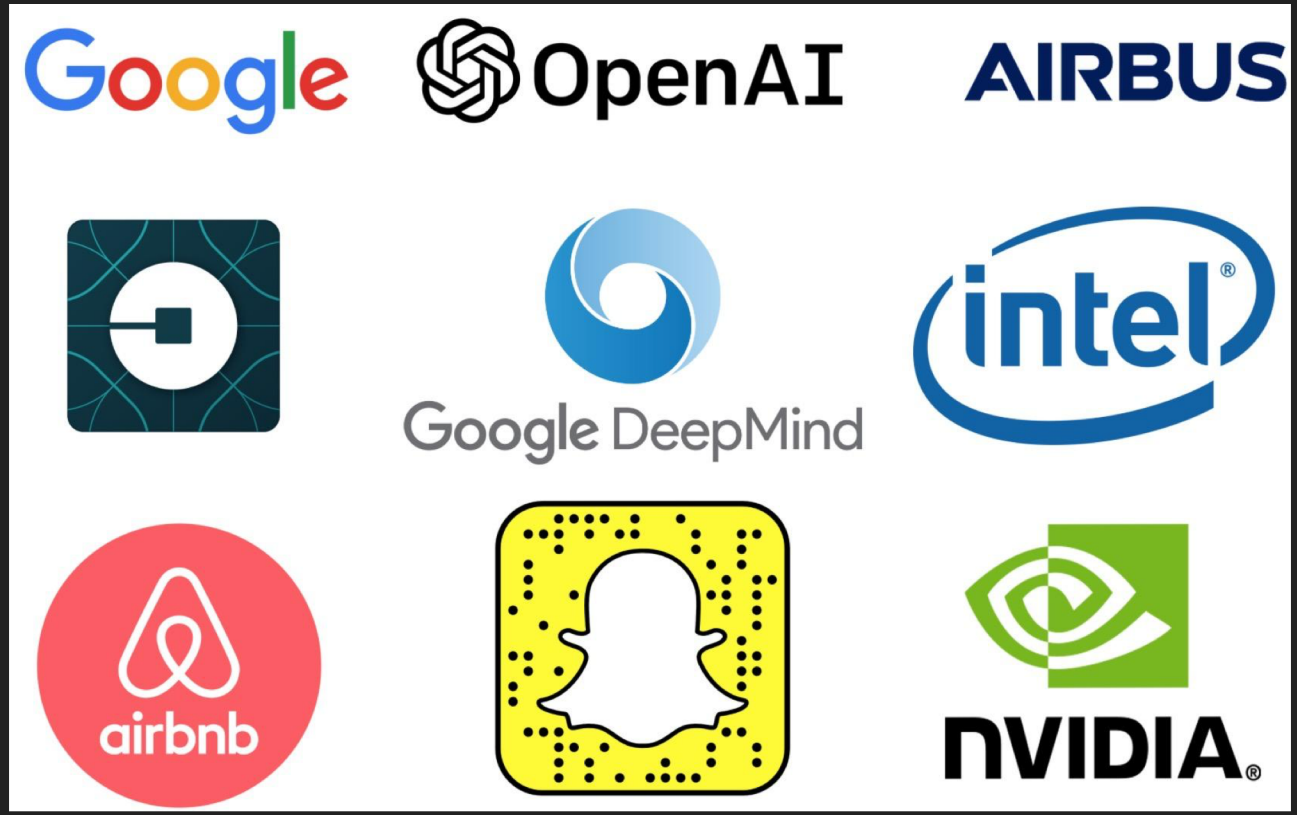
What's TensorFlow™?

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

Why TensorFlow?

- Flexibility + Scalability
  Originally developed by Google as a single infrastructure for machine learning in both production and research

Companies using TensorFlow



Neural Style Translation

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

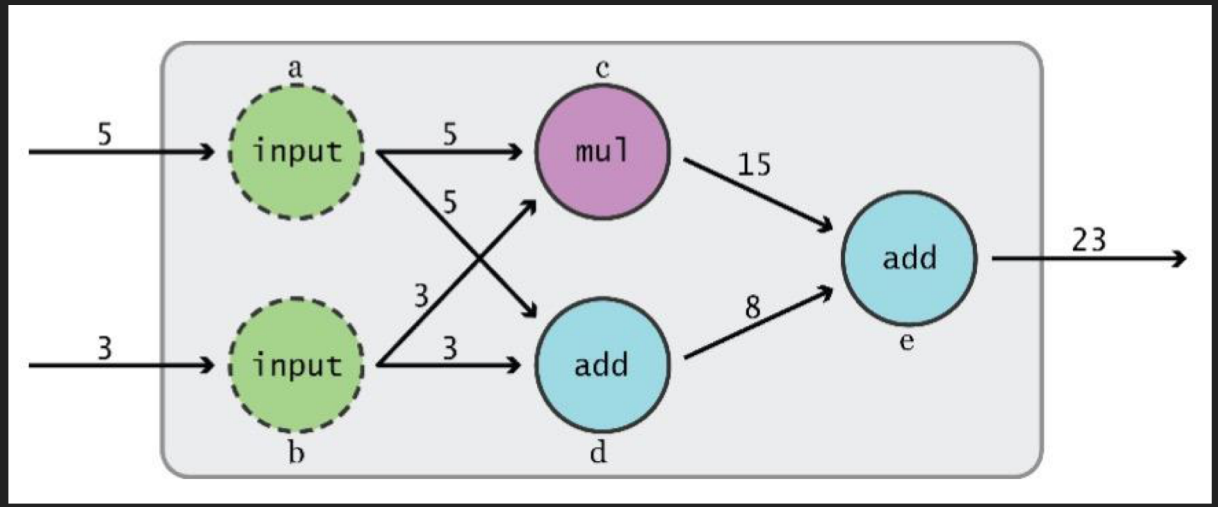
Goals

- Understand TF's computation graph approach
- Explore TF's built-in functions and classes
- Learn how to build and structure models best suited for a deep learning project

```
import tensorflow as tf
```

Data Flow Graphs

TensorFlow separates definition of computations from their execution

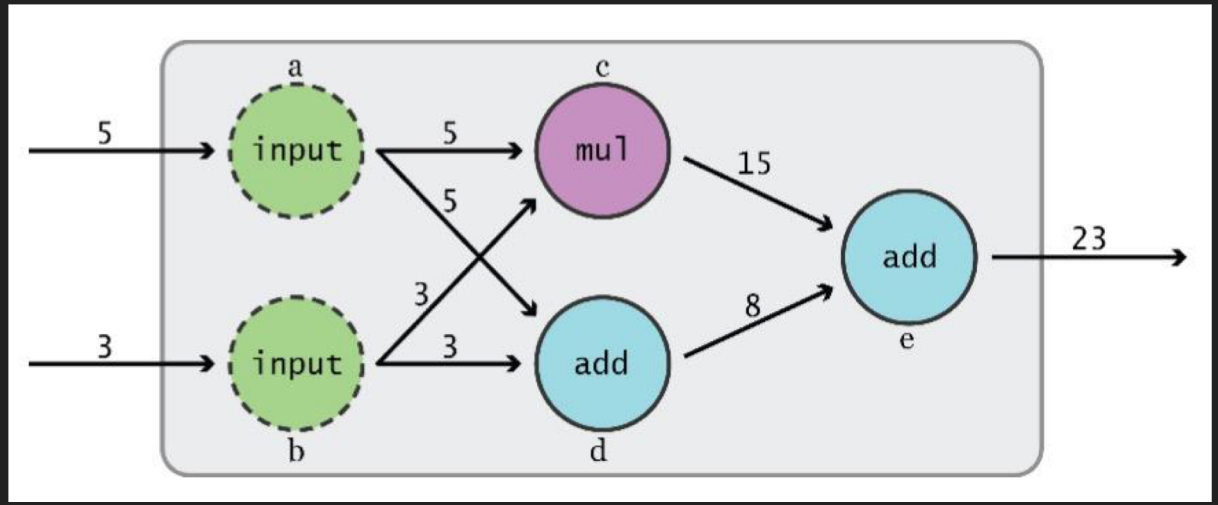


Graph from *TensorFlow for Machine Intelligence*

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*

What’s a tensor?

What's a tensor?

An n-dimensional array

0-d tensor: scalar (number)

1-d tensor: vector

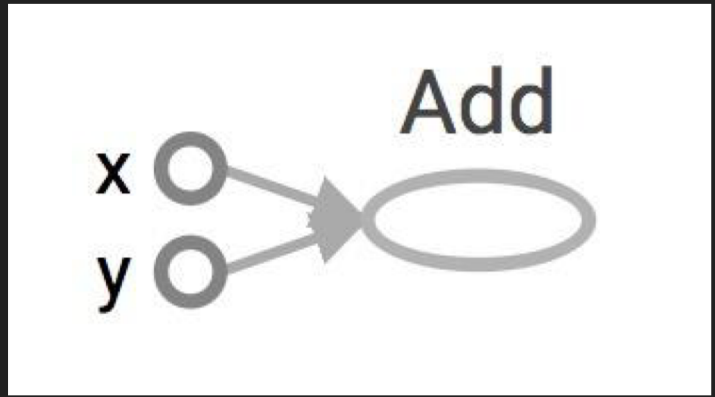
2-d tensor: matrix

and so on

Data Flow Graphs

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

Visualized by TensorBoard



Data Flow Graphs

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

writer = tf.summary.FileWriter("./tmp", tf.get_default_graph())

writer.close()

### >> tensorboard --logdir ./tmp --port 6006

### >> tensorboard --logdir ./tmp --host 127.0.0.1

### http://localhost:6006
```

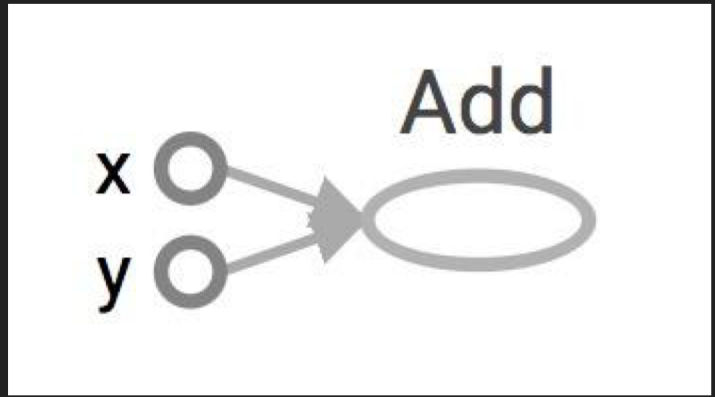
Data Flow Graphs

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

Why x, y?

TF automatically names the nodes when you don't explicitly name them.
x = 3
y = 5

Visualized by TensorBoard

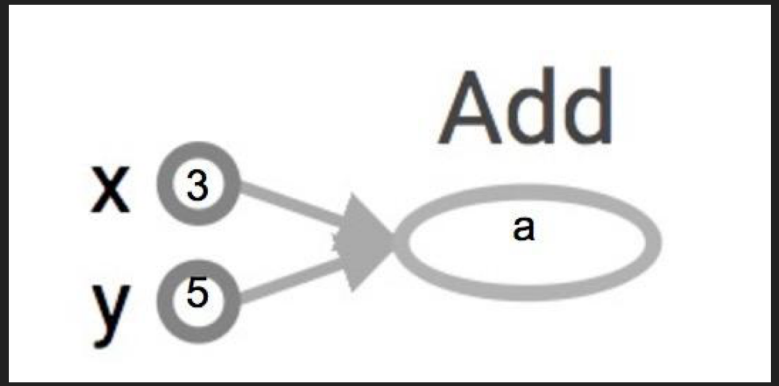


How to get the value of a?

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()
```



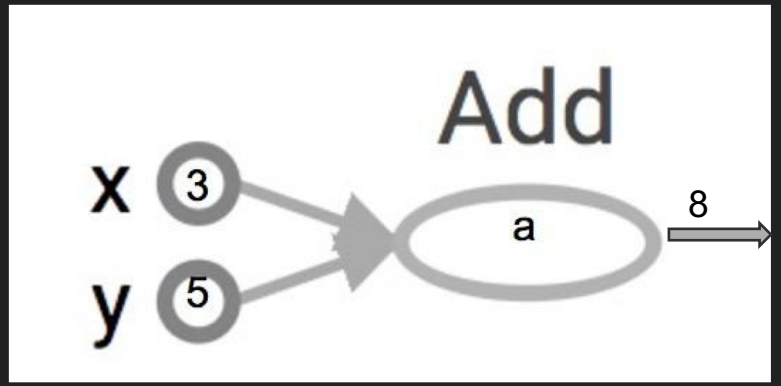
The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

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()
```



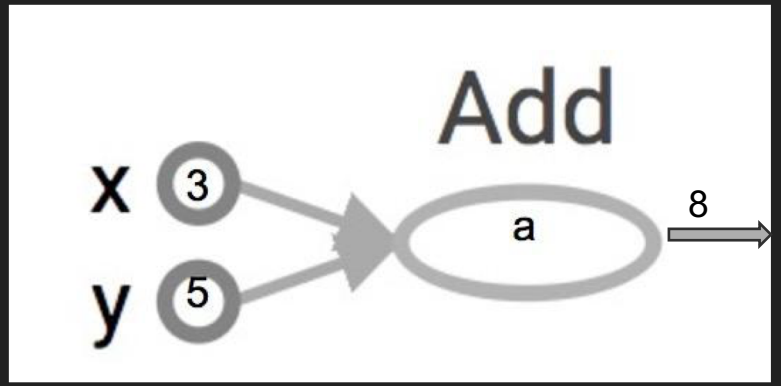
The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

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()

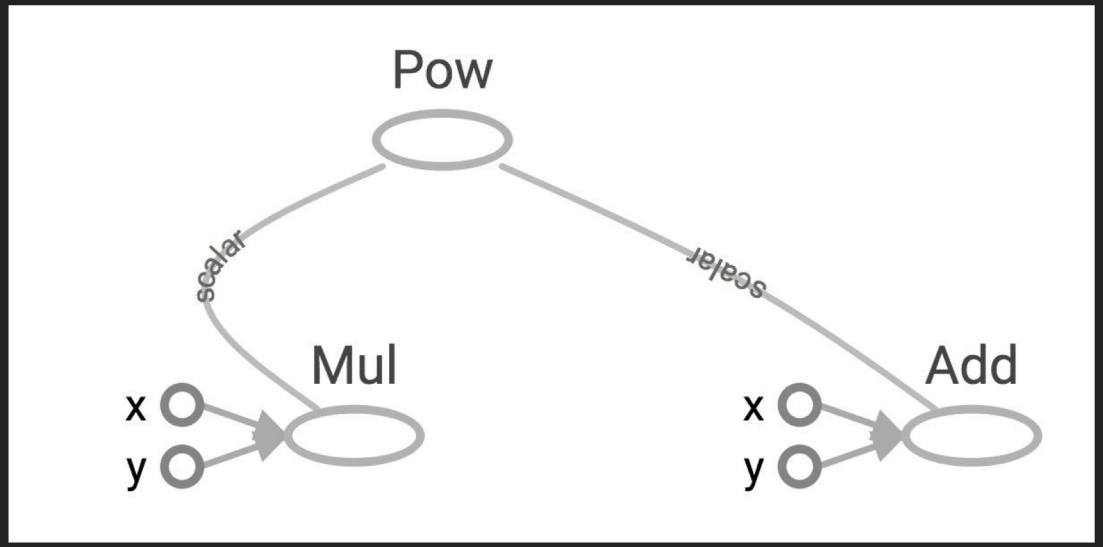
A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

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

More graph

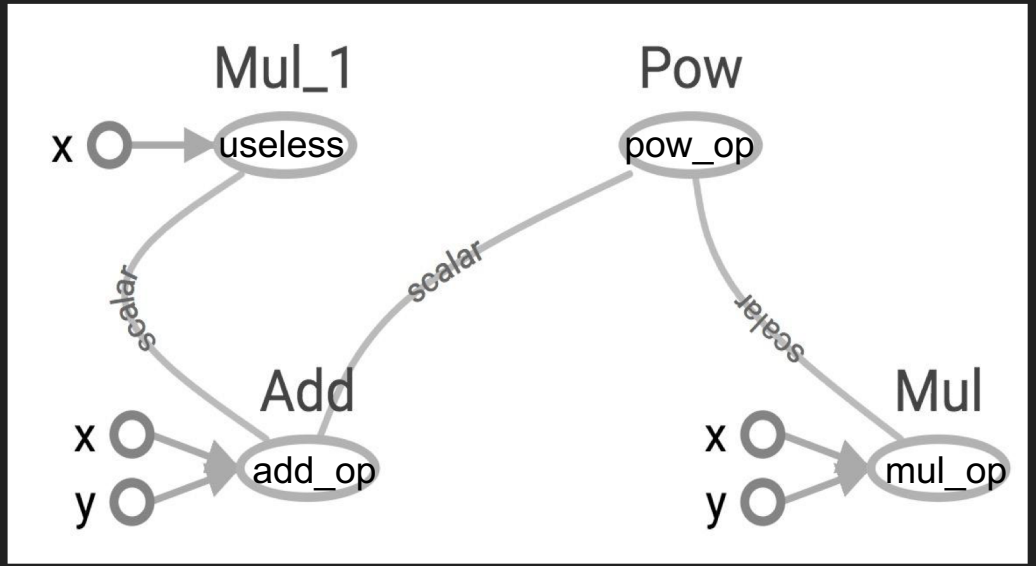
Visualized by TensorBoard

```
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

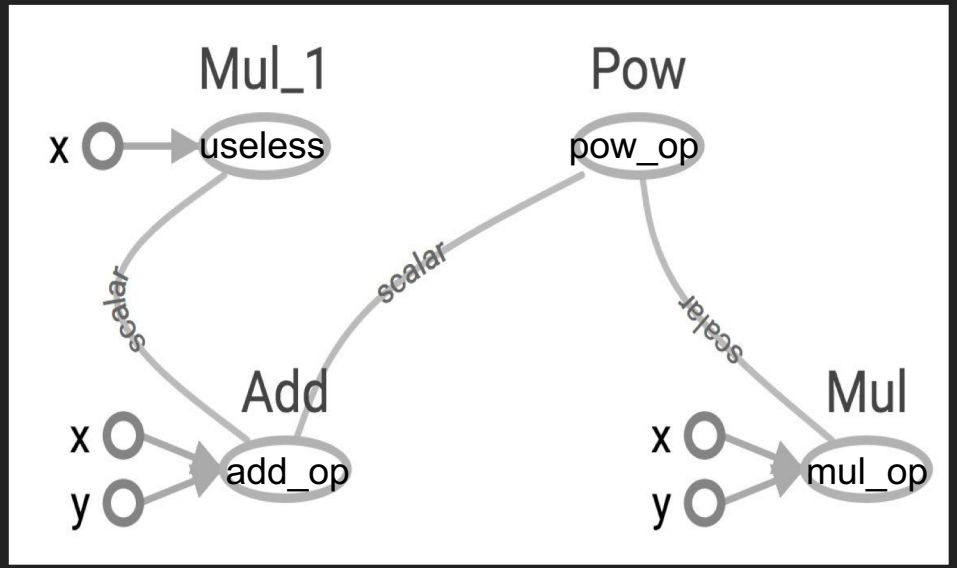
```
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
            z = sess.run(pow_op)
```



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

BUG ALERT!

- 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 handle the default graph:

```
g = tf.get_default_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

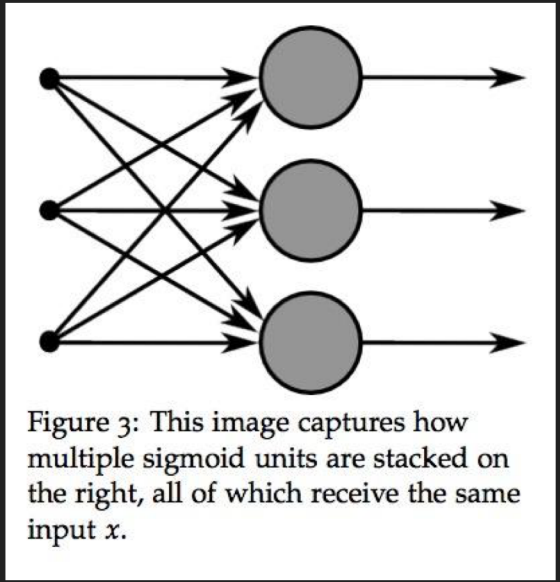


Figure 3: This image captures how multiple sigmoid units are stacked on the right, all of which receive the same input x.

A neural net graph from Stanford's CS224N course

Your first TensorFlow program

```
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))
```

Explicitly name them

```
import tensorflow as tf

a = tf.constant(2, name='a')
b = tf.constant(3, name='b')
x = tf.add(a, b, name='add')

writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
with tf.Session() as sess:
        print(sess.run(x)) # >> 5
```

TensorBoard can do much more than just visualizing your graphs.
Learn to use TensorBoard well and often!

Tensors filled with a specific value

```
tf.zeros(shape, dtype=tf.float32, name=None)
```

creates a tensor of shape and all elements will be zeros

Similar to numpy.zeros

```
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
```

Tensors filled with a specific value

```
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True)
```

creates a tensor of shape and type (unless type is specified) as the input_tensor but all elements are zeros.

Similar to numpy.zeros_like

```
# input_tensor is [[0, 1], [2, 3], [4, 5]]

tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]
```

Tensors filled with a specific value

```
tf.ones(shape, dtype=tf.float32, name=None)

tf.ones_like(input_tensor, dtype=None, name=None, optimize=True)
```

Similar to numpy.ones, numpy.ones_like

Tensors filled with a specific value

```
tf.fill(dims, value, name=None)
```

creates a tensor filled with a scalar value.

Similar to NumPy.full

```
tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]
```

Constants as sequences

tf.lin_space(start, stop, num, name=None)

```
tf.lin_space(10.0, 13.0, 4) ==> [10. 11. 12. 13.]
```

tf.range(start, limit=None, delta=1, dtype=None, name='range')

```
tf.range(3, 18, 3) ==> [3 6 9 12 15]
tf.range(5) ==> [0 1 2 3 4]
```

NOT THE SAME AS NUMPY SEQUENCES

Tensor objects are not iterable

```
for _ in tf.range(4): # TypeError
```

Randomly Generated Constants

```
tf.random_normal
tf.truncated_normal
tf.random_uniform
tf.random_shuffle
tf.random_crop
tf.multinomial
tf.random_gamma
```

Randomly Generated Constants

```
tf.set_random_seed(seed)
```

Operations

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

Buduma. *Fundamentals of Deep Learning*. O'Reilly, 2017

Arithmetic Ops

- `tf.abs`
- `tf.negative`
- `tf.sign`
- `tf.reciprocal`
- `tf.square`
- `tf.round`
- `tf.sqrt`
- `tf.rsqrt`
- `tf.pow`
- `tf.exp`

Pretty standard, quite similar to numpy.

- `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.

Variables

```
# create variables with tf.Variable
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))
```

Variables

```
# create variables with tf.Variable
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))

# create variables with tf.get_variable
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("big_matrix", shape=(784, 10), initializer=tf.zeros_initializer())
```

You have to initialize your variables

The easiest way is initializing all variables at once:

```
with tf.Session() as sess:
          sess.run(tf.global_variables_initializer())
```

Initializer is an op. You need to execute it within the context of a session

You have to initialize your variables

The easiest way is initializing all variables at once:

```
with tf.Session() as sess:
          sess.run(tf.global_variables_initializer())
```

Initialize only a subset of variables:

```
with tf.Session() as sess:
          sess.run(tf.variables_initializer([a, b]))
```

You have to initialize your variables

The easiest way is initializing all variables at once:

```
with tf.Session() as sess:
          sess.run(tf.global_variables_initializer())
```

Initialize only a subset of variables:

```
with tf.Session() as sess:
          sess.run(tf.variables_initializer([a, b]))
```

Initialize a single variable

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

A quick reminder

A TF program often has 2 phases:

1. Assemble a graph
2. Use a session to execute operations in the graph.

Placeholders

A TF program often has 2 phases:

1. Assemble a graph
2. Use a session to execute operations in the graph.

⇒ Assemble the graph first without knowing the values needed for computation

Placeholders

A TF program often has 2 phases:

1. Assemble a graph
2. Use a session to execute operations in the graph.

⇒ Assemble the graph first without knowing the values needed for computation

Analogy:
Define the function f(x, y) = 2 * x + y without knowing value of x or y.
x, y are placeholders for the actual values.

Why placeholders?

We, or our clients, can later supply their own data when they need to execute the computation.

Placeholders

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

```
# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

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

Placeholders

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

```
# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

with tf.Session() as sess:
            print(sess.run(c))                              # >> InvalidArgumentError: a doesn't an actual
value
```

Supplement the values to placeholders using a dictionary

Placeholders

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

```
# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

with tf.Session() as sess:
            print(sess.run(c, feed_dict={a: [1, 2, 3]}))    # the tensor a is the key, not the string ‘a’

# >> [6, 7, 8]
```

Placeholders

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

```
# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

with tf.Session() as sess:
            print(sess.run(c, feed_dict={a: [1, 2, 3]}))

# >> [6, 7, 8]
```

Quirk:
shape=None means that tensor of any shape will be accepted as value for placeholder.

shape=None is easy to construct graphs, but nightmarish for debugging

References:

CS20

http://web.stanford.edu/class/cs20si/syllabus.html