HU Extension Assignment 10 E63 Big Data Analytics

Handed out: 11/03/2017 Due by 9:30AM EST on Saturday, 11/11/2017

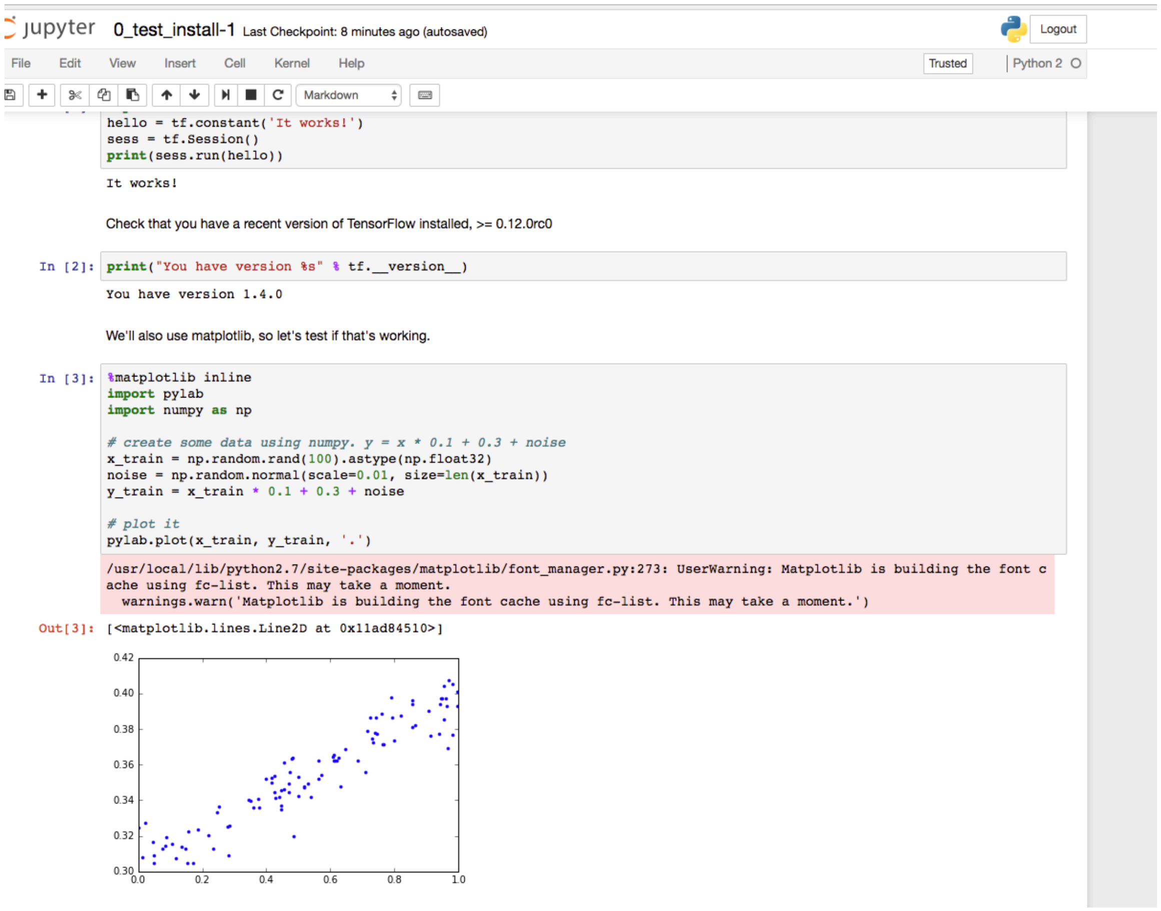
You are welcome to implement TensorFlow problems in this problem set in any of supported languages.

Problem 1. Install newest release of TensorFlow 1.4 on the operating system of your choice. Use installation instructions on https://www.tensorflow.org site and instructions on https://github.com/tensorflow/tensorflow . If you know what you are doing install TensorFlow for GPU. Otherwise, install TensorFlow for CPU. Use attach Jupyter notebook: `0_test_install.ipynb` to demonstrate that TensorFlow is properly installed. Please document all installation steps including the version of Python you are using. (15%)

Installing Tensorflow

```
swaite@Rmt-mac-swaite:/usr/local/lib/python2.7/site-packages|
⇒  brew unlink python && brew link python
----------------------------------------------------------------
Had an issue with installing and upgrading python for tensorflow.
Probably an issue with my computer.

error: [Errno 13] Permission denied:
'/usr/local/lib/python2.7/site-
packages/pkg_resources/__init__.py'
Warning: The post-install step did not complete successfully
FIX
swaite@Rmt-mac-swaite:/usr/local/bin|
⇒  sudo chown -R `whoami` /usr/local/lib/python2.7/site-packages/
---------------------------------------------------------------
swaite@Rmt-mac-swaite:/Library/Frameworks|
⇒  sudo easy_install --upgrade pip
swaite@Rmt-mac-swaite:/Library/Frameworks|
⇒  sudo easy_install --upgrade six
swaite@Rmt-mac-swaite:/Library/Frameworks|
⇒  sudo pip install --upgrade
https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-
1.4.0-py2-none-any.whl
swaite@Rmt-mac-swaite:/Library/Frameworks|
⇒  sudo pip install --upgrade protobuf
swaite@Rmt-mac-swaite:/Library/Frameworks|
⇒  sudo pip install google
```

Running TensorFlow



```
hello = tf.constant('It works!')
sess = tf.Session()
print(sess.run(hello))
```

```
It works!
```

Check that you have a recent version of TensorFlow installed, >= 0.12.0rc0

```
In [2]: print("You have version %s" % tf.__version__)
```

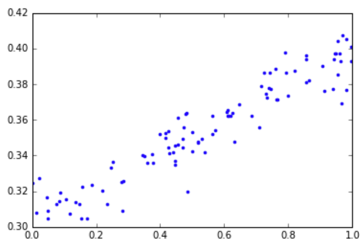
```
You have version 1.4.0
```

We'll also use matplotlib, so let's test if that's working.

```
In [3]: %matplotlib inline
        import pylab
        import numpy as np

        # create some data using numpy. y = x * 0.1 + 0.3 + noise
        x_train = np.random.rand(100).astype(np.float32)
        noise = np.random.normal(scale=0.01, size=len(x_train))
        y_train = x_train * 0.1 + 0.3 + noise

        # plot it
        pylab.plot(x_train, y_train, '.')
```

```
/usr/local/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarning: Matplotlib is building the font c
ache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a moment.')
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x11ad84510>]
```



Problem 2. Construct a simple neural network (a network of logistic units) which will implement (X1 XOR X2) AND X3 function. Choose weights (θ_i-s) of all dendritic inputs and bias inputs. Demonstrate that your network works by presenting the truth table. Present your network by a simple graph. You can produce the graph in any way convenient including pan and paper. (25%)

slide 35 & 36

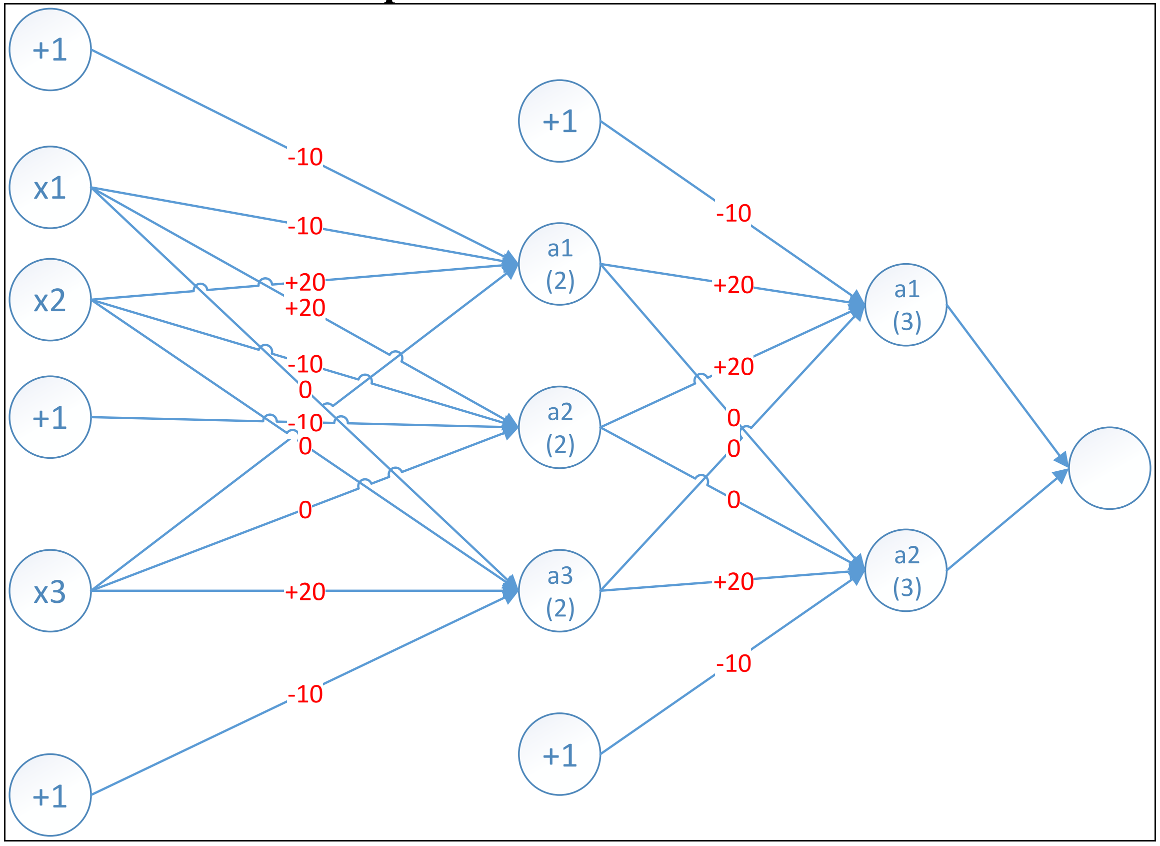
= (X1 XOR X2) AND X3

Logic gates:

= (X1 XOR X2) AND X3

= ((X1 AND NOT X2) OR (NOT X1 AND X2)) AND X3 (A1(2) OR A2(2)) AND X3
= A1(3) AND A2(3) = H0(X)

Neural Network Graph:



Truth table:

x1	x2	X3	a1(2)	a2(2)	a3(2)	a1(3)	a2(3)	h0(x)
0	0	0	0	0	0	0	0	**0**
0	0	1	0	0	1	0	1	**0**
0	1	0	1	0	0	1	0	**0**
1	0	0	0	1	0	1	0	**0**
0	1	1	1	0	1	1	1	**1**
1	1	0	0	0	0	0	0	**0**
1	0	1	0	1	1	1	1	**1**
1	1	1	0	0	1	0	1	**0**

Problem 3. Determine the value of number e = 2.7183… to 6 decimal places using Taylor expansion. Export the TensorBoard graph of your process. Perform similar calculation using expression for e as $\lim_{n \to \infty}(1 + \frac{1}{n})^n$. Again export the TensorBoard graph of you process. Provide working code for both approaches.
(25%)

Notes For Myself

- http://iamtrask.github.io/2015/07/12/basic-python-network/
- https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1
- http://firsttimeprogrammer.blogspot.com/2015/03/taylor-series-with-python-and-sympy.html

CODE

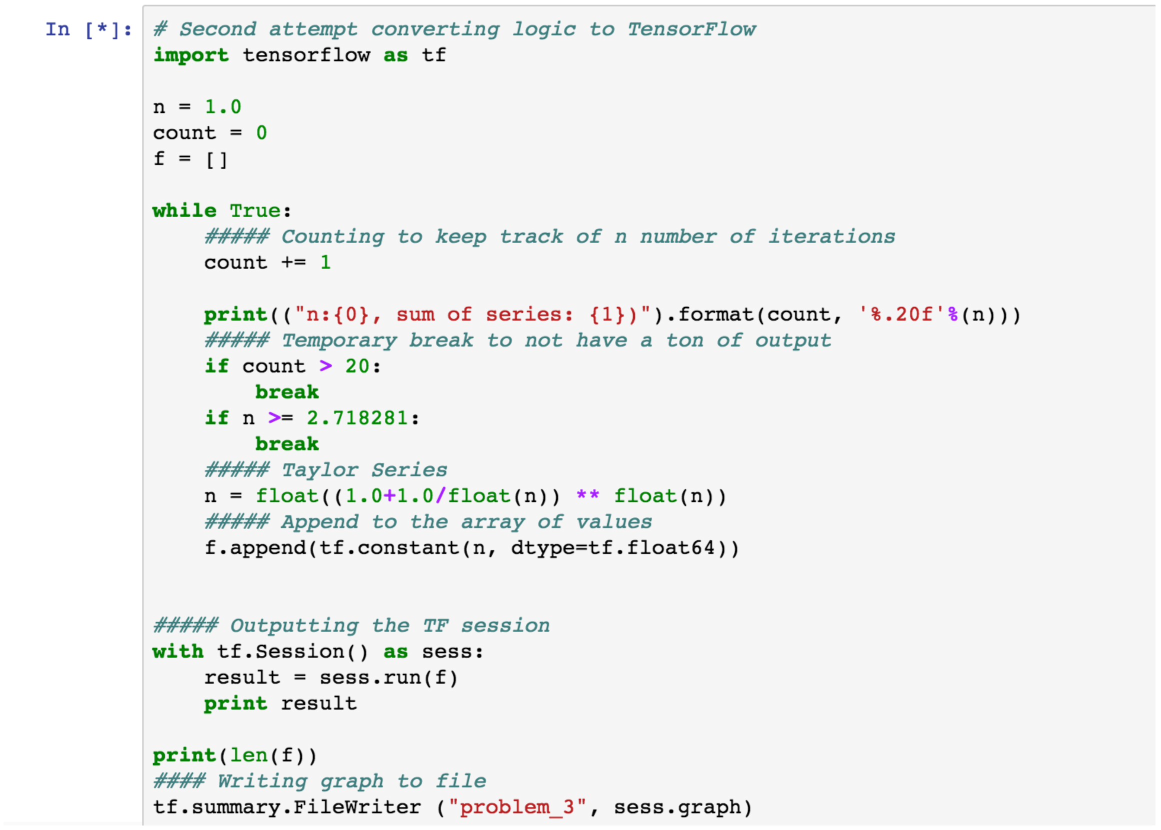
Please see problem-3.ipynb

Python Attempt

```
In [265]: # Attempt at problem in Python first
          n = 1.0
          count = 1
          while True:
              print(("n:{0}, sum of series: {1})").format(count, '%.20f'%(n)))
              print num_after_point(n)
              if n >= 2.718281:
                  break
              # Temporary break to not have a ton of output
              if count > 20:
                  break
              n = float((1.0+1.0/float(n)) ** float(n))
              count += 1
```

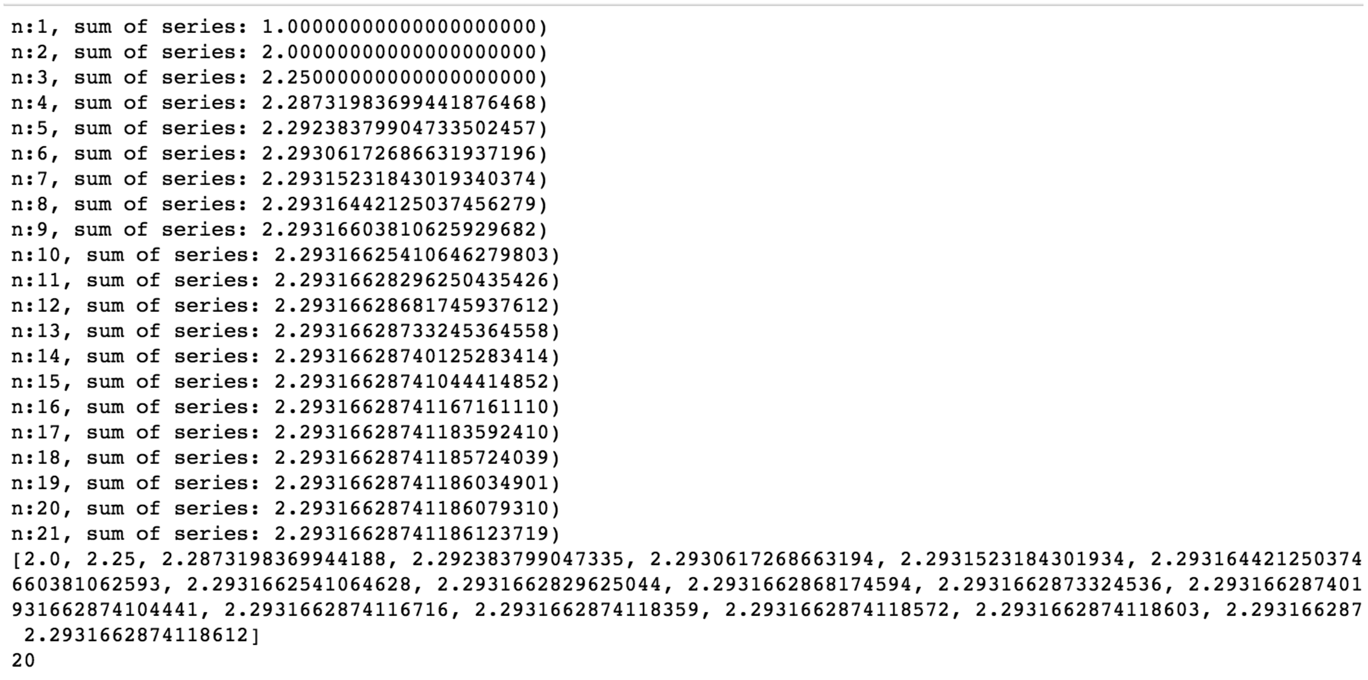
```
n:1, sum of series: 1.00000000000000000000)
1
n:2, sum of series: 2.00000000000000000000)
1
n:3, sum of series: 2.25000000000000000000)
2
n:4, sum of series: 2.28731983699441876468)
11
n:5, sum of series: 2.29238379904733502457)
11
n:6, sum of series: 2.29306172686631937196)
11
n:7, sum of series: 2.29315231843019340374)
11
n:8, sum of series: 2.29316442125037456279)
11
n:9, sum of series: 2.29316603810625929682)
11
n:10, sum of series: 2.29316625410646279803)
11
```

Tensorflow Attempt



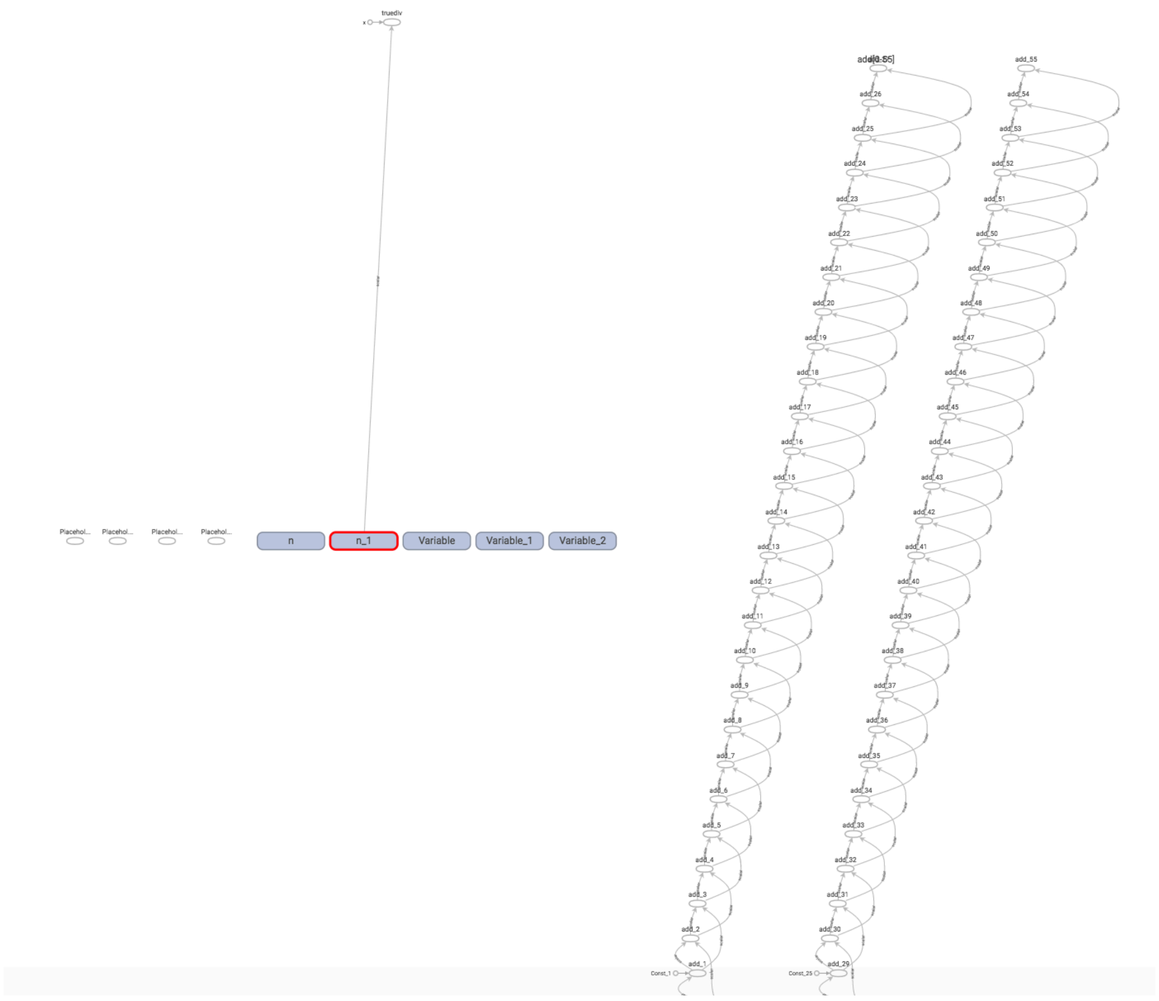
```
In [*]: # Second attempt converting logic to TensorFlow
        import tensorflow as tf

        n = 1.0
        count = 0
        f = []

        while True:
            ##### Counting to keep track of n number of iterations
            count += 1

            print(("n:{0}, sum of series: {1})").format(count, '%.20f'%(n)))
            ##### Temporary break to not have a ton of output
            if count > 20:
                break
            if n >= 2.718281:
                break
            ##### Taylor Series
            n = float((1.0+1.0/float(n)) ** float(n))
            ##### Append to the array of values
            f.append(tf.constant(n, dtype=tf.float64))


        ##### Outputting the TF session
        with tf.Session() as sess:
            result = sess.run(f)
            print result

        print(len(f))
        #### Writing graph to file
        tf.summary.FileWriter ("problem_3", sess.graph)
```

Output of TensorFlow attempt



```
n:1, sum of series: 1.00000000000000000000)
n:2, sum of series: 2.00000000000000000000)
n:3, sum of series: 2.25000000000000000000)
n:4, sum of series: 2.28731983699441876468)
n:5, sum of series: 2.29238379904733502457)
n:6, sum of series: 2.29306172686631937196)
n:7, sum of series: 2.29315231843019340374)
n:8, sum of series: 2.29316442125037456279)
n:9, sum of series: 2.29316603810625929682)
n:10, sum of series: 2.29316625410646279803)
n:11, sum of series: 2.29316628296250435426)
n:12, sum of series: 2.29316628681745937612)
n:13, sum of series: 2.29316628733245364558)
n:14, sum of series: 2.29316628740125283414)
n:15, sum of series: 2.29316628741044414852)
n:16, sum of series: 2.29316628741167161110)
n:17, sum of series: 2.29316628741183592410)
n:18, sum of series: 2.29316628741185724039)
n:19, sum of series: 2.29316628741186034901)
n:20, sum of series: 2.29316628741186079310)
n:21, sum of series: 2.29316628741186123719)
[2.0, 2.25, 2.2873198369944188, 2.292383799047335, 2.2930617268663194, 2.2931523184301934, 2.293164421250374
660381062593, 2.2931662541064628, 2.2931662829625044, 2.2931662868174594, 2.2931662873324536, 2.293166287401
931662874104441, 2.2931662874116716, 2.2931662874118359, 2.2931662874118572, 2.2931662874118603, 2.293166287
 2.2931662874118612]
20
```

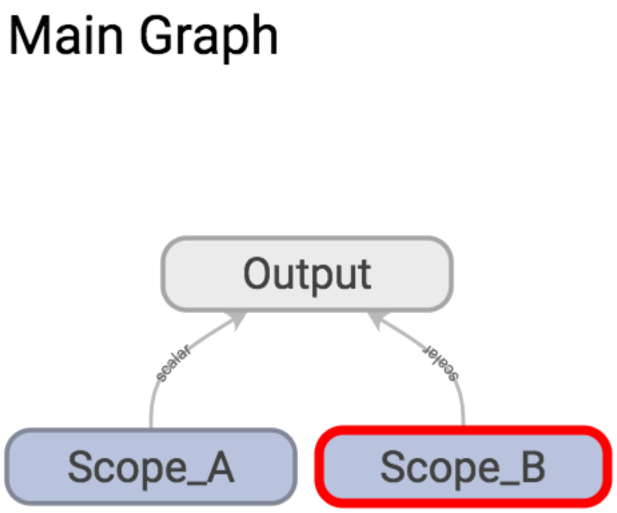
TenorBoard Graph

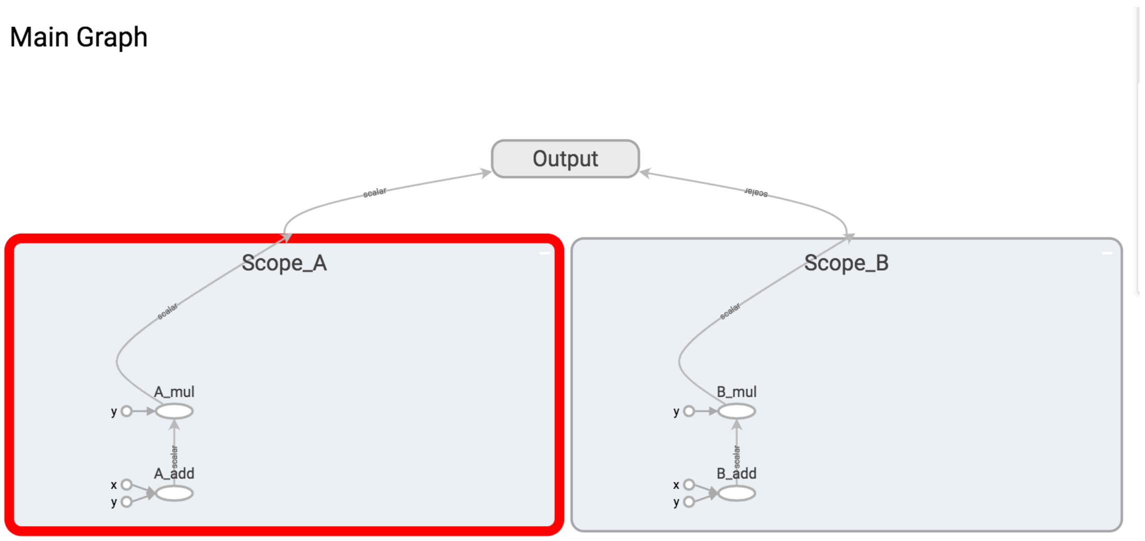


Problem 4. When I tried running code on page 63 on my notes for lecture 10, the resulting TensorBoard graph was not entirely identical with the graph on page 64. Please fix the code on page 63 in order to produce the graph identical to the graph on page 64.
(15%)

CODE

```
import tensorflow as tf

with tf.name_scope("Scope_A"):
    a = tf.add(1, 2, name="A_add")
    b = tf.multiply(a, 3, name="A_mul")
with tf.name_scope("Scope_B"):
    c = tf.add(4, 5, name="B_add")
    d = tf.multiply(c, 6, name="B_mul")

with tf.name_scope("Output"):
    e = tf.add(b, d, name="output")

writer = tf.summary.FileWriter('logs',graph=tf.get_default_graph())
writer.close()
```





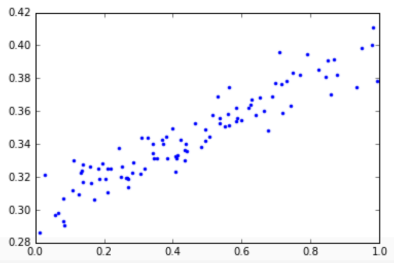
Problem 5. Please examine attached Jupyter notebook `2_linear_regression.ipynb`. As you are running its cells, the notebook will complain about non-existent API calls. This notebook was written in an earlier version of TensorFlow API and some calls changed their names. Fix all code by replacing older calls with calls in TF 1.5. Uncomment all optional (print) lines.

CODE CAN BE FOUND IN PROBLEM-5.IPYNB

In [2]:
```
sess = None
def resetSession():
    tf.reset_default_graph()
    global sess
    if sess is not None: sess.close()
    sess = tf.InteractiveSession()
```

In [3]:
```
resetSession()

# Create input data using NumPy. y = x * 0.1 + 0.3 + noise
x_train = np.random.rand(100).astype(np.float32)
noise = np.random.normal(scale=0.01, size=len(x_train))
y_train = x_train * 0.1 + 0.3 + noise

# Uncomment the following line to plot our input data.
pylab.plot(x_train, y_train, '.')
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x11905efd0>]
```



In [4]:
```
# Create some fake evaluation data
x_eval = np.random.rand(len(x_train)).astype(np.float32)
noise = np.random.normal(scale=0.01, size=len(x_train))
y_eval = x_eval * 0.1 + 0.3 + noise
```

In [5]:
```
# Build inference graph.
# Create Variables W and b that compute y_data = W * x_data + b
W = tf.Variable(tf.random_normal([1]), name='weights')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Uncomment the following lines to see what W and b are.
print(W)
print(b)

# Create a placeholder we'll use later to feed x's into the graph for training and eval.
# shape=[None] means we can put in any number of examples.
# This is used for minibatch training, and to evaluate a lot of examples at once.
x = tf.placeholder(shape=[None], dtype=tf.float32, name='x')

# Uncomment this line to see what x is
print(x)

# This is the same as tf.add(tf.mul(W, x), b), but looks nicer
y = W * x + b
print(y)
```

```
<tf.Variable 'weights:0' shape=(1,) dtype=float32_ref>
<tf.Variable 'bias:0' shape=(1,) dtype=float32_ref>
Tensor("x:0", shape=(?,), dtype=float32)
Tensor("add:0", shape=(?,), dtype=float32)
```

At this point, we have:

- x_train: x input features
- y_train: observed y for each x that we will train on
- x_eval, y_eval: Same as above, but a smaller set that we will not train on, and instead evaluate our effectiveness.

```
In [6]: # Write the graph so we can look at it in TensorBoard
        # https://www.tensorflow.org/versions/r0.12/how_tos/summaries_and_tensorboard/index.html

        sw = tf.summary.FileWriter('summaries/', graph=tf.get_default_graph())
```

```
In [7]: # Create a placeholder we'll use later to feed the correct y value into the graph
        y_label = tf.placeholder(shape=[None], dtype=tf.float32, name='y_label')
        print (y_label)

        Tensor("y_label:0", shape=(?,), dtype=float32)
```

```
In [8]: # Build training graph.
        loss = tf.reduce_mean(tf.square(y - y_label))  # Create an operation that calculates loss.
        optimizer = tf.train.GradientDescentOptimizer(0.5)  # Create an optimizer.
        train = optimizer.minimize(loss)  # Create an operation that minimizes loss.

        # Uncomment the following 3 lines to see what 'loss', 'optimizer' and 'train' are.
        print("loss:", loss)
        print("optimizer:", optimizer)
        print("train:", train)

        loss: Tensor("Mean:0", shape=(), dtype=float32)
        optimizer: <tensorflow.python.training.gradient_descent.GradientDescentOptimizer object at 0x11900fc10>
        train: name: "GradientDescent"
        op: "NoOp"
        input: "^GradientDescent/update_weights/ApplyGradientDescent"
        input: "^GradientDescent/update_bias/ApplyGradientDescent"
```

```
In [9]: # Create an operation to initialize all the variables.
        init = tf.global_variables_initializer()
        print(init)
        sess.run(init)

        name: "init"
        op: "NoOp"
        input: "^weights/Assign"
        input: "^bias/Assign"
```

```
In [10]: # Uncomment the following line to see the initial W and b values.
         print(sess.run([W, b]))

         [array([-0.28865173], dtype=float32), array([ 0.16082609], dtype=float32)]
```

```
In [11]: # Uncomment these lines to test that we can compute a y from an x (without having trained anything).
         # x must be a vector, hence [3] not just 3.
         x_in = [3]
         sess.run(y, feed_dict={x: x_in})

Out[11]: array([-0.70512915], dtype=float32)
```

```
In [12]: # Calculate loss on the evaluation data before training
         def eval_loss():
             return sess.run(loss, feed_dict={x: x_eval, y_label: y_eval})
         eval_loss()

Out[12]: 0.12074754
```

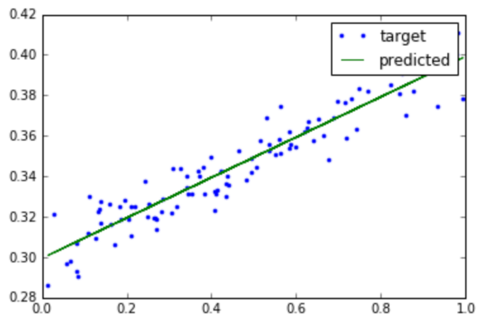
```
In [13]: # Track of how loss changes, so we can visualize it in TensorBoard
         tf.summary.scalar('loss', loss)
         summary_op = tf.summary.merge_all()
```

In [14]:
```
# Perform training.
for step in range(201):
    # Run the training op; feed the training data into the graph
    summary_str, _ = sess.run([summary_op, train], feed_dict={x: x_train, y_label: y_train})
    sw.add_summary(summary_str, step)
    # Uncomment the following two lines to watch training happen real time.
    if step % 20 == 0:
        print(step, sess.run([W, b]))
```

```
0 [array([-0.12323727], dtype=float32), array([ 0.47364864], dtype=float32)]
20 [array([ 0.01639085], dtype=float32), array([ 0.33894798], dtype=float32)]
40 [array([ 0.0719093], dtype=float32), array([ 0.31264591], dtype=float32)]
60 [array([ 0.09046041], dtype=float32), array([ 0.30385727], dtype=float32)]
80 [array([ 0.09665915], dtype=float32), array([ 0.30092058], dtype=float32)]
100 [array([ 0.09873041], dtype=float32), array([ 0.2999393], dtype=float32)]
120 [array([ 0.09942251], dtype=float32), array([ 0.29961142], dtype=float32)]
140 [array([ 0.09965377], dtype=float32), array([ 0.29950187], dtype=float32)]
160 [array([ 0.09973103], dtype=float32), array([ 0.29946527], dtype=float32)]
180 [array([ 0.09975686], dtype=float32), array([ 0.29945302], dtype=float32)]
200 [array([ 0.09976549], dtype=float32), array([ 0.29944894], dtype=float32)]
```

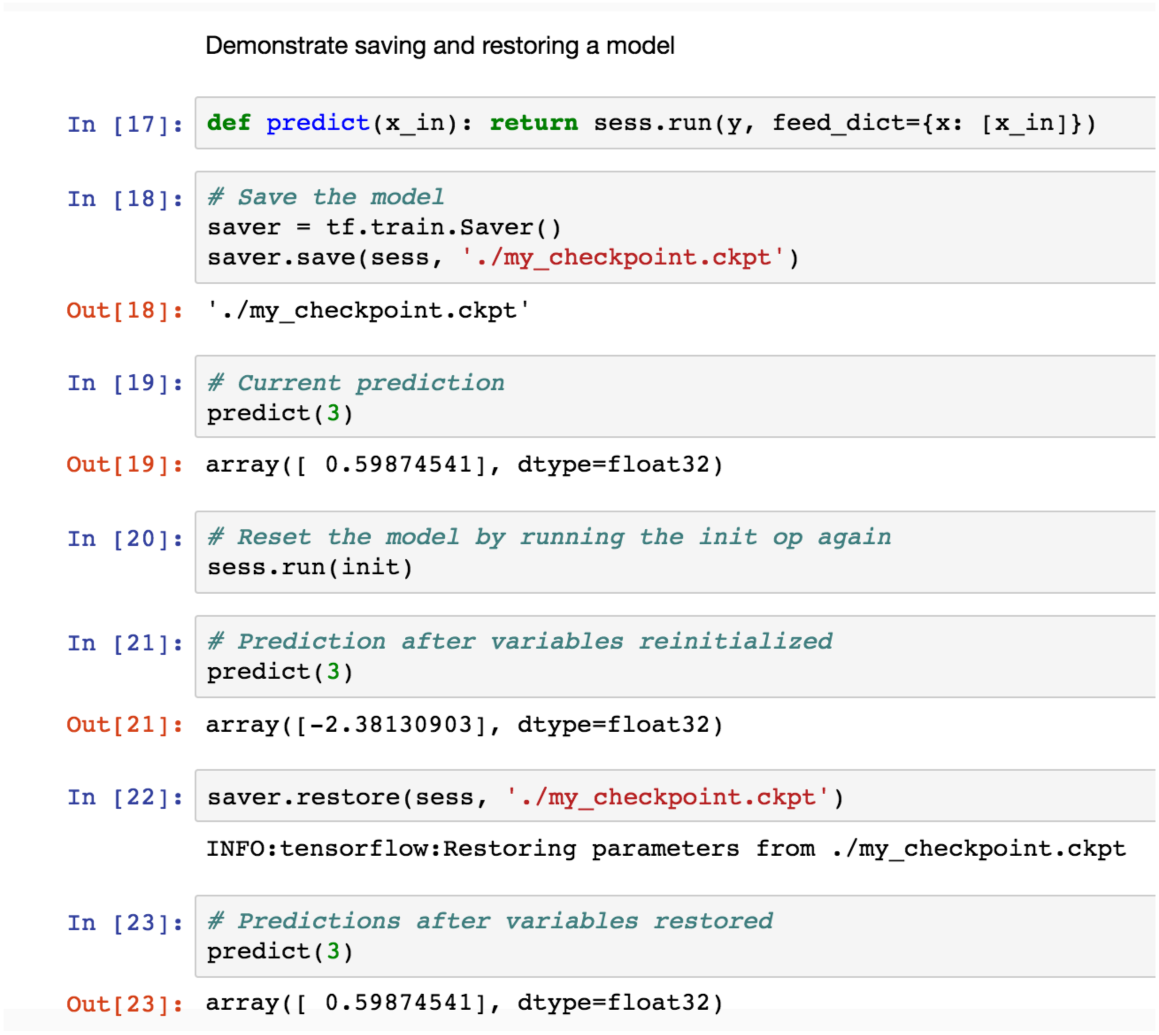
In [15]:
```
# Uncomment the following lines to plot the predicted values
pylab.plot(x_train, y_train, '.', label="target")
pylab.plot(x_train, sess.run(y, feed_dict={x: x_train, y_label: y_train}), label="predicted")
pylab.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x1195e7850>
```



In [16]:
```
# Check accuracy on eval data after training
eval_loss()
```

Out[16]:
```
6.4712018e-05
```



Demonstrate saving and restoring a model

```
In [17]: def predict(x_in): return sess.run(y, feed_dict={x: [x_in]})
```

```
In [18]: # Save the model
         saver = tf.train.Saver()
         saver.save(sess, './my_checkpoint.ckpt')

Out[18]: './my_checkpoint.ckpt'
```

```
In [19]: # Current prediction
         predict(3)

Out[19]: array([ 0.59874541], dtype=float32)
```

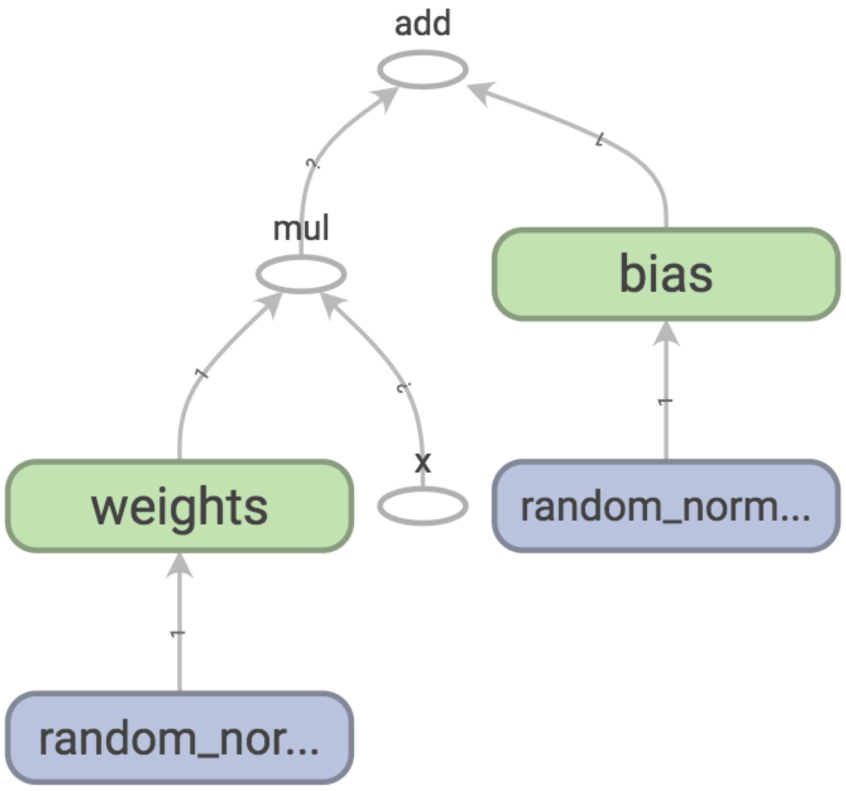
```
In [20]: # Reset the model by running the init op again
         sess.run(init)
```

```
In [21]: # Prediction after variables reinitialized
         predict(3)

Out[21]: array([-2.38130903], dtype=float32)
```

```
In [22]: saver.restore(sess, './my_checkpoint.ckpt')

         INFO:tensorflow:Restoring parameters from ./my_checkpoint.ckpt
```

```
In [23]: # Predictions after variables restored
         predict(3)

Out[23]: array([ 0.59874541], dtype=float32)
```

Provide a copy of this notebook with all intermediate results and the image of TensorFlow graph as captured by the TensorBoard.
(20%)



Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board.