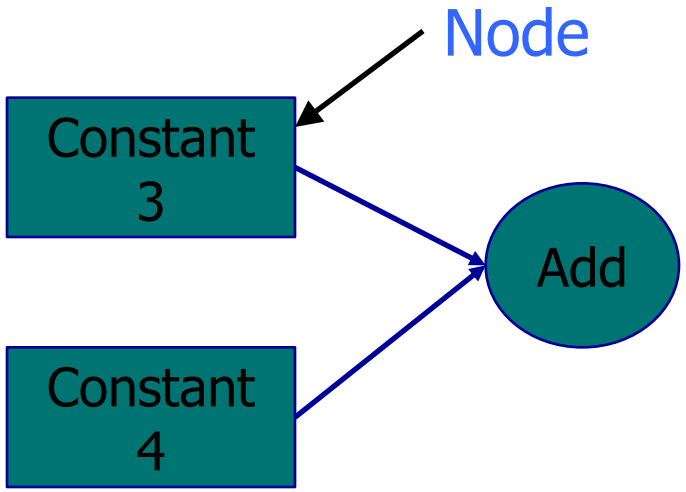
CS 5450

TensorFlow

Goals

- Library for numerical computations (read: ML)
  - Primitives for defining functions on tensors and automatically computing their derivatives
  - Tensor ~ multi-dimensional array of numbers
- Specify computation as a data-flow graph
  - Nodes: operations with any number of tensor inputs and tensor outputs
  - Edges: tensors that flow between operations

Dataflow Programming



All nodes return tensors

TensorFlow does not care how a node computes

define

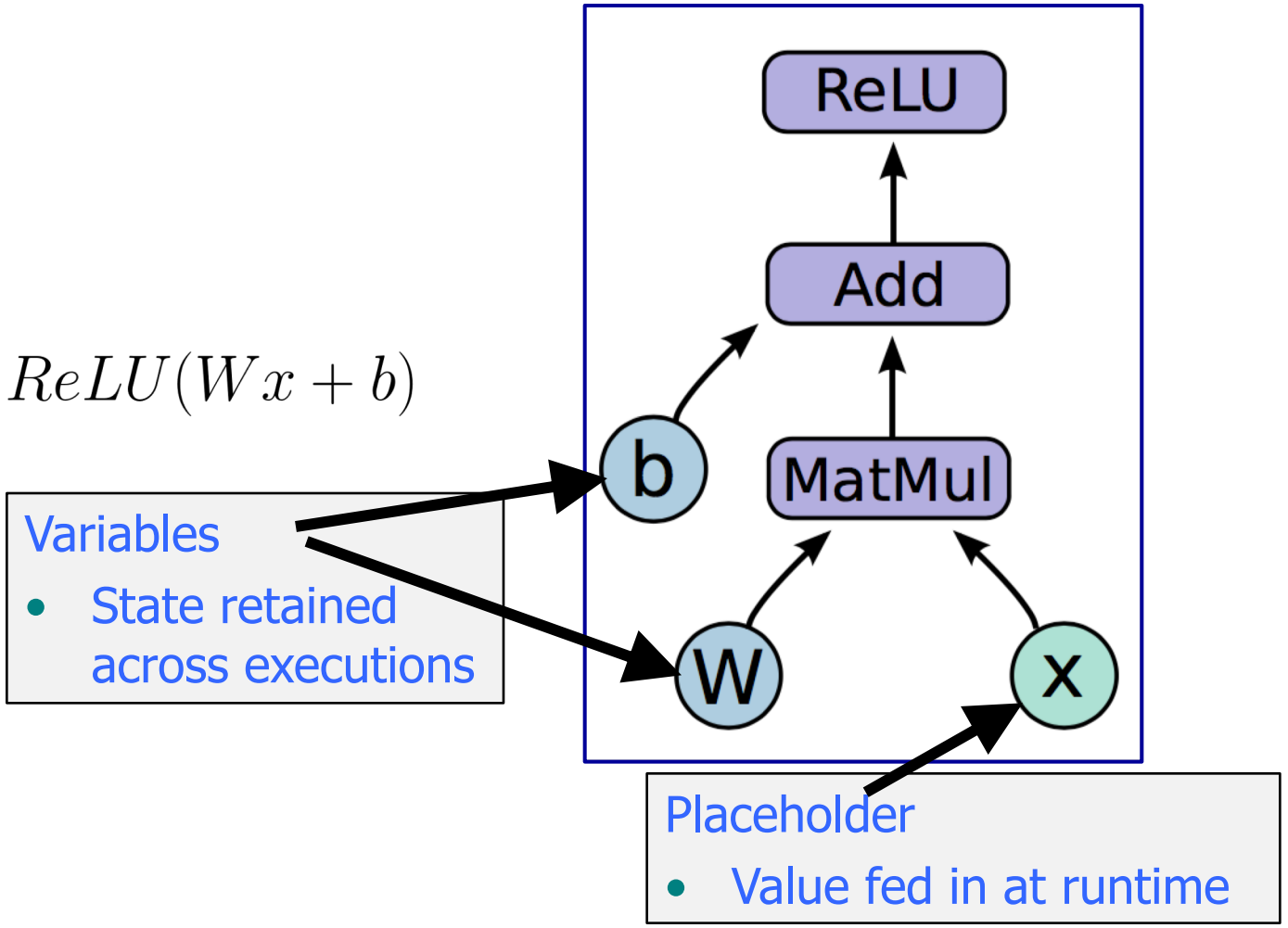
```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0, dtype=tf.float32)
node3 = tf.add(node1,node2)
```

exec

```
tf.Session().run(node3) #returns 7
```

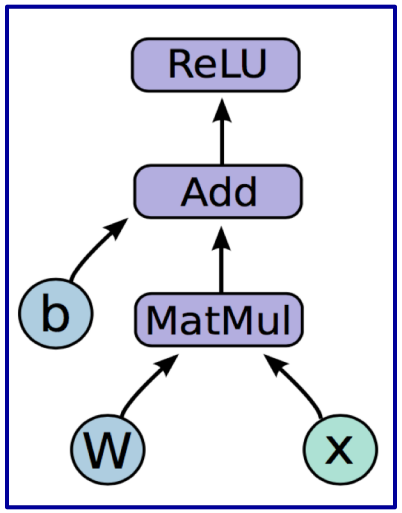
Example

$$h = ReLU(Wx + b)$$

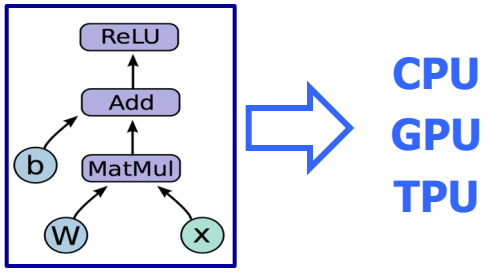


In Code

```
import tensorflow as tf
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))
x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)
```



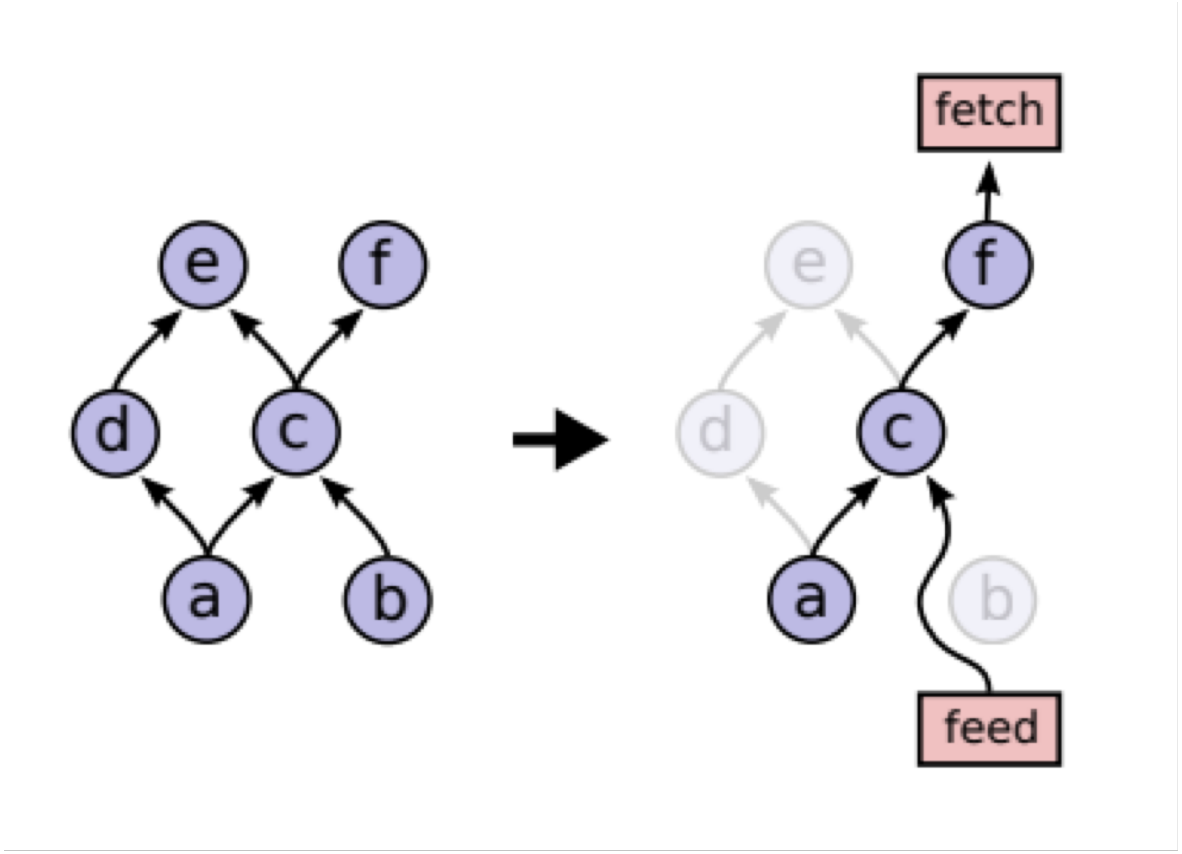
Run Graph in Execution Environment




```
import numpy as np
import tensorflow as tf
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))
x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
   sess.run(tf.initialize_all_variables())
   sess.run(h, {x: np.random.random(100, 784)})
```

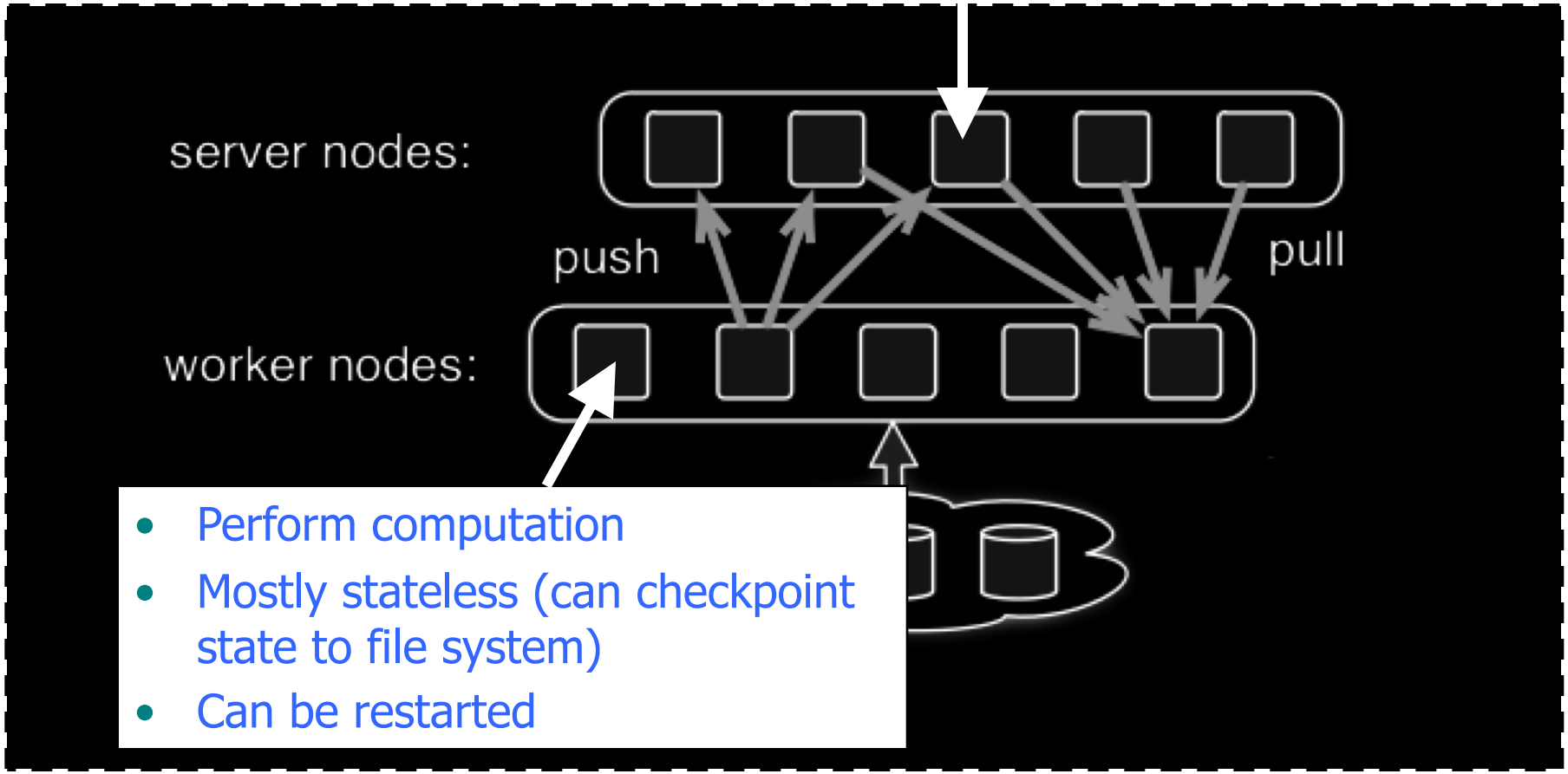
Graph executes here

Partial Execution of Subgraphs



Parameter Server

- Hold mutable state
- Apply updates
- Maintain availability



- Perform computation
- Mostly stateless (can checkpoint state to file system)
- Can be restarted

Parameter Server Example

```
with tf.device("/jobs:ps/task:0/cpu:0"):
  W = tf.Variable(...)
  b = tf.Variable(...)
inputs = tf.split(0,num_workers,input)
outputs = []
for i in range (num_workers):
  with tf.device("/job:worker/task:%d/gpu:0" % i):
    outputs.append(tf.matmul(input[i],W) + b)
```

Computing Gradients

TensorFlow nodes have attached gradient operations

```
tf.train.GradientDescentOptimizer(…).minimize(…)
```

adds optimization operation to computation graph

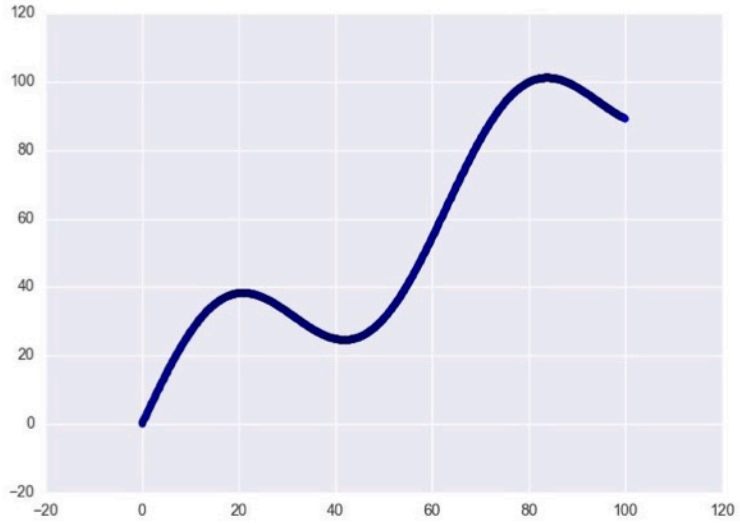
Gradients with respect to parameters are computed automatically via backpropagation

Training an ML Model

```
prediction = tf.nn.softmax(...)
label = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(label *
tf.log(prediction), reduction_indices=[1]))

train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.Session()
sess.run(tf.initialize_all_variables())

for i in range(1000):
    batch_x, batch_label = data.next_batch()
    sess.run(train_step, feed_dict={x: batch_x, label: batch_label})
```

Ex: Linear Regression in TensorFlow (1)

```
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)
```



credit: Bharath Ramsundar, Stanford CS224d tutorial

Ex: Linear Regression in TensorFlow (2)

```
# 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))
```

credit: Bharath Ramsundar, Stanford CS224d tutorial

Ex: Linear Regression in TensorFlow (3)

```
# 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)
```

Note `reuse=False` *so these tensors are created anew*

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

credit: Bharath Ramsundar, Stanford CS224d tutorial

Ex: Linear Regression in TensorFlow (4)

```
# Sample code to run one step of gradient descent
In [136]: opt = tf.train.AdamOptimizer()

In [137]: opt_operation = opt.minimize(loss)

In [138]: with tf.Session() as sess:
   .....:     sess.run(tf.initialize_all_variables())
   .....:     sess.run([opt_operation], feed_dict={X: X_data, y: y_data})
   .....:
```

Note TensorFlow scope is not python scope! Python variable `loss` is still visible.

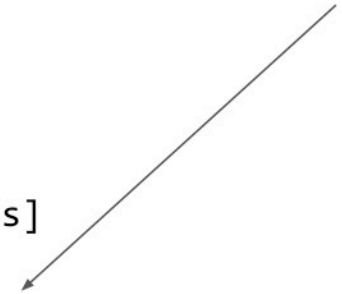
But how does this actually work under the hood? Will return to TensorFlow computation graphs and explain.

credit: Bharath Ramsundar, Stanford CS224d tutorial

Ex: Linear Regression in TensorFlow (4)

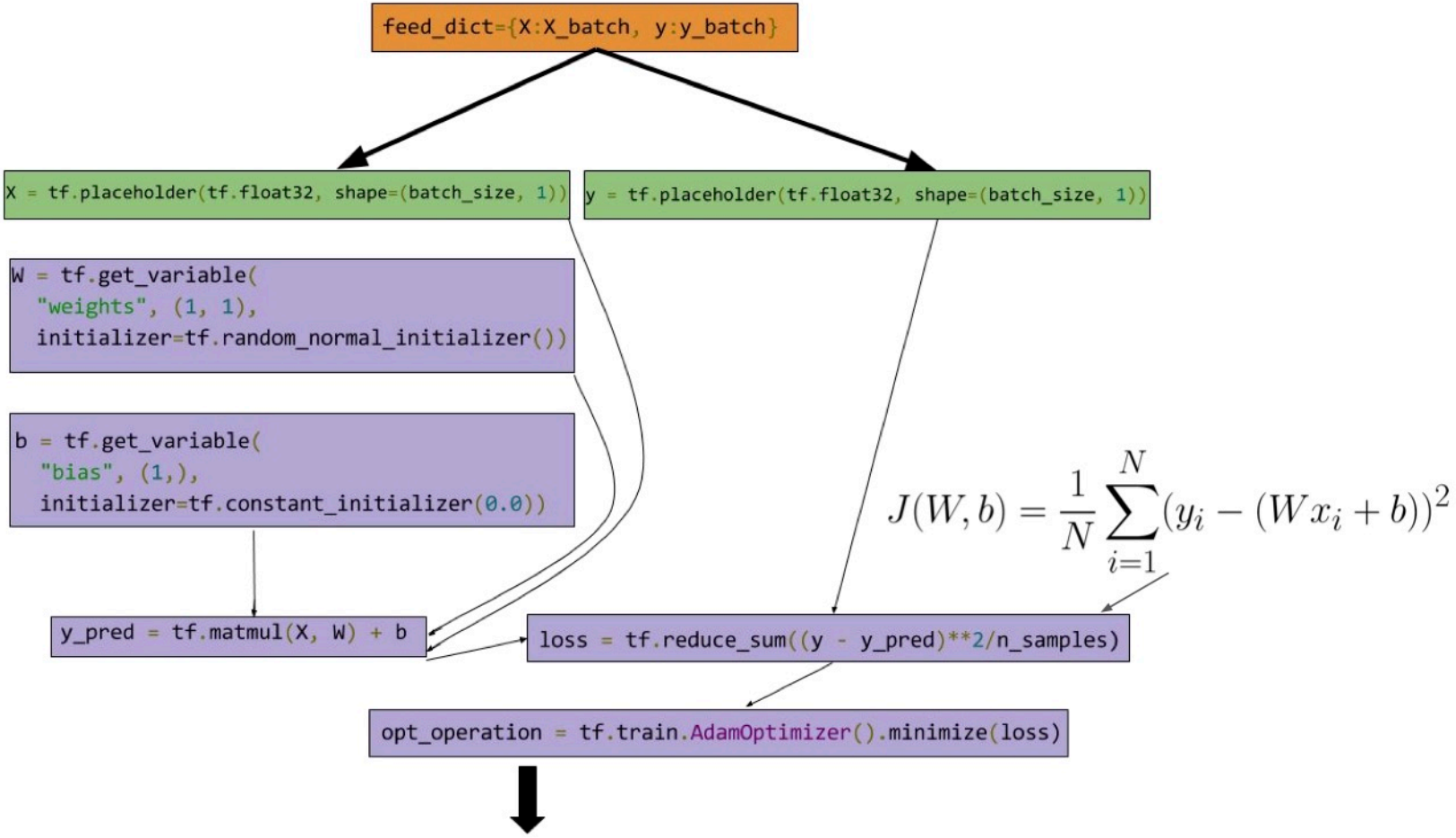
```
# Sample code to run full gradient descent:
# Define optimizer operation
opt_operation = tf.train.AdamOptimizer().minimize(loss)

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



credit: Bharath Ramsundar, Stanford CS224d tutorial

Ex: Linear Regression in TensorFlow (5)



credit: Bharath Ramsundar, Stanford CS224d tutorial