Tensorflow 工具介绍
Introduction to Tensorflow

Ivan
Aug. 27th, 2017

Disclaimer: Slides based on Stanford CS20, Tensorflow for Deep Learning Research

Outline（提纲）

Tensorflow概览	• Overview of TensorFlow
图与会话	• Graphs and Sessions
TensorBoard简介	• TensorBoard Introduction
常量和占位符	• Constants and Placeholders
在Tensorflow中使用逻辑回归模型对MNIST进行分类	• MNIST Classification with logistic regression using TensorFlow

TensorFlow

- What's TensorFlow?
  - Open source library for numeric and symbolic computation with GPU support
  - Developed by the Google Brain Team for machine learning related research

We will compare TensorFlow with Numpy to highlight its key features

- 什么是TensorFlow?

它是GPU支持下的进行数值和符号计算的开源库

它是由Google Brain团队开发、应用于机器学习相关领域研究的工具

用TensorFlow与Numpy进行比较以突出其重要特性

TensorFlow

- Besides TensorFlow

- Caffe / Caffe 2 (Berkeley / Facebook)
- Torch (NYU)
- Theano (University of Montreal)
- CNTK (Microsoft)
- Paddle (Baidu)
- MXNet (Amazon)
- PyTorch (Facebook)

- 除了TensorFlow，其它框架有...

Caffe/Caffe 2(加州大学伯克利分校/Facebook)

Torch(纽约大学)

Theano(蒙特利尔大学)

CNTK(微软)

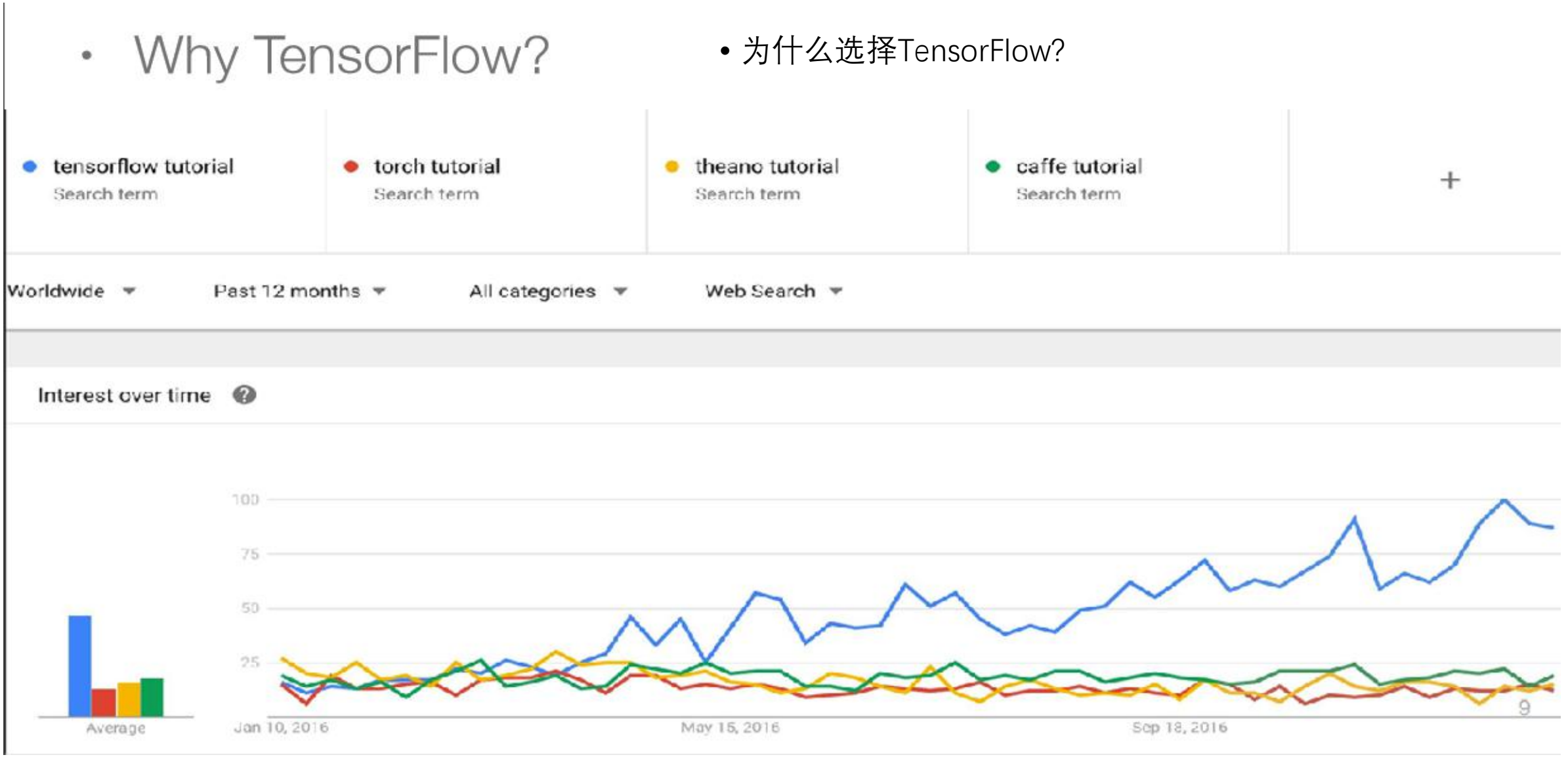
Paddle(百度)

MXNet(亚马逊)

PyTorch(Facebook)

- Why TensorFlow?

- 为什么选择TensorFlow?



TensorFlow

- Why TensorFlow?

- Python API
- Portability: easy to deploy computations over one or more CPUs/ GPUs, with the same API
- Flexibility: easy to extend to mobile devices, including Android, iOS, etc.
- Visualization: TensorBoard is great!
- **Auto-differentiation: autodiff, no need to compute the gradient manually**
- Large community: > 10,000 commits and > 3,000 TF-related repos in 1 year

- 为什么选择TensorFlow?

- Python API
- 可移植性：容易用相同的API在一个或多个CPU/GPU上部署计算
- 灵活性：容易拓展到移动端设备，包括安卓，ios等
- 可视化：可视模块TensorBoard非常出色
- 自动差分：autodiff，无需手工计算梯度
- 大社区：一年内有超过10000份提交和超过3000个TF相关项目

- Companies that use TensorFlow

- Google
- DeepMind
- Dropbox
- Snapchat
- Uber
- eBay
- OpenAI
- ...

- 使用TensorFlow的公司

- Goals of this lecture

- Understand TF's computation graph approach
- Explore TF's built-in functions
- Be familiar with the pipeline of a typical machine learning project (MNIST image classification using TF)

- 本节课程的目标
- 理解TF的计算图方法
- 探索TF的内置函数
- 熟悉典型的机器学习项目的流程（使用TF对MNIST图像进行分类）

import tensorflow as tf

引言

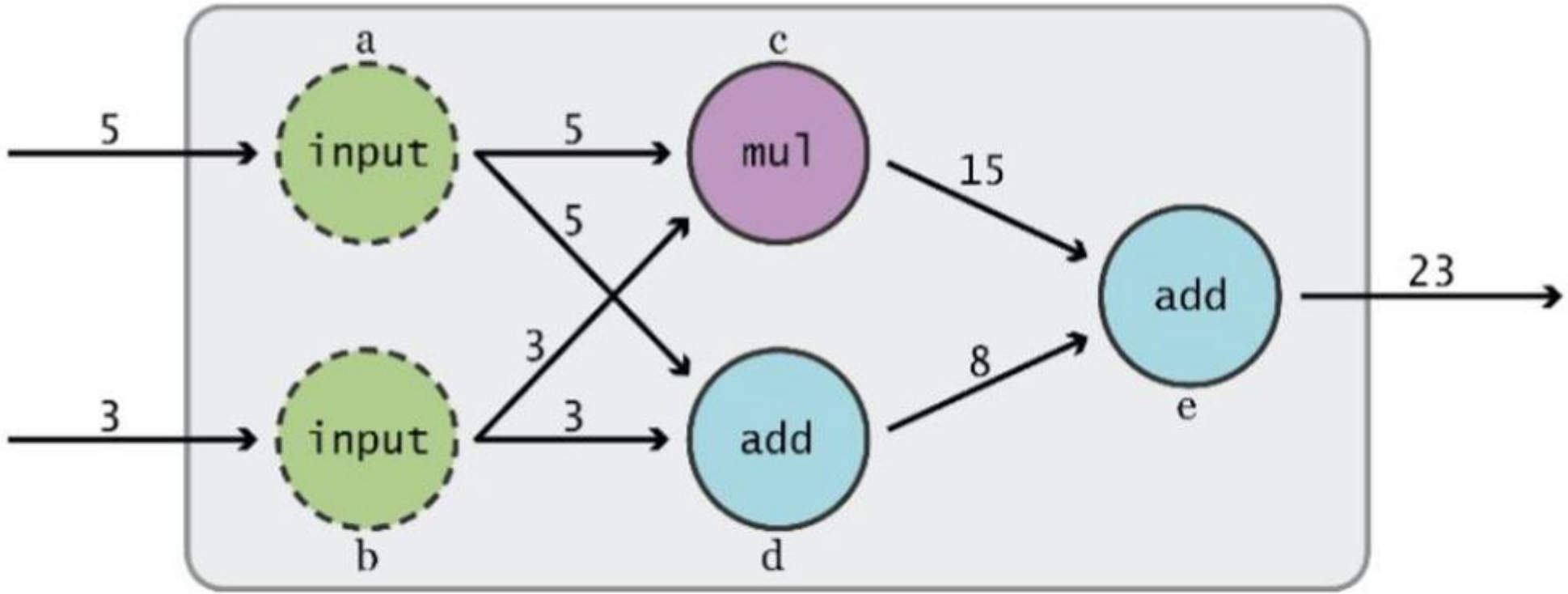
- Even higher level abstraction of TF:

- TF Learn (tf.contrib.learn): simplified interface of TensorFlow, similar to scikit-learn
- TF Slim (tf.contrib.slim): lightweight library for defining and running complex models in TensorFlow
- Even more: **Keras**

- 更高层的TF抽象模型

- TF Learn（tf.contrib.learn）：TensorFlow的简化版接口，类似于scikit-learn
- TF Slim（tf.contrib.slim）：用于在TensorFlow中定义和运行复杂模型的轻量库
- 更多：**Keras**

- Data flow graphs • 数据流图示

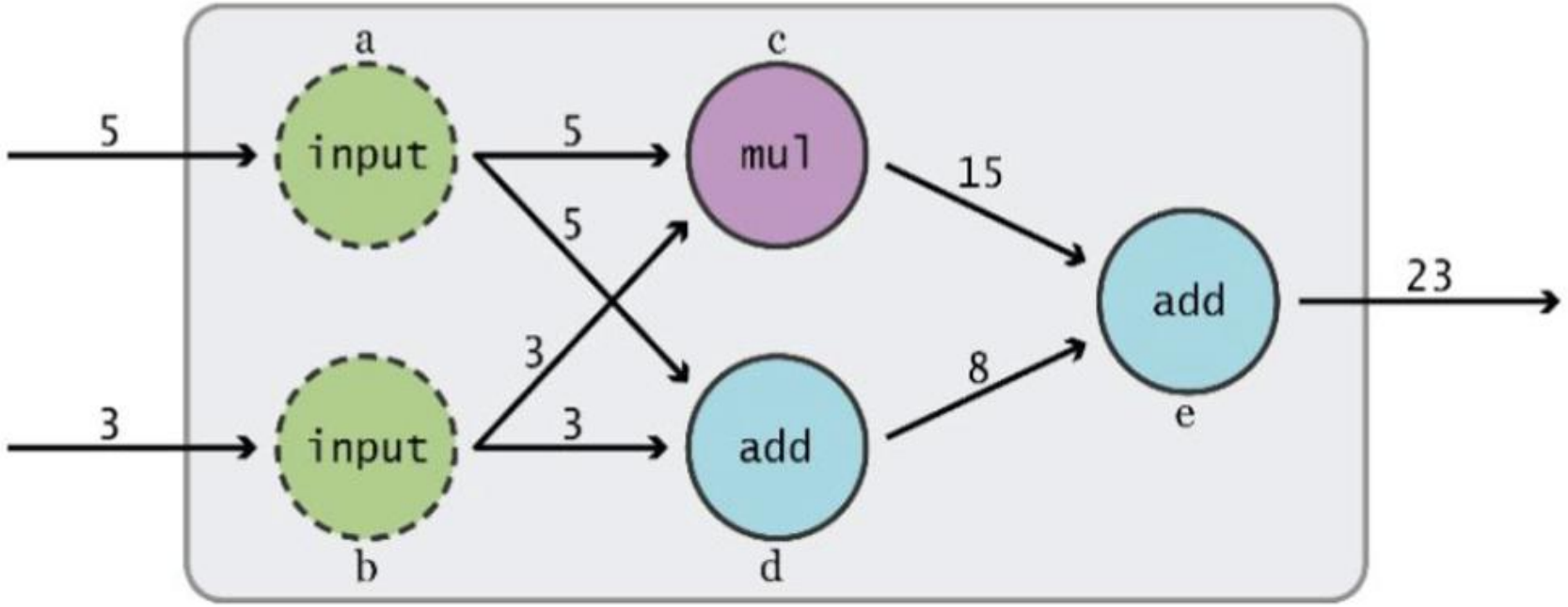


Defining computation ≠ Execution of Computation

(定义运算≠运算的执行)

图表与会话

- Data flow graphs
- 数据流图示

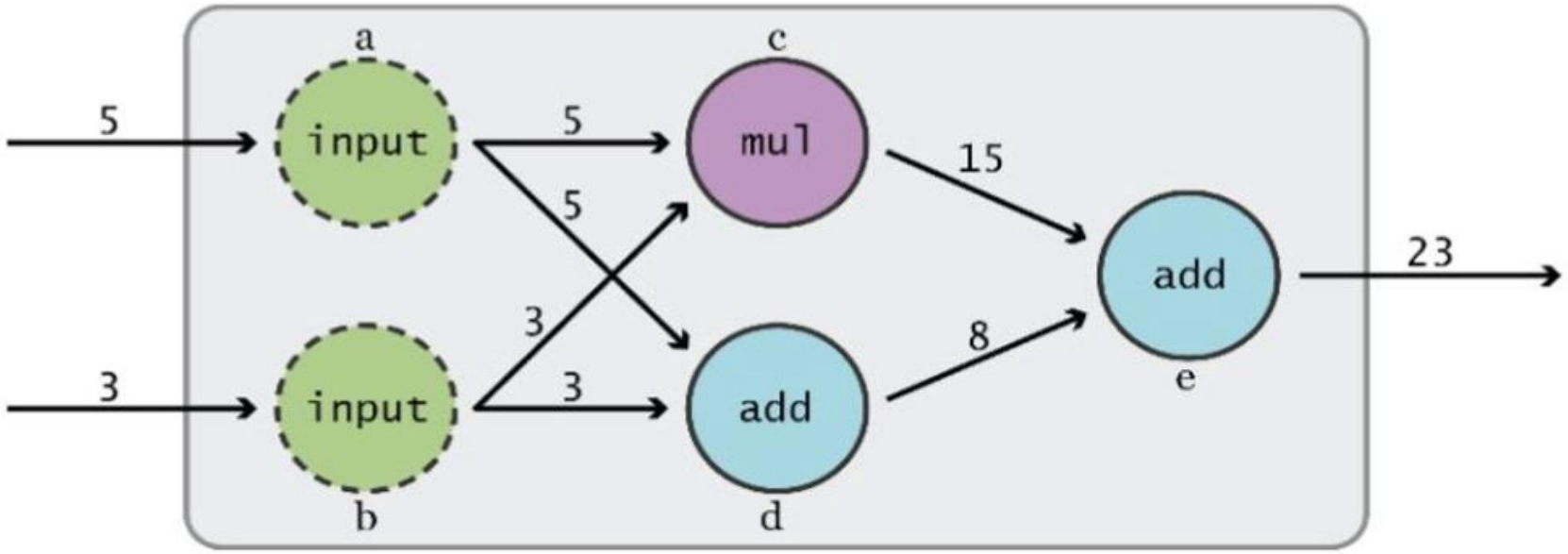


- Typical pipeline in TF:
  - Define the computation graph
  - Run session to execute the computational graph

- TF的典型流程：
  - 定义运算图表
  - 执行会话以运行计算图

图表与会话

- Data flow graphs •数据流图示



- TensorFlow = Tensor + Flow:
  - Tensor = data
  - Flow = operators

- TensorFlow=Tensor+Flow
  - Tensor=数据
  - Flow=操作符

Data are transmitted and transformed by operators (op) in the computational graph
（数据经过操作符时进行变换，并依据计算图进行传递）

TensorFlow

- What is a tensor?
- An n-dimensional array
  - 0-d tensor: scalar (number)
  - 1-d tensor: vector
  - 2-d tensor: matrix
  - …

- 张量是什么？
- 一个n维数组
  - 0维张量：标量
  - 1维张量：向量
  - 2维张量：矩阵
  - …

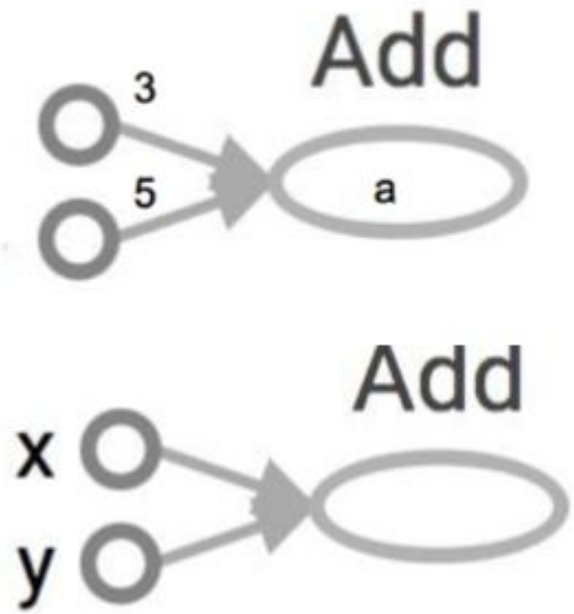
- Numpy vs TensorFlow

Numpy	TensorFlow
a = np.zeros((2,2)); b = np.ones((2,2))	a = tf.zeros((2,2)); b = tf.ones((2,2))
np.sum(b, axis=1)	tf.reduce_sum(b, reduction_indices=[1])
a.shape	a.get_shape()
np.reshape(a, (1, 4))	tf.reshape(a, (1, 4))
5*b+1	5*b+1
np.dot(a, b)	tf.matmul(a, b)
a[1, 1], a[:, 1], a[1, :]	a[1, 1], a[:, 1], a[1, :]

Data flow graphs (数据流图)

```
import tensor flow as tf
a=tf.add(3,5)
print a
```

What is x, y? (x,y是什么？)
x = 3, y = 5;
TF will automatically name variables
（TF会自动对变量进行命名）
Nodes: operators, variables, or constants Edges: tensors
（节点：操作符，变量或常量　　边：张量）

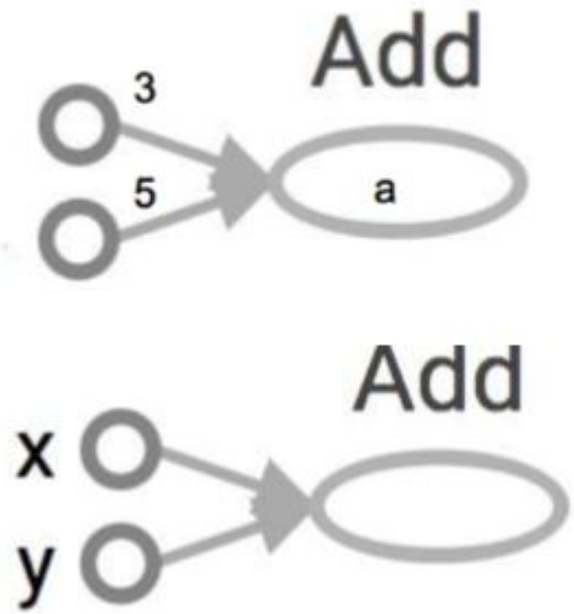


Visualization from TensorBoard
(TensorBoard可视化结果)

TensorFlow

Data flow graphs (数据流图)

```
import tensor flow as tf
a=tf.add(3,5)
print a
```

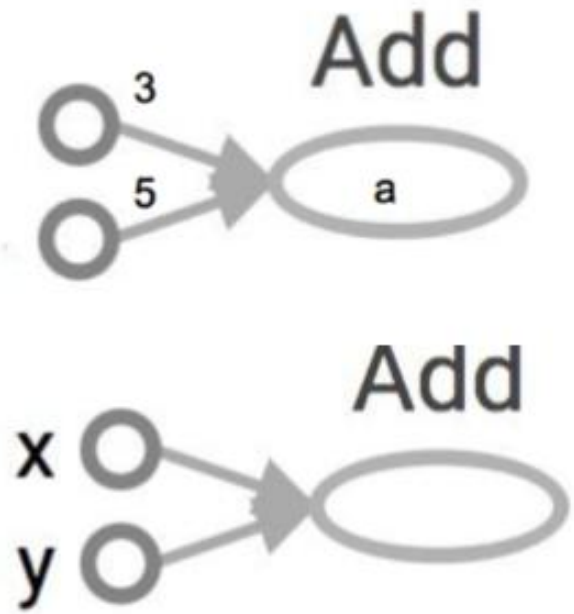


Visualization from TensorBoard
(TensorBoard可视化结果)

>> Tensor("Add:0", shape=(), dtype=int32)
(Not 8)
Why? 为啥不是8？

Data flow graphs (数据流图)

```
import tensor flow as tf
a=tf.add(3,5)
print a
```

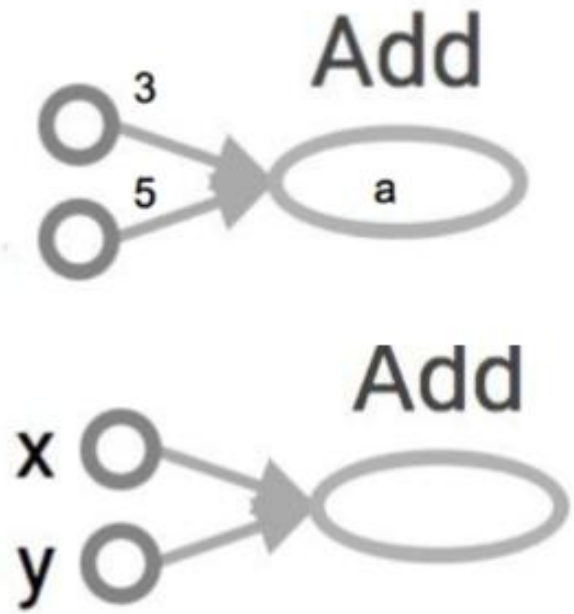


Visualization from TensorBoard
(TensorBoard可视化结果)

>> Tensor("Add:0", shape=(), dtype=int32)
(Not 8)
Symbolic variable! How to get the value of a?
（结果不是8，符号变量！怎样才能得到a的值？）

Data flow graphs (数据流图)

```
import tensor flow as tf
a=tf.add(3,5)
print a
```

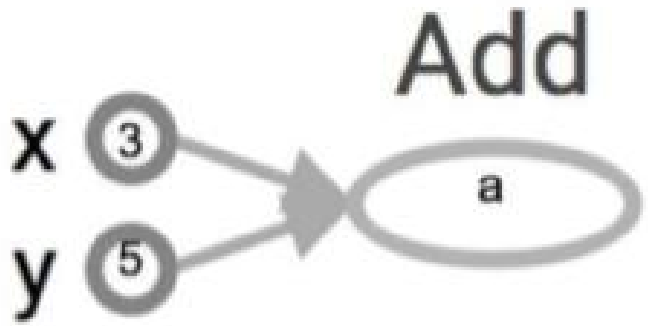


We need to create a **session** in order to get the value of a
(我们需要创建一个会话以得到a的值)

Visualization from TensorBoard
(TensorBoard可视化结果)

Create a session（创建会话）

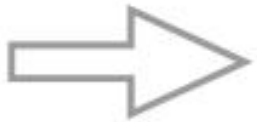
```
import tensor flow as tf
a=tf.add(3,5)
sess = tf.Session()
print sess.run(a)      >> output 8
sess.close()
```



Session will find the dependency of a, and computes all the nodes that lead to a.
(会话将找到a的依赖，并且计算所有指向a的节点)

Create a session(Recommended practice)
(创建会话，操作规程建议)

```
import tensor flow as tf
a=tf.add(3,5)
sess = tf.Session()
print sess.run(a)
sess.close()
```



```
import tensor flow as tf
a = tf.add(3, 5)
with tf.Session() as sess:
    print sess.run(a)
```

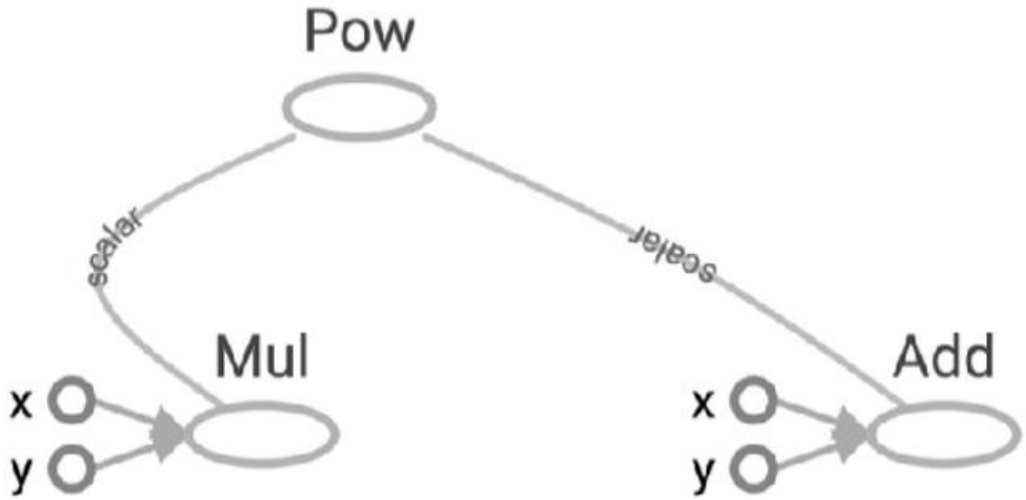
Session will find the dependency of a, and computes all the nodes that lead to a.
(会话将找到a的依赖，并且计算所有指向a的节点)

Summary (摘要)

A Session object encapsulates the running environment such that operators are executed and tensors are evaluated
(一个会话对象封装了运行的环境后，这样操作符就会被执行，并且对张量进行计算)

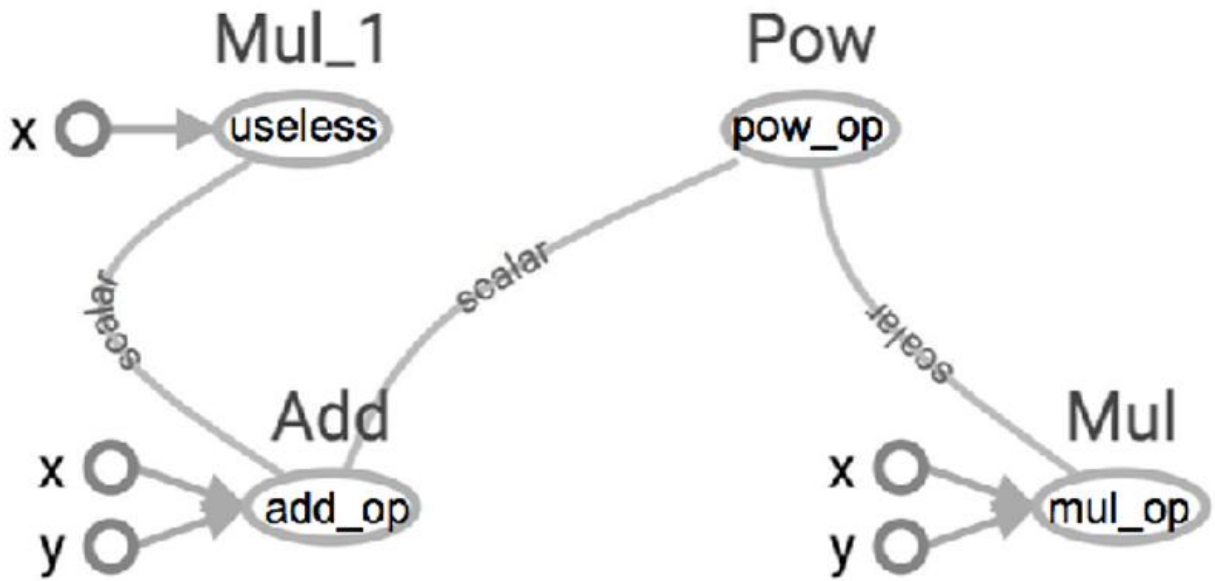
More examples (更多示例)

```
import tensor flow as tf
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as sess:
    op3 = sess.run(op3) print op3
>> output 7776
```



More examples (更多示例)

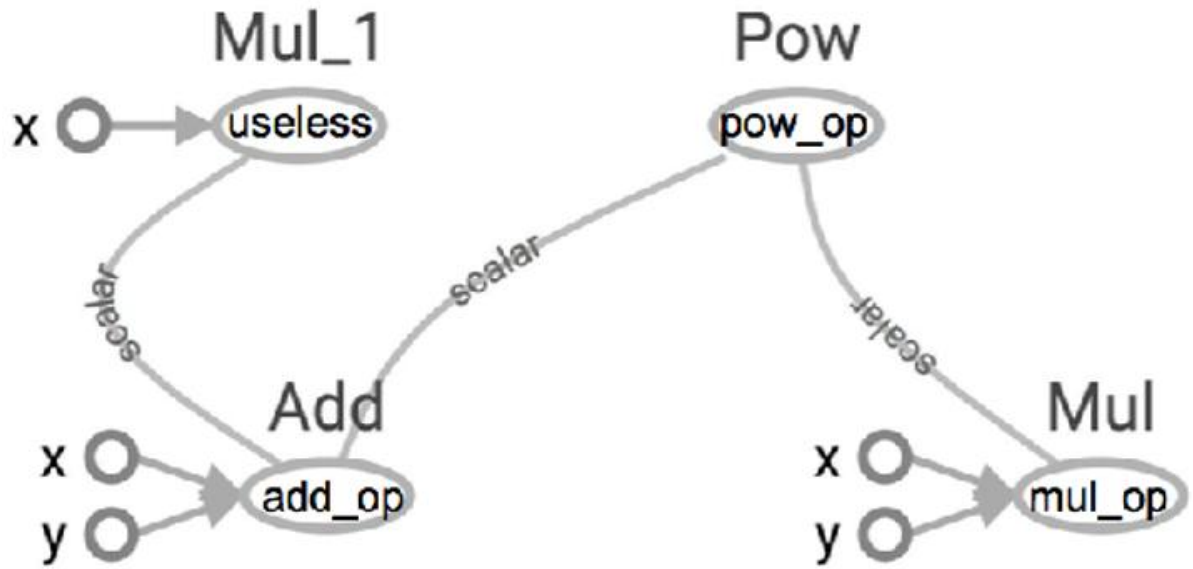
```
import tensor flow as tf
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
    z = sess.run(pow_op)
print z     >> output 15625
```



Think: will session also compute the value of useless?
（思考：会话也会计算useless的值吗）

More examples (更多示例)

```
import tensor flow as tf
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
    z, w = sess.run([pow_op, useless])
print z, w     >> output 15625, 10
```



API: tf.Session.run(**fetches**, feed_dict=None, options=None, run_metadata=None)
Pass all the variables you want to evaluate to a list in **fetches**
(将你想要计算的所有变量都传递给**fetches**中的列表)

Run session with specific device:
(在指定的设备中运行会话)

```
import tensor flow as tf
# build computational graph.
# 创建计算图
with tf.device("/gpu:0"):
    a = tf.constant([1.0, 2.0, 3.0, 4.0], shape=(2, 2), name="a")
    b = tf.constant([2.0, 4.0, 6.0, 8.0], shape=(2, 2), name="b")
    c = tf.matmul(a, b)

# build session, set log_device_placement=True
#创建会话，设置log_device_placement=True
with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)):
    print sess.run(c)
```

Why computational graph?
(为什么用计算图？)

- Save computations (only evaluates subgraphs which lead to the values you're interested in)
  (保存计算结果（只计算与你感兴趣的值 相关 的子图）)
- Facilitate distributed computation (model parallelization)
  (有助于分布式计算（模型并行化）)
- Directed acyclic graph is required in order to implement auto-differentiation
  (为了实现自动微分，需要有向非循环图)

TensorBoard for visualization (TensorBoard的可视化)

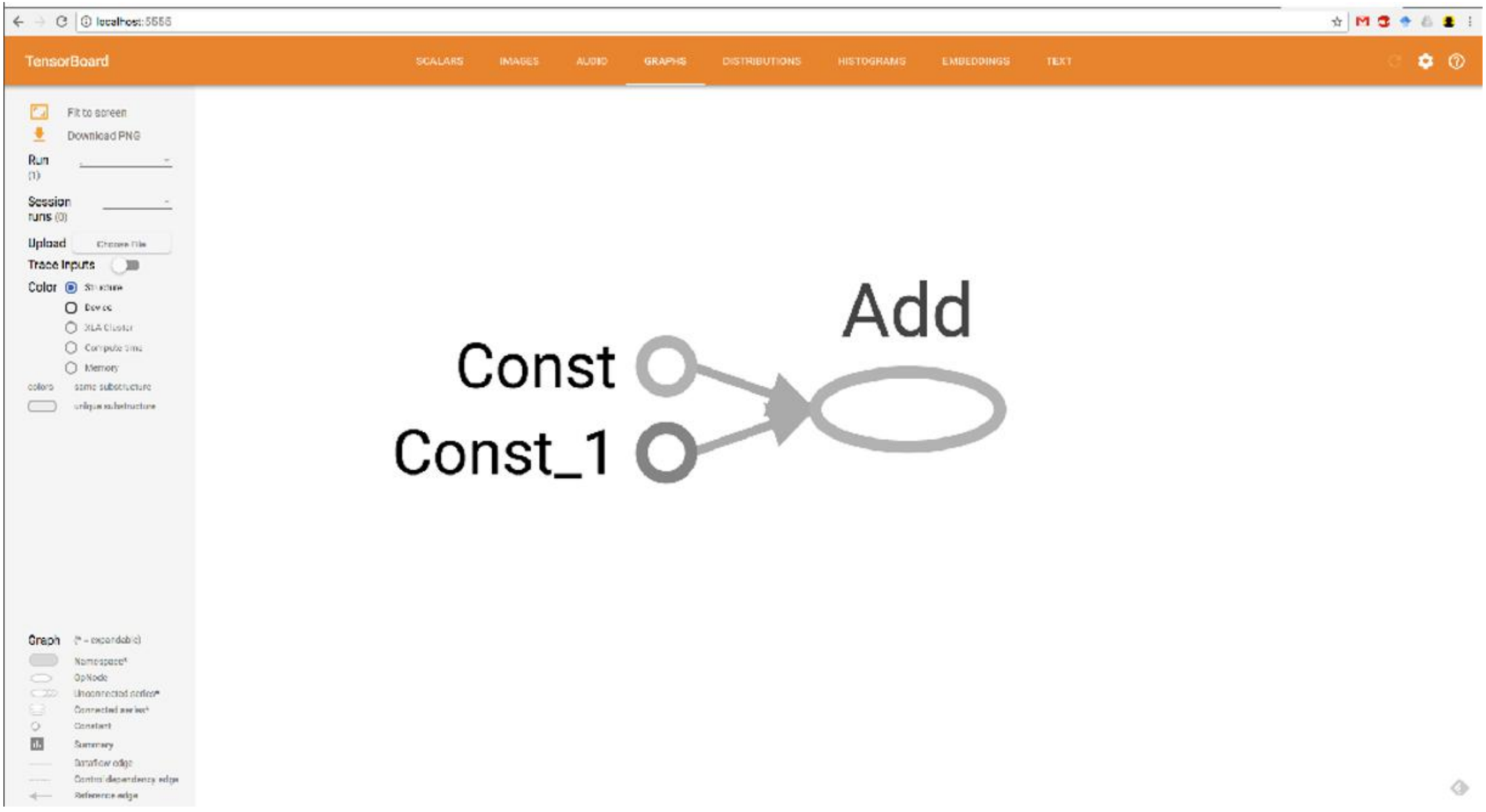
```
import tensor flow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)
with tf.Session() as sess:
    # add this line to use TensorBoard
    #加这一行是为了使用TensorBoard
    writer = tf.summary.FileWriter("./graphs", sess.graph)
    print sess.run(x)
writer.close()
```

Create the summary writer after graph definition but before running the session
(在图定义之后，运行会话之前，创建摘要写句柄)

TensorBoard for visualization (TensorBoard的可视化)

Open terminal, run（打开终端，运行）：

```
$ python [thisprogram].py
$ tensor board —logdir=“./graphs” —port 5555
```

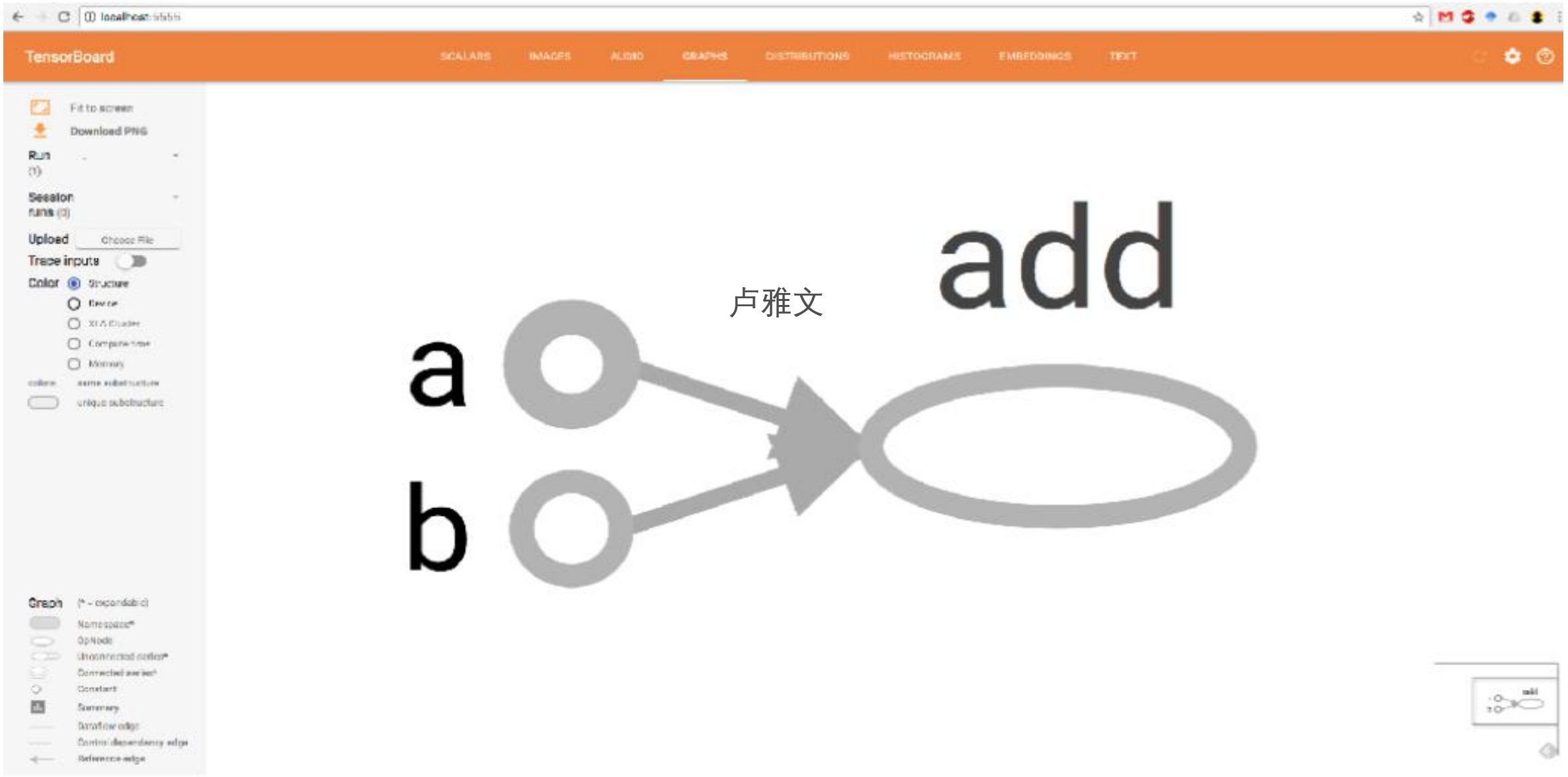


TensorBoard for visualization (TensorBoard的可视化)

We can name the variables, as well as the operators(可以对变量和操作符进行命名):

```
import tensor flow as tf
a = tf.constant(2, name="a")
b = tf.constant(3, name="b")
x = tf.add(a, b, name="add")
with tf.Session() as sess:
    # add this line to use TensorBoard
    #加这一行是为了使用TensorBoard
    writer = tf.summary.FileWriter("./graphs", sess.graph)
    print sess.run(x)
writer.close()
```

TensorFlow



Constants (常量)

tf.constant(value, dtype=None, shape=None, name="Const", verify_shape=False)

Very similar to that of Numpy （与Numpy的用法非常类似）

- tf.zeros
- tf.zeros_like
- tf.ones
- tf.ones_like
- tf.fill
- tf.constant
- tf.linspace
- tf.range

.....

https://www.tensorflow.org/api_guides/python/constant_op

Random variables (随机变量)

- tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)
- tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)
- tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None, name=None)
- tf.random_shuffle(value, seed=None, name=None)
- tf.random_crop(value, size, seed=None, name=None)
- tf.multinomial(logits, num_samples, seed=None, name=None)
- tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)

- Set random seed: tf.set_random_seed(seed)

https://www.tensorflow.org/api_guides/python/constant_op#Random_Tensors

TensorFlow

In TensorFlow we can perform all the usual matrix operations in Numpy
（在TensorFlow中可以用Numpy完成所有矩阵常规计算）

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ..
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

类别	例子
元素数学运算	加，减，乘，除，幂，对数，大于，小于，等于
数组运算	连接，分片，分割，常数，秩，形状，洗牌
矩阵运算	乘法，求逆，行列式
状态操作	变量，赋值（=），+=，
神经网络组件	Softmax, Sigmoid,ReLU,2维卷积，池化
检查点操作	存储，恢复
队列及同步操作	入队，出队，请求互斥量，释放互斥量
控制流操作	合并，转换，进入，离开，下一循环

Operations （运算）

```
import tensor flow as tf
a = tf.constant([3, 6])
b = tf.constant([2, 2])
tf.add(a, b) >> [5, 8]
tf.add_n([a, b, b]) >> [7, 10] = a + b + b
tf.multiply(a, b) >> [6, 12], elementwise multiplication
tf.matmul(a, b) >> Error, shape inconsistency for matrix multiplication
tf.matmul(tf.reshape(a, [1, 2]), tf.reshape(b, [2, 1])) >> 18
tf.dvi(a, b) >> [1, 3], elementwise division tf.mod(a, b) >> [1, 0], elementwise modulus
```

More math operations at: （数学运算相关内容详见）

https://www.tensorflow.org/api_guides/python/math_ops

TensorFlow data types （TensorFlow数据类型）

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.

详见

https://www.tensorflow.org/programmers_guide/dims_types#data_types

TensorFlow

Variables (tf.Variable is a class, tf. constant is an op)
变量（ tf.Variable 是一个类, tf. constant 是一种运算）
Constants are stored in graph definition, but variables are not!
常量存于图的定义中，但变量不是！

```
# create variable with a scalar value（创建一个标量变量）
a = tf.Variable(2, name="scalar")
# create variable with a vector value（创建一个向量变量）
b = tf.Variable([2, 3], name="vector")
# create variable with a matrix value（创建一个矩阵变量）
c = tf.Variable([[1, 2], [3, 4]], name="matrix")
# create variable with zeros （创建零值变量）
W = tf.Variable(tf.zeros([784, 10]))
```

https://www.tensorflow.org/programmers_guide/variables

TensorFlow

Variables contain operations:
变量相关操作

```
x = tf.Variable(...)
x.initializer # init op, 初始化
x.value() # read op 读
x.assign(...) # write op 写
x.assign_add(...) # and more更多内容详见
```

https://www.tensorflow.org/programmers_guide/variables

TensorFlow

Variables contain operations:
变量相关操作

```
x = tf.Variable(...)
x.initializer # init op, 初始化
x.value() # read op 读
x.assign(...) # write op 写
x.assign_add(...) # and more更多内容详见
```

https://www.tensorflow.org/programmers_guide/variables

Variables should be initialized before running
（变量运行前必须初始化）

```
# The easiest way to initialize all the variables -变量初始化最简单的方法
init = tf.global_variables_initializer()
with tf.Session() as sess:
   sess.run(init)

# Initialize a subset of variables - 初始化变量的子集
init_ab = tf.variables_initializer([a, b], name="init_ab")
with tf.Session() as sess:
    sess.run(init_ab)

# Initialize a single variable – 单个变量初始化
W = tf.Variable(tf.zeros([784, 10]))
with tf.Session() as sess:
sess.run(W.initializer)
```

Print the values of variables
（打印变量的值）

```
W = tf.Variable(tf.random_normal([784, 10]))
with tf.Session() as less:
    sess.run(W.initializer)
    print W
    print W.eval()
```

>> <tf.Variable 'Variable_1:0', shape=(784, 10), dtype=float32) >> [[-0.4778471, -1.3822577, ...]]

TensorFlow

Placeholders
（占位符）

Symbolic variables that do not take actual value when defined
（符号变量定义时并不需要赋值）

Can be filled with actual values when executed, （可以在执行时才赋值）
Examples: （例如）

$$f(x, y) = 2x + y$$

The x, y in the above function are placeholders — they don't have specific values when defined
（上式中x,y只是占位符-在定义时没有赋值）

However we can evaluate f(x, y) by giving them specific values
（但是他们赋值后就可以计算f(x,y)了）

Placeholders （占位符）

tf.placeholder(dtype, shape=None, name=None)

```
# create a placeholder of float32, shape is 1x3
#（创建32位浮点型占位符，形状为1*3）
a = tf.placeholder(tf.float32, shape=[3])
# create a constant of type float32, shape is 1x3
#（创建32位浮点常量，形状为1*3）
b = tf.constant([5, 5, 5], tf.float32)
c= a+ b
with tf.Session() as sess:
print sess.run(c)  >> Error? （出错了？）
```

Before running c, we need to provide a with specific values!
（执行C之前，首先需要赋值）

Placeholders （占位符）

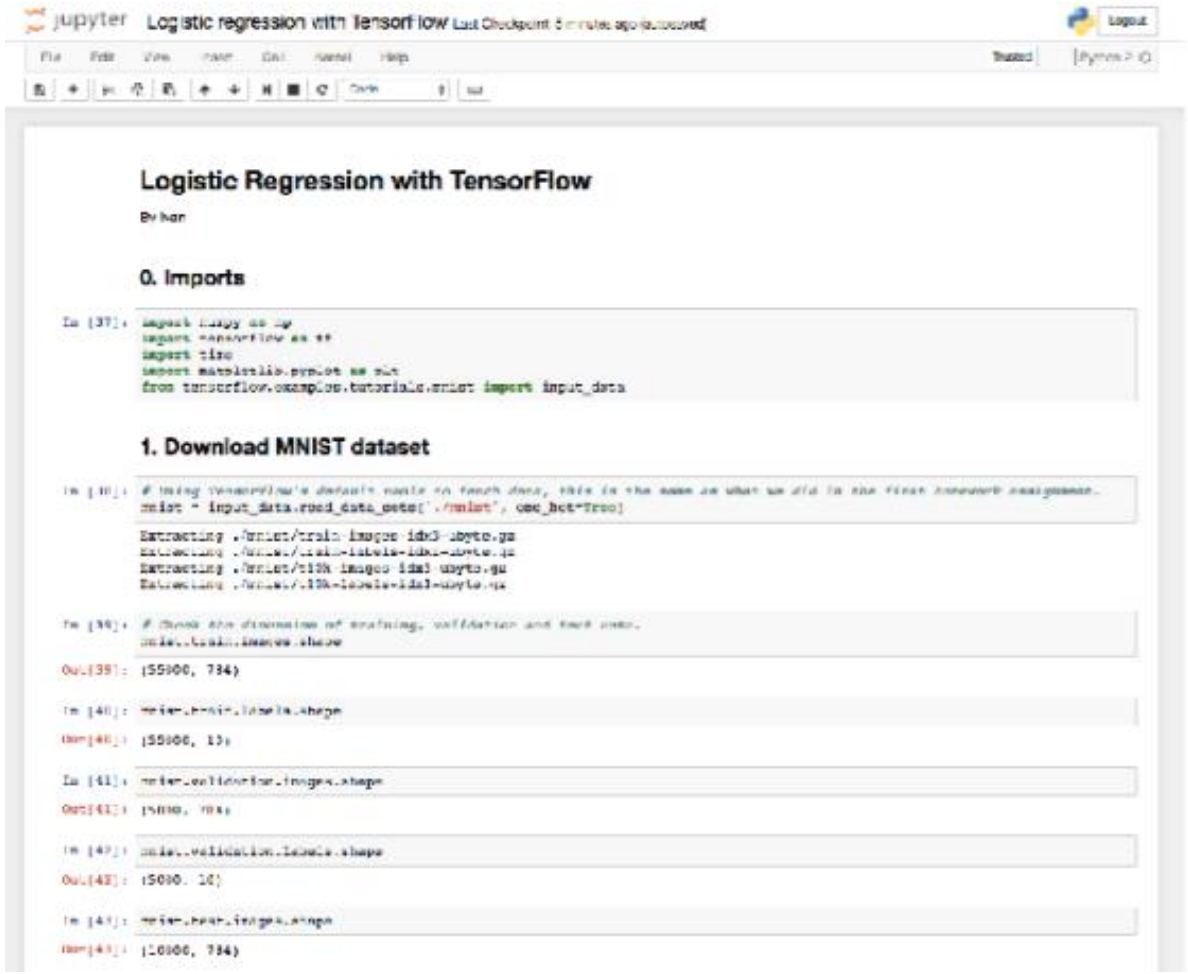
tf.placeholder(dtype, shape=None, name=None)

```
# create a placeholder of float32, shape is 1x3
#创建32位浮点占位符，形状为1*3

a = tf.placeholder(tf.float32, shape=[3])
# create a constant of type float32, shape is 1x3
#（创建32位浮点常量，形状为1*3）

b = tf.constant([5, 5, 5], tf.float32)
c= a + b
with tf.Session() as sess:
    # feed [1, 2, 3] to placeholder a via a dictionary-通过字典把[1,2,3]喂给占位符a
    print sess.run(c, {a: [1, 2, 3]})  >> the tensor a can be used as a key, [6, 7, 8]
```

Note: shape=None means the placeholder can have any shape

（注意： shape – None 意味着占位符可以是任何形状）

Logistic regression （逻辑回归）



家庭作业

Image classification on MNIST with a three layer MLP:

- Image size 28x28, size of MLP: 784-500-10, with ReLU as the nonlinear activation function
- batch size = 200
- learning rate = 0.1
- iterations = 20
- We provide an initial python script for you to work on

What you need to implement

- Use TensorFlow to build the computational graph
- Build necessary operator for SGD optimization
- Run the graph to train the model
- If implemented correctly, you should see a test set classification accuracy ~ 0.975 Running on GTX-1080, taking around ~80 seconds (graph compilation take time).

You need to install numpy and tensorflow

用三层MLP对MNIST的图像进行分类:

- 图像大小为28x28, MLP的各层: 784-500-10, 用ReLU 作为非线性激活函数
- 批处理数量 = 200
- 学习率 = 0.1
- 循环次数 = 20
- 可以用提供Python脚本为基础

你需要完成：

- 用Tensorflow创建计算图
- 为SGD优化创建必要的算子
- 运行图来训练模型
- 如果一切正常，在测试集上，分类的准确率约为： 0.975。在GTX-1080上耗时约80秒（计算图的编译需要花点时间）

你需要安装numpy和TensorFlow

作者与译者介绍

BIG DATA DIGEST
大数据文摘

稀牛学院
www.xiniuedu.com

作者

Ivan老师

前Google AI实验室工程师。清华大学本科，CMU PhD，在AAAI、IJCAI、AISTATS等顶级会议上发表过多篇论文。熟练机器学习、深度学习与统计、凸优化。

翻译团队成员

一天（组长）

通信行业老码农，产品经理，对机器学习和深度学习感兴趣，望广交同好，请联系QQ:343208108。

卢雅文

通信专业本科，信号图像处理方向新加坡南洋理工大学硕士在读。有医学图像处理，三维图像重建，图像分割、检测、分类和VR等项目研究经历，愿与志同道合好友一同学习，互相交流QQ:523565239。

郭浩闻

北京理工大学软件学院本科在读，备考雅思ing，对机器学习数据挖掘方向有浓烈兴趣，在实验室由学长带领已投洱海水质数据分析论文一篇。

翻译说明

Deep Learning领域最新突破常以英文发表和公布，所以英文课件的表述更精准。

《深度学习与计算机视觉》课程的学员以Case的形式，对照讲师发布的原英文课件，进行了中文的翻译和校对，以降低英语基础较弱学员的学习难度。

翻译团队为精准、一致的翻译进行了多种努力，但仍然难免疏漏。欢迎学员提出修改意见和建议！

最后，英语不难哦！希望大家不要有畏难情绪~

版权原因，讲师课件(英文版和中文翻译版)不宜公开传播，课程相关度非常高的笔记也不宜公开发布。

课程合作请联系
info@bigdatadigest.cn

BIG DATA DIGEST
大数据文摘

X

稀牛学院

扫码获取最新数据科学资讯