Tensorflow Cheatsheet

Quick Markdown Cheatsheet

```
Copy paste this into markdown for page breaks:
<div style="page-break-after: always;"></div>
```

Tensorflow Code generally looks like this

1. Declare variables
2. Build computation graph
3. Put variables into graph and run session

Basic Operations

Running a session

```
sess = tf.Session()
x = tf.placeholder(tf.float32, shape=(0))
x = 1
y = tf.print(x)
sess.run(y)
```

Print

```
#Easier:
A = tf.keras.backend.print_tensor(A, message = '')

#Harder: (also need to do sess.run)
tf.print()

#Can also print like this:
with tf.Session() as sess:
    print(sess.run(tf.shape(a_array)))
    print(sess.run(tf.shape(b_list)))
    print(sess.run(tf.shape(c_tensor)))
```

Check shape of tensor

```
tf.shape(tensor)
```

Get shape of tensor as a list

```
act_dim = a.shape.as_list()[-1]
```

Quick way to add to a list

```
list(hidden_sizes)+[act_dim]
```

What does if **name** == '**main**': do?

Source: https://stackoverflow.com/questions/419163/what-does-if-name-main-do

```
See link for how interpreter sets __name__ to __main__
Rough idea is that this allows you to run only the modules you want when
you're importing them into a different "main" program.
Example:
So let's say you want to use a function from a different python file. The
__main__ function from that python file will not run (which is what you
want because you only want to import the functions)
```

Converting from tf to numpy and vice versa

tf to numpy

```
numpy_array = tensor.eval()
```

numpy to tf

```
tensor = tf.constant(np_array)
```

Multiplication

Element-wise multiplication

```
tf.multiply(X, Y)
```

Matrix multiplication

```
tf.matmul(X, Y)
or
tf.matmul(X, Y, transpose_b=True) if you wanna multiply by Y transpose
```

Sum

```
tf.reduce_sum(_, axis=1) //sums over rows
```

See Documentation:
https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/math/reduce_sum

```
Axis = None; All dimensions are reduced, tensor of single element is
returned
keepdims = True; retains reduced dimensions with length 1; So like, [[3],
[3]] instead of [3, 3]
```

Numpy-like tensor indexing

See full discussion here: https://stackoverflow.com/questions/33736795/tensorflow-numpy-like-tensor-indexing

```
import numpy as np
import tensorflow as tf

m = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
rows = np.array([[0, 1], [0, 1], [1, 0], [0, 0]])
cols = np.array([[2, 1], [1, 2], [0, 2], [0, 0]])

x = tf.placeholder('float32', (None, None))
idx1 = tf.placeholder('int32', (None, None))
idx2 = tf.placeholder('int32', (None, None))
result = tf.gather_nd(x, tf.stack((idx1, idx2), -1))

with tf.Session() as sess:
    r = sess.run(result, feed_dict={
        x: m,
        idx1: rows,
        idx2: cols,
    })
print(r)
```

Tensorflow Placeholder

Shape

Discussion: https://stackoverflow.com/questions/46940857/what-is-the-difference-between-none-none-and-for-the-shape-of-a-placeh

As can be seen, placeholder with [] shape takes a single scalar value directly. Placeholder with [None] shape takes a 1-dimensional array and placeholder with None shape can take in any value while computation takes place.

Difference between placeholder and variable

Source: https://stackoverflow.com/questions/36693740/whats-the-difference-between-tf-placeholder-and-tf-variable

Placeholders are where you put your training examples, variables are for trainable variables such as W (weight) and b (bias)

Neural Networks Example

Multilayer perceptron example

Source: https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/

Code

```
# Network Parameters
n_hidden_1 = 256 # 1st layer number of features
n_hidden_2 = 256 # 2nd layer number of features
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])

def multilayer_perceptron(x, weights, biases):
# Hidden layer with ReLU activation
layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
layer_1 = tf.nn.relu(layer_1)
# Hidden layer with ReLU activation
layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
layer_2 = tf.nn.relu(layer_2)
# Output layer with linear activation
out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
return out_layer

# Store layers weight &amp;amp; bias
weights = {
'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
```

```
'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
}
biases = {
'b1': tf.Variable(tf.random_normal([n_hidden_1])),
'b2': tf.Variable(tf.random_normal([n_hidden_2])),
'out': tf.Variable(tf.random_normal([n_classes]))
}

# Construct model
pred = multilayer_perceptron(x, weights, biases)
```