MACHINE LEARNING

TensorFlow Tutorial

Contact Information

Instructor: Qi Hao

E-mail: hao.q@sustc.edu.cn

Office: Nanshan iPark A7 Room 906

Office Hours: M 2:00-4:00pm

Available other times by appointment or the open door policy

Office Phone: (0755) 8801-8537

QQ: 463715202 机器学习2018

Web:

http://hqlab.sustc.science/teaching/

What's TensorFlow™?

"Open source software library for
numerical computation using data flow graphs"

Launched Nov 2015

Why TensorFlow?

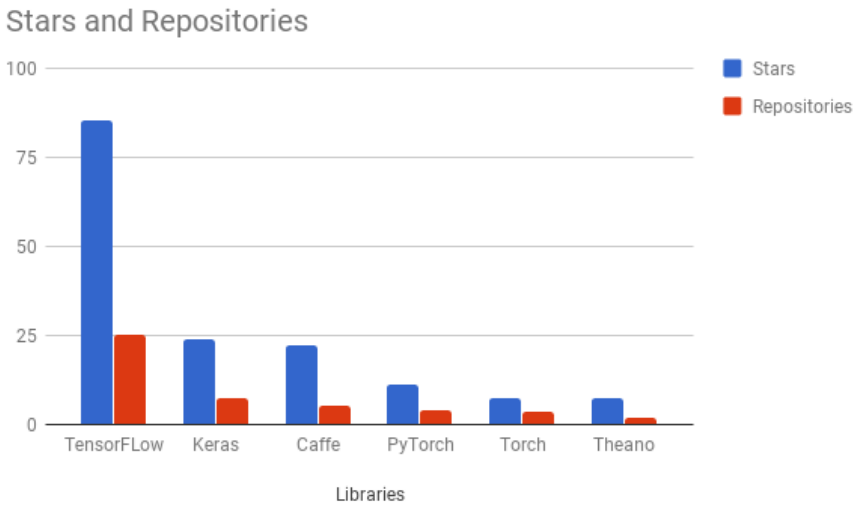
- Many machine learning libraries

Why TensorFlow?

- Flexibility + Scalability
  - TensorFlow
  - efficient, scalable, and maintainable
  - flexible operations
- Flexible enough less scalable
  - Chainer and PyTorch
- Scalable but less flexible
  - Caffe and MXNet

Why TensorFlow?

- Flexibility + Scalability
- Popularity



Why TensorFlow?

- High level APIs on top of TensorFlow
  - Keras [also for CNTK and Theano] , TFLearn, and Sonnet.

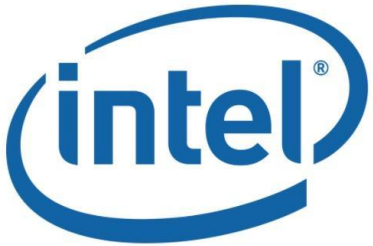
- Faster experimentation
  - a few lines of code
  - a sizeable number of users

- An extensive suite of functions and classes
  - define models from scratch
  - easy to plot network structure and results


Google

OpenAI

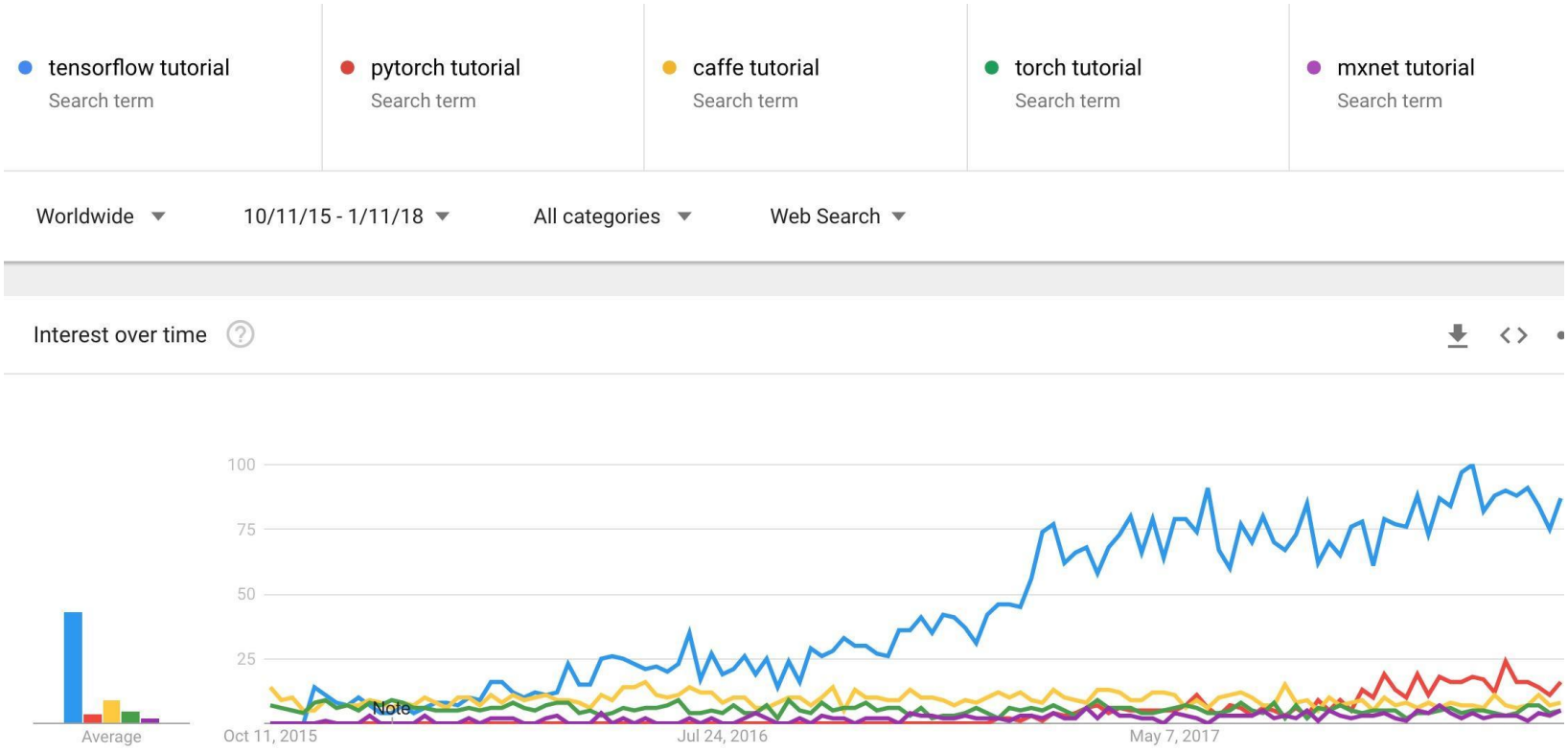
AIRBUS


Google DeepMind

intel

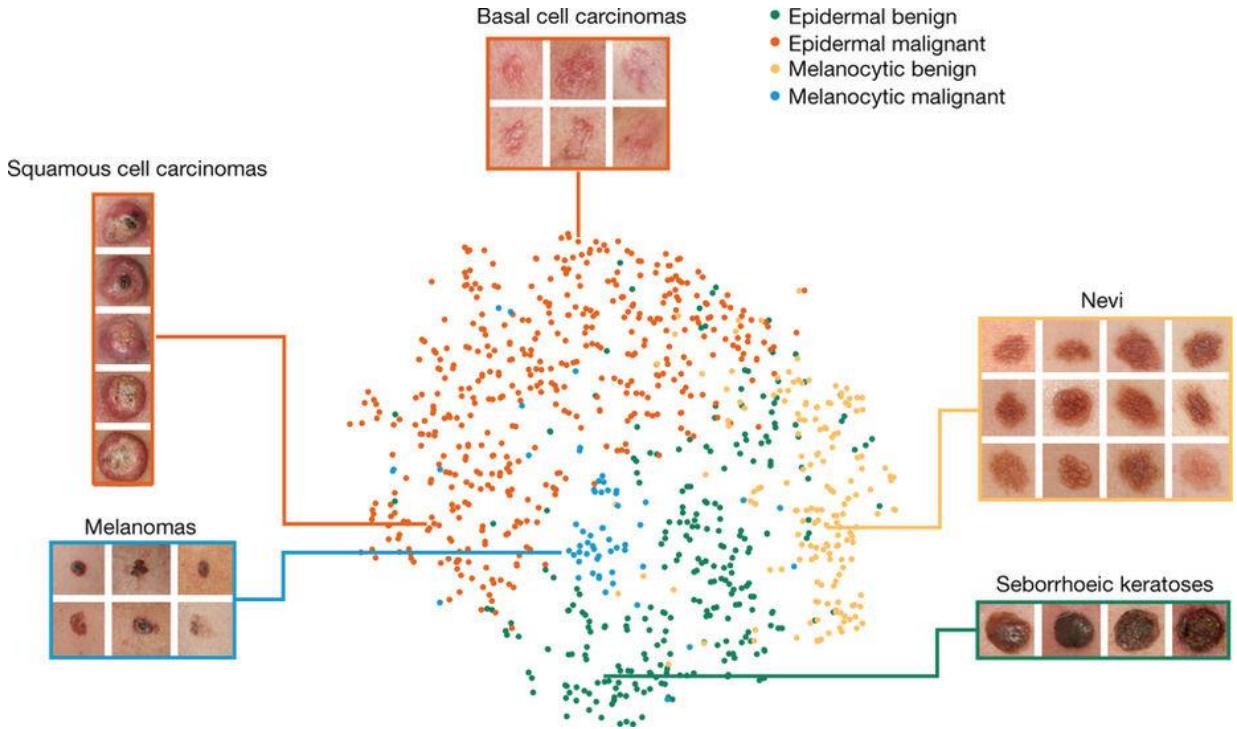
airbnb


NVIDIA

Demand for tutorials on TensorFlow



Some cool projects using TensorFlow



Dermatologist-level classification of skin cancer with deep neural networks (Esteva et al., Nature 2017)

Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016)
Tensorflow adaptation by Cameroon Smith (cysmith@github)

TensorFlow：the tool to build cool projects like those!

Summary: why we use tensorflow?

- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- Visualization (TensorBoard)
- Save and restore models, graphs
- Auto-differentiation autodiff (no more taking derivatives by hand.)
- Large community (~300k commits, ~85k repositories)
- Awesome projects already using TensorFlow

Resources

- The official documentations
- TensorFlow's official sample models
- StackOverflow should be your first port of call in case of bug
- Books
  - Aurélien Géron's Hands-On Machine Learning with Scikit-Learn and TensorFlow (O'Reilly, March 2017)
  - François Chollet's Deep Learning with Python (Manning Publications, November 2017)
  - Nishant Shukla's Machine Learning with TensorFlow (Manning Publications, January 2018)
  - Lieder et al.'s Learning TensorFlow A Guide to Building Deep Learning Systems (O'Reilly, August 2017)

What’s a tensor?

An n-dimensional array

0-d tensor: scalar (number)

1-d tensor: vector

2-d tensor: matrix

and so on

Data Types

TensorFlow takes in Python native types such as Python boolean values, numeric values (integers, floats), and strings. Single values will be converted to 0-d tensors (or scalars), lists of values will be converted to 1-d tensors (vectors), lists of lists of values will be converted to 2-d tensors (matrices), and so on.

```
t_0 = 19 # Treated as a 0-d tensor, or "scalar"
tf.zeros_like(t_0)                  # ==> 0
tf.ones_like(t_0)                   # ==> 1
t_1 = [b"apple", b"peach", b"grape"] # treated as a 1-d tensor, or
"vector"
tf.zeros_like(t_1)                  # ==> [b'' b'' b'']
tf.ones_like(t_1)                   # ==> TypeError
t_2 = [[True, False, False],
       [False, False, True],
       [False, True, False]]        # treated as a 2-d tensor, or
"matrix" tf.zeros_like(t_2)                 # ==> 3x3 tensor, all
elements are Falsetf.ones_like(t_2)                  # ==> 3x3
tensor, all elements are True
```

Graph from *TensorFlow for Machine Intelligence*

TensorFlow Native Types

Like NumPy, TensorFlow also has its own data types as you've seen: tf.int32, tf.float32, together with more exciting types such as tf.bfloat, tf.complex, tf.quint.

- `tf.float16` : 16-bit half-precision floating-point.
- `tf.float32` : 32-bit single-precision floating-point.
- `tf.float64` : 64-bit double-precision floating-point.
- `tf.bfloat16` : 16-bit truncated floating-point.
- `tf.complex64` : 64-bit single-precision complex.
- `tf.complex128` : 128-bit double-precision complex.
- `tf.int8` : 8-bit signed integer.
- `tf.uint8` : 8-bit unsigned integer.
- `tf.uint16` : 16-bit unsigned integer.
- `tf.int16` : 16-bit signed integer.
- `tf.int32` : 32-bit signed integer.
- `tf.int64` : 64-bit signed integer.
- `tf.bool` : Boolean.
- `tf.string` : String.
- `tf.qint8` : Quantized 8-bit signed integer.
- `tf.quint8` : Quantized 8-bit unsigned integer.
- `tf.qint16` : Quantized 16-bit signed integer.
- `tf.quint16` : Quantized 16-bit unsigned integer.
- `tf.qint32` : Quantized 32-bit signed integer.
- `tf.resource` : Handle to a mutable resource.

TensorFlow vs Numpy

- Few people make this comparison, but TensorFlow and Numpy are quite similar. (Both are N-d array libraries!)
- Numpy has Narrary support but dosen't offer methods to create tensor functions and automatically compute derivatives (+no GPU support)

TensorFlow vs Numpy

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(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

TensorFlow requires explicit evaluation

```
In [37]: a = np.zeros((2,2))

In [38]: ta = tf.zeros((2,2))

In [39]: print(a)
[[ 0.  0.]
 [ 0.  0.]]

In [40]: print(ta)
Tensor("zeros_1:0", shape=(2, 2), dtype=float32)

In [41]: print(ta.eval())
[[ 0.  0.]
 [ 0.  0.]]
```

*TensorFlow computations define a **computation graph** that has no numerical value until evaluated!*

How to get the value of 'a'?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of a

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

```
>> Tensor("Add:0", shape=(), dtype=int32)
```

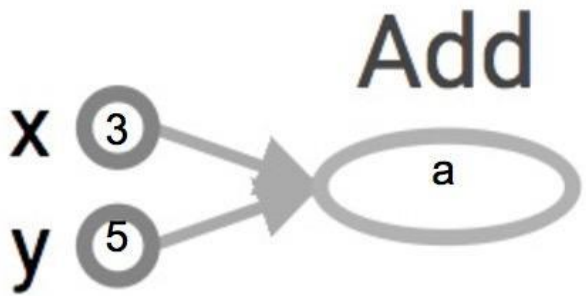
(Not 8)

How to get the value of a?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of a

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



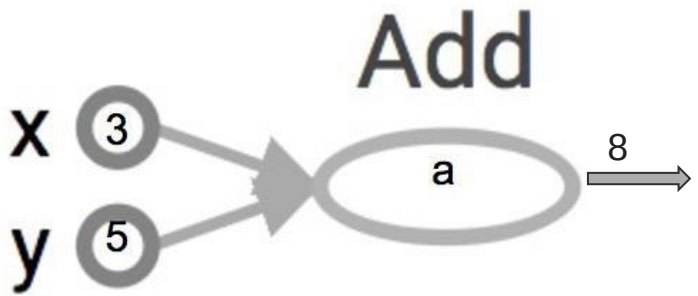
The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

How to get the value of a?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of a

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



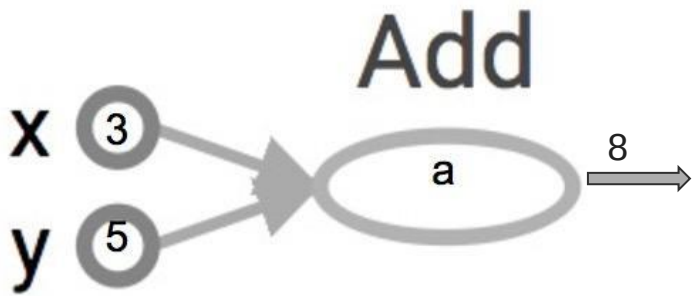
The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

How to get the value of a?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of a

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



tf.Session()

A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

tf.Session()

A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

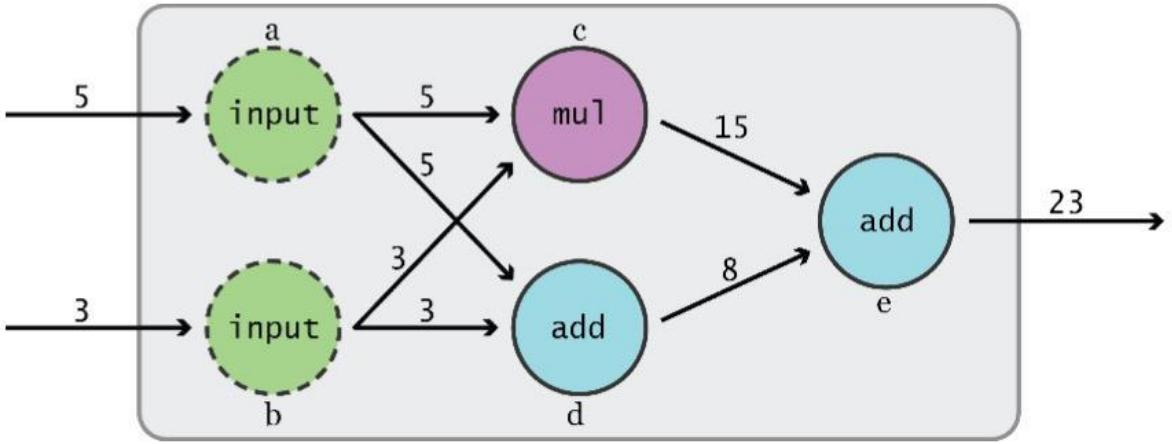
Session will also allocate memory to store the current values of variables.

TensorFlow Computation Graph

- TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute operations in the graph.
- All computations add nodes to global default graph

Data Flow Graphs

TensorFlow separates definition of computations from their execution

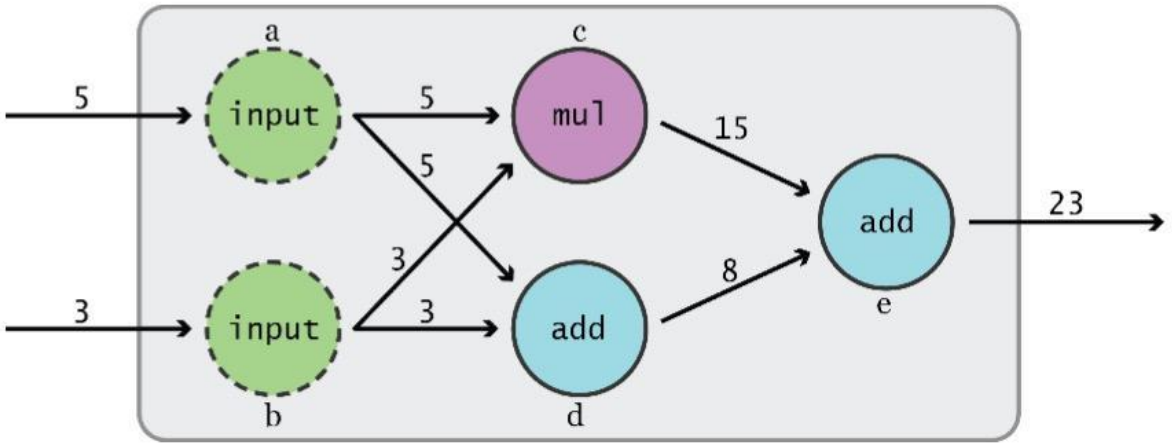


Graph from *TensorFlow for Machine Intelligence*

Data Flow Graphs

Phase 1: assemble a graph

Phase 2: use a session to execute operations in the graph.



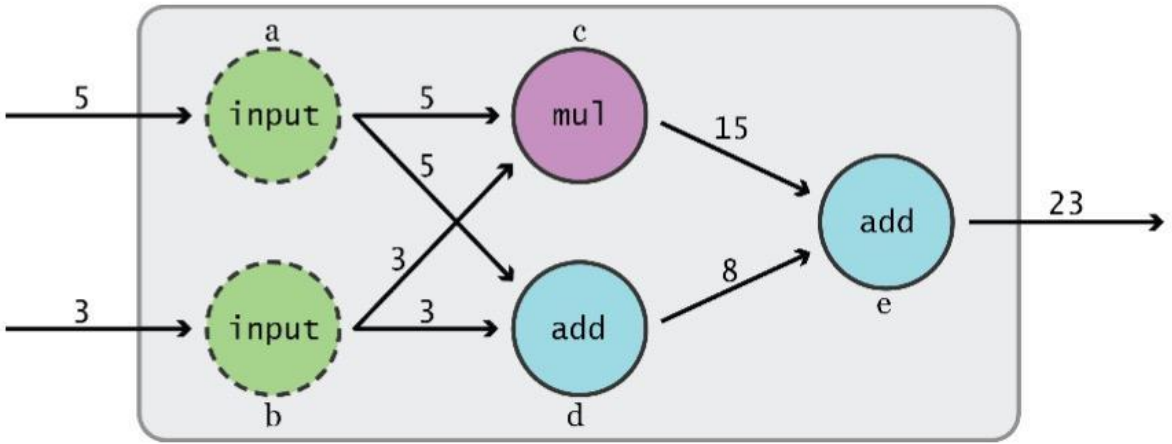
Graph from *TensorFlow for Machine Intelligence*

Data Flow Graphs

Phase 1: assemble a graph

This might change in the future with eager mode!!

Phase 2: use a session to execute operations in the graph.



Graph from *TensorFlow for Machine Intelligence*

TensorBoard

In TensorFlow, you collectively call constants, variables, operators as ops. TensorFlow is not just a software library, but a suite of software that include TensorFlow, TensorBoard, and Tensor Serving. To make the most out of TensorFlow, we should know how to use all of the above in conjunction with one another. So we will explore TensorBoard.
TensorBoard is a graph visualization software included with any standard TensorFlow installation. In Google's own words:

“The computations you'll use TensorFlow for - like training a massive deep neural network - can be complex and confusing. To make it easier to understand, debug, and optimize TensorFlow programs, we've included a suite of visualization tools called TensorBoard.”

TensorBoard GRAPHS INACTIVE

Fit to screen
Download PNG
Run (4) simple
Session runs (0)
Upload Choose File
Trace inputs
Color Structure
Device

Close legend.
Graph (* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?

Main GraphAuxiliary Nodes

TensorBoard

When a user perform certain operations in a TensorBoard-activated TensorFlow program, these operations are exported to an event log file. TensorBoard is able to convert these event files to visualizations that can give insight into a model's graph and its runtime behavior. Learning to use TensorBoard early and often will make working with TensorFlow that much more enjoyable and productive.

TensorBoard

Let's write your first TensorFlow program and visualize its computation graph with TensorBoard.

```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)
with tf.Session() as sess:
        print(sess.run(x))
```

To visualize the program with TensorBoard, we need to write log files of the program. To write event files, we first need to create a writer for those logs, using this code:

```
writer = tf.summary.FileWriter([logdir], [graph])
```

TensorBoard

[graph] is the graph of the program you are working on. You can either call it using tf.get_default_graph(), which returns the default graph of the program, or through sess.graph, which returns the graph the session is handling. The latter requires you to already have created a session. Either way, make sure to create a writer only after you've defined your graph, else the graph visualized on TensorBoard would be incomplete.

[logdir] is the folder where you want to store those log files. You can choose [logdir] to be something meaningful such as './graphs'

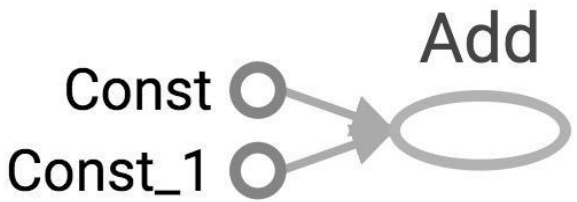
```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)writer = tf.summary.FileWriter('./graphs',
tf.get_default_graph())
with tf.Session() as sess:
# writer = tf.summary.FileWriter('./graphs', sess.graph) # if you
prefer creating your writer using session's graph
          print(sess.run(x))writer.close()
```

TensorBoard

Next, go to Terminal, run the program. Make sure that your present working directory is the same as where you ran your Python code.

```
$ python3 [my_program.py] $ tensorboard --logdir="./graphs"
--port 6006
```

Open your browser and go to http://localhost:6006/ (or the port of your choice), you will see the TensorBoard page. Go to the Graph tab and you can verify that the graph indeed has 3 nodes, two constants and an Add op.

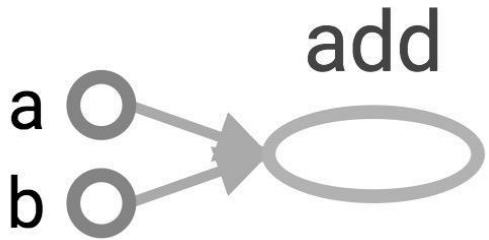


TensorBoard

"Const" and "Const_1" correspond to a and b, and the node "Add" corresponds to x. The names we give them (a, b, and x) are for us to access them when we write code. They mean nothing for the internal TensorFlow. To make TensorBoard understand the names of your ops, you have to explicitly name them.

```
a = tf.constant(2, name="a")
b = tf.constant(2, name="b")
x = tf.add(a, b, name="add")
```

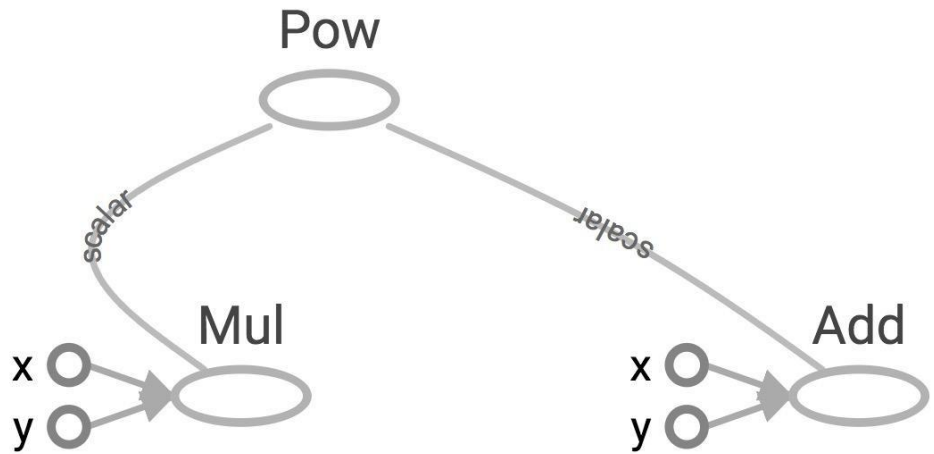
Now if you run TensorBoard again, you see this graph:



More graph

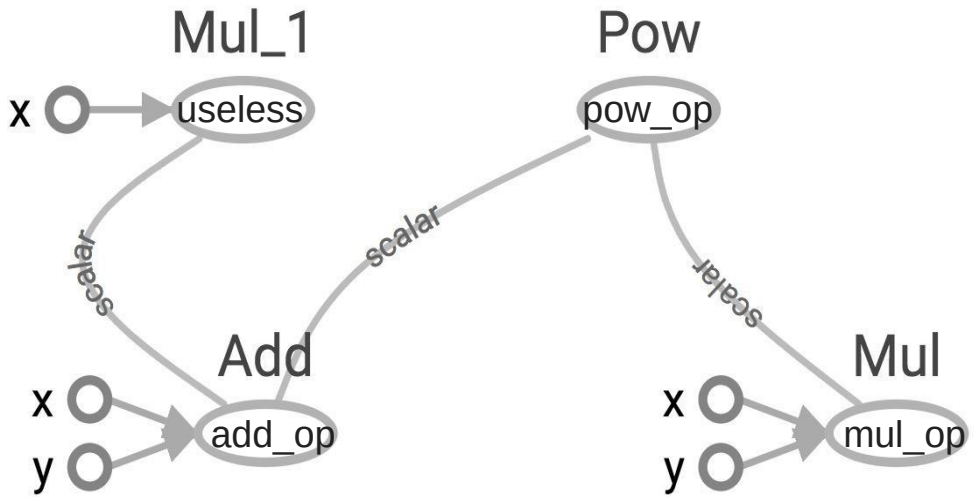
Visualized by TensorBoard

```
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)
```



Subgraphs

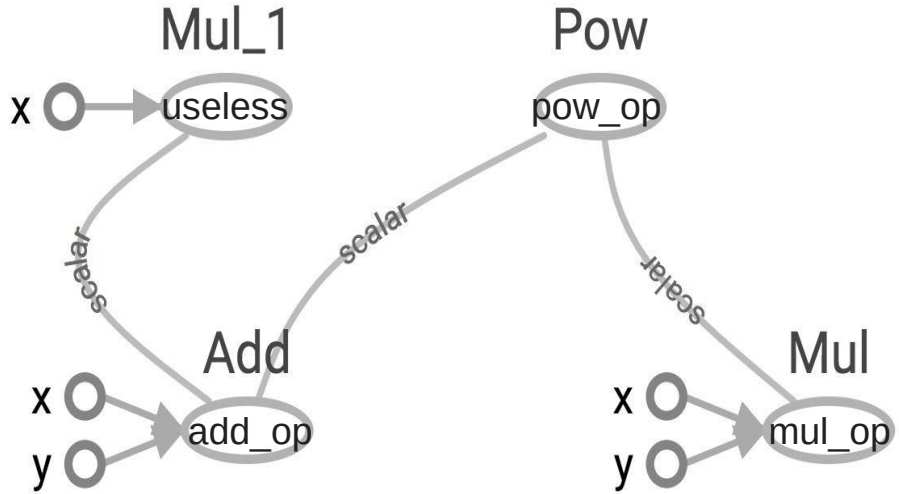
```
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
            z = sess.run(pow_op)
```



Because we only want the value of pow_op and pow_op doesn't depend on useless, session won't compute value of useless
→ save computation

Subgraphs

```
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, not_useless = sess.run([pow_op,
useless])
```



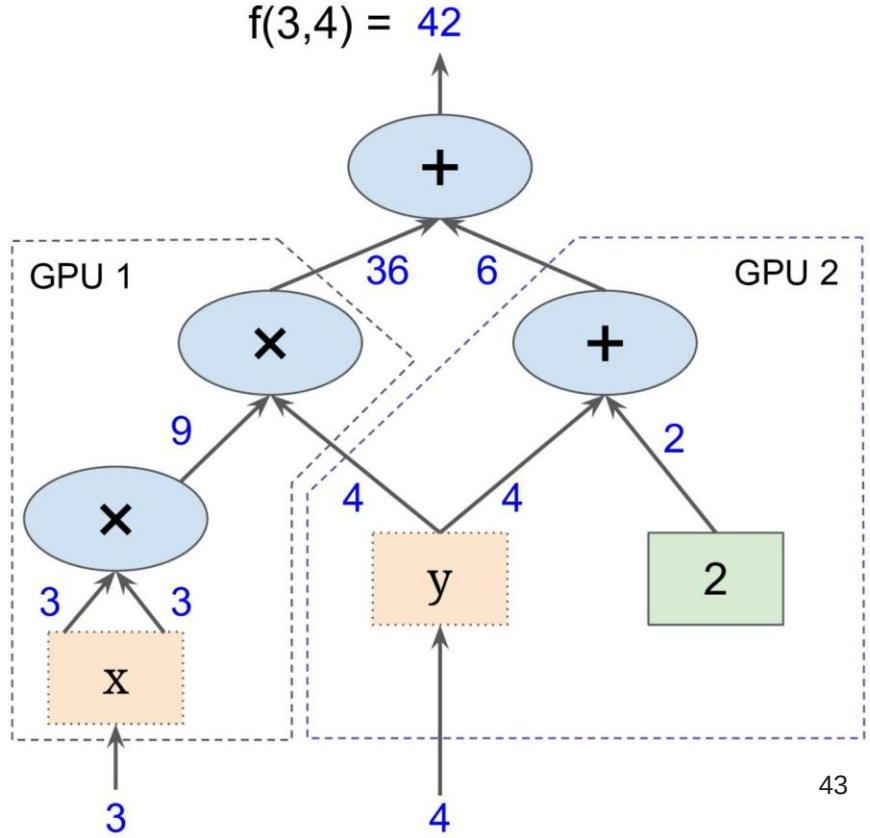

```
tf.Session.run(fetches,
                    feed_dict=None,
               options=None,
               run_metadata=None)
```

fetches is a list of tensors whose values you want

Subgraphs

Possible to break graphs into several chunks and run them parallelly across multiple CPUs, GPUs, TPUs, or other devices

Example: AlexNet



Graph from *Hands-On Machine Learning with Scikit-Learn and TensorFlow*

Distributed Computation

To put part of a graph on a specific CPU or GPU:

```
# Creates a graph.
with tf.device('/gpu:2'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='b')
  c = tf.multiply(a, b)

# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))

# Runs the op.
print(sess.run(c))
```

TensorFlow Variables

- When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors.

- Multiple graphs require multiple sessions, each will try to use all available resources by default
- Can't pass data between them without passing them through python/numpy, which doesn't work in distributed
- It’s better to have disconnected subgraphs within one graph

tf.Graph()

create a graph:

```
g = tf.Graph()
```

tf.Graph()

to add operators to a graph, set it as default:

```
g = tf.Graph()
with g.as_default():
           x = tf.add(3, 5)
sess = tf.Session(graph=g)
with tf.Session() as sess:
           sess.run(x)
```

tf.Graph()

To handle the default graph:

```
g = tf.get_default_graph()
```

tf.Graph()

Do not mix default graph and user created graphs

```
g = tf.Graph()

# add ops to the default graph
a = tf.constant(3)

# add ops to the user created graph
with g.as_default():
            b = tf.constant(5)
```

Prone to errors

tf.Graph()

Do not mix default graph and user created graphs

```
g1 = tf.get_default_graph()
g2 = tf.Graph()

# add ops to the default graph
with g1.as_default():
           a = tf.Constant(3)

# add ops to the user created graph
with g2.as_default():
           b = tf.Constant(5)
```

Better
But still not good enough because no more than one graph!

Why graphs

1. Save computation. Only run subgraphs that lead to the values you want to fetch.

Why graphs

1. Save computation. Only run subgraphs that lead to the values you want to fetch.
2. Break computation into small, differential pieces to facilitate auto-differentiation

Why graphs

1. Save computation. Only run subgraphs that lead to the values you want to fetch.
2. Break computation into small, differential pieces to facilitate auto-differentiation
3. Facilitate distributed computation, spread the work across multiple CPUs, GPUs, TPUs, or other devices

Why graphs

1. Save computation. Only run subgraphs that lead to the values you want to fetch.
2. Break computation into small, differential pieces to facilitate auto-differentiation
3. Facilitate distributed computation, spread the work across multiple CPUs, GPUs, TPUs, or other devices
4. Many common machine learning models are taught and visualized as directed graphs

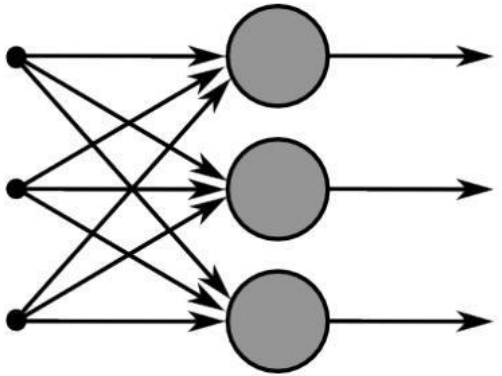


Figure 3: This image captures how multiple sigmoid units are stacked on the right, all of which receive the same input x.

A neural net graph from Stanford's course