Tensorflow Python API 翻译（constant_op）

该章介绍有关常量张量，序列操作，随机数张量的API

常量张量

Tensorflow提供了很多的操作，去帮助你构建常量。

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

解释：这个函数返回一个全是零的张量，数据维度是 `shape`，数据类型是 `dtype`。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.zeros(shape = [2, 3], dtype = tf.int32, name = "input_data")
print sess.run(data)
```

输入参数：

- `shape`: 一个整型的数组，或者一个一维的Tensor，数据类型是：`int32`。
- `dtype`: 输出结果`Tensor`的数据类型。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，里面的所以数据都是0。

`tf.zeros_like(tensor, dtype = None, name = None)`

解释：这个函数返回一个全是零的张量，数据维度是和`Tensor`一样，数据类型是默认是和`Tensor`一样，但是我们也可以自己指定。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.zeros(shape = [2, 3], dtype = tf.int32, name = "input_data")
d_1 = tf.zeros_like(data)
d_2 = tf.zeros_like(data, tf.float32)
print sess.run(d_1)
print sess.run(d_2)
```

输入参数：

- `tensor`: 一个`Tensor`。
- `dtype`: 输出结果`Tensor`的数据类型，必须是 `float32`，`float64`，`int8`，`int16`，`int32`，`int64`，`uint8`或者`complex64`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，里面的所以数据都是0。

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

解释：这个函数返回一个全是1的张量，数据维度是`shape`，数据类型是`dtype`。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.ones(shape = [2, 3], dtype = tf.int32, name = "input_data")
print sess.run(data)
```

输入参数：

- `shape`: 一个整型的数组，或者一个一维的`Tensor`，数据类型是 `int32`。
- `dtype`: 输出结果`Tensor`的数据类型。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，里面的所以数据都是1。

`tf.ones_like(tensor, dtype = None, name = None)`

解释：这个函数返回一个全是一的张量，数据维度是和`Tensor`一样，数据类型是默认是和`Tensor`一样，但是我们也可以自己指定。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.zeros(shape = [2, 3], dtype = tf.int32, name = "input_data")
d_1 = tf.ones_like(data)
d_2 = tf.ones_like(data, tf.float32)
print sess.run(d_1)
print sess.run(d_2)
```

输入参数：

- `tensor`: 一个`Tensor`。
- `dtype`: 输出结果`Tensor`的数据类型，必须是 `float32`，`float64`，`int8`，`int16`，`int32`，`int64`，`uint8`或者`complex64`。
- `name`:（可选）为这个操作取一个名字。

输出参数：

- 一个`Tensor`，里面的所以数据都是1。

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

解释：这个函数返回一个`Tensor`，数据维度是`dims`，填充的数据都是`value`。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.fill([2,3], 9)
print sess.run(data)
```

输入参数：

- dim: 一个Tensor，数据类型是int32，表示输出数据的维度。
- value: 一个Tensor，数据维度是0维，即是一个常量（标量），输出数据所以填充的都是该值。
- 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value相同。

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

解释：这个函数返回一个常量Tensor。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.constant([1, 2, 3])
print sess.run(data)
data = tf.constant(-1.0, shape = [2, 3])
print sess.run(data)
data = tf.constant(2.0, dtype = tf.float32, shape = [2, 3])
print sess.run(data)
```

输入参数：

- value: 一个常量或者是一个数组，该数据类型就是输出的数据类型。
- dtype: 输出数据的类型。
- shape:（可选）输出数据的维度。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个常量Tensor。

序列操作

Tensorflow提供了一些函数，去帮助我们构建序列。

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

解释：这个函数返回一个序列数组，数组的第一个元素是start，如果num>1，那么序列的最后一个元素就是stop - start / num - 1。也就是说，最后一个元素肯定是stop。

使用例子：

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

import tensorflow as tf
import numpy as np
```

```
sess = tf.Session()
data = tf.linspace(10.0, 15.0, 10)
print sess.run(data)
```
输入参数：

- start: 一个Tensor。数据类型必须是float32或者float64。该值是输出序列的第一个元素。
- stop: 一个Tensor。数据类型必须和start相同。该值是输出序列的最后一个元素。
- num: 一个Tensor，数据类型是int32。该值确定输出序列的个数
- 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start相同，数据维度是一维。

---

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

解释：这个函数返回一个序列数组，数组的第一个元素是start，之后的每一个元素都在前一个元素的基础上，加上delta，直到limit，但是不包括limit。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.range(3, 15, 3)
print sess.run(data)
```
输入参数：

- start: 一个0维的Tensor，即一个标量。数据类型必须是int32。该值是输出序列的第一个元素。
- limit: 一个0维的Tensor，即一个标量。数据类型必须是int32。该值是输出序列的最后限制，但不包含该值。
- delta: 一个0维的Tensor，即一个标量。数据类型必须是int32。（可选）该值默认是1，也就是说输出数据从start开始。
- name:（可选）为这个操作取一个名字。

输出参数：

- 一个Tensor，数据类型int32，数据维度是一维。

随机数张量

Tensorflow提供了一些函数，去帮助我们构建随机数张量。

---

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

解释：这个函数返回一个随机数序列，数组里面的值按照正态分布。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.random_normal([2, 3])
print sess.run(data)
```

输入参数：

- shape: 一个一维的Tensor，或者是一个python数组。该值是确定输出序列的数据维度。
- mean: 一个0维的Tensor，或者一个数据类型是dtype的python值。该值表示正态分布的均值。
- stddev: 一个0维的Tensor，或者一个数据类型是dtype的python值，该值表示正态分布的标准偏差。
- dtype: 输出数据的数据类型。
- seed: 一个python整型，为分布产生一个随机种子，具体可以参见set_random_seed函数。
- name:（可选）为这个操作取一个名字。

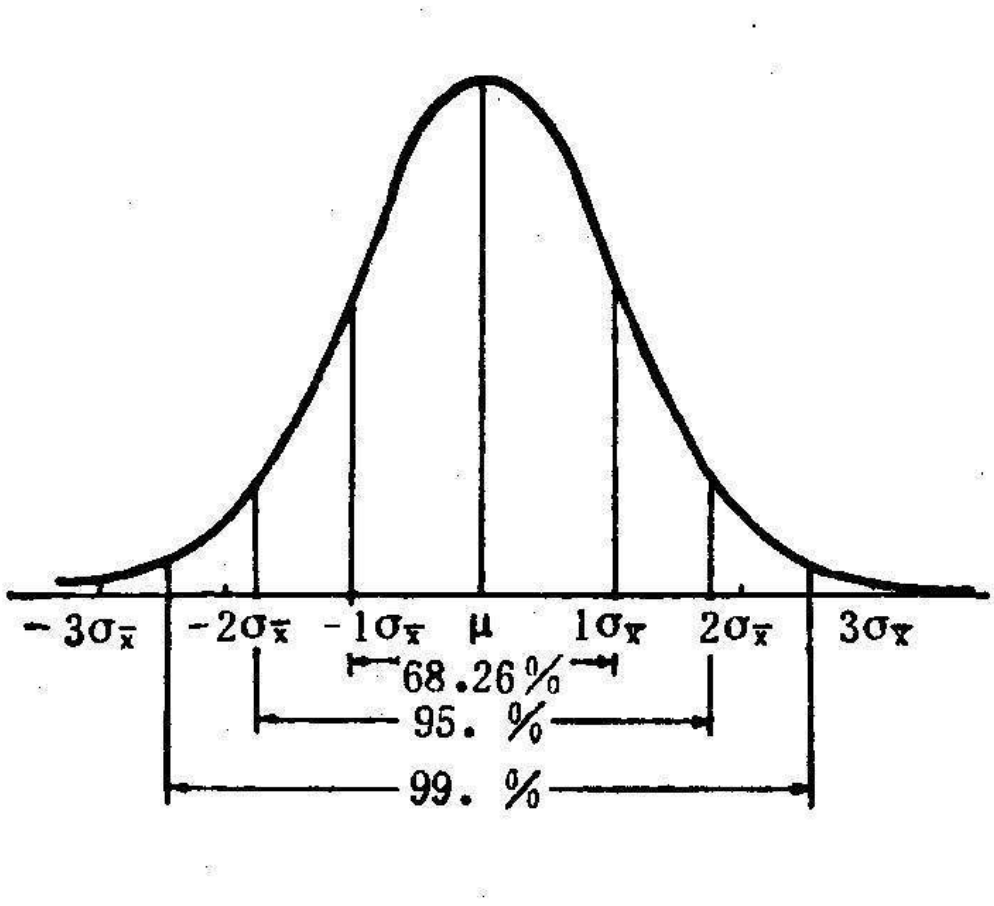
输出参数：

- 一个Tensor，数据类型是dtype，数据维度是shape，里面的值符合正态分布。

---

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

解释：这个函数返回一个随机数序列，数组里面的值按照正态分布，但和random_normal函数不同的是，该值返回的是一个截断的正态分布类型。也就是说，产生出来的值范围都是在 [mean - 2 * standard_deviations, mean + 2 * standard_deviations]内，下图可以告诉你这个具体范围在哪。



truncated_normal

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.truncated_normal([2, 3])
print sess.run(data)
```

输入参数：

- shape: 一个一维的Tensor，或者是一个python数组。该值是确定输出序列的数据维度。
- mean: 一个0维的Tensor，或者一个数据类型是dtype的python值。该值表示正态分布的均值。
- stddev: 一个0维的Tensor，或者一个数据类型是dtype的python值，该值表示正态分布的标准偏差。
- dtype: 输出数据的数据类型。
- seed: 一个python整型，为分布产生一个随机种子，具体可以参见set_random_seed函数。
- 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dtype，数据维度是shape，里面的值是一个截断的正态分布。

---

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

解释：这个函数返回一个随机数序列，数组里面的值按照均匀分布，数据范围是 [minval, maxval)。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.random_uniform([2, 3])
print sess.run(data)
```

输入参数：

- shape: 一个一维的Tensor，或者是一个python数组。该值是确定输出序列的数据维度。
- minval: 一个0维的Tensor，或者一个数据类型是dtype的python值。该值表示均匀分布的最小值。
- maxval: 一个0维的Tensor，或者一个数据类型是dtype的python值，该值表示均匀分布的最大值，但是不能取到该值。
- dtype: 输出数据的数据类型。
- seed: 一个python整型，为分布产生一个随机种子，具体可以参见set_random_seed函数。
- 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dtype，数据维度是shape，里面的值符合均匀分布。

---

tf.random_shuffle(value, seed = None, name = None)

解释：这个函数返回一个随机数序列，将value中的数据打乱输出。

使用例子：

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

import tensorflow as tf
import numpy as np

sess = tf.Session()
data = tf.constant([[1, 2], [3, 4], [5, 6]])
shuff_data = tf.random_shuffle(data)
print sess.run(data)
print sess.run(shuff_data)

data = tf.constant([1, 2, 3, 4, 5, 6])
```

```
shuff_data = tf.random_shuffle(data)
print sess.run(data)
print sess.run(shuff_data)
```

输入参数：

- `value`: 一个`Tensor`，需要打乱的数据。
- `seed`: 一个python整型，为分布产生一个随机种子，具体可以参见`set_random_seed`函数。
- `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`value`相同。

`tf.set_random_seed(seed)`

解释：这个函数是设置图层面的随机种子。随机种子分为两类，一类是图层面的随机种子，另一类是操作层面的随机种子。具体区别如下：

第一种，如果图层面和操作层面的随机种子都没有设置，那么随机种子将在每个操作中被更新。例子如下：

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

import tensorflow as tf
import numpy as np

a = tf.random_uniform([1])
b = tf.random_normal([1])

print "Session 1"
with tf.Session() as sess1:
    print sess1.run(a)  # generates 'A1'
    print sess1.run(a)  # generates 'A2'
    print sess1.run(b)  # generates 'B1'
    print sess1.run(b)  # generates 'B2'

print "Session 2"
with tf.Session() as sess2:
    print sess2.run(a)  # generates 'A3'
    print sess2.run(a)  # generates 'A4'
    print sess2.run(b)  # generates 'B3'
    print sess2.run(b)  # generates 'B4'
```

第二种，如果图层面的随机种子被设置了，但是操作层面的随机种子没有被设置。那么，系统将把图层面的随机种子设置成操作层面的随机种子，以至于操作层面的随机种子将被确定下来。

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

import tensorflow as tf
import numpy as np

tf.set_random_seed(1234)
a = tf.random_uniform([1])
b = tf.random_normal([1])

# Repeatedly running this block with the same graph will generate different
# sequences of 'a' and 'b'.
print "Session 1"
with tf.Session() as sess1:
    print sess1.run(a)  # generates 'A1'
    print sess1.run(a)  # generates 'A2'
    print sess1.run(b)  # generates 'B1'
    print sess1.run(b)  # generates 'B2'

print "Session 2"
with tf.Session() as sess2:
    print sess2.run(a)  # generates 'A1'
```

```
    print sess2.run(a)  # generates 'A2'
    print sess2.run(b)  # generates 'B1'
    print sess2.run(b)  # generates 'B2'
```

第三种，如果图层面的随机种子没有被设置，但是操作层面的随机种子被设置了，那么被设置随机种子的操作层将有确定的唯一种子，其他操作层不具有唯一种子。

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

import tensorflow as tf
import numpy as np

a = tf.random_uniform([1], seed=1)
b = tf.random_normal([1])

# Repeatedly running this block with the same graph will generate the same
# sequence of values for 'a', but different sequences of values for 'b'.
print "Session 1"
with tf.Session() as sess1:
    print sess1.run(a)  # generates 'A1'
    print sess1.run(a)  # generates 'A2'
    print sess1.run(b)  # generates 'B1'
    print sess1.run(b)  # generates 'B2'

print "Session 2"
with tf.Session() as sess2:
    print sess2.run(a)  # generates 'A1'
    print sess2.run(a)  # generates 'A2'
    print sess2.run(b)  # generates 'B3'
    print sess2.run(b)  # generates 'B4'
```

第四种，如果图层面和操作层面都设置了随机种子，那么这两个随机种子都将被使用，但是最后起作用的随机种子是唯一的，即操作的随机输出值是确定的。

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

import tensorflow as tf
import numpy as np

tf.set_random_seed(1234)
a = tf.random_uniform([1], seed = 1)
b = tf.random_normal([1], seed = 2)

# Repeatedly running this block with the same graph will generate the same
# sequence of values for 'a', but different sequences of values for 'b'.
print "Session 1"
with tf.Session() as sess1:
    print sess1.run(a)  # generates 'A1'
    print sess1.run(a)  # generates 'A2'
    print sess1.run(b)  # generates 'B1'
    print sess1.run(b)  # generates 'B2'

print "Session 2"
with tf.Session() as sess2:
    print sess2.run(a)  # generates 'A1'
    print sess2.run(a)  # generates 'A2'
    print sess2.run(b)  # generates 'B1'
    print sess2.run(b)  # generates 'B2'
```

输入参数：

- `seed`: 一个整数类型。