



TensorFlow Ops

CS 20: TensorFlow for Deep Learning Research

Lecture 2

1/17/2017

Agenda

Basic operations

Tensor types

Importing data

Lazy loading



Fun with TensorBoard!!!

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))
```

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))
```

Warning?

The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.

Your first TensorFlow program

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
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))
```

No more warning

Visualize it with TensorBoard

```
import tensorflow as tf
```

```
a = tf.constant(2)
```

```
b = tf.constant(3)
```

```
x = tf.add(a, b)
```

Create the summary writer after graph definition and before running your session

```
writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
```

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

```
    # writer = tf.summary.FileWriter('./graphs', sess.graph)
```

```
    print(sess.run(x))
```

```
writer.close() # close the writer when you're done using it
```

‘graphs’ or any location where you want to keep your event files

Run it


Go to terminal, run:


```
$ python3 [yourprogram].py
```

```
$ tensorboard --logdir="./graphs" --port 6006
```

6006 or any port you want

Then open your browser and go to: <http://localhost:6006/>

 Fit to screen

 Download PNG

Run

simple ▾

(4)

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



Visualize it with TensorBoard

```
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())
writer.close()
```



Visualize it with TensorBoard

```
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())
writer.close()
```



Question:

How to change Const, Const_1 to the names we give the variables?

Explicitly name them

```
import tensorflow as tf

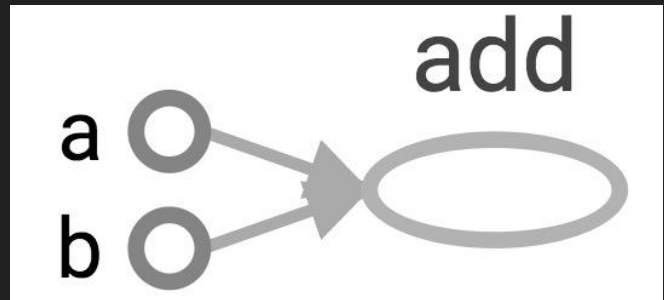
a = tf.constant(2, name='a')
b = tf.constant(3, name='b')
x = tf.add(a, b, name='add')

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

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!**



Constants, Sequences, Variables, Ops

Constants

```
import tensorflow as tf
```

```
a = tf.constant([2, 2], name='a')
```

```
b = tf.constant([[0, 1], [2, 3]], name='b')
```

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


Constants

```
import tensorflow as tf

a = tf.constant([2, 2], name='a')
b = tf.constant([[0, 1], [2, 3]], name='b')
x = tf.multiply(a, b, name='mul')
```

Broadcasting similar to NumPy

```
with tf.Session() as sess:
    print(sess.run(x))
```

```
# >> [[0 2]
#      [4 6]]
```

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

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 vs NP Data Types

TensorFlow integrates seamlessly with NumPy

```
tf.int32 == np.int32          # ⇒ True
```

Can pass numpy types to TensorFlow ops

```
tf.ones([2, 2], np.float32)   # ⇒ [[1.0 1.0], [1.0 1.0]]
```

For **tf.Session.run(fetches)**: if the requested fetch is a Tensor , output will be a NumPy ndarray.

```
sess = tf.Session()
a = tf.zeros([2, 3], np.int32)
print(type(a))          # ⇒ <class 'tensorflow.python.framework.ops.Tensor'>
a = sess.run(a)
print(type(a))          # ⇒ <class 'numpy.ndarray'>
```

TF vs NP Data Types

TensorFlow integrates seamlessly with NumPy

```
tf.int32 == np.int32          # ⇒ True
```

Can pass numpy types to TensorFlow ops

```
tf.ones([2, 2], np.float32)  # ⇒ [[1.0 1.0], [1.0 1.0]]
```

For **tf.Session.run(fetches)**: if the requested fetch is a Tensor , output will be a NumPy ndarray.

```
sess = tf.Session()
a = tf.zeros([2, 3], np.int32)
print(type(a))
a = sess.run(a)
print(type(a))
```

<<<< Avoid doing this. Use `a_out = sess.run(a)`

Use TF DType when possible

- Python native types: TensorFlow has to infer Python type

Use TF DType when possible

- Python native types: TensorFlow has to infer Python type
- NumPy arrays: NumPy is not GPU compatible

What's wrong with constants ...

... other than being constant?

What's wrong with constants?

Constants are stored in the graph definition

What's wrong with constants?

This makes loading graphs expensive when constants are big

What's wrong with constants?

This makes loading graphs expensive when constants are big



Only use constants for primitive types.

Use variables or readers for more data that requires more memory

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())
```

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())
```



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]))
```

Why tf.constant but tf.Variable?

```
# 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())
```

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]))
```

tf.constant is an op

tf.Variable is a class with many ops

```
# 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())
```

tf.Variable class

```
# 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())
```

tf.Variable holds several ops:

```
x = tf.Variable(...)

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

tf.Variable class

```
# 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())

with tf.Session() as sess:
    print(sess.run(W))    >> FailedPreconditionError: Attempting to use uninitialized value Variable
```

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)
```



Placeholder

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

Placeholders

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

```
# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])
```

```
# create a constant of type float 32-bit, shape is a vector of 3 elements
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, {a: [1, 2, 3]}))
```

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

Quirk:

`shape=None` also breaks all following shape inference, which makes many ops not work because they expect certain rank.

Placeholders are valid ops

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

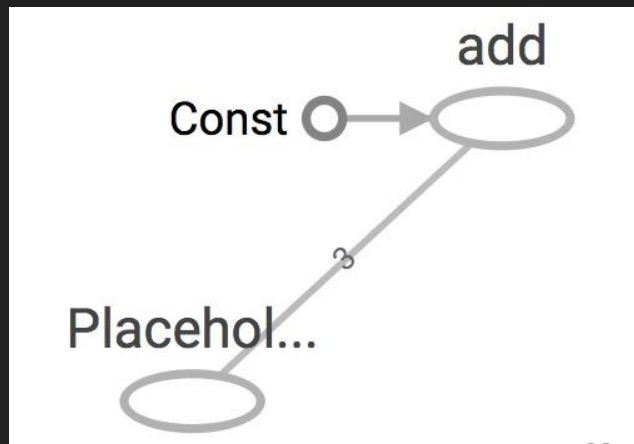
```
# create a placeholder of type float 32-bit, shape is a vector of 3 elements  
a = tf.placeholder(tf.float32, shape=[3])
```

```
# create a constant of type float 32-bit, shape is a vector of 3 elements  
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, {a: [1, 2, 3]}))
```

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



What if want to feed multiple data points in?

You have to do it one at a time

```
with tf.Session() as sess:  
    for a_value in list_of_values_for_a:  
        print(sess.run(c, {a: a_value}))
```

**You can feed_dict any feedable tensor.
Placeholder is just a way to indicate that
something must be fed**

```
tf.Graph.is_feedable(tensor)  
# True if and only if tensor is feedable.
```

Feeding values to TF ops

```
# create operations, tensors, etc (using the default graph)
a = tf.add(2, 5)
b = tf.multiply(a, 3)

with tf.Session() as sess:
    # compute the value of b given a is 15
    sess.run(b, feed_dict={a: 15})           # >> 45
```