Deep Learning

Lecture 11

Tensorflow

TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems

(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng
Google Research*

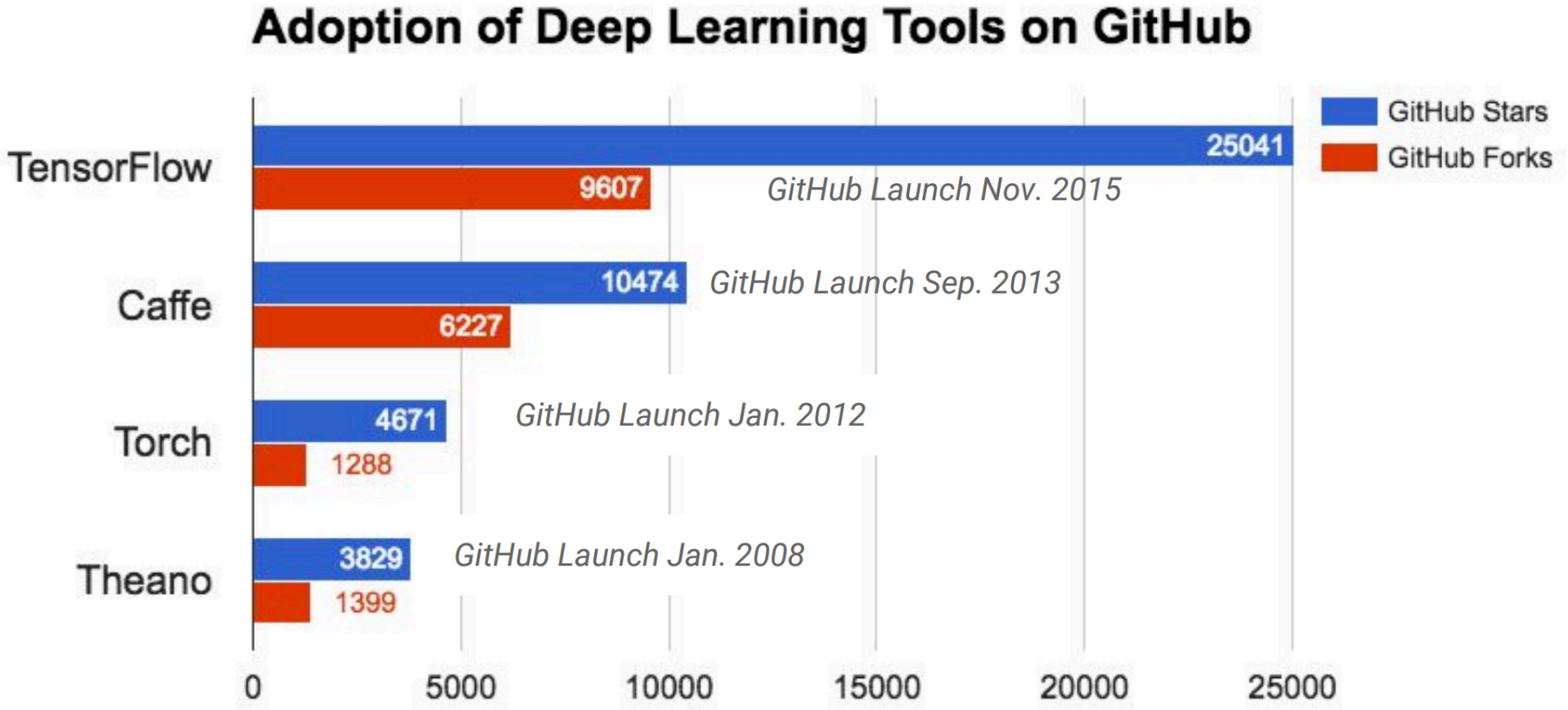
Abstract

TensorFlow [1] is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. A computation expressed using TensorFlow can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms, including training and inference algorithms for deep neural network models, and it has been used for conducting research and for deploying machine learning systems into production across more than a dozen areas of computer science and other fields, including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and computational drug discovery. This paper describes the TensorFlow interface and an implementation of that interface that we have built at Google. The TensorFlow API and a reference

sequence prediction [47], move selection for Go [34], pedestrian detection [2], reinforcement learning [38], and other areas [17, 5]. In addition, often in close collaboration with the Google Brain team, more than 50 teams at Google and other Alphabet companies have deployed deep neural networks using DistBelief in a wide variety of products, including Google Search [11], our advertising products, our speech recognition systems [50, 6, 46], Google Photos [43], Google Maps and StreetView [19], Google Translate [18], YouTube, and many others.

Based on our experience with DistBelief and a more complete understanding of the desirable system properties and requirements for training and using neural networks, we have built TensorFlow, our second-generation system for the implementation and deployment of large-scale machine learning models. TensorFlow takes computations described using a dataflow-like model and maps them onto a wide variety of different hardware platforms, ranging from running inference on mobile

Strong External Adoption

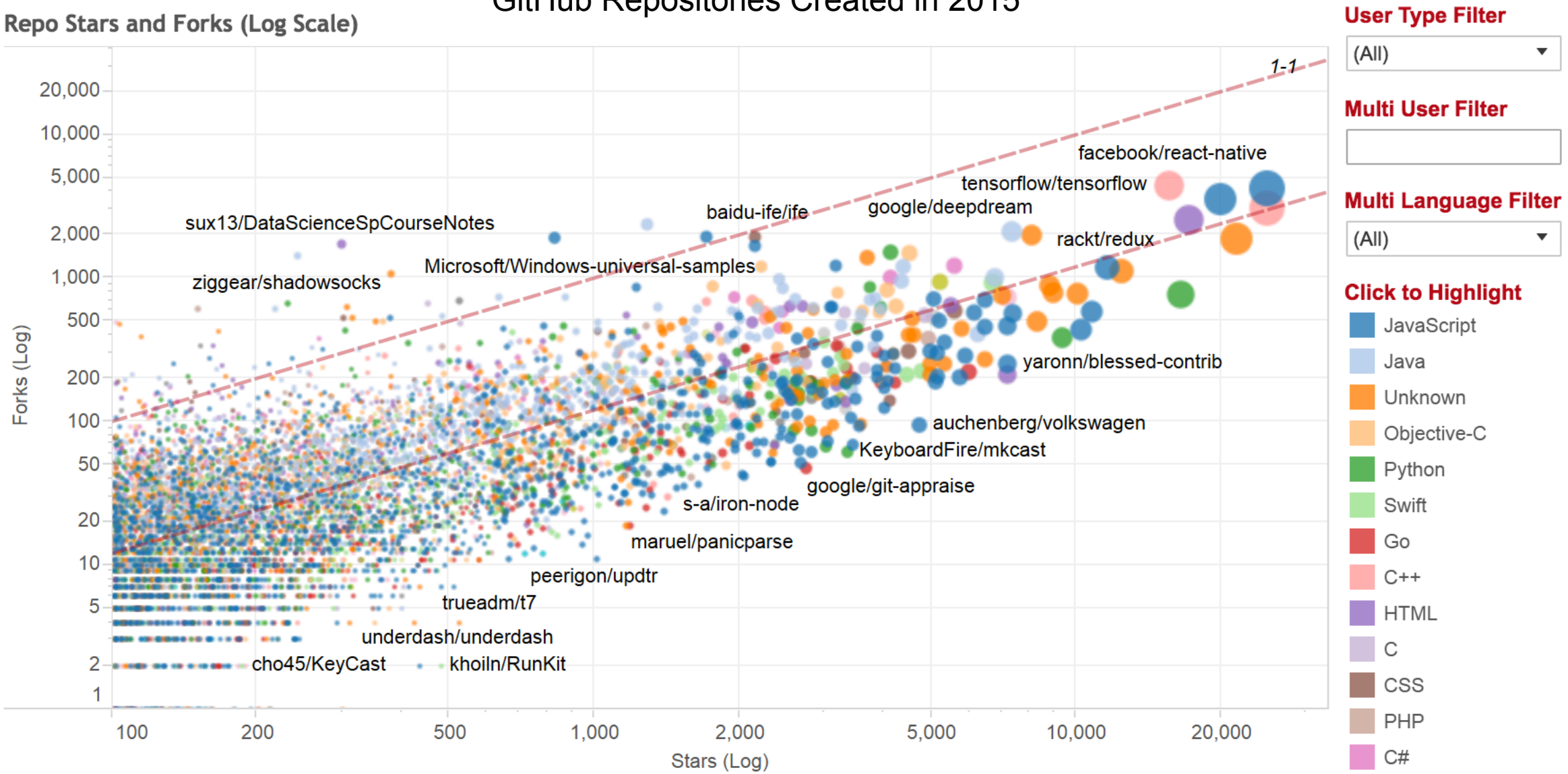


50,000+ binary installs in 72 hours, 500,000+ since Nov, 2015

Most forks of any GitHub repo in 2015, despite only being available starting in Nov, 2015

(source: http://donnemartin.com/viz/pages/2015)

GitHub Repositories Created in 2015



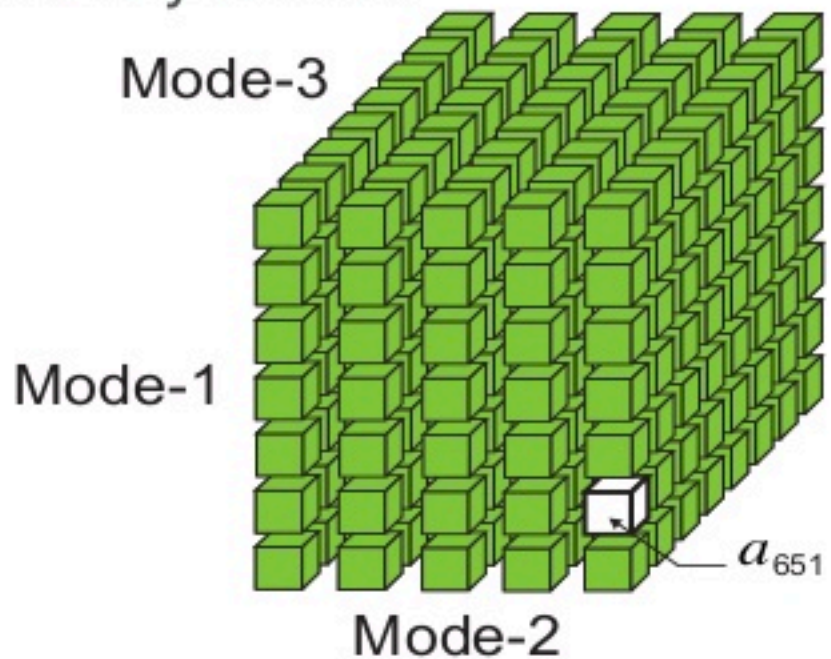
Hover: *View info* | ***Click***: *View repo url*
Interact *with the* ***filters***

Data: *Repos created in 2015, >= 100 stars* | ***Date Range***: *1/1/2015 to 1/1/2016*
FAQ: *See the final tab "A" for more info*

http://donnemartin.com/viz/pages/2015

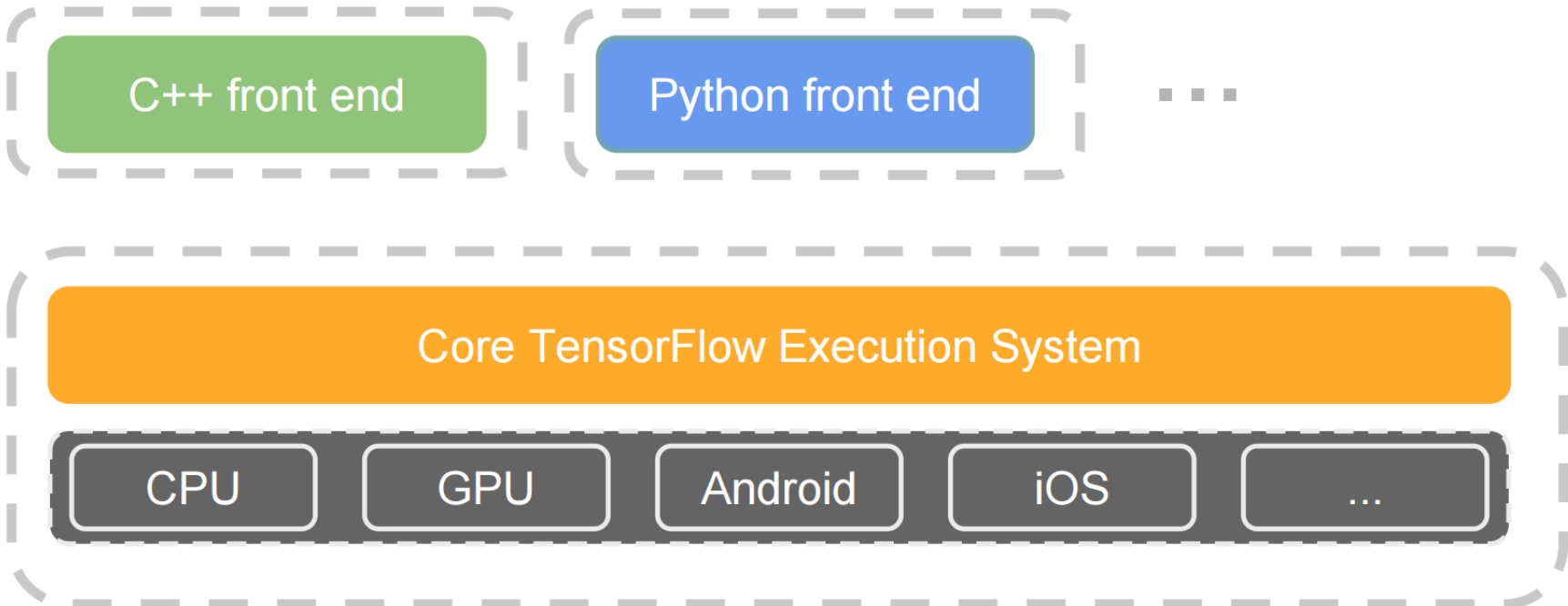
WHAT IS A TENSOR?

A tensor is a multidimensional array
E.g., three-way tensor:



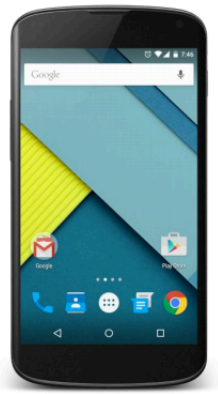
TensorFlow: Expressing High-Level ML Computations

- Core in C++
  - Very low overhead
- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more

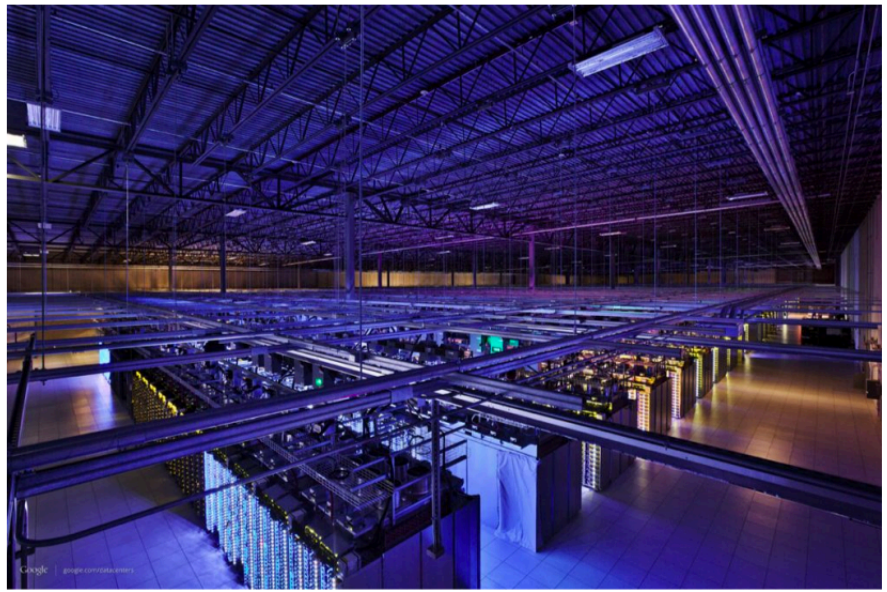


Automatically Runs on Variety of Platforms

phones

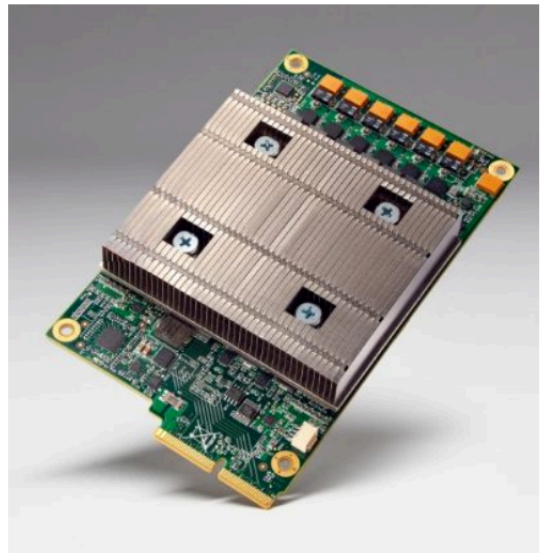


distributed systems of 100s of machines and/or GPU cards



single machines (CPU and/or GPUs) …

custom ML hardware



TensorFlow

From Google
All about building and executing computation graphs
Easy visualizations (TensorBoard)
Multi-GPU and multi-node training

Consists of:

- API for defining computation graphs
- Automatic differentiation engine
- Optimization algorithms
- ML goodies – mini-batching, test/train splitting, etc.
- Visualization tool - Tensorboard

TensorFlow: Two-Layer Net

```python
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

TensorFlow: Two-Layer Net

Create placeholders for data and labels: These will be fed to the graph

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

TensorFlow: Two-Layer Net

Create Variables to hold weights; similar to Theano shared variables

Initialize variables with numpy arrays

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

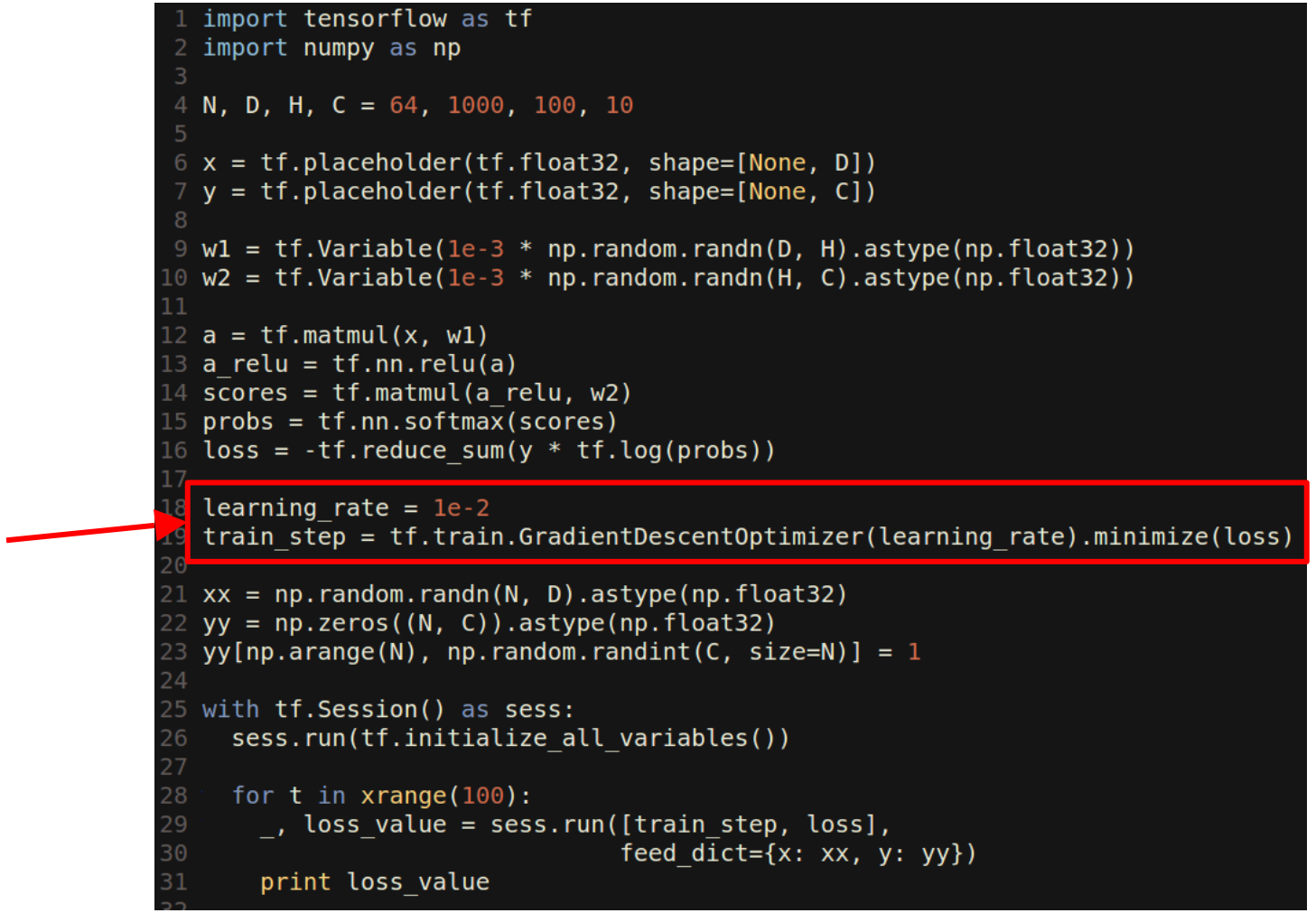
TensorFlow: Two-Layer Net

Forward: Compute scores, probs, loss (symbolically)

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

TensorFlow: Two-Layer Net

Running train_step will use SGD to minimize loss



```python
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

TensorFlow: Two-Layer Net

Create an artificial dataset; y is one-hot like Keras

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

TensorFlow: Two-Layer Net

Actually train the model

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    _, loss_value = sess.run([train_step, loss],
                             feed_dict={x: xx, y: yy})
    print loss_value
```

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

x = tf.placeholder(tf.float32, [None, 784])

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

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

init = tf.initialize_all_variables()

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

for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch( 100 )
  sess.run( train_step, feed_dict={x: batch_xs, y_: batch_ys} )
  print( sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}) )
```

Tensorboard

TensorFlow: Tensorboard

Tensorboard makes it easy to visualize what's happening inside your models

```python
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  merged = tf.merge_all_summaries()
  writer = tf.train.SummaryWriter('/tmp/fc_logs', sess.graph_def)
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    summary_str, _, loss_value = sess.run(
                                  [merged, train_step, loss],
                                  feed_dict={x: xx, y: yy})
    writer.add_summary(summary_str, t)
    print loss_value
```

TensorFlow: Tensorboard

Tensorboard makes it easy to visualize what's happening inside your models

Same as before, but now we create summaries for loss and weights

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  merged = tf.merge_all_summaries()
  writer = tf.train.SummaryWriter('/tmp/fc_logs', sess.graph_def)
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    summary_str, _, loss_value = sess.run(
                                  [merged, train_step, loss],
                                  feed_dict={x: xx, y: yy})
    writer.add_summary(summary_str, t)
    print loss_value
```

TensorFlow: Tensorboard

Tensorboard makes it easy to visualize what's happening inside your models

Create a special "merged" variable and a SummaryWriter object

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  merged = tf.merge_all_summaries()
  writer = tf.train.SummaryWriter('/tmp/fc_logs', sess.graph_def)
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    summary_str, _, loss_value = sess.run(
                                  [merged, train_step, loss],
                                  feed_dict={x: xx, y: yy})
    writer.add_summary(summary_str, t)
    print loss_value
```

TensorFlow: Tensorboard

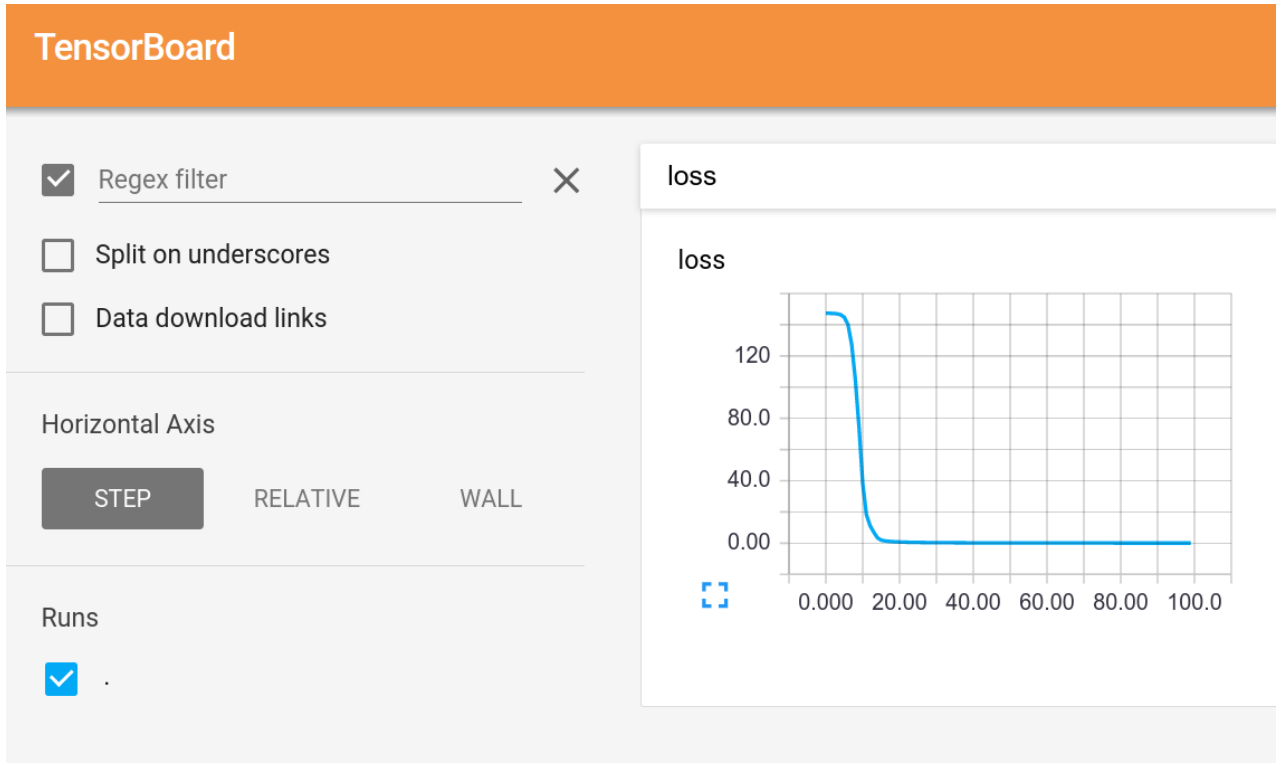
Tensorboard makes it easy to visualize what's happening inside your models

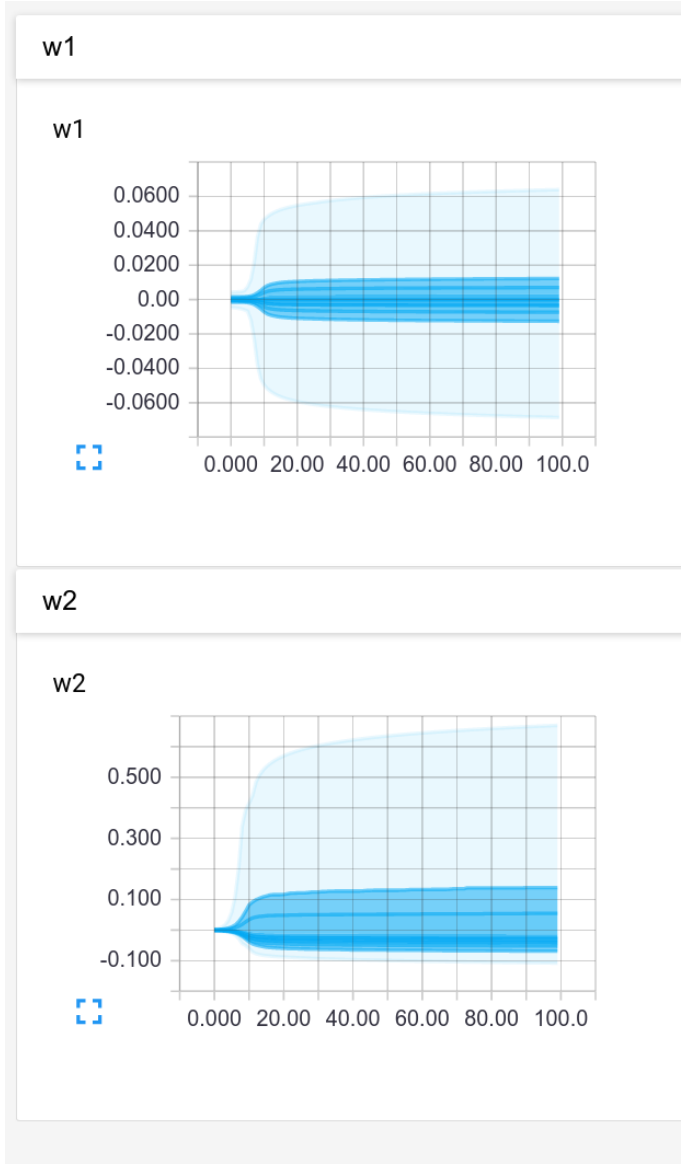
In the training loop, also run merged and pass its value to the writer

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D])
y = tf.placeholder(tf.float32, shape=[None, C])

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32))
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32))

a = tf.matmul(x, w1)
a_relu = tf.nn.relu(a)
scores = tf.matmul(a_relu, w2)
probs = tf.nn.softmax(scores)
loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1

with tf.Session() as sess:
  merged = tf.merge_all_summaries()
  writer = tf.train.SummaryWriter('/tmp/fc_logs', sess.graph_def)
  sess.run(tf.initialize_all_variables())

  for t in xrange(100):
    summary_str, _, loss_value = sess.run(
                                  [merged, train_step, loss],
                                  feed_dict={x: xx, y: yy})
    writer.add_summary(summary_str, t)
    print loss_value
```

TensorFlow: Tensorboard

Start Tensorboard server, and we get graphs!





TensorFlow: TensorBoard

```python
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D], name='x')
y = tf.placeholder(tf.float32, shape=[None, C], name='y')

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32), name='w1')
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32), name='w2')

with tf.name_scope('scores') as scope:
  a = tf.matmul(x, w1)
  a_relu = tf.nn.relu(a)
  scores = tf.matmul(a_relu, w2)
with tf.name_scope('loss') as scope:
  probs = tf.nn.softmax(scores)
  loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1
```

TensorFlow: TensorBoard

Add names to placeholders and variables

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D], name='x')
y = tf.placeholder(tf.float32, shape=[None, C], name='y')

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32), name='w1')
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32), name='w2')

with tf.name_scope('scores') as scope:
  a = tf.matmul(x, w1)
  a_relu = tf.nn.relu(a)
  scores = tf.matmul(a_relu, w2)
with tf.name_scope('loss') as scope:
  probs = tf.nn.softmax(scores)
  loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1
```

TensorFlow: TensorBoard

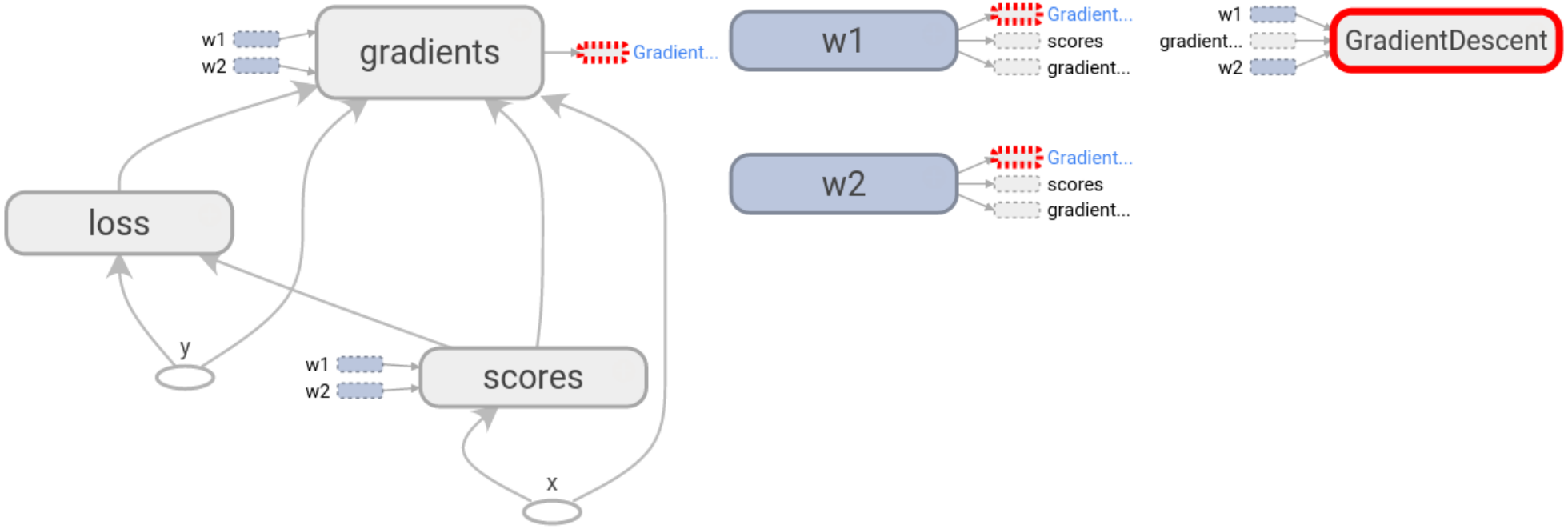
Add names to placeholders and variables

Break up the forward pass with name scoping

```
import tensorflow as tf
import numpy as np

N, D, H, C = 64, 1000, 100, 10

x = tf.placeholder(tf.float32, shape=[None, D], name='x')
y = tf.placeholder(tf.float32, shape=[None, C], name='y')

w1 = tf.Variable(1e-3 * np.random.randn(D, H).astype(np.float32), name='w1')
w2 = tf.Variable(1e-3 * np.random.randn(H, C).astype(np.float32), name='w2')

with tf.name_scope('scores') as scope:
  a = tf.matmul(x, w1)
  a_relu = tf.nn.relu(a)
  scores = tf.matmul(a_relu, w2)
with tf.name_scope('loss') as scope:
  probs = tf.nn.softmax(scores)
  loss = -tf.reduce_sum(y * tf.log(probs))

loss_summary = tf.scalar_summary('loss', loss)
w1_hist = tf.histogram_summary('w1', w1)
w2_hist = tf.histogram_summary('w2', w2)

learning_rate = 1e-2
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

xx = np.random.randn(N, D).astype(np.float32)
yy = np.zeros((N, C)).astype(np.float32)
yy[np.arange(N), np.random.randint(C, size=N)] = 1
```

TensorFlow: TensorBoard

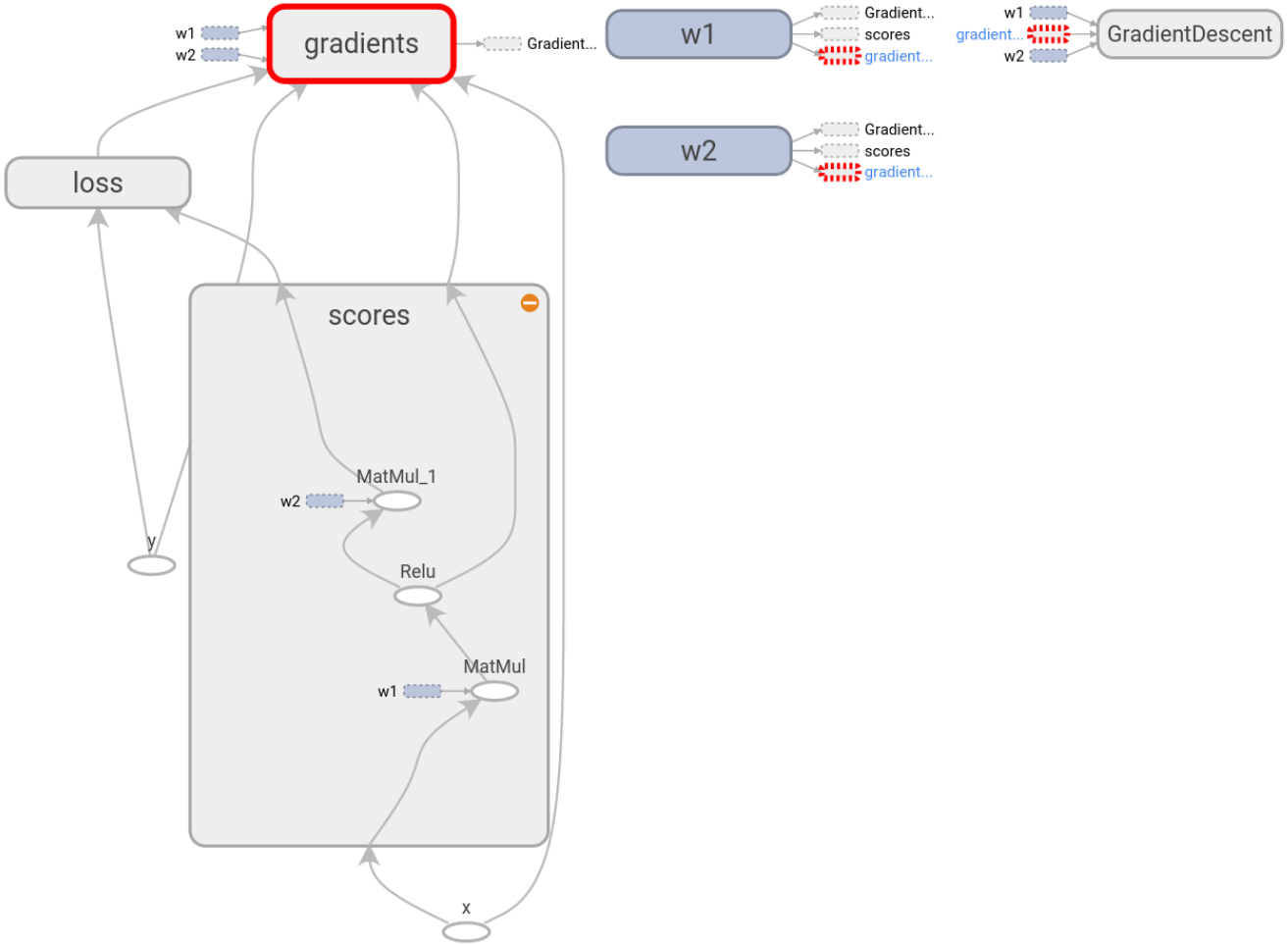
Tensorboard shows the graph!



TensorFlow: TensorBoard

Tensorboard shows the graph!

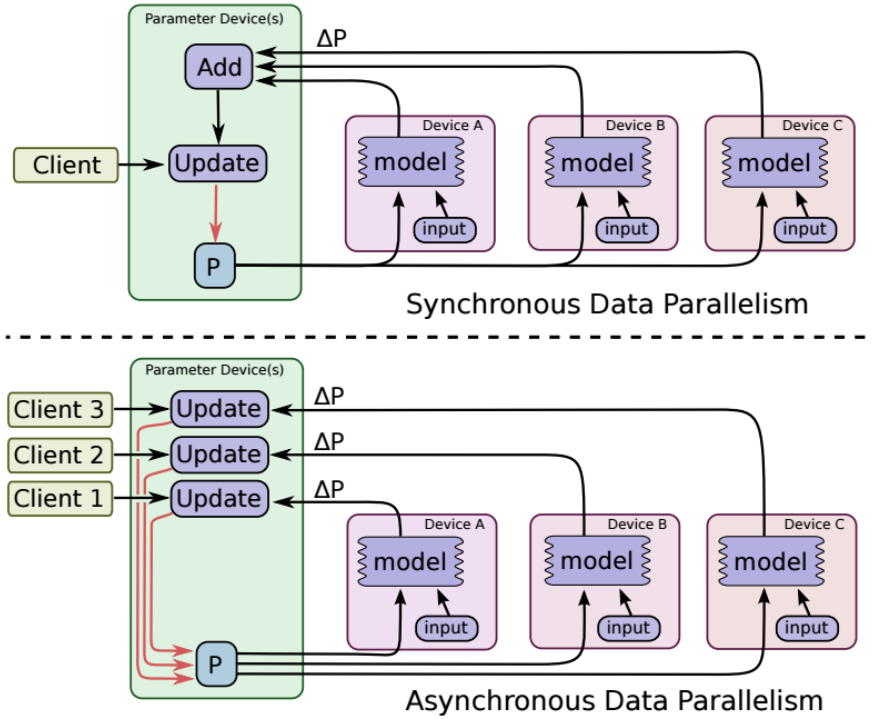
Name scopes expand to show individual operations



TensorFlow: Multi-GPU

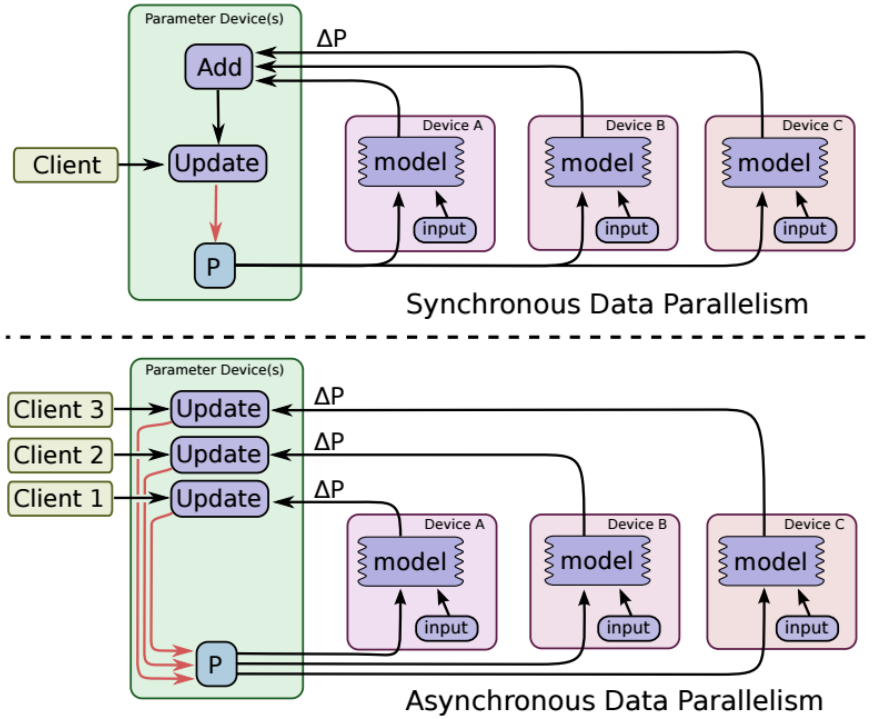
Data parallelism:

synchronous or asynchronous

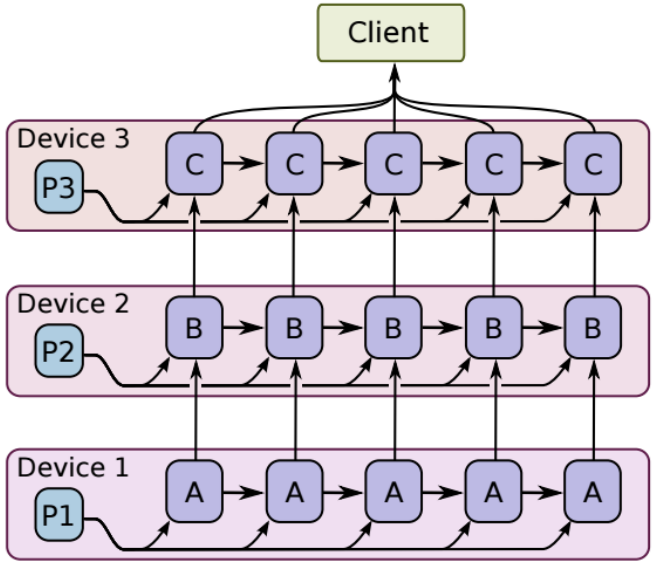


TensorFlow: Multi-GPU

Data parallelism:
synchronous or asynchronous

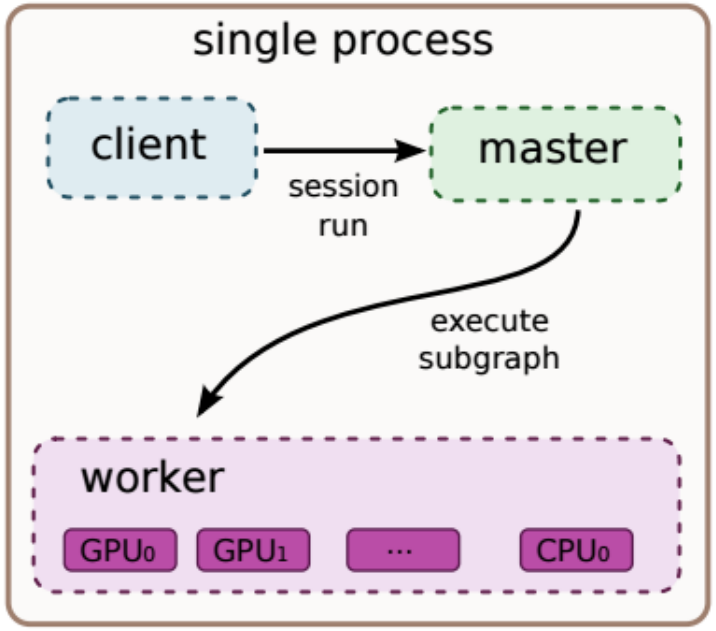


Model parallelism:
Split model across GPUs

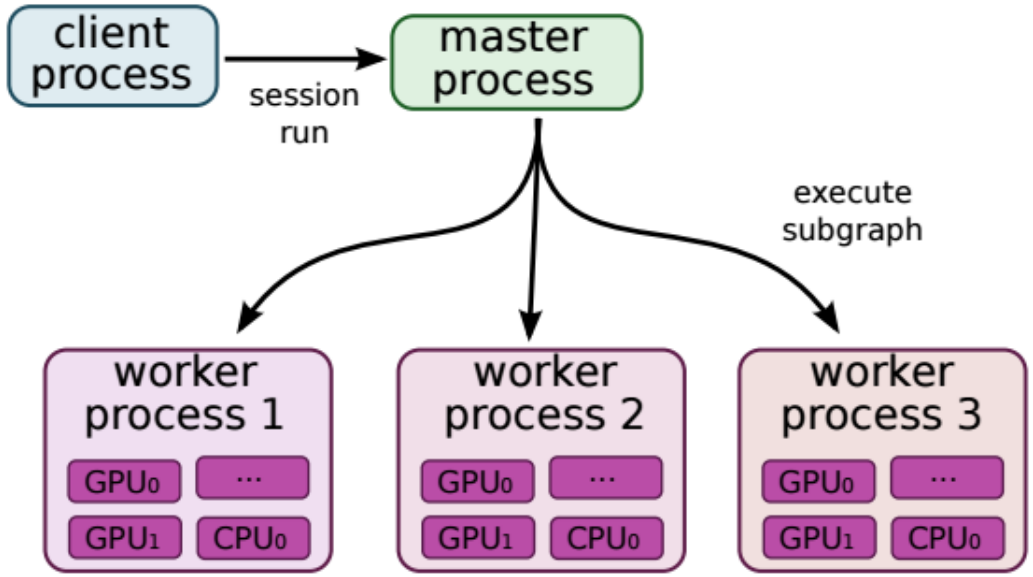


TensorFlow: Distributed

Single machine:
Like other frameworks



Many machines:



Overview

TensorFlow: Pros / Cons

- (+) Python + numpy
- (+) Computational graph abstraction (great for RNNs)
- (+) TensorBoard for visualization
- (+) Data AND model parallelism; best of all frameworks
- (+) Distributed models
- (-) Slower than other frameworks right now
- (-) Much “fatter” than Torch; more magic
- (-) Not many pretrained models

Comparison to other DL frameworks

	Caffe	Torch	Theano	TensorFlow
Language	C++, Python	Lua	Python	Python
Pretrained	Yes ++	Yes ++	Yes (Lasagne)	Inception
Multi-GPU: Data parallel	Yes	Yes cunn.DataParallelTable	Yes platoon	Yes
Multi-GPU: Model parallel	No	Yes fbcunn.ModelParallel	Experimental	Yes (best)
Readable source code	Yes (C++)	Yes (Lua)	No	No
Good at RNN	No	Mediocre	Yes	Yes (best)