Cheatsheet for TensorFlow

Importing

The line to import tensorflow by convention is as follows:

```
import tensorflow as tf
```

Placeholders (inputs)

A `placeholder` is a value to input on a TensorFlow computation. It cannot be evaluated, as it must be fed a value using a `feed_dict`.

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
y = tf.matmul(x, x) # Matrix Multiplication (or Dot product)

with tf.Session() as session:
    print(session.run(y)) # Error: x wasn't fed any value

    rand_array = np.random.rand(1024, 1024)
    print(session.run(y, feed_dict={x: rand_array}) # Success
```

Variables

Variables are useful to hold and update parameters. Unlike placeholders, variables are initialized with some value and can be updated through the computation. They are in-memory buffers containing tensors.

```
# create two variables
w = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                name='w')
b = tf.Variable(tf.zeros([200]), name='b')
```

Initialization

A variable initializer must be run before other ops in the model can be run. Exists an operation that runs all variable initilizations: `tf.initialize_all_variables()`.

```
# create two variables
w = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                name='w')
b = tf.Variable(tf.zeros([200]), name='b')
...
```

```
# add an op to initialize the variables
init_op = tf.initialize_all_variables()

# later, when launching the model
with tf.Session() as sess:
    # run the init operation
    sess.run(init_op)
    ...
    # use the model
    ...
```

Initialization from another variable

```
# create a variable with a random value.
w1 = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                                  name='w1')
# create another variable with the same value as 'w1'.
w2 = tf.Variable(w1.initialized_value(), name='w1')
# Create another variable with twice the value of 'w1'
w1_twice = tf.Variable(w1.initialized_value() * 2.0, name="w1_twice")
```

Saving a session's variables

Use a `tf.train.Saver()` object to save the session's variables (all or some) and restore it later.

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add an op to initialize the variables.
init_op = tf.initialize_all_variables()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, initialize the variables, do some work, save the
# variables to disk.
with tf.Session() as sess:
    sess.run(init_op)
    # Do some work with the model.
    ..
    # Save the variables to disk.
    save_path = saver.save(sess, "/tmp/model.ckpt")
    print("Model saved in file: %s" % save_path)
```

Restoring a session's variables

How to restore variables:

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
    # Restore variables from disk.
    saver.restore(sess, "/tmp/model.ckpt")
    print("Model restored.")
    # Do some work with the model
    ...
```

Choose variables to save in a session

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore only 'v2' using the name "my_v2"
saver = tf.train.Saver({"my_v2": v2})
# Use the saver object normally after that.
...
```