Deep Learning for Natural Language Processing

Stephen Clark et al.
University of Cambridge and DeepMind

7. Tensorflow

Stephen Clark
University of Cambridge and DeepMind

Borrowing from https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Deep Learning Packages

theano

∂y/net

torch

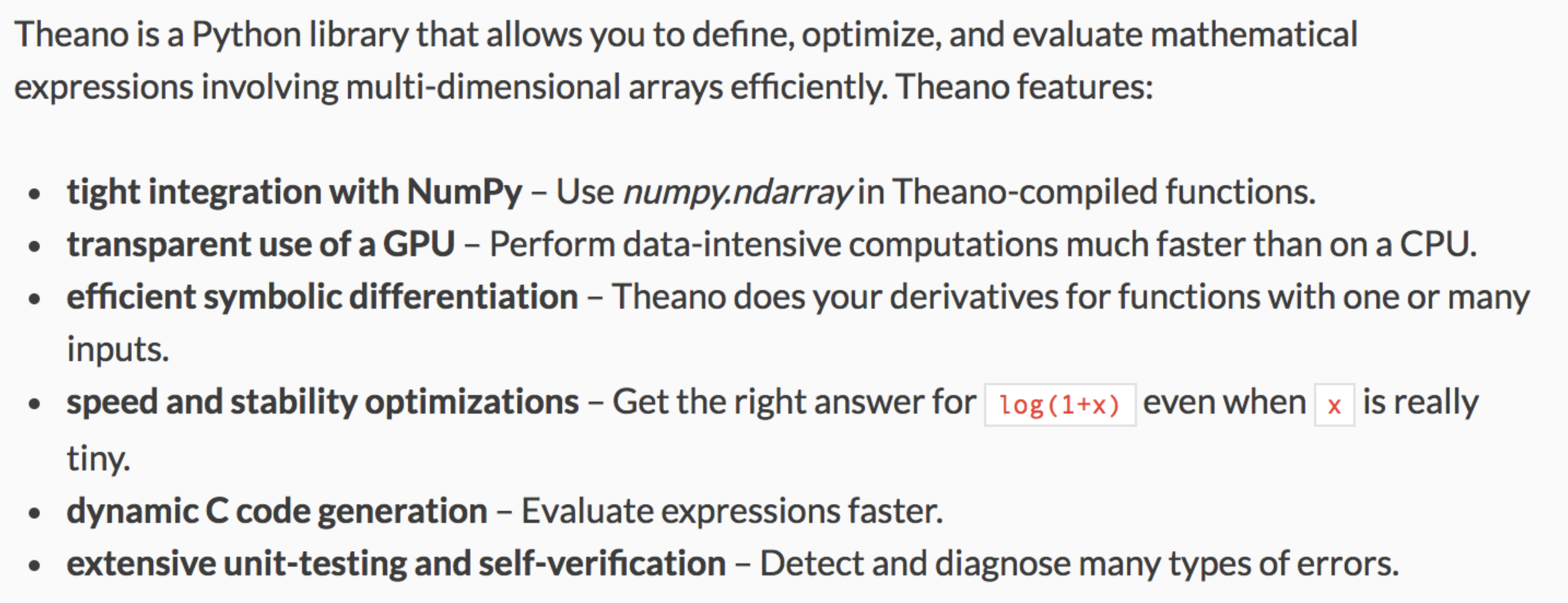
TensorFlow

cuDNN

PYTORCH

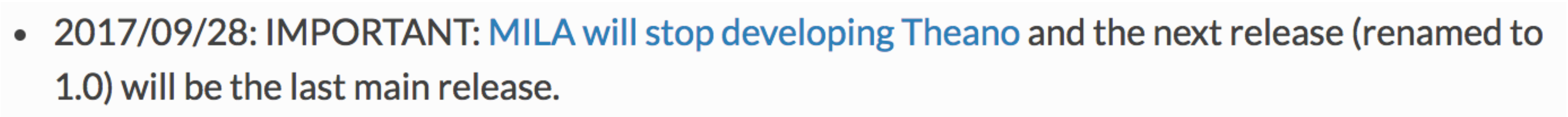
K Keras

Theano



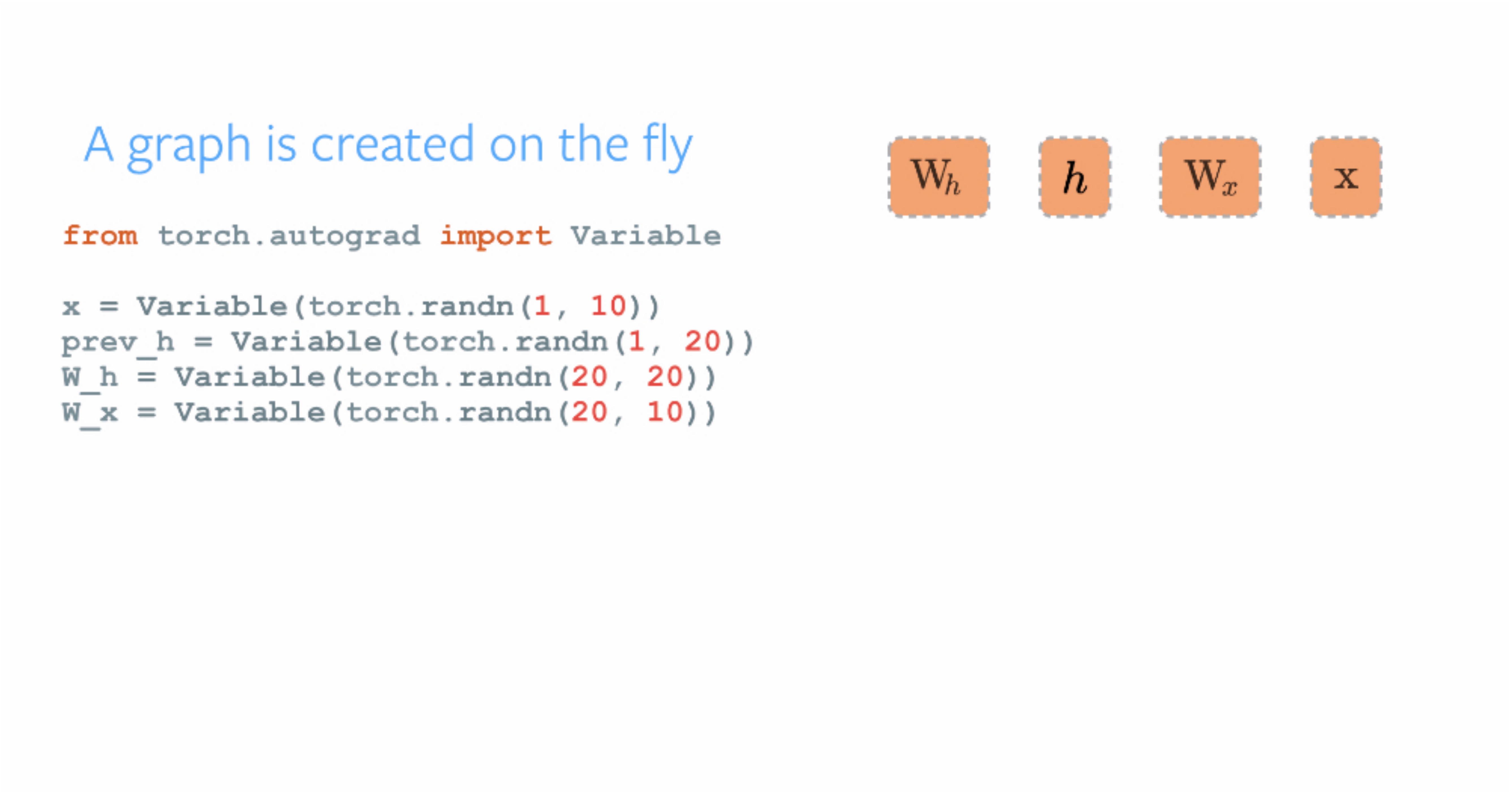
Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** – Use *numpy.ndarray* in Theano-compiled functions.
- **transparent use of a GPU** – Perform data-intensive computations much faster than on a CPU.
- **efficient symbolic differentiation** – Theano does your derivatives for functions with one or many inputs.
- **speed and stability optimizations** – Get the right answer for `log(1+x)` even when `x` is really tiny.
- **dynamic C code generation** – Evaluate expressions faster.
- **extensive unit-testing and self-verification** – Detect and diagnose many types of errors.



- 2017/09/28: IMPORTANT: MILA will stop developing Theano and the next release (renamed to 1.0) will be the last main release.

PyTorch

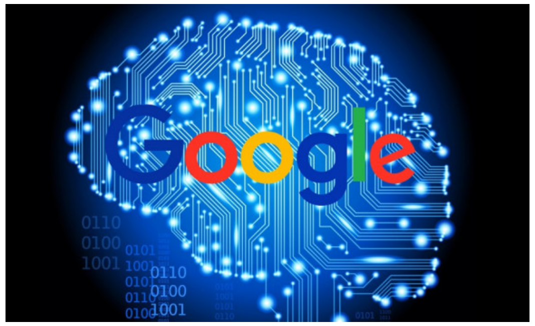


A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

http://pytorch.org/about/

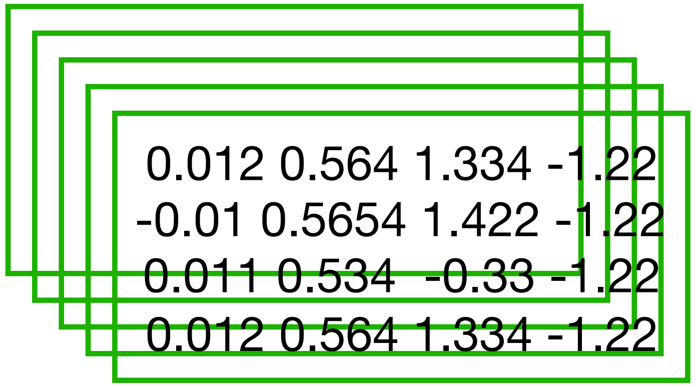
Tensorflow



"When Graham Bell invented the telephone, he saw a missed call from Jeff Dean."

TensorFlow, as the name indicates, is a framework to define and run computations involving tensors. A **tensor** is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes. (https://www.tensorflow.org/programmers_guide/tensors)

Tensors Mathematically



0.012 0.564 1.334 -1.22
-0.01 0.5654 1.422 -1.22
0.011 0.534 -0.33 -1.22
0.012 0.564 1.334 -1.22

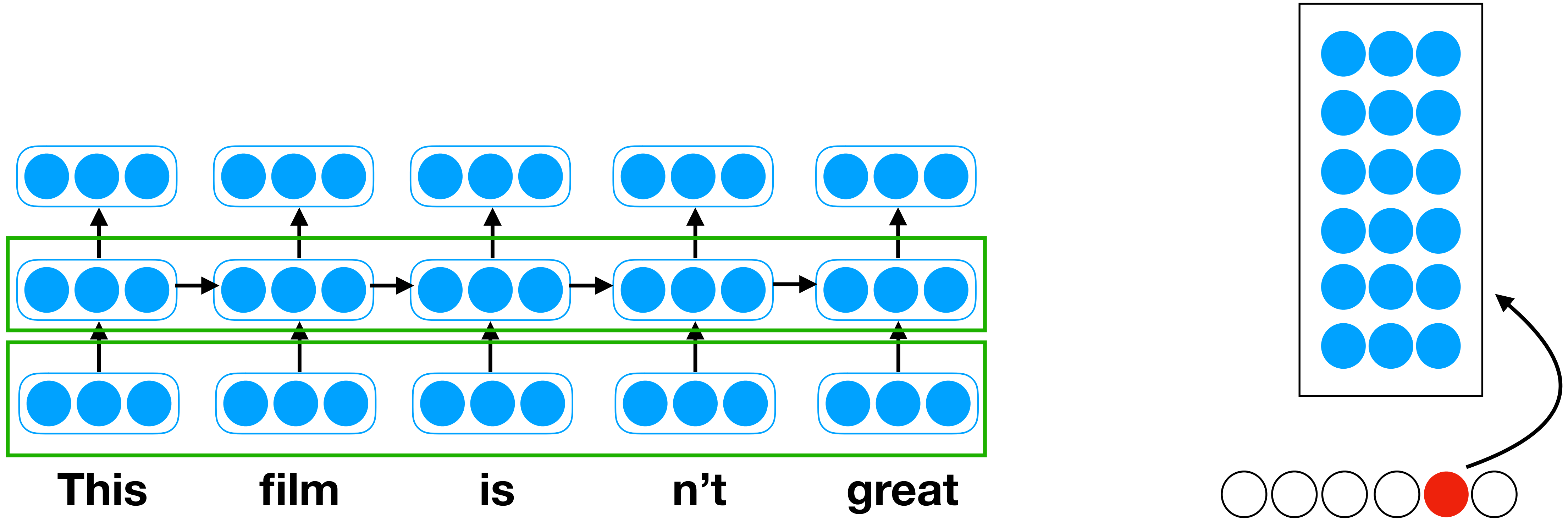
3rd order tensor

Tensors are higher-order generalizations of matrices (*multi*-linear algebra)

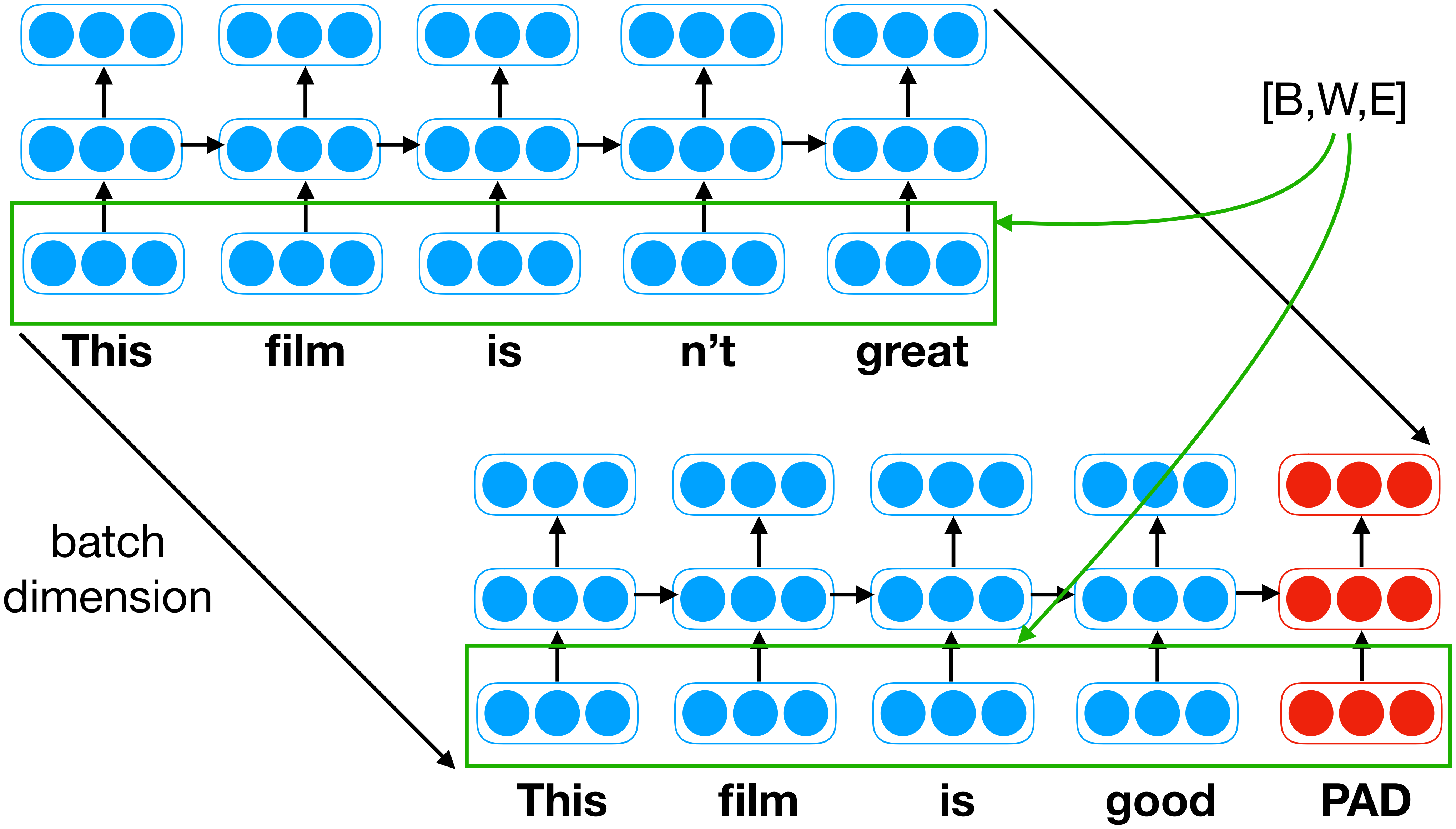
Matrices can be thought of as 2-D tables of numbers,
or linear maps;
likewise Tensors are *N*-D tables of numbers,
or multilinear maps:

$$f : V_1 \times \cdots \times V_n \rightarrow W$$

Matrixflow?



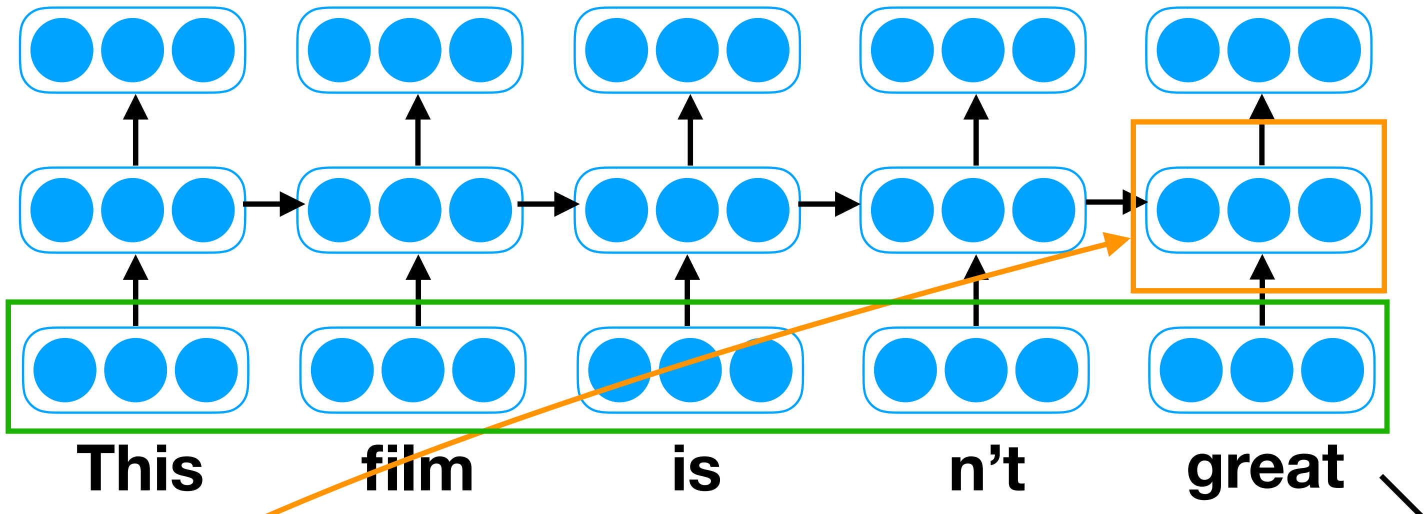
Tensors



DeepMind

UNIVERSITY OF CAMBRIDGE

Tensors



batch dimension

[B,W,E]

```
# embeddings for the batch of definitions (glosses).
embs = tf.nn.embedding_lookup(embedding_matrix, gloss_in)
# RNN encoder for the definitions.
if encoder_type == "recurrent":
  cell = tf.nn.rnn_cell.LSTMCell(emb_size)
  # state is the final state of the RNN.
  _, state = tf.nn.dynamic_rnn(cell, embs, dtype=tf.float32)
  # state is a pair: (hidden_state, output)
  core_out = state[0]
else:
  core_out = tf.reduce_mean(embs, axis=1)
```

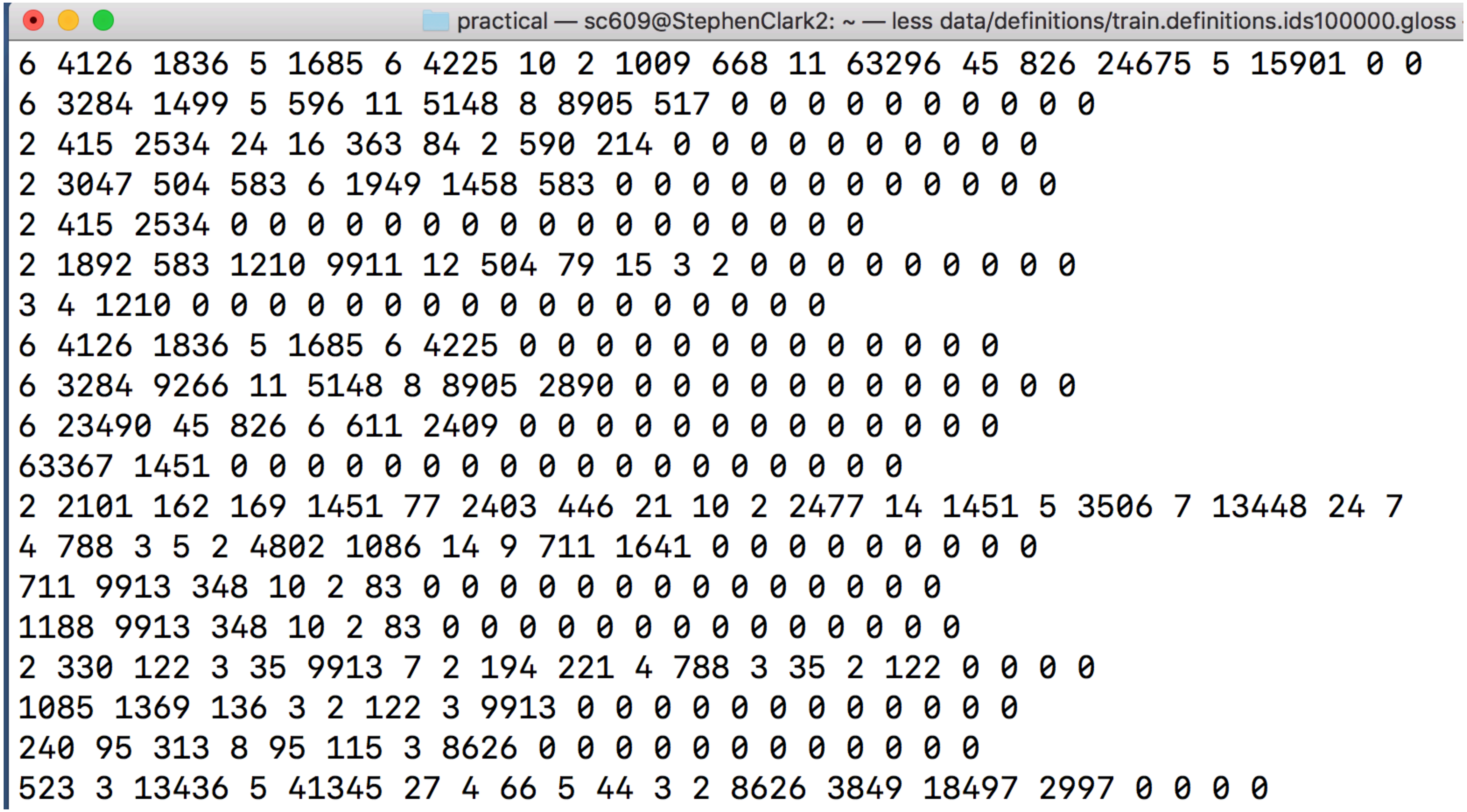
[B,E]

$$\begin{pmatrix} 5 & 76 & 9998 & 5 & 17 \\ 4 & 65 & 543 & 0 & 0 \\ & & \cdots & & \\ 5 & 435 & 4 & 765 & 0 \end{pmatrix}$$

DeepMind

UNIVERSITY OF CAMBRIDGE

Padded Input



practical — sc609@StephenClark2: ~ — less data/definitions/train.definitions.ids100000.gloss

```
6 4126 1836 5 1685 6 4225 10 2 1009 668 11 63296 45 826 24675 5 15901 0 0
6 3284 1499 5 596 11 5148 8 8905 517 0 0 0 0 0 0 0 0 0 0
2 415 2534 24 16 363 84 2 590 214 0 0 0 0 0 0 0 0 0 0
2 3047 504 583 6 1949 1458 583 0 0 0 0 0 0 0 0 0 0 0 0
2 415 2534 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1892 583 1210 9911 12 504 79 15 3 2 0 0 0 0 0 0 0 0 0
3 4 1210 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 4126 1836 5 1685 6 4225 0 0 0 0 0 0 0 0 0 0 0 0 0
6 3284 9266 11 5148 8 8905 2890 0 0 0 0 0 0 0 0 0 0 0 0
6 23490 45 826 6 611 2409 0 0 0 0 0 0 0 0 0 0 0 0 0
63367 1451 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2101 162 169 1451 77 2403 446 21 10 2 2477 14 1451 5 3506 7 13448 24 7
4 788 3 5 2 4802 1086 14 9 711 1641 0 0 0 0 0 0 0 0 0
711 9913 348 10 2 83 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1188 9913 348 10 2 83 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 330 122 3 35 9913 7 2 194 221 4 788 3 35 2 122 0 0 0 0
1085 1369 136 3 2 122 3 9913 0 0 0 0 0 0 0 0 0 0 0 0
240 95 313 8 95 115 3 8626 0 0 0 0 0 0 0 0 0 0 0 0
523 3 13436 5 41345 27 4 66 5 44 3 2 8626 3849 18497 2997 0 0 0 0
```

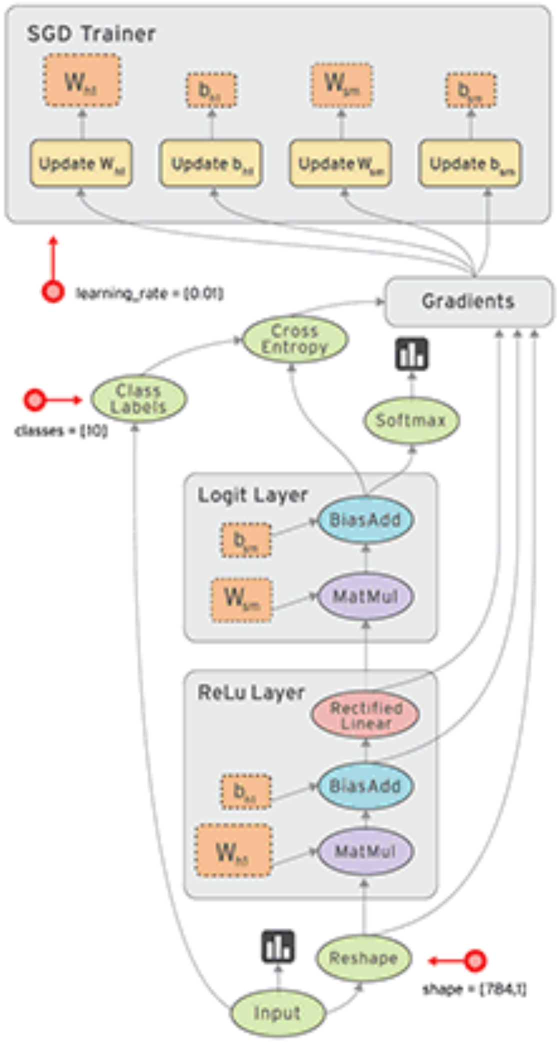
Numpy vs. Tensorflow

- Tensorflow shares many features with numpy
- Values returned from a TF fetch are numpy values
- Common programming paradigm is:
  - compute values using the TF graph (e.g. training a model);
  - fetch particular values from the graph (e.g. value of the loss function);
  - and then perform further computation in numpy/python (e.g. for evaluation)

Programming in Tensorflow

Tensorflow uses a declarative programming paradigm,
and builds a computation graph statically,
with python and C++ APIs

Dataflow Graphs

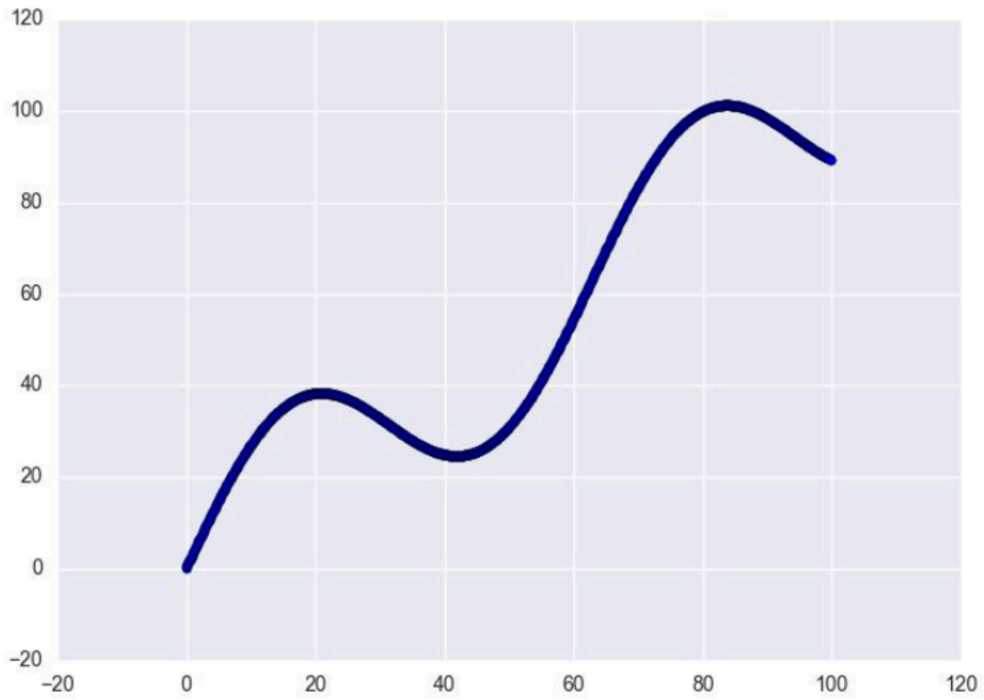


TensorFlow uses a **dataflow graph** to represent your computation in terms of the dependencies between individual operations. This leads to a low-level programming model in which you first define the dataflow graph, then create a TensorFlow **session** to run parts of the graph (https://www.tensorflow.org/programmers_guide/graphs)

DeepMind

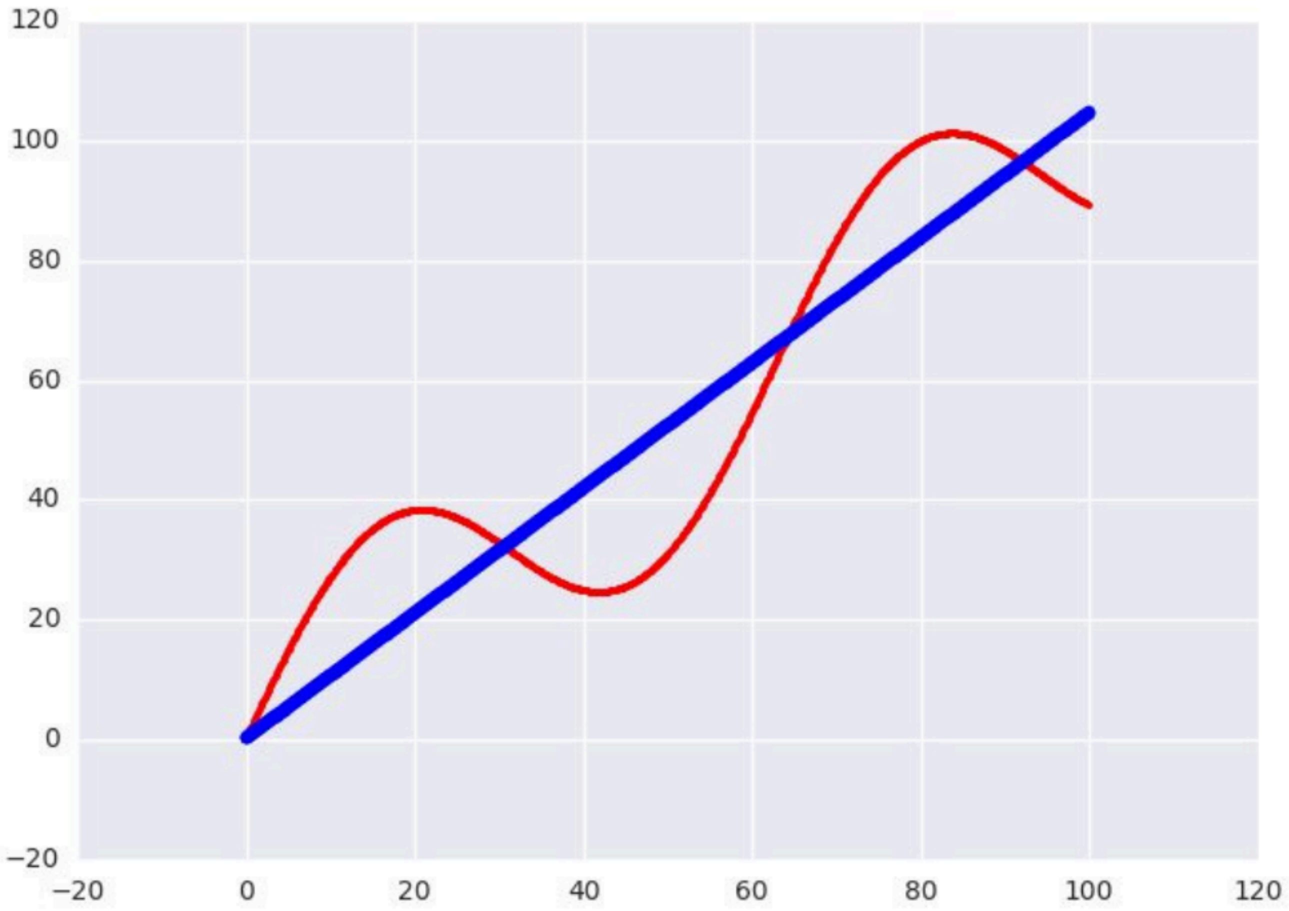
UNIVERSITY OF CAMBRIDGE

Linear Regression Example



Taken from https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Linear Regression Example

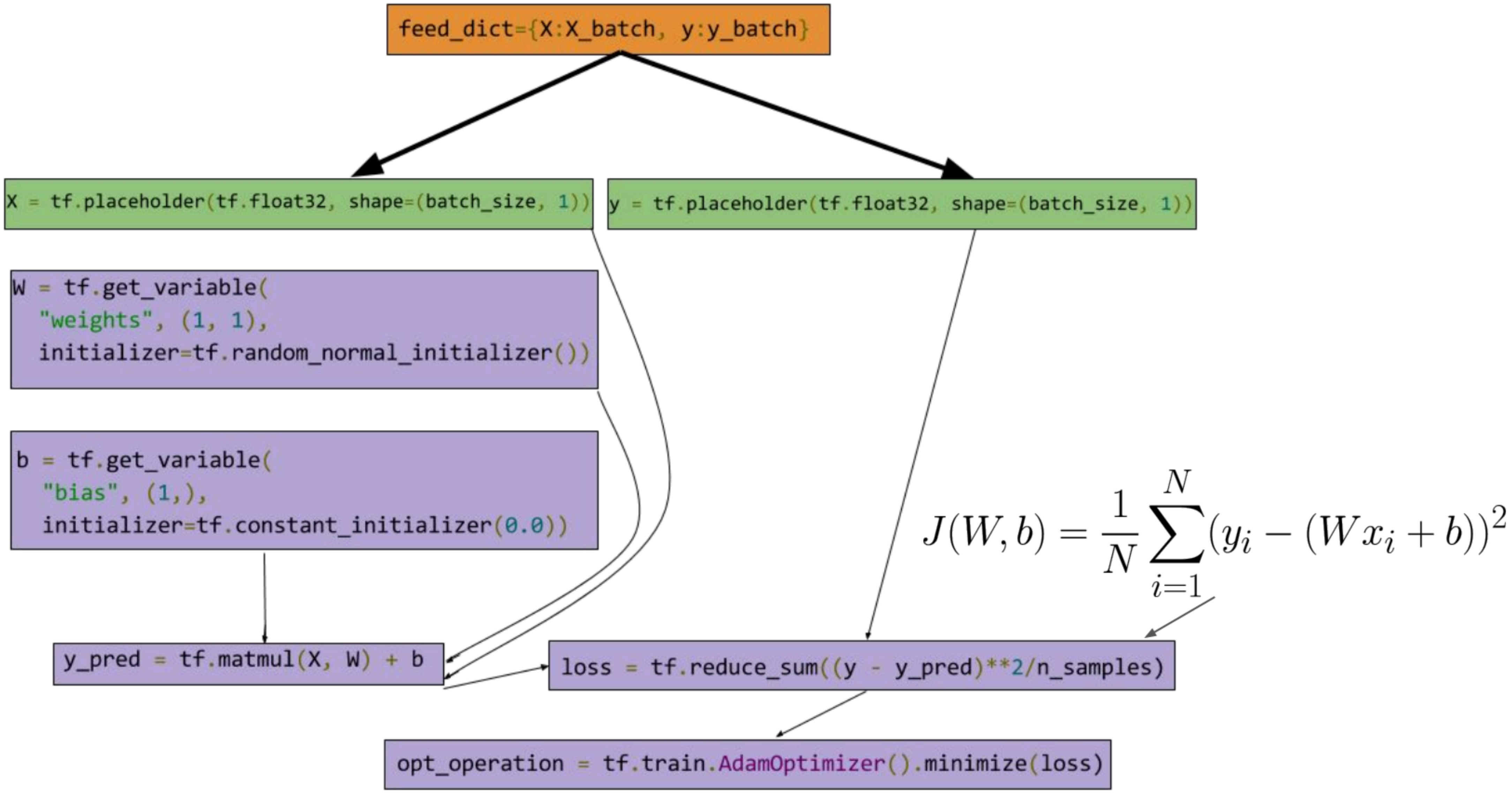


Taken from https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

DeepMind

UNIVERSITY OF CAMBRIDGE

Linear Regression Example



```
feed_dict={X:X_batch, y:y_batch}
```

```
X = tf.placeholder(tf.float32, shape=(batch_size, 1))
```

```
y = tf.placeholder(tf.float32, shape=(batch_size, 1))
```

```
W = tf.get_variable(
  "weights", (1, 1),
  initializer=tf.random_normal_initializer())
```

```
b = tf.get_variable(
  "bias", (1,),
  initializer=tf.constant_initializer(0.0))
```

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b))^2$$

```
y_pred = tf.matmul(X, W) + b
```

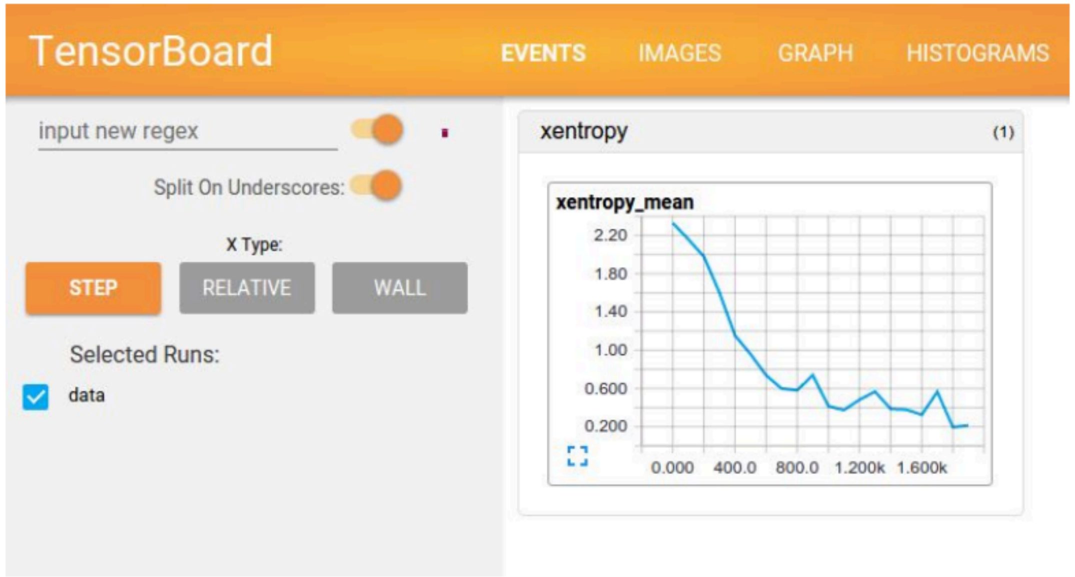
```
loss = tf.reduce_sum((y - y_pred)**2/n_samples)
```

```
opt_operation = tf.train.AdamOptimizer().minimize(loss)
```

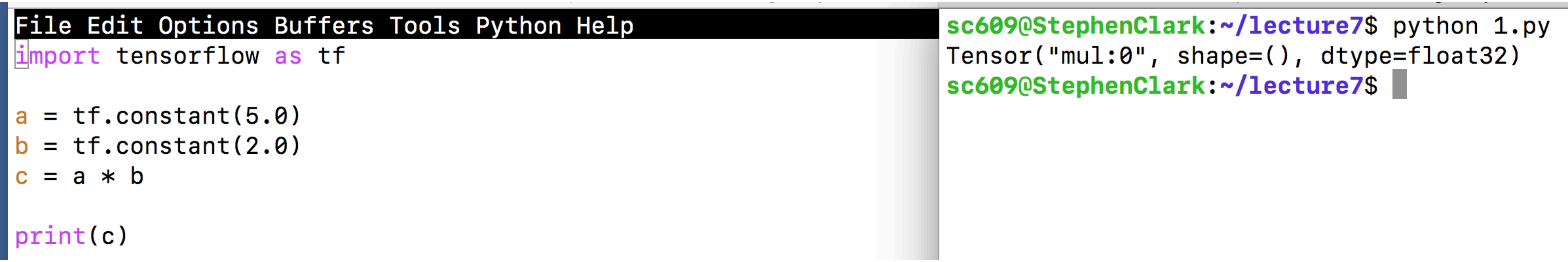
Taken from https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensorboard

- Great for graph visualization



A Simple Tensorflow Graph



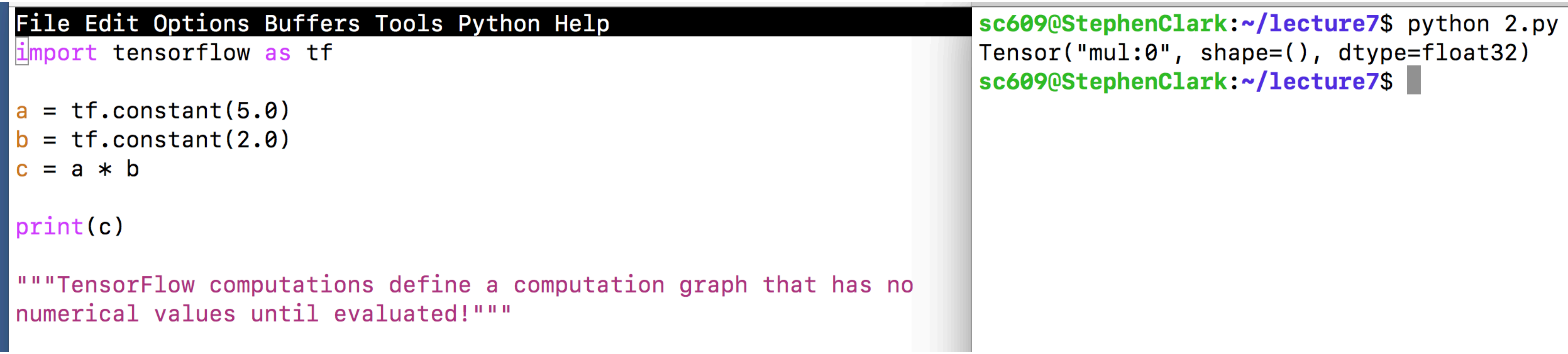
```python
import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

print(c)
```

```
sc609@StephenClark:~/lecture7$ python 1.py
Tensor("mul:0", shape=(), dtype=float32)
sc609@StephenClark:~/lecture7$
```

DeepMind

UNIVERSITY OF
CAMBRIDGE

A Simple Tensorflow Graph



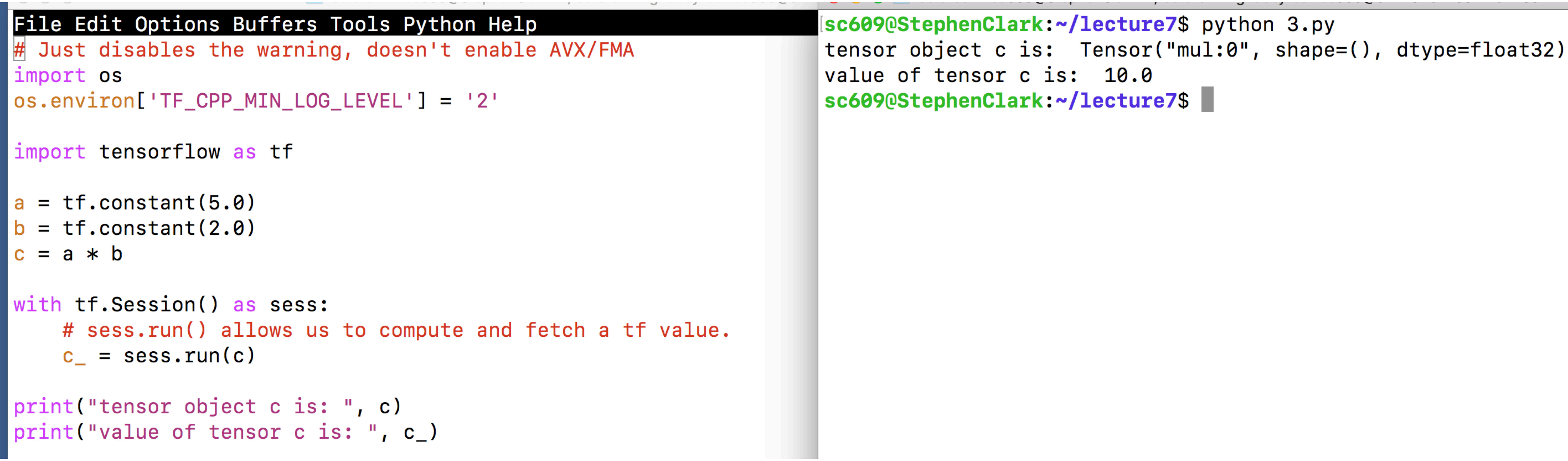
```python
import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

print(c)

"""TensorFlow computations define a computation graph that has no
numerical values until evaluated!"""
```

```
sc609@StephenClark:~/lecture7$ python 2.py
Tensor("mul:0", shape=(), dtype=float32)
sc609@StephenClark:~/lecture7$
```

DeepMind

UNIVERSITY OF CAMBRIDGE

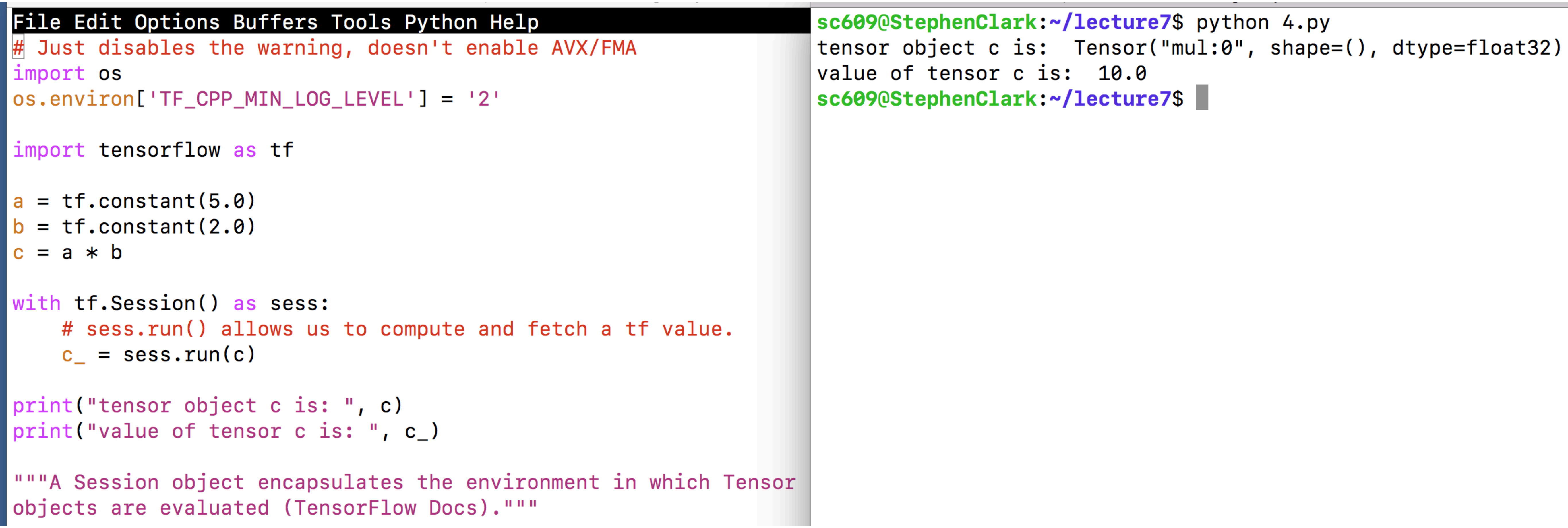
Tensorflow Sessions



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    c_ = sess.run(c)

print("tensor object c is: ", c)
print("value of tensor c is: ", c_)
```

```
sc609@StephenClark:~/lecture7$ python 3.py
tensor object c is:  Tensor("mul:0", shape=(), dtype=float32)
value of tensor c is:  10.0
sc609@StephenClark:~/lecture7$
```

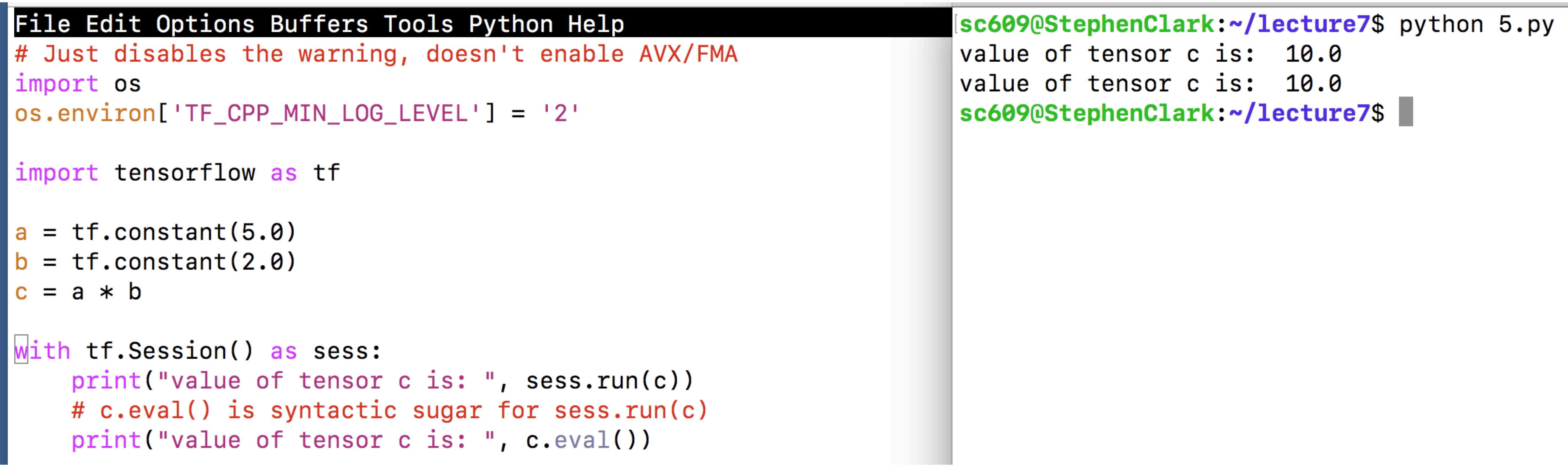
Tensorflow Sessions



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    c_ = sess.run(c)

print("tensor object c is: ", c)
print("value of tensor c is: ", c_)

"""A Session object encapsulates the environment in which Tensor
objects are evaluated (TensorFlow Docs)."""
```

```
sc609@StephenClark:~/lecture7$ python 4.py
tensor object c is:  Tensor("mul:0", shape=(), dtype=float32)
value of tensor c is:  10.0
sc609@StephenClark:~/lecture7$
```

Evaluation in Sessions



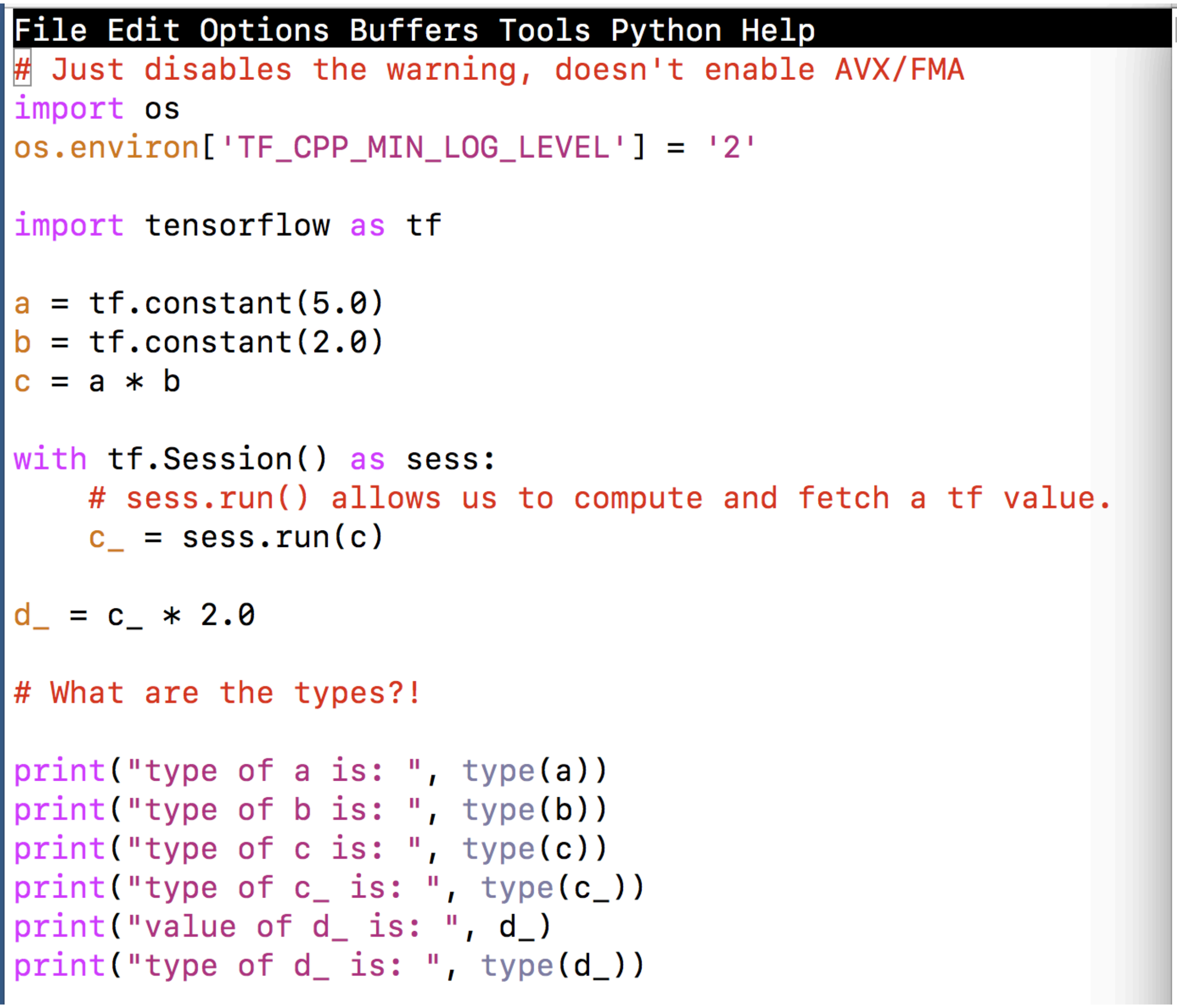
```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

with tf.Session() as sess:
    print("value of tensor c is: ", sess.run(c))
    # c.eval() is syntactic sugar for sess.run(c)
    print("value of tensor c is: ", c.eval())
```

```
sc609@StephenClark:~/lecture7$ python 5.py
value of tensor c is:  10.0
value of tensor c is:  10.0
sc609@StephenClark:~/lecture7$
```

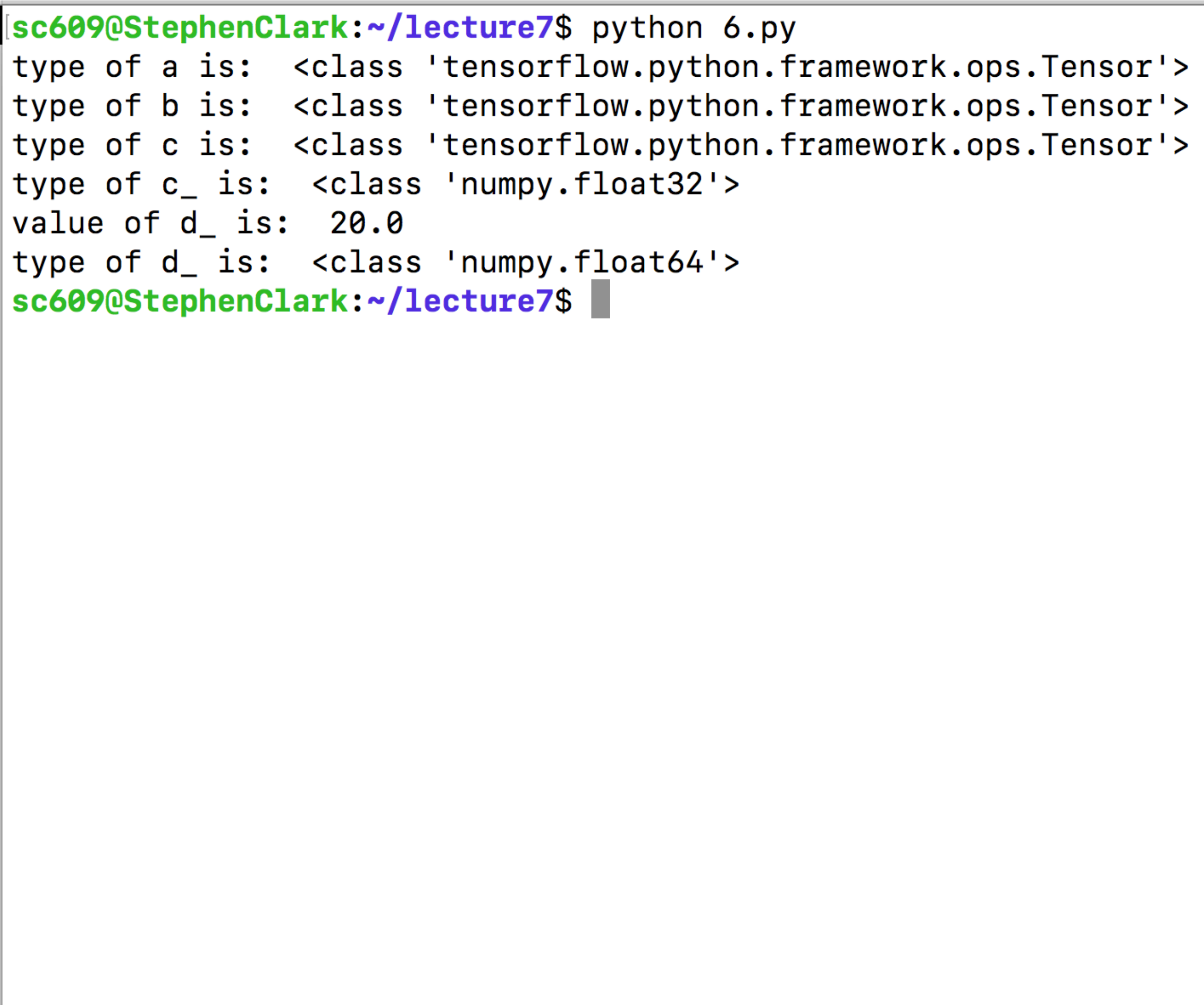
DeepMind

UNIVERSITY OF CAMBRIDGE

Types in a TF Program

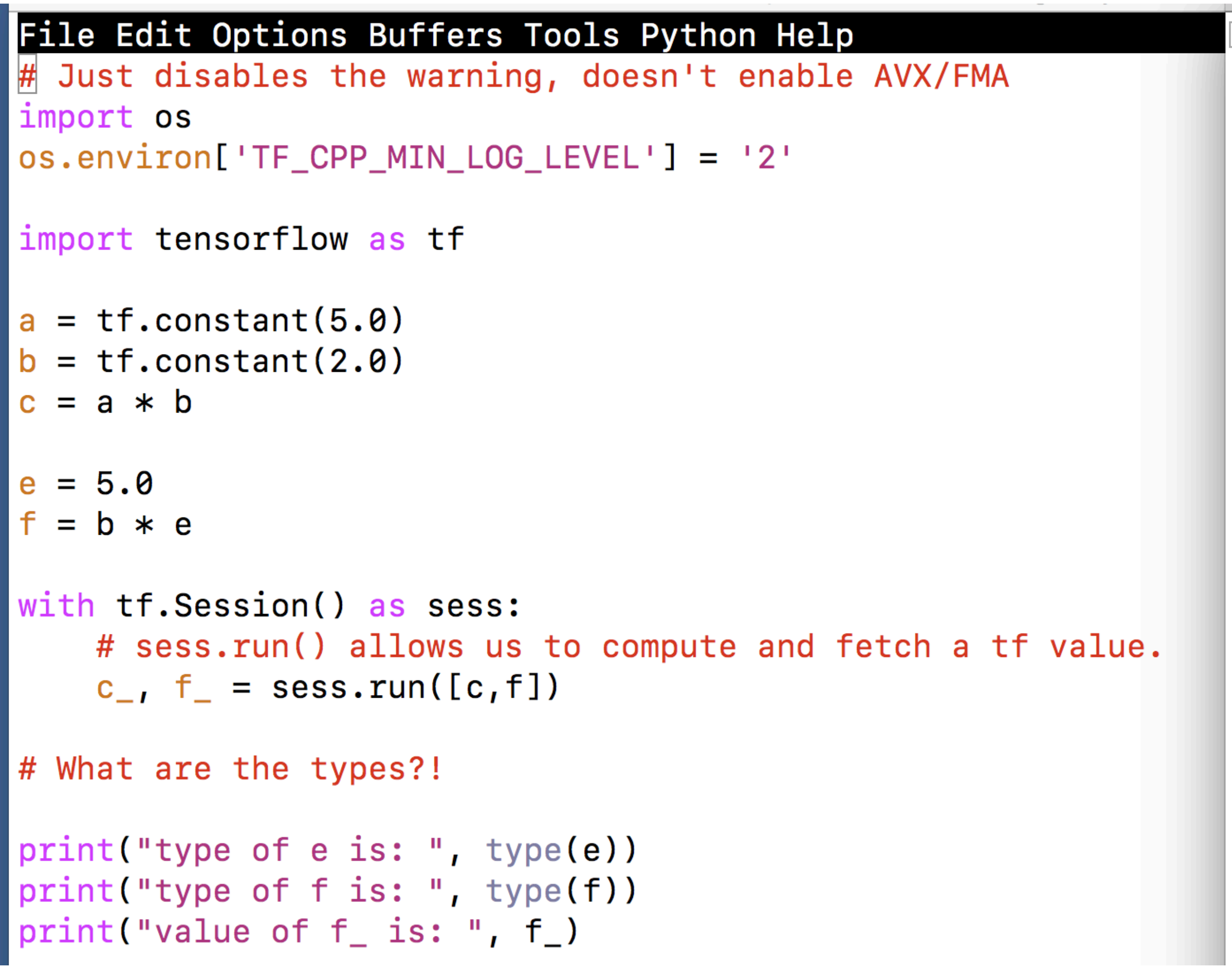


```
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    c_ = sess.run(c)

d_ = c_ * 2.0

# What are the types?!

print("type of a is: ", type(a))
print("type of b is: ", type(b))
print("type of c is: ", type(c))
print("type of c_ is: ", type(c_))
print("value of d_ is: ", d_)
print("type of d_ is: ", type(d_))
```

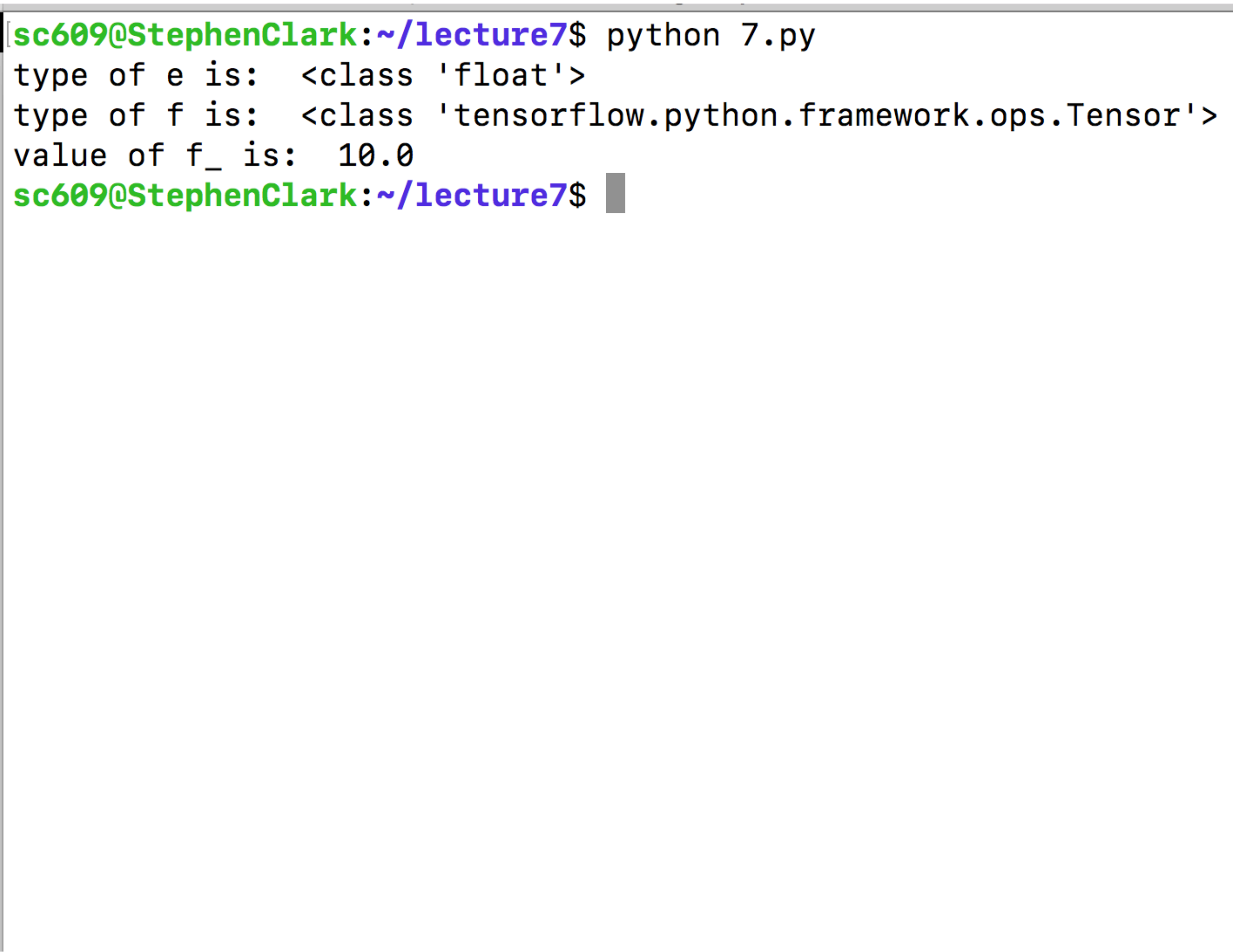


```
sc609@StephenClark:~/lecture7$ python 6.py
type of a is:  <class 'tensorflow.python.framework.ops.Tensor'>
type of b is:  <class 'tensorflow.python.framework.ops.Tensor'>
type of c is:  <class 'tensorflow.python.framework.ops.Tensor'>
type of c_ is:  <class 'numpy.float32'>
value of d_ is:  20.0
type of d_ is:  <class 'numpy.float64'>
sc609@StephenClark:~/lecture7$ 
```

Types in a TF Program

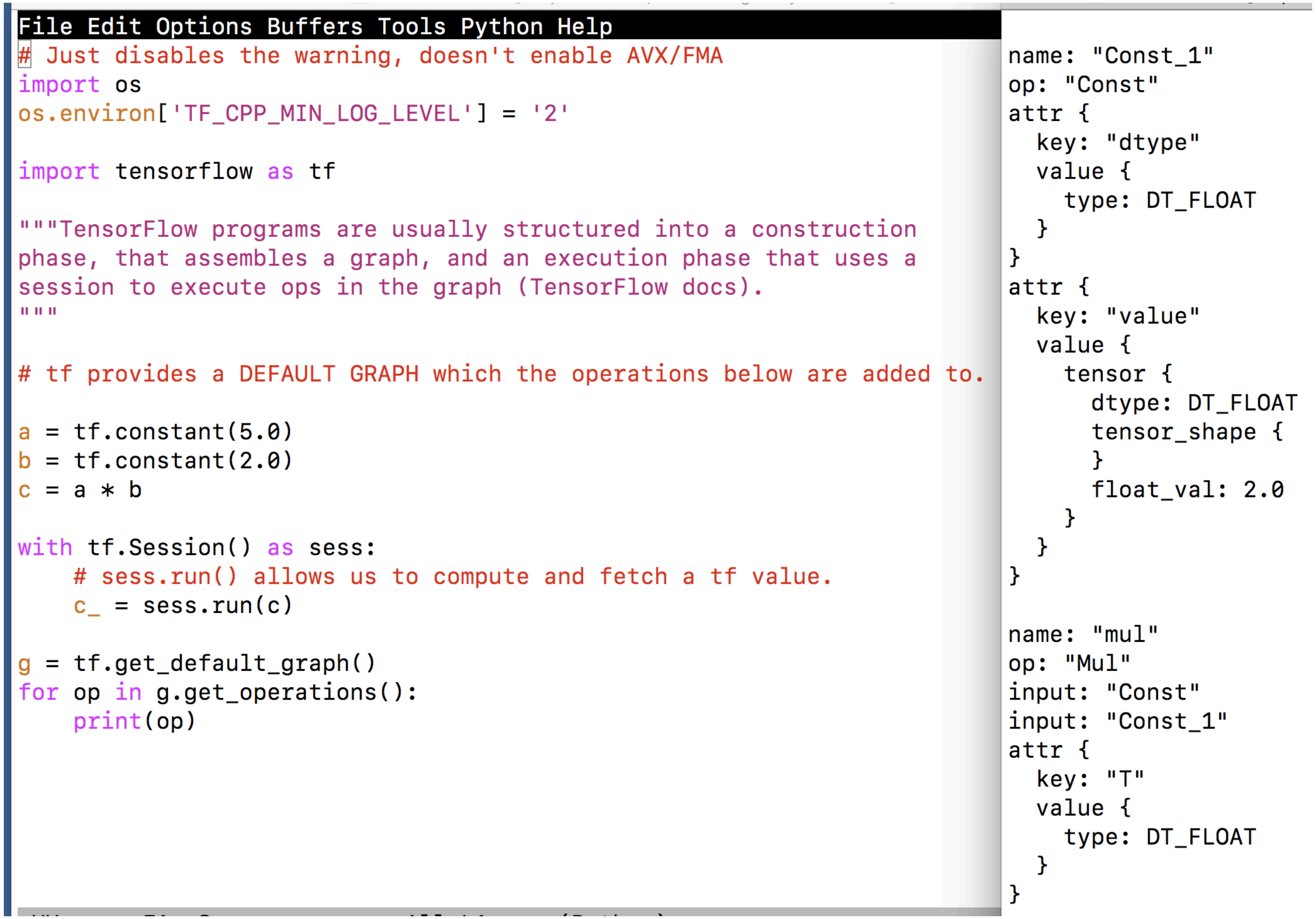


```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

e = 5.0
f = b * e

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    c_, f_ = sess.run([c,f])

# What are the types?!

print("type of e is: ", type(e))
print("type of f is: ", type(f))
print("value of f_ is: ", f_)
```



```
sc609@StephenClark:~/lecture7$ python 7.py
type of e is:  <class 'float'>
type of f is:  <class 'tensorflow.python.framework.ops.Tensor'>
value of f_ is:  10.0
sc609@StephenClark:~/lecture7$
```

The Default Graph



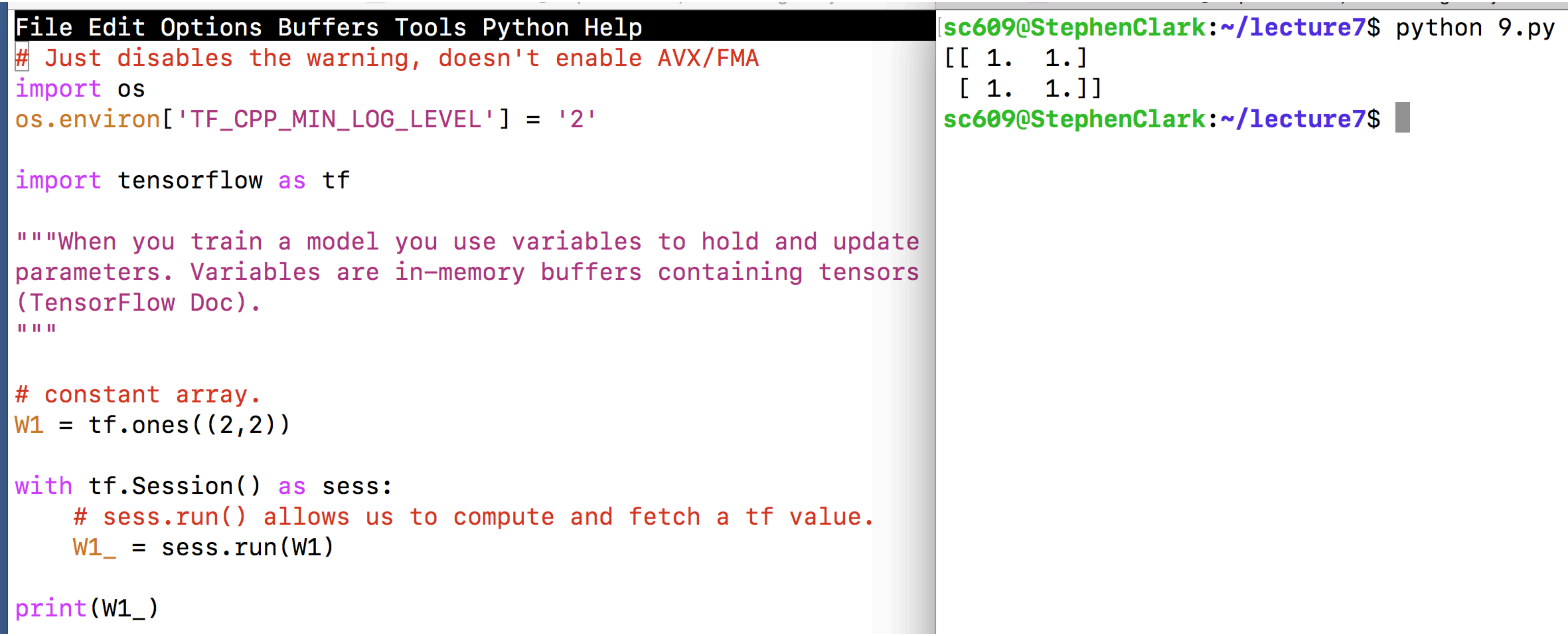
```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""TensorFlow programs are usually structured into a construction
phase, that assembles a graph, and an execution phase that uses a
session to execute ops in the graph (TensorFlow docs).
"""

# tf provides a DEFAULT GRAPH which the operations below are added to.

a = tf.constant(5.0)
b = tf.constant(2.0)
c = a * b

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    c_ = sess.run(c)

g = tf.get_default_graph()
for op in g.get_operations():
    print(op)
```

```
name: "Const_1"
op: "Const"
attr {
  key: "dtype"
  value {
    type: DT_FLOAT
  }
}
attr {
  key: "value"
  value {
    tensor {
      dtype: DT_FLOAT
      tensor_shape {
      }
      float_val: 2.0
    }
  }
}

name: "mul"
op: "Mul"
input: "Const"
input: "Const_1"
attr {
  key: "T"
  value {
    type: DT_FLOAT
  }
}
```

DeepMind

UNIVERSITY OF CAMBRIDGE

Variables in Tensorflow



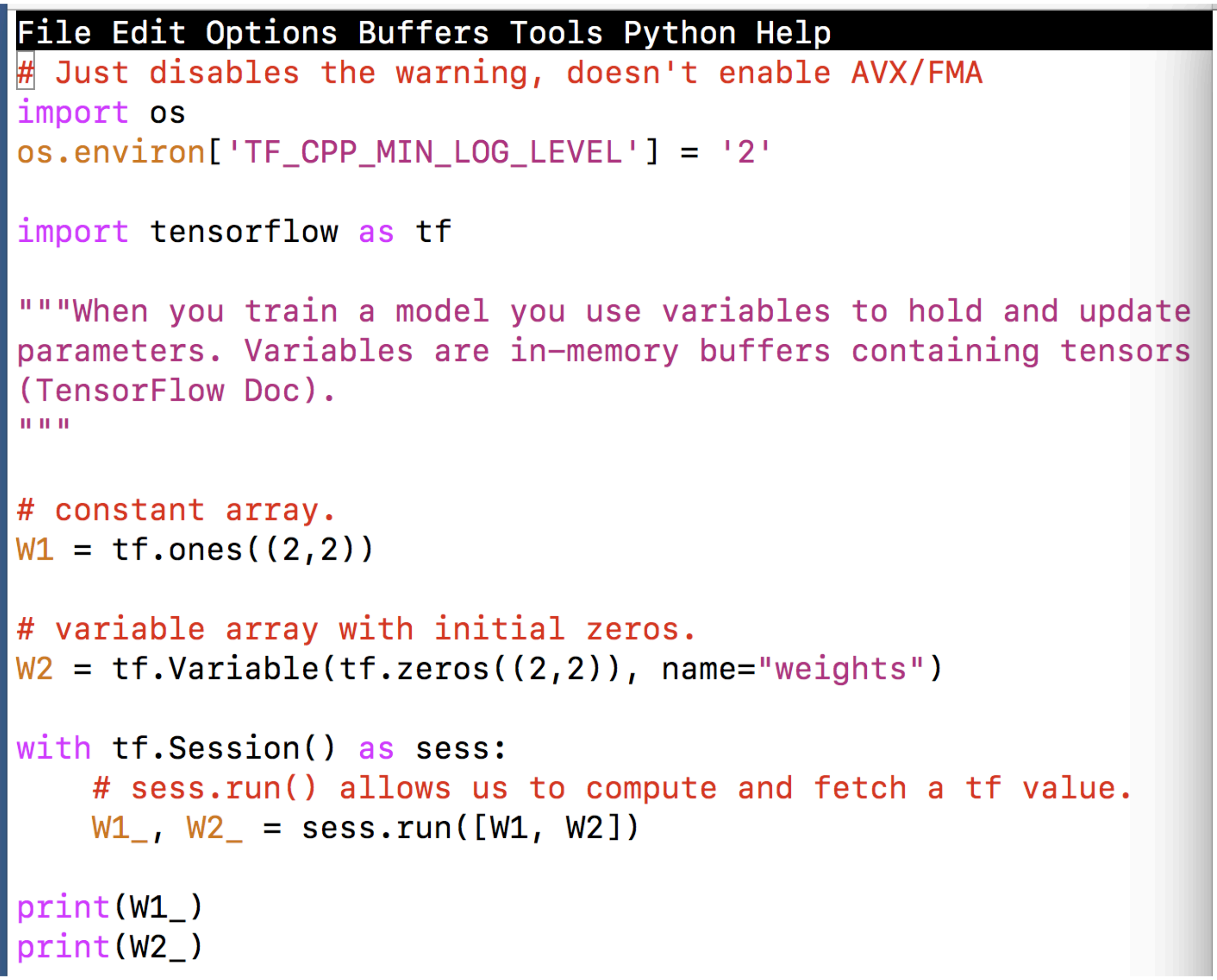
```
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""When you train a model you use variables to hold and update
parameters. Variables are in-memory buffers containing tensors
(TensorFlow Doc).
"""


# constant array.
W1 = tf.ones((2,2))

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    W1_ = sess.run(W1)

print(W1_)
```

```
sc609@StephenClark:~/lecture7$ python 9.py
[[ 1.  1.]
 [ 1.  1.]]
sc609@StephenClark:~/lecture7$
```

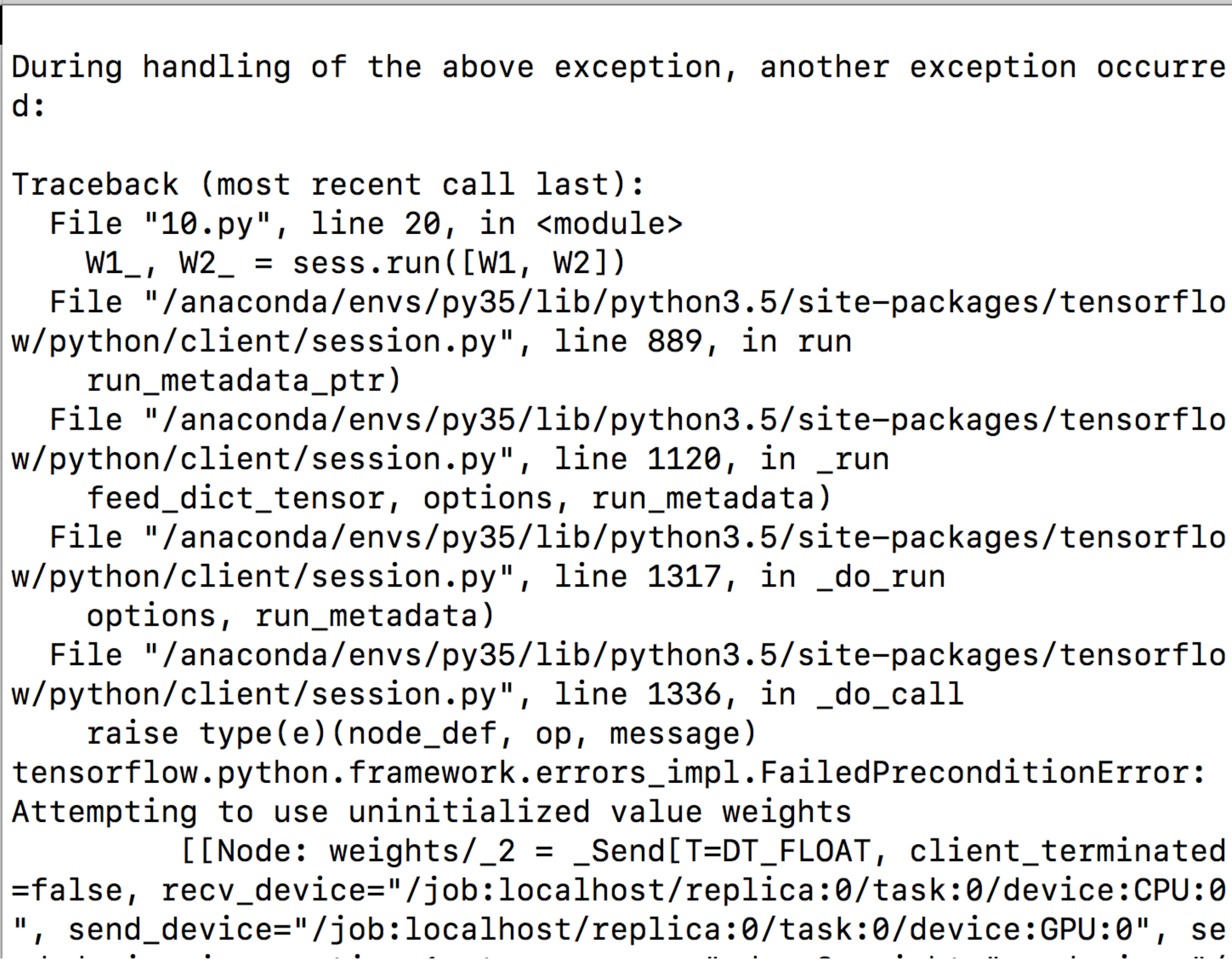
DeepMind

UNIVERSITY OF CAMBRIDGE

Initializing Variables



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""When you train a model you use variables to hold and update
parameters. Variables are in-memory buffers containing tensors
(TensorFlow Doc).
"""

# constant array.
W1 = tf.ones((2,2))

# variable array with initial zeros.
W2 = tf.Variable(tf.zeros((2,2)), name="weights")

with tf.Session() as sess:
    # sess.run() allows us to compute and fetch a tf value.
    W1_, W2_ = sess.run([W1, W2])

print(W1_)
print(W2_)
```

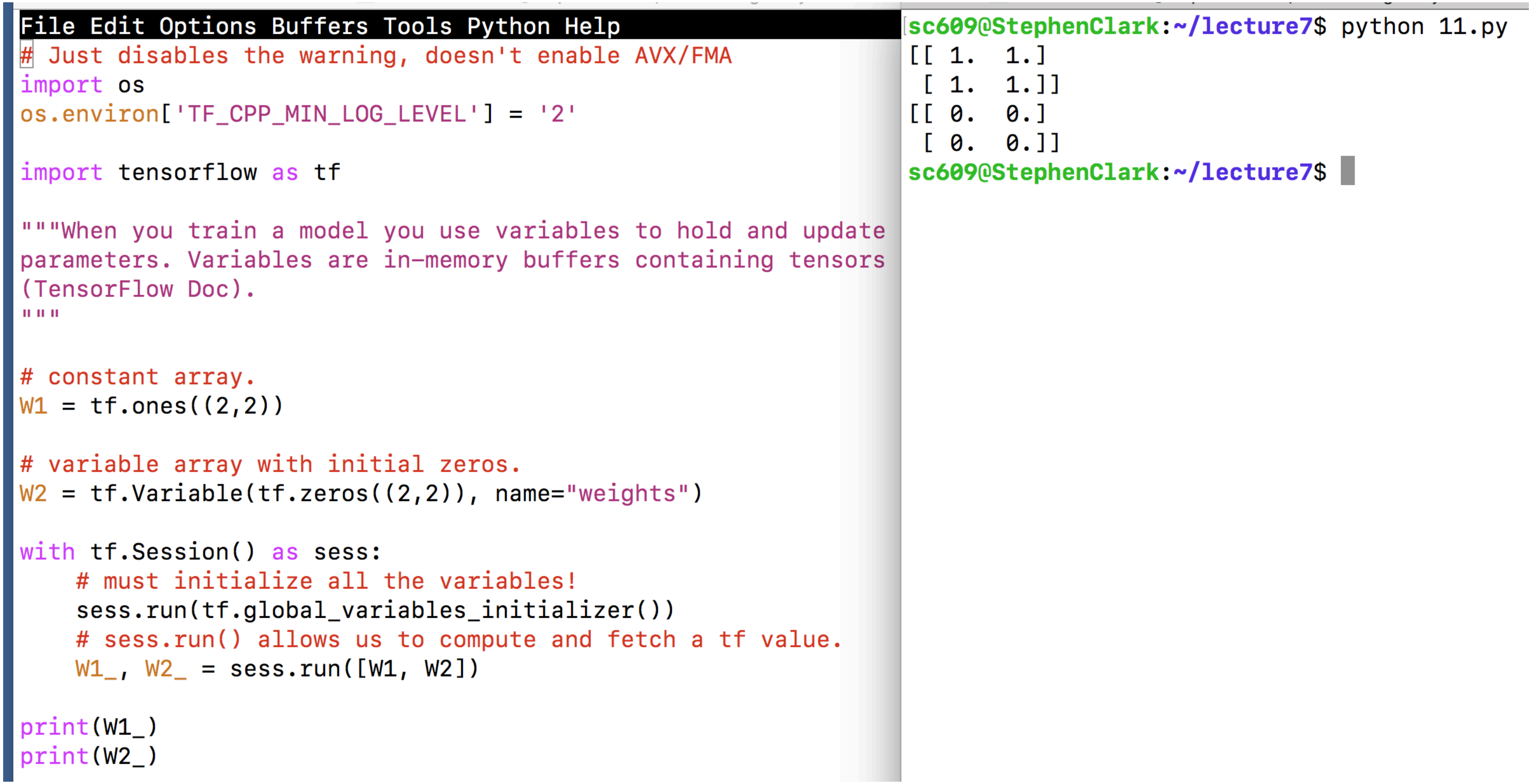


```
During handling of the above exception, another exception occurre
d:

Traceback (most recent call last):
  File "10.py", line 20, in <module>
    W1_, W2_ = sess.run([W1, W2])
  File "/anaconda/envs/py35/lib/python3.5/site-packages/tensorflo
w/python/client/session.py", line 889, in run
    run_metadata_ptr)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/tensorflo
w/python/client/session.py", line 1120, in _run
    feed_dict_tensor, options, run_metadata)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/tensorflo
w/python/client/session.py", line 1317, in _do_run
    options, run_metadata)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/tensorflo
w/python/client/session.py", line 1336, in _do_call
    raise type(e)(node_def, op, message)
tensorflow.python.framework.errors_impl.FailedPreconditionError:
Attempting to use uninitialized value weights
         [[Node: weights/_2 = _Send[T=DT_FLOAT, client_terminated
=false, recv_device="/job:localhost/replica:0/task:0/device:CPU:0
", send_device="/job:localhost/replica:0/task:0/device:GPU:0", se
```

DeepMind

UNIVERSITY OF
CAMBRIDGE

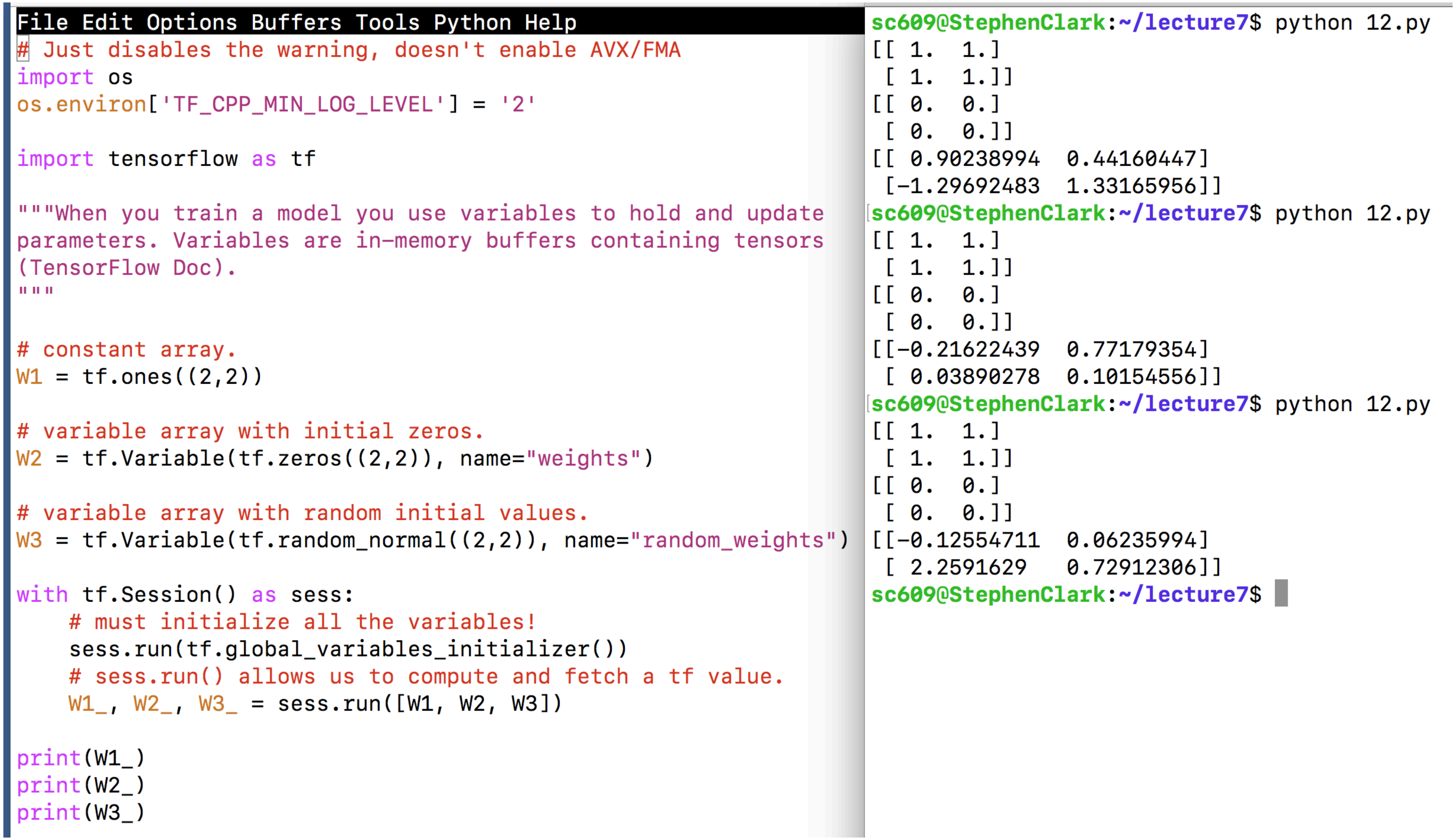
Initializing Variables



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""When you train a model you use variables to hold and update
parameters. Variables are in-memory buffers containing tensors
(TensorFlow Doc).
"""

# constant array.
W1 = tf.ones((2,2))

# variable array with initial zeros.
W2 = tf.Variable(tf.zeros((2,2)), name="weights")

with tf.Session() as sess:
    # must initialize all the variables!
    sess.run(tf.global_variables_initializer())
    # sess.run() allows us to compute and fetch a tf value.
    W1_, W2_ = sess.run([W1, W2])

print(W1_)
print(W2_)
```

```
sc609@StephenClark:~/lecture7$ python 11.py
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
sc609@StephenClark:~/lecture7$ 
```

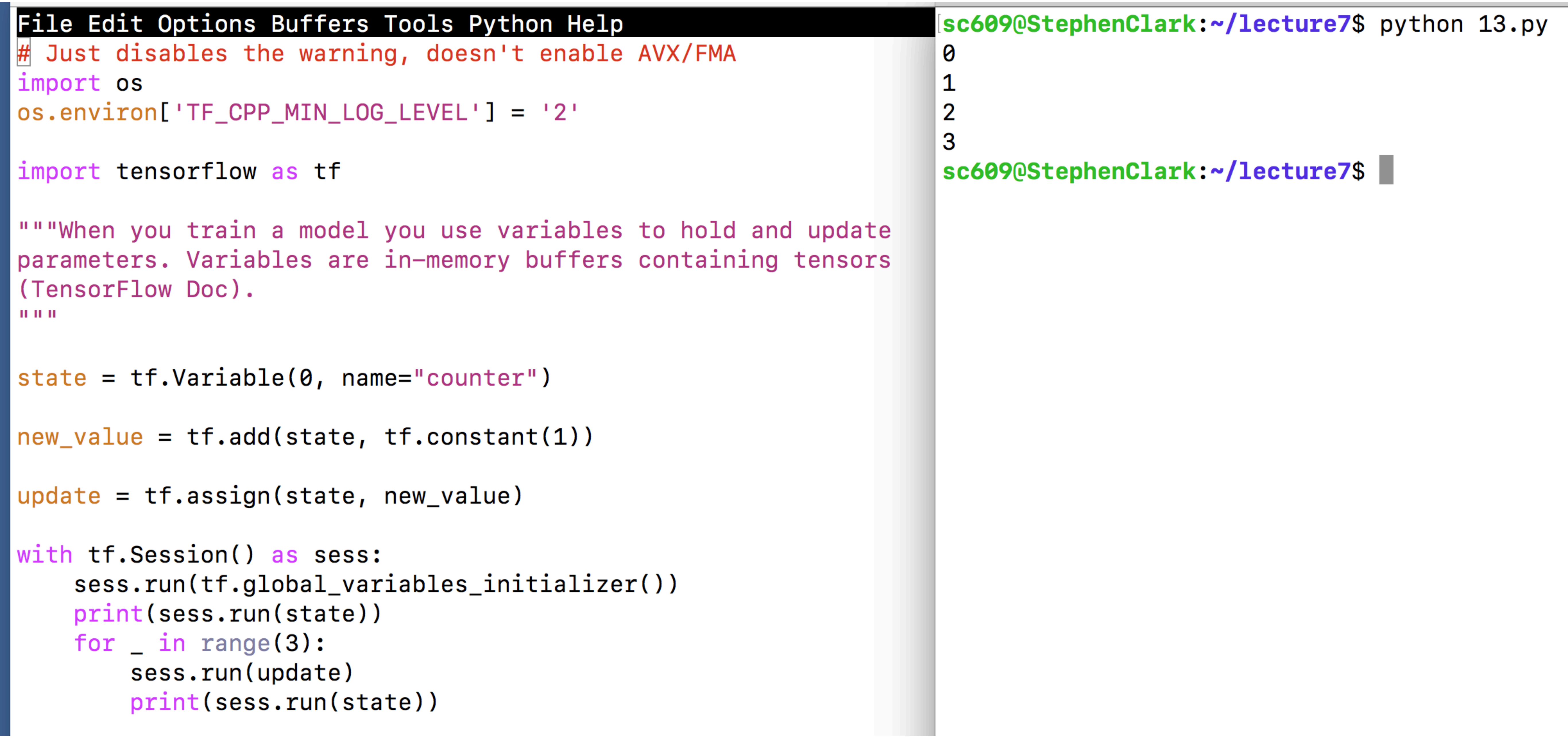
Initializing Variables



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""When you train a model you use variables to hold and update
parameters. Variables are in-memory buffers containing tensors
(TensorFlow Doc).
"""

# constant array.
W1 = tf.ones((2,2))

# variable array with initial zeros.
W2 = tf.Variable(tf.zeros((2,2)), name="weights")

# variable array with random initial values.
W3 = tf.Variable(tf.random_normal((2,2)), name="random_weights")

with tf.Session() as sess:
    # must initialize all the variables!
    sess.run(tf.global_variables_initializer())
    # sess.run() allows us to compute and fetch a tf value.
    W1_, W2_, W3_ = sess.run([W1, W2, W3])

print(W1_)
print(W2_)
print(W3_)
```

```
sc609@StephenClark:~/lecture7$ python 12.py
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
[[ 0.90238994  0.44160447]
 [-1.29692483  1.33165956]]
sc609@StephenClark:~/lecture7$ python 12.py
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
[[-0.21622439  0.77179354]
 [ 0.03890278  0.10154556]]
sc609@StephenClark:~/lecture7$ python 12.py
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
[[-0.12554711  0.06235994]
 [ 2.2591629   0.72912306]]
sc609@StephenClark:~/lecture7$ 
```

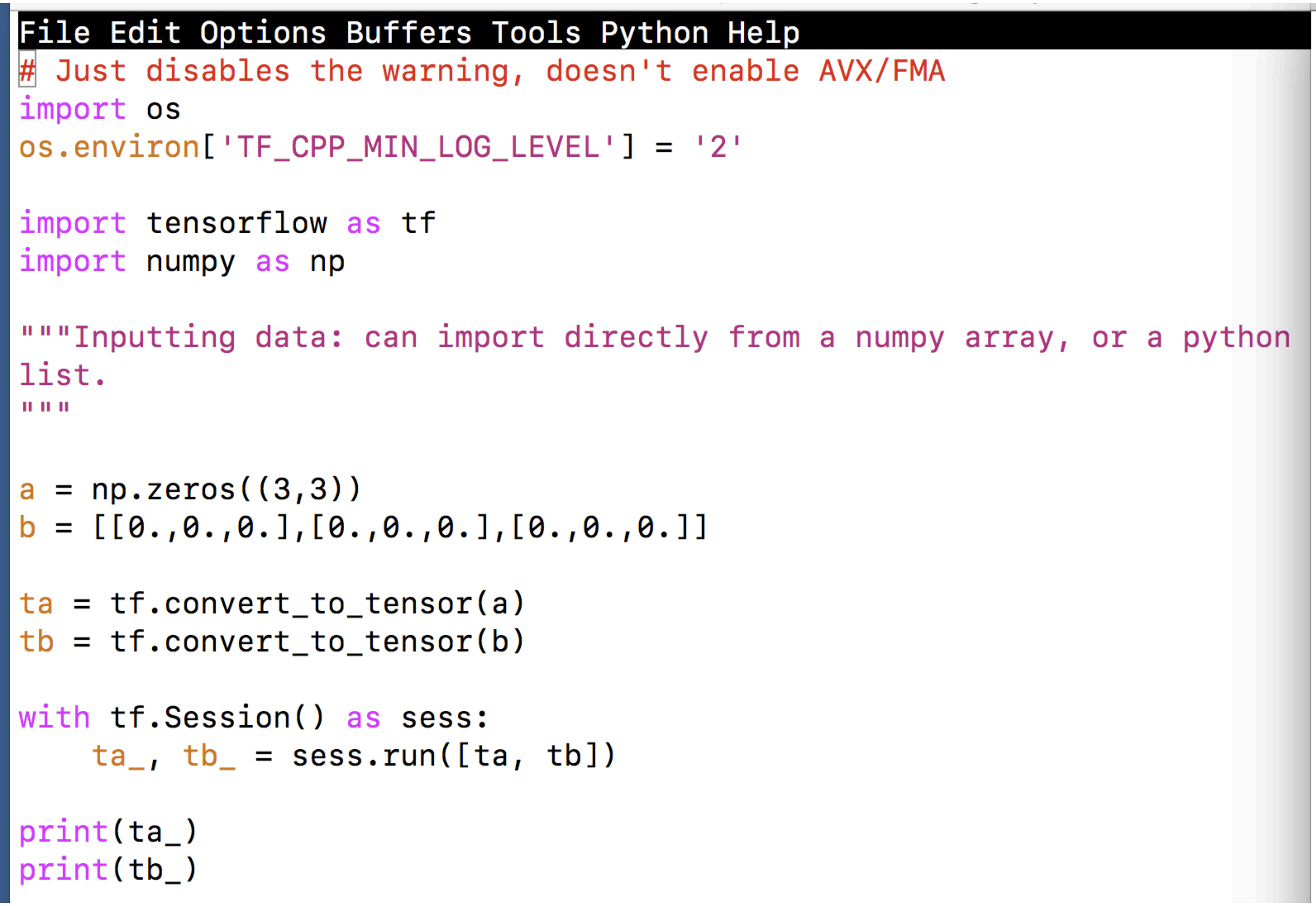
Updating Variables



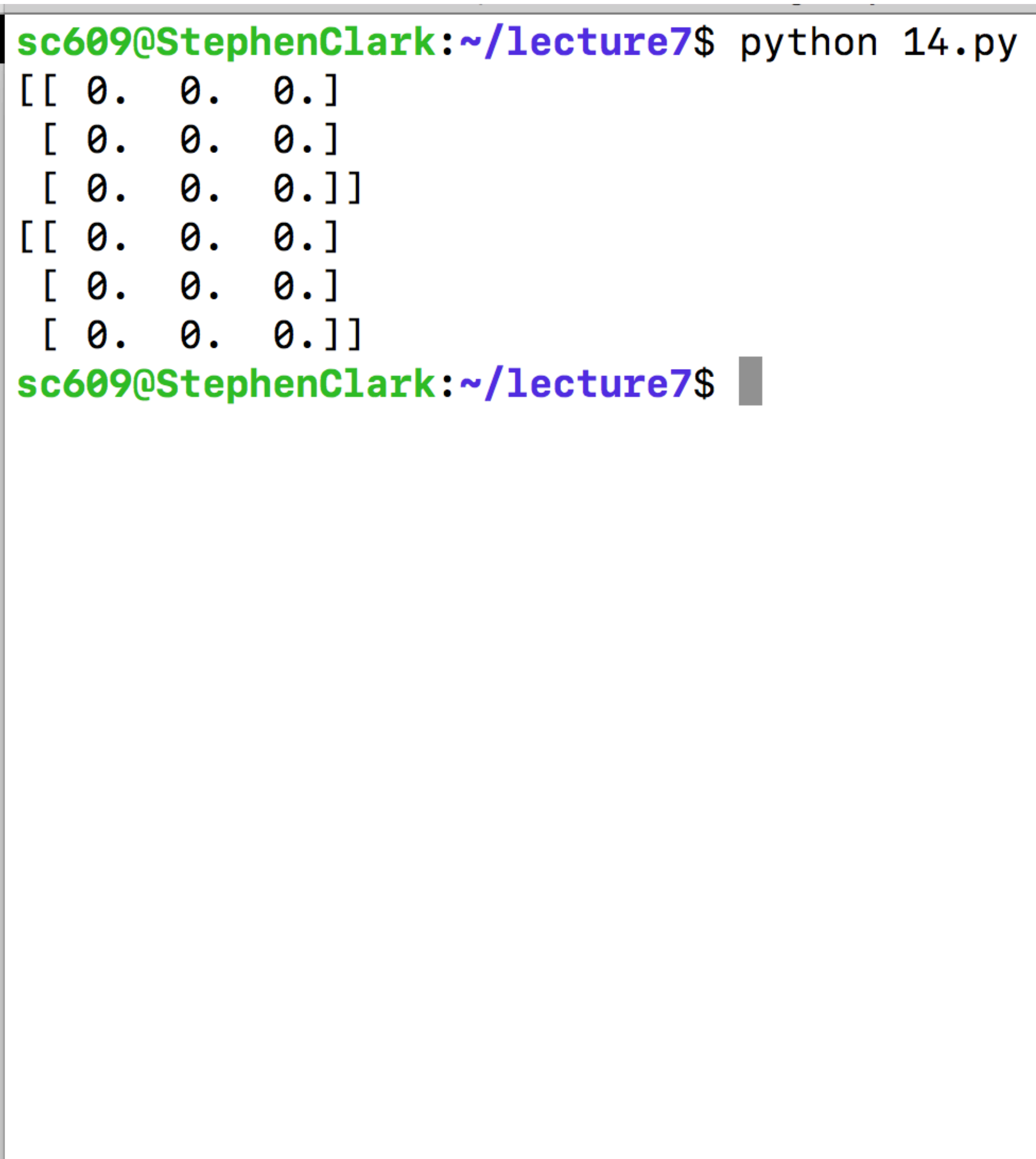
```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf

"""When you train a model you use variables to hold and update
parameters. Variables are in-memory buffers containing tensors
(TensorFlow Doc).
"""

state = tf.Variable(0, name="counter")

new_value = tf.add(state, tf.constant(1))

update = tf.assign(state, new_value)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(state))
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))
```

```
sc609@StephenClark:~/lecture7$ python 13.py
0
1
2
3
sc609@StephenClark:~/lecture7$
```

Importing Data

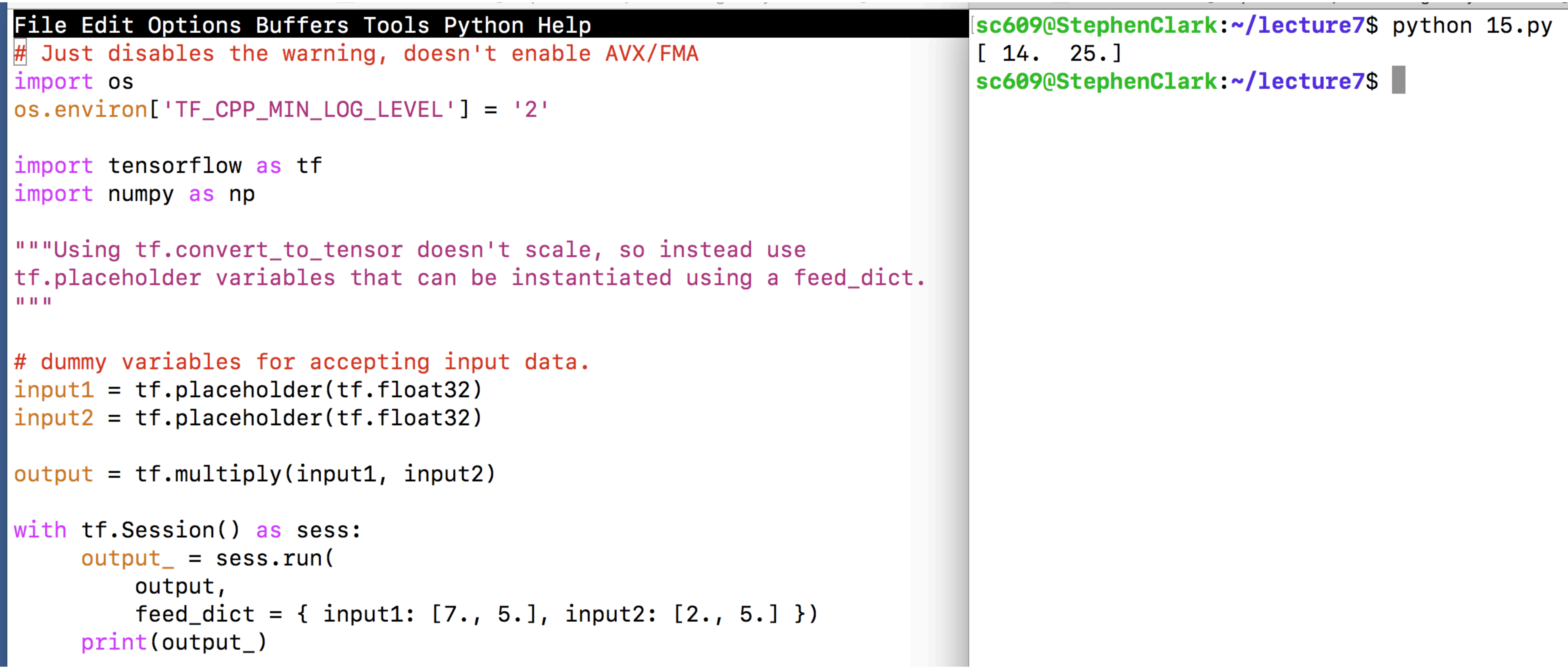


```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

"""Inputting data: can import directly from a numpy array, or a python
list.
"""

a = np.zeros((3,3))
b = [[0.,0.,0.],[0.,0.,0.],[0.,0.,0.]]

ta = tf.convert_to_tensor(a)
tb = tf.convert_to_tensor(b)

with tf.Session() as sess:
    ta_, tb_ = sess.run([ta, tb])

print(ta_)
print(tb_)
```



```
sc609@StephenClark:~/lecture7$ python 14.py
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
sc609@StephenClark:~/lecture7$ 
```

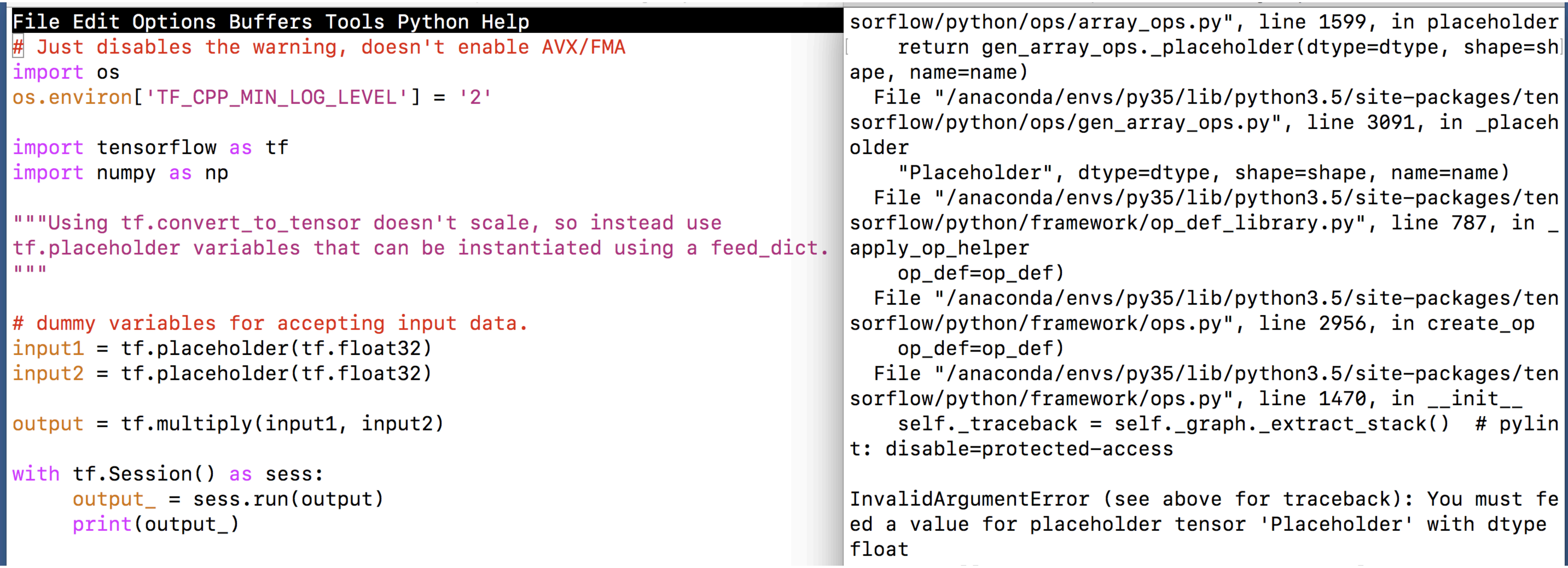
Importing Data



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

"""Using tf.convert_to_tensor doesn't scale, so instead use
tf.placeholder variables that can be instantiated using a feed_dict.
"""

# dummy variables for accepting input data.
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2)

with tf.Session() as sess:
    output_ = sess.run(
        output,
        feed_dict = { input1: [7., 5.], input2: [2., 5.] })
    print(output_)
```

```
sc609@StephenClark:~/lecture7$ python 15.py
[ 14.  25.]
sc609@StephenClark:~/lecture7$
```

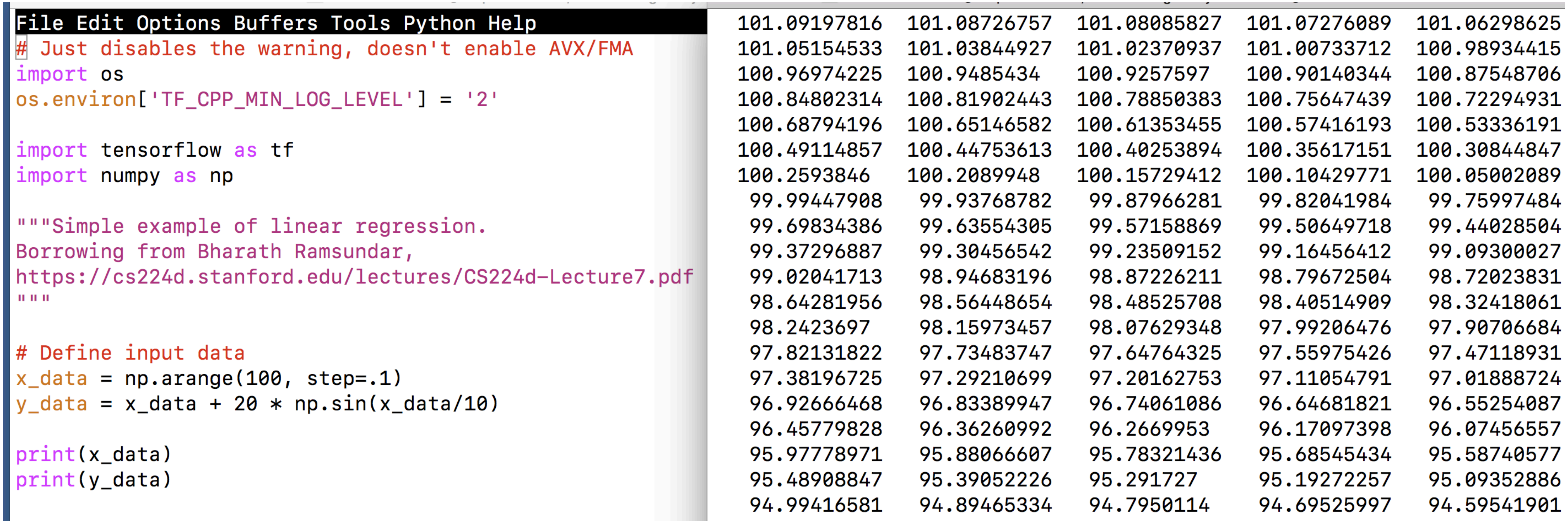
Importing Data



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

"""Using tf.convert_to_tensor doesn't scale, so instead use
tf.placeholder variables that can be instantiated using a feed_dict.
"""

# dummy variables for accepting input data.
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2)

with tf.Session() as sess:
     output_ = sess.run(output)
     print(output_)
```

```
sorflow/python/ops/array_ops.py", line 1599, in placeholder
    return gen_array_ops._placeholder(dtype=dtype, shape=sh
ape, name=name)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/ten
sorflow/python/ops/gen_array_ops.py", line 3091, in _placeh
older
    "Placeholder", dtype=dtype, shape=shape, name=name)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/ten
sorflow/python/framework/op_def_library.py", line 787, in _
apply_op_helper
    op_def=op_def)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/ten
sorflow/python/framework/ops.py", line 2956, in create_op
    op_def=op_def)
  File "/anaconda/envs/py35/lib/python3.5/site-packages/ten
sorflow/python/framework/ops.py", line 1470, in __init__
    self._traceback = self._graph._extract_stack()  # pylin
t: disable=protected-access

InvalidArgumentError (see above for traceback): You must fe
ed a value for placeholder tensor 'Placeholder' with dtype
float
```

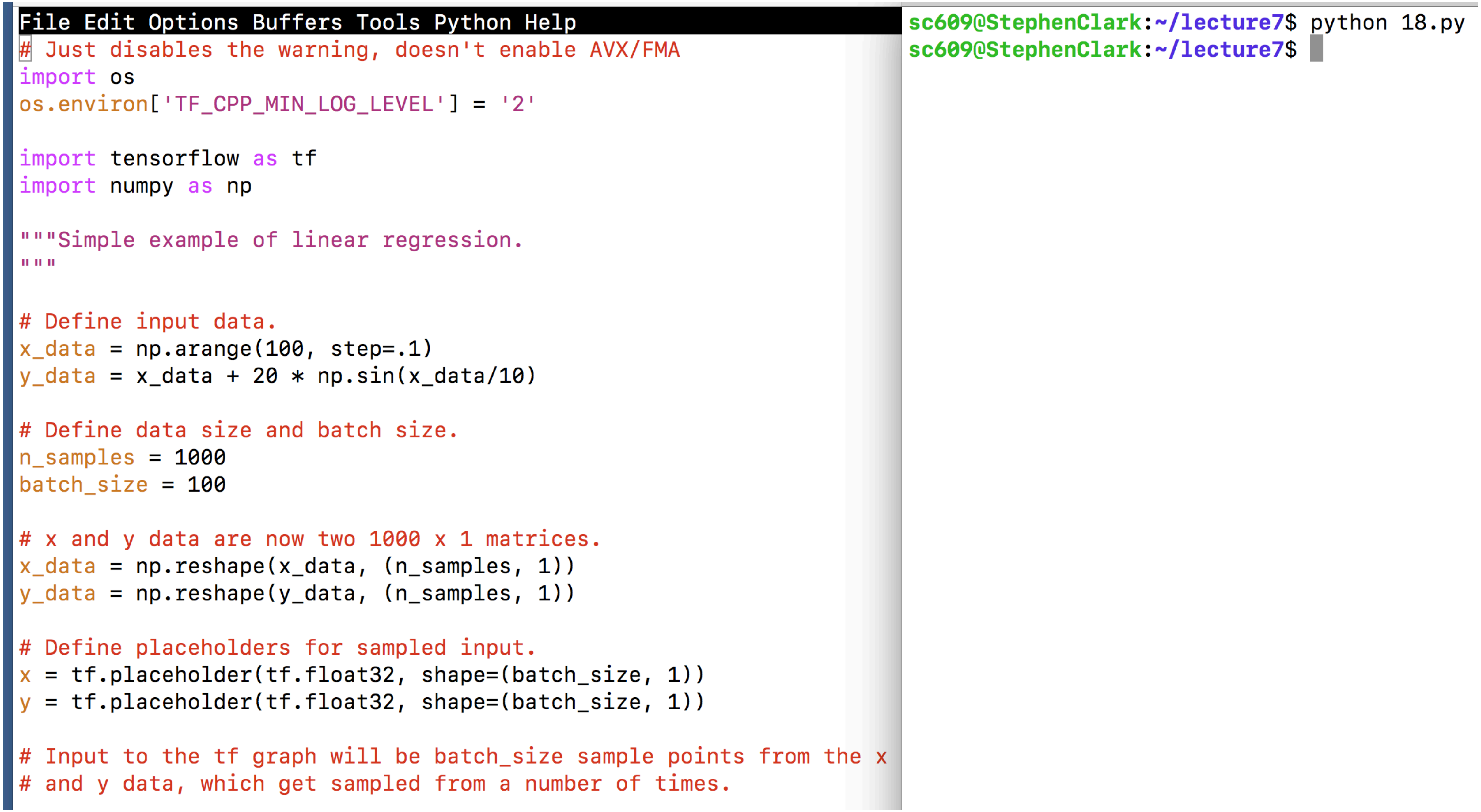
Linear Regression Example



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

"""Simple example of linear regression.
Borrowing from Bharath Ramsundar,
https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf
"""

# Define input data
x_data = np.arange(100, step=.1)
y_data = x_data + 20 * np.sin(x_data/10)

print(x_data)
print(y_data)
```

```
 101.09197816  101.08726757  101.08085827  101.07276089  101.06298625
 101.05154533  101.03844927  101.02370937  101.00733712  100.98934415
 100.96974225  100.9485434   100.9257597   100.90140344  100.87548706
 100.84802314  100.81902443  100.78850383  100.75647439  100.72294931
 100.68794196  100.65146582  100.61353455  100.57416193  100.53336191
 100.49114857  100.44753613  100.40253894  100.35617151  100.30844847
 100.2593846   100.2089948   100.15729412  100.10429771  100.05002089
  99.99447908   99.93768782   99.87966281   99.82041984   99.75997484
  99.69834386   99.63554305   99.57158869   99.50649718   99.44028504
  99.37296887   99.30456542   99.23509152   99.16456412   99.09300027
  99.02041713   98.94683196   98.87226211   98.79672504   98.72023831
  98.64281956   98.56448654   98.48525708   98.40514909   98.32418061
  98.2423697    98.15973457   98.07629348   97.99206476   97.90706684
  97.82131822   97.73483747   97.64764325   97.55975426   97.47118931
  97.38196725   97.29210699   97.20162753   97.11054791   97.01888724
  96.92666468   96.83389947   96.74061086   96.64681821   96.55254087
  96.45779828   96.36260992   96.2669953    96.17097398   96.07456557
  95.97778971   95.88066607   95.78321436   95.68545434   95.58740577
  95.48908847   95.39052226   95.291727     95.19272257   95.09352886
  94.99416581   94.89465334   94.7950114    94.69525997   94.59541901
```

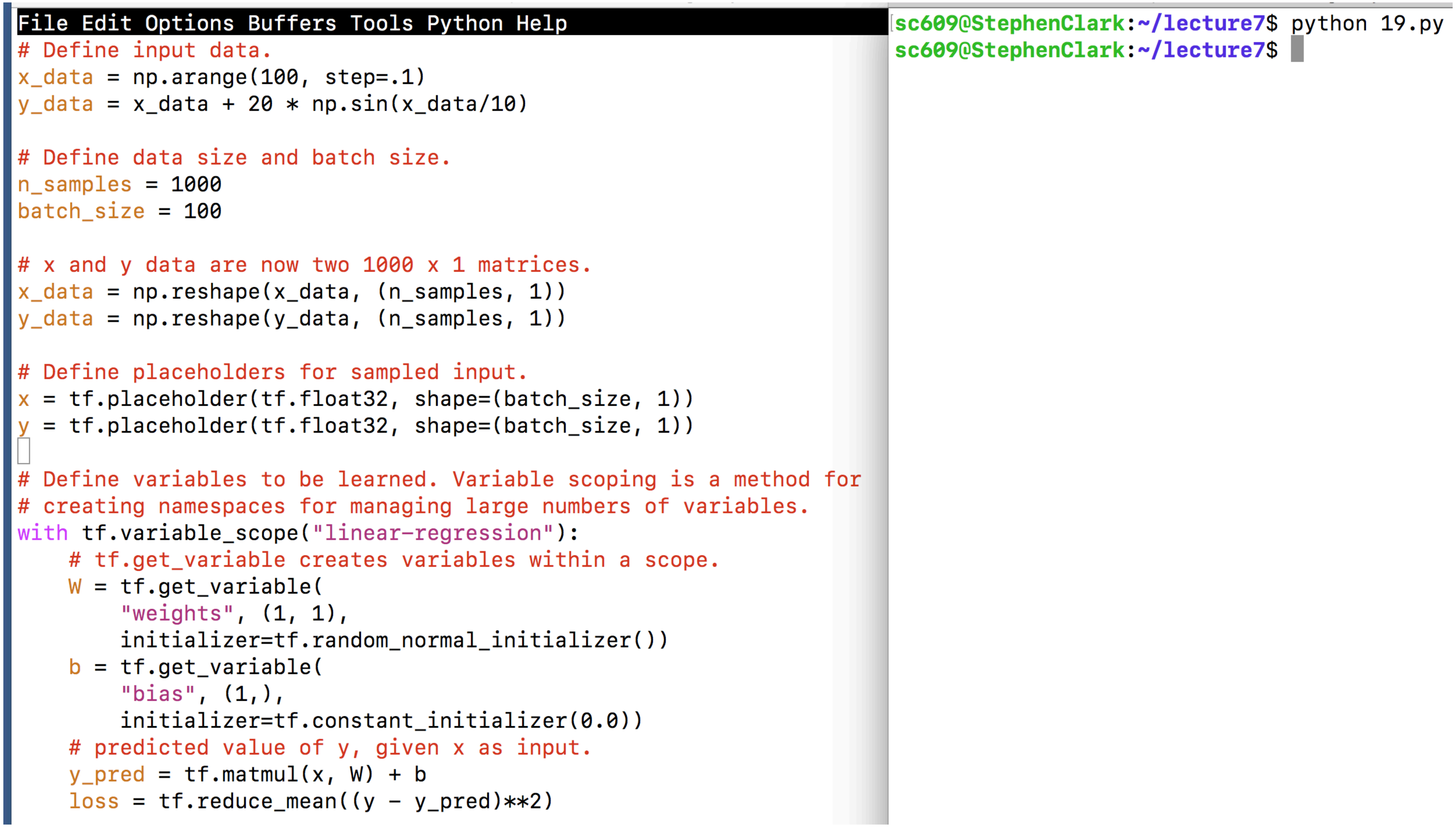
Linear Regression Example



```python
# Just disables the warning, doesn't enable AVX/FMA
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

"""Simple example of linear regression.
"""

# Define input data.
x_data = np.arange(100, step=.1)
y_data = x_data + 20 * np.sin(x_data/10)

# Define data size and batch size.
n_samples = 1000
batch_size = 100

# x and y data are now two 1000 x 1 matrices.
x_data = np.reshape(x_data, (n_samples, 1))
y_data = np.reshape(y_data, (n_samples, 1))

# Define placeholders for sampled input.
x = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))

# Input to the tf graph will be batch_size sample points from the x
# and y data, which get sampled from a number of times.
```

```
sc609@StephenClark:~/lecture7$ python 18.py
sc609@StephenClark:~/lecture7$
```

Linear Regression Example



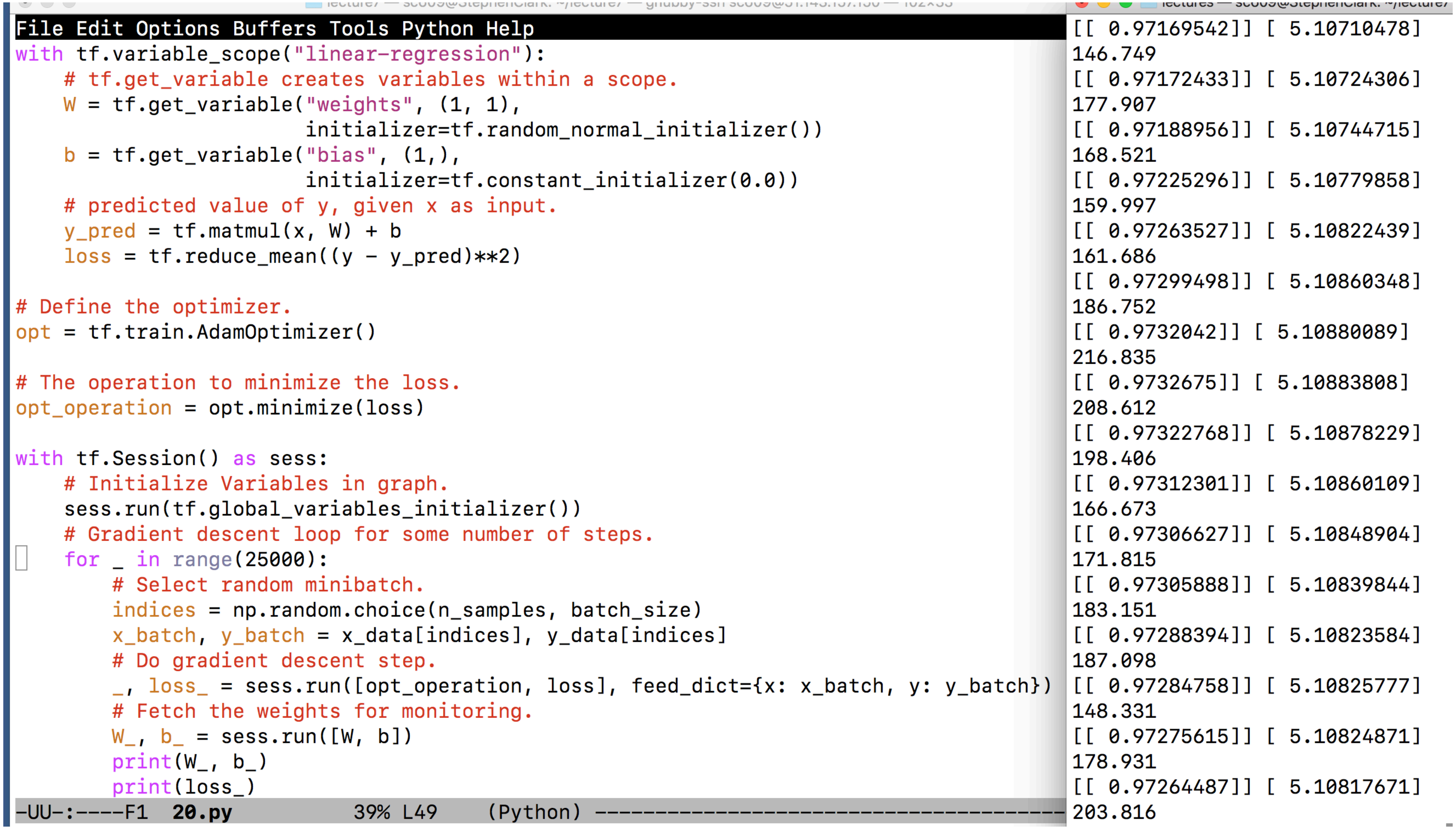
```
# Define input data.
x_data = np.arange(100, step=.1)
y_data = x_data + 20 * np.sin(x_data/10)

# Define data size and batch size.
n_samples = 1000
batch_size = 100

# x and y data are now two 1000 x 1 matrices.
x_data = np.reshape(x_data, (n_samples, 1))
y_data = np.reshape(y_data, (n_samples, 1))

# Define placeholders for sampled input.
x = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))

# Define variables to be learned. Variable scoping is a method for
# creating namespaces for managing large numbers of variables.
with tf.variable_scope("linear-regression"):
    # tf.get_variable creates variables within a scope.
    W = tf.get_variable(
        "weights", (1, 1),
        initializer=tf.random_normal_initializer())
    b = tf.get_variable(
        "bias", (1,),
        initializer=tf.constant_initializer(0.0))
    # predicted value of y, given x as input.
    y_pred = tf.matmul(x, W) + b
    loss = tf.reduce_mean((y - y_pred)**2)
```

```
sc609@StephenClark:~/lecture7$ python 19.py
sc609@StephenClark:~/lecture7$
```

DeepMind

UNIVERSITY OF CAMBRIDGE

Linear Regression Example


File Edit Options Buffers Tools Python Help
-UU-:----F1 20.py 39% L49 (Python) --

```python
with tf.variable_scope("linear-regression"):
    # tf.get_variable creates variables within a scope.
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1,),
                        initializer=tf.constant_initializer(0.0))
    # predicted value of y, given x as input.
    y_pred = tf.matmul(x, W) + b
    loss = tf.reduce_mean((y - y_pred)**2)

# Define the optimizer.
opt = tf.train.AdamOptimizer()

# The operation to minimize the loss.
opt_operation = opt.minimize(loss)

with tf.Session() as sess:
    # Initialize Variables in graph.
    sess.run(tf.global_variables_initializer())
    # Gradient descent loop for some number of steps.
    for _ in range(25000):
        # Select random minibatch.
        indices = np.random.choice(n_samples, batch_size)
        x_batch, y_batch = x_data[indices], y_data[indices]
        # Do gradient descent step.
        _, loss_ = sess.run([opt_operation, loss], feed_dict={x: x_batch, y: y_batch})
        # Fetch the weights for monitoring.
        W_, b_ = sess.run([W, b])
        print(W_, b_)
        print(loss_)
```

```
[[ 0.97169542]] [ 5.10710478]
146.749
[[ 0.97172433]] [ 5.10724306]
177.907
[[ 0.97188956]] [ 5.10744715]
168.521
[[ 0.97225296]] [ 5.10779858]
159.997
[[ 0.97263527]] [ 5.10822439]
161.686
[[ 0.97299498]] [ 5.10860348]
186.752
[[ 0.9732042]] [ 5.10880089]
216.835
[[ 0.9732675]] [ 5.10883808]
208.612
[[ 0.97322768]] [ 5.10878229]
198.406
[[ 0.97312301]] [ 5.10860109]
166.673
[[ 0.97306627]] [ 5.10848904]
171.815
[[ 0.97305888]] [ 5.10839844]
183.151
[[ 0.97288394]] [ 5.10823584]
187.098
[[ 0.97284758]] [ 5.10825777]
148.331
[[ 0.97275615]] [ 5.10824871]
178.931
[[ 0.97264487]] [ 5.10817671]
203.816
```