Tensorflow 工具介绍
Introduction to Tensorflow

Ivan

Aug. 27th, 2017

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

Outline

- Overview of TensorFlow
- Graphs and Sessions
- TensorBoard Introduction
- Constants and Placeholders
- 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

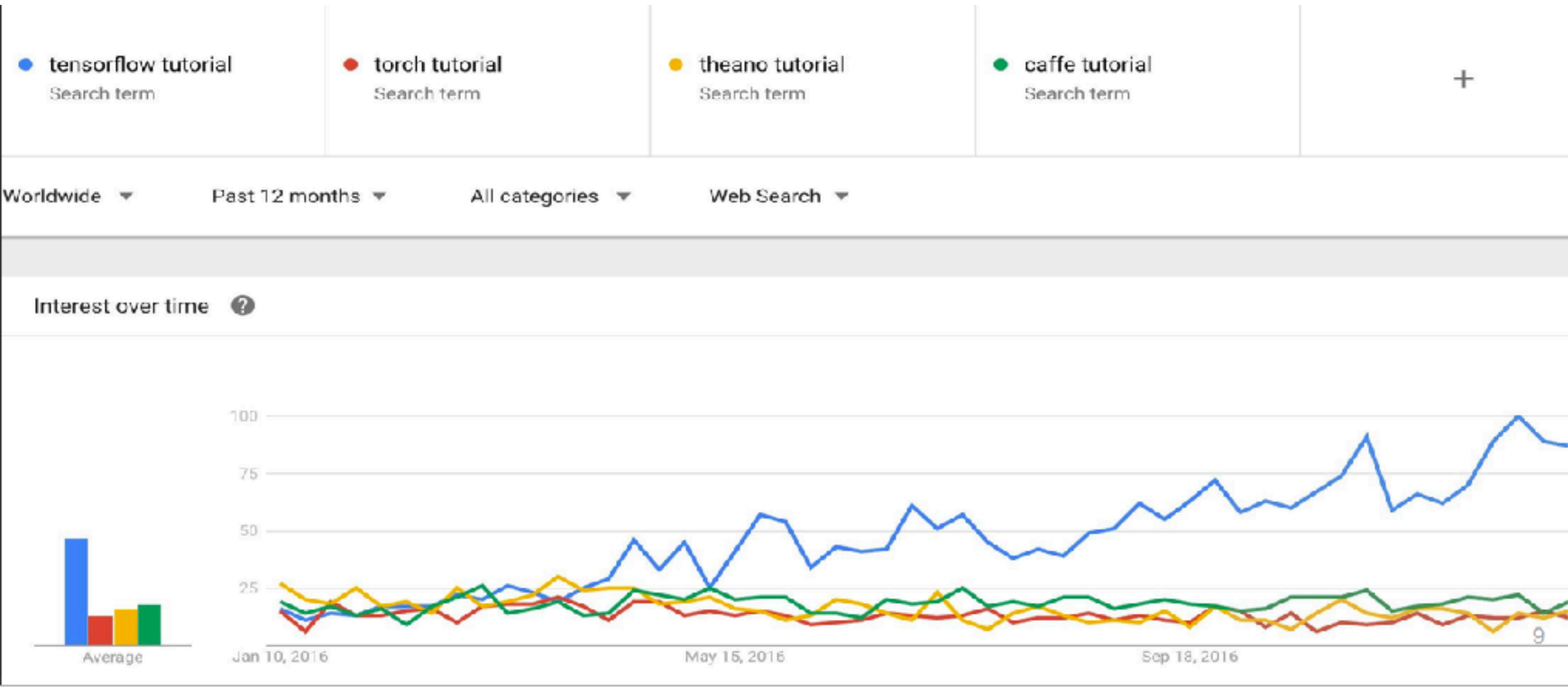
TensorFlow

- Besides TensorFlow

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

TensorFlow

- Why 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

TensorFlow

- Companies that use TensorFlow

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

TensorFlow

- 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)

Introduction

```
import tensorflow as tf
```

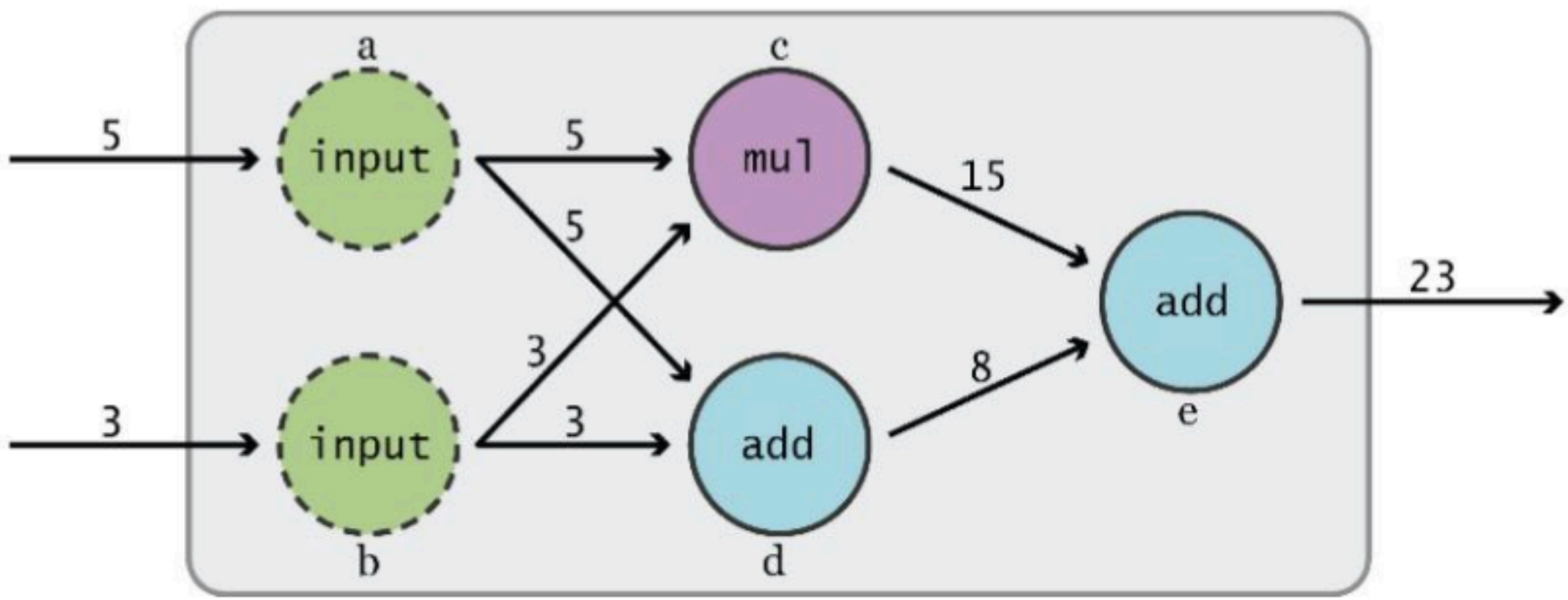
Introduction

- 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**

Graphs and Sessions

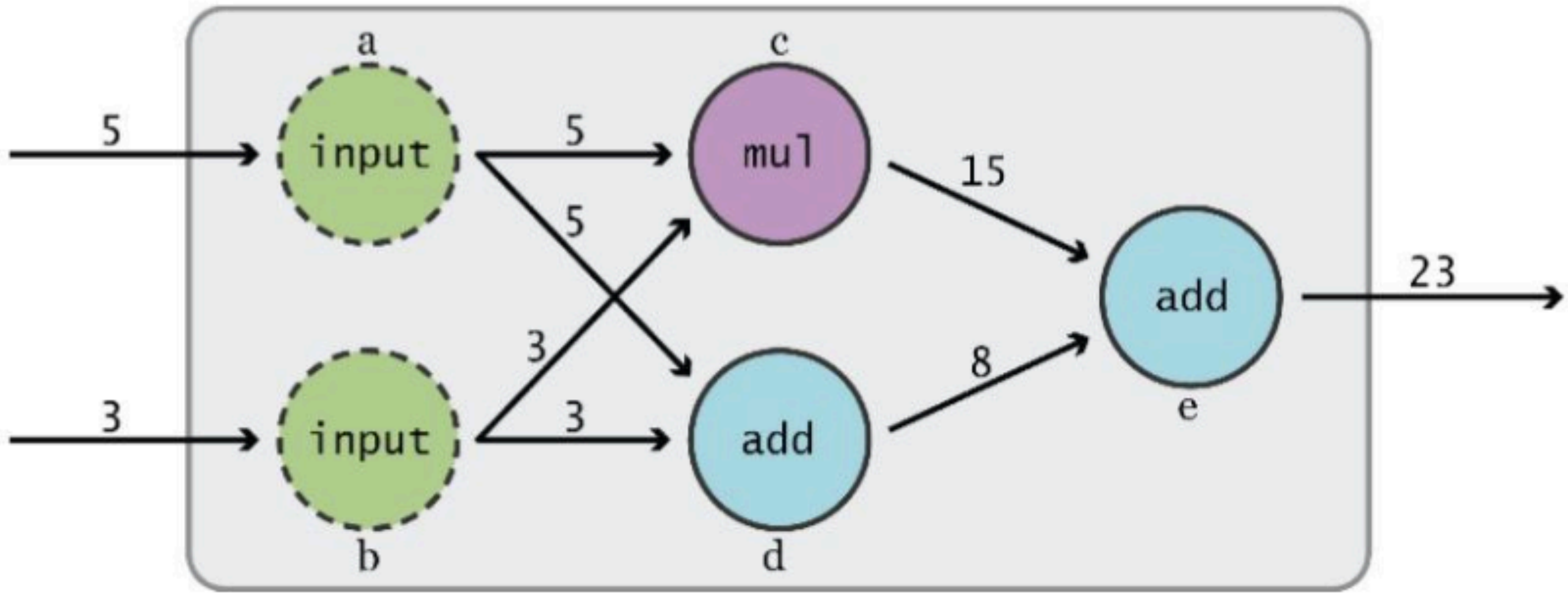
- Data flow graphs



Defining computation $\neq$ Execution of Computation

Graphs and Sessions

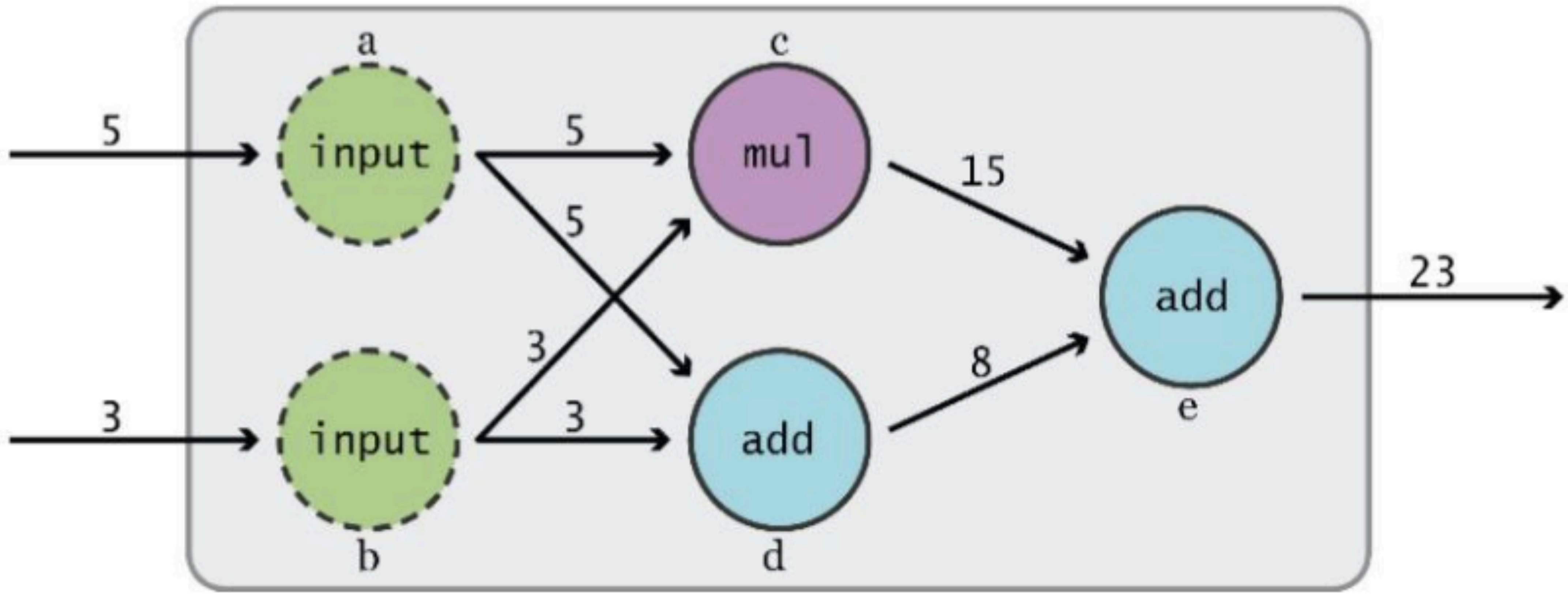
- Data flow graphs



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

Graphs and Sessions

- Data flow graphs



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

Data are transmitted and transformed by operators (op) in the computational graph

TensorFlow

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

TensorFlow

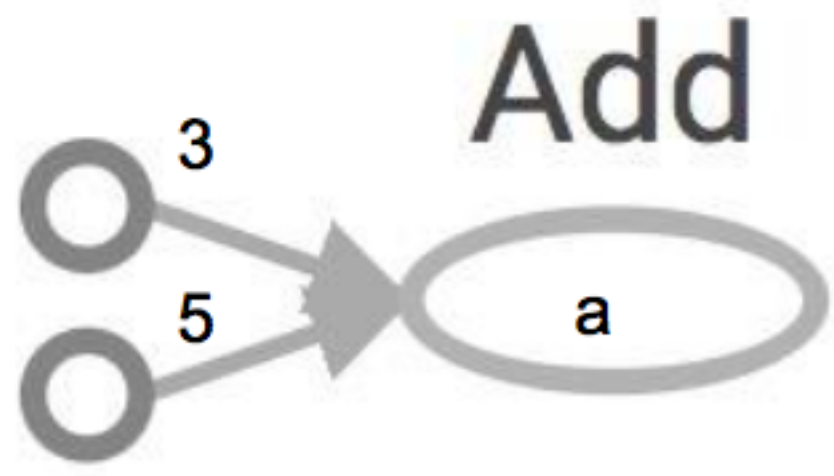
- Numpy vs TensorFlow

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, :]

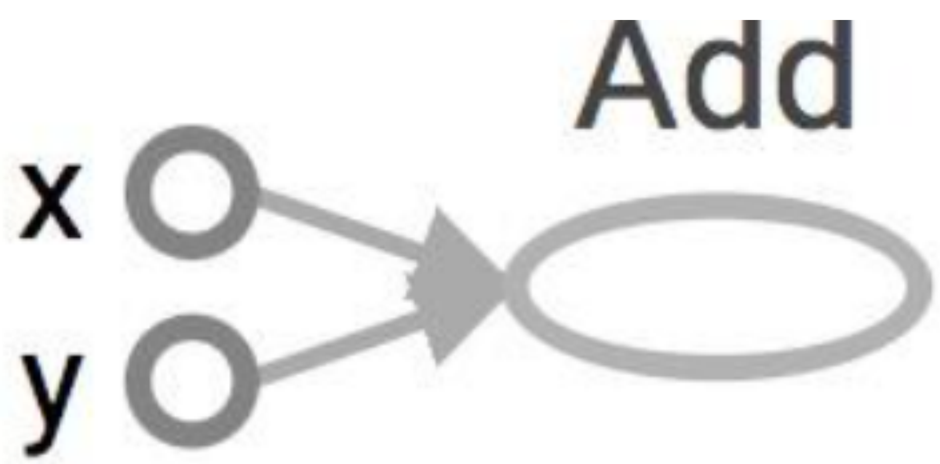
TensorFlow

- Data flow graphs

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



What is x, y?
x = 3, y = 5;
TF will automatically name variables



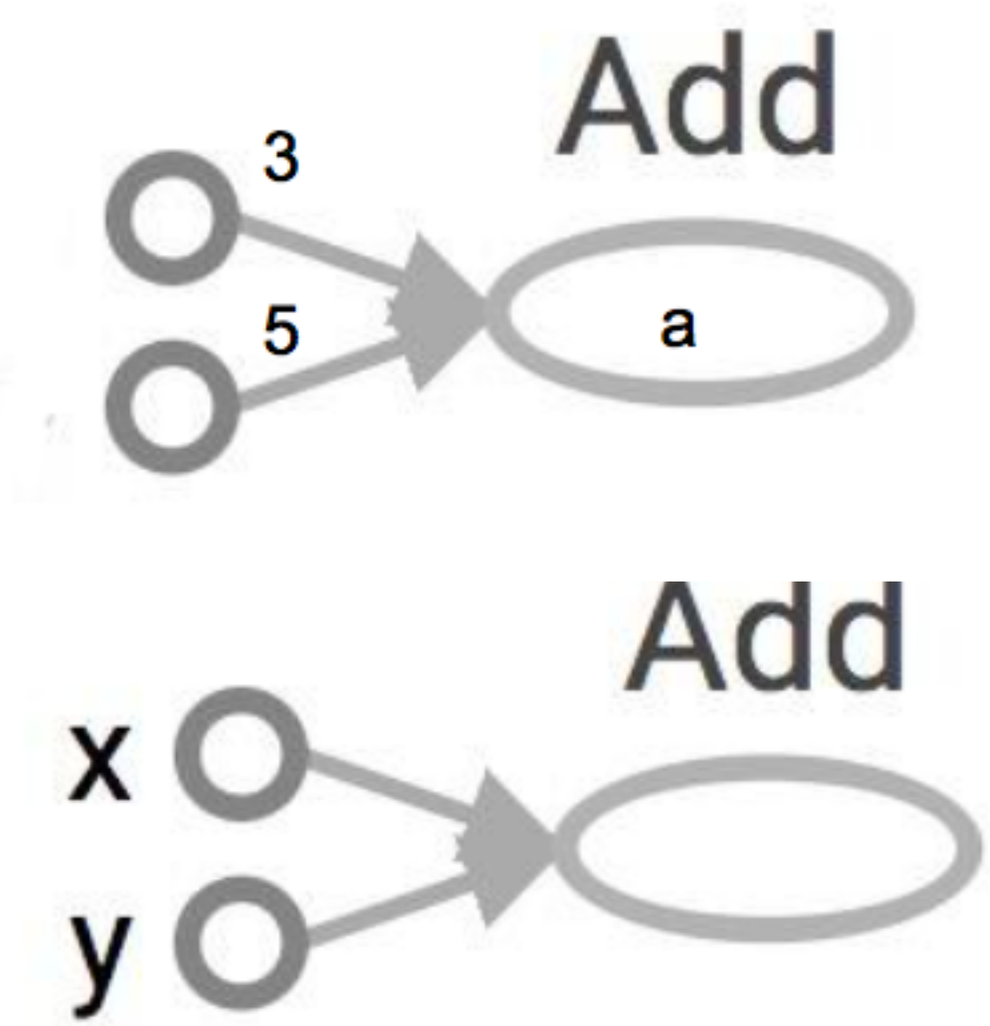
Visualization from TensorBoard

Nodes: operators, variables, or constants
Edges: tensors

TensorFlow

- Data flow graphs

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



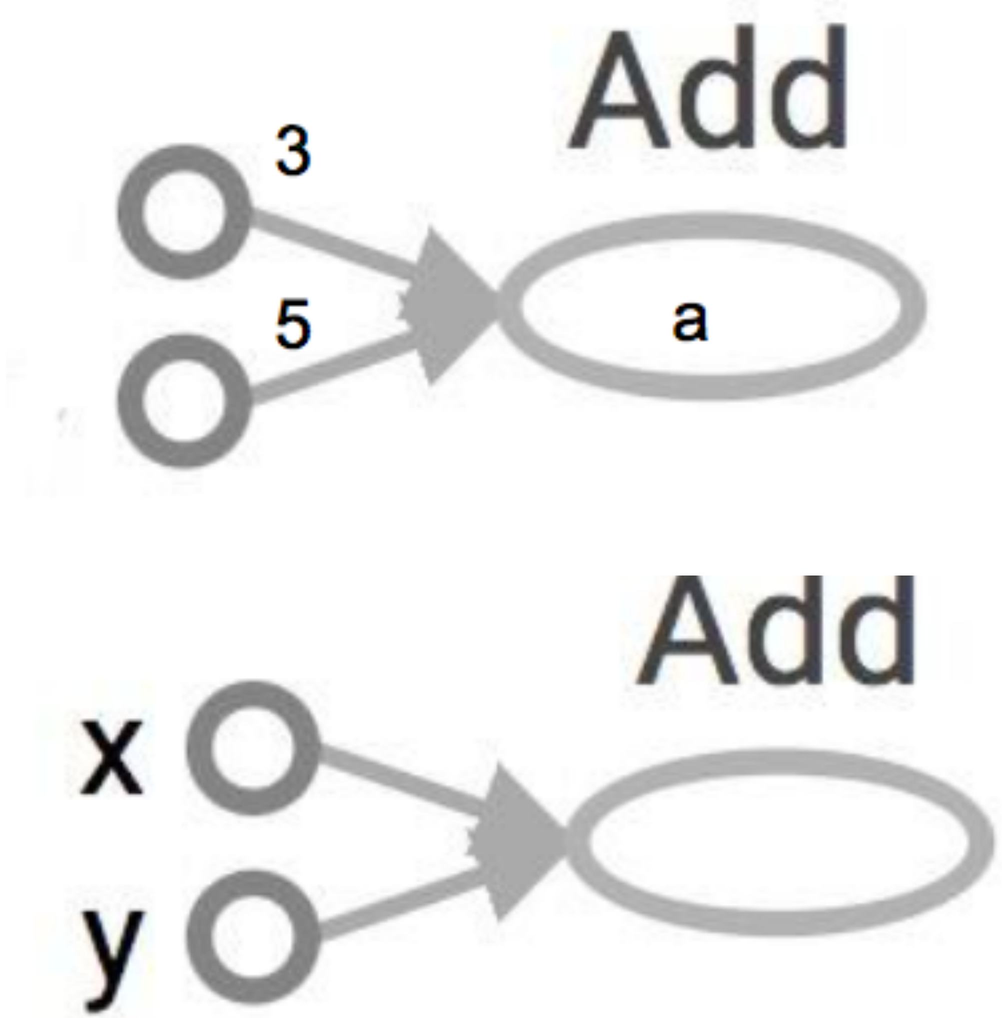
Visualization from TensorBoard

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

TensorFlow

- Data flow graphs

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



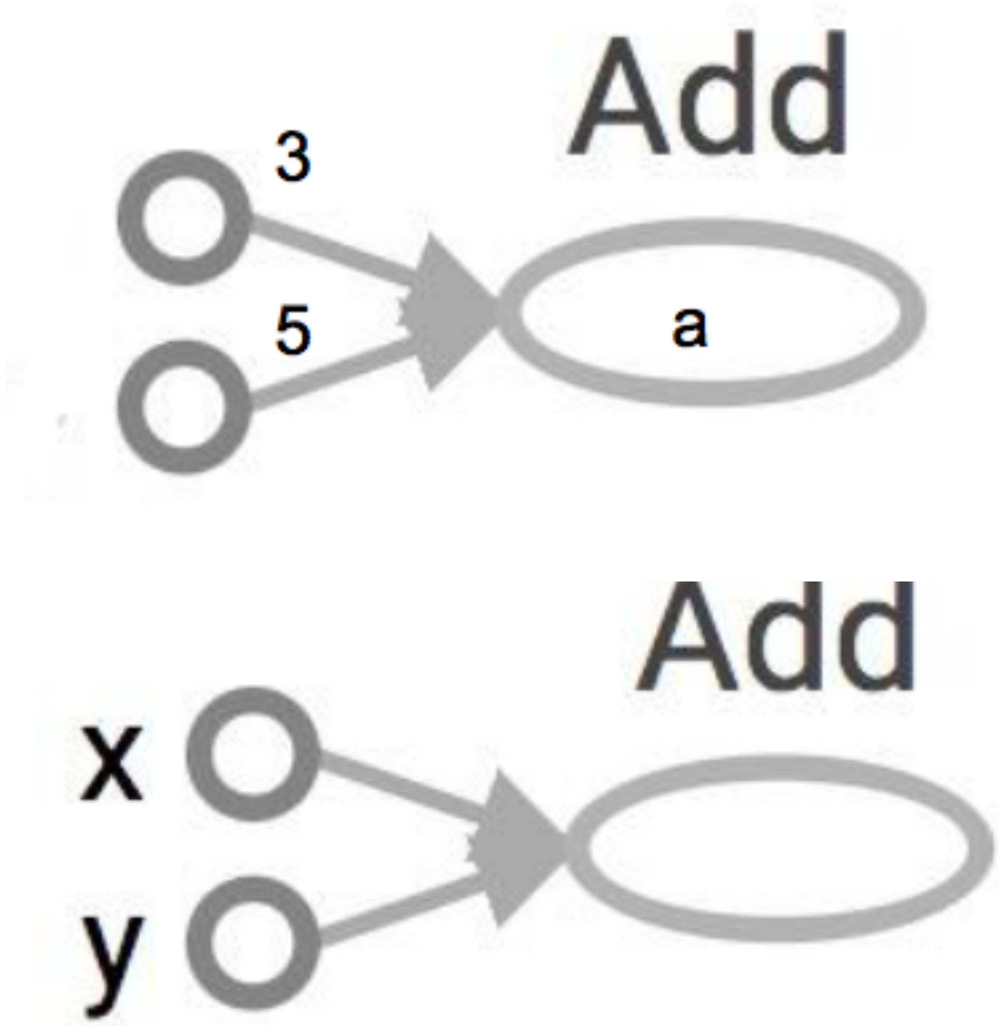
Visualization from TensorBoard

>> Tensor(“Add:0”, shape=(), dtype=int32)
(Not 8)
Symbolic variable! How to get the value of a?

TensorFlow

- Data flow graphs

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



Visualization from TensorBoard

We need to create a **session** in order to get the value of a

TensorFlow

- Create a session

```
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

TensorFlow

- Create a session (Recommended practice)

```
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

TensorFlow

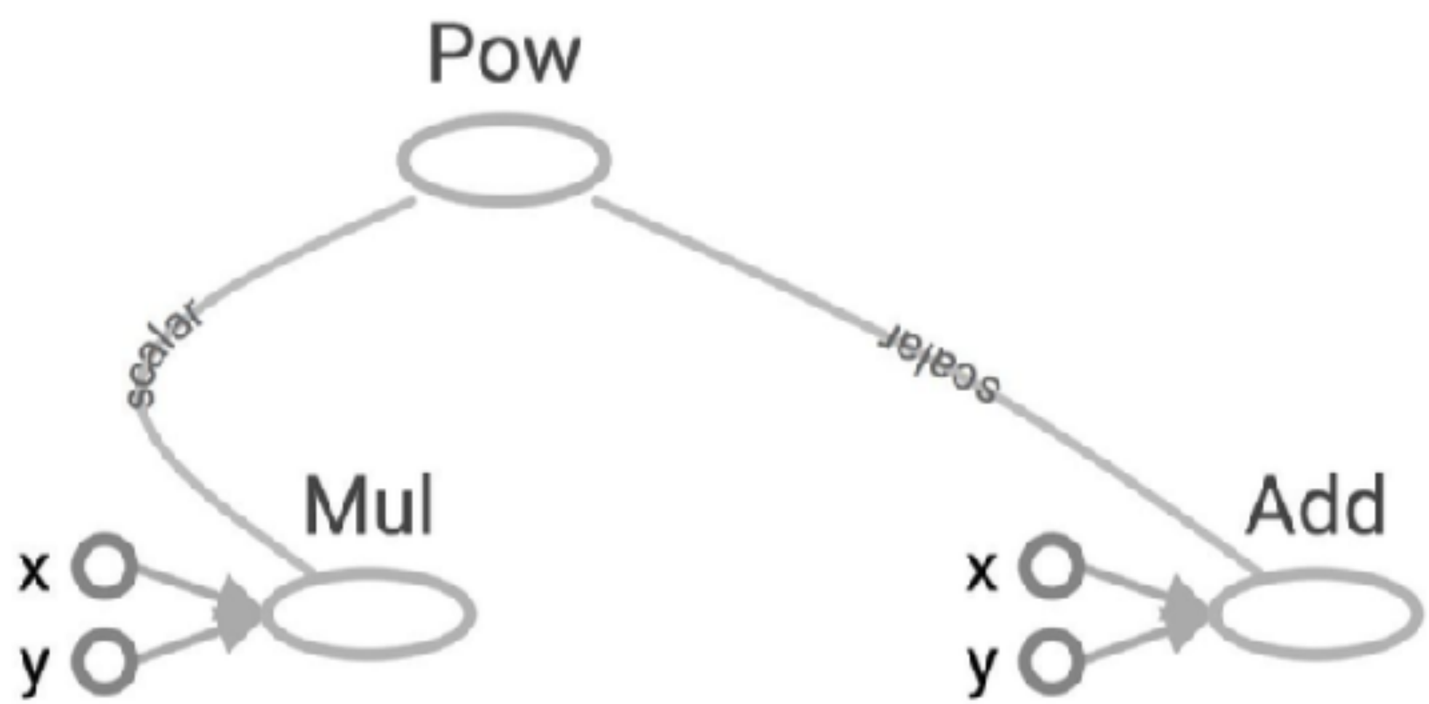
- Summary

A Session object encapsulates the running environment such that operators are executed and tensors are evaluated

TensorFlow

- More examples

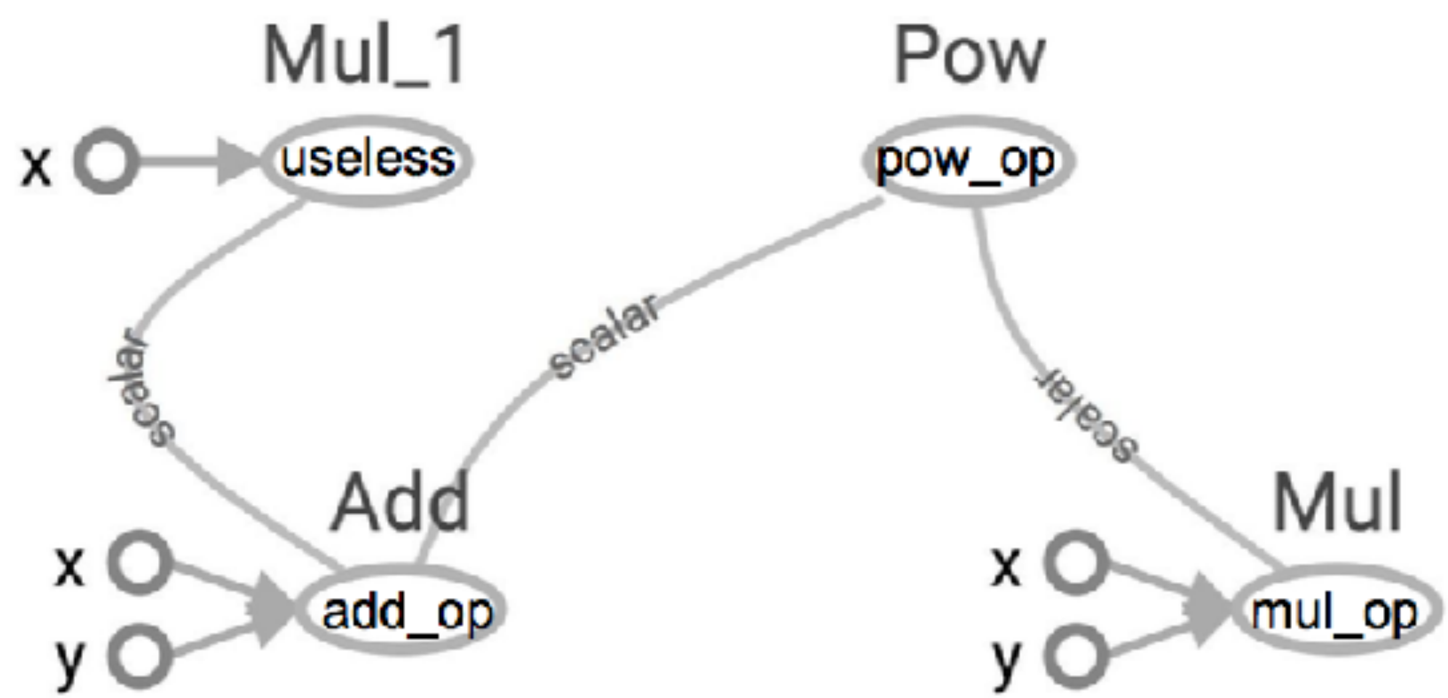
```
import tensor flow as tf
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as sess:
    op3 = sess.run(op3)
print op3     >> output 7776
```



TensorFlow

- More examples

```
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
```

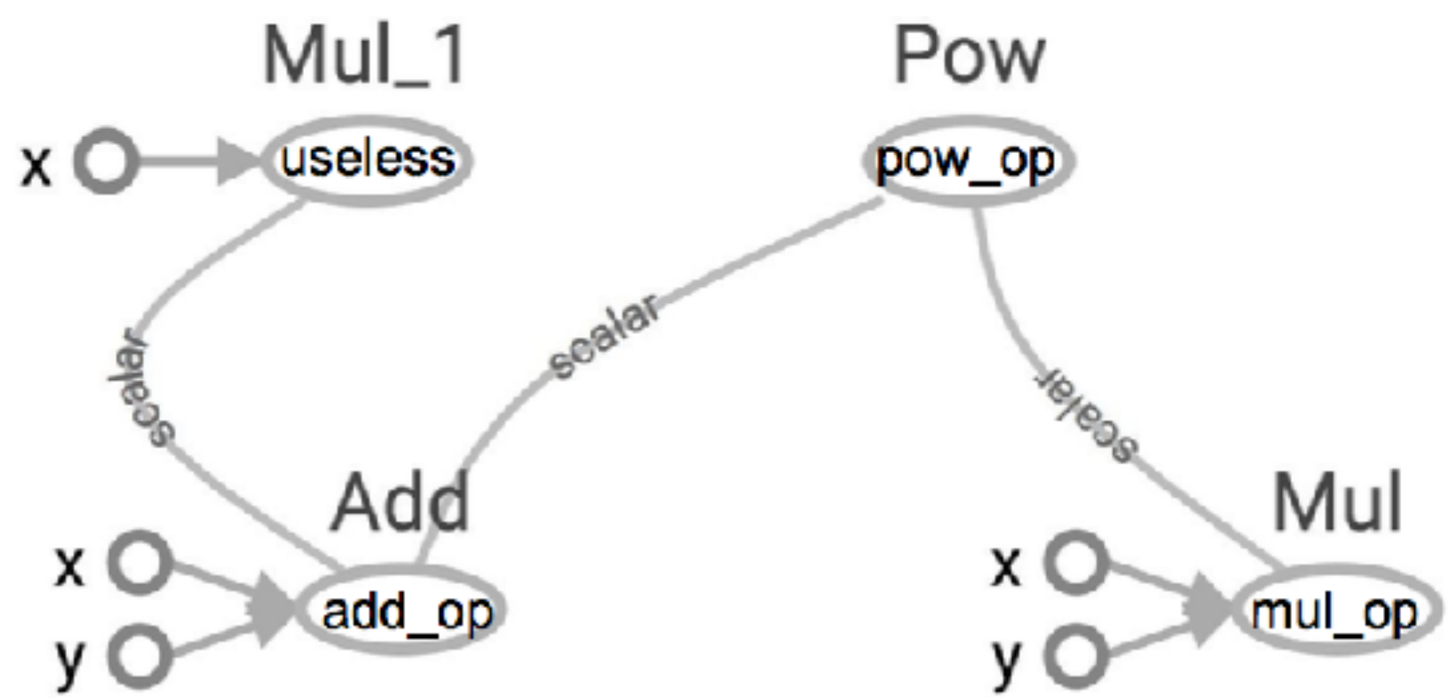


Think: will session also compute the value of useless?

TensorFlow

- More examples

```
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
```

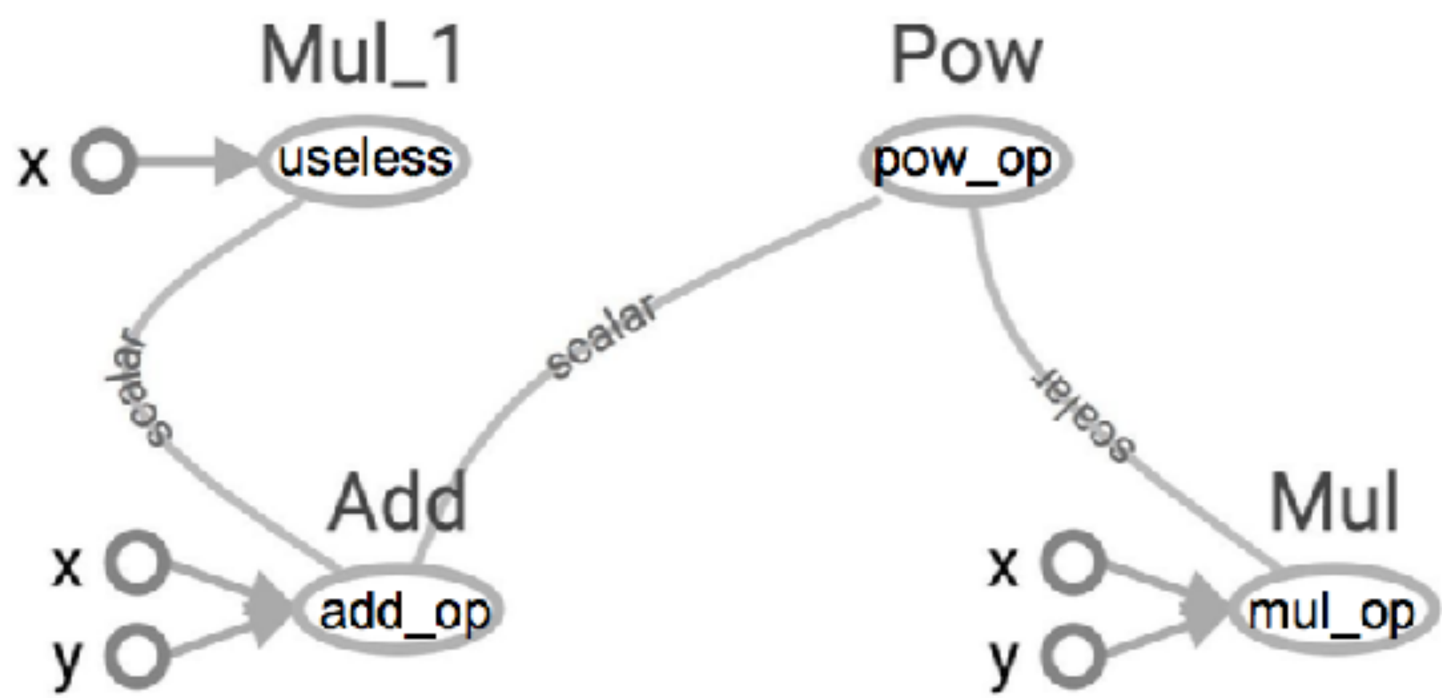


No: print useless >> Tensor(“Mul_3:0”, shape=(), dtype=int32)

TensorFlow

- More examples

```
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

TensorFlow

- Run session with specific device:

```
import tensor flow as tf
# build computational graph.
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
with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)):
  print sess.run(c)
```

TensorFlow

- Why computational graph?

- Save computations (only evaluates subgraphs which lead to the values you're interested in)
- Facilitate distributed computation (model parallelization)
- Directed acyclic graph is required in order to implement auto-differentiation

TensorFlow

- TensorBoard for visualization

```
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
   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

TensorFlow

- TensorBoard for visualization

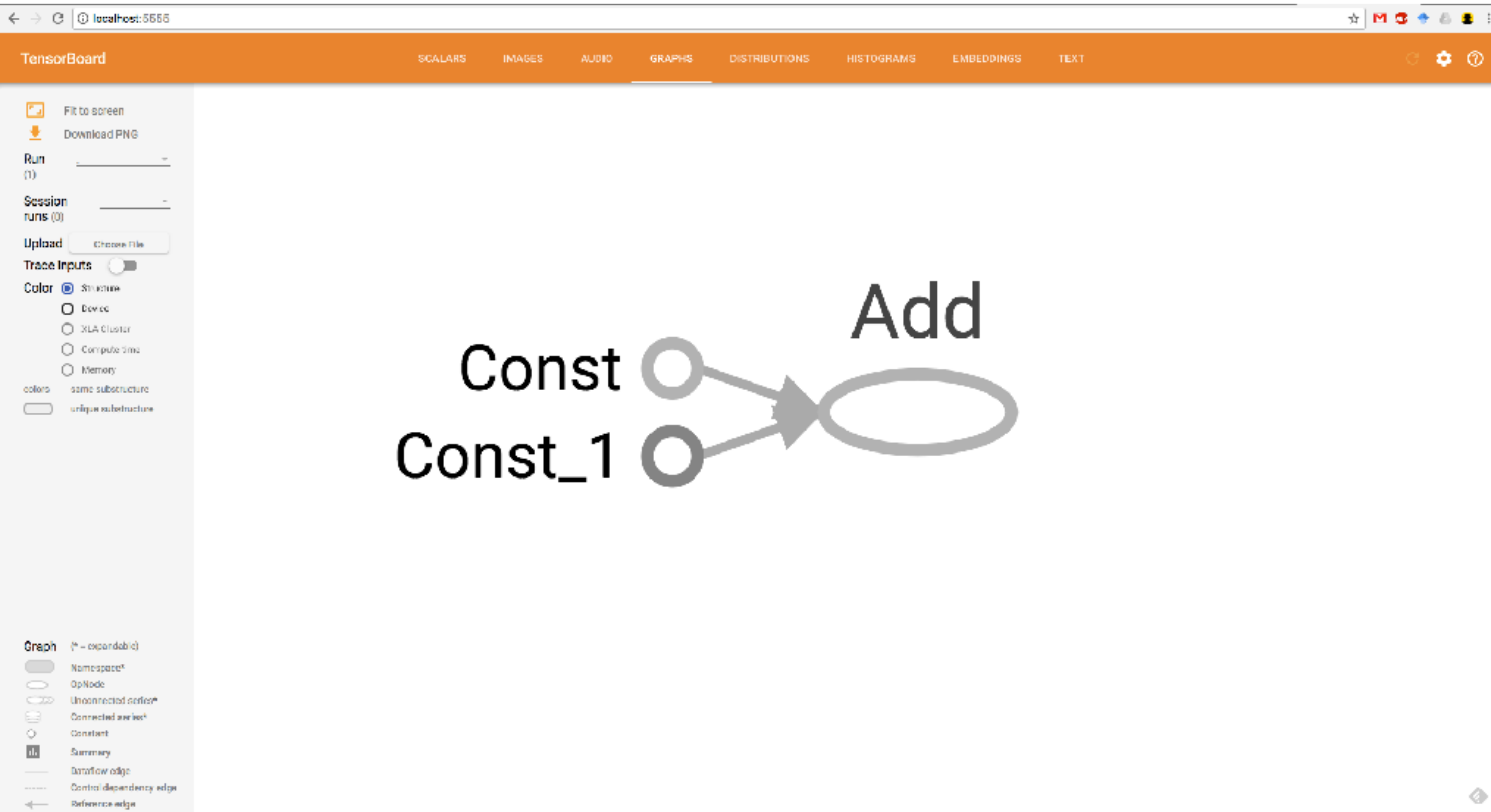
Open terminal, run:

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

BIG DATA DIGEST 大数据文摘 稀牛学院

xiniuedu.com 网址

TensorFlow



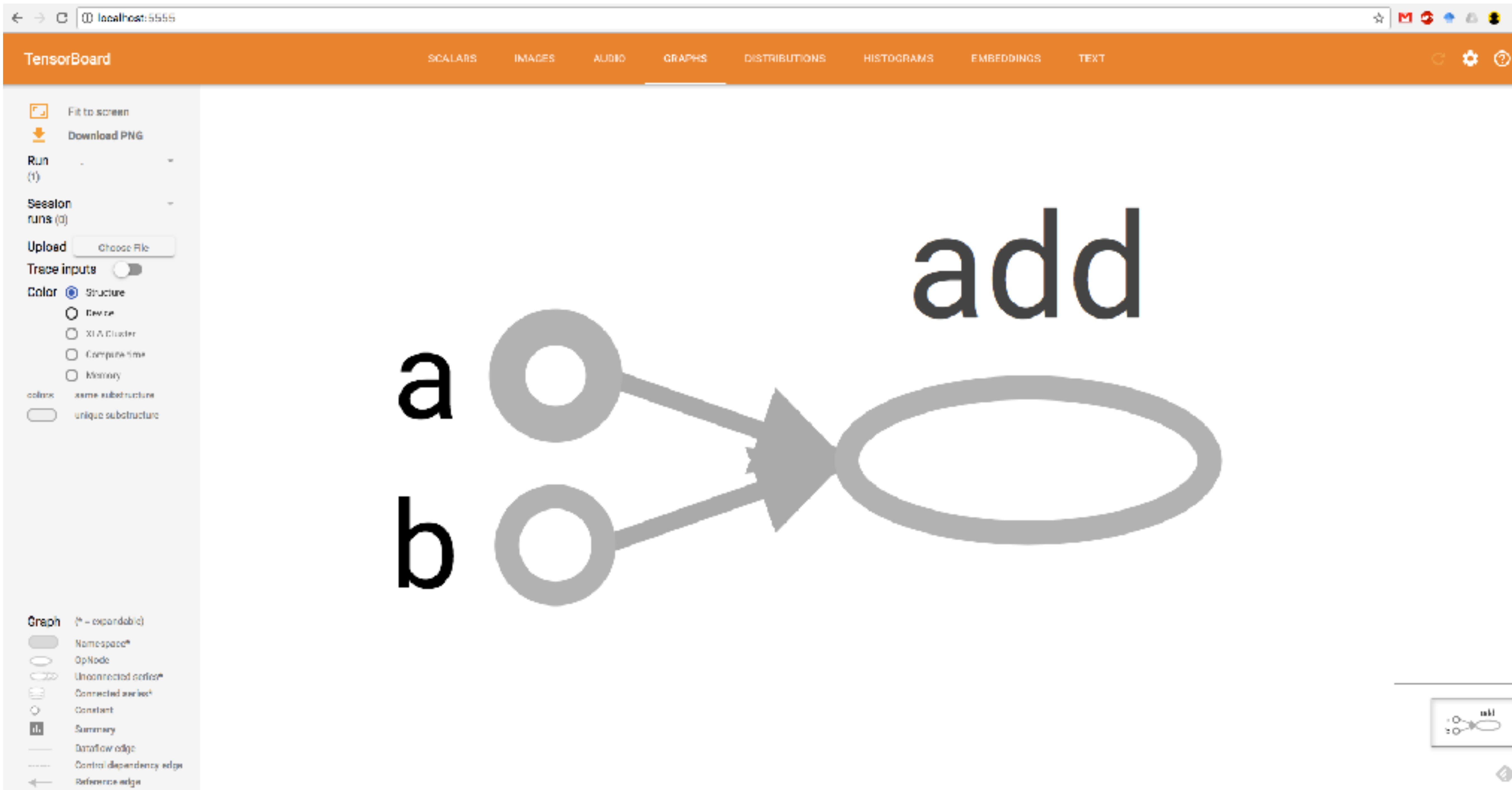
TensorFlow

- TensorBoard for visualization

We can name the variables, as well as the operators:

```
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
   writer = tf.summary.FileWriter(“./graphs”, sess.graph)
   print sess.run(x)
writer.close()
```

TensorFlow


localhost:5555
TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Fit to screen
Download PNG
Run
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
unique substructure
Graph
Namespace*
OpNode
Summary
add
a
b

TensorFlow

- Constants

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

- Very similar to that of Numpy
- 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

TensorFlow

- Random variables
- 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

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

TensorFlow

- Operations

```
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:

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

TensorFlow

- TensorFlow data types

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.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

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

TensorFlow

- Variables (tf.Variable is a class, tf. constant is an op)
- Constants are stored in graph definition, but variables are not!

```
# create variable with a scalar value
a = tf.Variable(2, name=“scalar”)
# create variable with a vector value
b = tf.Variable([2, 3], name=“vector”)
# create variable with a matrix value
c = tf.Variable([[1, 2], [3, 4]], name=“matrix”)
# create variable with zeros
W = tf.Variable(tf.zeros([784, 10]))
```

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

TensorFlow

- Variables contain operations:

```
x = tf.Variable(…)

x.initializer # init op
x.value() # read op
x.assign(…) # write op
x.assign_add(…) # and more
```

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

TensorFlow

- Variables should be initialized before running

```
# The easiest way to initialize all the variables
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)

# Initialize a subset of variables
init_ab = tf.variables_initializer([a, b], name="init_ab")
with tf.Session() as sess:
    sess.run(init_ab)

# Initialize a single variable
W = tf.Variable(tf.zeros([784, 10]))
with tf.Session() as sess:
    sess.run(W.initializer)
```

TensorFlow

- Print the values of variables

```
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
- Symbolic variables that do not take actual value when defined
- Can be filled with actual values when executed
- Examples:

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

  - The x, y in the above function are placeholders — they don't have specific values when defined
  - However we can evaluate f(x, y) by giving them specific values

TensorFlow

- Placeholders
- **tf.placeholder(dtype, shape=None, name=None)**

```
# create a placeholder of float32, shape is 1x3
a = tf.placeholder(tf.float32, shape=[3])
# create a constant of type float32, shape is 1x3
b = tf.constant([5, 5, 5], tf.float32)
c = a + b

with tf.Session() as sess:
  print sess.run(c)           >> Error?
```

Before running c, we need to provide a with specific values!

TensorFlow

- Placeholders
- **tf.placeholder(dtype, shape=None, name=None)**

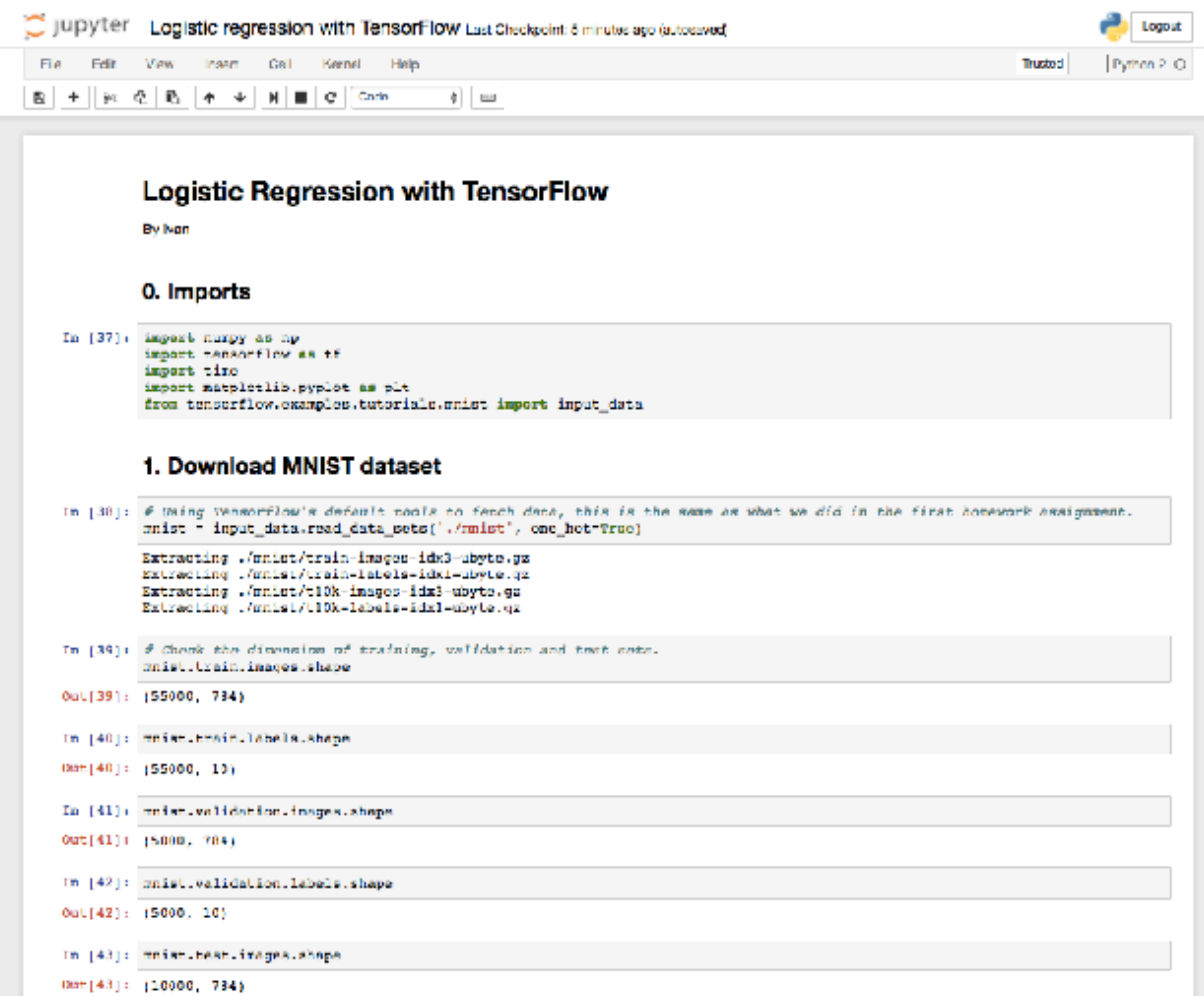
```
# create a placeholder of float32, shape is 1x3
a = tf.placeholder(tf.float32, shape=[3])
# create a constant of type float32, shape is 1x3
b = tf.constant([5, 5, 5], tf.float32)
c = a + b

with tf.Session() as sess:
  # feed [1, 2, 3] to placeholder a via a dictionary
  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

TensorFlow

- Logistic regression



jupyter Logistic regression with TensorFlow Last Checkpoint: 8 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 2

Logistic Regression with TensorFlow

By Ivan

0. Imports

```
In [37]: import numpy as np
import tensorflow as tf
import time
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
```

1. Download MNIST dataset

```
In [38]: # Using Tensorflow's default tools to fetch data, this is the same as what we did in the first homework assignment.
mnist = input_data.read_data_sets('./mnist', one_hot=True)

Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz

In [39]: # Check the dimension of training, validation and test sets.
mnist.train.images.shape
Out[39]: (55000, 784)

In [40]: mnist.train.labels.shape
Out[40]: (55000, 10)

In [41]: mnist.validation.images.shape
Out[41]: (5000, 784)

In [42]: mnist.validation.labels.shape
Out[42]: (5000, 10)

In [43]: mnist.test.images.shape
Out[43]: (10000, 784)
```

Homework

- 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

BIG DATA DIGEST 大数据文摘

稀牛学院

扫码获取最新数据科学资讯