Tensorflow Conditionals and While Loops

Tensorflow conditionals and while loops

Recntly I found myself needing to implement more advanced control flow in some models I have been hacking on in my free time. In past I never really needed any graph conditionals or loops or any combinations thereof, so I had to dive into documentation and read up on them.

This blog post covers `tf.cond` and `tf.while_loop` control flow operations and was written to document and share my experience learning about them. Both of the operations seem intuitive on the first look, but I got bitten by them so I wanted to document their usage on practical examples so I have something as a reference I can return to in the future should I need to.

tf.cond: simple lambdas

This is basically a slightly modified code from the official documentation. Here we have two constant tensors `t1` and `t2` and we execute either `f1()` or `f2()` based on the result of `tf.less()` operation:

```
t1 = tf.constant(1)
t2 = tf.constant(2)

def f1(): return t1+t2
def f2(): return t1-t2

res = tf.cond(tf.less(t1, t2), f1, f2)

with tf.Session() as sess:
    print(sess.run(res))
```

As expected the printed number is `3` since `1<2` and thus the `f1()` gets executed. It's worth noting that both lambdas here are single line functions, neither of which accepts any parameters.

tf.cond: parametrised lambdas

Now if you want to pass some values into the `f1()` and/or `f2()` lambdas defined earlier, you need to write a simple closure that encloses the passed in parameter i.e. something like this:

```
def f1(p):
    def res(): return t1+t2+p
    return res

def f2(p):
    def res(): return t1-t2+p
    return res

t1 = tf.constant(1)
t2 = tf.constant(2)
p1 = tf.constant(3)
p2 = tf.constant(4)

res = tf.cond(tf.less(t1, t2), f1(p1), f2(p2))

with tf.Session() as sess:
    print(sess.run(res))
```

As expected the above code will print `6` since `1<2` and thus the `f1(3)` gets executed: `1+2+3`.

tf.while_loop: basics

Let's start again with a simple, slightly modified example of `tf.while_loop` usage borrowed from the official documentation. Once again, we will have two constant tensors `t1` and `t2` and we will run a loop that will be incrementing `t1` while it's less than `t2`:

```
def cond(t1, t2):
    return tf.less(t1, t2)

def body(t1, t2):
    return [tf.add(t1, 1), t2]

t1 = tf.constant(1)
t2 = tf.constant(5)

res = tf.while_loop(cond, body, [t1, t2])

with tf.Session() as sess:
    print(sess.run(res))
```

Note that both `cond` and `body` function must accept as many arguments as there are `loop_vars`; in this case our `loop_vars` are the constant tensors `t1` and `t2`. `tf.while_loop` then returns the result as a tensor of the same shape os `loop_var` (let's forget about `shape_invariants` for now):

```
[5, 5]
```

This result makes perfect sense: we keep incrementing the original value (`1`) until it's less than `5`. Once it reaches `5` the `tf.while_loop` stops and the last value returned by `body` is returned as a result.

tf.while_loop: fixed number of iterations

Now if we wanted a fixed number of iterations we would modify the code such as follows:

```
def cond(t1, t2, i, iters):
    return tf.less(i, iters)

def body(t1, t2, i, iters):
    return [tf.add(t1, 1), t2, tf.add(i, 1), iters]

t1 = tf.constant(1)
t2 = tf.constant(5)
iters = tf.constant(3)

res = tf.while_loop(cond, body, [t1, t2, 0, iters])

with tf.Session() as sess:
    print(sess.run(res))
```

There is a couple of things to notice in this code: * third item in `loop_vars` is `0`; this is the value we will be incrementing * loop incrementation happens in `body` function: `tf.add(i, 1)` * once again the returned value (in this particular example) has as many elements as there are in `loop_vars`

This code prints the following result:

```
[4, 5, 3, 3]
```

First we have the `t1` value incremented `iters` times (`3`); we don't modify `t2` in this code; the third parameter is the final increment increment value we started with `0` and finished once `iters` (`3`) of iterations was reached (this is controlled by `cond` function).

tf.while_loop: conditional break

With all the knowledge of `tf.cond` and `tf.while_loop` we are now well equipped to do conditional loops i.e. writing loops whose behaviour changes based on some condition, sort of like `break` clauses in imperative programming. For brevity we will stick the code into a dedicated function called `cond_loop` which will return a tensor operation we will run in the session.

This is what our code is going to do: * we will be looping fixed number of loops set by `iters` * in each loop we will increment our familiar constant tensors `t1` and `t2` * in the final loop, we will swap the tensors instead of incrementing them

You can see this in the code below:

```
def cond_loop(t1, t2, iters):
    def cond(t1, t2, i):
        return tf.less(i, iters)

    def body(t1, t2, i):
        def increment(t1, t2):
            def f1(): return tf.add(t1, 1), tf.add(t2, 1)
            return f1

        def swap(t1, t2):
            def f2(): return t2, t1
            return f2

        t1, t2 = tf.cond(tf.less(i+1, iters),
                         increment(t1, t2),
                         swap(t1, t2))

        return [t1, t2, tf.add(i, 1)]

    return tf.while_loop(cond, body, [t1, t2, 0])

t1 = tf.constant(1)
t2 = tf.constant(5)
iters = tf.constant(3)

with tf.Session() as sess:
    loop = cond_loop(t1, t2, iters)
    print(sess.run(loop))
```

The main difference between this code and the code we discussed previously is the presence of `tf.cond` in the `body` function that gets executed in each `tf.while_loop` "iteration" (the double quotes here are deliberate: `while_loop` calls `cond` and `body` exactly once). This `tf.cond` causes `body` to execute `swap` lambda instead of `increment` at the end of the loop. The return result confirms this:

```
[7, 3, 3]
```

We set out to run `3` iterations which will increment the constants `1` and `5` in each iteration except for the last one when we swap both values and return them along with the iteration counter. (`1+1+1`, `5+1+1` <-> `7`, `3`).

Summary

`TensorFlow` provides powerful data flow control structures. We have hardly scratched on the surface here. You should definitely check out the official documentation and read about things like `shape_invariants`, `parallel_iterations` or `swap_memory` memory parameters that can speed up your code and save you some headaches. Happy hacking!

tensorflow

Tweet

Comments