Getting started with TensorFlow

In this notebook, you play around with the TensorFlow Python API.

In [1]:

```
import tensorflow as tf
import numpy as np

print(tf.__version__)
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of
the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

1.8.0
```

Adding two tensors

First, let's try doing this using numpy, the Python numeric package. numpy code is immediately evaluated.

In [2]:

```
a = np.array([5, 3, 8])
b = np.array([3, -1, 2])
c = np.add(a, b)
print(c)
```

```
[ 8  2 10]
```

The equivalent code in TensorFlow consists of two steps:

Step 1: Build the graph

In [3]:

```
a = tf.constant([5, 3, 8])
b = tf.constant([3, -1, 2])
c = tf.add(a, b)
print(c)
```

```
Tensor("Add:0", shape=(3,), dtype=int32)
```

c is an Op ("Add") that returns a tensor of shape (3,) and holds int32. The shape is inferred from the computation graph.

Try the following in the cell above:

1. Change the 5 to 5.0, and similarly the other five numbers. What happens when you run this cell?
2. Add an extra number to a, but leave b at the original (3,) shape. What happens when you run this cell?
3. Change the code back to a version that works

Step 2: Run the graph

In [4]:

```
with tf.Session() as sess:
  result = sess.run(c)
  print(result)
```

```
[ 8  2 10]
```

Using a feed_dict

Same graph, but without hardcoding inputs at build stage

In [5]:

```
a = tf.placeholder(dtype=tf.int32, shape=(None,))  # batchsize x scalar
b = tf.placeholder(dtype=tf.int32, shape=(None,))
c = tf.add(a, b)
with tf.Session() as sess:
  result = sess.run(c, feed_dict={
      a: [3, 4, 5],
      b: [-1, 2, 3]
    })
  print(result)
```

```
[2 6 8]
```

Heron's Formula in TensorFlow

The area of triangle whose three sides are (a, b, c) is $\sqrt{s(s-a)(s-b)(s-c)}$ where $s=\frac{a+b+c}{2}$

Look up the available operations at https://www.tensorflow.org/api_docs/python/tf (https://www.tensorflow.org/api_docs/python/tf)

In [6]:

```
def compute_area(sides):
  # slice the input to get the sides
  a = sides[:,0]  # 5.0, 2.3
  b = sides[:,1]  # 3.0, 4.1
  c = sides[:,2]  # 7.1, 4.8

  # Heron's formula
  s = (a + b + c) * 0.5   # (a + b) is a short-cut to tf.add(a, b)
  areasq = s * (s - a) * (s - b) * (s - c) # (a * b) is a short-cut to tf.multiply(a, b), not tf.matmul(a, b)
  return tf.sqrt(areasq)

with tf.Session() as sess:
  # pass in two triangles
  area = compute_area(tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8]
    ]))
  result = sess.run(area)
  print(result)
```

```
[6.278497 4.709139]
```

Placeholder and feed_dict

More common is to define the input to a program as a placeholder and then to feed in the inputs. The difference between the code below and the code above is whether the "area" graph is coded up with the input values or whether the "area" graph is coded up with a placeholder through which inputs will be passed in at run-time.

In [7]:

```
with tf.Session() as sess:
  sides = tf.placeholder(tf.float32, shape=(None, 3))  # batchsize number of triangles, 3 sides
  area = compute_area(sides)
  result = sess.run(area, feed_dict = {
      sides: [
        [5.0, 3.0, 7.1],
        [2.3, 4.1, 4.8]
      ]
    })
  print(result)
```

```
[6.278497 4.709139]
```

tf.eager

tf.eager allows you to avoid the build-then-run stages. However, most production code will follow the lazy evaluation paradigm because the lazy evaluation paradigm is what allows for multi-device support and distribution.

One thing you could do is to develop using tf.eager and then comment out the eager execution and add in the session management code.

To run this block, you must first reset the notebook using Reset on the menu bar, then run this block only.

In [1]:

```
import tensorflow as tf
from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

def compute_area(sides):
  # slice the input to get the sides
  a = sides[:,0]  # 5.0, 2.3
  b = sides[:,1]  # 3.0, 4.1
  c = sides[:,2]  # 7.1, 4.8

  # Heron's formula
  s = (a + b + c) * 0.5   # (a + b) is a short-cut to tf.add(a, b)
  areasq = s * (s - a) * (s - b) * (s - c) # (a * b) is a short-cut to tf.multiply(a, b), not tf.matmul(a, b)
  return tf.sqrt(areasq)

area = compute_area(tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8]
    ]))

print(area)
```

```
/usr/local/envs/py3env/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of
the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

tf.Tensor([6.278497 4.709139], shape=(2,), dtype=float32)
```