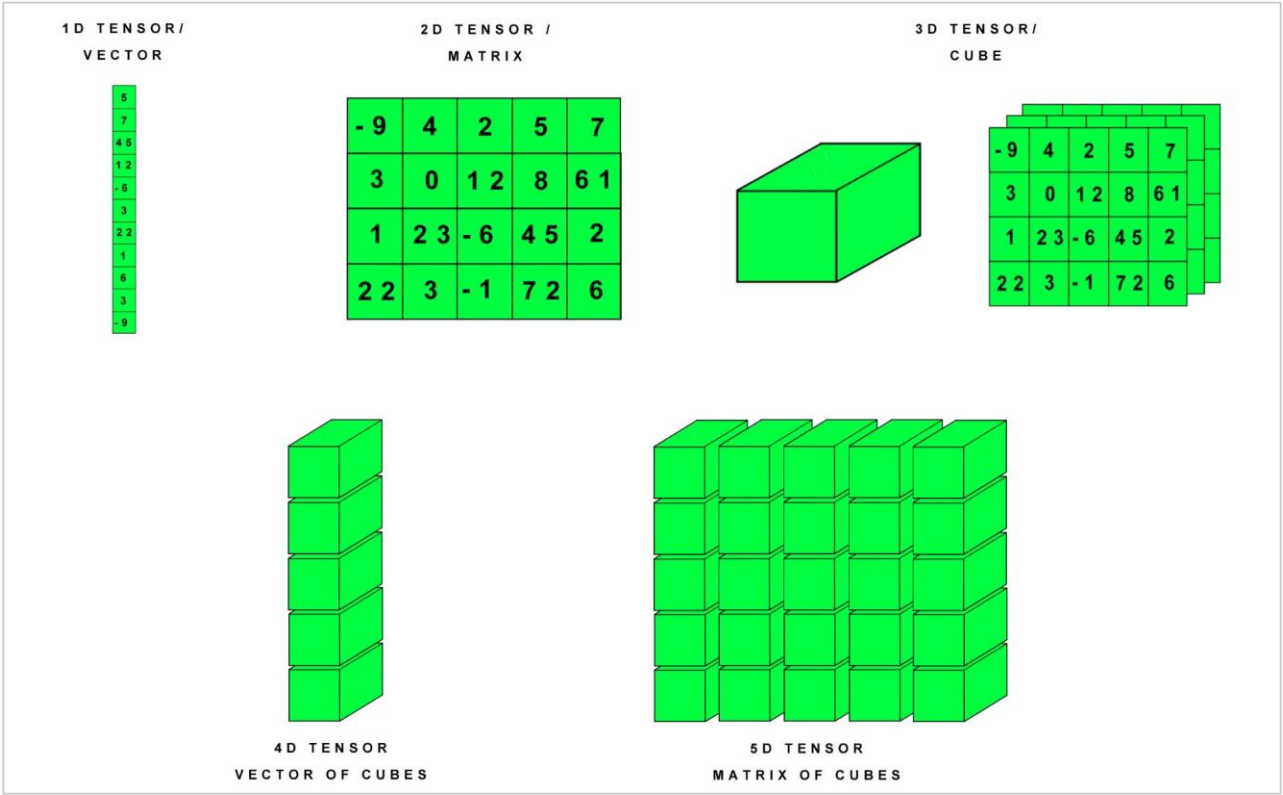
Tensorflow Overview

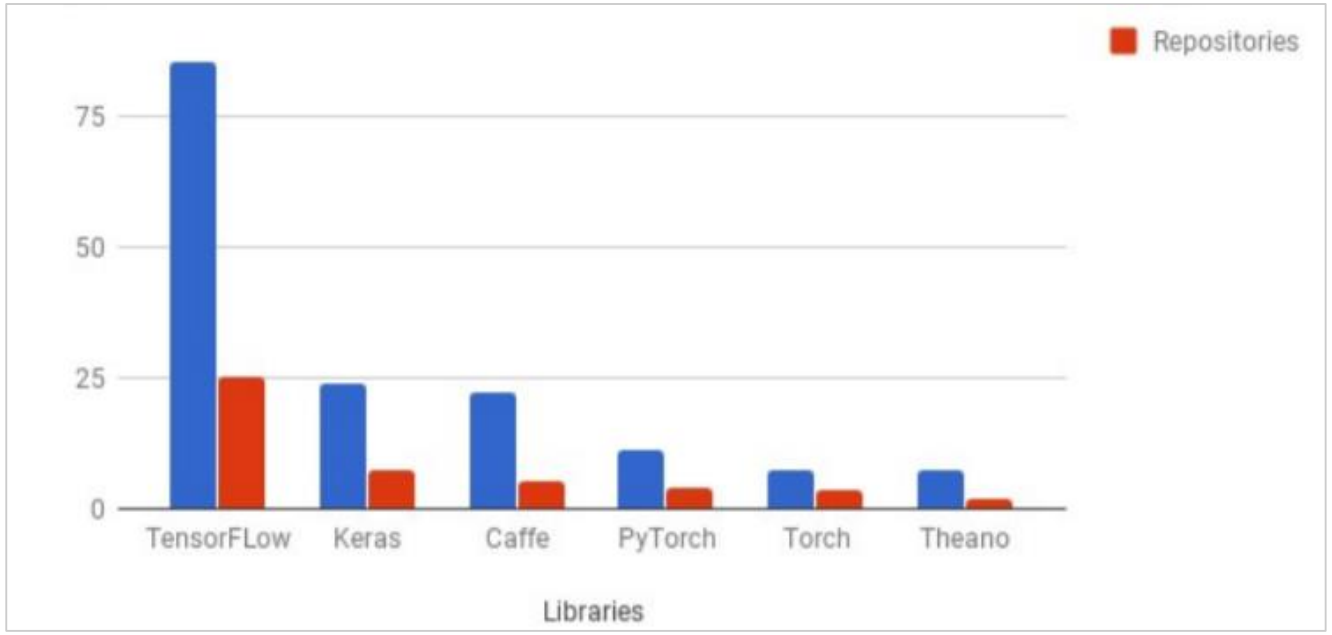
Contents

What is a tensor?



- Formally, tensors are multilinear maps from vector spaces to real numbers
- A tensor can be represented as a multidimensional array of numbers. (Matrix, vector or scalar)

What is tensorflow?



- Open source software library for numerical computation using **data flow graphs**
- Other libraries eg Keras, PyTorch, Caffe etc

Tensorflow VS Numpy

- Both are N-d array libraries
- Numpy has Ndarray support, but doesn't offer methods to create tensor functions and automatically compute derivatives (+ no GPU support).

np *vs* tf

NUMPY

```
In [23]: import numpy as np

In [24]: a = np.zeros((2,2)); b = np.ones((2,2))

In [25]: np.sum(b, axis=1)
Out[25]: array([ 2.,  2.])

In [26]: a.shape
Out[26]: (2, 2)

In [27]: np.reshape(a, (1,4))
Out[27]: array([[ 0.,  0.,  0.,  0.]])
```

TENSORFLOW

```
In [31]: import tensorflow as tf

In [32]: tf.InteractiveSession()

In [33]: a = tf.zeros((2,2)); b = tf.ones((2,2))

In [34]: tf.reduce_sum(b, reduction_indices=1).eval()
Out[34]: array([ 2.,  2.], dtype=float32)

In [35]: a.get_shape()
Out[35]: TensorShape([Dimension(2), Dimension(2)])

In [36]: tf.reshape(a, (1, 4)).eval()
Out[36]: array([[ 0.,  0.,  0.,  0.]], dtype=float32)
```

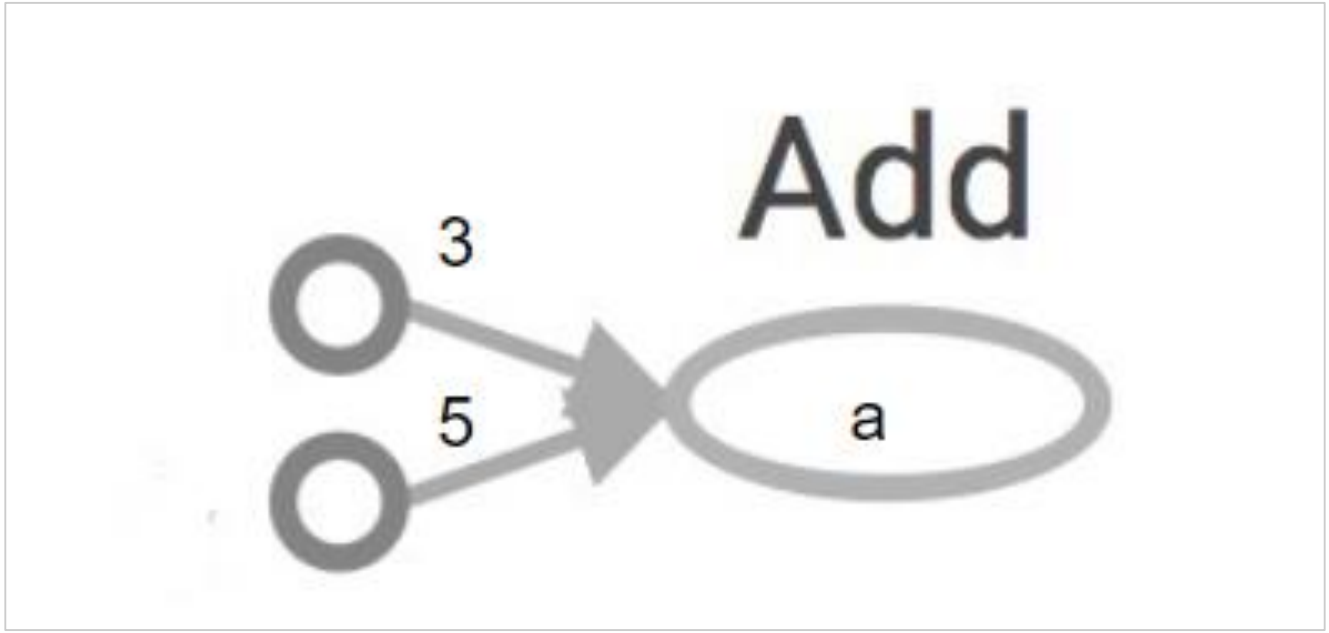
Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

Data Flow Graphs

Tensorflow Computation / Data flow Graph

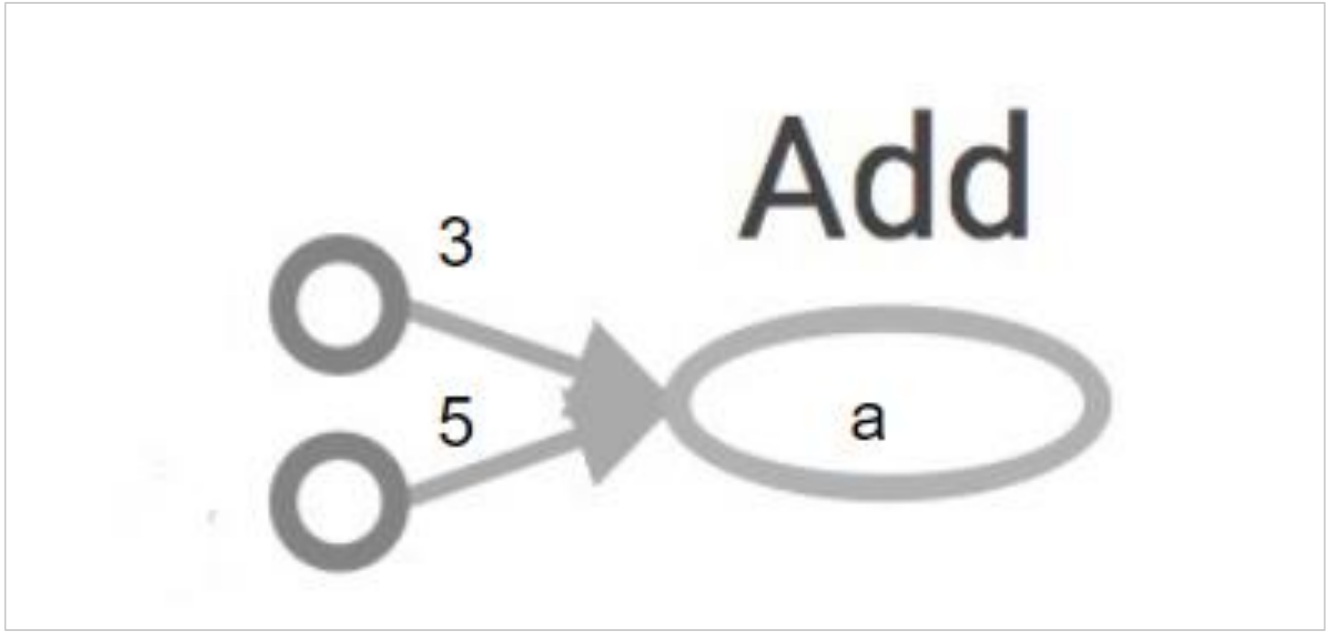
- "TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph." - TensorFlow docs
- All computations add nodes to global default graph

Simple Computation Graph



- import tensorflow as tf
- a = tf.add(3, 5)
- print(a)
- >>

Simple Computation Graph



- import tensorflow as tf
- a = tf.add(3, 5)
- print(a)
- >> Tensor("Add:0", shape=(), dtype=int32) #(Not 8)

HOW TO GET VALUE OF ‘a’ ???

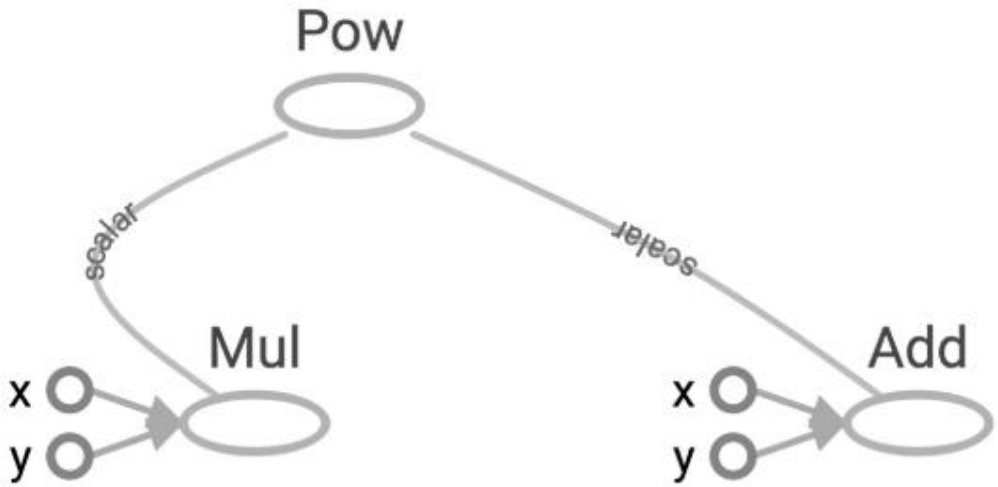
HOW TO GET VALUE OF ‘a’ ???

- Create a **session**, assign it to variable sess so we can call it later
- Within the session, evaluate the graph to fetch the value of a
- import tensorflow as tf
  a = tf.add(3, 5)
  with tf.Session() as sess:
  print(sess.run(a))

tf.Session()

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
- Session will also allocate memory to store the current values of variables.

Another Example



- x = 2
  y = 3
  op1 = tf.add(x, y)
  op2 = tf.multiply(x, y)
  op3 = tf.pow(op2, op1)
  with tf.Session() as sess:
  op3 = sess.run(op3) # (x*y) ^ (x+y)

Why graphs?

- Save computation, run subgraphs that lead to the values you want to fetch only
- Break computation into small, differential pieces to facilitate auto-differentiation
- Facilitate distributed computation, spread the work across multiple CPUs, GPUs, TPUs, or other devices

tf.constants & tf.Variable

tf.constants

```
In [95]: a = tf.constant(value=6.0, name='scalar', dtype=tf.float32, shape=[])
         print(a)

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

```
In [96]: a = tf.constant(value=6.0, name='array', dtype=tf.float32, shape=[2])
         print(a)

         Tensor("array:0", shape=(2,), dtype=float32)
```

```
In [97]: a = tf.constant(value=6.0, name='matrix', dtype=tf.float32, shape=[2,2])
         print(a)

         Tensor("matrix:0", shape=(2, 2), dtype=float32)
```

tf.constants

Useful constants

In [101]:
```
a = tf.zeros(shape=[3,2], dtype=tf.float32, name='matrix')
print(a)
with tf.Session() as sess:
    a_val = sess.run(a)
    print(a_val)
```

```
Tensor("matrix_4:0", shape=(3, 2), dtype=float32)
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
```

In [102]:
```
a = tf.ones(shape=[3,2], dtype=tf.float32, name='matrix')
print(a)
with tf.Session() as sess:
    a_val = sess.run(a)
    print(a_val)
```

```
Tensor("matrix_5:0", shape=(3, 2), dtype=float32)
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
```

tf.constants

Randomly Generated Constants

- tf.set_random_seed(seed)

```
tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.random_uniform(shape, inval=0, maxval=None, dtype=tf.float32, seed=None, name=None)
```

Operations

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

tf.Variable

```
In [32]: W1 = tf.ones((2,2))

In [33]: W2 = tf.Variable(tf.zeros((2,2)), name="weights")

In [34]: with tf.Session() as sess:
             print(sess.run(W1))
             sess.run(tf.initialize_all_variables())
             print(sess.run(W2))
   ....:
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
```

Note the initialization step `tf.initialize_all_variables()`

- "When you train a model you use variables to **hold and update parameters**. Variables are in-memory buffers containing tensors" - TensorFlow Docs
- **TensorFlow variables must be initialized before they have values! Contrast with constant tensors.**

Updating Variable State

```
In [63]: state = tf.Variable(0, name="counter")

In [64]: new_value = tf.add(state, tf.constant(1))

In [65]: update = tf.assign(state, new_value)

In [66]: with tf.Session() as sess:
   ....:     sess.run(tf.initialize_all_variables())
   ....:     print(sess.run(state))
   ....:     for _ in range(3):
   ....:         sess.run(update)
   ....:         print(sess.run(state))
   ....:
0
1
2
3
```

Roughly `new_value = state + 1`

Roughly `state = new_value`

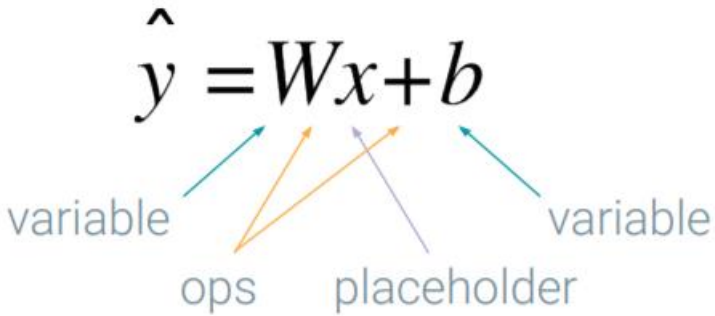
Roughly

```
state = 0
print(state)
for _ in range(3):
  state = state + 1
  print(state)
```

Tensorflow Placeholders

Inputting Data

- Ops are functions on tensors
- tf.constant -> tensors with constant values
- tf.Variable -> tensors params which can be updated
- tf.placeholder -> ‘dummy’ tensor that holds input data.



Inputting Data

- Ops are functions on tensors
- tf.constant -> tensors with constant values (fixed | not trainable)
- tf.Variable -> tensors params which can be updated (trainable)
- tf.placeholder -> 'dummy' tensor that holds input data. Will need to pass actual data in tf.Session() using a *feed_dict*

```
# create a placeholder for a vector of 2 elements, type tf.float32
x = tf.placeholder(dtype=tf.float32, shape=[2], name='p')

y = tf.constant(value=[1, 2], dtype=tf.float32, name='c')
```

feed_dict

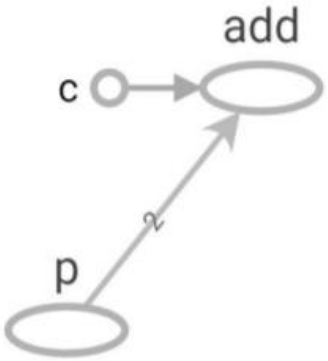
```
# create a placeholder for a vector of 2 elements, type tf.float32
x = tf.placeholder(dtype=tf.float32, shape=[2], name='p')

y = tf.constant(value=[1, 2], dtype=tf.float32, name='c')

z = x + y

with tf.Session() as sess:
    z_val = sess.run(z, feed_dict={x: [3, 4]})
    print(z_val)
```

```
[ 4.  6.]
```



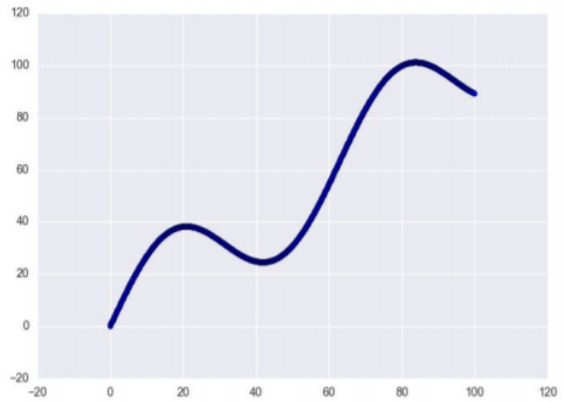
Linear Regression Example

1a) Create Input Data

```
import numpy as np
import seaborn

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

# Plot input data
plt.scatter(X_data, y_data)
```



1b) Define hyper-params

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

# Tensorflow is finicky about shapes, so resize
X_data = np.reshape(X_data, (n_samples,1))
y_data = np.reshape(y_data, (n_samples,1))

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

2) Define placeholders

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

# Tensorflow is finicky about shapes, so resize
X_data = np.reshape(X_data, (n_samples,1))
y_data = np.reshape(y_data, (n_samples,1))

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

3) Define Variables

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1,),
                        initializer=tf.constant_initializer(0.0))
    y_pred = tf.matmul(X, W) + b
    loss = tf.reduce_sum((y - y_pred)**2/n_samples)
```

get_variable
same as creating
a variable

4) Define Loss Function

```python
# Define variables to be learned
with tf.variable_scope("linear-regression"):
  W = tf.get_variable("weights", (1, 1),
                      initializer=tf.random_normal_initializer())
  b = tf.get_variable("bias", (1,),
                      initializer=tf.constant_initializer(0.0))
y_pred = tf.matmul(X, W) + b
loss = tf.reduce_sum((y - y_pred)**2/n_samples)
```

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

5) Define optimizer

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

Let's do a deeper. graphical dive into this operation

6) Define training procedure in tf.Session()

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)
```

```
with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

Let's do a deeper. graphical dive into this operation

6a) Initialize variables

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

Let's do a deeper. graphical dive into this operation

6b) Grad Descent Iteration loop

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

Let's do a deeper. graphical dive into this operation

6c) Inputs to placeholders

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

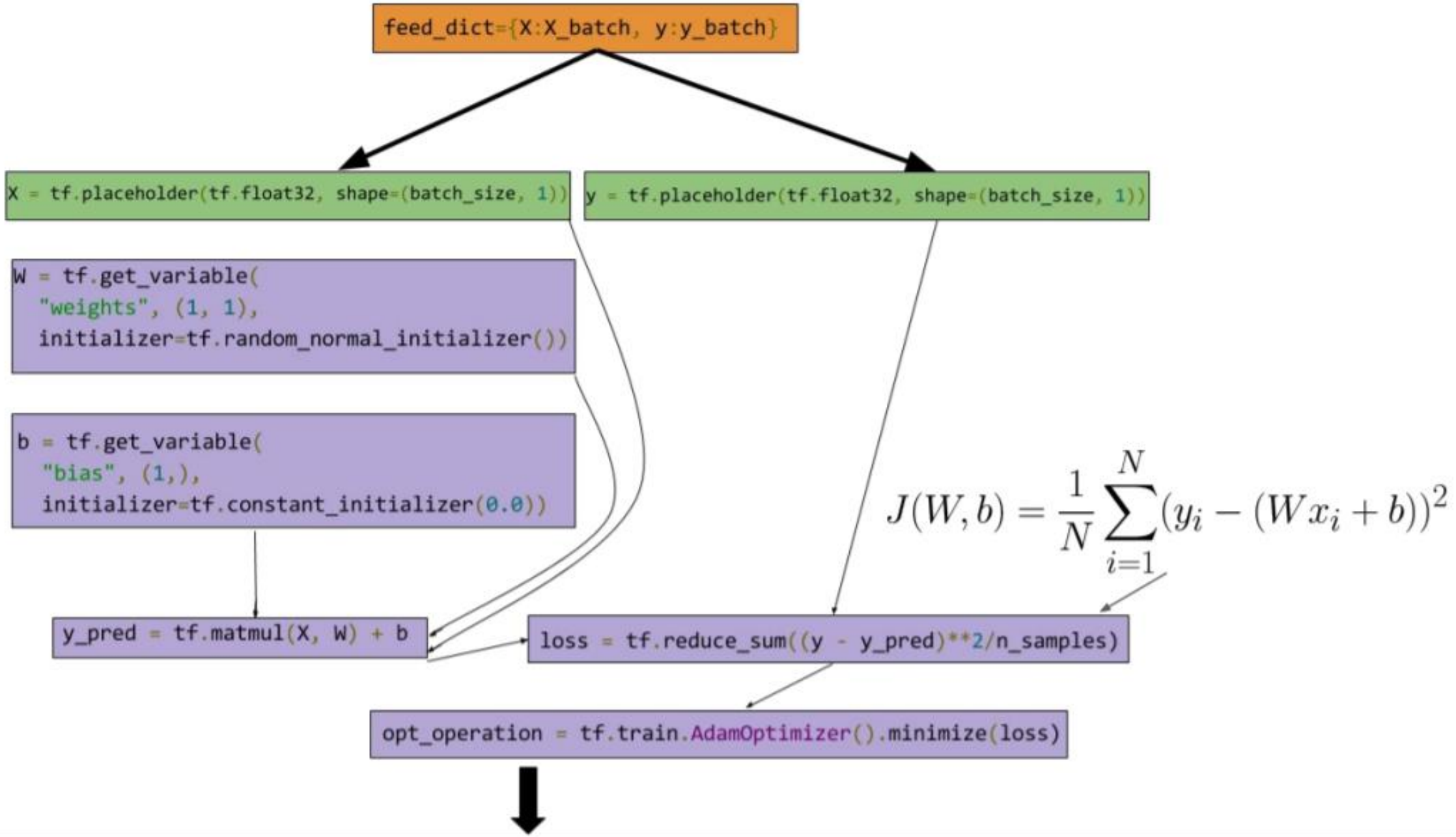
Let's do a deeper. graphical dive into this operation

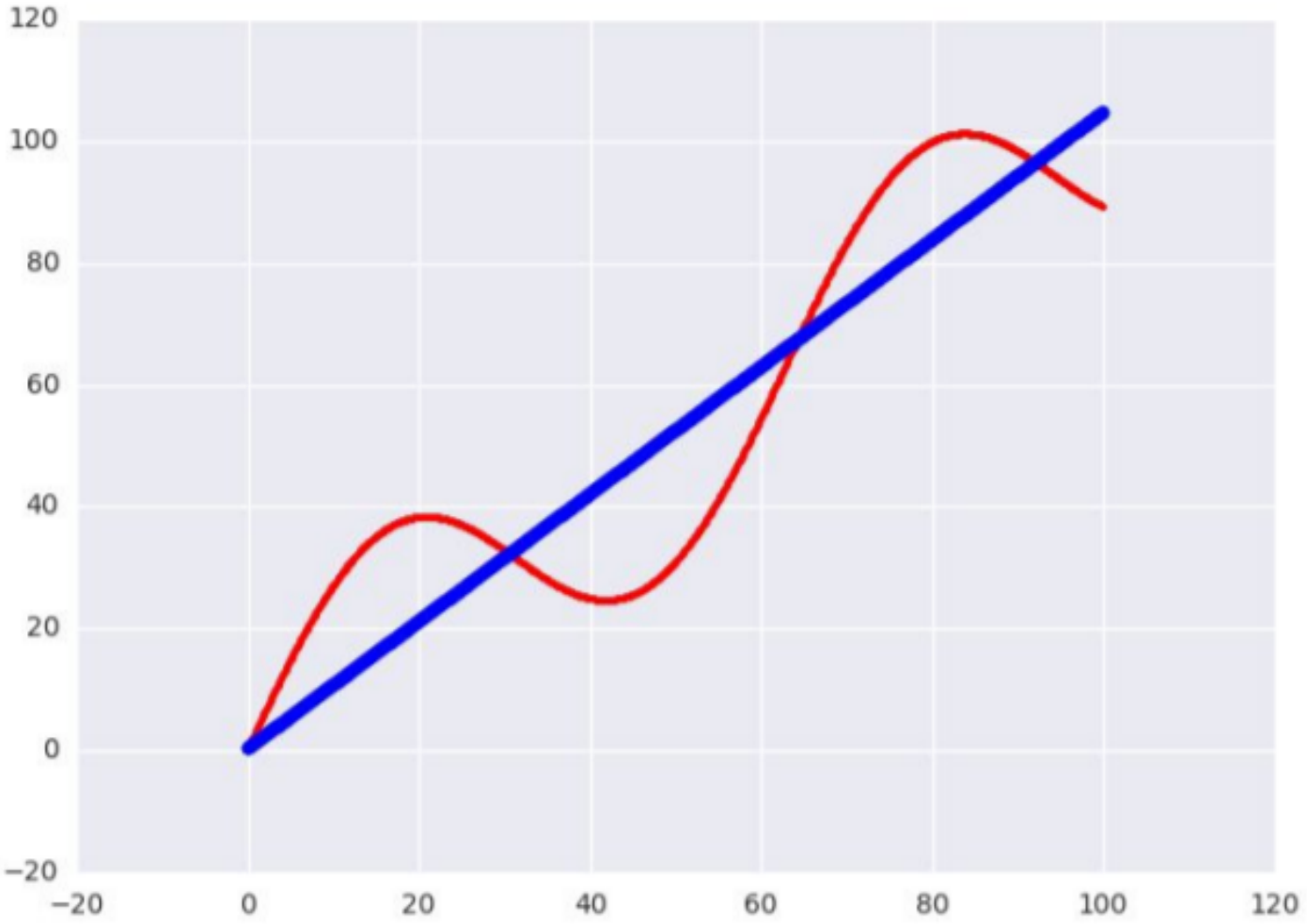
6d) Grad Descent step

```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # 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_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch}
```

Let's do a deeper. graphical dive into this operation


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)



Learned model offers nice fit to data.

Tensorboard

```
x = tf.get_variable(name='x', initializer=tf.random_normal([2]))
y = tf.get_variable(name='y', initializer=tf.random_normal([2]))
z = x+y

writer = tf.summary.FileWriter(logdir='./graphs', graph=tf.get_default_graph())

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(z))
writer.close()
```

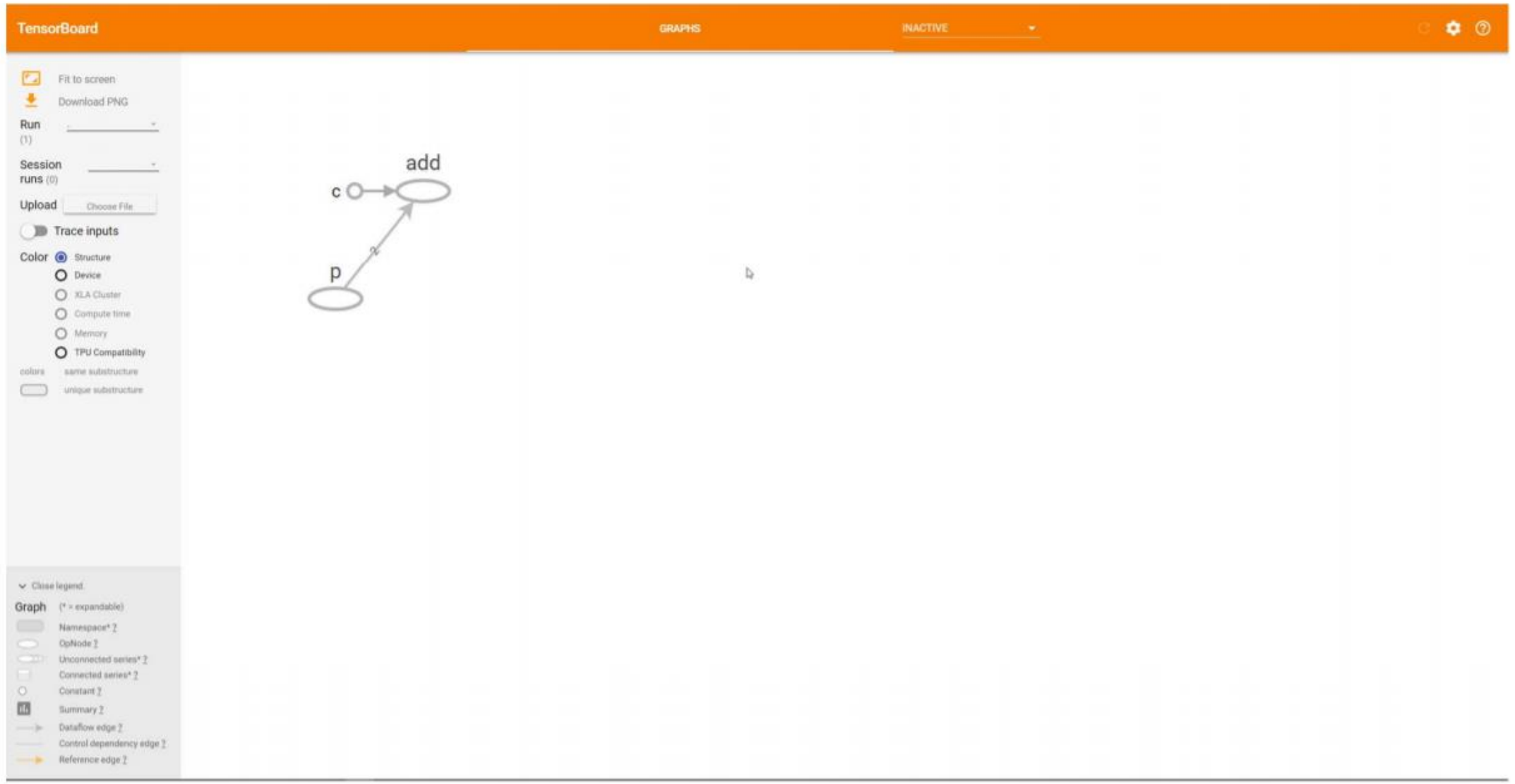
```
[-2.20665431  0.04801518]
```

- **In terminal:**

  $ tensorboard --logdir="./graphs" --port 6006

- **In web browser:**

  http://localhost:6006/


TensorBoard
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
add
c
p
2
Close legend
Graph
(* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?

Resources

- https://www.tensorflow.org/guide
- http://web.stanford.edu/class/cs20si/syllabus.html
- Youtube - https://www.youtube.com/user/hvasslabs
- Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2017.