TensorFlow:
A Framework for Scalable Machine Learning

Yuefeng Zhou, Google Brain

You probably want to know...

What is TensorFlow?

Why did we create TensorFlow?

How does Tensorflow Work?

Example: Linear Regression

Example: Convolutional Neural Network

Distributed TensorFlow

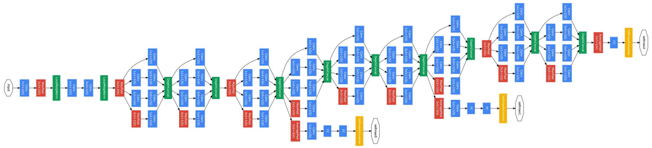
Fast, flexible, and scalable open-source machine learning library

One system for research and production

Runs on CPU, GPU, TPU, and Mobile

Apache 2.0 license

TensorFlow Handles Complexity

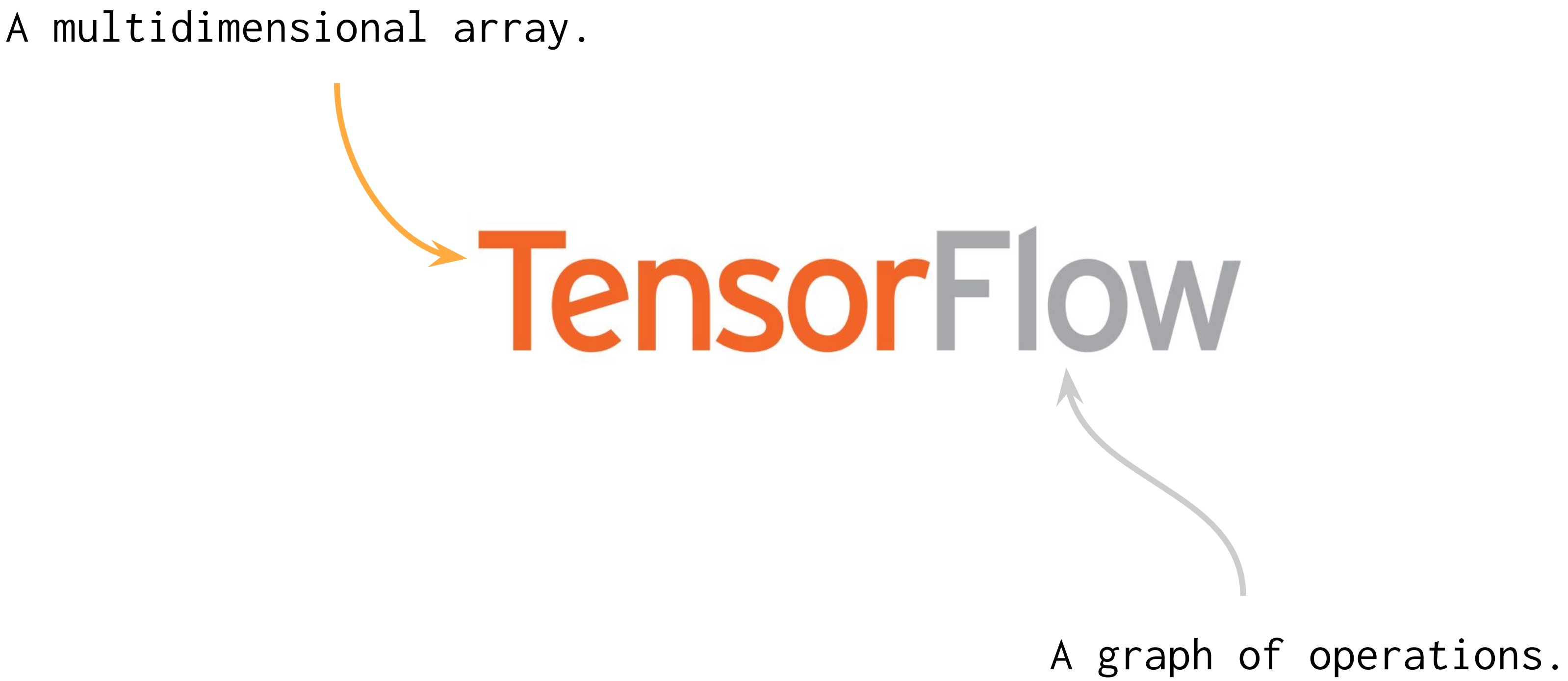


Modeling complexity

Distributed System

Heterogenous System

Under the Hood



A multidimensional array.

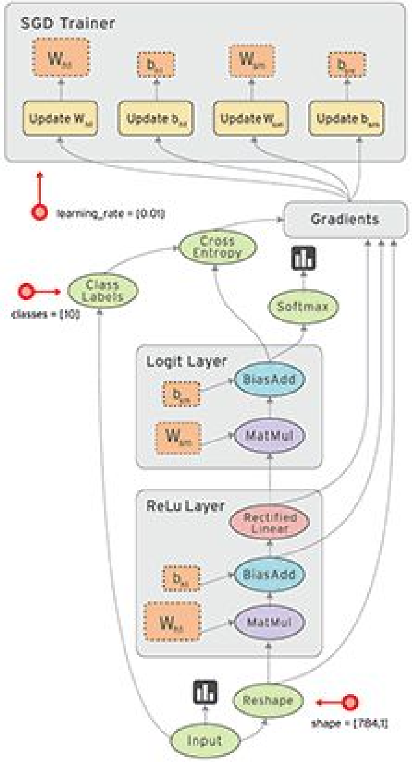
TensorFlow

A graph of operations.

The TensorFlow Graph

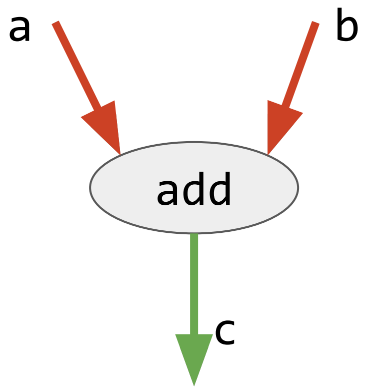
Computation is defined as a graph

- Graph is defined in high-level language (Python)
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU, TPU)
- Nodes represent computations and state
- Data (tensors) flow along edges

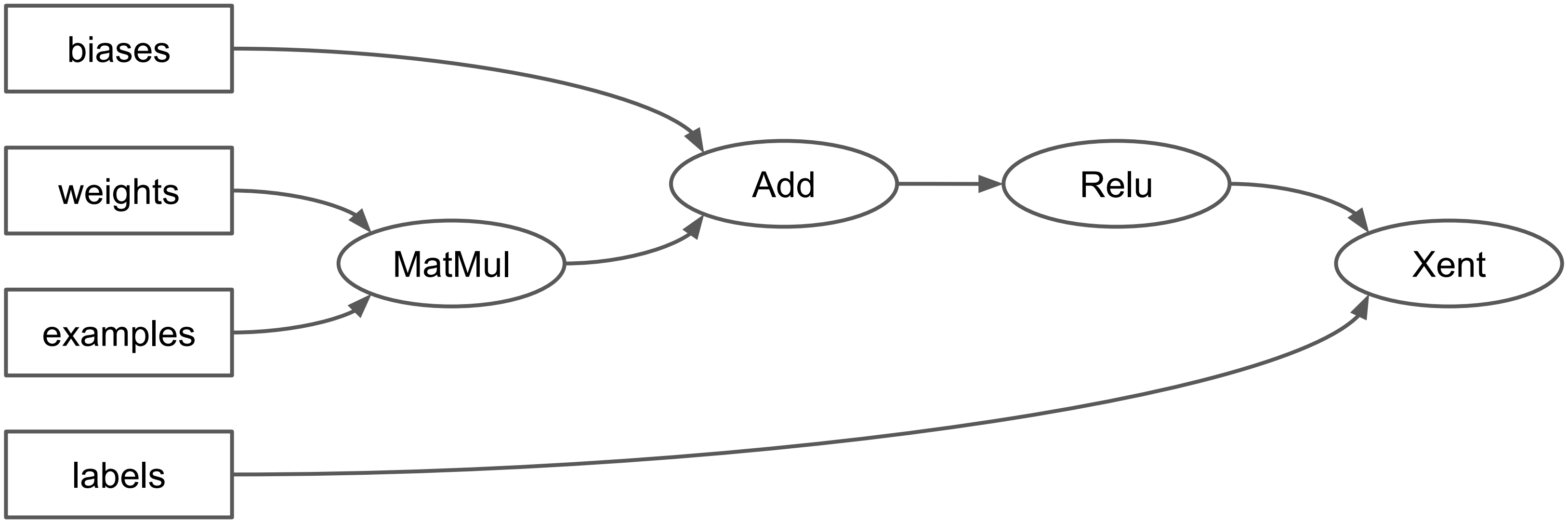


Build a graph; then run it.

```
...
c = tf.add(a, b)


...

session = tf.Session()
value_of_c = session.run(c, {a=1, b=2})
```

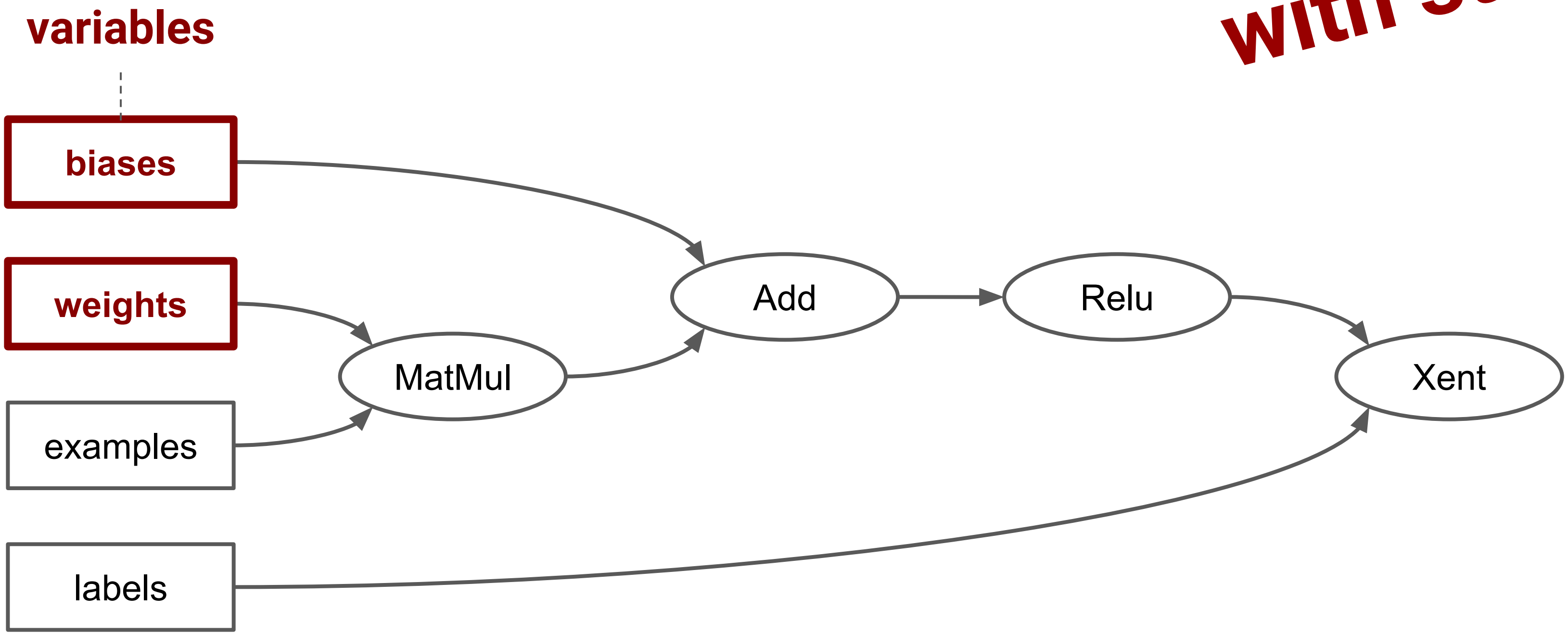


Any Computation is a TensorFlow Graph



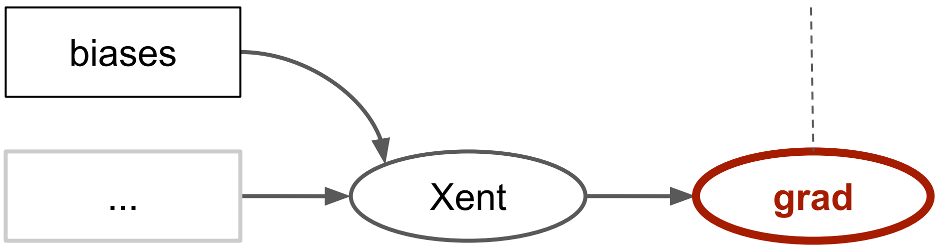
Any Computation is a TensorFlow Graph

with state



Automatic Differentiation

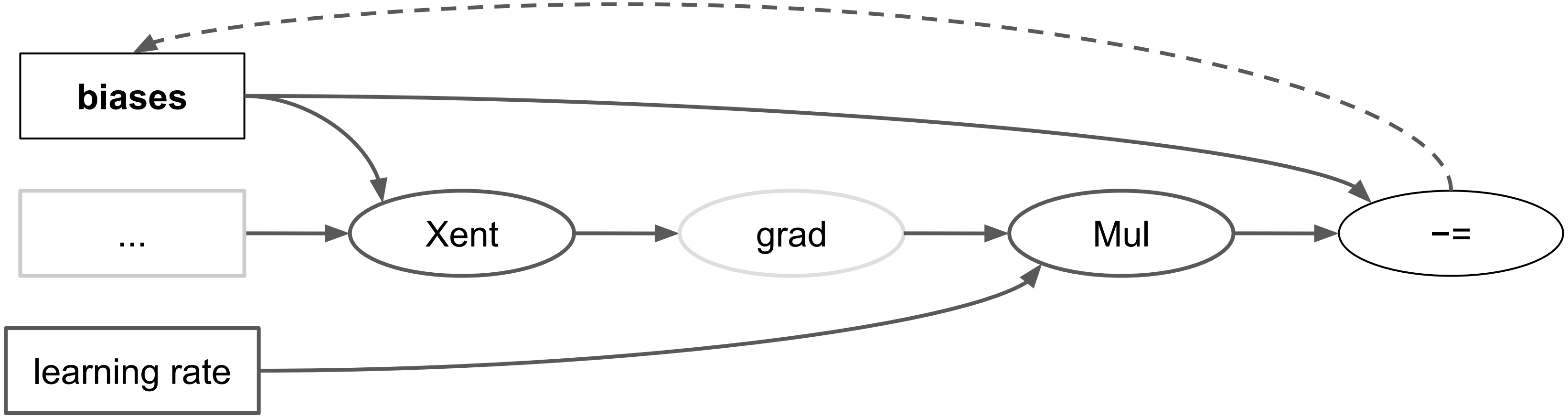
Automatically add ops which compute gradients for variables



Any Computation is a TensorFlow Graph

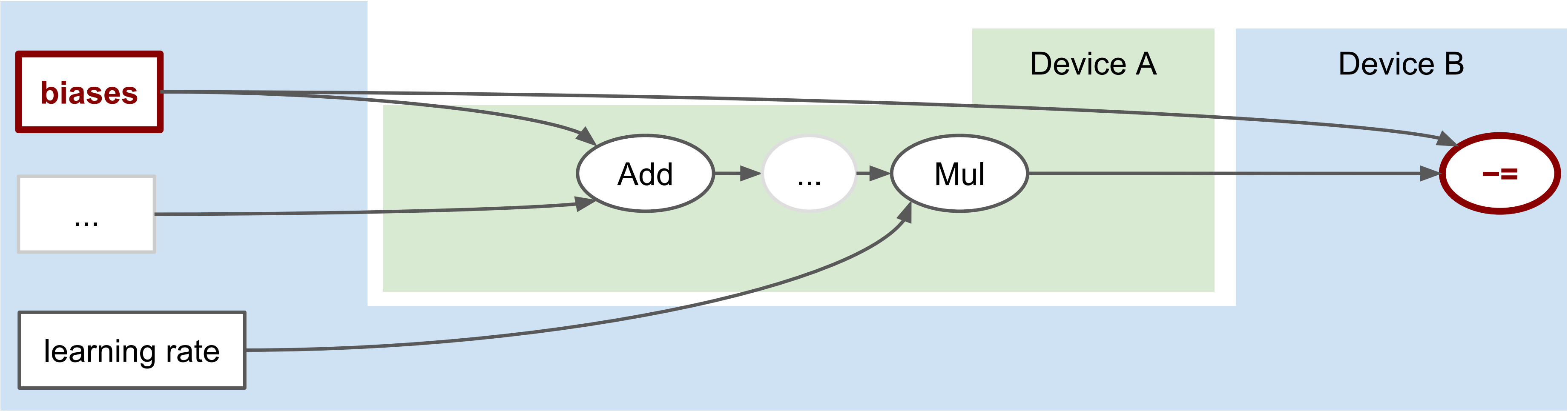
Simple gradient descent:

with state



Any Computation is a TensorFlow Graph

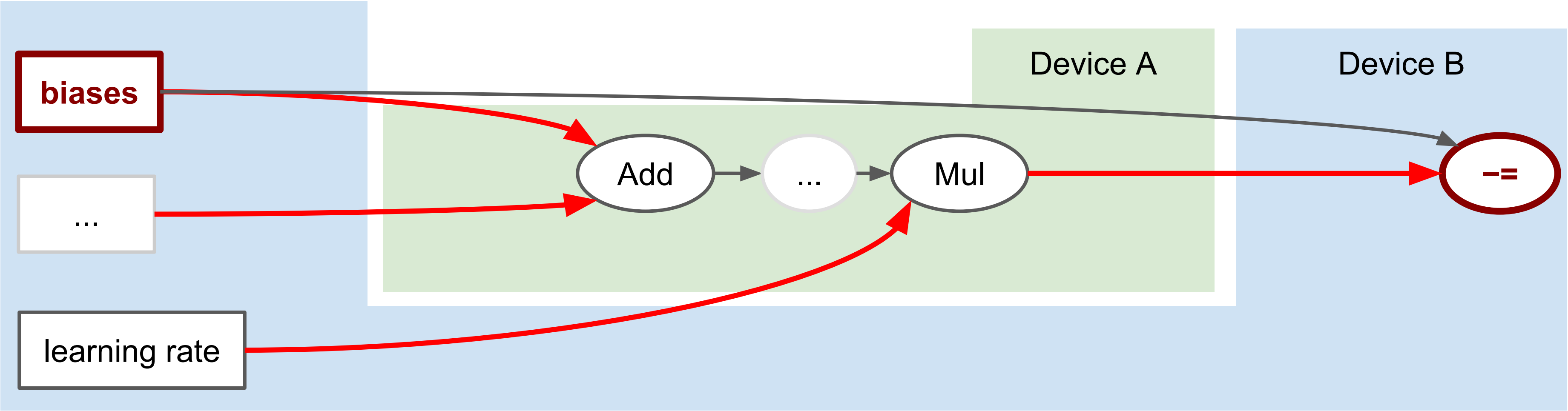
distributed



Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

Send and Receive Nodes

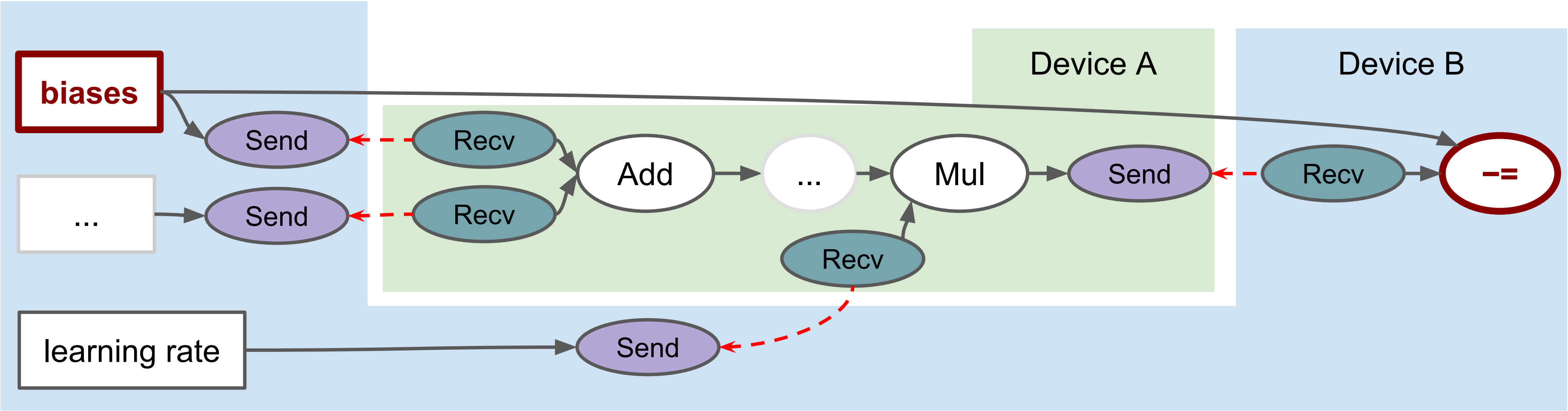
distributed



Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

Send and Receive Nodes

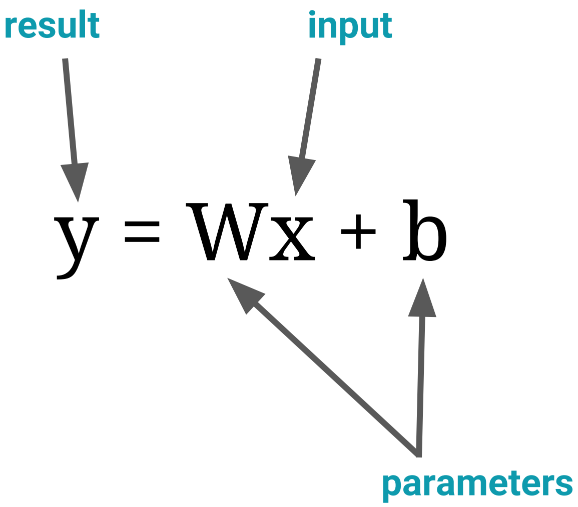
distributed



Devices: Processes, Machines, CPUs, GPUs, TPUs, etc

Linear Regression

Linear Regression

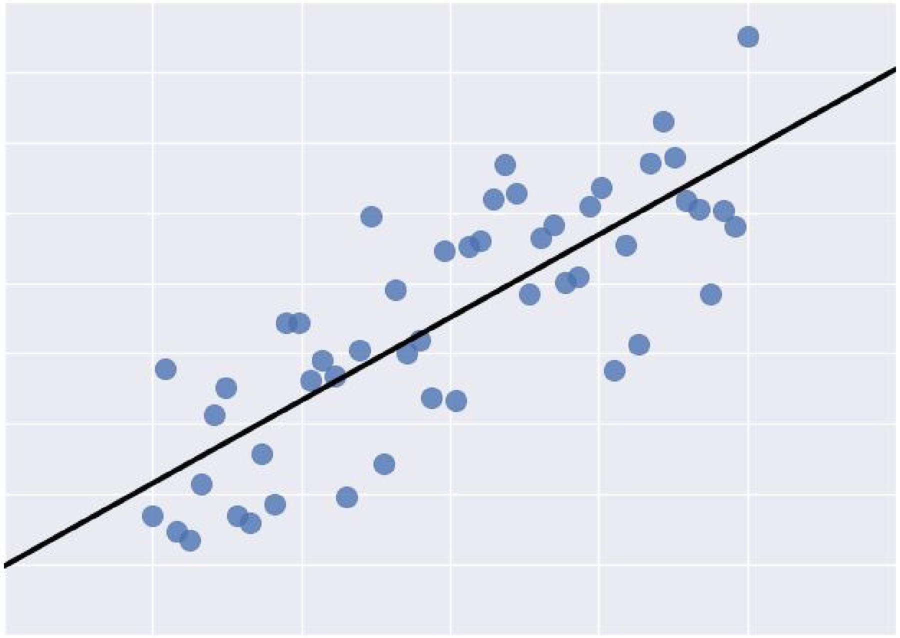


result

input

$$y = Wx + b$$

parameters



What are we trying to do?

Mystery equation: `y = 0.1 * x + 0.3 + noise`

Model: `y = W * x + b`

Objective: Given enough (`x`, `y`) value samples, figure out the value of `W` and `b`.

$y = Wx + b$ in TensorFlow

```
import tensorflow as tf
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)
```

y = Wx + b in TensorFlow

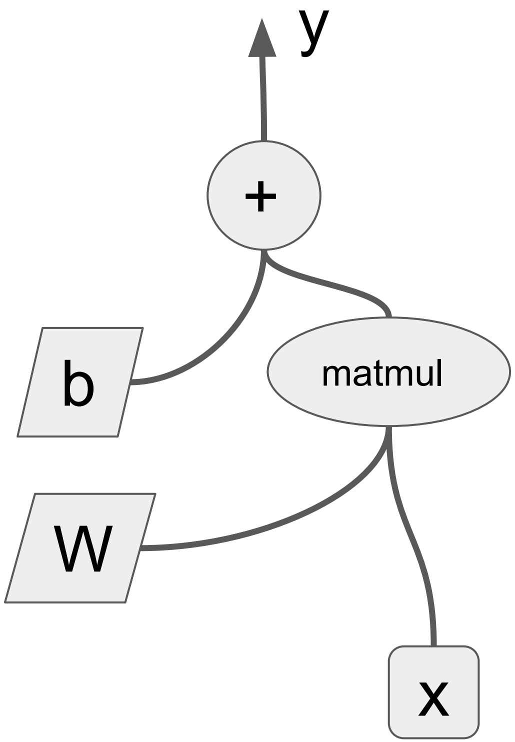
```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[], name=”W”)
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[], name=”W”)

b = tf.get_variable(shape=[], name=”b”)
```

y = Wx + b in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[None],
                   dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[], name=”W”)

b = tf.get_variable(shape=[], name=”b”)

y = W * x + b
```



Variables Must be Initialized

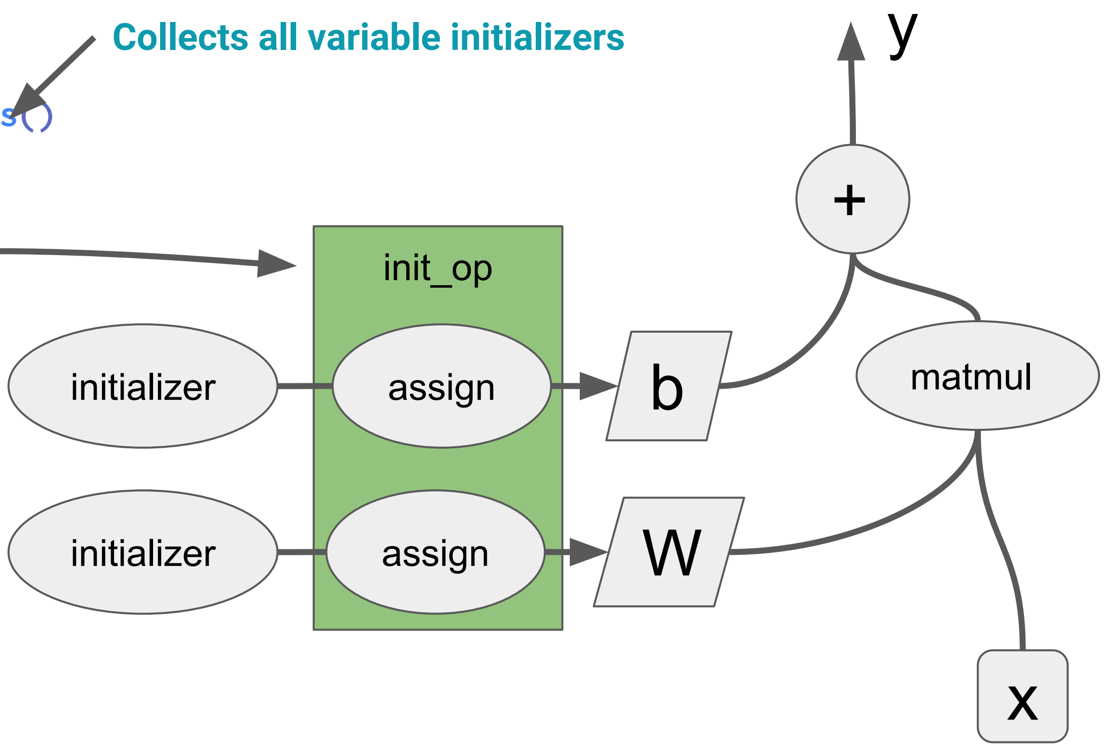
Collects all variable initializers

```
init_op = tf.initialize_all_variables()
```

Makes an execution environment

```
sess = tf.Session()

sess.run(init_op)
```

Actually initialize the variables

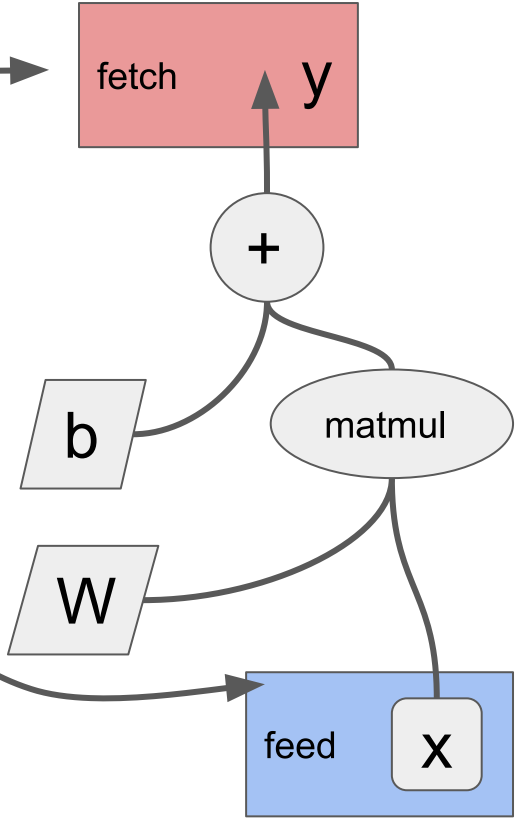


Running the Computation

```
x_in = [3]

sess.run(y, feed_dict={x: x_in})
```

Only what's used to compute a fetch will be evaluated

All Tensors can be fed, but all placeholders must be fed



Putting it all together

```
import tensorflow as tf
x = tf.placeholder(shape=[None],
                        dtype=tf.float32,
                        name='x')
W = tf.get_variable(shape=[], name='W')
b = tf.get_variable(shape=[], name='b')
y = W * x + b
```

Build the graph

```
with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())
  print(sess.run(y, feed_dict={x: x_in}))
```

Prepare execution environment

Initialize variables

Run the computation (usually often)

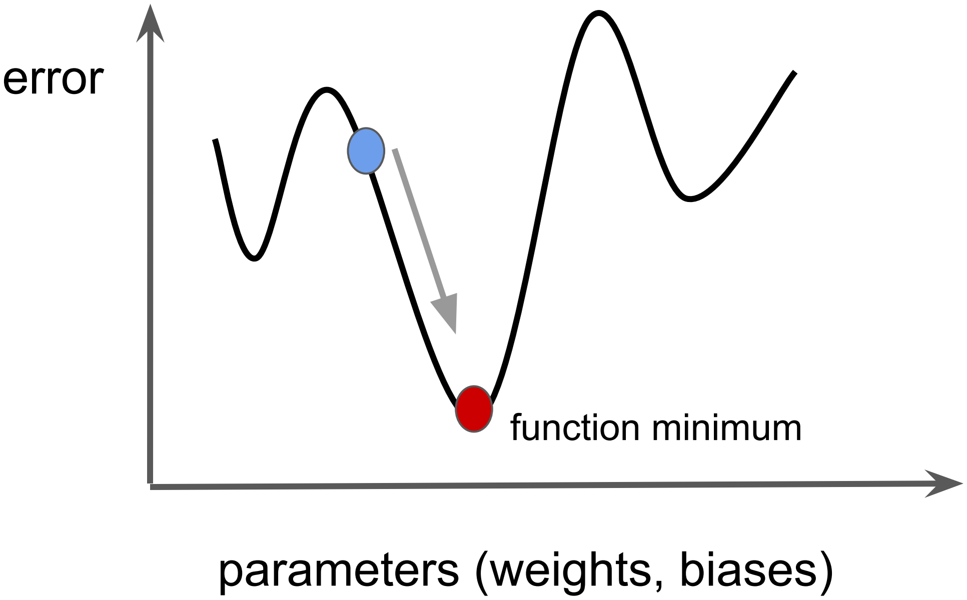
Define a Loss

Given y, y_train compute a loss, for instance:

$$L = (y - y_{train})^2$$

```
# create an operation that calculates loss.
loss = tf.reduce_mean(tf.square(y - y_train))
```

Minimize loss: optimizers



```
tf.train.AdadeltaOptimizer

tf.train.AdagradOptimizer

tf.train.AdagradDAOptimizer

tf.train.AdamOptimizer

…
```

Train

Feed (x, y_{label}) pairs and adjust W and b to decrease the loss.

$$W \leftarrow W - \eta \, (dL/dW)$$

$$b \leftarrow b - \eta \, (dL/db)$$

TensorFlow computes gradients automatically

```
# Create an optimizer
optimizer = tf.train.GradientDescentOptimizer(0.5)
# Create an operation that minimizes loss.
train = optimizer.minimize(loss)
```

Learning rate

Putting it all together

```
loss = tf.reduce_mean(tf.square(y - y_train))

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

with tf.Session() as sess:

  sess.run(tf.initialize_all_variables())

  for i in range(1000):

    sess.run(train, feed_dict={x: x_in, y_label: y_in})
```

Define a loss

Create an optimizer

Op to minimize the loss

Initialize variables

Iteratively run the training op

Putting it all together

```
import tensorflow as tf
x = tf.placeholder(shape=[None], dtype=tf.float32, name='x')
W = tf.get_variable(shape=[], name='W')
b = tf.get_variable(shape=[], name='b')
y = W * x + b

loss = tf.reduce_mean(tf.square(y - y_train))

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

with tf.Session() as sess:

  sess.run(tf.initialize_all_variables())

  for i in range(1000):

    sess.run(train, feed_dict={x: x_in, y_label: y_in})
```

Build the graph

Define a loss

Create an optimizer

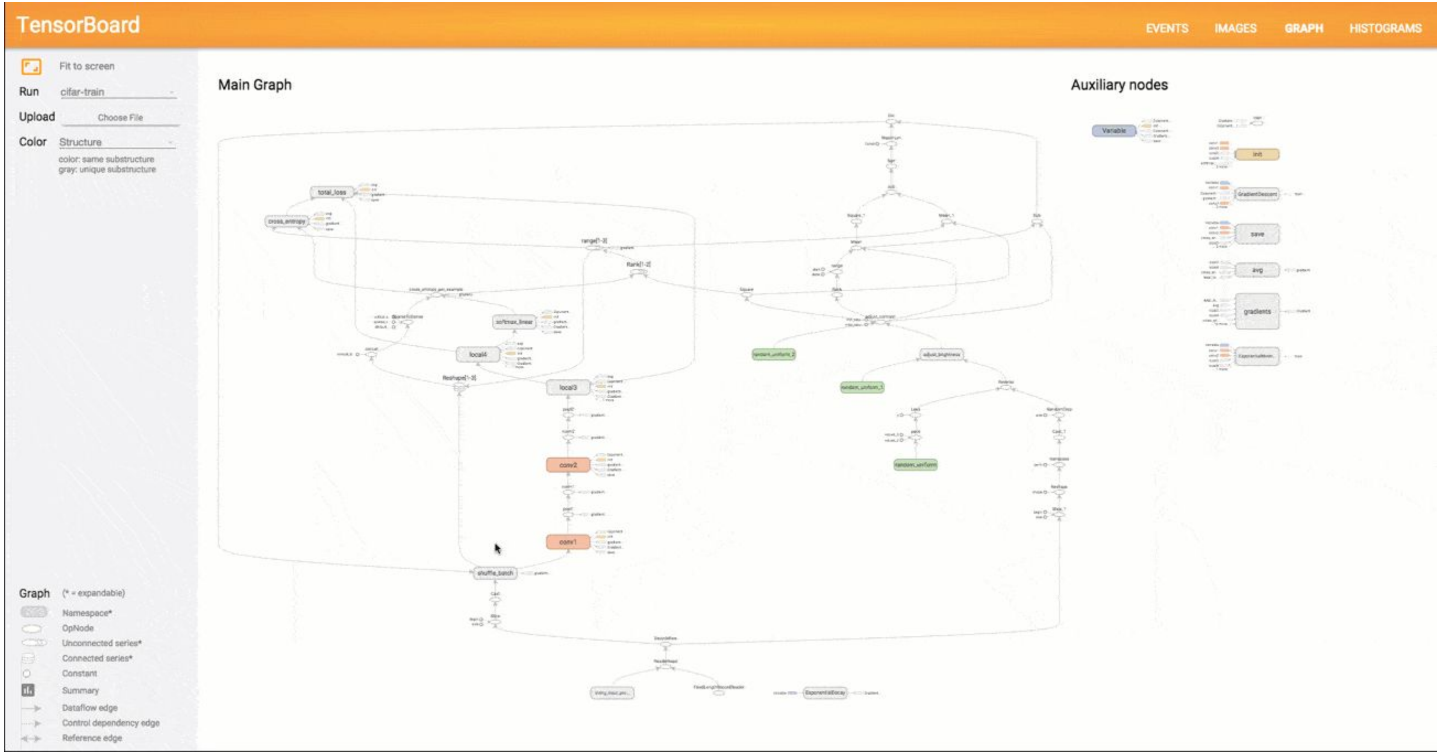
Op to minimize the loss

Prepare environment

Initialize variables

Iteratively run the training op

TensorBoard



Run cifar-train

Upload

Color Structure

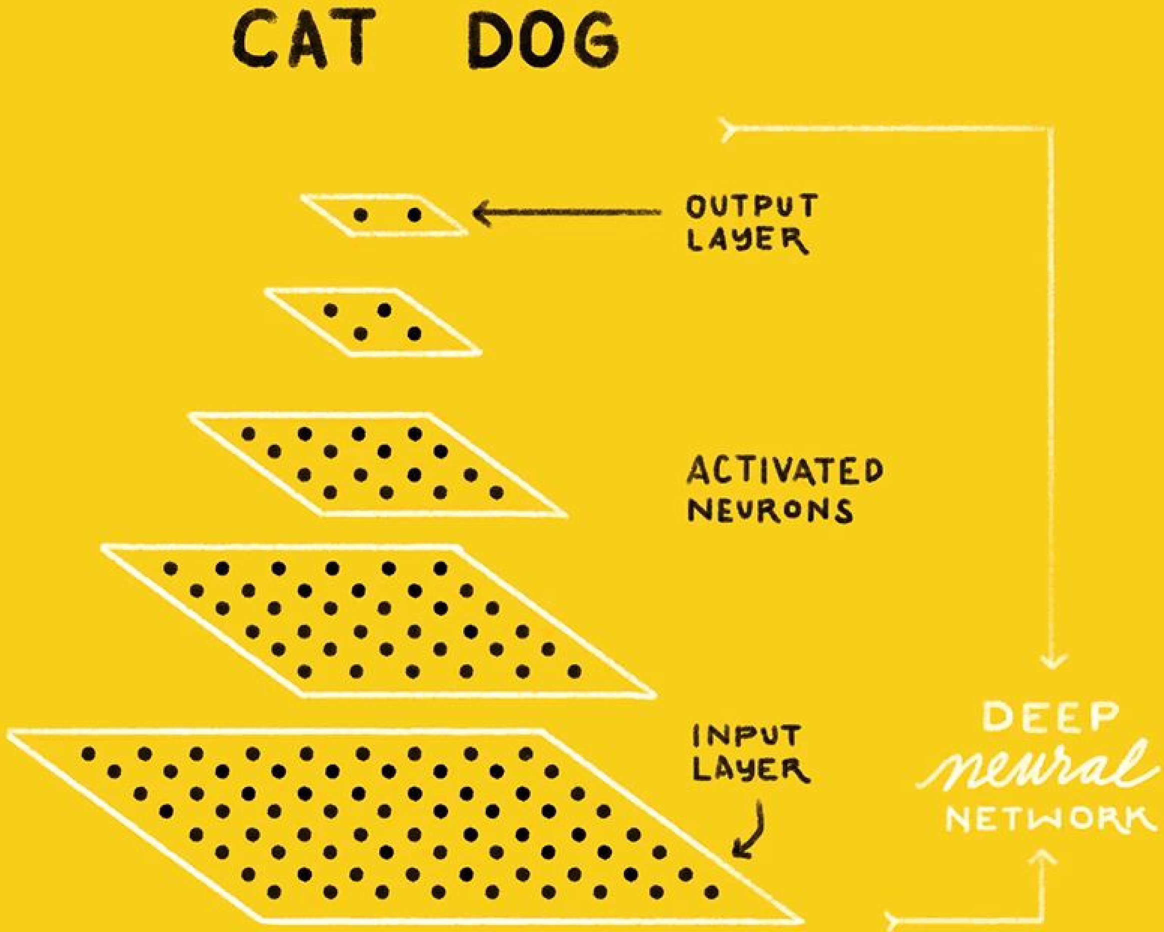
color: same substructure
gray: unique substructure

Graph (* = expandable)

Namespace*
OpNode
Unconnected series*
Connected series*
Constant
Summary
Dataflow edge
Control dependency edge
Reference edge

Deep Convolutional Neural Network

IS THIS A CAT or DOG?



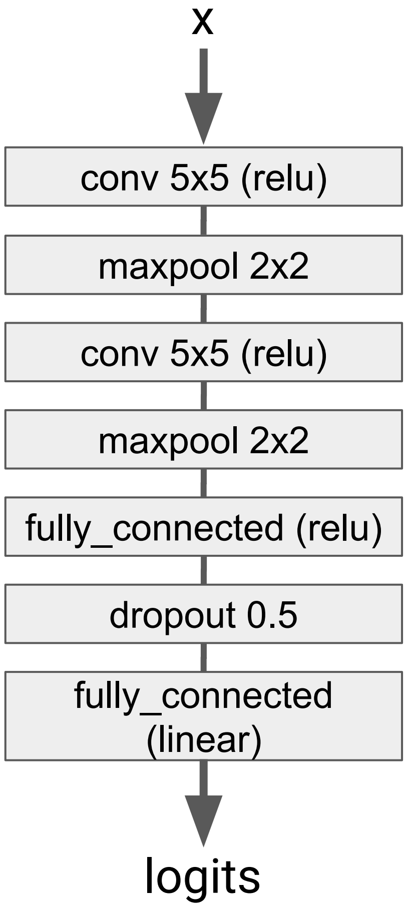
Remember linear regression?

```
import tensorflow as tf
x = tf.placeholder(shape=[None],
                        dtype=tf.float32,
                        name='x')
W = tf.get_variable(shape=[], name='W')
b = tf.get_variable(shape=[], name='b')
y = W * x + b

loss = tf.reduce_mean(tf.square(y - y_label))

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

...
```

Build the graph

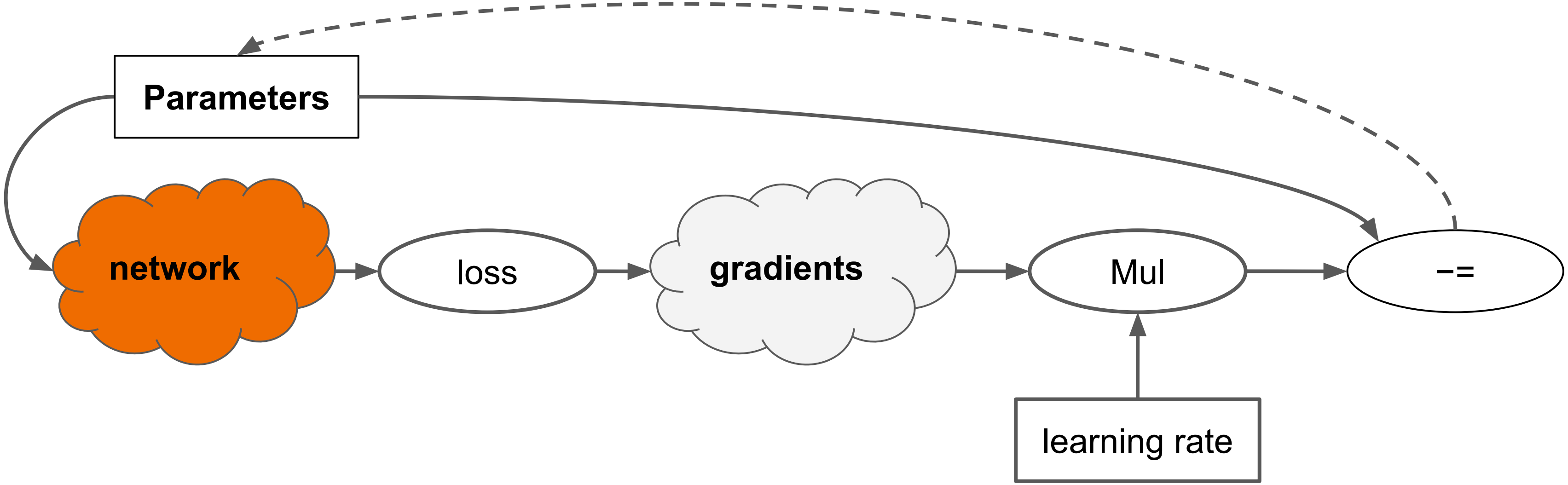
Convolutional DNN

```
x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)

x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)

x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)

x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)

x = tf.contrib.layers.fully_connected(x, activation_fn=tf.nn.relu)

x = tf.contrib.layers.dropout(x, 0.5)

logits = tf.config.layers.linear(x)
```



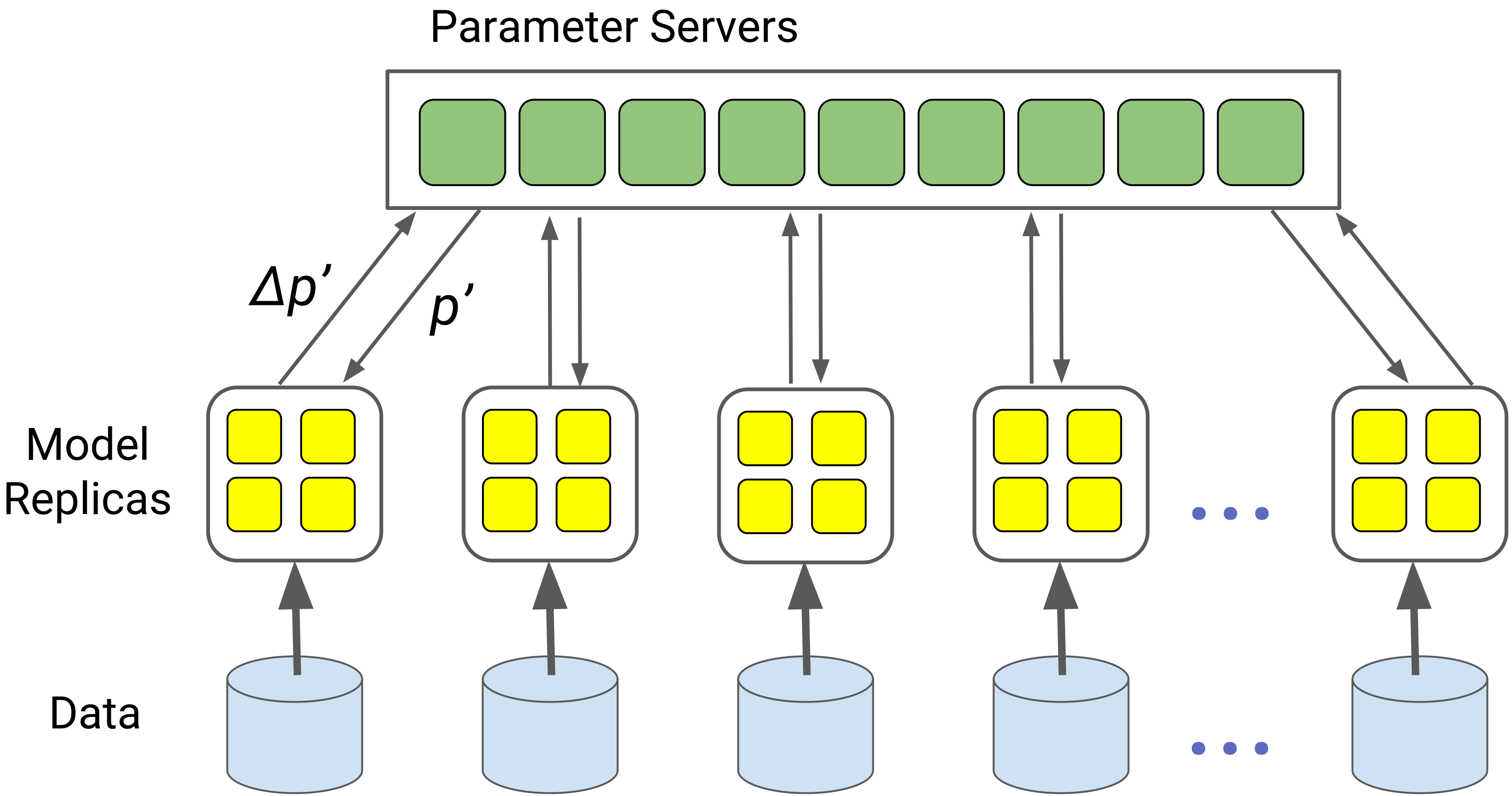
https://github.com/martinwicke/tensorflow-tutorial/blob/master/2_mnist.ipynb

Defining Complex Networks



Distributed TensorFlow

Data Parallelism (Between-Graph Replication)



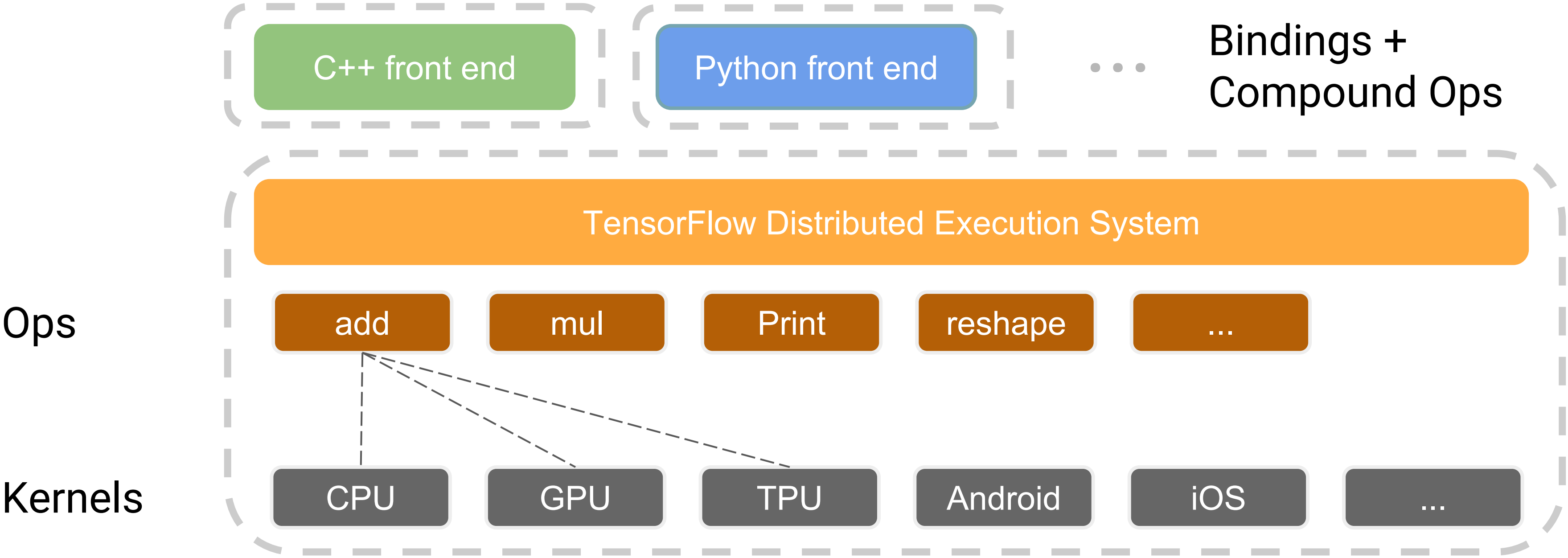
Describe a cluster: ClusterSpec

```
tf.train.ClusterSpec({
    "worker": [
        "worker0.example.com:2222",
        "worker1.example.com:2222",
        "worker2.example.com:2222"
    ],
    "ps": [
        "ps0.example.com:2222",
        "ps1.example.com:2222"
    ]})
```

Share the graph across devices

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.Variable(...)
  biases_1 = tf.Variable(...)

with tf.device("/job:ps/task:1"):
  weights_2 = tf.Variable(...)
  biases_2 = tf.Variable(...)

with tf.device("/job:worker/task:7"):
  input, labels = ...
  layer_1 = tf.nn.relu(tf.matmul(input, weights_1) + biases_1)
  logits = tf.nn.relu(tf.matmul(layer_1, weights_2) + biases_2)
  train_op = ...

with tf.Session("grpc://worker7.example.com:2222") as sess:
  for _ in range(10000):
    sess.run(train_op)
```

Architecture



Tutorials

Tutorials on tensorflow.org

Image recognition: https://www.tensorflow.org/tutorials/image_recognition

Word embeddings: https://www.tensorflow.org/tutorials/word2vec

Language Modeling: https://www.tensorflow.org/tutorials/recurrent

Translation: https://www.tensorflow.org/tutorials/seq2seq

Udacity Course: https://www.udacity.com/course/deep-learning--ud730

Q & A

Google Developer Day