，数据类型是float32或者float64，数据维度是[M, M]。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型和input相同，数据维度是[M, M]。

---

**tf.batch_matrix_inverse(input, name=None)**

解释：这个函数的作用是计算每个批（切片）的`n`阶矩阵的逆矩阵，并且检查可逆性。
输入`tensor`的数据类型是`[..., M, M]`，其中内部必须是一个二维的方阵，对于每一个子矩阵，输出的矩阵的逆和输入数据有相同的数据维度。
使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant(np.random.rand(2, 3, 3))
c = tf.batch_matrix_inverse(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `input`: 一个`Tensor`，数据类型是`float32`或者`float64`，数据维度是`[..., M, M]`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`input`相同，数据维度是`[..., M, M]`。

---

**`tf.cholesky(input, name=None)`**
解释：这个函数的作用是计算`n`阶矩阵的Cholesky分解。
输入数据必须是一个对称的正定矩阵，并且这个操作我们只会读入矩阵的下三角部分，不会读取矩阵的上三角部分。
输出结果是经过Cholesky分解之后的一个对角线元素为正数的下三角实矩阵。
使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([[2, np.random.rand()], [-2, 5]], tf.float32)
c = tf.cholesky(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `input`: 一个`Tensor`，数据类型是`float32`或者`float64`，数据维度是`[M, M]`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`input`相同，数据维度是`[M, M]`。

---

**`tf.batch_cholesky(input, name=None)`**
解释：这个函数的作用是计算每个批（切片）的`n`阶矩阵的Cholesky分解。
输入`tensor`的数据类型是`[..., M, M]`，其中内部必须是一个二维的方阵，并且满足Cholesky分解的条件。输出`tensor`和输入数据有相同的数据类型，并且每个切片都是经过Cholesky分解之后的值。
使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
import tensorflow as tf
import numpy as np

a = tf.constant([[[2, np.random.rand()], [-2, 5]], [[2, np.random.rand()], [-2, 5]]], tf.float32)
c = tf.batch_cholesky(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `input`: 一个`Tensor`，数据类型是`float32`或者`float64`，数据维度是`[..., M, M]`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型和`input`相同，数据维度是`[..., M, M]`。

---

**复数函数**

TensorFlow提供了一些复数函数，使得你可以去操作复数，你可以将它们添加到你的图表。

---

**`tf.complex(real, imag, name=None)`**

解释：这个函数的作用是将两个实数转换成一个复数。

这个操作就是去计算复数`a + bj`，其中`a`来自于输入数据`real`，表示实部，`b`来自于输入数据`imag`，表示虚部。

输入数据`real`和`imag`必须拥有相同的数据维度。

比如：

```
# tensor 'real' is [2.25, 3.25]
# tensor `imag` is [4.75, 5.75]
tf.complex(real, imag) ==> [[2.25 + 4.74j], [3.25 + 5.75j]]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([2.25, 3.25])
b = tf.constant([4.75, 5.75])
c = tf.complex(a, b)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `real`: 一个`Tensor`，数据类型是`float`。
- `imag`: 一个`Tensor`，数据类型是`float`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型是`complex64`。

---

**`tf.complex_abs(x, name=None)`**

解释：这个函数的作用是计算复数的绝对值。

给定一个复数张量`x`，这个操作是计算`x`中的每个值的绝对值，并且返回一个`float`类型的张量。在`x`中的所有复数的形式必须是`a + bj`的形式，那么绝对值计算公式如下：

$$\sqrt{a^2+b^2}$$

比如：

```
# tensor 'x' is [[-2.25 + 4.75j], [-3.25 + 5.75j]]
tf.complex_abs(x) ==> [5.25594902, 6.60492229]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([2.25 + 3.25j])
c = tf.complex_abs(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是`complex64`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型是`float32`。

---

**`tf.conj(in_, name=None)`**

解释：这个函数的作用是计算复数的复共轭。

给定一个复数张量`in_`，这个操作是计算`in_`中的每一个复数的复共轭。在`in_`中所有复数的形式必须是`a + bj`的形式，其中`a`是实数部分，`b`是虚数部分。

经过这个操作，复共轭的返回形式是`a - bj`。

比如：

```
# tensor 'in' is [-2.25 + 4.75j, 3.25 + 5.75j]
tf.conj(in) ==> [-2.25 - 4.75j, 3.25 - 5.75j]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([2.25 + 3.25j])
c = tf.conj(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- `x`: 一个`Tensor`，数据类型是`complex64`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，数据类型是`complex64`。

---

**`tf.imag(in_, name=None)`**

解释：这个函数的作用是返回复数的虚部。

给定一个复数张量`in_`，这个操作是返回`in_`中的每一个复数的虚部。在`in_`中所有复数的形式必须是`a + bj`的

形式，其中a是实数部分，b是虚数部分。

比如：

```
# tensor 'in' is [-2.25 + 4.75j, 3.25 + 5.75j]
tf.imag(in) ==> [4.75, 5.75]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([2.25 + 3.25j])
c = tf.imag(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是complex64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型是float32。

---

**tf.real(in_, name=None)**

解释：这个函数的作用是返回复数的实部。

给定一个复数张量in_，这个操作是返回in_中的每一个复数的实部。在in_中所有复数的形式必须是a + bj的形式，其中a是实数部分，b是虚数部分。

比如：

```
# tensor 'in' is [-2.25 + 4.75j, 3.25 + 5.75j]
tf.real(in) ==> [-2.25, 3.25]
```

使用例子：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.constant([2.25 + 3.25j])
c = tf.real(a)
sess = tf.Session()
print sess.run(c)
sess.close()
```

输入参数：

- x: 一个Tensor，数据类型是complex64。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型是float32。