利用 TensorFlow 实现排序和搜索算法

当我们提到 TensorFlow 的时候，我们仅仅只会关注它是一个很好的神经网络和深度学习的库。但是，其实 TensorFlow 具有 tf.cond 和 tf.while_loop 函数，前者可以处理判断语句，后者可以处理循环语句，所以它也具有一般编程语言相同的表达式。简单的说，我们可以用 C 语言或者 Python 语言实现的排序和搜索算法都可以在 TensorFlow 图中实现。

在本文中，我们就是要介绍 TensorFlow 的另一面，它的一般编程语言表达方式。我们利用 TensorFlow 图实现了一些简单算法，诸如 **FizzBuzz** 问题，线性搜索，冒泡排序 等等。

1. API 解释

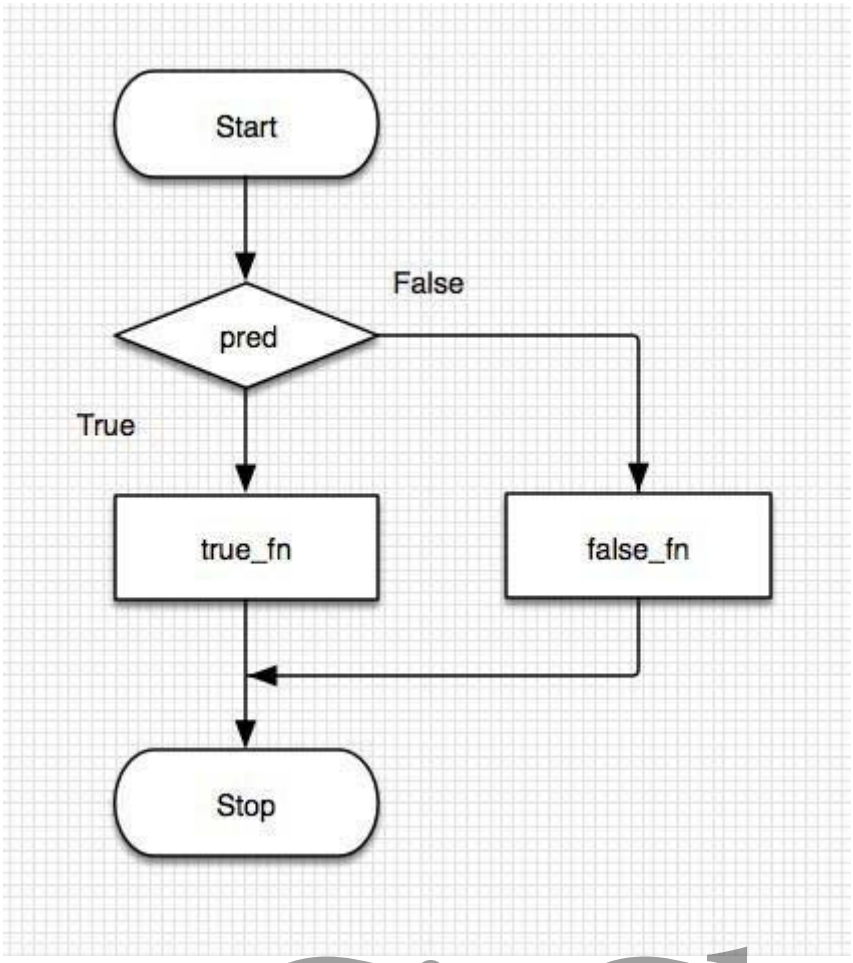
1.1 类似判断语句的 API：tf.cond()


```
cond(
    pred,
    true_fn=None,
    false_fn=None,
    strict=False,
    name=None,
    fn1=None,
    fn2=None
)
```

`tf.cond(...)` 是一个等效于 **if** 语句的节点。根据其中的参数 `pred` 返回的布尔值来判断返回什么值，比如当参数 `pred` 为 `true` 值时，节点返回参数 `true_fn` 的值，当参数 `pred` 为 `false` 时，节点返回参数 `false_fn` 的值。但是，其中的参数 `true_fn` 和参数 `false_fn` 都是需要是 lambda 或者函数。比如：

```
z = tf.multiply(a, b)
result = tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))
```

当 x < y 是 true 时，节点就会去执行 tf.add 操作。当 x < y 是 false 时，节点就会去执行 tf.square 操作。



接下来，我们来看一个完整的例子，如下：

```
x = tf.constant(2)
y = tf.constant(5)
def f1():
    return tf.multiply(x, 17)

def f2():
    return tf.add(y, 23)

r = tf.cond(tf.less(x, y), f1, f2)
with tf.Session() as sess:
    print(sess.run(r))
```

请注意：*API* 中的某些参数被忽略了，因为它们将在以后的版本中被删除。

- tf.cond—TensorFlow API r1.3

1.2 类似判断语句的 API： tf.while_loop()

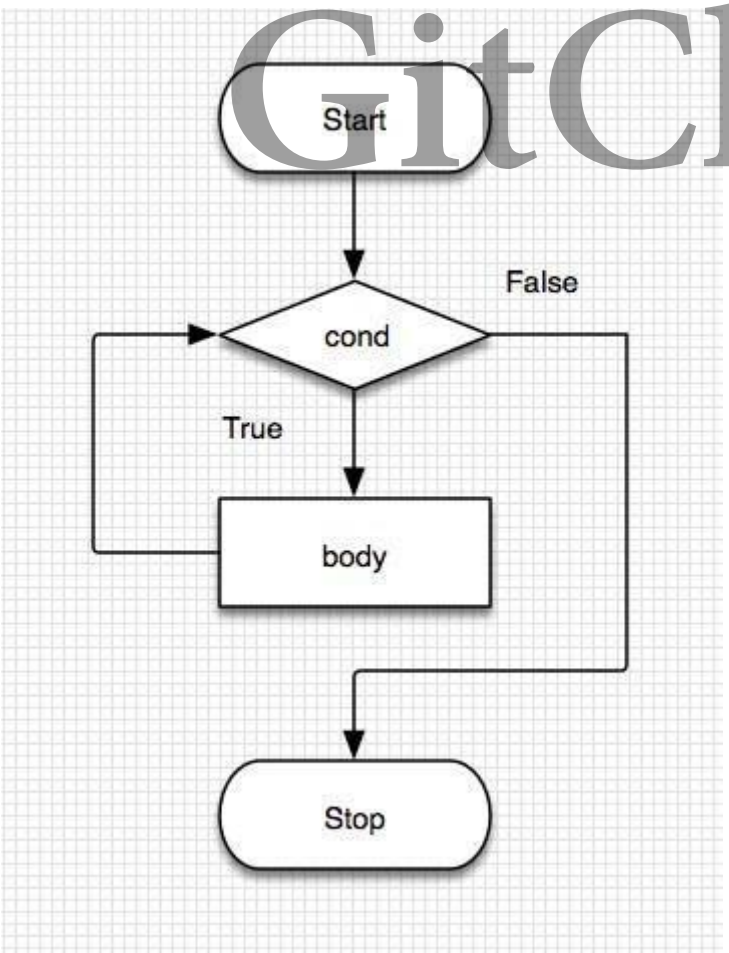
```
while_loop(
    cond, # Condition
    body, # Process to be executed when cond is True
```

```
    loop_vars, # Argument to body
    shape_invariants=None,
    parallel_iterations=10,
    back_prop=True,
    swap_memory=False,
    name=None
)
```

tf.while_loop(...) 是一个等效于 **while** 语句的节点。根据其中的参数 cond 的布尔值来判断是否将循环继续，比如当参数 pred 为 true 值时，节点去执行 body 中的语句，当参数 pred 为 false 时，那么退出这个函数。比如：

```
i = tf.constant(0)
c = lambda i: tf.less(i, 10)
b = lambda i: tf.add(i, 1)
r = tf.while_loop(c, b, [i])
```

当 i < 10 时，cond 返回的值是 true，所以节点会去执行 body 中的语句。当 i == 10 时，cond 返回的值是 false，那么节点就会退出。这种执行方式和一般语言中的 while 非常像。



我们也可以将循环式表达成如下：

```
while(cond(loop_vars))
{
     loop_vars = body(loop_vars);
}
```

接下来，我们来看一个完整的例子，如下：

```
import tensorflow as tf
import numpy as np

def body(x):
    a = tf.random_uniform(shape=[2, 2], dtype=tf.int32, maxval=100)
    b = tf.constant(np.array([[1, 2], [3, 4]]), dtype=tf.int32)
    c = a + b
    return tf.nn.relu(x + c)

def condition(x):
    return tf.reduce_sum(x) < 100

x = tf.Variable(tf.constant(0, shape=[2, 2]))

with tf.Session():
    tf.initialize_all_variables().run()
    result = tf.while_loop(condition, body, [x])
    print(result.eval())
```


- tf.while_loop — TensorFlow API r1.3

2. 在 TensorFlow 中实现算法

2.1 Fizz Buzz 问题

请依次打印从1至100的整数，在该数能被3整除的时候，打印”Fizz”，能被5整除的时候打印”Buzz”，如果既能被3又能被5整除的时候，打印”FizzBuzz”。

```
import tensorflow as tf

class FizzBuzz():
    def __init__(self, length=30):
        self.length = length  # 程序需要执行的序列长度
        self.array = tf.Variable([str(i) for i in range(1, length+1)],
dtype=tf.string, trainable=False)  # 最后程序返回的结果
        self.graph = tf.while_loop(self.cond, self.body, [1,
self.array],)   # 对每一个值进行循环判断
```

```
    def run(self):
        with tf.Session() as sess:
            tf.global_variables_initializer().run()
            return sess.run(self.graph)

    def cond(self, i, _):
        return (tf.less(i, self.length+1)) # 判断是否是最后一个值

    def body(self, i, _):
        flow = tf.cond(
            tf.equal(tf.mod(i, 15), 0),  # 如果值能被 15 整除，那么就把该位置赋值为 FizzBuzz
                lambda: tf.assign(self.array[i - 1], 'FizzBuzz'),

                lambda: tf.cond(tf.equal(tf.mod(i, 3), 0), # 如果值能被 3 整除，那么就把该位置赋值为 Fizz
                                lambda: tf.assign(self.array[i - 1], 'Fizz'),
                                lambda: tf.cond(tf.equal(tf.mod(i, 5), 0),   # 如果值能被 5 整除，那么就把该位置赋值为 Buzz
                                                lambda: tf.assign(self.array[i - 1], 'Buzz'),
                                                lambda: self.array  # 最后返回的结果
                                )
                )
        )
        return (tf.add(i, 1), flow)

if __name__ == '__main__':
    fizzbuzz = FizzBuzz(length=50)
    ix, array = fizzbuz.run()
    print(array)
```


输出结果：

```
['1' '2' 'Fizz' '4' 'Buzz' 'Fizz' '7' '8' 'Fizz' 'Buzz' '11' 'Fizz'
'13'
 '14' 'FizzBuzz' '16' '17' 'Fizz' '19' 'Buzz' 'Fizz' '22' '23' 'Fizz'
 'Buzz' '26' 'Fizz' '28' '29' 'FizzBuzz' '31' '32' 'Fizz' '34' 'Buzz'
 'Fizz' '37' '38' 'Fizz' 'Buzz' '41' 'Fizz' '43' '44' 'FizzBuzz' '46'
'47'
 'Fizz' '49' 'Buzz']
```

2.2 线性搜索

给定一个序列和一个目标值，从这个序列中找到这个目标值的位置。

```
import numpy as np
import tensorflow as tf

class LinearSearch():
    def __init__(self, array, x):
        self.x = tf.constant(x)
        self.array = tf.constant(array)
        self.length = len(array)
        self.graph = tf.while_loop(self.cond, self.body, [0, self.x,
False])

    def run(self):
        with tf.Session() as sess:
            tf.global_variables_initializer().run()
            return sess.run(self.graph)

    def cond(self, i, _, is_found):
        return tf.logical_and(tf.less(i, self.length),
tf.logical_not(is_found))

    def body(self, i, _, is_found):
        return tf.cond(tf.equal(self.array[i], self.x),
                       lambda: (i, self.array[i], True),
                       lambda: (tf.add(i, 1), -1, False))

if __name__ == '__main__':
    array, x = [1, 22, 33, 1, 7, 3, 8], 3
    search = LinearSearch(array, x)
    ix, xx, is_found = search.run()
    print('Array :', array)
    print('Number to search :', x)
    if is_found:
        print('{} is at index {}.'.format(xx, ix))
    else:
        print('Not found.')
```


输出结果：

```
Array : [1, 22, 33, 1, 7, 3, 8]
Number to search : 3
3 is at index 5.
```

2.3 冒泡排序

给定一个数组，利用冒泡排序进行排序，最后输出排好序的数组。冒泡排序算法可以查看这个文档。

```
import numpy as np
import tensorflow as tf

class BubbleSort():
    def __init__(self, array):
        self.i = tf.constant(0)
        self.j = tf.constant(len(array)-1)
        self.array = tf.Variable(array, trainable=False)
        self.length = len(array)

        cond = lambda i, j, _: tf.less(i-1, self.length-1)
        self.graph = tf.while_loop(cond, self.outer_loop, loop_vars=
[self.i, self.j, self.array])

    def run(self):
        with tf.Session() as sess:
            tf.global_variables_initializer().run()
            return sess.run(self.graph)

    def outer_loop(self, i, j, _):
        cond = lambda i, j, _: tf.greater(j, i)
        loop = tf.while_loop(cond, self.inner_loop, loop_vars=[i,
self.length-1, self.array])
        return tf.add(i, 1), loop[1], loop[2]

    def inner_loop(self, i, j, _):
        body = tf.cond(tf.greater(self.array[j-1], self.array[j]),
                       lambda: tf.scatter_nd_update(self.array, [[j-1],
[j]], [self.array[j],self.array[j-1]]),
                       lambda: self.array)
        return i, tf.subtract(j, 1), body

if __name__ == '__main__':
    x = np.array([1.,7.,3.,8.])
    _, _, sorted_array = BubbleSort(x).run()
    print(x)
    print(sorted_array)
```

输出结果：

```
[ 1.  7.  3.  8.]
[ 1.  3.  7.  8.]
```

TensorFlow 还有更多的实现算法，你可以查看这个 Github。