Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 17

Tensorflow essentials

Overview

- Understanding the TensorFlow workflow
- Creating interactive notebooks with Jupyter
- Visualizing algorithms by using TensorBoard

Overview

Computing the inner product of two vectors without using a library

```
revenue = 0
for price, amount in zip(prices, amounts):
    revenue += price * amount
```

Computing the inner product using NumPy

```
import numpy as np
revenue = np.dot(prices, amounts)
```

Ensuring that TensorFlow works

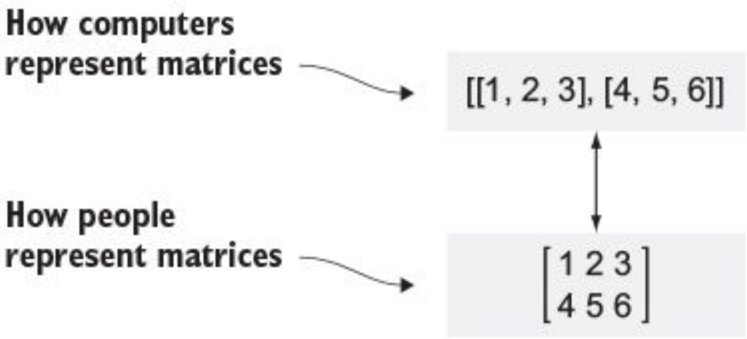
Once you install tensorflow in your machine, you should be able to import it:

```
import tensorflow as tf
```

Detailed documentation about various functions for the Python and C++ APIs are available at www.tensorflow.org/api_docs/.

Representing tensors

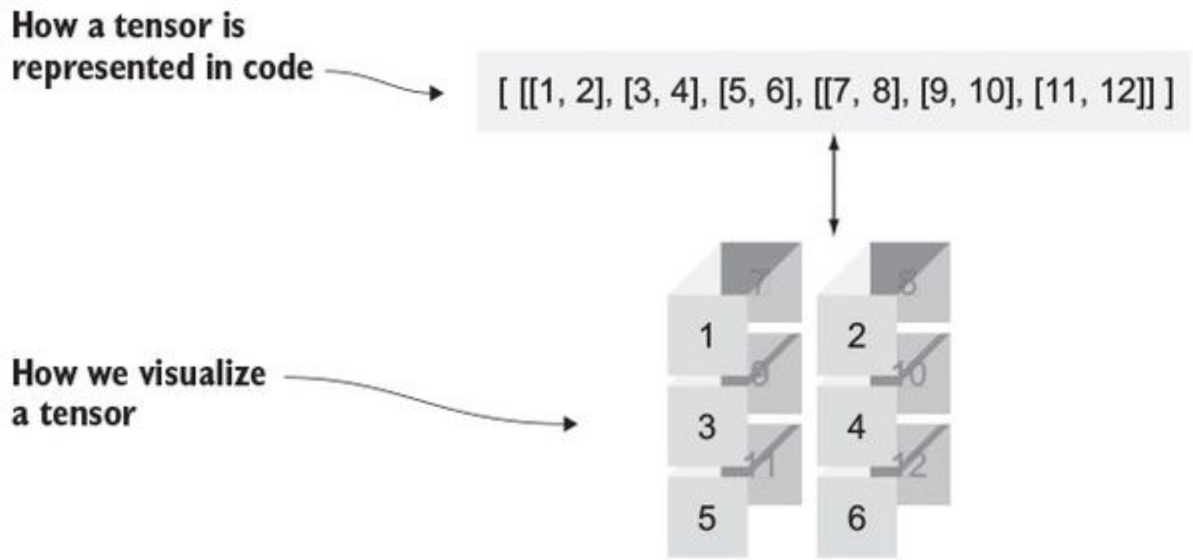
The matrix in the lower half of the diagram is a visualization from its compact code notation in the upper half of the diagram. This form of notation is a common paradigm in most scientific computing libraries.



A tensor is a generalization of a matrix that specifies an element by an arbitrary number of indices.

Representing tensors

This tensor can be thought of as multiple matrices stacked on top of each other. To specify an element, you must indicate the row and column, as well as which matrix is being accessed. Therefore, the rank of this tensor is 3.



Representing tensors

Different ways to represent tensors

```
import tensorflow as tf
import numpy as np                                1

m1 = [[1.0, 2.0],
      [3.0, 4.0]]                         2

m2 = np.array([[1.0, 2.0],
               [3.0, 4.0]], dtype=np.float32)      2

m3 = tf.constant([[1.0, 2.0],
                  [3.0, 4.0]])                 2

print(type(m1))                             3
print(type(m2))                             3
print(type(m3))                             3

t1 = tf.convert_to_tensor(m1, dtype=tf.float32)    4
t2 = tf.convert_to_tensor(m2, dtype=tf.float32)    4
t3 = tf.convert_to_tensor(m3, dtype=tf.float32)    4

print(type(t1))                             5
print(type(t2))                             5
print(type(t3))                             5
```

1 You'll use NumPy matrices in TensorFlow.
2 Defines a 2 × 2 matrix in three ways
3 Prints the type for each matrix
4 Creates tensor objects out of the various types
5 Notice that the types will be the same now.
The code outputs the following three times:

```
<class 'tensorflow.python.framework.ops.Tensor'>
```

Representing tensors

Creating tensors

- 1 Defines a 2 × 1 matrix
- 2 Defines a 1 × 2 matrix
- 3 Defines a rank-3 tensor
- 4 Try printing the tensors.

```
import tensorflow as tf

m1 = tf.constant([[1., 2.]])      1

m2 = tf.constant([[1],
                  [2]])           2

m3 = tf.constant([ [[1,2],
                    [3,4],
                    [5,6]],
                    [[7,8],
                    [9,10],
                    [11,12]] ])   3

print(m1)                         4
print(m2)                         4
print(m3)                         4
```

Representing tensors

The code produces the following output:

```
Tensor( "Const:0",
        shape=TensorShape([Dimension(1), Dimension(2)]),
        dtype=float32 )
Tensor( "Const_1:0",
        shape=TensorShape([Dimension(2), Dimension(1)]),
        dtype=int32 )
Tensor( "Const_2:0",
        shape=TensorShape([Dimension(2), Dimension(3),
Dimension(2)]),
        dtype=int32 )
```

Representing tensors

Exercise 1

Initialize a 500×500 tensor with all elements equaling 0.5.

Representing tensors

Exercise 1

Initialize a 500 × 500 tensor with all elements equaling 0.5.

```
tf.ones([500,500]) * 0.5
```

Creating operators

Using the negation operator

```
import tensorflow as tf

x = tf.constant([[1, 2]])         1
negMatrix = tf.negative(x)        2
print(negMatrix)                  3
```

- 1 Defines an arbitrary tensor
- 2 Negates the tensor
- 3 Prints the object, generates the following output:

```
Tensor("Neg:0", shape=TensorShape([Dimension(1), Dimension(2)]), dtype=int32)
```

Creating operators

Useful TensorFlow operators:

- tf.add(x, y)—Adds two tensors of the same type, x + y
- tf.subtract(x, y)—Subtracts tensors of the same type, x – y
- tf.multiply(x, y)—Multiplies two tensors element-wise
- tf.pow(x, y)—Takes the element-wise x to the power of y
- tf.exp(x)—Equivalent to pow(e, x), where e is Euler's number (2.718 ...)
- tf.sqrt(x)—Equivalent to pow(x, 0.5)
- tf.div(x, y)—Takes the element-wise division of x and y
- tf.truediv(x, y)—Same as tf.div, except casts the arguments as a float
- tf.floordiv(x, y)—Same as truediv, except rounds down the final answer into an integer
- tf.mod(x, y)—Takes the element-wise remainder from division

Creating operators

Exercise 2

Use the TensorFlow operators you’ve learned so far to produce the Gaussian distribution (also known as the normal distribution).

Creating operators

Answer:

```
from math import pi
mean = 0.0
sigma = 1.0
(tf.exp(tf.negative(tf.pow(x – mean, 2.0) /
               (2.0 * tf.pow(sigma, 2.0) ))) *
 (1.0 / (sigma * tf.sqrt(2.0 * pi) )))
```

Executing operators with sessions

Using a session

```
import tensorflow as tf

x = tf.constant([[1., 2.]])          1
neg_op = tf.negative(x)              2

with tf.Session() as sess:           3
    result = sess.run(neg_op)        4
print(result)                        5
```

- 1 Defines an arbitrary matrix
- 2 Runs the negation operator on it
- 3 Starts a session to be able to run operations
- 4 Tells the session to evaluate neg_op
- 5 Prints the resulting matrix

Executing operators with sessions

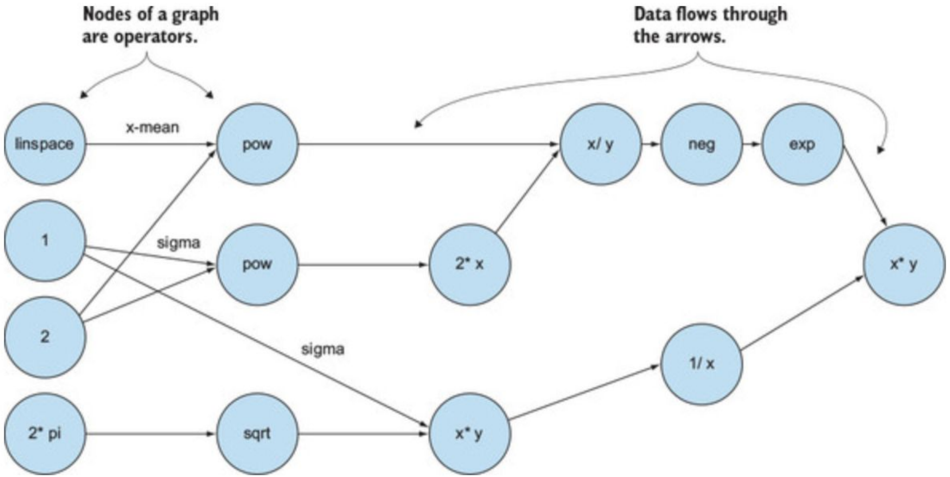
Using the interactive session mode

```
import tensorflow as tf
sess = tf.InteractiveSession()        1

x = tf.constant([[1., 2.]])           2
negMatrix = tf.negative(x)            2

result = negMatrix.eval()             3
print(result)                         4

sess.close()                          5
```

1 Starts an interactive session so the sess variable no longer needs to be passed around
2 Defines an arbitrary matrix and negates it
3 You can now evaluate negMatrix without explicitly specifying a session.
4 Prints the negated matrix
5 Remember to close the session to free up resources.

Executing operators with sessions

The graph represents the operations needed to produce a Gaussian distribution. The links between the nodes represent how data flows from one operation to the next. The operations themselves are simple, but the complexity arises from the way they intertwine.



Executing operators with sessions

Logging a session

```
import tensorflow as tf

x = tf.constant([[1., 2.]])                                   1
negMatrix = tf.negative(x)                                    1

with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess: 2
    result = sess.run(negMatrix)                              3

print(result)                                                 4
```

1 Defines a matrix and negates it
2 Starts the session with a special config passed into the constructor to enable logging
3 Evaluates negMatrix
4 Prints the resulting value

```
Neg: /job:localhost/replica:0/task:0/cpu:0
```

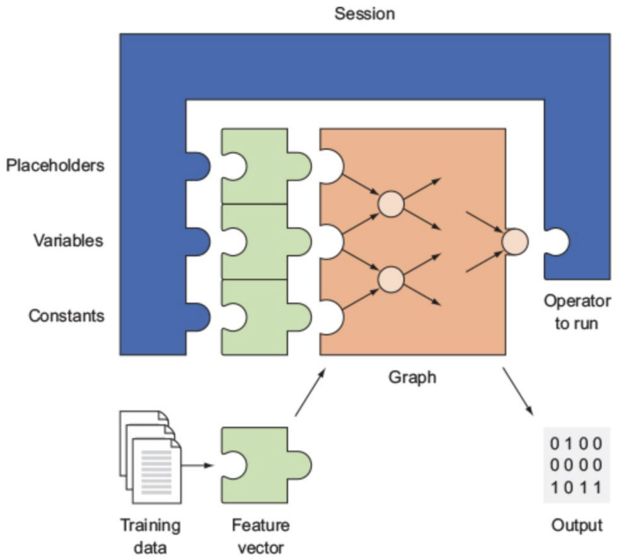
Executing operators with sessions

Here's a quick overview of these three types of values:

- Placeholder—A value that's unassigned but will be initialized by the session wherever it's run. Typically, placeholders are the input and output of your model.
- Variable—A value that can change, such as parameters of a machine-learning model. Variables must be initialized by the session before they're used.
- Constant—A value that doesn't change, such as hyperparameters or settings.

Executing operators with sessions

The session dictates how the hardware will be used to process the graph most efficiently. When the session starts, it assigns the CPU and GPU devices to each of the nodes. After processing, the session outputs data in a usable format, such as a NumPy array. A session optionally may be fed placeholders, variables, and constants.



Writing code in Jupyter

```
$ cd ~/MyTensorFlowStuff
$ jupyter notebook
```

Writing code in Jupyter

Running the Jupyter Notebook will launch an interactive notebook on http://localhost:8888.

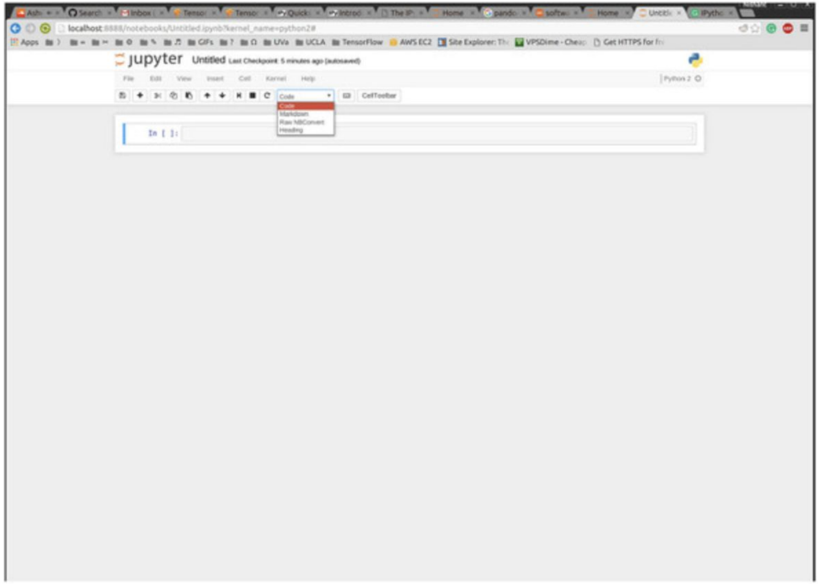
Writing code in Jupyter

There are three common ways to evaluate cells:

- Pressing Shift-Enter on a cell executes the cell and highlights the next cell below.
- Pressing Ctrl-Enter maintains the cursor on the current cell after executing it.
- Pressing Alt-Enter executes the cell and then inserts a new empty cell directly below.

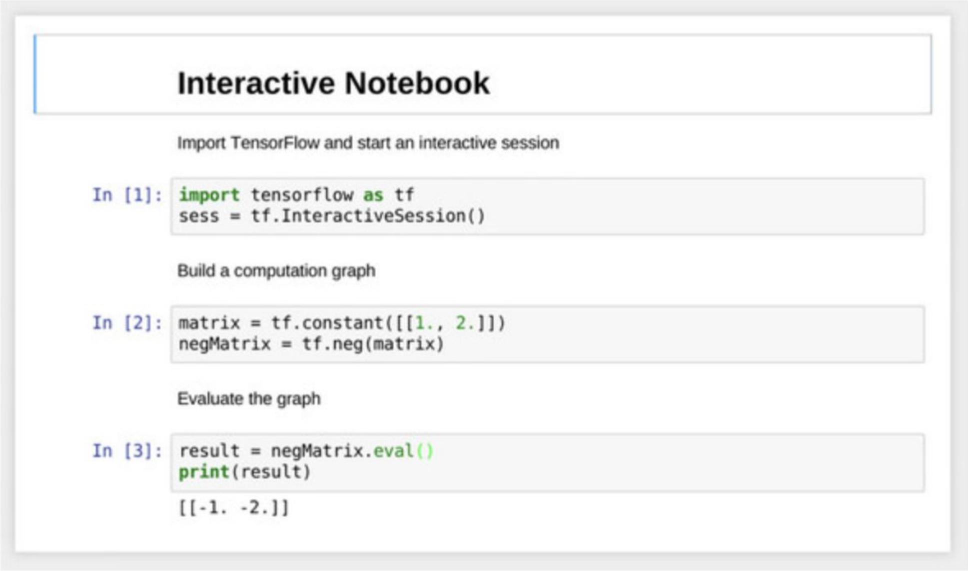
Writing code in Jupyter

The drop-down menu changes the type of cell in the notebook. The Code cell is for Python code, whereas the Markdown code is for text descriptions.



Writing code in Jupyter

An interactive Python notebook presents both code and comments grouped for readability.



Writing code in Jupyter

Exercise 3: If you look closely at the previous figure, you'll notice that it uses tf.neg instead of tf.negative. That's strange. Could you explain why we might have done that?

Writing code in Jupyter

ANSWER: You should be aware that the TensorFlow library changed naming conventions, and you may run into these artifacts when following old TensorFlow tutorials online.

Using variables

Using a variable

```
import tensorflow as tf
sess = tf.InteractiveSession()                    1

raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]     2
spike = tf.Variable(False)                        3
spike.initializer.run()                           4

for i in range(1, len(raw_data)):                 5
    if raw_data[i] - raw_data[i-1] > 5:
        updater = tf.assign(spike, True)          6
        updater.eval()                            6
    else:
        tf.assign(spike, False).eval()
    print("Spike", spike.eval())

sess.close()                                      7
```

1 Starts the session in interactive mode so you won’t need to pass around sess
2 Let’s say you have some raw data like this.
3 Creates a Boolean variable called spike to detect a sudden increase in a series of numbers
4 Because all variables must be initialized, initialize the variable by calling run() on its initializer.
5 Loops through the data (skipping the first element) and updates the spike variable when there’s a significant increase
6 To update a variable, assign it a new value using tf.assign(<var name>, <new value>). Evaluate it to see the change.
7 Remember to close the session after it’ll no longer be used.

Using variables

Expected output:

```
('Spike', False)
('Spike', True)
('Spike', False)
('Spike', False)
('Spike', True)
('Spike', False)
('Spike', True)
```

Saving and loading variables

Saving variables

```
import tensorflow as tf                                   1
sess = tf.InteractiveSession()                            1

raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]             2
spikes = tf.Variable([False] * len(raw_data), name='spikes')        3
spikes.initializer.run()                                  4

saver = tf.train.Saver()                                  5

for i in range(1, len(raw_data)):                         6
    if raw_data[i] - raw_data[i-1] > 5:                   6
        spikes_val = spikes.eval()                        7
        spikes_val[i] = True                              7
        updater = tf.assign(spikes, spikes_val)           7
        updater.eval()                                    8

save_path = saver.save(sess, "spikes.ckpt")               9
print("spikes data saved in file: %s" % save_path)        10

sess.close()
```

1 Imports TensorFlow and enables interactive sessions
2 Let’s say you have a series of data like this.
3 Defines a Boolean vector called spikes to locate a sudden spike in raw data
4 Don’t forget to initialize the variable.
5 The saver op will enable saving and restoring variables. If no dictionary is passed into the constructor, then it saves all variables in the current program.
6 Loop through the data and update the spikes variable when there’s a significant increase.
7 Updates the value of spikes by using the tf.assign function
8 Don’t forget to evaluate the updater; otherwise, spikes won’t be updated.
9 Saves the variable to disk
10 Prints out the relative file path of the saved variables

Saving and loading variables

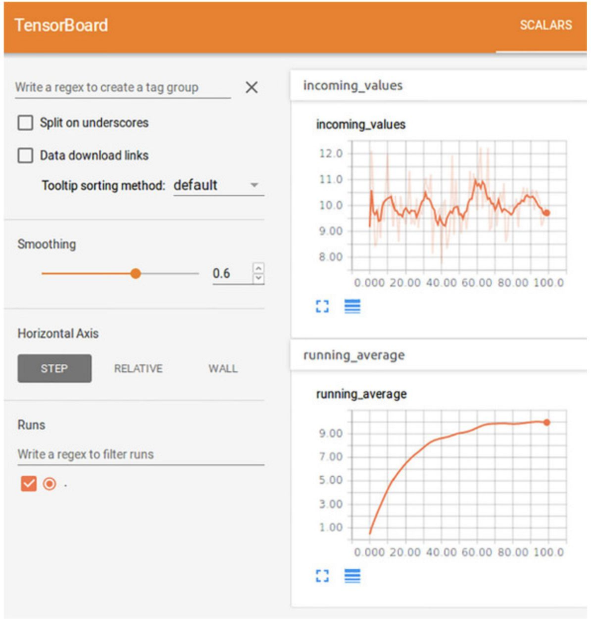
Loading variables

```
import tensorflow as tf
sess = tf.InteractiveSession()

spikes = tf.Variable([False]*8, name='spikes')        1
# spikes.initializer.run()                        2
saver = tf.train.Saver()                          3

saver.restore(sess, "./spikes.ckpt")              4
print(spikes.eval())                              5

sess.close()
```

1 Creates a variable of the same size and name as the saved data
2 You no longer need to initialize this variable because it'll be directly loaded.
3 Creates the saver op to restore saved data
4 Restores data from the spikes.ckpt file
5 Prints the loaded data

Visualizing data using TensorBoard

The summary display in TensorBoard created to calculate average. TensorBoard provides a user-friendly interface to visualize data produced in TensorFlow.



Visualizing data using TensorBoard

Defining the average update operator

```
update_avg = alpha * curr_value + (1 - alpha) * prev_avg          1
```

1 alpha is a tf.constant, curr_value is a placeholder, and prev_avg is a variable.

Visualizing data using TensorBoard

Running iterations of the exponential average algorithm

```
raw_data = np.random.normal(10, 1, 100)

with tf.Session() as sess:
    for i in range(len(raw_data)):
        curr_avg = sess.run(update_avg,
feed_dict={curr_value:raw_data[i]}
        sess.run(tf.assign(prev_avg, curr_avg))
```

Visualizing data using TensorBoard

Filling in missing code to complete the exponential average algorithm

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)                    1

alpha = tf.constant(0.05)                                  2
curr_value = tf.placeholder(tf.float32)                    3
prev_avg = tf.Variable(0.)                                 4
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for i in range(len(raw_data)):                         5
        curr_avg = sess.run(update_avg, feed_dict={curr_value:
raw_data[i]})
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
```

1 Creates a vector of 100 numbers with a mean of 10 and standard deviation of 1
2 Defines alpha as a constant
3 A placeholder is just like a variable, but the value is injected from the session.
4 Initializes the previous average to zero
5 Loops through the data one by one to update the average

Visualizing data using TensorBoard

Annotating with a summary op

```
img = tf.placeholder(tf.float32, [None, None, None, 3])
cost = tf.reduce_sum(...)

my_img_summary = tf.summary.image("img", img)
my_cost_summary = tf.summary.scalar("cost", cost)
```

Visualizing data using TensorBoard

Run the following command to make a directory called logs in the same folder as this source code:

```
$ mkdir logs
```

Run TensorBoard with the location of the logs directory passed in as an argument:

```
$ tensorboard --logdir=./logs
```

Visualizing data using TensorBoard

Writing summaries to view in TensorBoard

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)

alpha = tf.constant(0.05)
curr_value = tf.placeholder(tf.float32)
prev_avg = tf.Variable(0.)
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

avg_hist = tf.summary.scalar("running_average", update_avg)
1
value_hist = tf.summary.scalar("incoming_values", curr_value)
2
merged = tf.summary.merge_all()                                   3
writer = tf.summary.FileWriter("./logs")                        4
init = tf.global_variables_initializer()
```

1 Creates a summary node for the averages
2 Creates a summary node for the values
3 Merges the summaries to make it easier to run all at once
4 Passes in the logs directory’s location to the writer

Visualizing data using TensorBoard

```
with tf.Session() as sess:
    sess.run(init)
    sess.add_graph(sess.graph)                                  5
    for i in range(len(raw_data)):
        summary_str, curr_avg = sess.run([merged, update_avg],
            feed_dict={curr_value: raw_data[i]})                6
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
        writer.add_summary(summary_str, i)                      7
```

5 Optional, but allows you to visualize the computation graph in TensorBoard
6 Runs the merged op and the update_avg op at the same time
7 Adds the summary to the writer