Advanced Spark and TensorFlow

Meetup

METIS

METIS

May 26, 2016

Meetup Agenda

Meetup Updates (Chris F)

Technology Updates (Chris F)

Spark + TensorFlow Model Serving/Deployment (Chris F)

Neural Net and TensorFlow Landscape (Chris F)

TensorFlow Core, Best Practices, Hidden Gems (Sam A)

TensorFlow Distributed (Fabrizio M)

Advanced Spark and TensorFlow Meetup

Meetup Updates

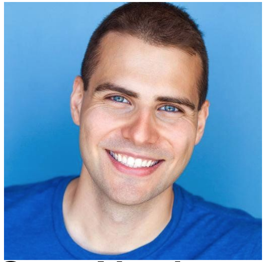
New Sponsor

METIS

Meetup Metrics

Spark and TensorFlow Experts	3,461
Group reviews	3
Upcoming Meetups	22
Past Meetups	52

Co-presenters



Sam Abrahams
“LA Kid (from DC)

Fabrizio Milo
“Italian Stallion”

Chris Fregly
“Chicago Fire”

Workshop - June 4th - Spark ML + TensorFlow

PANCAKE STACK

Presto

Arrow

NiFi

Cassandra

AirFlow

Kafka

ElasticSearch

Spark

TensorFlow

Algebird

CoreNLP

Kibana

advancedspark.com.

advancedspark.com

Chris Fregly

Technology Updates

github.com/fluxcapacitor/pipeline

Neural Network Tool Landscape

TensorFlow Tool Landscape

TensorFlow Serving

Spark ML Serving

pipeline.io (Shh… Stealth)

Technology Updates...

Spark 2.0

Kafka 0.10.0 + Confluent 3.0

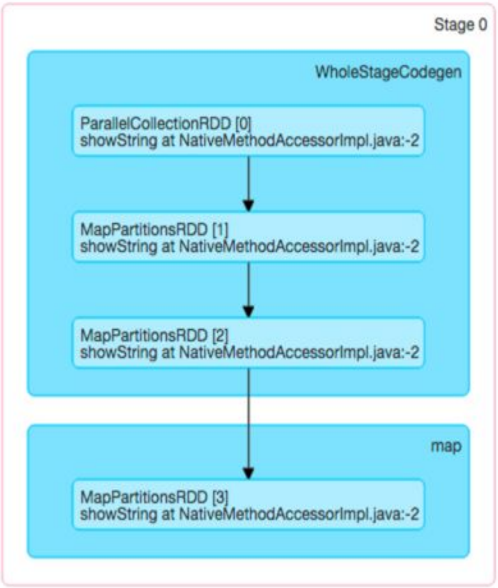
CUDA 8.0 + cuDNN v5

```
> spark.range(1000).filter("id > 100").selectExpr("sum(id)").explain()

== Physical Plan ==
*TungstenAggregate(key=[], functions=[(sum(id#201L),mode=Final,isDistinct=false)], output=[sum(id)#212L])
+- Exchange SinglePartition, None
   +- *TungstenAggregate(key=[], functions=[(sum(id#201L),mode=Partial,isDistinct=false)], output=[sum#214L])
      +- *Filter (id#201L > 100)
         +- *Range 0, 1, 3, 1000, [id#201L]
```

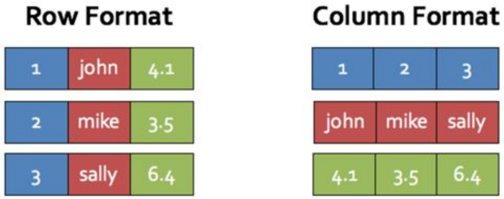
Spark 2.0: Core

Whole-Stage Code Gen

- SPARK-12795
- Physically Fuse Together Operators (within a Stage) into 1 Operation
- Avoids Excessive Virtual Function Calls
- Utilize CPU Registers vs. L1, L2, Main Memory
- Loop Unrolling and SIMD Code Generation
- Speeds up CPU-bound workloads (not I/O-bound)



Vectorization (SPARK-12992)



- Operate on Batches of Data
- Reduces Virtual Function Calls
- Fallback if Whole-Stage Not an Option

Spark 2.0: ML

Save/Load Support for All Models and Pipelines!

- Python and Scala
- Saved as Parquet ONLY

Local Linear Algebra Library

- SPARK-13944, SPARK-14615
- Drop-in Replacement for Distributed Linear Algebra library
- Opens up door to my new **pipeline.io** prediction layer!

Kafka v0.10 + Confluent v3.0 Platform

Kafka Streams

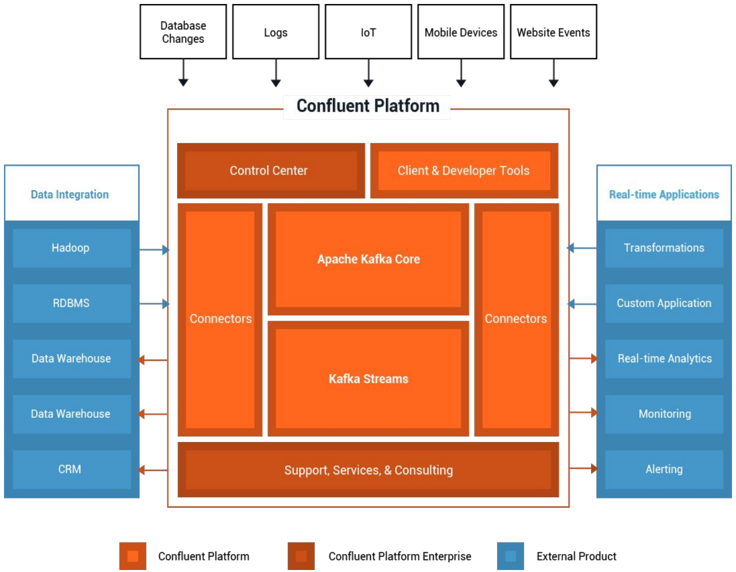
New Event-Creation Timestamp

Rack Awareness (THX NFLX!)

Kerberos + SASL Improvements

Ability to Pause a Connector (ie. Maintenance)

New **max.poll.records** to limit messages retrieved

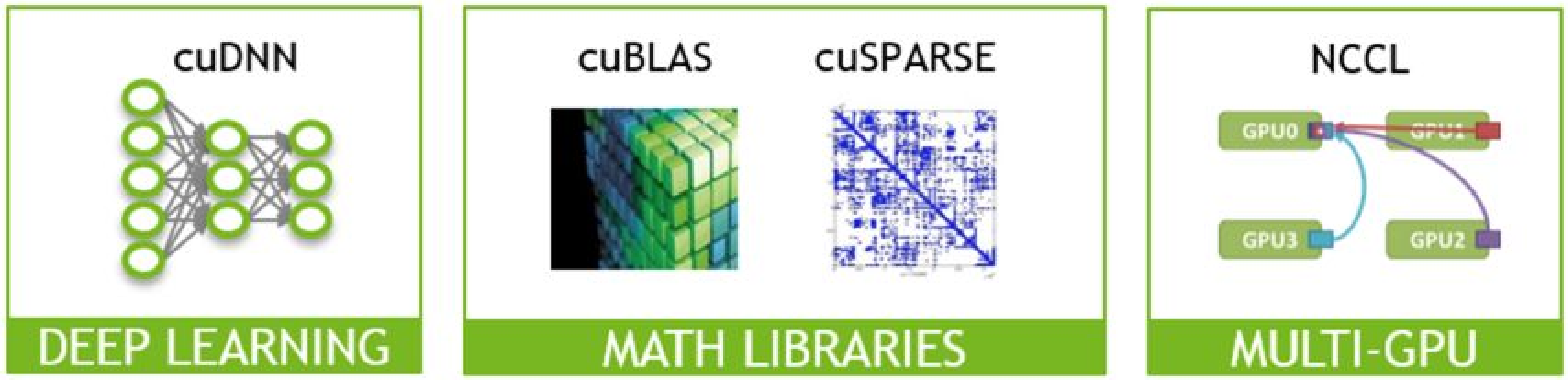


CUDA Deep Neural Network (cuDNN) v5 Updates

LSTM (Long Short-Term Memory) RNN Support for NLP use cases (6x speedup)

Optimized for NVIDIA Pascal GPU Architecture including FP16 (low precision)

Highly-optimized networks with 3x3 convolutions (GoogleNet)



github.com/fluxcapacitor/pipeline Updates

Tools and Examples

- JupyterHub and Python 3 Support
- Spark-Redis Connector
- Theano
- Keras (TensorFlow + Theano Support)

Code

- Spark ML DecisionTree Code Generator (Janino JVM ByteCode Generator)
- Hot-swappable ML Model Watcher (similar to TensorFlow Serving)
- Eigenface-based Image Recommendations
- Streaming Matrix Factorization w/ Kafka
- Netflix Hystrix-based Circuit Breaker - Prediction Service @ Scale

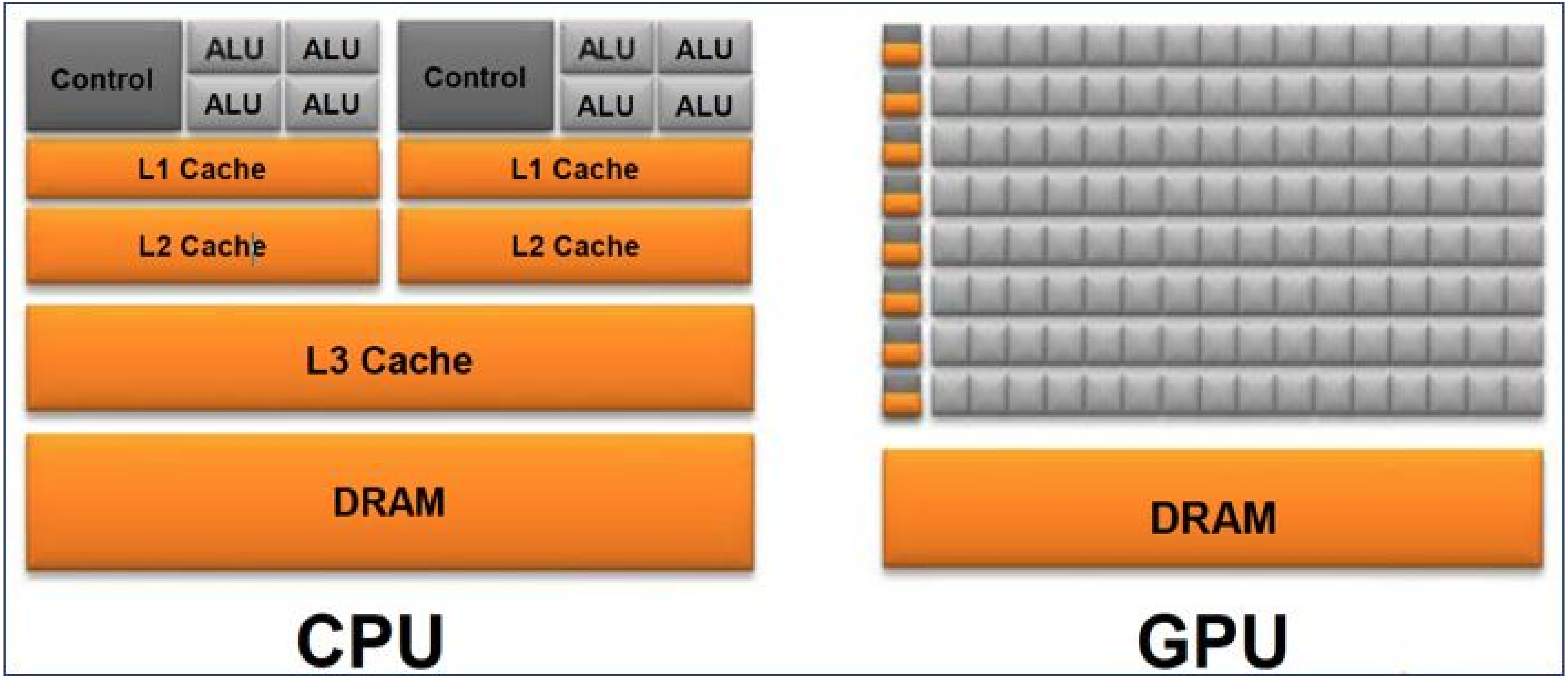
Neural Network Landscape

Caffe
Chainer
DL4J Deeplearning4j
K KERAS
Microsoft CNTK
MatConvNet

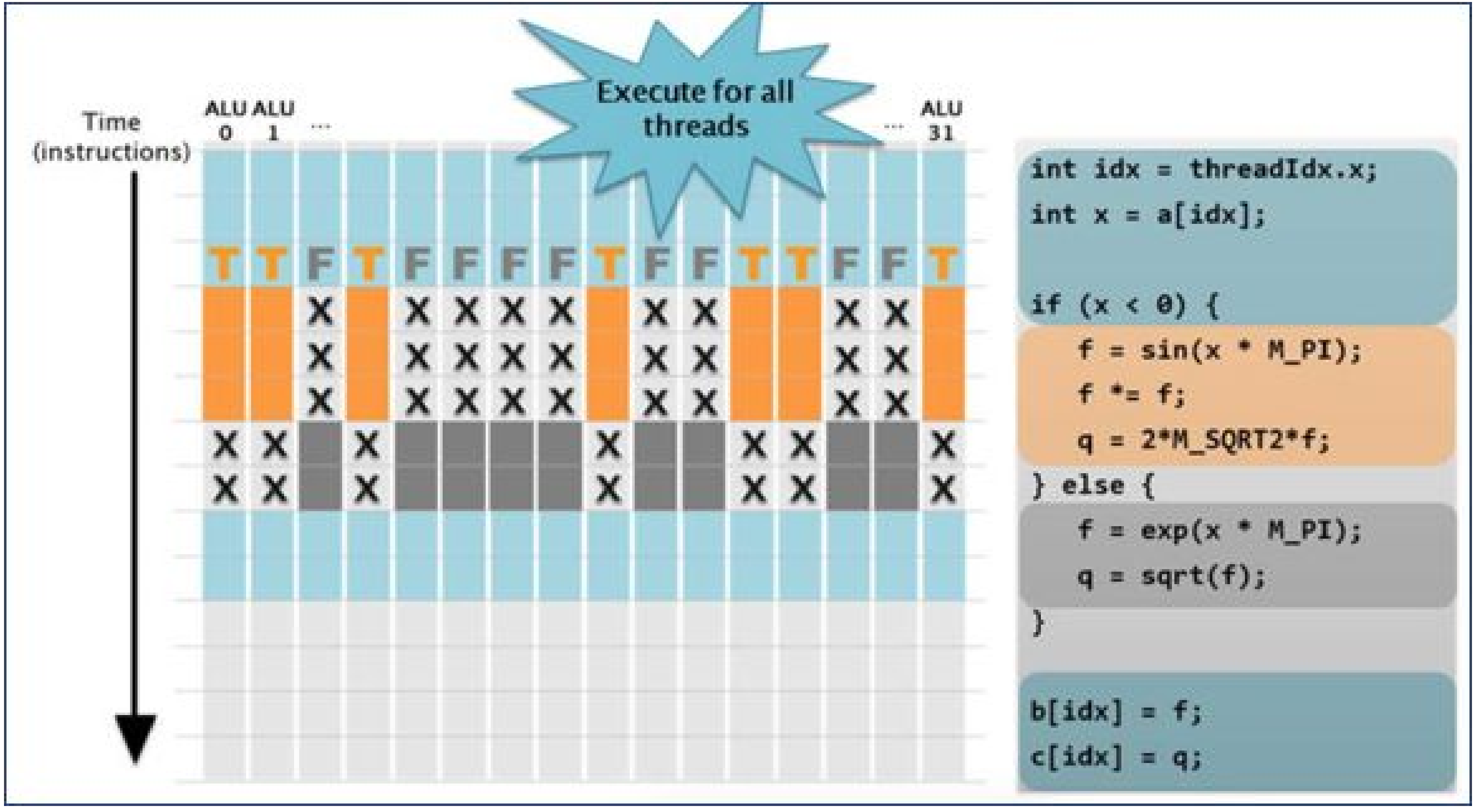
MINERVA
mxnet
Purine
TensorFlow
theano
torch

Interesting Neural Net Use Case

Bonus: CPU vs. GPU



Bonus! GPUs and Branching



TensorFlow Landscape

TensorFlow Core

TensorFlow Distributed

TensorFlow Serving (similar to Prediction.IO, Pipeline.IO)

TensorBoard (Visualize Neural Network Training)

Playground (Toy)

SkFlow (Scikit-Learn + TensorFlow)

Keras (High-level API for both TensorFlow and Theano)

Models (Parsey McParseface/SyntaxNet)

TensorFlow Serving (Model Deployment)

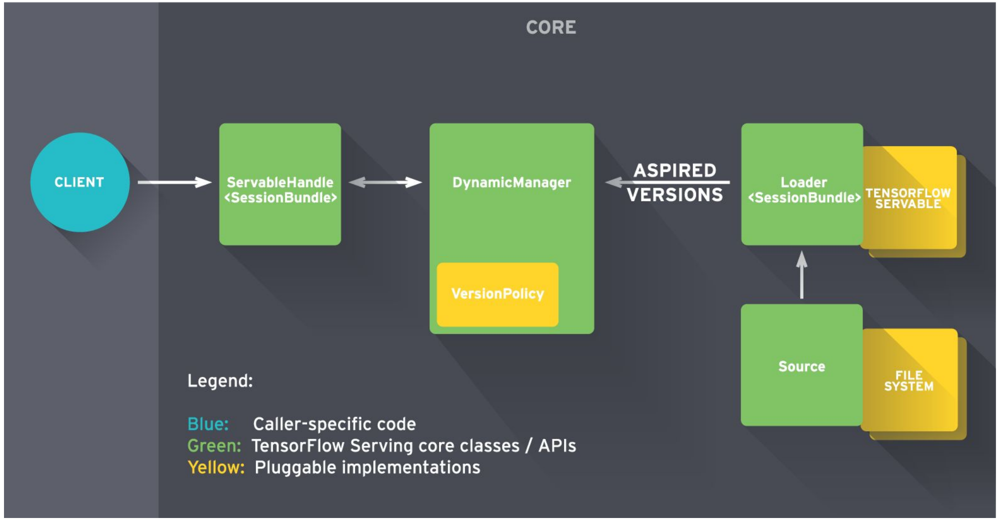
Dynamic Model Loader

Model Versioning & Rollback

Written in C/C++

Extend to Serve any Model!

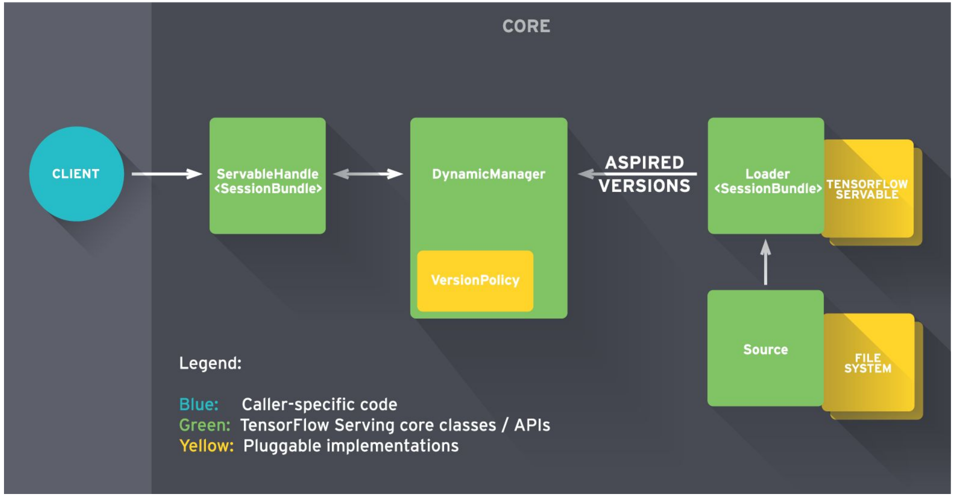
Demos!!



Spark ML Serving (Model Deployment)

Same thing except for Spark...

Keep an eye on **pipeline.io!**



Sam Abrahams

TensorFlow Core
Best Practices
Hidden Gems

TensorFlow Core Terminology

gRPC (?) - Should this go here or in distributed?

TensorBoard Overview (?)

Explaining this will go a long way ->

Symbol	Meaning
	High-level node representing a name scope. Double-click to expand a high-level node.
	Sequence of numbered nodes that are not connected to each other.
	Sequence of numbered nodes that are connected to each other.
	An individual operation node.
	A constant.
	A summary node.
	Edge showing the data flow between operations.
	Edge showing the control dependency between operations.
	A reference edge showing that the outgoing operation node can mutate the incoming tensor.

Who dis?

- My name is Sam Abrahams
- Los Angeles based machine learning engineer
- TensorFlow White Paper Notes
- TensorFlow for Raspberry Pi
- Contributor to the TensorFlow project

samjabrahams

@sabraha

This Talk: Sprint Through:

- Core TensorFlow API and terminology
- TensorFlow workflow
- Example TensorFlow Code
- TensorBoard

TensorFlow Programming Model

- **Very** similar to Theano
- The primary user-facing API is Python, though there is a partial C++ API for executing models
- Computational code is written in C++, implemented for CPUs, GPUs, or both
- Integrates tightly with NumPy

TensorFlow Programming

Generally boils down to two steps:

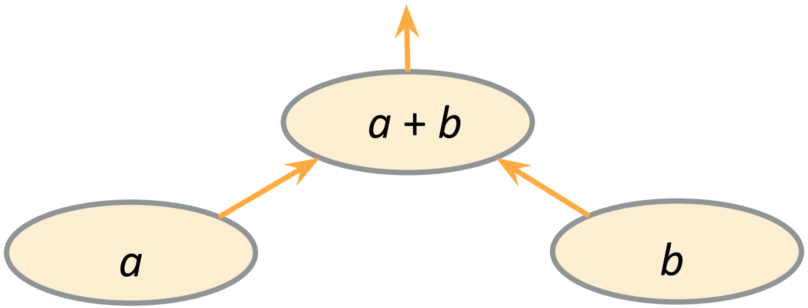
1. Build the computational graph(s) that you'd like to run
2. Use a TensorFlow Session to run your graph one or more times

Graph

- The primary structure of a TensorFlow model
- Generally there is one graph per program, but TensorFlow can support multiple Graphs
- Nodes represent computations or data transformations
- Edges represent data transfer or computational control

What is a data flow graph?

- Also known as a "computational graph", or just "graph"
- Nice way to visualize series of mathematical computations
- Here's a simple graph showing the addition of two variables:



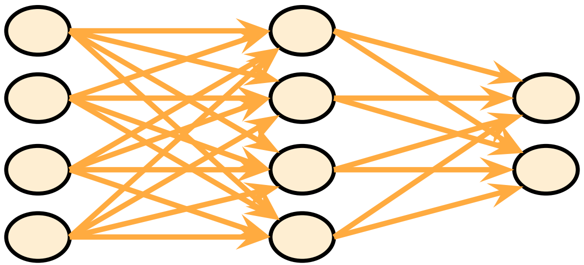
Components of a Graph

Graphs are composed of two types of elements:

- **Nodes**
  - These are the elliptical shapes in the graph, and represent some form of computation
- **Edges**
  - These are the arrows in the graph, and represent data flowing from one node into another node.

Why Use Graphs?

- They are highly compositional
  - Useful for calculating derivatives
- It’s easier to implement distributed computation
  - Computation is already segmented nicely
- Neural networks are already implemented as computational graphs!



Tensors

- N-Dimensional Matrices
  - 0-Dimensional → Scalar
  - 1-Dimensional → Vector
  - 2-Dimensional → Matrix

- All data that moves through a TensorFlow graph is a Tensor
- TensorFlow can convert Python native types or NumPy arrays into Tensor objects

Tensors

- Python

```
# Scalar
> 3

# Matrix
> [[1, 2, 3],
   [4, 5, 6],
   [7, 8, 9]]
```

```
# Vector
> [1, 2, 3]

# 3-D Tensor
> [[ [0,0], [0,1], [0,2] ],
   [ [1,0], [1,1], [1,2] ],
   [ [2,0], [2,1], [2,2] ]
```

- NumPy

```
# Matrix as NumPy Array
> np.array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]],
           dtype=np.int64)
```

Tensors

- Best practice is to use NumPy arrays when directly defining Tensors
  - Can explicitly set data type
- This presentation does not create Tensors with NumPy
  - Space is limited
  - I am lazy
- Tensors returned by TensorFlow graphs are NumPy arrays

Operations

- "Op" for short
- Represent any sort of computation
- Take in zero or more Tensors as input, and output zero or more Tensors
- Numerous uses: perform matrix algebra, initialize variables, print info to the console, etc.
- Do not run when defined: they must be called from a TensorFlow Session (coming up later)

Operations

- Quick Example

```
> import tensorflow as tf
> a = tf.mul(3,5)       #  Returns handle to new tf.mul node
> sess = tf.Session()   #  Creates TF Session
> sess.run(a)           #  Actually runs the Operation

out: 15
```

Placeholders

- Define “input” nodes
  - Specifies information that be provided when graph is run
  - Typically used for training data
- Define the tensor shape and data type when created:

```
import tensorflow as tf

# Create a placeholder of size 100x400
# With 32-bit floating point data type
my_placeholder = tf.placeholder(tf.float32, shape=(100,400))
```

TensorFlow Session

- In charge of coordinating graph execution
- Most important method is run()
  - This is what actually runs the Graph
  - It takes in two parameters, ‘fetches’ and ‘feed_dict’
  - ‘fetches’ is a list of objects you’d like to get the results for, such as the final layer in a neural network
  - ‘feed_dict’ is a dictionary that maps tensors (often Placeholders) to values those tensors should use

TensorFlow Variables

- Contain tensor data that persists each time you run your Graph
  - Typically used to hold weights and biases of a machine learning model
  - The final values of the weights and biases, along with the Graph shape, define a trained model
- Before being run in a Graph, must be initialized (will discuss this at the Session slide)

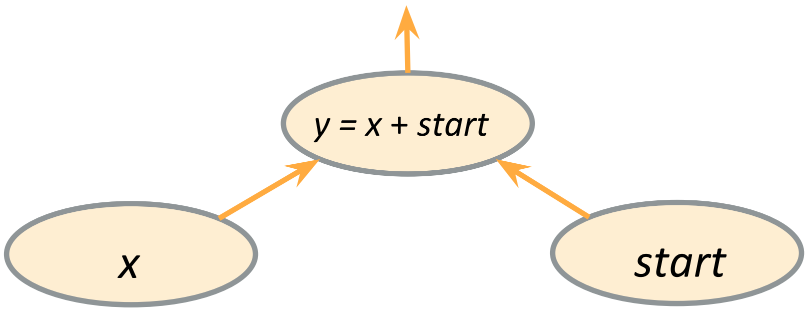
TensorFlow Variables

- Old school: Define with tensorflow.Variable()
- Best practices: tf.get_variable()
- Update its information with the assign() method

```
Import tensorflow as tf

# Create a variable with value 0 and name ‘my_variable’
my_var = tf.get_variable(0, name=‘my_variable’)

# Increment the variable by one
my_var.assign(my_var + 1)
```

Building the graph!

```
import tensorflow as tf

# create a placeholder for inputting floating point data
x = tf.placeholder(tf.float32)

# Make a Variable with the starting value of 0
start = tf.Variable(0.0)

# Create a node that is the value of (start + x)
y = start.assign(start + x)
```



Running a Session (using previous graph)

- Start a Session with tensorflow.Session()
- Close it when you're done!

```
# Open up a TensorFlow Session
# and assign it to the handle 'sess'
sess = tf.Session()

# Important: initialize the Variable
init = tf.initialize_all_variables
sess.run(init)

# Run the graph to get the value of y
# Feed in different values of x each time
print(sess.run(y, feed_dict={x:1}))    # Prints 1.0
print(sess.run(y, feed_dict={x:0.5}))  # Prints 1.5
print(sess.run(y, feed_dict={x:2.2}))  # Prints 3.7

# Close the Session
sess.close()
```

Devices

- A single device is a CPU, GPU, or other computational unit (TPU?!)
- One machine can have multiple devices
- “Distributed” $\longrightarrow$ Multiple machines
- Multi-device != Distributed

TensorBoard

- One of the most useful (and underutilized) aspects of TensorFlow - Takes in serialized information from graph to visualize data.
- Complete control over what data is stored

Let’s do a brief live demo. Hopefully nothing blows up!

TensorFlow Codebase

Python client API

Swig

C++ kernel implementation

TensorFlow core framework

Standard and third party C++ libraries (Eigen)

TensorFlow Codebase Structure: Core

tensorflow/core : Primary C++ implementations and runtimes

- core/ops – Registration of Operation signatures
- core/kernels – Operation implementations
- core/framework – Fundamental TensorFlow classes and functions
- core/platform – Abstractions for operating system platforms

Based on information from Eugene Brevdo, Googler and TensorFlow member

TensorFlow Codebase Structure: Python

tensorflow/python : Python implementations, wrappers, API

- python/ops – Code that is accessed through the Python API
- python/kernel_tests – Unit tests (which means examples)
- python/framework – TensorFlow fundamental units
- python/platform – Abstracts away system-level Python calls

contrib/ : contributed or non-fully adopted code

Based on information from Eugene Brevdo, Googler and TensorFlow member

Learning TensorFlow: Beyond the Tutorials

- There is a **ton** of excellent documentation in the code itself- especially in the C++ implementations
- If you ever want to write a custom Operation or class, you need to immerse yourself
- Easiest way to dive in: look at your #include statements!
- Use existing code as reference material
- If you see something new, learn something new

Tensor - tensorflow/core/framework/tensor.h

- dtype() - returns data type
- shape() - returns TensorShape representing tensor's shape
- dims() – returns the number of dimensions in the tensor
- dim_size(int d) - returns size of specified dimension
- NumElements() - returns number of elements
- isSameSize(const Tensor& b) – do these Tensors dimensions match?
- TotalBytes() - estimated memory usage
- CopyFrom() – Copy another tensor and share its memory
- ...plus many more

Some files worth checking out:

tensorflow/core/framework

- tensor_util.h: deep copying, concatenation, splitting
- resource_mgr.h: Resource manager for mutable Operation state
- register_types.h: Useful helper macros for registering Operations
- kernel_def_builder.h: Full documentation for kernel definitions

tensorflow/core/util

- cuda_kernel_helper.h: Helper functions for cuda kernel implementations
- sparse/sparse_tensor.h: Documentation for C++ SparseTensor class

General advice for GPU implementation:

1. If possible, always register for GPU, even if it's not a full implementation
    - Want to be able to run code on GPU if it won't bottleneck the system
2. Before writing a custom GPU implementation, sanity check! Will it help?
    - Not everything benefits from parallelization
3. Utilize Eigen!
    - TensorFlow's core Tensor class is based on Eigen
4. Be careful with memory: **you** are in charge of mutable data structures

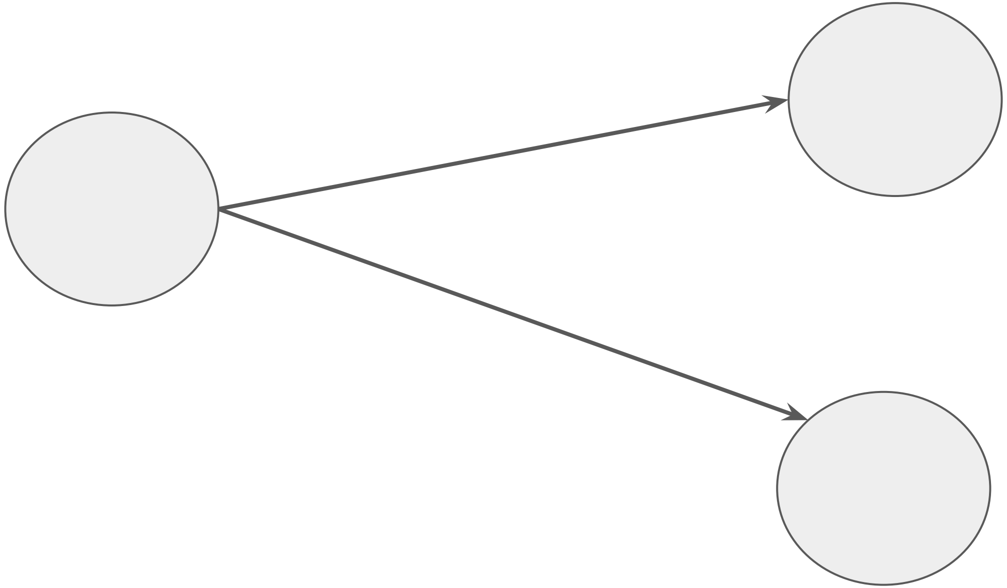
Fabrizio Milo

TensorFlow Distributed

githbu.com/Mistobaan

twitter.com/fabmilo

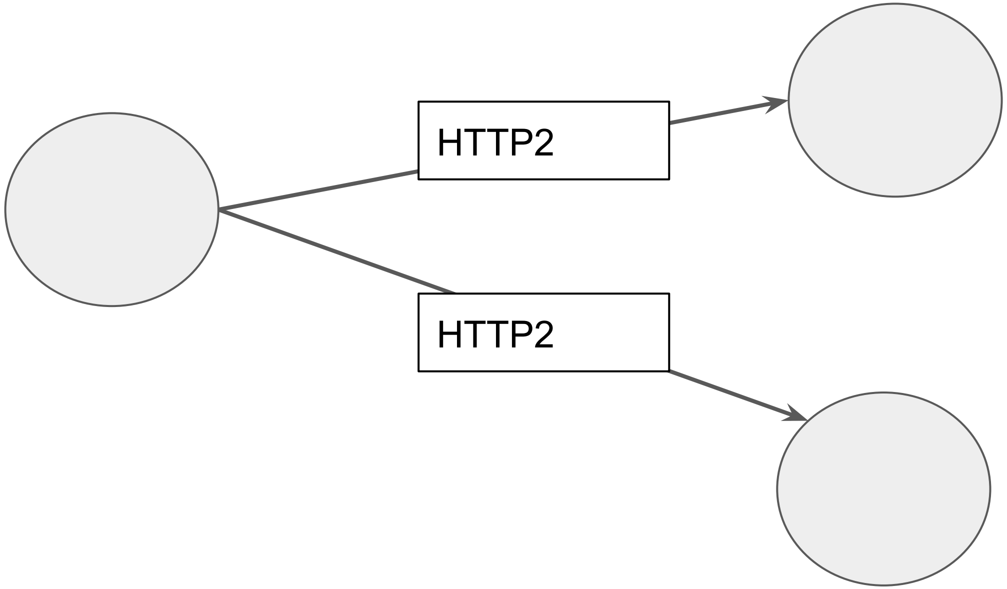
Distributed Tensorflow



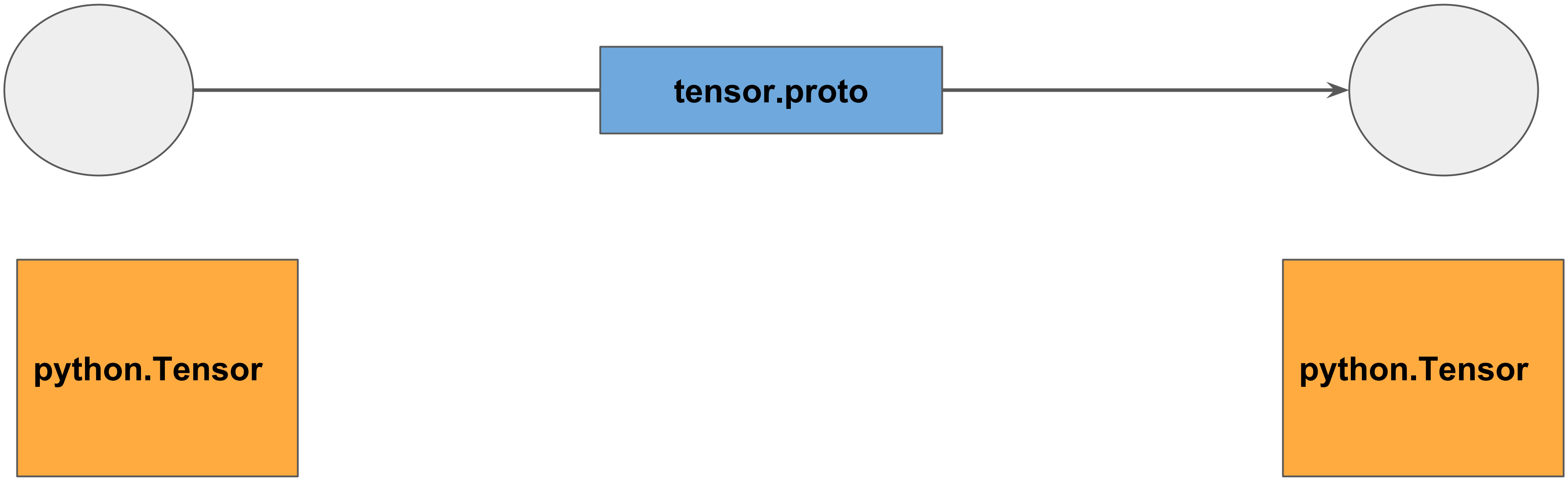
gRPC (http://www.grpc.io/)

- A high performance, open source, general RPC framework that puts mobile and HTTP/2 first.

- Protocol Buffers
- HTTP2



gRPC



Distributed Tensorflow - Terminology

- Cluster
- Jobs
- Tasks

Distributed Tensorflow - Terminology

- Cluster: **ClusterSpec**
- Jobs: Parameter Server, Worker
- Tasks: Usually 0, 1, 2, …

Example of a Cluster Spec

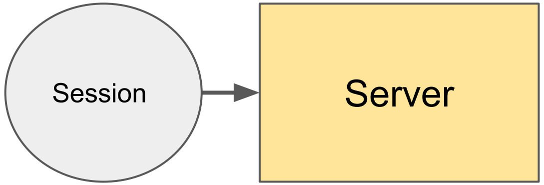
```
{
    "ps":[
        "8.9.15.20:2222",
    ],
    "workers":[
        "8.34.25.90:2222",
        "30.21.18.24:2222",
        "4.17.19.14:2222"
    ]
}
```

One Session One Server One Job One Task

```
# Start a TensorFlow server as a single-process "cluster".
$ python
>>> import tensorflow as tf
>>> c = tf.constant("Hello, distributed TensorFlow!")
>>> server = tf.train.Server.create_local_server()
>>> sess = tf.Session(server.target)  # Create a session on the server.
>>> sess.run(c)
'Hello, distributed TensorFlow!'
```

One Session One Server One Job One Task

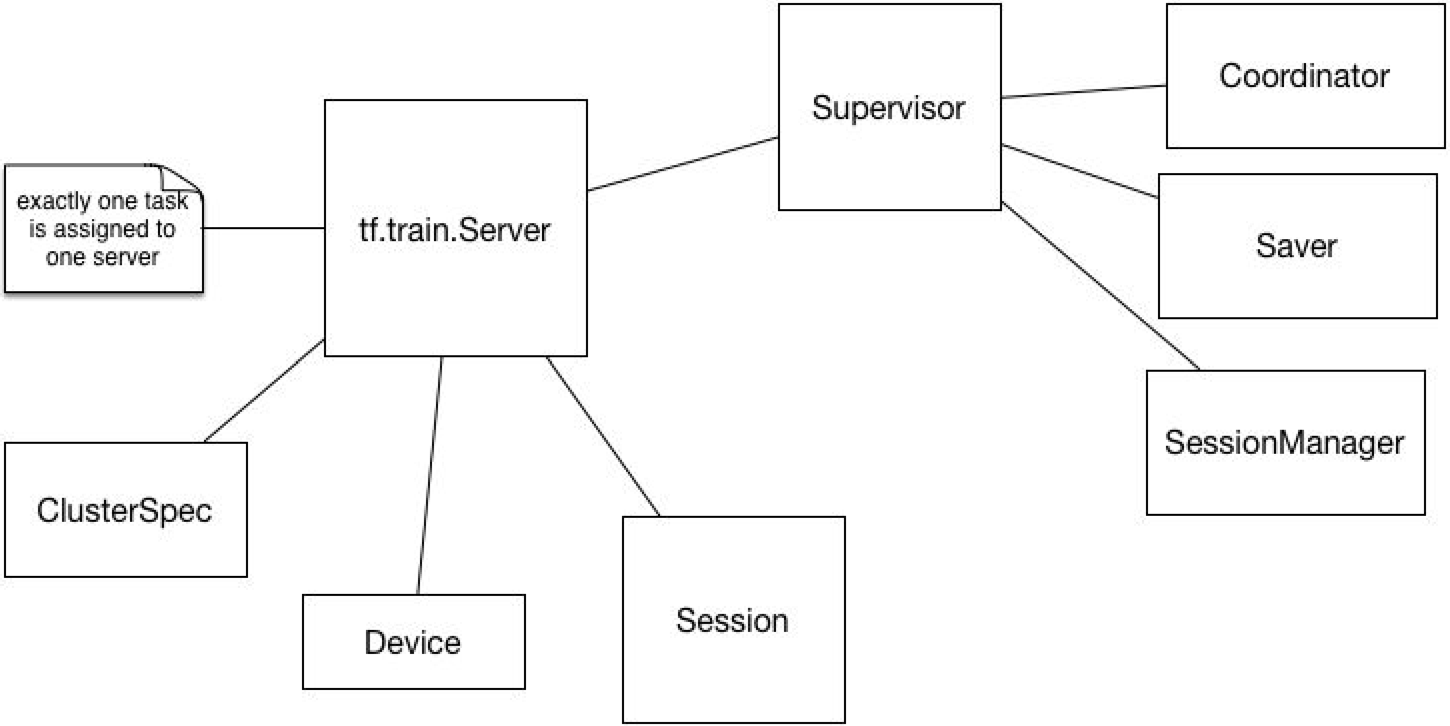
```
# Start a TensorFlow server as a single-process "cluster".
$ python
>>> import tensorflow as tf
>>> c = tf.constant("Hello, distributed TensorFlow!")
>>> server = tf.train.Server.create_local_server()
>>> sess = tf.Session(server.target)  # Create a session on the server.
>>> sess.run(c)
'Hello, distributed TensorFlow!'
```



Use Cases

- Training
- Hyper Parameters Optimization
- Ensembling

Graph Components



tf.python.training.session_manager.SessionManager

1. Checkpointing trained variables as the training progresses.
2. Initializing variables on startup, restoring them from the most recent checkpoint after a crash, or wait for checkpoints to become available.

tf.python.training.supervisor.Supervisor

```
# Single Program
 with tf.Graph().as_default():
    ...add operations to the graph...
    # Create a Supervisor that will checkpoint the model in '/tmp/mydir'.
    sv = Supervisor(logdir='/tmp/mydir')
    # Get a Tensorflow session managed by the supervisor.
    with sv.managed_session(FLAGS.master) as sess:
      # Use the session to train the graph.
      while not sv.should_stop():
        sess.run(<my_train_op>)
```

tf.python.training.supervisor.Supervisor

```
# Multiple Replica Program
  is_chief = (server_def.task_index == 0)
  server = tf.train.Server(server_def)

  with tf.Graph().as_default():
    ...add operations to the graph...
    # Create a Supervisor that uses log directory on a shared file system.
    # Indicate if you are the 'chief'
    sv = Supervisor(logdir='/shared_directory/...', is_chief=is_chief)
    # Get a Session in a TensorFlow server on the cluster.
    with sv.managed_session(server.target) as sess:
      # Use the session to train the graph.
      while not sv.should_stop():
        sess.run(<my_train_op>)
```

Use Cases: Asynchronous Training

Use Cases: Synchronous Training

tf.train.SyncReplicasOptimizer

This optimizer avoids stale gradients by collecting gradients from all replicas, summing them, then applying them to the variables in one shot, after which replicas can fetch the new variables and continue.

tf.train.replica_device_setter

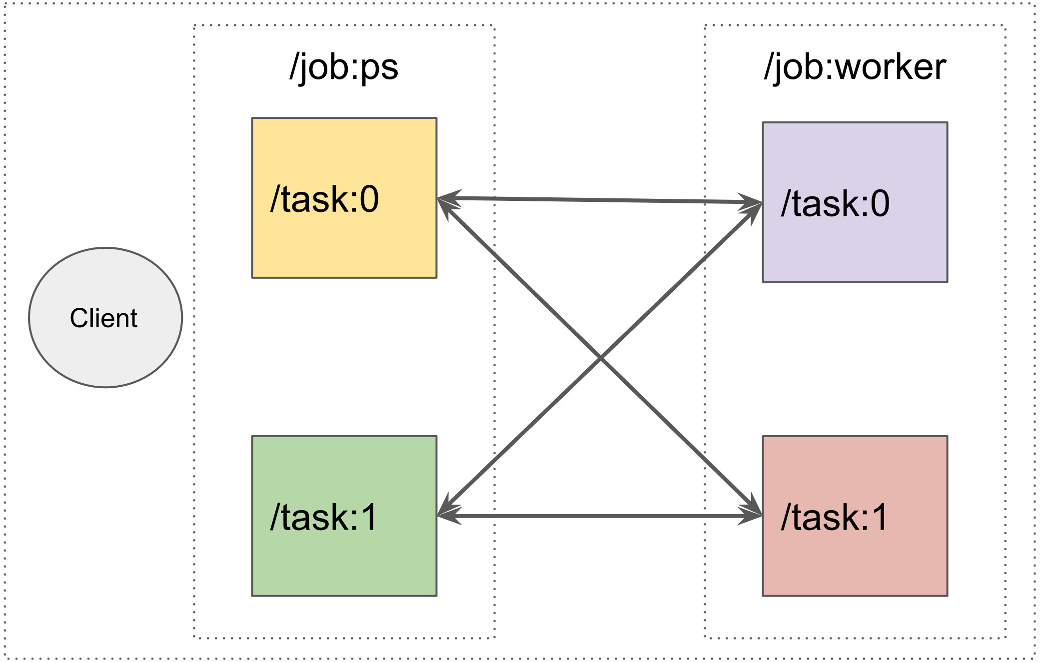
The device setter will automatically place Variables ops on separate parameter servers (ps). The non-Variable ops will be placed on the workers.

```
tf.train.replica_device_setter(cluster_def)
with tf.device(device_setter):
    pass
```

Use Cases: Training

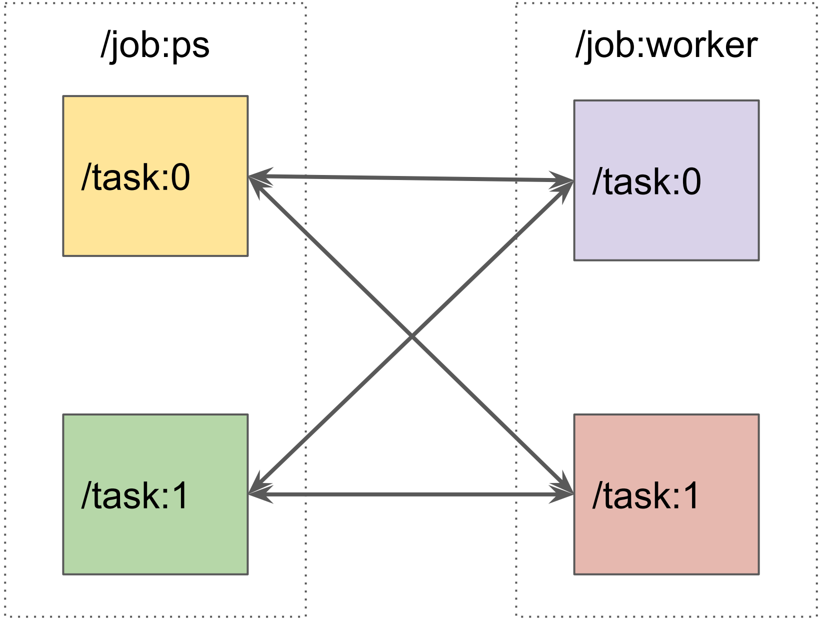
Training \ Replication	**In Graph**	**Between Graphs**
Asynchronous		
Synchronous		

In-Graph Replication




```
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:0"):
  input, labels = ...
  layer_1 = tf.nn.relu(...)

with tf.device("/job:worker/task:0"):
  train_op = ...
 logits = tf.nn.relu(...)

with tf.Session() as sess:
  for _ in range(10000):
    sess.run(train_op)
```

Replication Between Graph

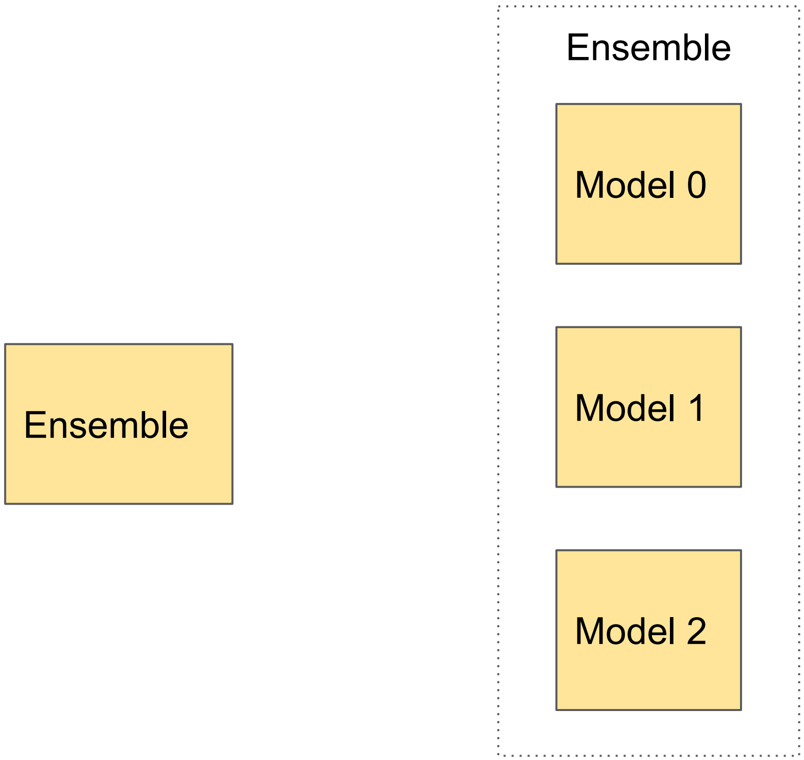



```python
# To build a cluster with two ps jobs on hosts ps0 and ps1, and 3 worker
# jobs on hosts worker0, worker1 and worker2.
cluster_spec = {
    "ps": ["ps0:2222", "ps1:2222"],
    "worker": ["worker0:2222", "worker1:2222", "worker2:2222"]}
with tf.device(tf.replica_device_setter(cluster=cluster_spec)):
  # Build your graph
  v1 = tf.Variable(...)  # assigned to /job:ps/task:0
  v2 = tf.Variable(...)  # assigned to /job:ps/task:1
  v3 = tf.Variable(...)  # assigned to /job:ps/task:0
# Run compute
...
```

Use Cases: Train HyperParameter Optimization

- Grid Search
- Random Search
- Gradient Optimization
- Bayesian

Ensembling



Comparison with other frameworks

mxnet

Surprise Demo : Rasberry Pi Cluster

Distributed TensorFlow Cluster

- Distributed TensorFlow
- TensorBoard Visualizations

Workshop Demo - LARGE GCE Cluster

Super-Large Cluster

Attendee GCE (Google Cloud Engine) Spec

- 50 GB RAM, 100 GB SSD, 8 CPUs (No GPUs)

TODO:

- Build Script for Each Attendee to Run as Either a Worker or Parameter Server
- Figure out split between Worker and Parameter Server

ImageNet

- TODO: Train full 64-bit precision, quantitize down to 8-bit (Sam A)

Workshop Demo - Initial Cluster

8.34.215.90 (Sam)

130.211.128.240 (Fabrizio)

104.197.159.134 (Fregly)

Cluster Spec: “8.34.215.90, 130.211.128.240, 104.197.159.134”

SSH PEM file: http://advancedspark.com/keys/pipeline-training-gce.pem

chmod 600 pipeline-training-gce.pem

Username: pipeline-training

Password: password9

sudo docker images (VERIFY fluxcapacitor/pipeline)

START: sudo docker run -it --privileged --name pipeline --net=host -m 48g fluxcapacitor/pipeline bash

TensorFlow Operations: Hidden Gems

- Go over macros, functions provided in the library, but not mentioned in documentation
- Brief brief overview of writing custom op
- Testing GPU code
- Leveraging Eigen and existing code
- Python wrapper