

Tensorflow2.0 Tutorial

Jingjing Gong

什么是TensorFlow?

- Google brain开发用来做机器学习以及深度学习研究的工具
- 多平台支持，服务器，个人电脑，移动设备
- 支持多卡，分布式计算
- 灵活，具有通用性，可以用在其它领域
- 当前支持python, java以及c++接口
- 可以自动求导
- 可以方便的利用GPU

还有哪些主流深度学习框架？

- Pytorch (Dynamic Graph) [facebook]
- PaddlePaddle (Static Graph) [百度]
- MxNet (Static Graph). [amazon]
- caffe, Theano, etc. (过时的历史框架)

按照执行方式分类

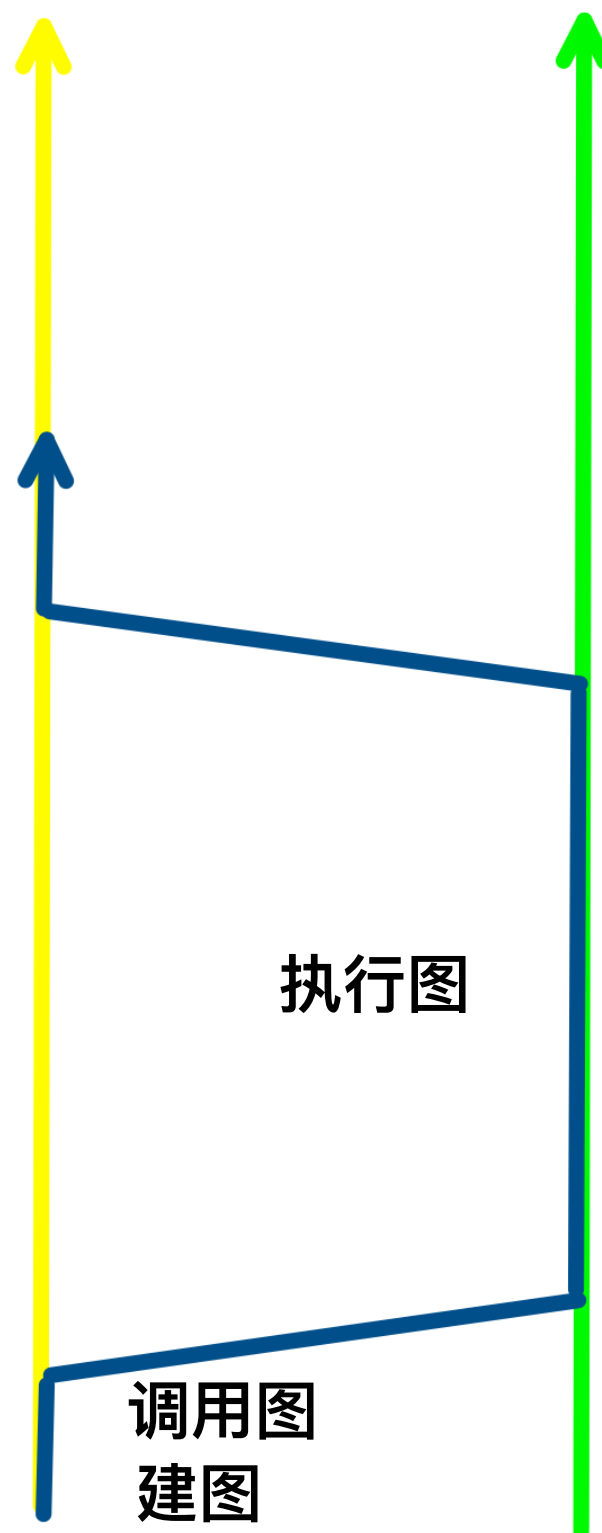
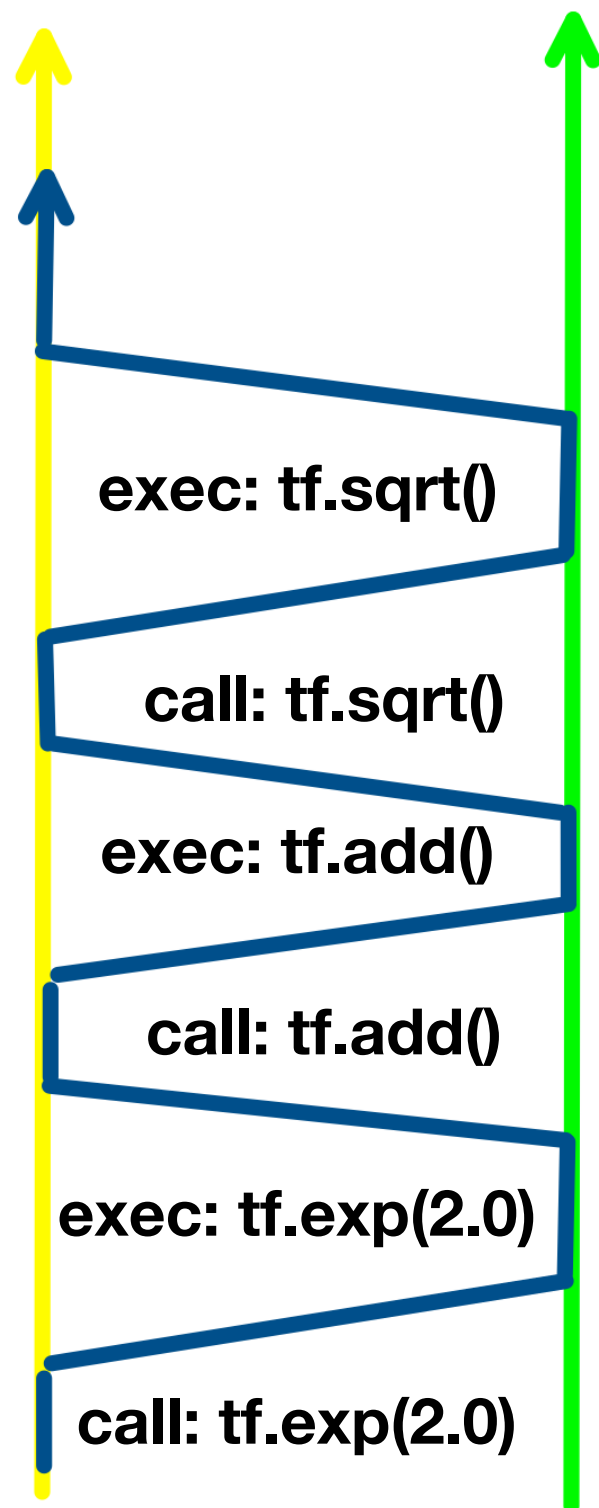
动态图

- Tensorflow (**Eager Mode**)
- Pytorch

静态图

- Tensorflow (Graph Mode)
- MxNet
- caffe
- Theano
- PaddlePaddle

Dynamic Graph vs. Static Graph



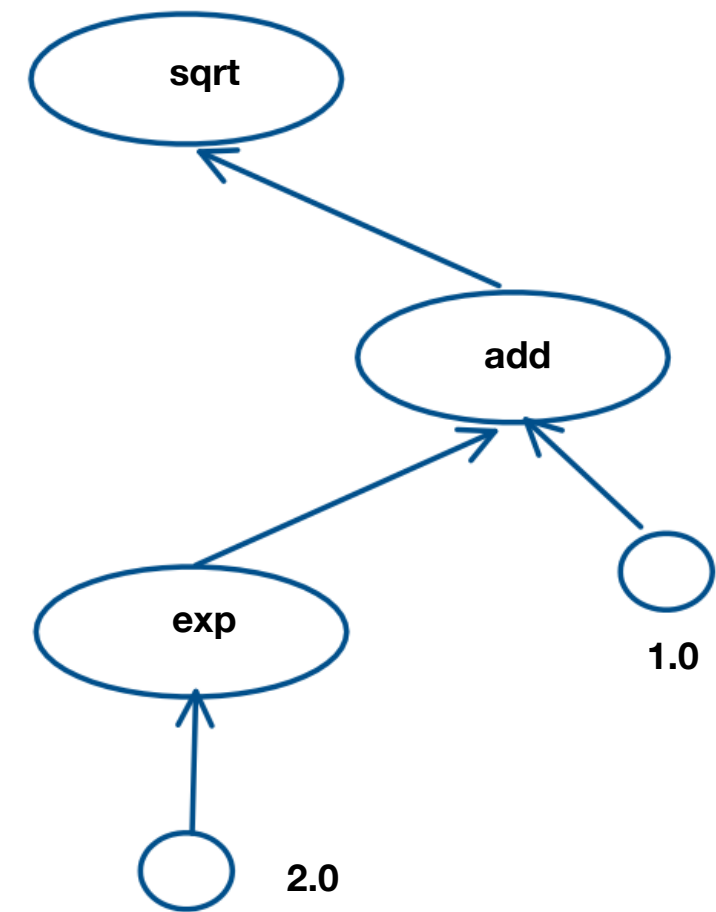
Python Routine



TF Sub-routine



```
aa = tf.exp(2.0)  
bb = tf.add(aa, 1.)  
cc = tf.sqrt(bb)
```



为什么需要这些框架？

这得从学习问题本身说起

学习问题（不严谨）

对于一些观测样本 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, 其中 y 是对应输入 \mathbf{x} 的标签。

我们定义一个函数 $y = f(\mathbf{x}, \mathbf{w})$, 其中 $f(\cdot, \mathbf{w})$ 是一个函数家族, 我们也把它叫做模型, \mathbf{w} 是这个模型的参数。

学习目标就是找到一个 \mathbf{w}^* 使得观测样本尽可能满足 $y = f(\mathbf{x}, \mathbf{w}^*)$ 这个等式（经验风险最小化），我们把它称作优化过程

怎么优化？ 梯度下降

我们可以定义一个连续平滑的函数 \mathcal{L} 用来描述模型预测 $f(\mathbf{x}, \mathbf{w})$ 跟实际标签 y 的距离：

$$\mathcal{L}(y, f(\mathbf{x}, \mathbf{w}))$$

对于所有样本这个误差可以描述为：

$$\mathbf{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, f(\mathbf{x}_i, \mathbf{w}))$$

我们可以使用梯度下降找到 \mathbf{w}^* ，但这要求我们求到： $\frac{\partial \mathbf{L}}{\partial \mathbf{w}}$ ，这样我们可以迭代优化 \mathbf{w} ：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \frac{\partial \mathbf{L}}{\partial \mathbf{w}_t}$$

怎么求 $\frac{\partial \mathbf{L}}{\partial \mathbf{w}}$? 链式求导

$$\frac{\partial \mathbf{L}}{\partial \mathbf{w}} = \frac{1}{n} \sum_{i=1}^n \frac{\partial \mathcal{L}(y_i, f(\mathbf{x}_i, \mathbf{w}))}{\partial \mathbf{w}}$$

从前，对于**每一个模型**我们都需要用通过链式法则用笔求出梯度公式：

$$\frac{\partial \mathcal{L}(y_i, f(\mathbf{x}_i, \mathbf{w}))}{\partial \mathbf{w}}$$

然后写大量代码通过给 \mathbf{w} 加扰动计算数值的梯度跟上面公式求出的梯度进行对比。这个过程会耗费大量精力，然而链式求导却是一个非常机械的过程。

在Tensorflow中，求梯度非常简单

```
#take gradient of a variable
```

```
def func(w):  
    aa = tf.exp(w)  
    bb = tf.add(aa, 1.)  
    cc = tf.sqrt(bb)  
    return cc
```

定义的目标函数，可以任意复杂

```
w = tf.Variable(2.0, dtype=tf.float32)  
with tf.GradientTape() as tape:  
    y = func(w)  
print(tape.gradient(y, w))
```

参数

求导数

```
tf.Tensor(1.2755646, shape=(), dtype=float32)
```

Easy like a pie :)

Tensorflow的一些基础操作

- `tf.Variable()`
- `tf.constant()`
- `tf.zeros`
- `tf.ones`
- `tf.zeros_like`
- `tf.ones_like`
- `tf.random`
- `tf.add`, `tf.subtract`
- `tf.multiply`, `tf.divide`
- `tf.exp`, `tf.sqrt`
- `tf.sin`, `tf.cos`
- 基本的数学运算都有

tf.Variable

- tf.Variable叫做变量对象，可以对应之前讲的参数 w 。在人工神经网络中，通常用来存储连接的权重。对于所有样本可访问的全局量。
- 在Tensorflow中只有Variable 可以被赋值：

```
aa = tf.Variable(2.0)
print(aa)
aa.assign(3.0)
print(aa)
aa.assign_add(2.0)
print(aa)
aa.assign_sub(1.0)
print(aa)
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=2.0>
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=3.0>
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=5.0>
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=4.0>
```

tf.GradientTape

- 在这个context下的操作如果它的至少一个输入被watch了，那么这个操作也会被记录（中间结果会被记住）
- 对于变量（tf.Variable），会被自动记录。
- 配合demo_code:Gradient Tape watch 演示

```
x = tf.ones((2, 2))

with tf.GradientTape() as t:
    t.watch(x)
    y = tf.reduce_sum(x)
    z = tf.multiply(y, y)

# Derivative of z with respect to the original input tensor x
dz_dx = t.gradient(z, x)
```

tf.GradientTape

- 取了一次gradient之后记录的中间结果就会被销毁，如果需要取多次gradient，那么把persistent设置成True。

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as g:
    g.watch(x)
    y = x * x
    z = y * y
dz_dx = g.gradient(z, x)  # 108.0 (4*x^3 at x = 3)
dy_dx = g.gradient(y, x)  # 6.0
```

为什么要记住中间结果？

tf.function

- 对于一个函数使用tf.function 修饰符，相当于把整个函数中的很多operation打包成一个operation。
- 在函数中尽量使用原生tensorflow 的操作，尽管对于某一些python操作也会解析成Graph的一部分。

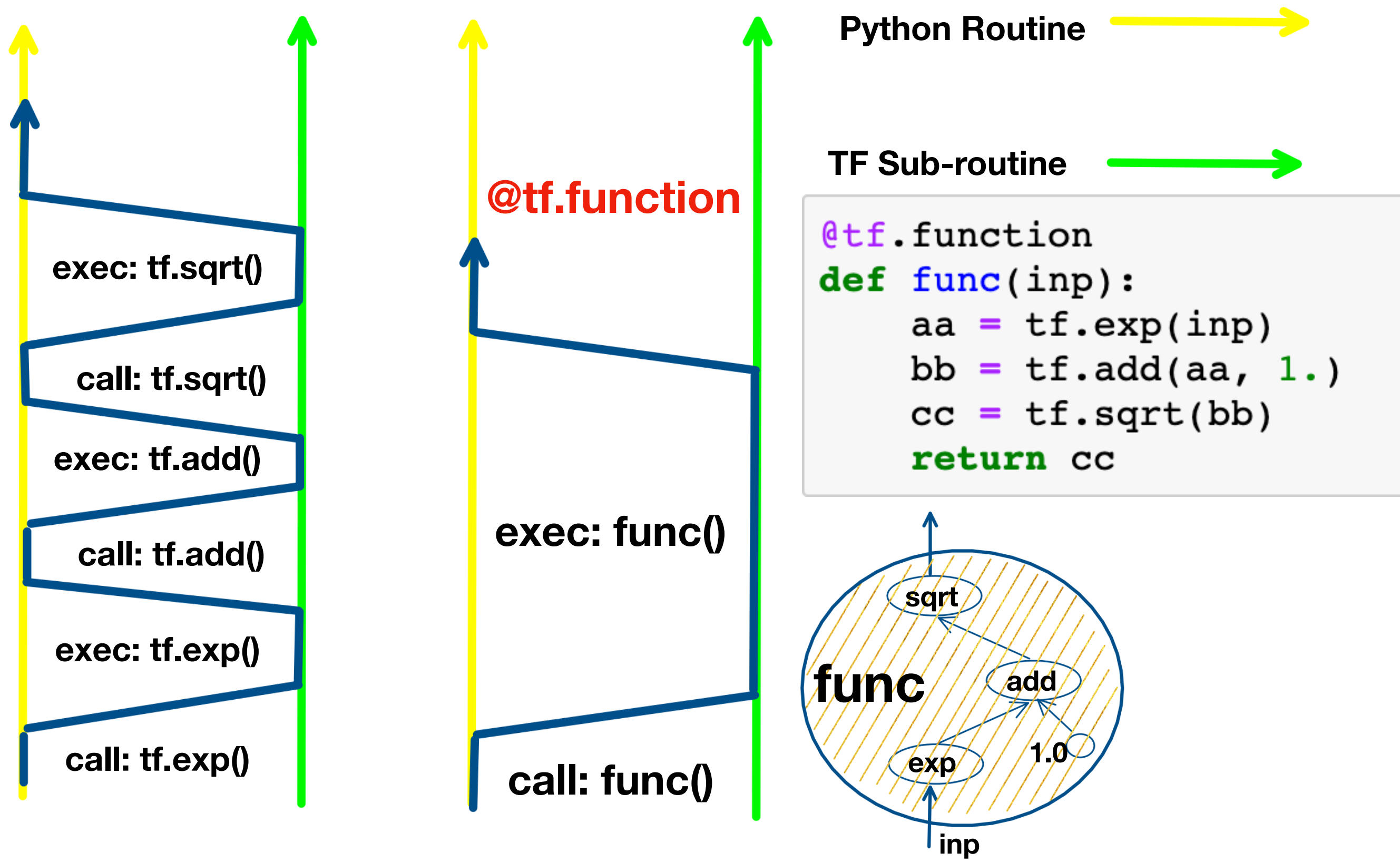
```
@tf.function
def func(inp):
    aa = tf.exp(inp)
    bb = tf.add(aa, 1.)
    cc = tf.sqrt(bb)
    return cc
```


为什么要用tf.function修饰?

- 对于没有使用tf.function的tensorflow计算函数，函数内部执行是在Eager Mode下完成的，每一个内部操作都需要做上下文切换。
- 而使用了tf.function修饰的函数被编译成一个完整的操作，内部操作是在Graph Mode下完成的，整个函数可以看作是一个操作，这样效率会相对比较高。

为什么要用tf.function修饰?

记住这张PPT



tensorflow2.0 线性回归GD

linear_regression-tf2.0.ipynb