Tensorflow Python API 翻译（sparse_ops）

该章介绍有关稀疏张量的API

稀疏张量表示

对于多维稀疏数据，TensorFlow提供了稀疏张量表示。稀疏张量里面的值都是采用`IndexedSlices`索引来表示，这样能更加高效的表示数据。

`class tf.SparseTensor`

解释：这个函数的作用是表示一个稀疏张量。

Tensorflow使用三个密集张量：`indices`，`values`，`dense_shape`，来表示一个稀疏张量。在Python接口中，这三个张量被整合到一个`SparseTensor`类中，如果你调换了这三个密集张量的位置，那么在进行操作之前，`SparseTensor`类会自动调换三个张量的位置。

具体的说，稀疏张量表示为`SparseTensor(values, indices, dense_shape)`：

- `indices`: 一个二维的张量，数据类型是`int64`，数据维度是`[N, ndims]`。
- `values`: 一个一维的张量，数据类型是任意的，数据维度是`[N]`。
- `dense_shape`: 一个一维的张量，数据类型是`int64`，数据维度是`[ndims]`。

其中，`N`表示稀疏张量中存在`N`个值，`ndims`表示`SparseTensor`的维度。

相应的密集张量满足：

```
dense.shape = dense_shape
dense[tuple(indices[i])] = values[i]
```

按照惯例，`indices`中的索引应该按照从小到大的顺序排序。`SparseTensor`中三个密集张量的顺序不是强制的，你可以乱序，`SparseTensor`会自动将它排序。

比如：

```
SparseTensor(values=[1, 2], indices=[[0, 0], [1, 2]], shape=[3, 4])
```

那么密集张量就是：

```
[[1, 0, 0, 0]
 [0, 0, 2, 0]
 [0, 0, 0, 0]]
```

`tf.SparseTensor.__init__(indices, values, shape)`

解释：这个函数的作用是构建一个`SparseTensor`。

输入参数：

- `indices`: 一个二维的张量，数据类型是`int64`，数据维度是`[N, ndims]`。
- `values`: 一个一维的张量，数据类型是任意的，数据维度是`[N]`。
- `dense_shape`: 一个一维的张量，数据类型是`int64`，数据维度是`[ndims]`。

输出参数：

- 一个稀疏张量`SparseTensor`。

`tf.SparseTensor.indices`

解释：这个函数的作用是取出密集矩阵中非零值得索引。

使用例子：

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices=[[4, 1], [1, 2]], values=[1, 2], shape=[3, 4])
b = a.indices
sess = tf.Session()
print sess.run(a)
print sess.run(b)
sess.close()
```

输出参数：

- 一个二维的张量，数据类型是`int64`，数据维度是`[N, ndims]`。其中，`N`表示在稀疏张量中非零值的个数，`ndims`表示稀疏张量的秩。

`tf.SparseTensor.values`

解释：这个函数的作用是取出密集矩阵中非零值。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices=[[4, 1], [1, 2]], values=[1, 2], shape=[3, 4])
b = a.values
sess = tf.Session()
print sess.run(a)
print sess.run(b)
sess.close()
```

输出参数：

- 一个一维的张量，数据类型是任意的。

`tf.SparseTensor.dtype`

解释：这个函数的作用是返回张量中元素的类型。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices=[[4, 1], [1, 2]], values=tf.constant([1, 2]), shape=[3, 4])
b = a.dtype
sess = tf.Session()
print b
sess.close()
```

输出参数：

- 返回张量中元素的类型。

`tf.SparseTensor.shape`

解释：这个函数的作用是返回稀疏张量的维度。

使用例子：

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

import tensorflow as tf
import numpy as np
```

```
a = tf.SparseTensor(indices=[[4, 1], [1, 2]], values=tf.constant([1, 2]), shape=[3, 4])
b = a.shape
sess = tf.Session()
print sess.run(b)
sess.close()
```

输出参数：

- 返回稀疏张量的维度。

`tf.SparseTensor.graph`

解释：这个函数的作用是返回包含该稀疏张量的图。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices=[[4, 1], [1, 2]], values=tf.constant([1, 2]), shape=[3, 4])
b = a.graph
sess = tf.Session()
print b
sess.close()
```

输出参数：

- 返回包含该稀疏张量的图。

`class tf.SparseTensorValue`

解释：这个函数的作用是查看设置稀疏张量的值。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensorValue(indices=[[4, 1], [1, 2]], values=tf.constant([1, 2]), shape=[3, 4])
sess = tf.Session()
print a
print a[0]
print a[1]
print a[2]
sess.close()
```

`tf.SparseTensorValue.indices`

解释：这个函数的作用是返回稀疏张量中值的存在位置。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensorValue(indices=[[4, 1], [1, 2]], values=tf.constant([1, 2]), shape=[3, 4])
sess = tf.Session()
print a.indices
sess.close()
```

输出参数：

- 返回稀疏张量中值的存在位置。

`tf.SparseTensorValue.shape`

解释：这个函数的作用是返回稀疏张量的维度。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensorValue(values=tf.constant([1, 2]), indices=[[4, 1], [1, 2]], shape=[3, 4])
sess = tf.Session()
print a.shape
sess.close()
```

输出参数：

- 返回稀疏张量的维度。

`tf.SparseTensorValue.shape`

解释：这个函数的作用是返回稀疏张量中的元素。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensorValue(values=tf.constant([1, 2]), indices=[[4, 1], [1, 2]], shape=[3, 4])
sess = tf.Session()
print sess.run(a.values)  # 这是一个张量，所以用sess.run()
sess.close()
```

输出参数：

- 返回稀疏张量中的元素。

稀疏张量与密集张量的转换

TensorFlow提供了稀疏张量与密集张量之间的转换操作。

`tf.sparse_to_dense(sparse_indices, output_shape, sparse_values, default_value, name=None)`

解释：这个函数的作用是将一个稀疏表示转换成一个密集张量。具体将稀疏张量`sparse`转换成密集张量`dense`如下：

```
# If sparse_indices is scalar
dense[i] = (i == sparse_indices ? sparse_values : default_value)

# If sparse_indices is a vector, then for each i
dense[sparse_indices[i]] = sparse_values[i]

# If sparse_indices is an n by d matrix, then for each i in [0, n)
dense[sparse_indices[i][0], ..., sparse_indices[i][d-1]] = sparse_values[i]
```

默认情况下，`dense`中的填充值`default_value`都是`0`，除非该值被设置成一个标量。

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.sparse_to_dense(sparse_indices = [[1,2],[2,1]], output_shape = [3,3],
    sparse_values = [2,3], default_value = 1)
sess = tf.Session()
print sess.run(a)
sess.close()
```

输入参数：

- sparse_indices: 一个Tensor，数据类型必须是int32或者int64。数据维度0维，一维或者二维都可以，或者更加高纬度的sparse_indices[i]。
- output_shape: 一个Tensor，数据类型必须和sparse_indices相同。数据维度是一维，表示输出密集张量的维度。
- sparse_values: 一个Tensor，数据维度是一维，其中的每一个元素对应sparse_indices中坐标的值。
- default_value: 一个Tensor，数据类型必须和sparse_values相同，数据维度是一个标量。设置稀疏索引不指定的值。
- 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sparse_values相同。密集张量的数据维度是output_shape。

---

tf.sparse_tensor_to_dense(sp_input, default_value, name=None)

解释：这个函数的作用是将一个稀疏张量SparseTensor转换成一个密集张量。

这个操作是一个便利的将稀疏张量转换成密集张量的方法。

比如，sp_input的数据维度是[3, 5]，非空值为：

```
[0, 1]: a
[0, 3]: b
[2, 0]: c
```

default_value值为x，那么输出的密集张量的维度是[3, 5]，具体的展示形式如下：

```
[[x a x b x]
 [x x x x x]
 [c x x x x]]
```

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[0, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[3, 5])
b = tf.sparse_tensor_to_dense(a, default_value = 11)
sess = tf.Session()
print sess.run(b)
sess.close()
```

输入参数：

- sp_input: 一个SparseTensor。
- default_value: 数据维度是一个标量，设置稀疏索引不指定的值。
- name: （可选）设置返回张量名称的前缀。

输出参数：

- 一个密集张量，数据维度是sp_input.shape，密集张量里面的值为sp_input中指定的值，没有索引的值为default_value值。

异常：

- 类型错误: 如果sp_input不是一个SparseTensor，将报错。

---

tf.sparse_to_indicator(sp_input, vocab_size, name=None)

解释：这个函数的作用是将稀疏张量SparseTensor的坐标转换成密集张量中的布尔坐标。

sp_input中的最后一维被丢弃，并且用sp_input在该位的值来代替，如果sp_input.shape = [D0, D1, D2, ...,

`Dn, K]`，其中`K`是最后一维，那么`output.shape = [D0, D1, D2, ..., Dn, vocab_size]`，其中：
`output[d_0, d_1, ..., d_n, sp_input[d_0, d_1, ..., d_n, k]] = True`
`output`中其余值为`False`。
比如，`sp_input.shape = [2, 3, 4]`，非空值如下：

```
[0, 0, 0]: 0
[0, 1, 0]: 10
[1, 0, 3]: 103
[1, 1, 2]: 112
[1, 1, 3]: 113
[1, 2, 1]: 121
```

并且`vocab_size = 200`，那么输出`output.shape = [2, 3, 200]`，并且`output`中的值都是`False`，除了以下位置：
`(0, 0, 0)`, `(0, 1, 10)`, `(1, 0, 103)`, `(1, 1, 112)`, `(1, 1, 113)`, `(1, 2, 121)`.
使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[0, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[3, 5])
b = tf.sparse_to_indicator(a, 10)
sess = tf.Session()
print sess.run(b)
sess.close()
```

输入参数：

- `sp_input`: 一个`SparseTensor`，数据类型是`int32`或者`int64`。
- `vocab_size`: `sp_Input`最后一维的新的维度，并且`0 <= sp_input.shape > vocab_size`。
- `name`: （可选）设置返回张量名称的前缀。

输出参数：

- 一个经过修改的密集布尔张量。

异常：

- `类型错误`: 如果`sp_input`不是一个`SparseTensor`，将报错。

稀疏张量的操作
TensorFlow提供了一些对于稀疏张量的操作函数。

`tf.sparse_concat(concat_dim, sp_inputs, name=None)`
解释：这个函数的作用是将一系列的`SparseTensor`，按照指定的维度进行合并。
具体合并思路是，先将稀疏张量看成是一个密集张量，然后按照指定的维度进行张量合并，最后将合并成的密集张量看成是一个稀疏张量。
输入的数据中，`SparseTensor`的数据维度必须是相同的，并且`indices`，`values`和`shapes`的长度必须相同。
输出数据的维度将由输入数据的维度决定，除了需要合并的那一维度，这一维度是所有数据该维度的相加总和。
输出张量中的元素将会被重新保存在稀疏张量中，并且按照原来的顺序进行排序。
这个操作的时间复杂度是`O(M log M)`，其中，`M`是输入数据中所有非空元素的个数总和。
比如，当`concat_dim = 1`时：

```
sp_inputs[0]: shape = [2, 3]
[0, 2]: "a"
[1, 0]: "b"
[1, 1]: "c"

sp_inputs[1]: shape = [2, 4]
```

```
[0, 1]: "d"
[0, 2]: "e"
```

那么输出数据为：

```
shape = [2, 7]
[0, 2]: "a"
[0, 4]: "d"
[0, 5]: "e"
[1, 0]: "b"
[1, 1]: "c"
```

用图形表示，如下：

```
[    a] concat [  d e  ] = [    a   d e  ]
[b c  ]        [        ]   [b c         ]
```

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[0, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[3, 5])
aa = tf.SparseTensor(indices = [[1, 1], [1, 3], [2, 1]], values=[11,12,13], shape=[3, 5])
b = tf.sparse_concat(0, [a, aa])
sess = tf.Session()
print sess.run(b)
print sess.run(tf.sparse_tensor_to_dense(b))
sess.close()
```

输入参数：

- `concat_dim`: 需要合并的维度。
- `sp_inputs`: 一个需要合并的`SparseTensor`列表。
- `name`: （可选）设置返回张量名称的前缀。

输出参数：

- 一个经过合并的`SparseTensor`。

异常：

- `类型错误`: 如果`sp_inputs`不是一个`SparseTensor`列表。

`tf.sparse_reorder(sp_input, name=None)`

解释：这个函数的作用是将`SparseTensor`中的元素进行重新排列，按照索引从小到大进行排序。

重排列不会影响`SparseTensor`的维度。

比如，如果`sp_input`的维度是`[4, 5]`，`indices` / `values`如下：

```
[0, 3]: b
[0, 1]: a
[3, 1]: d
[2, 0]: c
```

那么输出的`SparseTensor`的维度还是`[4, 5]`，`indices` / `values`如下：

```
[0, 1]: a
[0, 3]: b
[2, 0]: c
[3, 1]: d
```

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[2, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[3, 5])
b = tf.sparse_reorder(a)
```

```
sess = tf.Session()
print sess.run(b)
sess.close()
```
输入参数：

- sp_input: 一个SparseTensor。
- name: （可选）设置返回张量名称的前缀。

输出参数：

- 一个SparseTensor，数据维度和数据类型都不变，只有其中的值进行了有序的排序。

异常：

- 类型错误: 如果sp_input不是一个SparseTensor。

---

tf.sparse_retain(sp_input, to_retain, name=None)

解释：这个函数的作用是保留SparseTensor中指定的非空元素。

比如，如果sp_input的数据维度是[4, 5]，并且拥有4个非空值如下：

```
[0, 1]: a
[0, 3]: b
[2, 0]: c
[3, 1]: d
```

而且to_retain = [True, False, False, True]，那么最后输出数据SparseTensor的数据维度是[4, 5]，并且保留两个非空值如下：

```
[0, 1]: a
[3, 1]: d
```

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[2, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[3, 5])
b = tf.sparse_retain(a, [False, False, True])
sess = tf.Session()
print sess.run(b)
sess.close()
```

输入参数：

- sp_input: 一个SparseTensor，包含N个非空元素。
- to_retain: 一个布尔类型的向量，向量长度是N，并且其中包含M个True值。

输出参数：

- 一个SparseTensor，数据维度和输入数据相同，其中包含M个非空值，该值的位置根据True的位置来决定。

异常：

- 类型错误: 如果sp_input不是一个SparseTensor。

---

tf.sparse_fill_empty_rows(sp_input, default_value, name=None)

解释：这个函数的作用是将二维的SparseTensor中，将空的行中填充指定元素的值。

如果一行中不存在元素，那么就将改行的坐标[row, 0]填上default_value。

比如，我们假设sp_input的数据维度是[5, 6]，并且非空值如下：

```
[0, 1]: a
[0, 3]: b
[2, 0]: c
[3, 1]: d
```

因为在稀疏张量中，第一行和第四行中不存在值，那么我们需要在`[1, 0]`和`[4, 0]`坐标填上`default_value`，如下：

```
[0, 1]: a
[0, 3]: b
[1, 0]: default_value
[2, 0]: c
[3, 1]: d
[4, 0]: default_value
```

请注意，输入可能有空列在最后，但对这个操作没有任何影响。

输出的`SparseTensor`将是一个按照从小到大的顺序进行排序，并且输出数据和输入数据拥有相同的数据维度。

这个操作还会返回一个布尔向量，其中的布尔值，如果是`True`值，那么表示该行添加了一个`default_value`，计算公式如下：

```
empty_row_indicator[i] = True iff row i was an empty row.
```

使用例子：

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

import tensorflow as tf
import numpy as np

a = tf.SparseTensor(indices = [[2, 1], [0, 3], [2, 0]], values=[1,2,3], shape=[6, 5])
b, bb = tf.sparse_fill_empty_rows(a, 10)
sess = tf.Session()
print sess.run(b)
print '----'
print sess.run(bb)
sess.close()
```

输入参数：

- `sp_input`: 一个`SparseTensor`，数据维度是`[N, M]`。
- `default_value`: 需要向空行填充的值，数据类型和`sp_input`相同。
- `name`: （可选）设置返回张量名称的前缀。

输出参数：

- `sp_ordered_output`: 一个`SparseTensor`，数据维度是`[N, M]`，并且其中所有空行填充了`default_value`。
- `empty_row_indicator`: 一个布尔类型的向量，数据长度是`N`，如果该行填充了`default_value`，那么该位置的布尔值为``。

异常：

- `类型错误`: 如果`sp_input`不是一个`SparseTensor`。