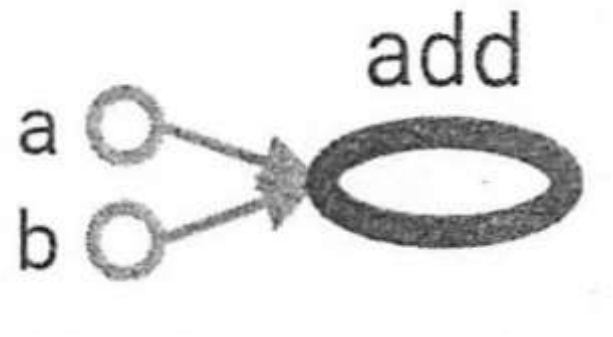
什么是张量

https://www.youtube.com/watch?v=f5liqUk0ZTw 英文解说
https://blog.csdn.net/wtq1993/article/details/51714121 中文解释

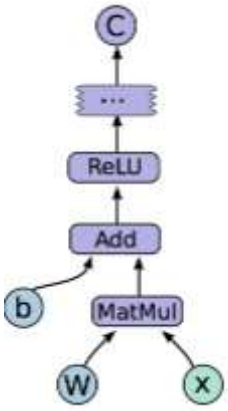
TensorFlow的名字由来

TensorFlow的名字可以拆开为tensor和flow来理解。tensor是张量，flow是流。因此，TensorFlow就是张量的流动。简单来讲，**tensorFlow的运行机制就是通过构建一个计算图，然后再根据这张图进行计算。**计算图包含很多节点，它们叫做**算子(operation)**。算子会接受张量作为输入，并输出张量给另一个算子。因此在运行计算图的过程中，张量会被一直操作并传递，直到计算图输出。

算子，计算图，张量的形象表示



在图1，a,b两个算子输出两个向量作为**add**算子的输入，add会对其输入进行加法操作，但没有输出。一个**计算图**就是由很多个算子相互联系而成的。比如图2，它所完成的事情显而易见。MatMul, Add 和 ReLU是**算子**。b, W, X是**张量**。整个流程是**计算图**。



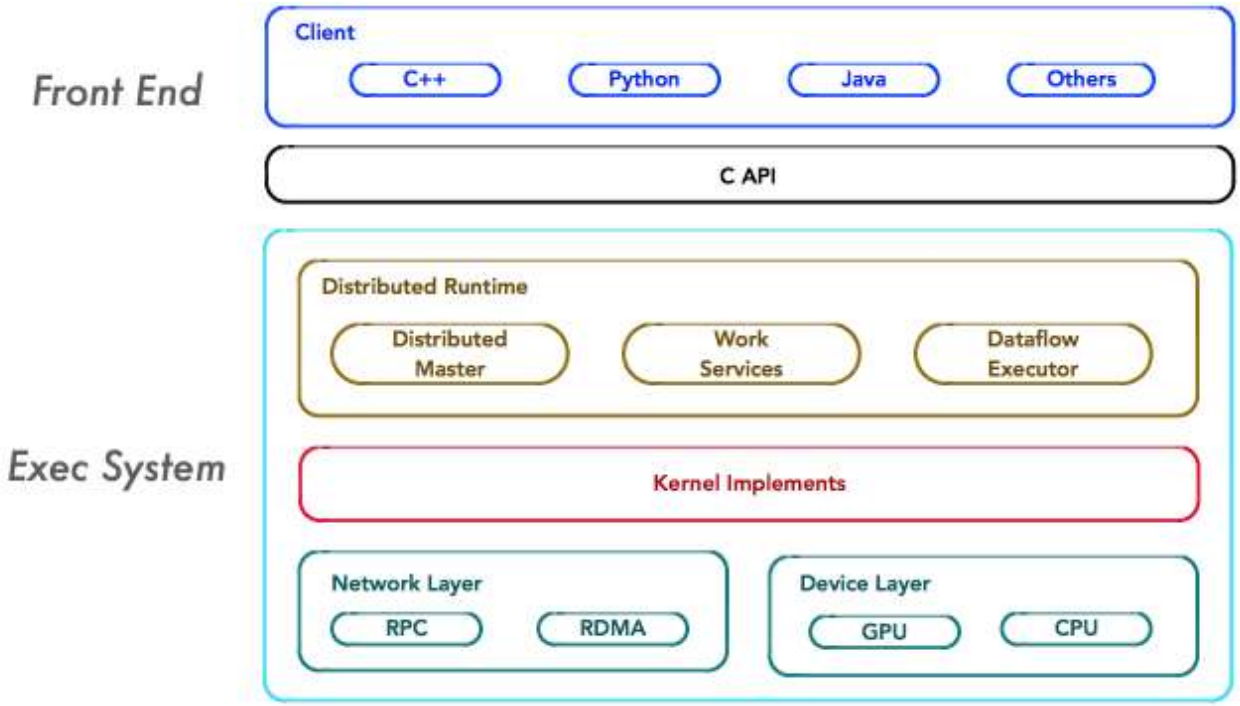
理解不了张量?

张量类似于多维的数组，在物理学有广泛的应用。但是在这里，我们不必过于纠结它的意义。我们只需要知道它在计算图中是输入、输出中流动的信息就好。它既是算子的输入，也是它的输出。

TensorFlow的运行机制

上文说过，TensorFlow的运行机制就是通过构建一个计算图，然后再根据这张图进行计算。所以一段TensorFlow代码一般包括两个部分， 构建计算图，以及运行计算图。 TensorFlow的系统结构以 C API 为界，将整个系统分为「前端」和「后端」两个子系统：

- 前端系统：提供编程模型，负责构造计算图；
- 后端系统：提供运行时环境，负责执行计算图。



然后我们看一段TensorFlow代码，它阐述了一个完整的从前端到后端的执行过程：

```
import tensorflow as tf
import numpy as np

"""
  构造计算图，此时工作由前端完成
"""
a = tf.constant (3)
b = tf.constant (2)
c = a + b

"""
   建立Session
"""
sess = tf.Session ()

"""
  交给后端执行
"""
print (sess.run(c))
```

我们再输出一些信息，来证明 a b和c 都只是张量，而不是计算的结果

```
import tensorflow as tf
import numpy as np

"""
  构造计算图，此时工作由前端完成
```

```
"""
a = tf.constant (3)
b = tf.constant (2)
c = a + b
print ("a is", a)
print ("b is", b)
print("c is", c)
"""
   建立Session
"""
sess = tf.Session ()
print(sess)
"""
  交给后端执行
"""
print (sess.run(c))

""" 输出:
  a is Tensor("Const:0", shape=(), dtype=int32)
  b is Tensor("Const_1:0", shape=(), dtype=int32)
  c is Tensor("add:0", shape=(), dtype=int32)
  <tensorflow.python.client.session.Session object at 0x00000182C14589E8>
  5
"""
```

从代码中我们可以看到，在执行 `c = a + b` 时，TensorFlow并没有开始计算，而只是生成了它的计算图。通过计算图，TensorFlow可以知道， `c` 是怎么得到的。直到执行 `sess.run(c)` ，TensorFlow才根据这个计算图得到 `c` 的具体值。

session的使用

显示调用session

```
"""
  手动生成，显式调用，手动关闭session。
  这种方式如果在后端执行时遇到错误，程序就会在session没关闭的情况下退出，导致内存泄漏。
"""
from __future__ import print_function,division
import tensorflow as tf

a=tf.constant(2.)
b=tf.constant(5.)
c=a*b

sess=tf.Session()
print(sess.run(c))
sess.close()
```

```
"""
  这种使用上下文管理器的方式，会在程序离开该代码段时自动释放资源，防止内存泄漏
```

```
"""
# Using the context manager.
with tf.Session() as sess:
  sess.run(...)
```

尽管用上下文管理器可以防止内存泄漏，我们仍需在调用时显式指定session(`sess.run`)。

隐式调用session

```
"""
  由于调用了sess.as_default()，在with代码段内，默认session已被指定为sess。
  这样当我们要计算某个张量的具体值时，只需像c.eval()这样调用即可，此时的session就默认使用sess。而不用显式地调用sess.run(c)。
  当然，就算要调用sess.run(c)也完全可行，两者的效果是等价的。
"""
import tensorflow as tf

c = tf.constant(3)
sess = tf.Session()

with sess.as_default():
  print(tf.get_default_session() is sess) # True
  print(c.eval()) # 3
```

```
"""
  如下展示了默认session在多个session间切换的方式。
"""
import tensorflow as tf

c = tf.constant(3)
sess = tf.Session()
sess2 = tf.Session()
with sess.as_default(): # sess为默认session
    print(c.eval())
# ...
with sess2.as_default(): #sess2为默认session
    print(c.eval())
```

还有另一种方法：使用 `sess = tf.InteractiveSession()` ，如此可以省去注册sess为默认session的过程。

```
"""
  省去注册sess为默认session的过程，有利于Jupyter NoteBook等交互式环境的体验。
"""
import tensorflow as tf

sess = tf.InteractiveSession()
# print(result.val())
sess.close()
```

graph的调用

在没有指定graph时，TensorFlow会隐式地构建一个graph作为默认graph。通过graph.as_default()可以指定默认的graph。如下代码证明了这一点。

```
"""
   在with代码段外，使用的是TensorFlow自己构建的graph
"""
import tensorflow as tf
import numpy as np

g = tf.Graph()

with g.as_default():
    # Define operations and tensors in `g`.
    c = tf.constant(30.0)
    print(c.graph is g) # True

print(tf.get_default_graph() is g) # False
```

神经网络的概念理解

遇到问题

```
 The TensorFlow library wasn't compiled to use FMA instructions,
but these are available on your machine and could speed up CPU computations.
```

参考

- https://blog.csdn.net/stdcoutzyx/article/details/51645396
- https://blog.csdn.net/xierhacker/article/details/53860379
- https://www.jianshu.com/p/a5574ebcdeab
- 《Tensorflow 实战Google深度学习框架》