TensorBoard Basic

To learn how to use TensorBoard, based on notes (https://github.com/fluxcapacitor/pipeline/blob/master/myapps/jupyter/TensorFlow/Fundamentals/Tens

In [ ]:
```
# TensorBoard Helper Functions and Constants

# Directory to export TensorBoard summary statistics, graph data, etc.
TB_DIR = '/tmp/tensorboard/helloworld'


def _start_tb(d):
    """
    Private function that calls `tensorboard` shell command

    args:
      d: The desired directory to launch in TensorBoard
    """
    !tensorboard --port=6006 --logdir=$d

def start_tensorboard():
    """
    Starts TensorBoard from the notebook in a separate thread.
    Prevents Jupyter Notebook from halting while TensorBoard runs.
    """
    import threading
    threading.Thread(target=_start_tb, args=(TB_DIR,)).start()
    del threading

def stop_tensorboard():
    """
    Kills all TensorBoard processes
    """
    !ps -aef | grep "tensorboard" | tr -s ' ' | cut -d ' ' -f2 | xargs kill -

def reset_tensorboard():
    stop_tensorboard()
    start_tensorboard()
```

In [ ]:
```
# Import core TensorFlow libraries
import tensorflow as tf
import numpy as np
```

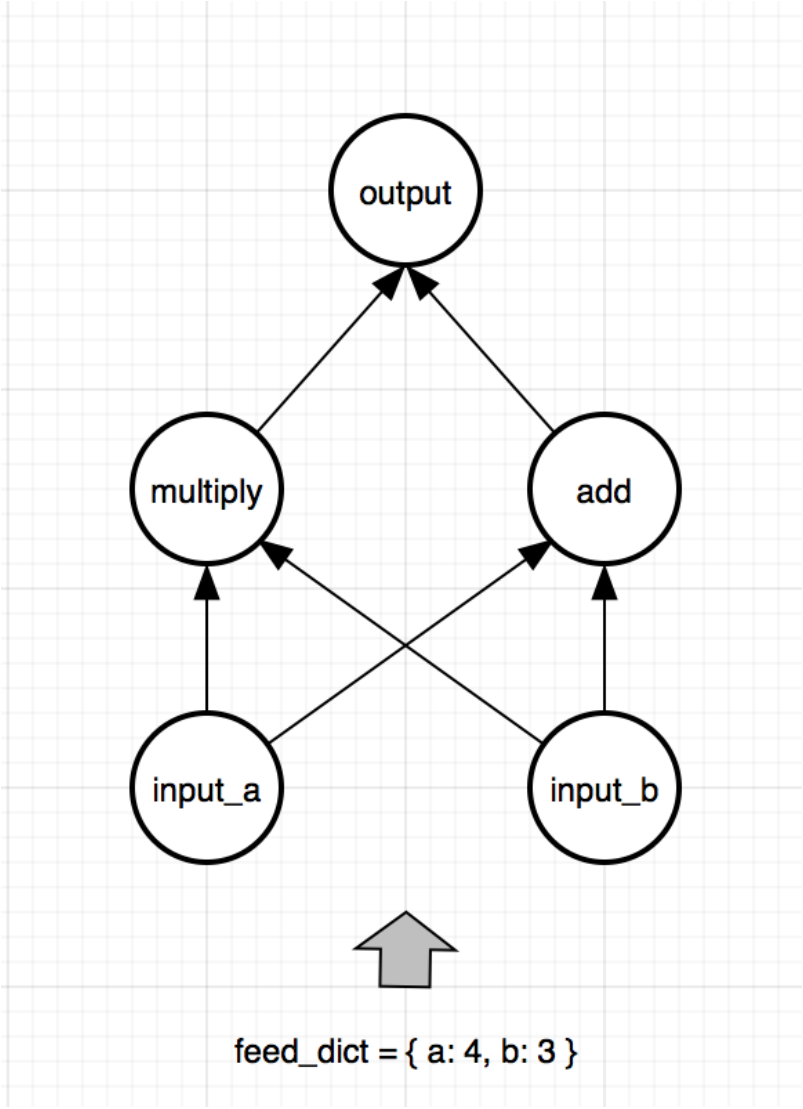
The first graph for TensorFlow

- nodes represent computation
- links represent data flow, which can have n-diamension, 0=scale, 1=vector, 2=matrix.

TensorFlow provides a placeholder operation that must be fed with data on execution. See doc (https://www.tensorflow.org/versions/r0.9/api_docs/python/io_ops.html) for details. We can set default value for placeholder too.

1-Define a Computational Graph

output = c + d = (a + b) + (a * b)



In [ ]:
```
# `tf.placeholder` creates an "input" node- we will give it value when we ru

a = tf.placeholder(tf.int32, name="input_a")
b = tf.placeholder(tf.int32, name="input_b")

# there are two nodes, named input_a and input_b, datatype is int32. They wi
```

In [ ]:
```
# `tf.add` is operation "add", and we call/name the node "add", may be using
c = tf.add(a, b, name="add")   # this give us c = a + b
```

In [ ]:
```
# `tf.mul` creates a multiplication node
d = tf.mul(a, b, name="multiply")   # this gives us d = a * b
```

In [ ]:
```

```

```
# Add up the results of the previous two nodes
out = tf.add(c, d, name="output")   # this gives us
```

In [ ]:
```
# OPTIONAL
# Create a scalar summary, which will log the value we tell it to when execu
# In this case, we'll tell it to save our output value from `out`
# This works in tandem with our SummaryWriter below
# To create the summary, we pass in two parameters:
# 1. A 'tag', which gives a label to the data
# 2. The value(s) we'd like to save
# We also give a `name` to the summary itself (does not affect behavior)
out_summary = tf.scalar_summary("output", out, name="output_summary")
```

2-Run the Graph

- Start a tf.Session to launch the graph
- Setup any necessary input values
- Use a tf.train.SummaryWriter to write information for TensorBoard (Recommended)
- Use Session.run() to compute values from the graph

In [ ]:
```
# Start a session
sess = tf.Session()
```

In [ ]:
```
# Create a "feed_dict" dictionary to define input values
# Keys to dictionary are handles to our placeholders
# Values to dictionary are values we'd like to feed in
feed_dict = { a: 4, b: 3 }
```

In [ ]:
```
# OPTIONAL
# Opens a `SummaryWriter` object, which can write stats about the graph to d
# We pass in two parameters into the SummaryWriter constructor
# The first is a string, specifies a directory to write to.
#   (Note: `TB_DIR` was specified earlier. "TB" stands for TensorBoard
# The second parameter passes in our graph. This allows us to visualize our
writer = tf.train.SummaryWriter(TB_DIR, graph=sess.graph)
```

In [ ]:
```
# Execute the graph using `sess.run()`, passing in two parameters:
# The first parameter, `fetches` lists which node(s) we'd like to receive as
# The second parameter, `feed_dict`, feeds in key-value pairs
#   to input or override the value of nodes
# In this case, we run both the output value, as well as its scalar summary
result, summary = sess.run([out, out_summary], feed_dict=feed_dict)

# Print output with fun formatting
print("(({0}*{1}) + ({0}+{1})) = ".format(feed_dict[a], feed_dict[b]) + str(
```

In [ ]:
```
# We add the summary to our SummaryWriter, which writes the data to disk:
# Normally, these summaries are used to generate statistics over time
# TensorBoard doesn't do well visualizing single points, so we fake a "globa
# With two points, it will generate a line
writer.add_summary(summary, global_step=0)
writer.add_summary(summary, global_step=100)
```

In [ ]:
```
# We're done! Close down our Session and SummaryWriter to tidy up.
sess.close()
writer.close()
```

3-Visualize with Tensorboard

In [ ]:
```
# Start TensorBoard
start_tensorboard()
```

In [ ]:
```
# Once you are done, stop TensorBoard

# stop_tensorboard()
```

In [ ]:
```

```