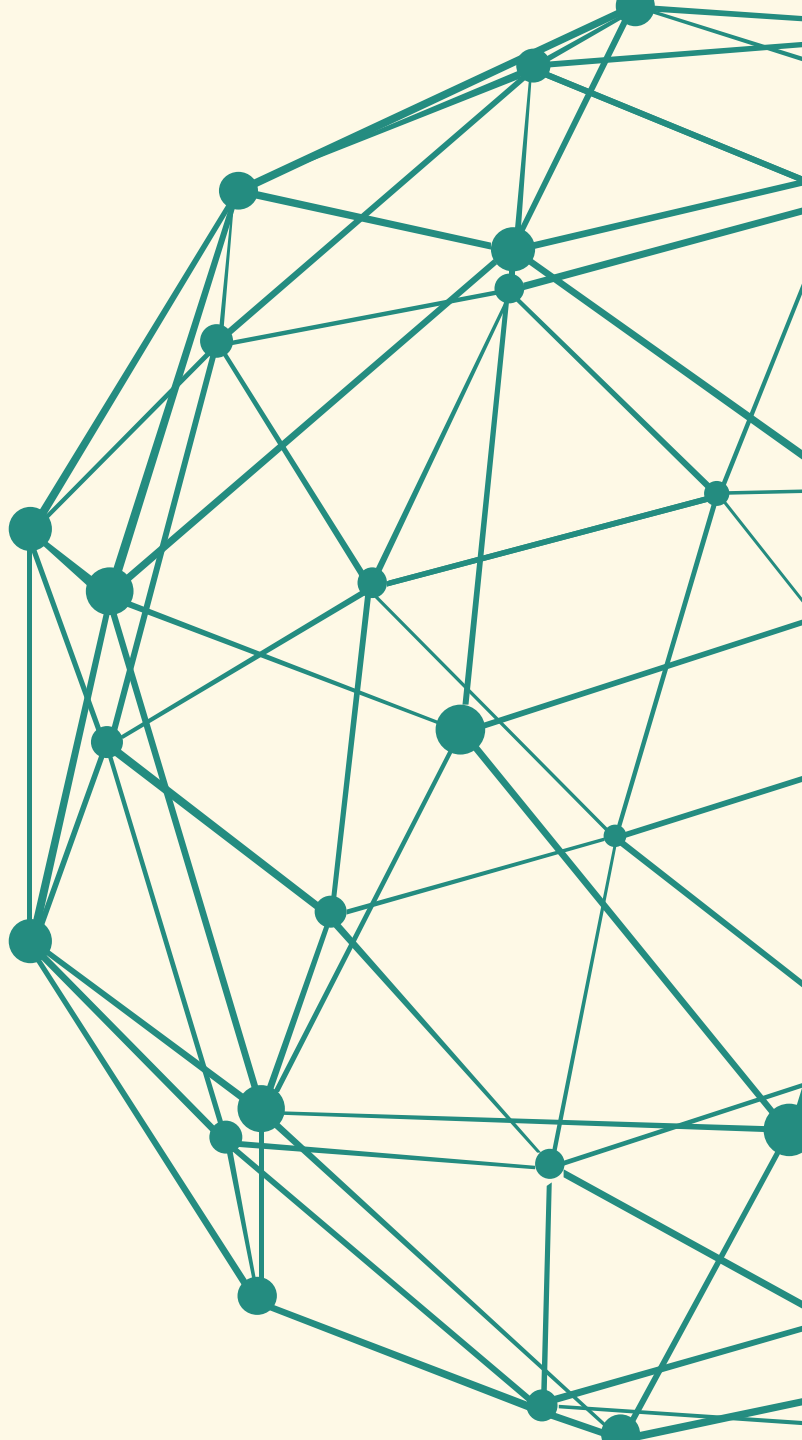
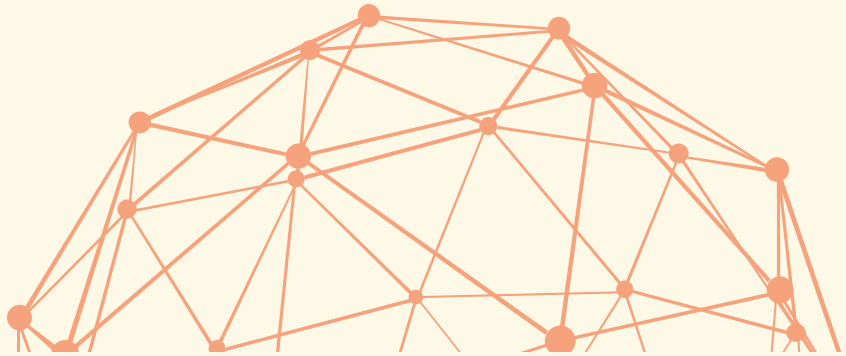
Tensor Wrold
人工智能讲座

TensorFlow基础及实现

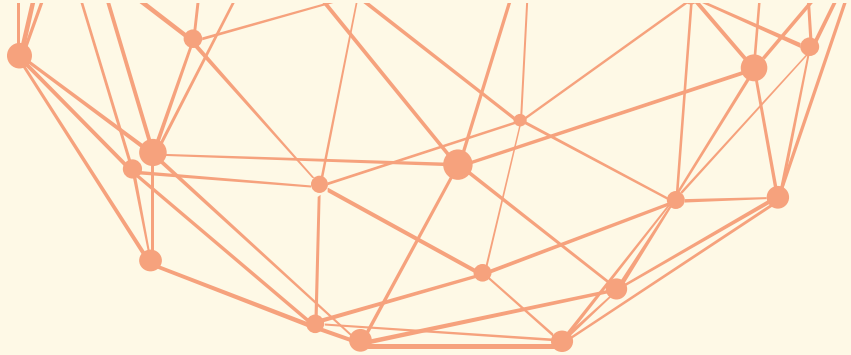
AI工程师讲座

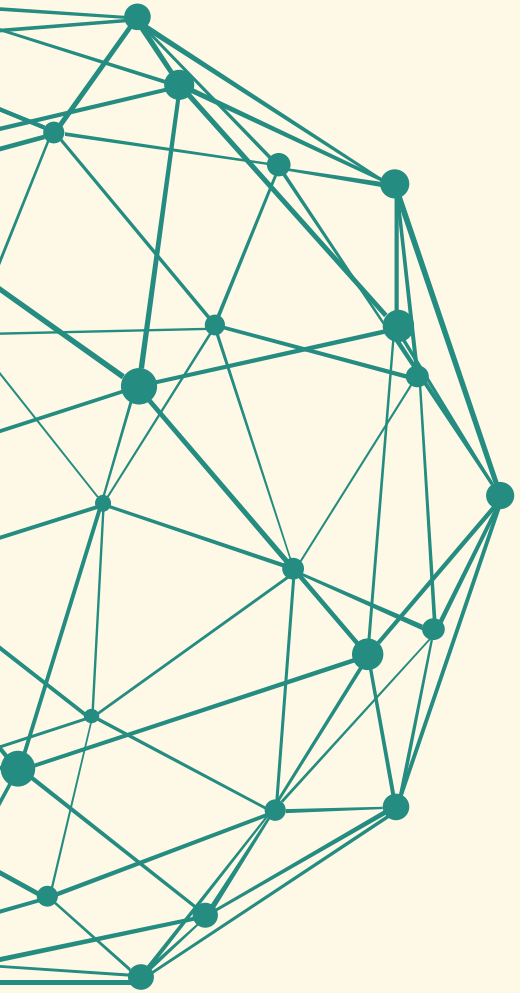
架构原理 深度学习架构建立





目录

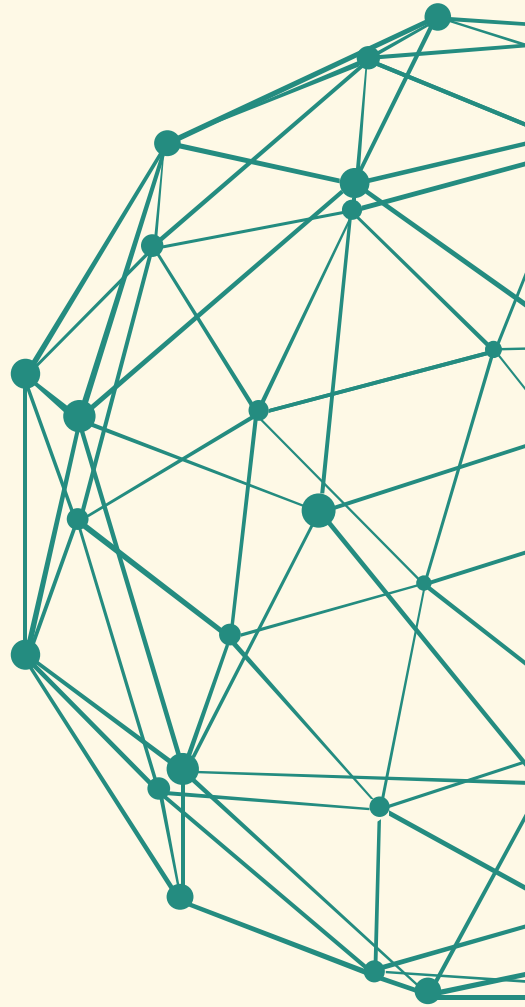




Part / 01

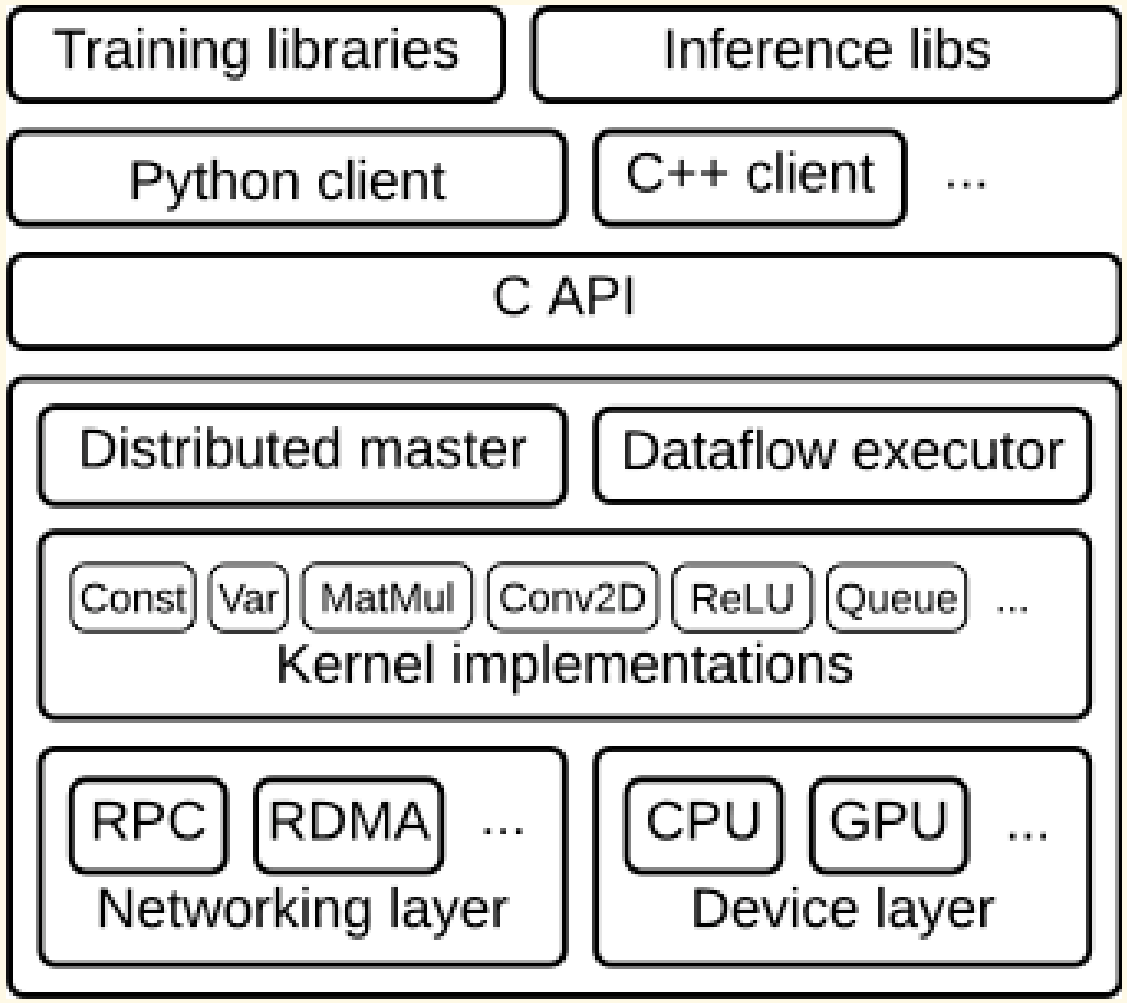
原理架构

ARCHITECTURE



原理架构
ARCHITECTURE

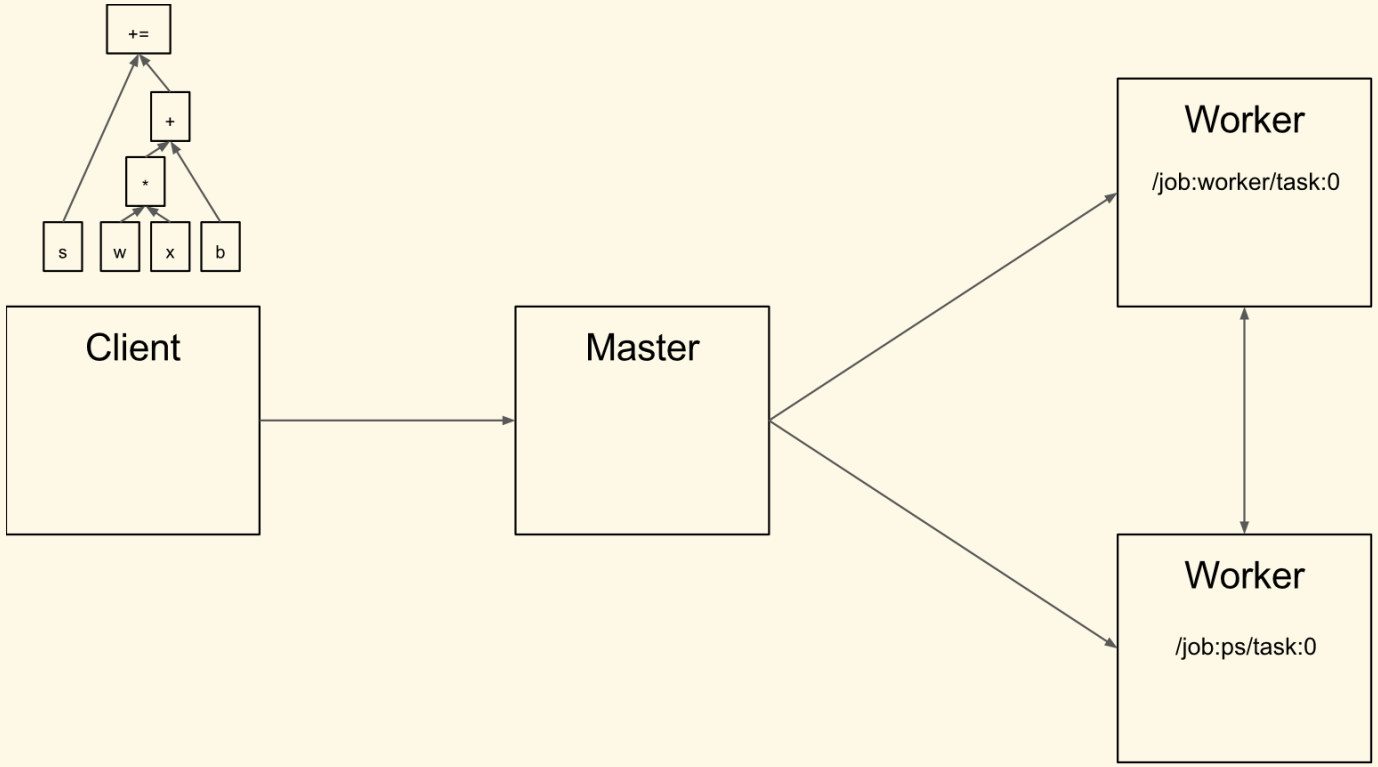
基础架构



计算核心的实现是建立于CPU与GPU之上的，在进行多设备计算的过程中需要考虑二者内存传递速度问题，信息传递用于实现分布式计算，在这之上实现的C API

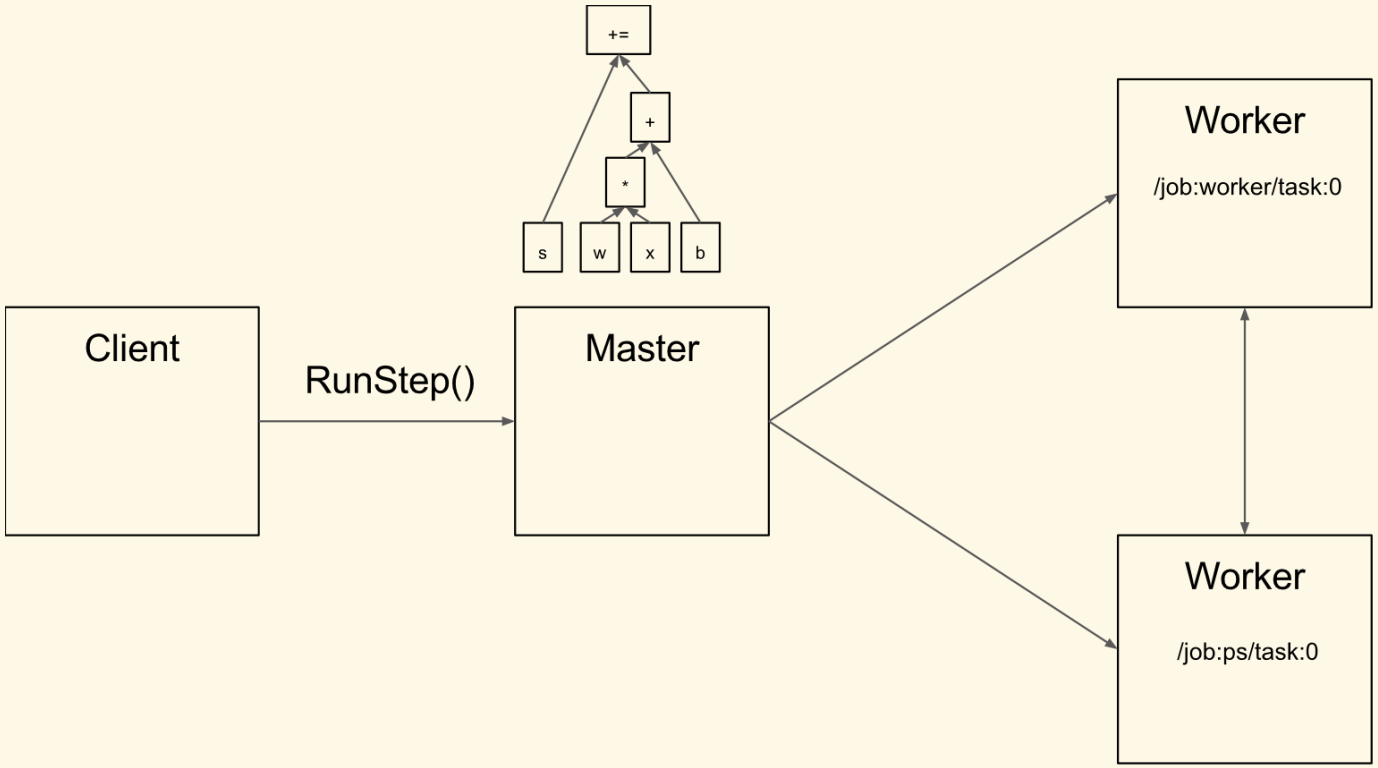
原理架构
ARCHITECTURE

计算图-单机执行过程抽象

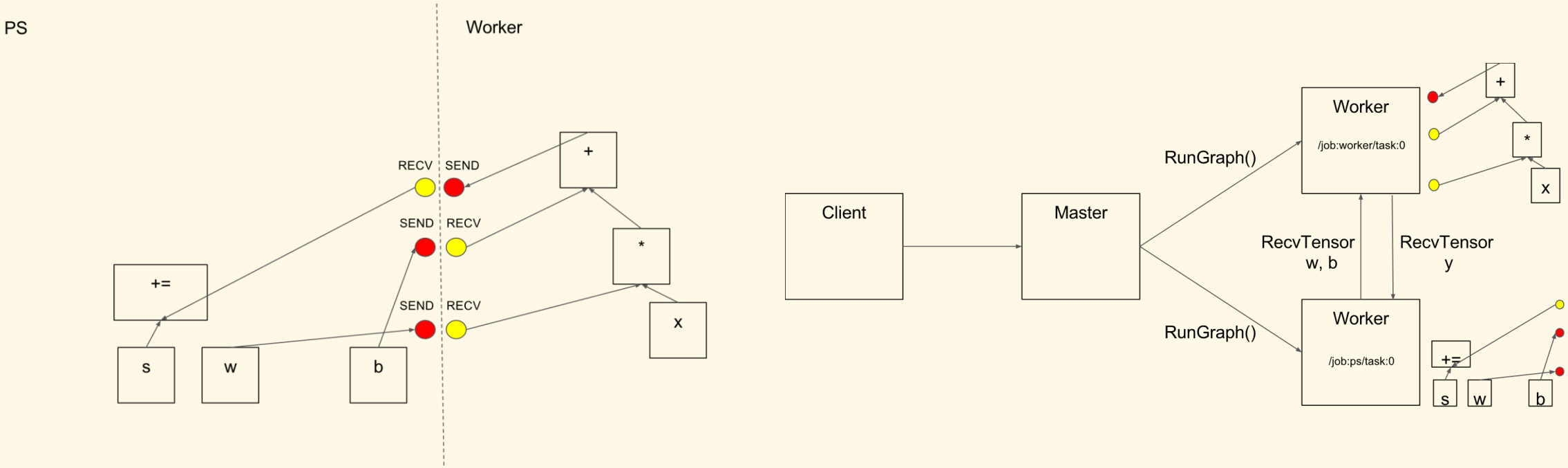


原理架构
ARCHITECTURE

计算图-分布式执行过程抽象



计算图-分布式执行过程抽象



并行化是在迭代计算过程中进行的，需要保证并行粒度足够大。

计算图-分布式执行过程抽象

设置计算任务输出计算设备信息：

```
tf.Session(config=tf.ConfigProto(log_device_placement=True))
```

输出：

```
MatMul: (MatMul):/job:localhost/replica:0/task:0/cpu:0
b: (Const): /job:localhost/replica:0/task:0/cpu:0
a: (Const): /job:localhost/replica:0/task:0/cpu:0
```

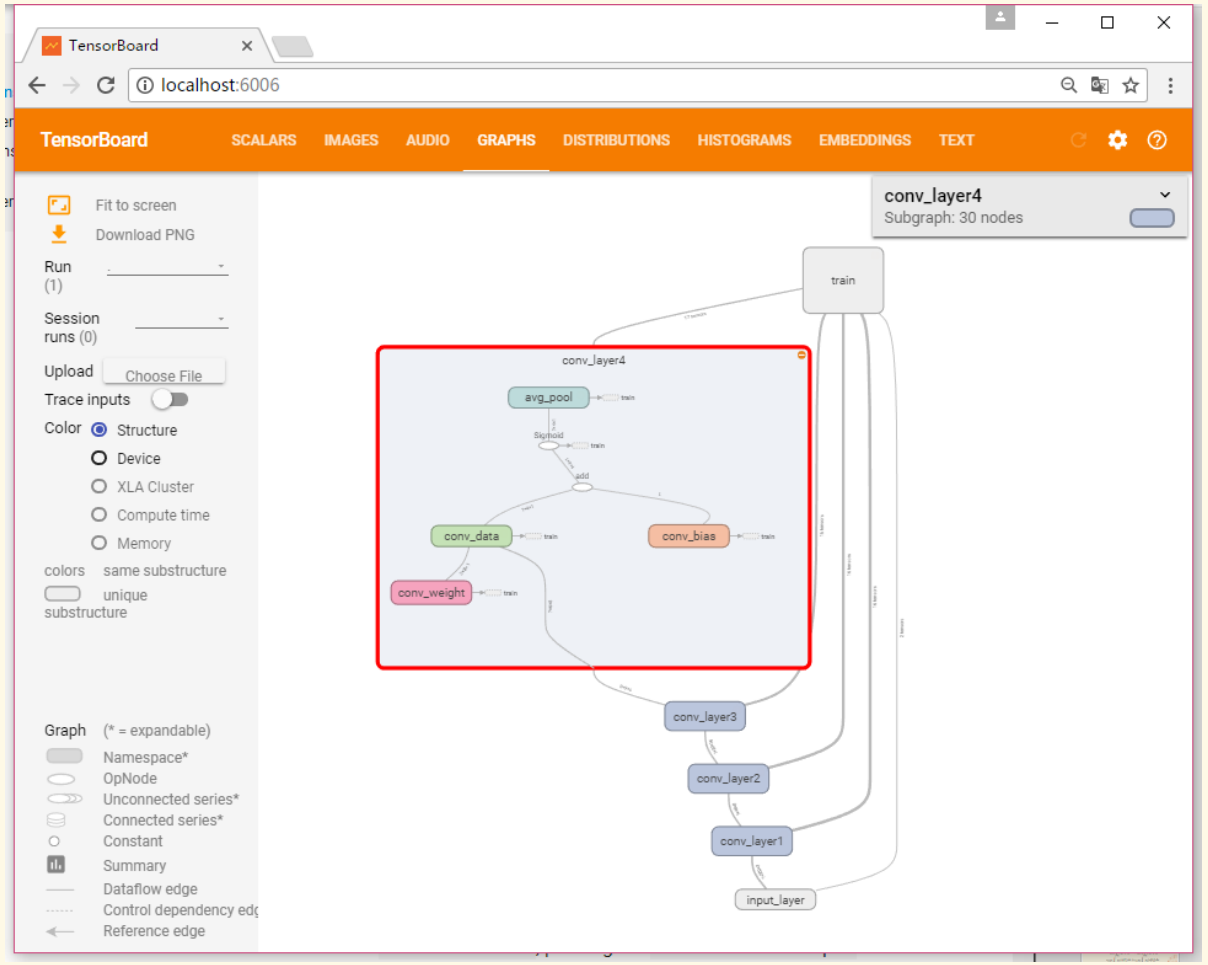
计算图

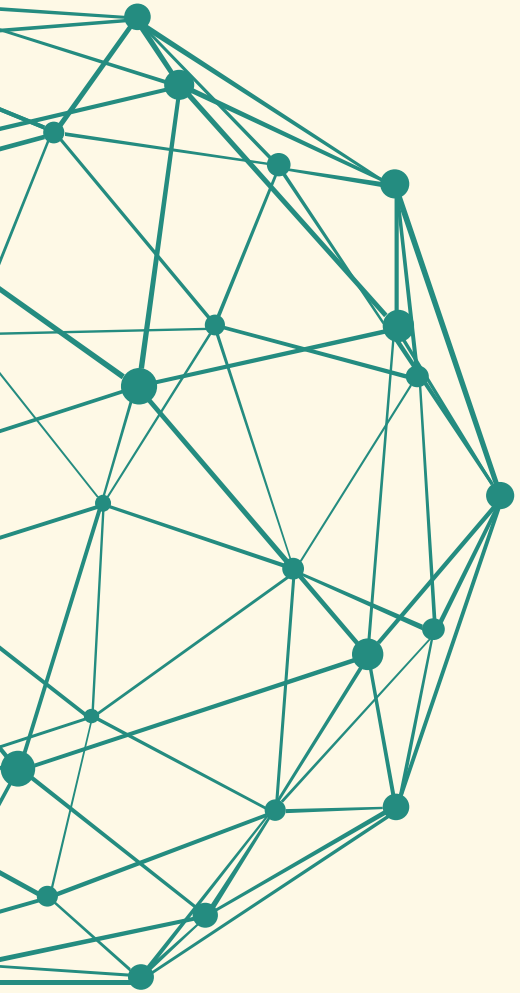
所有计算之目的在于获取梯度并更新权值。
回想数学章节的内容，这是整个机器学习优化过程的核心。
TensorFlow使用了“计算图”的方式去描述这个过程。

原理架构
ARCHITECTURE

计算图

TensorFlow输出的计算图(四层卷积)：

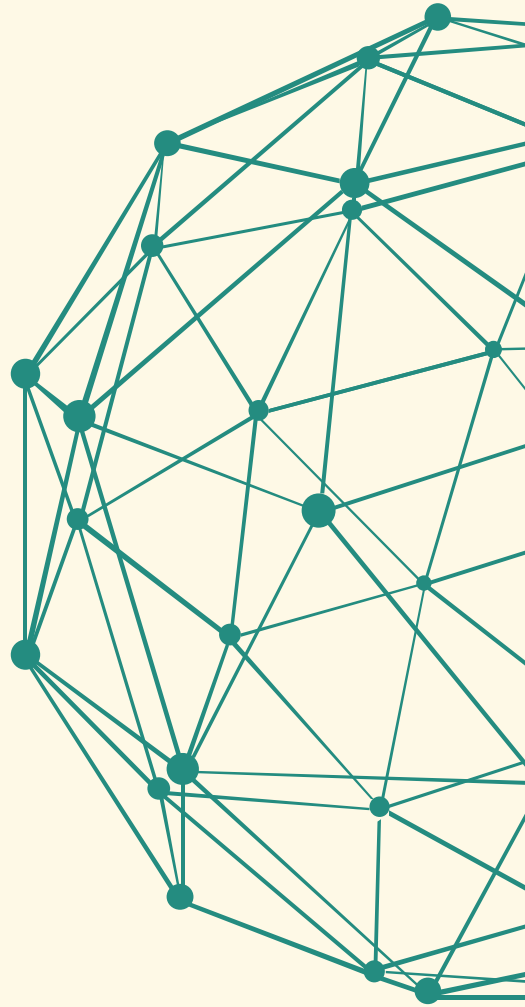




Part / 02

深度学习架构的建立

DESIGN



深度学习架构的建立

DESIGN

本节目的在于根据网上给出的深度学习架构能用TensorFlow的PythonAPI自行实现其过程。

本章中默认认为加入如下语句：

```
import tensorflow as tf
import numpy as np
```

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

在机器学习中可训练的量用矩阵表示为
tf.Variable()

对于W和b可以用如下语句:
W = tf.Variable(tf.zeros(W_shape))
使用初始化后的变量是好的习惯
或者使用
W = tf.get_variable(“W” ,[W_shape])

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

更多关于变量

上面的get_variable()已经展示了变量的域
这对于变量的复用是很重要的概念：

```
with tf.variable_scope("level1"):
    with tf.variable_scope("level2"):
        var = tf.get_variable("var", [3,3])
```

```
with tf.variable_scope("level3"):
    with tf.variable_scope("level4"):
        var = tf.get_variable("var", [3,3])
```

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

关于输入x

对于输入x，TensorFlow提供了placeholder类：

```
x = tf.placeholder(tf.float32, shape=batch_size+x_shape)
```

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

定义函数计算

W与x是矩阵相乘运算，与b是相加运算，这个运算过程定义如下：

matop = tf.matmul(x,W)+b

函数f是激活函数，这里选择激活函数为sigmoid

y = tf.nn.sigmoid(matop)

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

定义loss

利用计算出的y与d进行loss计算：

loss = tf.square(y-d)

这里需要注意的是，x,y,d是训练集的子集，所以需要进行平均计算(对于简单模型，不进行平均计算依然能得到结果)

loss_mean = tf.reduce_mean(loss)

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

定义优化算法

opmizer = tf.train.GradientDescentOpmizer(0.01)

定义迭代过程：

step = opmizer.minize(loss_mean)

迭代过程优化的变量为所有Variable()，除非将其设定为不可训练。

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

Session

至此计算过程并未执行，TensorFlow需要定义Session去执行定义的计算过程。

这里可以定义：

sess = tf.Session()

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

变量初始化

所有变量使用前都需要进行初始化

initer = tf.global_variables_initializer()

当然需要sess执行

sess.run（initer）

入门

回想：

$$\vec{y} = f(W \cdot \vec{x} + b)$$

$$\varepsilon = (d - y)^2$$

其中输入为x输出为y，训练标签为d

训练过程

```
for _ in range(step_len):
    data_d,data_x= get_data_class(batch_len)
    sess.run(step, feed_dict={x: data_x, d: data_d})
```

训练过程采用feed机制
对于定义的placeholder量。

子结构

CNN

tf.nn.conv2d(data, filters, strides, padding)

tf.nn.pool(data, w_shape, p_type, padding, strides)

tf.nn.dropout(data, keep_prob)

tf.nn.l2normalize(data, dim)

子结构

RNN

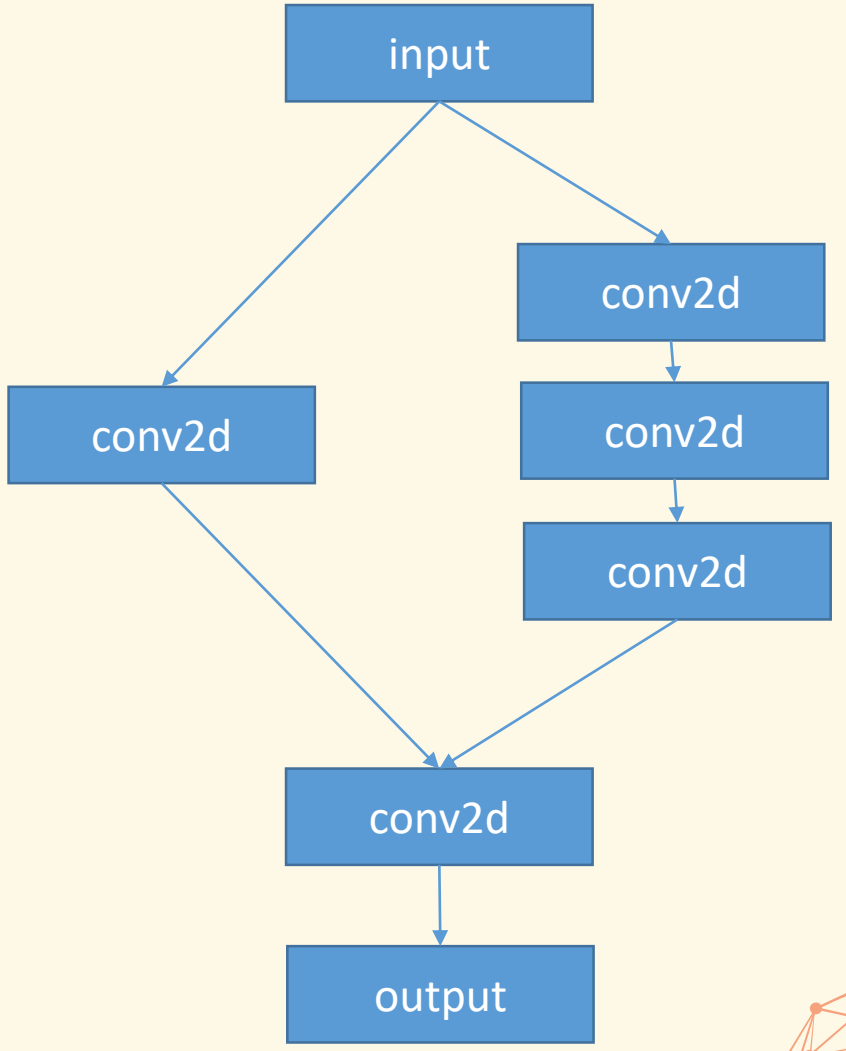
tf.contrib.rnn.BasicLSTMCell(lstm_size)

tf.contrib.rnn.DropoutWrapper()

tf.contrib.rnn.MultiRNNCell()

构建计算图

```
def def_struck(net, scale=1.0, activation_fn=tf.nn.relu, scope=None, reuse=None):
    """Def Struck"""
    with tf.variable_scope(scope, 'my_def', [net], reuse=reuse):
        with tf.variable_scope('Branch_0'):
            tower_conv = slim.conv2d(net, 128, 1, scope='Conv2d_1x1')
        with tf.variable_scope('Branch_1'):
            tower_conv1_0 = slim.conv2d(net, 128, 1, scope='Conv2d_0a_1x1')
            tower_conv1_1 = slim.conv2d(tower_conv1_0, 128, 7,
                                        scope='Conv2d_0b_1x7')
            tower_conv1_2 = slim.conv2d(tower_conv1_1, 128, 1,
                                        scope='Conv2d_0c_7x1')
        mixed = tf.concat([tower_conv, tower_conv1_2], 2)
        up = slim.conv2d(mixed, net.get_shape()[2], 1, normalizer_fn=None,
                         activation_fn=None, scope='Conv2d_1x1')
        net += scale * up
        If activation_fn:
            net = activation_fn(net)
        return net
```



THANKS

AI工程师讲座