## 基本概念 :

tensor: tf中数据被抽象的形式;0-d tensor：标量，1-d tensor：向量，2-d tensor：矩阵; 3-d tensor：张量(多个矩阵平行并排,cnn的输入即为张量)

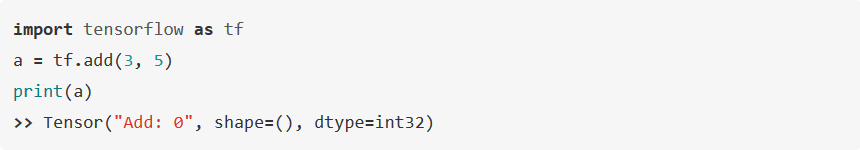
### 数据流图:

add

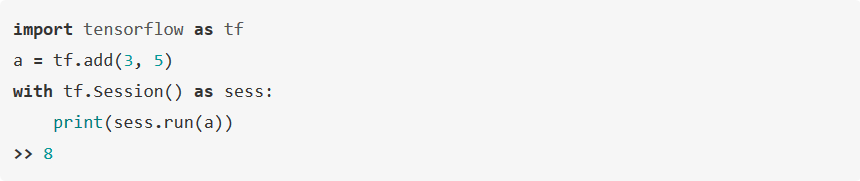
2

3

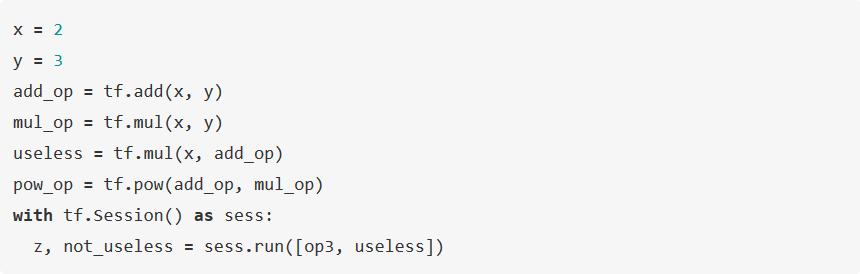
如图,数据流图里变量和操作(计算)都表示为节点



但此时并不能得到结果8,操作只有在session中才会被执行(跟spark类似)



sess.run可以用[]得到多个结果



使用数据流图但好处:

1. 节约计算资源，每次运算仅仅只需运行与结果有关的子图

2. 可以将图分成小块进行自动微分

3. 方便部署在多个设备上

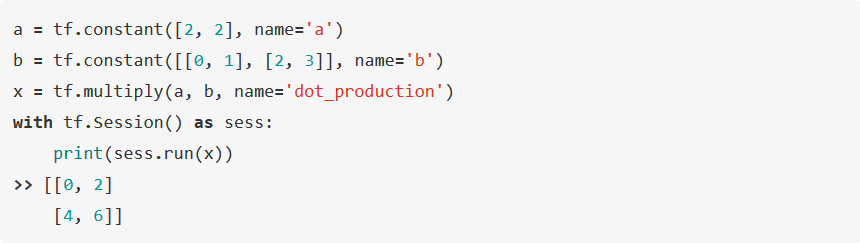
4. 很多机器学习算法都能够被可视化为图的结构

### 常数类型(Constant types)

创建一个常数

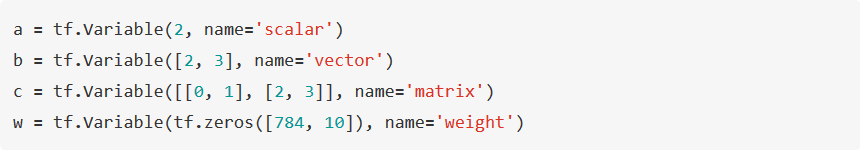
C:\Users\user\Downloads\Screenshot-2018-1-10 cs20si：tensorflow for research 学习笔记2.png

比如建立一维向量和矩阵，然后将他们乘起来(类似numpy里的操作)

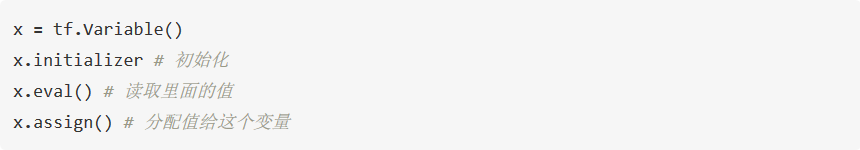


### 变量(Variable)

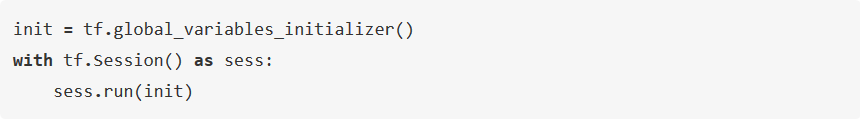
常量会存在计算图的定义当中，如果常量过多，这会使得加载计算图变得非常慢，同时常量的值不可改变，所以引入了变量。



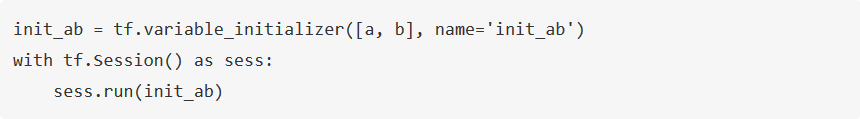
变量有以下操作



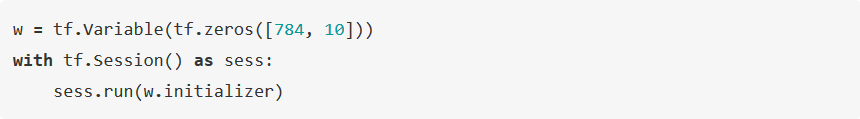
在使用变量之前必须对其进行初始化，初始化可以看作是一种变量的分配值操作。最简单的初始化方式是一次性初始化所有的变量(全局初始化)



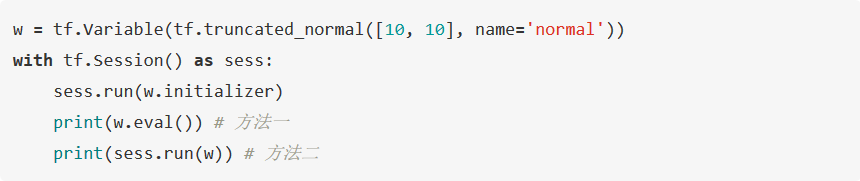
也可以对某一部分变量进行初始化



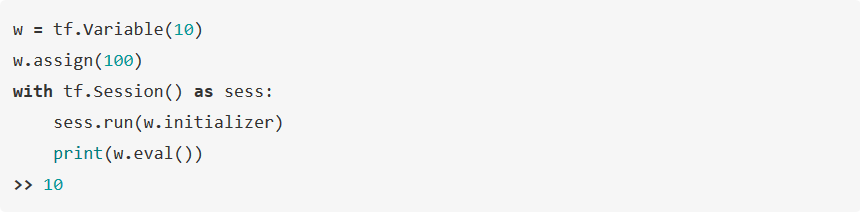
或者是对某一个变量进行初始化



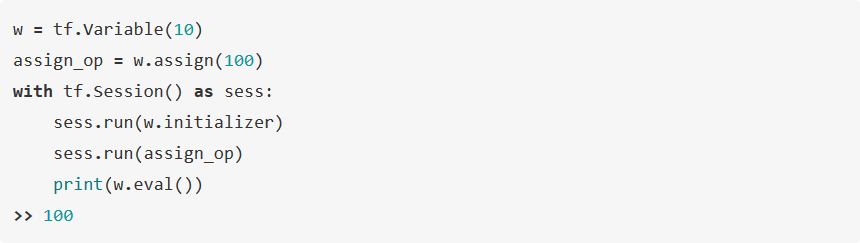
想取出变量的值，有两种方法



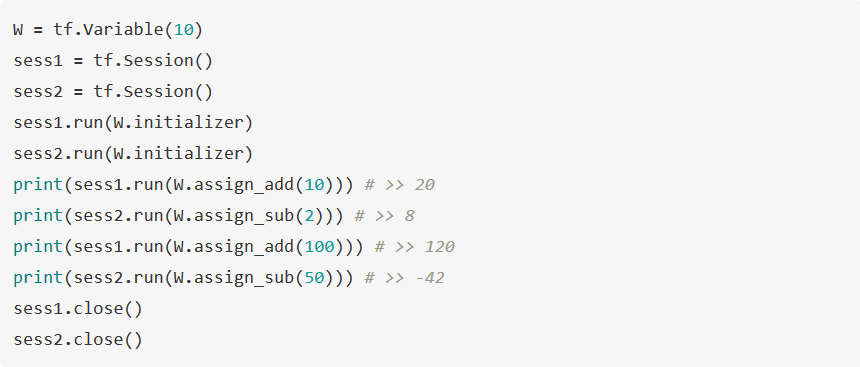
看以下例子



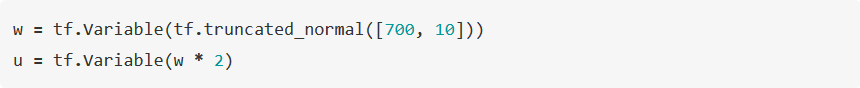
输出结果为10,是因为我们虽然使用里w.assign()进行赋值,但是这在数据图里被视为一个操作,而sess中没有执行这个操作;想要赋值就必须在sess中执行这个操作,如图



tensorflow的每个session是相互独立的，我们可以看看下面这个例子



可以根据一个变量来定义一个变量(turncated\_normal用于生成正态分布的数据)



### 占位符(Placeholders)

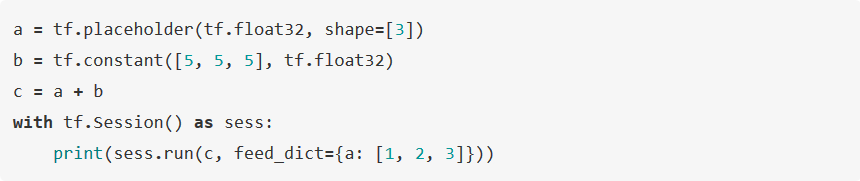
tensorflow中一般有两步，第一步是定义图，第二步是在session中进行图中的计算。对于图中我们暂时不知道值的量，我们可以定义为占位符，之后再用`feed\_dict`去赋值。

定义占位符的方式非常简单

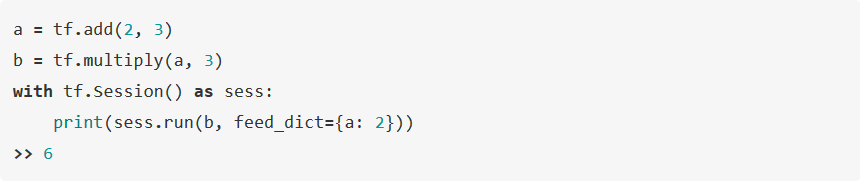
C:\Users\user\Downloads\Screenshot-2018-1-10 cs20si：tensorflow for research 学习笔记2(6).png

其中dtype是必须要指定的参数，shape如果是None，说明任何大小的tensor都能够接受，使用shape=None很容易定义好图，但是在debug的时候这将成为噩梦，所以最好是指定好shape。

给出下面的小例子

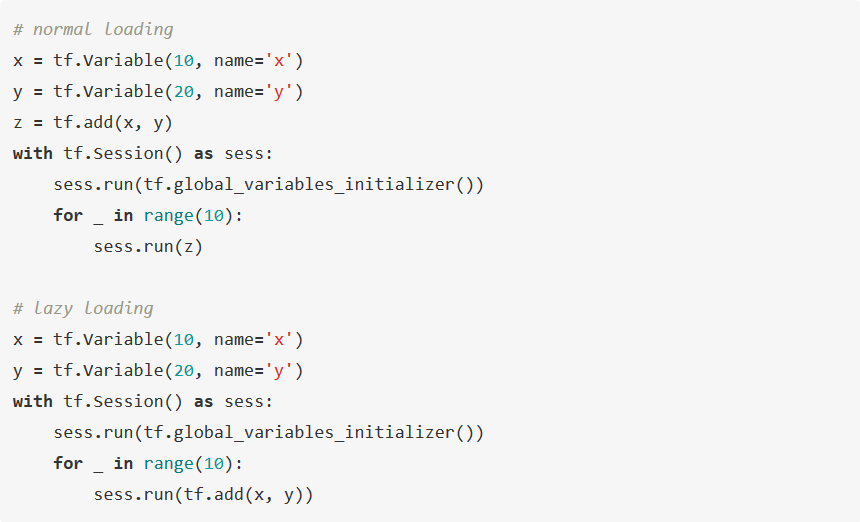


也可以给tensorflow中的运算进行feed操作



### lazy loading

lazy loading是指你推迟变量的创建直到你必须要使用他的时候。下面我们看看一般的loading和lazy loading的区别。



normal loading 会在图中创建x和y变量，同时创建x+y的运算，而lazy loading只会创建x和y两个变量。这不是一个bug，那么问题在哪里呢？

normal loading在session中不管做多少次x+y，只需要执行z定义的加法操作就可以了，而lazy loading在session中每进行一次x+y，就会在图中创建一个加法操作，如果进行1000次x+y的运算，normal loading的计算图没有任何变化，而lazy loading的计算图会多1000个节点，每个节点都表示x+y的操作。

这就是lazy loading造成的问题，这会严重影响图的读入速度。

## TensorBoard 可视化

tensorflow不仅仅是一个计算图软件，其还包含了tensorboard可视化工具，安装tensorflow的时候会默认安装，使用方法非常简单，使用writer = tf.summary.FileWriter('./graph', sess.graph)就能够创建一个文件写入器，./graph是存储目录，sess.graph表示读入的图结构。

我们可以写一个简单的小程序



然后打开终端，运行程序，接着输入`tensorboard --logdir="./graphs"`，然后打开网页输入 http://localhost:6006/，就能够进入tensorboard，可以得到下面的结果。

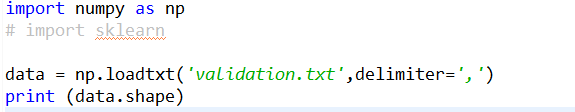
tf搭建神经网络

<https://www.jianshu.com/p/e112012a4b2d>

## Dataset

tf的数据集,用于导入数据以及做一些基本的处理

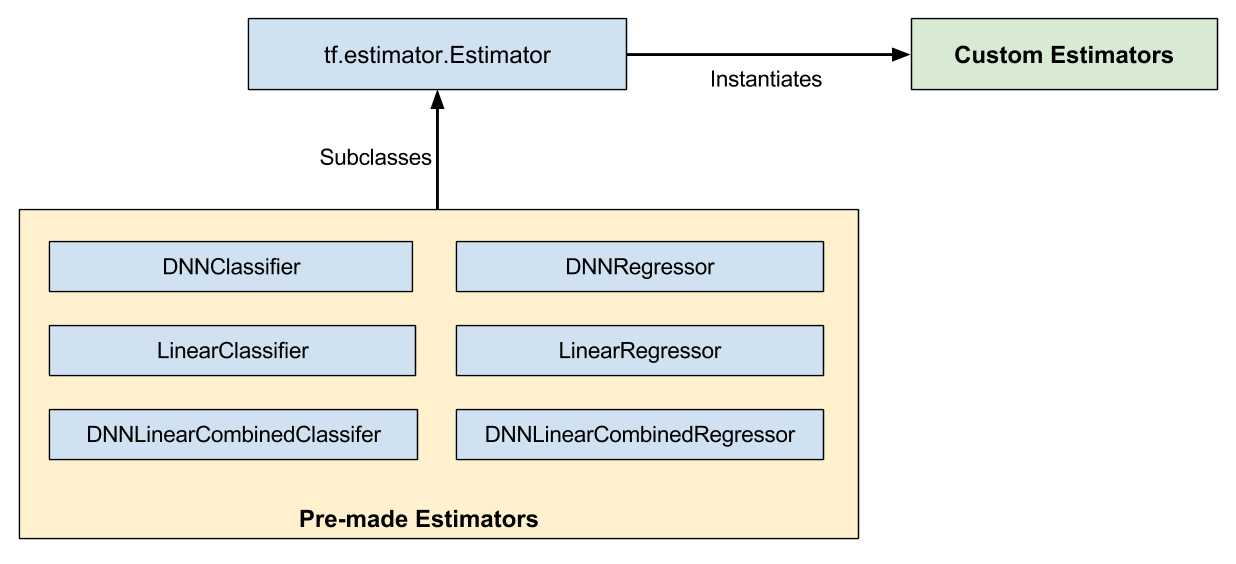
数据集划分:这里使用sklearn库做划分

示例:

## Dataset---Estimator

### 前言

estimer(估算器)是tensorflow的高级api,使用它可以快速的搭建模型框架，有预定义好的（比如dnnclassifier）,只需要配置少量关键参数即可进行得到一个神经网络，当然也允许自定义模型结构。



estimer是一个框架，提供模型的训练，模型的评估以及预测等功能

### mode\_fn

作为一个框架，你需要想它提供你想要使用的模型信息（自定义模型），通过传递模型（mode\_fn）函数实现，一个模型函数需要实现模型结构，定义损失函数，选择优化器（optimizer）等以及模型在三种状态下的行为，即训练，评估和预测

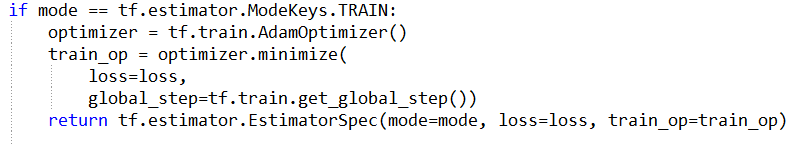
train

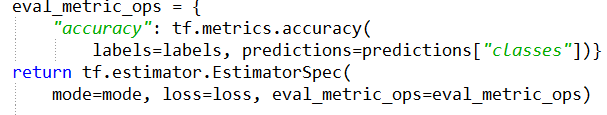
eval

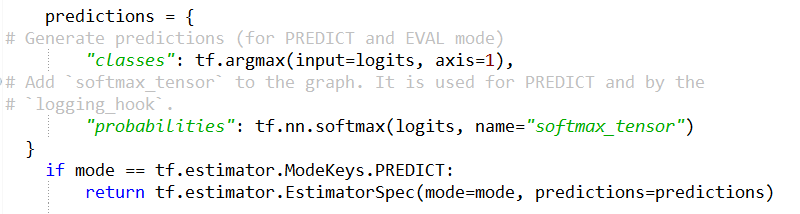
pridict

每个模式都必须返回estimator.EstimatorSpec

在train模式中，需要提供train\_op,它负责训练参数以优化损失函数



在eval模式中，需要提供所需要的评估指标，在下例中是准确率（accuracy），模型会打印该指标信息

在pridict模式中，需要提供分类类别标签，以及模型判段该分类的置信度，在下例中，输出层为logit层(有x个类别则输出x维向量)，tf.argmax选取其中最大维度的索引为输出，tf.nn.softmax则提供一种指数平滑作为其置信度

### estimator

定义好mode\_fn之后即可实例化estimator



然后对模型进行训练



### input\_fn

调用estimator任何一种模式都需要传递数据，这里通过input\_fn为模型提供数据，因为函数作为参数传递的时候无法指定其本身的参数，这里使用python的lambda传递。

input\_fn应具有以下格式：

返回值为 [标签]，｛’value\_name’:[values]｝

可选模式为perform\_shuffle：bool值，表示是否随机打乱数据集

repeat\_count，int值，表示遍历数据集的次数

可以通过tf.dataset API来构造input\_fn

以读取csv文件的数据为例

dataset = tf.data.TextLineDataset(file\_path)

读取csv文件的每一行为bytes数组，对每一行调用以下函数进行解码，并构造出特征列以及标签列

def parseCSVLine(line):

parsed\_line = tf.decode\_csv(line, \_CSV\_COLUMN\_DEFAULTS)

label = parsed\_line[-1]

del parsed\_line[-1]

features = parsed\_line

d = dict(zip(feature\_names, features))

return d,label

打乱数据集

if perform\_shuffle:

# Randomizes input using a window of 256 elements (read into memory)

dataset = dataset.shuffle(buffer\_size=256)

设置遍历数据集次数

dataset = dataset.repeat(repeat\_count)

设置批量提供数据

dataset = dataset.batch(32)

dataset只能通过构造迭代器逐行（批）读取，最后返回

iterator = dataset.make\_one\_shot\_iterator()

batch\_features, batch\_labels = iterator.get\_next()

return batch\_features, batch\_labels

### 参考文档

<http://developers.googleblog.cn/2017/12/tensorflow.html>

<http://developers.googleblog.cn/2018/01/tensorflow.html>

<https://www.tensorflow.org/tutorials/layers>

<https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html>