Tensorflow (1.x) Review Session

CS330: Deep Multi-task and Meta Learning
9/26/2019
Suraj Nair

Overview

- Installation
- Tensorflow Basics
    - Variables / Placeholders / Constants / Operations
    - Graphs / Sessions
    - Optimizers
    - Training loop
    - MNIST Example
- High Level APIs
    - tf.layers, Keras
- Advanced features
    - Memory layers
    - Manually extracting gradients
    - Distributions
    - Eager execution

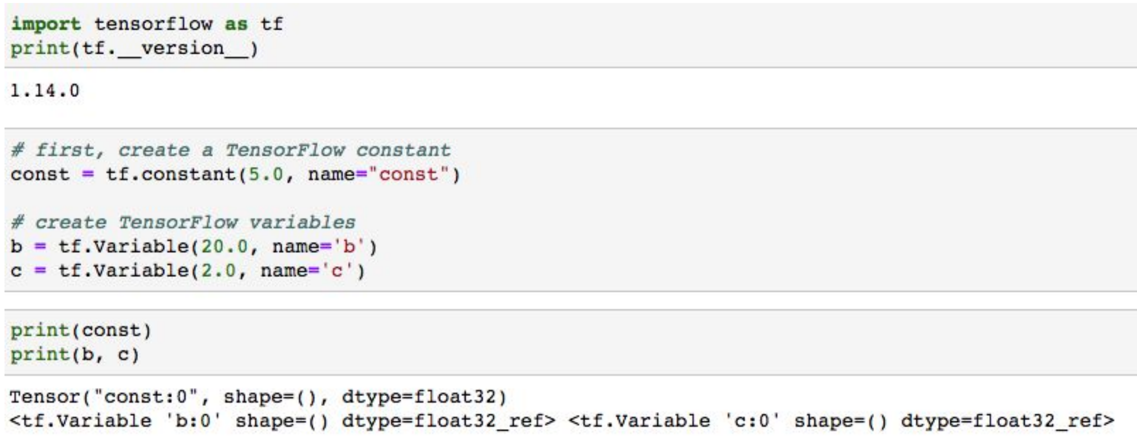
Some examples adapted from
https://adventuresinmachinelearning.com/python-tensorflow-tutorial/

Installation

- In this class we will use Python 3.5+, tensorflow 1.14.0
- `pip install tensorflow(-gpu)==1.14.0`
- We recommend a python virtual environment
  - virtualenv
  - Anaconda
- Notes about GPUs:
  - Tf 1.14 requires CUDA 10
  - If you want to run your code on a gpu you will need to make sure CUDA 10 is installed (CUDA 9 will not work)

Tensorflow Basics: Variables/Constants/Operations

- Standard definition of a constant or variable
    - A constant is fixed
    - A variable can be assigned to any value, can be optimized.
    - Both can be used in a computation graph.
- Operations can be done on Variables to define new Variables



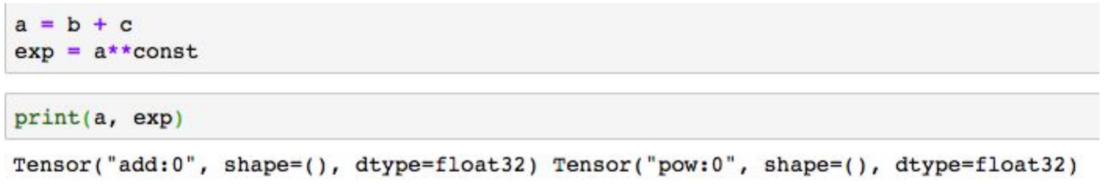
```
import tensorflow as tf
print(tf.__version__)
```

```
1.14.0
```

```
# first, create a TensorFlow constant
const = tf.constant(5.0, name="const")

# create TensorFlow variables
b = tf.Variable(20.0, name='b')
c = tf.Variable(2.0, name='c')
```

```
print(const)
print(b, c)
```

```
Tensor("const:0", shape=(), dtype=float32)
<tf.Variable 'b:0' shape=() dtype=float32_ref> <tf.Variable 'c:0' shape=() dtype=float32_ref>
```



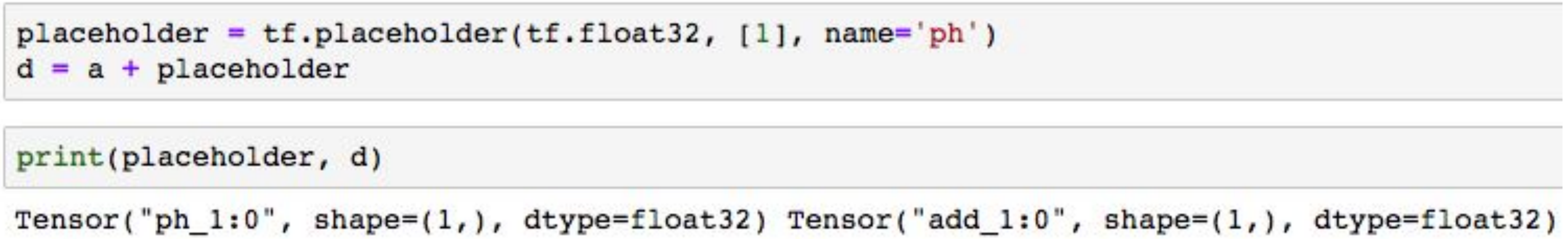
```
a = b + c
exp = a**const
```

```
print(a, exp)
```

```
Tensor("add:0", shape=(), dtype=float32) Tensor("pow:0", shape=(), dtype=float32)
```

Tensorflow Basics: Placeholders

- What if you don't know the value of a variable yet?
- Placeholder in a graph is a variable whose value can be filled in later



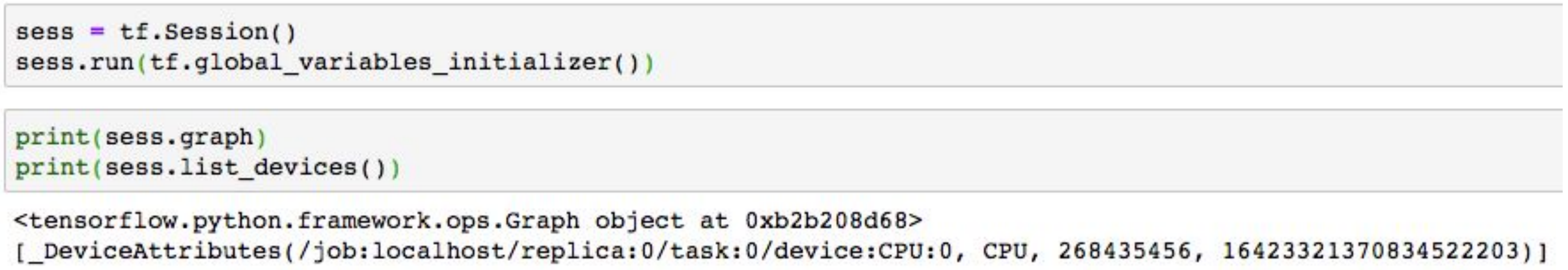
```
placeholder = tf.placeholder(tf.float32, [1], name='ph')
d = a + placeholder
```

```
print(placeholder, d)
```

```
Tensor("ph_1:0", shape=(1,), dtype=float32) Tensor("add_1:0", shape=(1,), dtype=float32)
```

Tensorflow Basics: Graphs/Sessions

- A set of variables/constants/placeholders and the operations between them form a **computation graph -** stored as a tf.Graph object
- A tf.Session stores a graph, as well as computation information (i.e. GPU/CPU)
- tf.Session.run(Variable) runs the computation graph and outputs the value of the variable
- Need to create session and initialize the variables

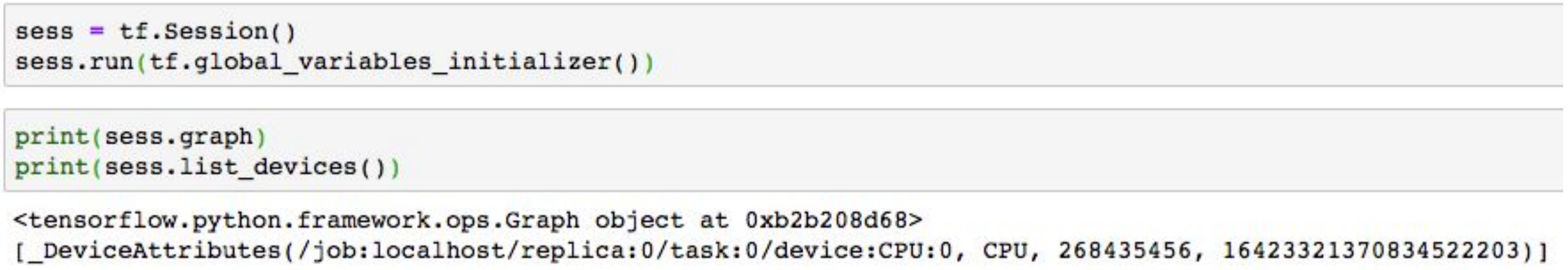


```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
print(sess.graph)
print(sess.list_devices())
```

```
<tensorflow.python.framework.ops.Graph object at 0xb2b208d68>
[_DeviceAttributes(/job:localhost/replica:0/task:0/device:CPU:0, CPU, 268435456, 16423321370834522203)]
```

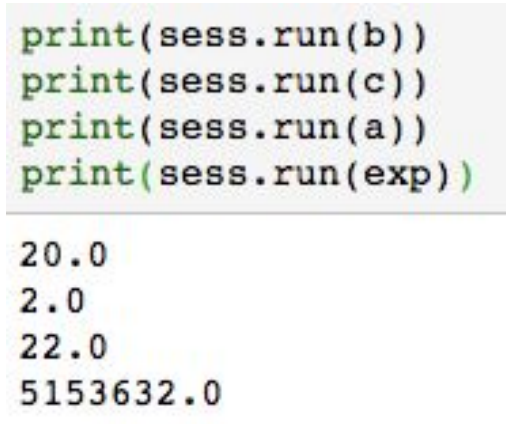
Tensorflow Basics: Graphs/Sessions



```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
print(sess.graph)
print(sess.list_devices())
```

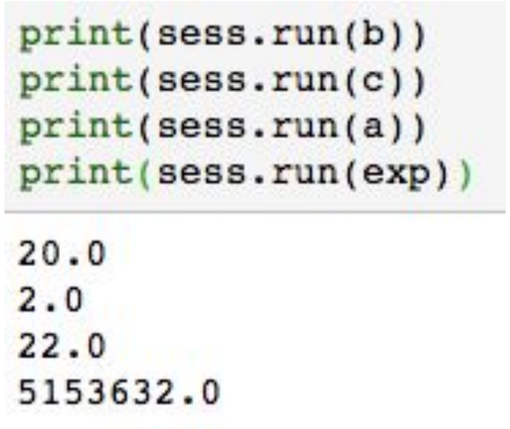
```
<tensorflow.python.framework.ops.Graph object at 0xb2b208d68>
[_DeviceAttributes(/job:localhost/replica:0/task:0/device:CPU:0, CPU, 268435456, 16423321370834522203)]
```



```
print(sess.run(b))
print(sess.run(c))
print(sess.run(a))
print(sess.run(exp))
```

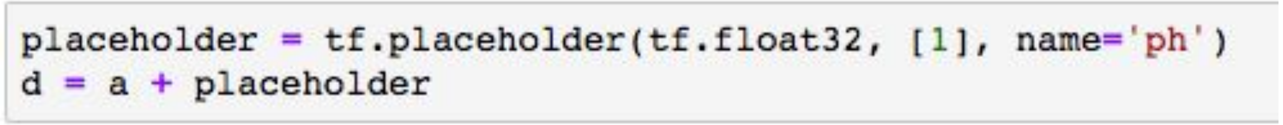
```
20.0
2.0
22.0
5153632.0
```

Tensorflow Basics: Graphs/Sessions

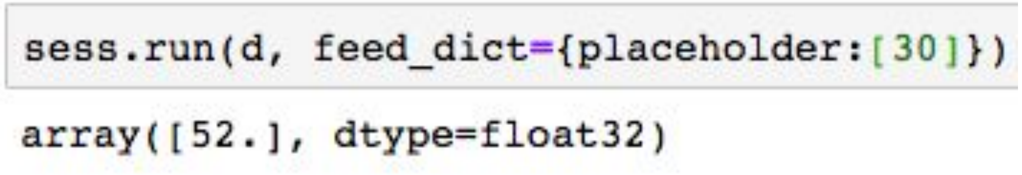


```
print(sess.run(b))
print(sess.run(c))
print(sess.run(a))
print(sess.run(exp))
```

```
20.0
2.0
22.0
5153632.0
```



```
placeholder = tf.placeholder(tf.float32, [1], name='ph')
d = a + placeholder
```

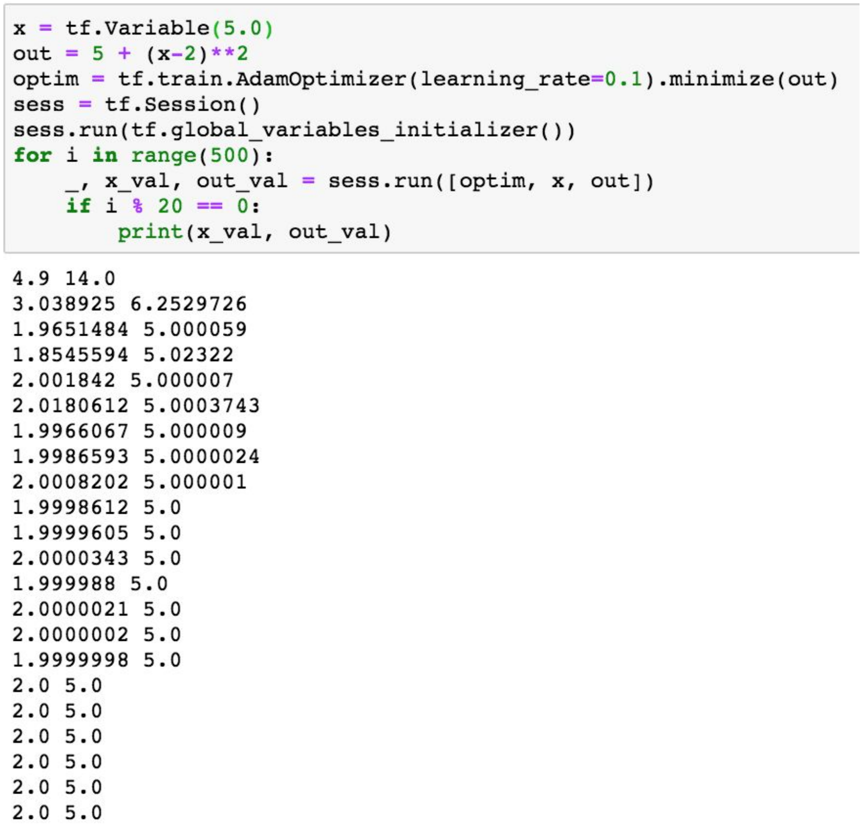


```
sess.run(d, feed_dict={placeholder:[30]})
```

```
array([52.], dtype=float32)
```

Tensorflow Basics: Optimizers

- Given a node in the graph, you can optimize the variables in the graph to minimize/maximize the node.
- Standard set of optimizers
  - Adam
  - SGD
- When you run the optimizer using sess.run, computes gradients and applies them

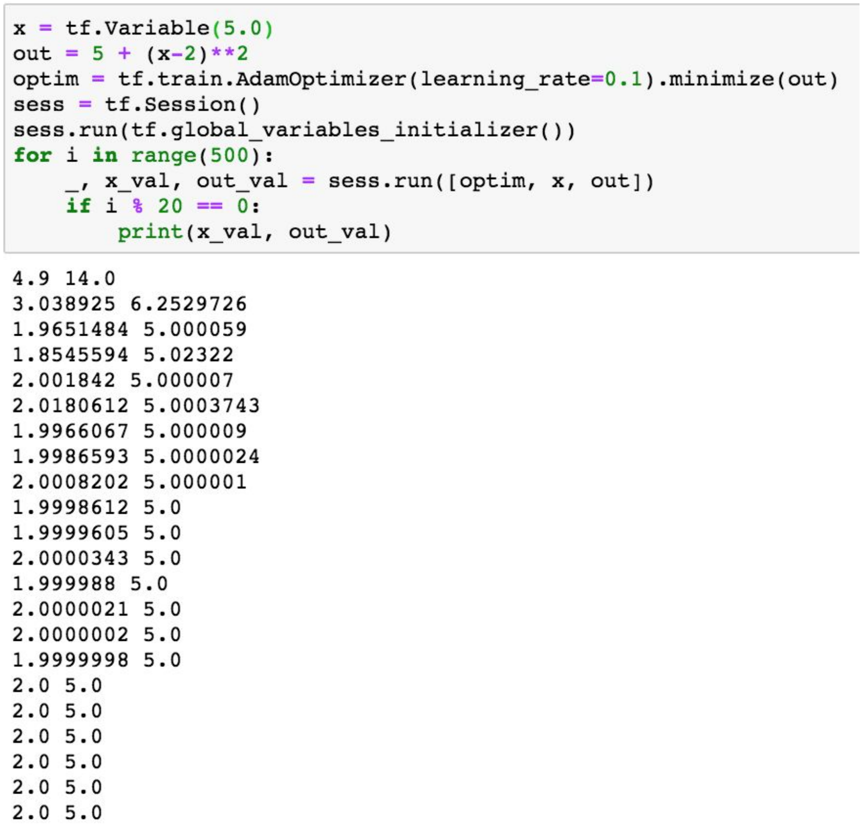


```python
x = tf.Variable(5.0)
out = 5 + (x-2)**2
optim = tf.train.AdamOptimizer(learning_rate=0.1).minimize(out)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(500):
    _, x_val, out_val = sess.run([optim, x, out])
    if i % 20 == 0:
        print(x_val, out_val)
```

```
4.9 14.0
3.038925 6.2529726
1.9651484 5.000059
1.8545594 5.02322
2.001842 5.000007
2.0180612 5.0003743
1.9966067 5.000009
1.9986593 5.0000024
2.0008202 5.000001
1.9998612 5.0
1.9999605 5.0
2.0000343 5.0
1.999988 5.0
2.0000021 5.0
2.0000002 5.0
1.9999998 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
```

Tensorflow Basics: Optimizers/Training Loop

- Given a node in the graph, you can optimize the variables in the graph to minimize/maximize the node.
- Standard set of optimizers
  - Adam
  - SGD
- When you run the optimizer using sess.run, computes gradients and applies them
- Normally data inputs will be placeholders



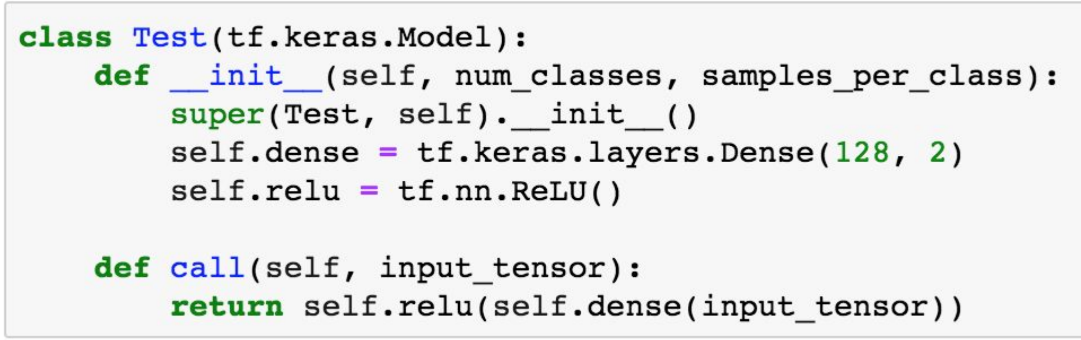
```python
x = tf.Variable(5.0)
out = 5 + (x-2)**2
optim = tf.train.AdamOptimizer(learning_rate=0.1).minimize(out)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(500):
    _, x_val, out_val = sess.run([optim, x, out])
    if i % 20 == 0:
        print(x_val, out_val)
```

```
4.9 14.0
3.038925 6.2529726
1.9651484 5.000059
1.8545594 5.02322
2.001842 5.000007
2.0180612 5.0003743
1.9966067 5.000009
1.9986593 5.0000024
2.0008202 5.000001
1.9998612 5.0
1.9999605 5.0
2.0000343 5.0
1.999988 5.0
2.0000021 5.0
2.0000002 5.0
1.9999998 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
2.0 5.0
```

Tensorflow Basics: MNIST Example

High Level APIs

- Writing out every Variable in a big network is time consuming
- Some helpful wrapper functions which contain standard network layers
  - Tf.layers
    - tf.layers.Dense
    - tf.layers.Conv2D
  - Tf.keras
    - tf.keras.Conv2D
    - tf.keras.layers.ConvLstm2D
- Wrapper loss functions
  - Cross Entropy loss
  - Mean squared error



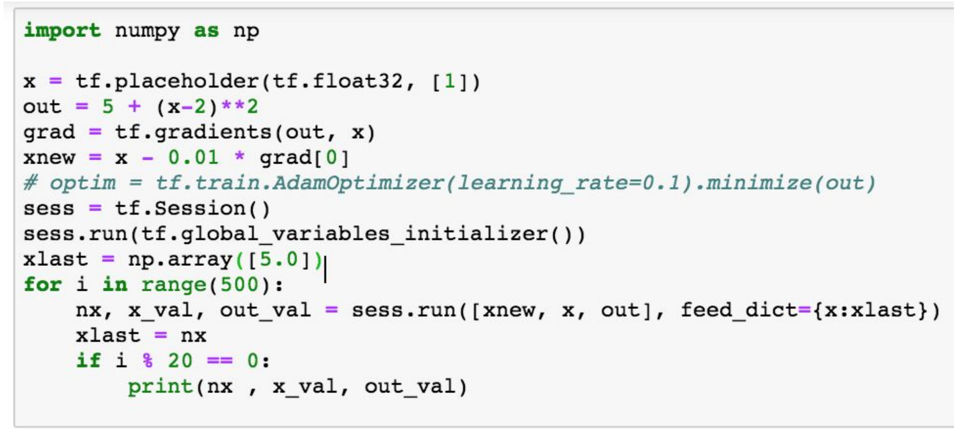
```python
class Test(tf.keras.Model):
    def __init__(self, num_classes, samples_per_class):
        super(Test, self).__init__()
        self.dense = tf.keras.layers.Dense(128, 2)
        self.relu = tf.nn.ReLU()

    def call(self, input_tensor):
        return self.relu(self.dense(input_tensor))
```

Advanced Features: Memory

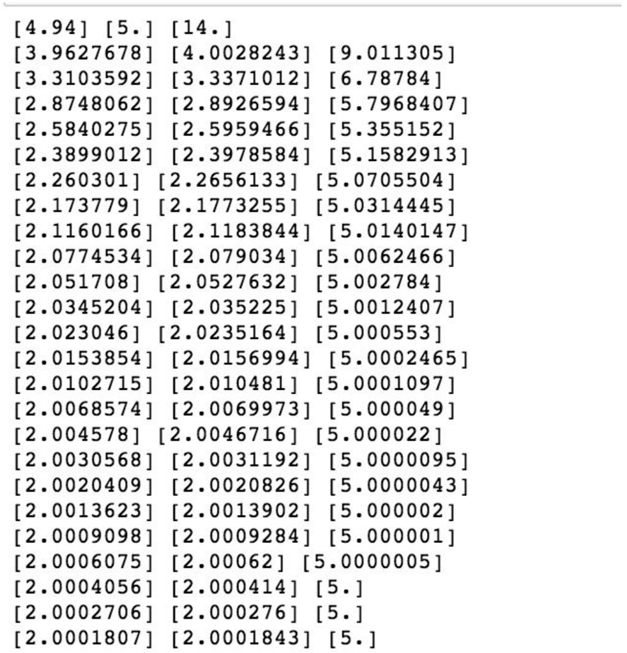
- Recurrent Cell:
  - lstm_cell = tf.keras.layers.LSTMCell(hidden_size)
  - output, hidden_state = lstm_cell(input, hidden_state)
- Or use a recurrent layer:
  - lstmlayer = tf.keras.layers.LSTM(hidden_size, return_sequences=True)
  - output = lstmlayer(input)
    - Where input is (batch_size, seq_len, input_dim)
    - And output is (batch_size, seq_len, hidden_size)

Advanced Features: Using Gradients

- Can directly get gradient da / db using tf.gradients(a , b)
- For example - can do gradient descent without using optimizer



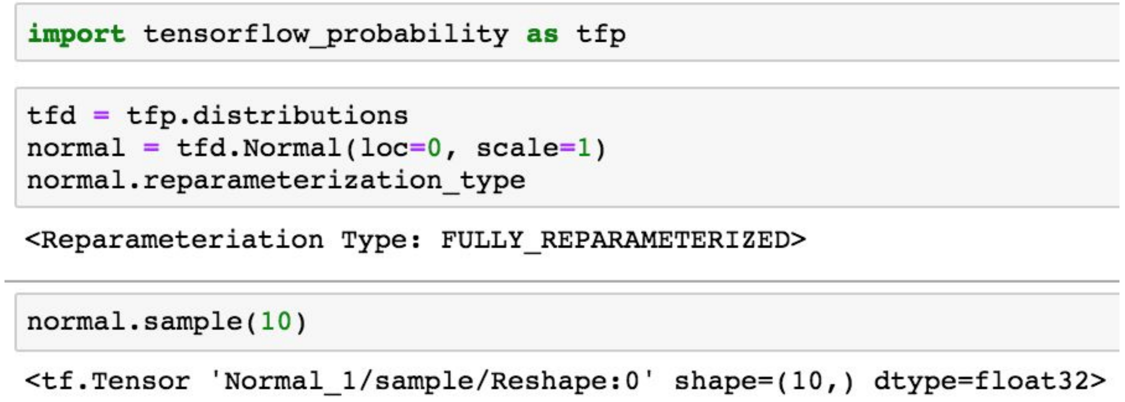
```python
import numpy as np

x = tf.placeholder(tf.float32, [1])
out = 5 + (x-2)**2
grad = tf.gradients(out, x)
xnew = x - 0.01 * grad[0]
# optim = tf.train.AdamOptimizer(learning_rate=0.1).minimize(out)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
xlast = np.array([5.0])
for i in range(500):
    nx, x_val, out_val = sess.run([xnew, x, out], feed_dict={x:xlast})
    xlast = nx
    if i % 20 == 0:
        print(nx , x_val, out_val)
```



```
[4.94] [5.] [14.]
[3.9627678] [4.0028243] [9.011305]
[3.3103592] [3.3371012] [6.78784]
[2.8748062] [2.8926594] [5.7968407]
[2.5840275] [2.5959466] [5.355152]
[2.3899012] [2.3978584] [5.1582913]
[2.260301] [2.2656133] [5.0705504]
[2.173779] [2.1773255] [5.0314445]
[2.1160166] [2.1183844] [5.0140147]
[2.0774534] [2.079034] [5.0062466]
[2.051708] [2.0527632] [5.002784]
[2.0345204] [2.035225] [5.0012407]
[2.023046] [2.0235164] [5.000553]
[2.0153854] [2.0156994] [5.0002465]
[2.0102715] [2.010481] [5.0001097]
[2.0068574] [2.0069973] [5.000049]
[2.004578] [2.0046716] [5.000022]
[2.0030568] [2.0031192] [5.0000095]
[2.0020409] [2.0020826] [5.0000043]
[2.0013623] [2.0013902] [5.000002]
[2.0009098] [2.0009284] [5.000001]
[2.0006075] [2.00062] [5.0000005]
[2.0004056] [2.000414] [5.]
[2.0002706] [2.000276] [5.]
[2.0001807] [2.0001843] [5.]
```

Advanced Features: Distributions

- Often you may need distributions
    - Training a generative model
    - Stochastic policies
- Used to be tf.Distributions - now tensorflow_probability.Distributions
    - `pip install tensorflow-probability`
- If reparameterized - can backpropagate through sample()



```
import tensorflow_probability as tfp
```

```
tfd = tfp.distributions
normal = tfd.Normal(loc=0, scale=1)
normal.reparameterization_type
```

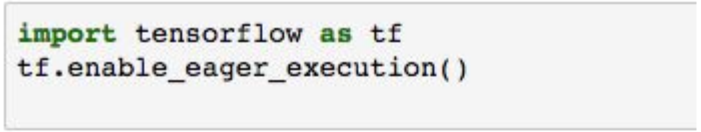
```
<Reparameteriation Type: FULLY_REPARAMETERIZED>
```

```
normal.sample(10)
```

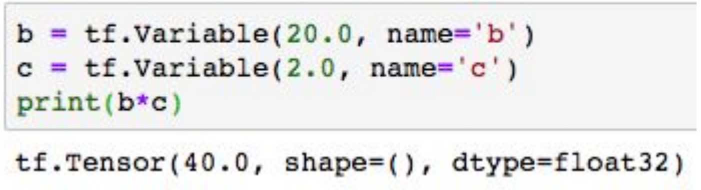
```
<tf.Tensor 'Normal_1/sample/Reshape:0' shape=(10,) dtype=float32>
```

Advanced Features: Eager Execution

- Standard TF uses a static graph
  - Graph is built and fixed
    - Makes it fast to compute gradients/feedforward since graph is static
- Eager execution
  - Graph is built dynamically
  - Track gradients with tf.GradientTape

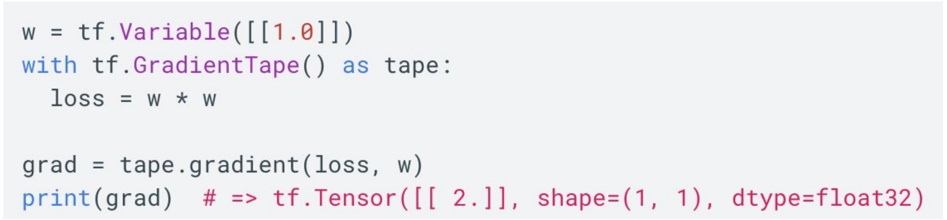


```
import tensorflow as tf
tf.enable_eager_execution()
```



```
b = tf.Variable(20.0, name='b')
c = tf.Variable(2.0, name='c')
print(b*c)
```

```
tf.Tensor(40.0, shape=(), dtype=float32)
```



```
w = tf.Variable([[1.0]])
with tf.GradientTape() as tape:
  loss = w * w

grad = tape.gradient(loss, w)
print(grad)  # => tf.Tensor([[ 2.]], shape=(1, 1), dtype=float32)
```