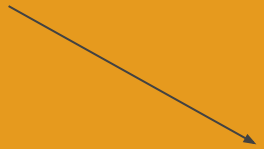
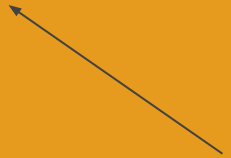
TensorFlow Tutorial

Yuan Tang
terrytangyuan@gmail.com

Multi-dimensional array of data



TensorFlow



Graph of operations

Agenda

- Motivation
- Architecture Overview
- Basic Mechanics
- High-level APIs
- Distributed TensorFlow
- Take-home Exercises

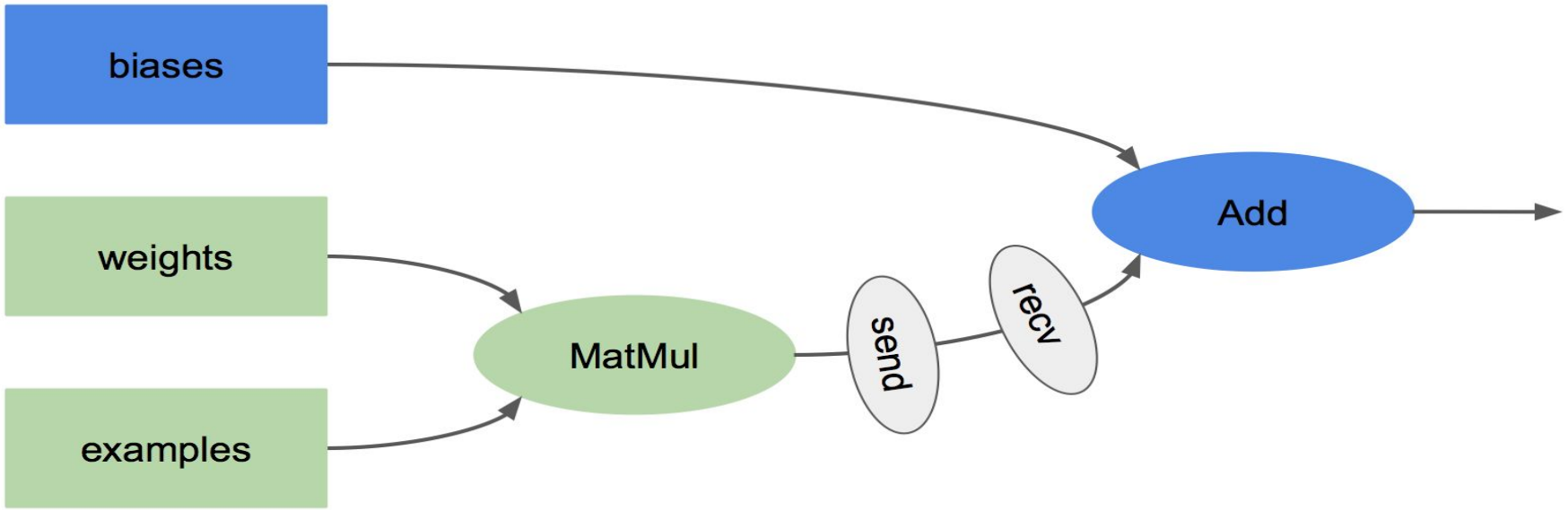
Why TensorFlow?

- General computational platform
- Hardware accelerated and distributed
- Mobile and Embedded
- Research friendly and production ready

Where is TensorFlow being used?

- Gmail
- Google Translate
- Youtube WatchNext
- DeepMind Research

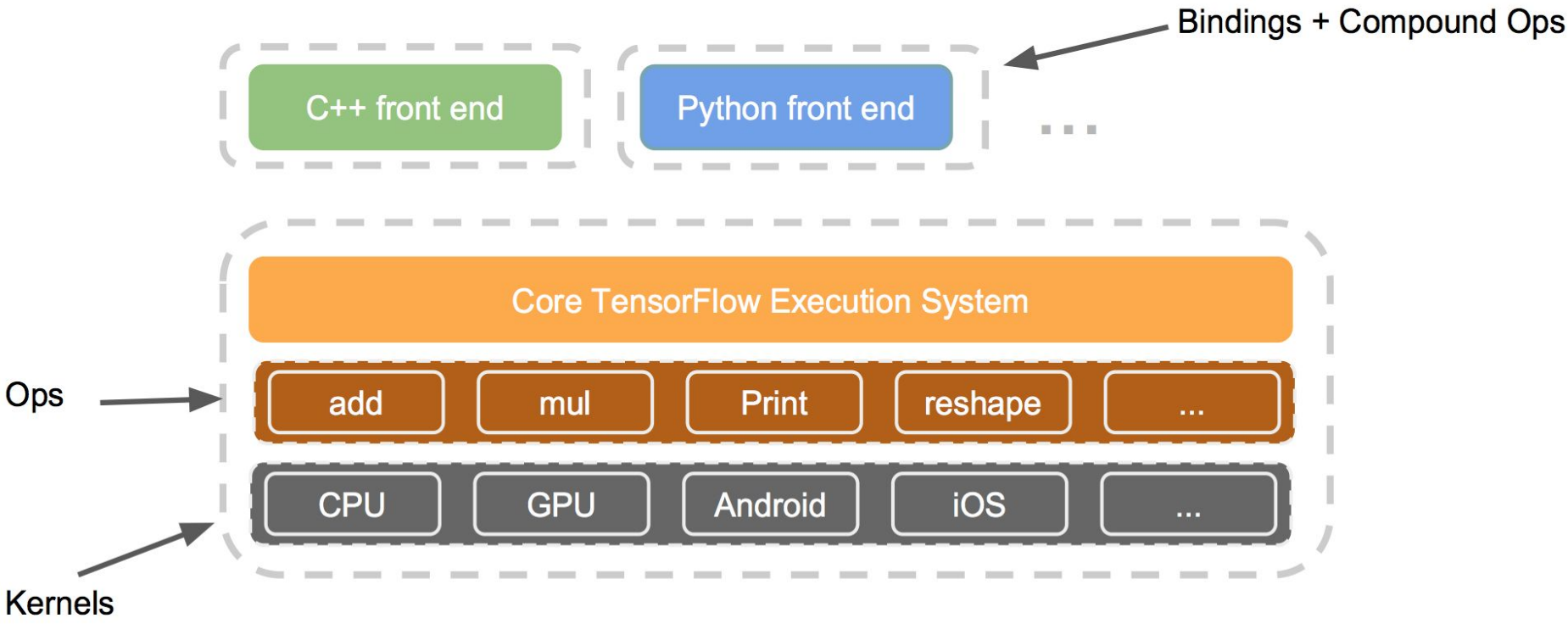
TensorFlow Components



TensorFlow Components - Ops

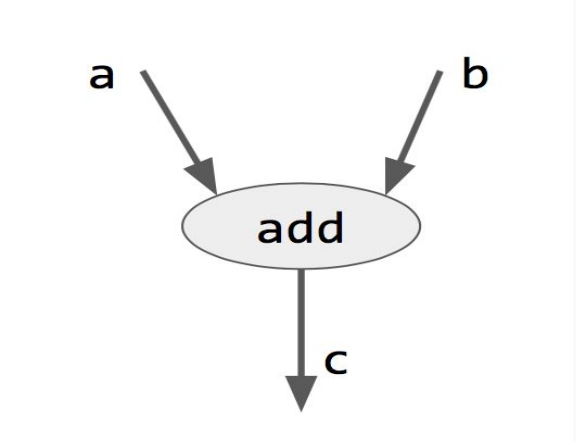
- Element-wise Math Ops - Add, Sub, Exp
- Array Ops - Slice, Shape, Split, Shuffle
- Matrix Ops - MatMul, MatrixDeterminant
- Stateful Ops - Variable, AssignAdd

TensorFlow Architecture



TensorFlow 101

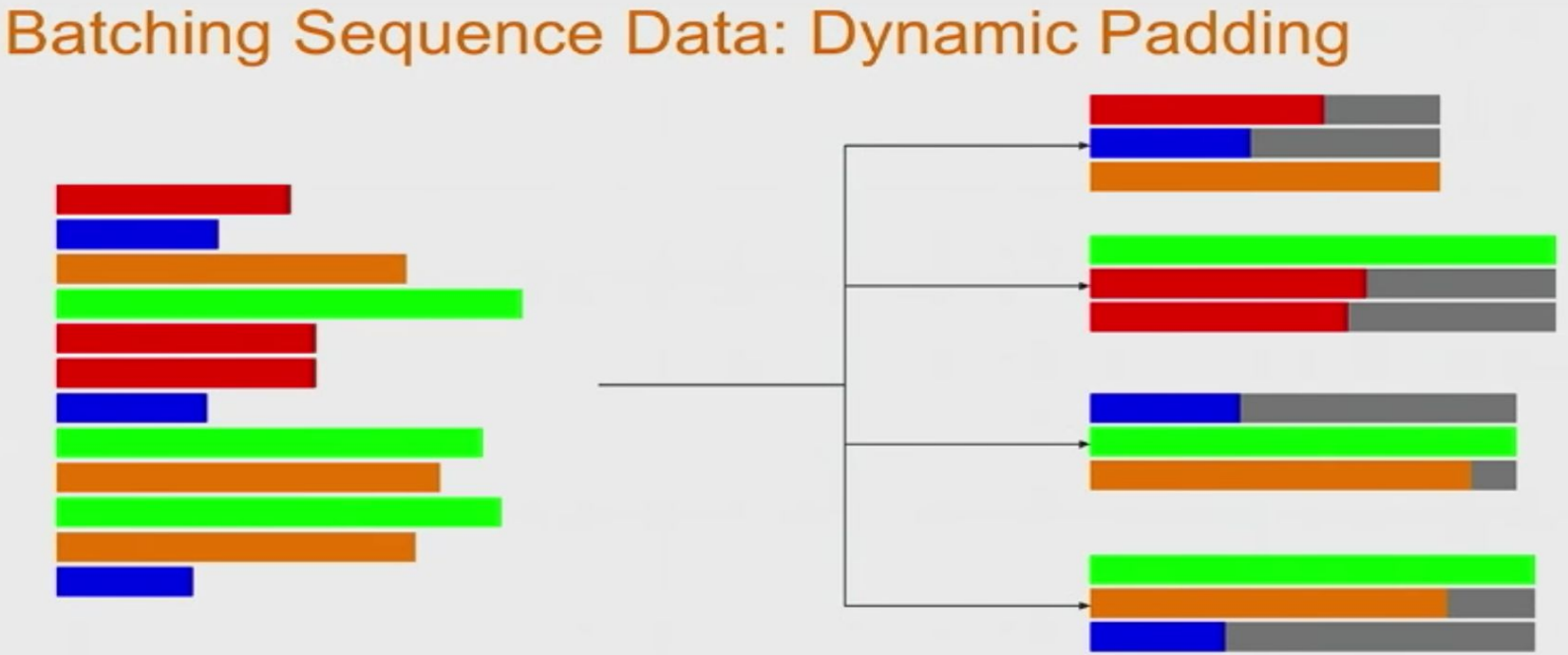
```
>> sess = tf.Session()
>> a = tf.constant(2)
>> b = tf.constant(8)
>> sess.run(a + b)
10

>> c = tf.add(a, b)
>> c.eval(sess)
10
```



Shape Inference 101

```
>> a = tf.constant([1, 2, 3, 4])
>> a
<tf.Tensor 'Const_1:0' shape=(4,) dtype=int32>

>> b = tf.tile(a, [2])
>> b
<tf.Tensor 'Const_1:0' shape=(8,) dtype=int32>

>> session.run(b)
array([1, 2, 3, 4, 1, 2, 3, 4], dtype=int32)
```

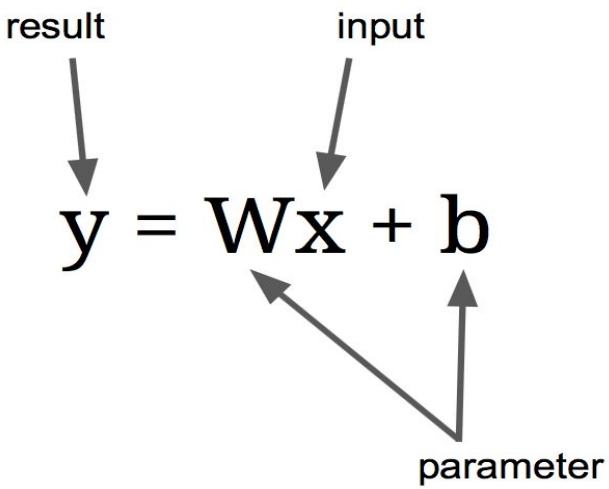
Why is shape inference important?

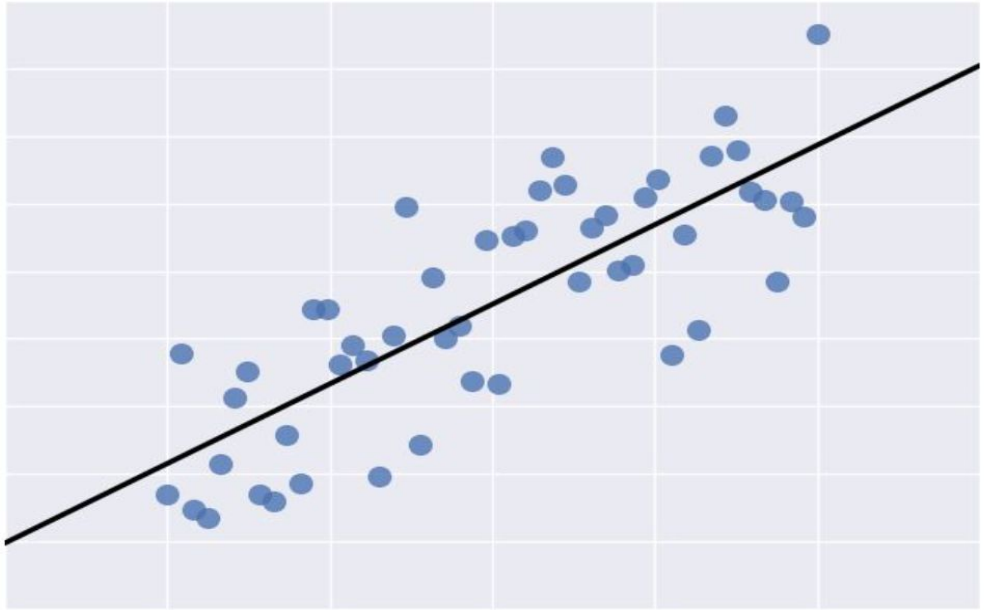


Writing Compound Ops

```
def correct_prediction_op(preds, labels):
    return tf.equal(tf.argmax(preds, 1), tf.argmax(labels, 1))

def calc_accuracy(correct_predictions):
    return tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
```

Linear Regression





Linear Regression

```
import tensorflow as tf
x = tf.placeholder(shape=[2,1], dtype=tf.float32, name="x")
W = tf.get_variable(shape=[1,2], name="W")
b = tf.get_variable(shape=[1], name="b")
y = tf.matmul(W, x) + b
x_in = [[3], [4]]

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

Linear Regression - Loss Function

Given x, y_{label}, compute a loss, for instance:

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

Linear Regression - Loss Function

```
y_label = tf.placeholder(shape=[1,1],dtype=tf.float32,
                         name="y_label")

diff = y - y_label
L = tf.reduce_sum(diff * diff)
```

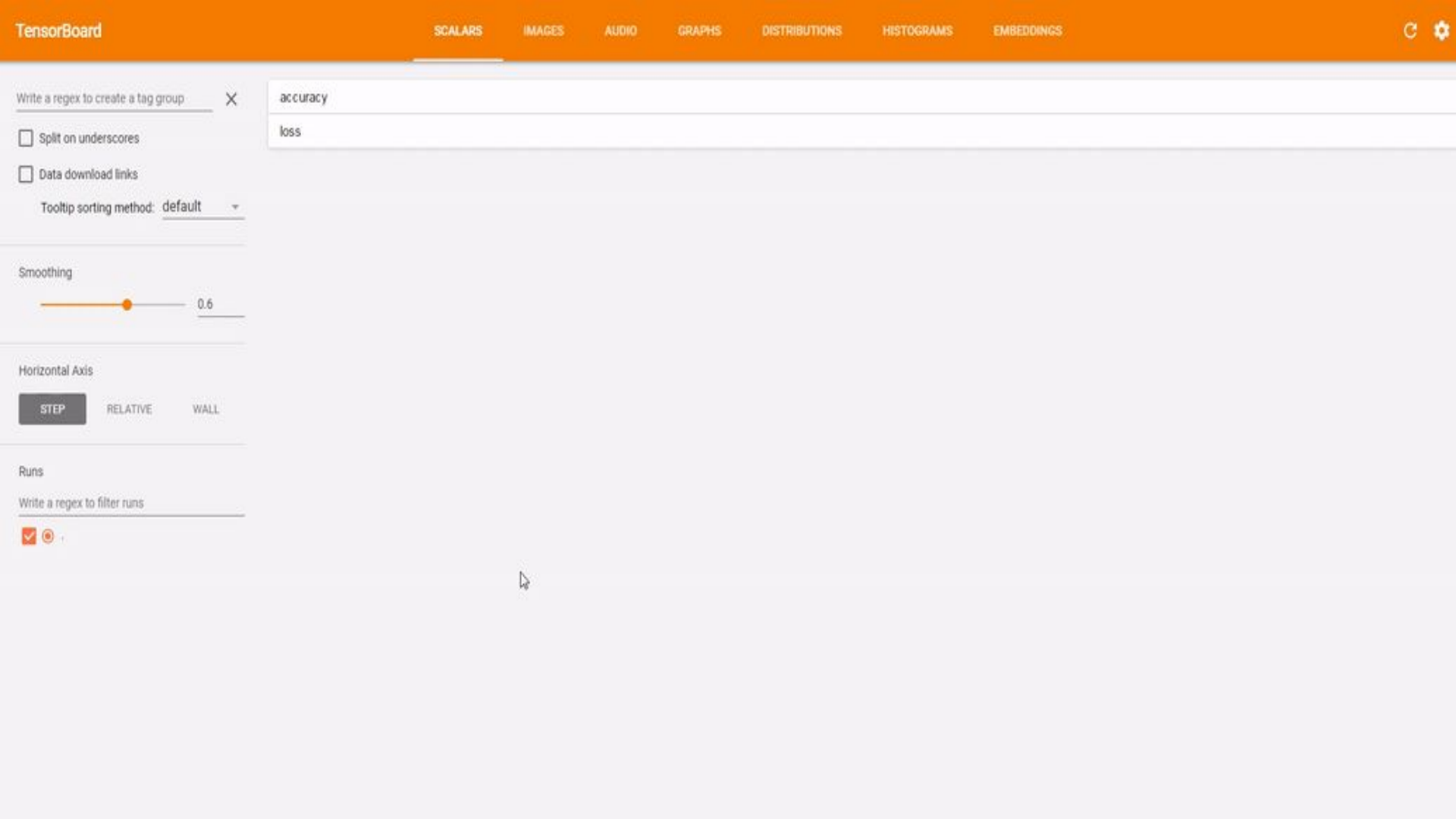
Linear Regression - Training

```
train_op = tf.train.GradientDescentOptimizer(learning_rate=0.01)
                   .minimize(L)
data = load_csv("training_data.csv", delimiter=",")
for x1, x2, y_in in data:
  sess.run(train_op, feed_dict={x: [[x1],[x2]], y_label: y_in})
```

Linear Regression - Evaluation

```
eval_data = load_csv("evaluation_data.csv", delimiter=",")
acc = 0.
for x1, x2, y_in in eval_data:
  acc += sess.run(L, feed_dict={x: [[x1],[x2]], y_label: y_in})
print acc / len(eval_data)
```

Visualization


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
Write a regex to create a tag group
Split on underscores
Data download links
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
accuracy
loss

TensorBoard

SCALARS IMAGES AUDIO GRAPHS

Write a regex to create a tag group

Split on underscores

Data download links

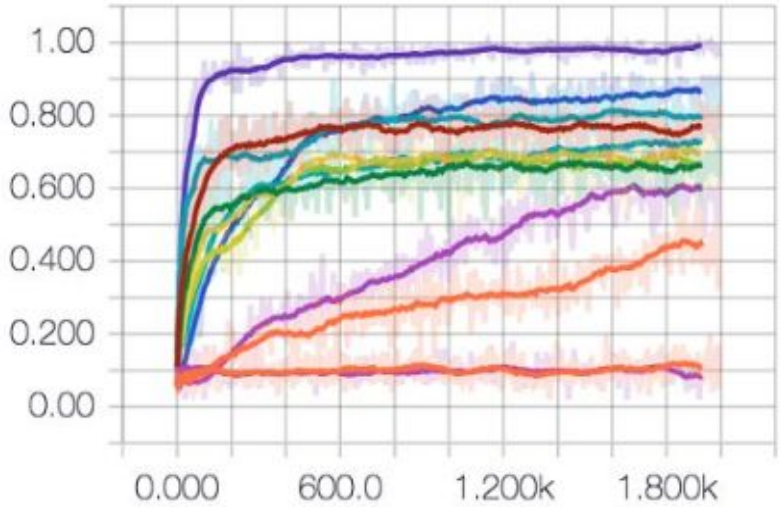
Tooltip sorting method: default

Smoothing

0.6

accuracy

accuracy/accuracy



TensorBoard

EVENTS IMAGES GRAPH HISTOGRAMS

Fit to screen

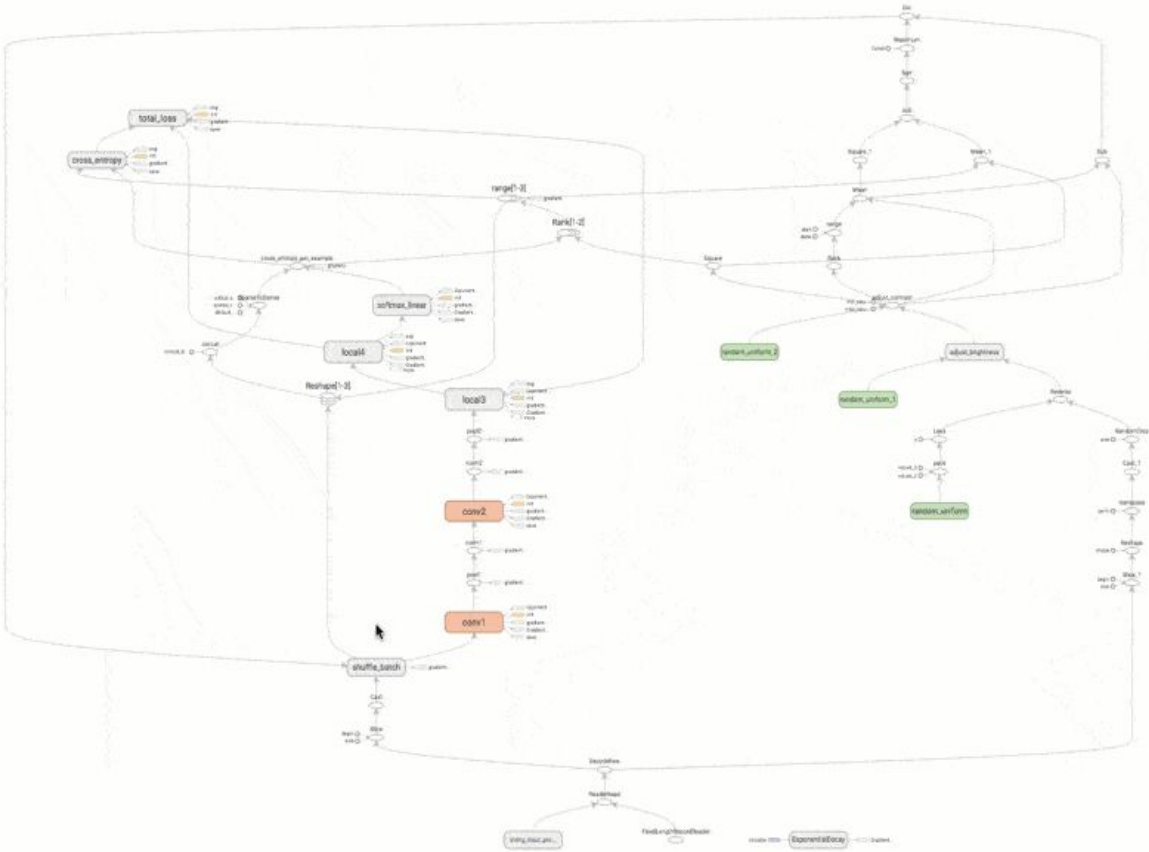
Run cifar-train

Upload Choose File

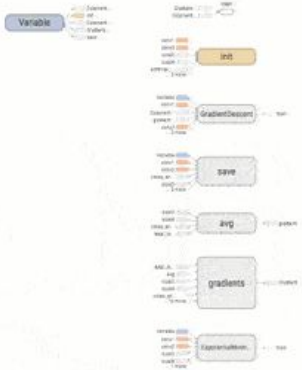
Color Structure

color: same substructure
gray: unique substructure

Main Graph



Auxiliary nodes



Graph (* = expandable)

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

Fit to screen

Download PNG

Run (1) train

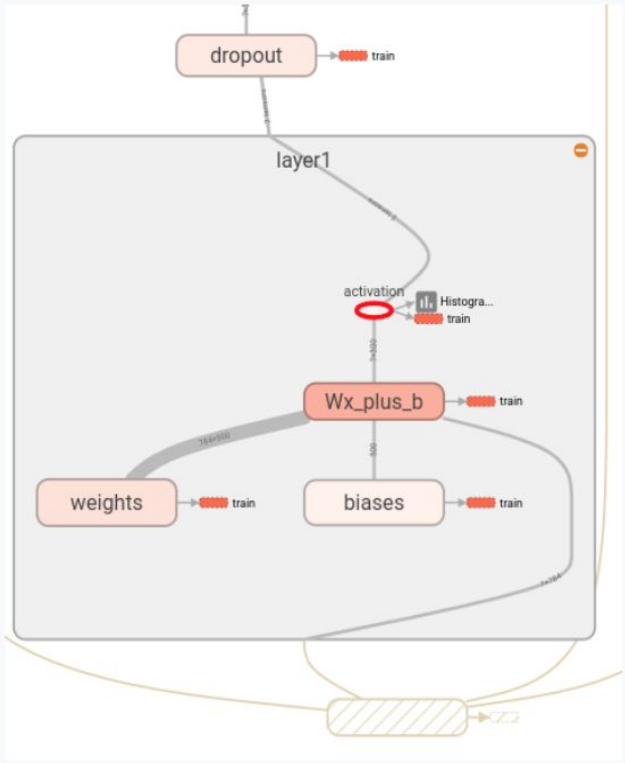
Session runs (10) step999

Upload Choose File

Color
- Structure
- Device
- Compute time
- Memory

0 µs 9.33 ms

unused substructure



Attributes (1)

T	{"type":"DT_FLOAT"}
Device	/job:localhost/replica:0/task:0/cpu:0

Inputs (1)

layer1/Wx_plus_b/add ?×500

Outputs (6)

- dropout/dropout/Shape ?×500
- dropout/dropout/mul ?×500
- train/gradients/dropout/dropout/mu ?×500
- train/gradients/dropout/dropout/mu ?×500
- train/gradients/layer1/activation_gr ?×500
- layer1/HistogramSummary ?×500

Node Stats

Memory	195 KB
Compute Time	93 µs
Tensor Output Sizes	[100, 500]

Remove from main graph

TensorBoard 101

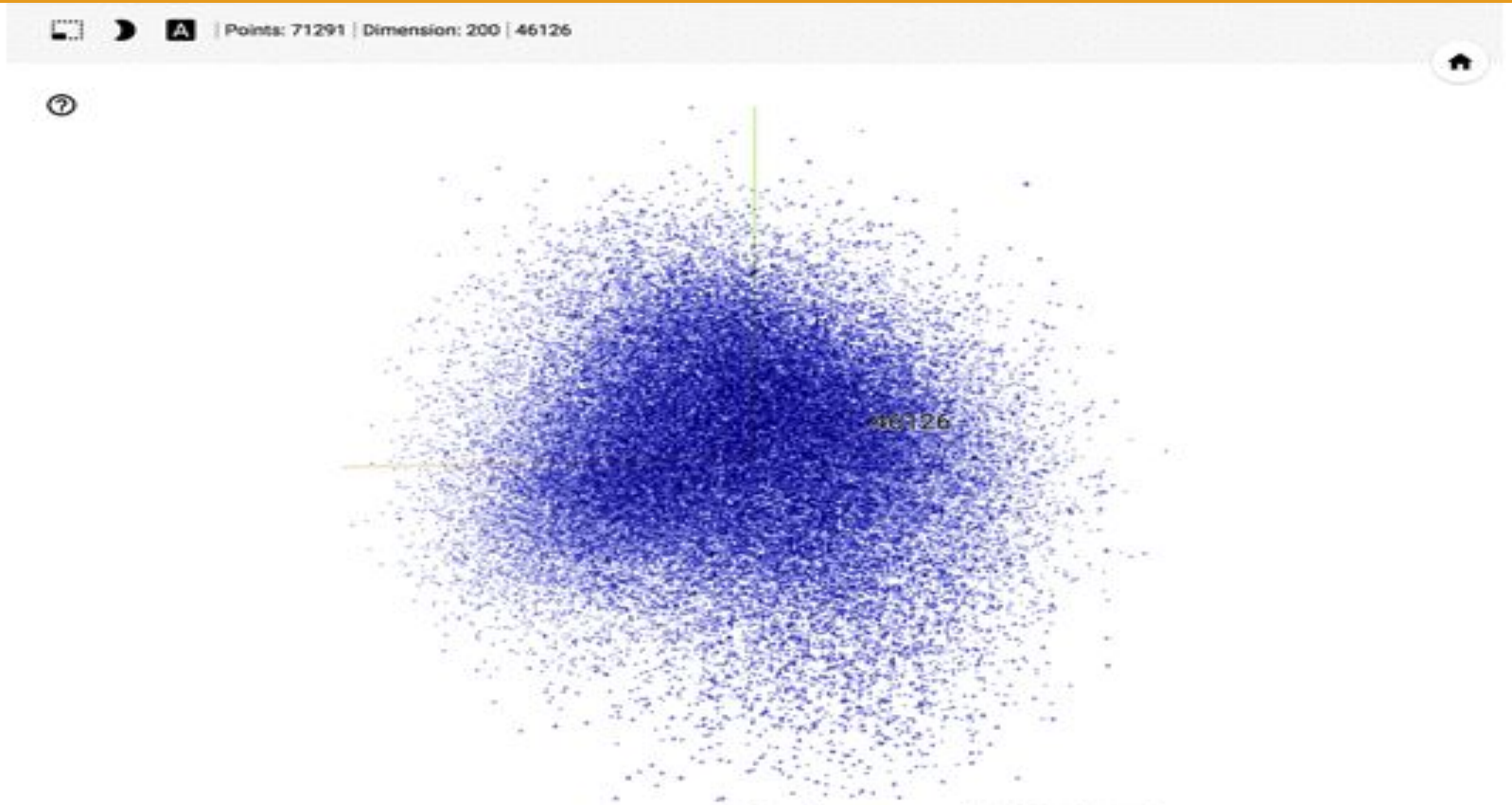
tf.summary.FileWriter

- Writes summary protocol buffers to event files
- Updates file contents asynchronously
- Keeps the training program efficient

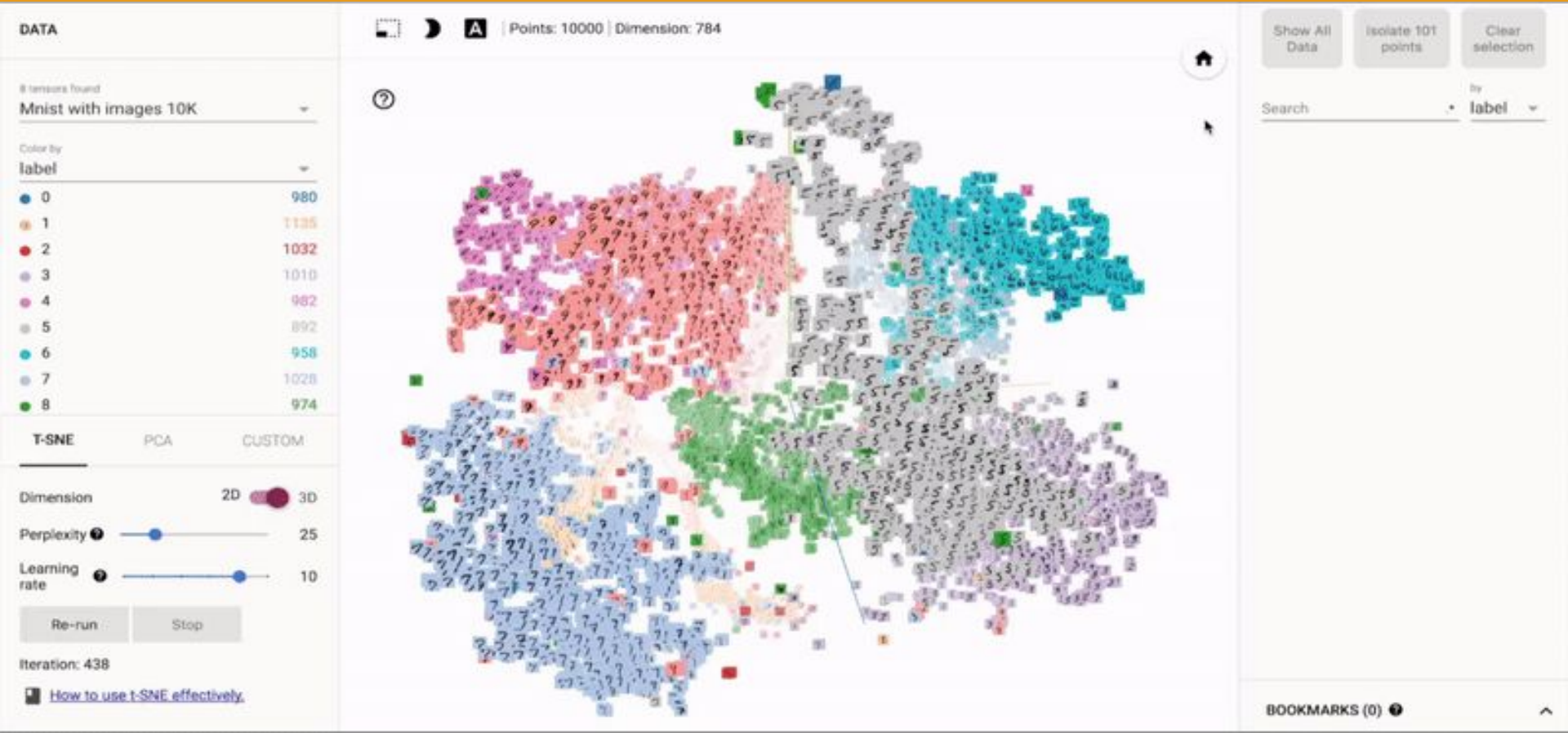
TensorBoard 101

```
stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
tf.summary.scalar('stddev', stddev)

tf.summary.scalar()
tf.summary.histogram()
tf.summary.audio()
tf.summary.image()

Tensorboard --logdir=path/to/log-directory
```

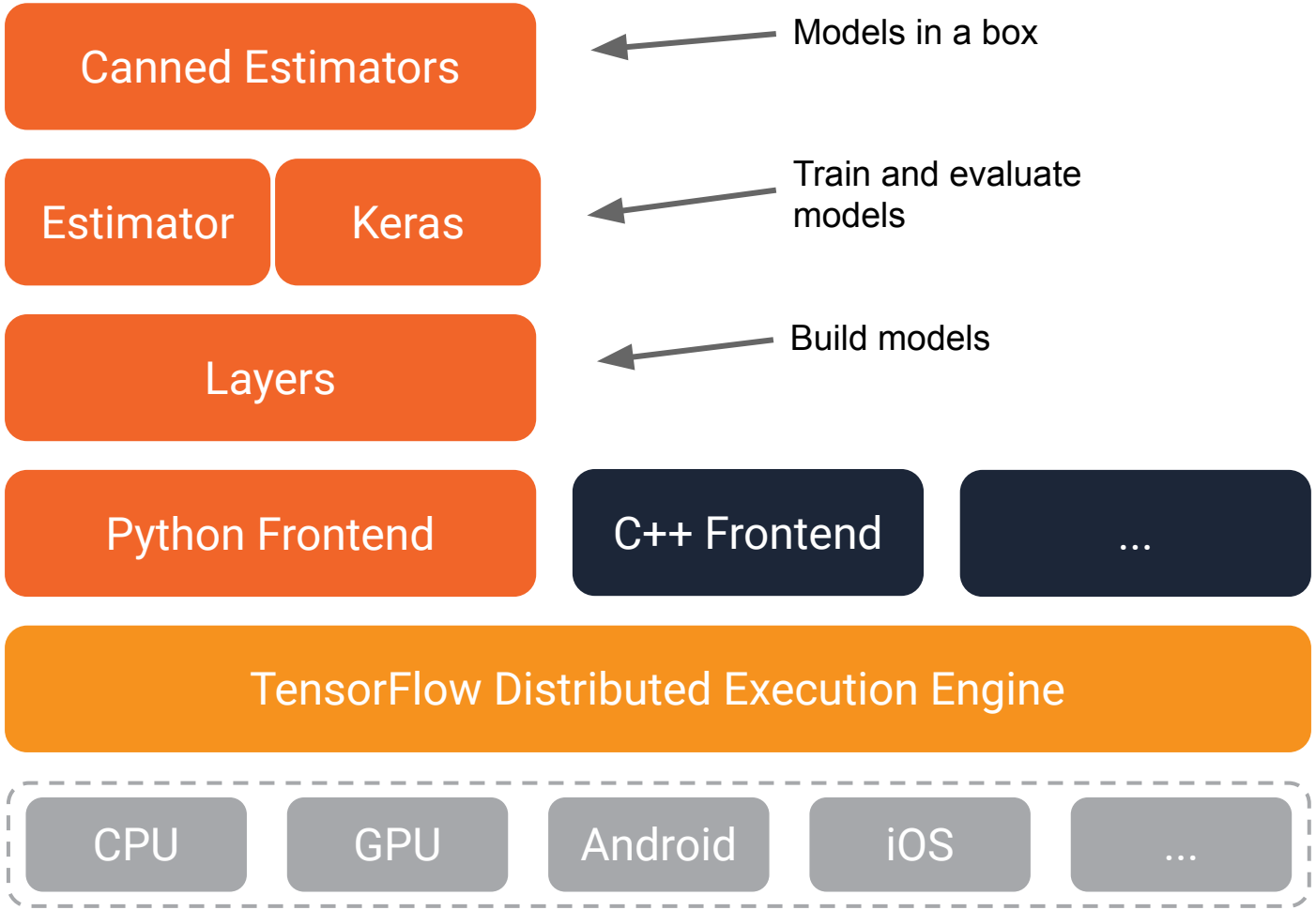
Visualization - Embedding Projector (link)



Visualization - Embedding Projector (link)



High-level APIs


Canned Estimators
Models in a box
Estimator
Keras
Train and evaluate models
Layers
Build models
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...

Keras

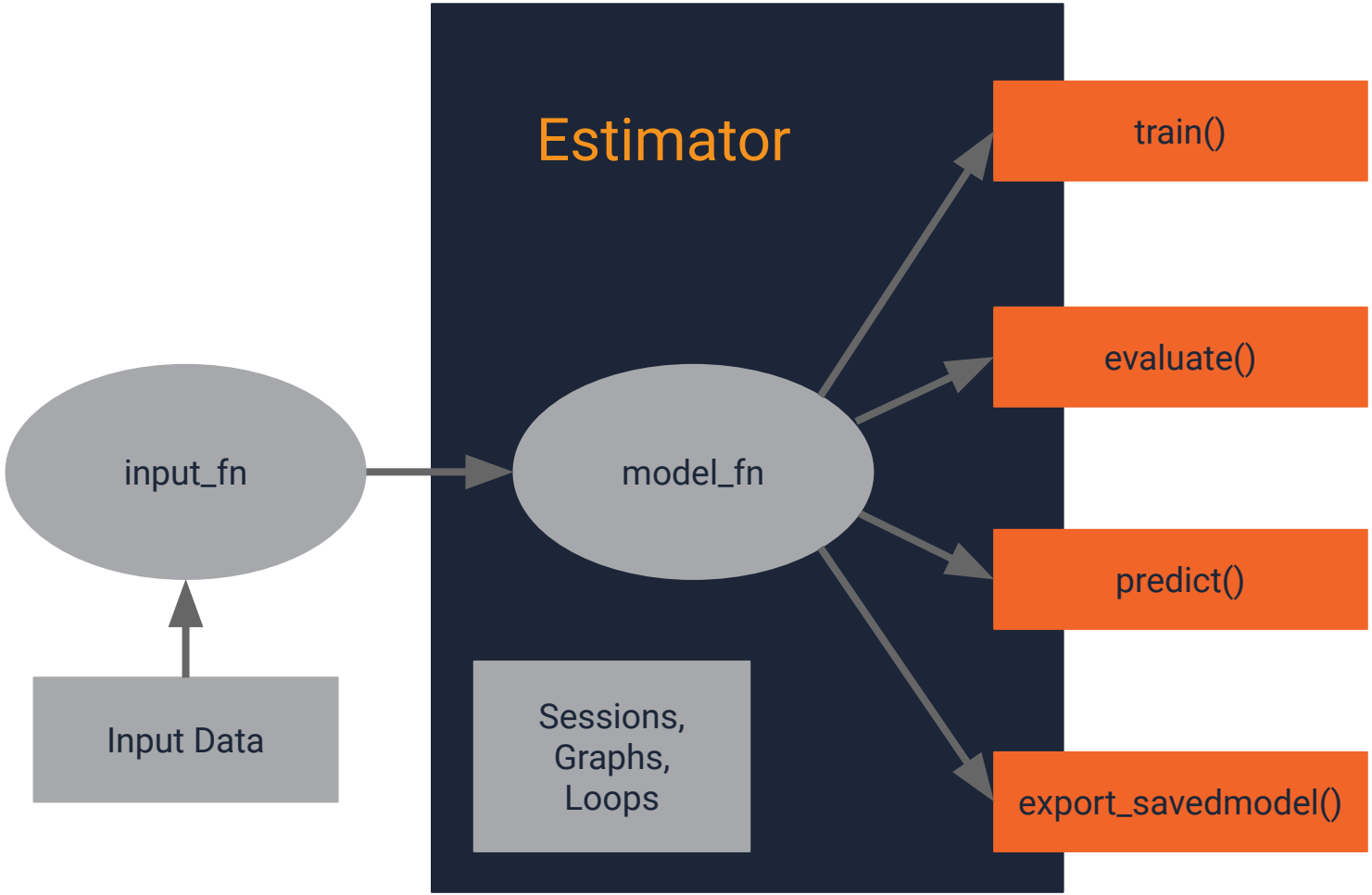
- Support multiple backends
  - Theano
  - TensorFlow
  - CNTK
- Focus on fast prototyping
- Used heavily in competitions and research

Keras - Example

```
from keras.layers import Dense, Activation

model.add(Dense(output_dim=64, input_dim=100))
mode.add(Activation('relu'))
model.add(Dense(output_dim=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

model.fit(x_train, y_train, nb_epoch=5, batch_size=32)
classes = model.predict_classes(x_test, batch_size=32)
probs = model.predict_proba(x_test, batch_size=32)
```

TensorFlow Estimators

- Easy transition for Scikit-learn users
- Hides all “grundy” parts of TensorFlow
- Avoids boilerplate code
- Transitions into learning TensorFlow low-level APIs
- Distributed training and evaluation


Estimator
train()
evaluate()
predict()
export_savedmodel()
input_fn
model_fn
Input Data
Sessions,
Graphs,
Loops

Feature Columns

```
age = numeric_column('age')
occupation = tf.feature_column.categorical_column_with_hash_bucket(
    'occupation', hash_bucket_size=1000)
age_buckets = tf.feature_column.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

Input Function

```
def input_fn(data_file, num_epochs, shuffle):
  """Input builder function."""
  df_data = pd.read_csv(
      tf.gfile.Open(data_file),
      names=['age', 'occupation', 'income_bracket'],
      skipinitialspace=True,
      engine="python",
      skiprows=1)
  # remove NaN elements
  df_data = df_data.dropna(how="any", axis=0)
  labels = df_data["income_bracket"].apply(lambda x: ">50K" in x).astype(int)
  return tf.estimator.inputs.pandas_input_fn(
      x=df_data,
      y=labels,
      batch_size=100,
      num_epochs=num_epochs,
      shuffle=shuffle,
      num_threads=5)
```

Canned Estimator

Pre-made Estimators encode best practices, providing the following benefits:

- Best practices for determining where different parts of the computational graph should run, implementing strategies on a single machine or on a cluster.
- Best practices for event (summary) writing and universally useful summaries.

Canned Estimator - Example

```
classifier = tf.estimator.DNNClassifier(
  feature_columns=[age, occupation, age_buckets],
  hidden_units=[10, 10])

classifier.train(
  input_fn=input_fn("data.csv", 20, True),
  steps=20)
classifier.evaluate(input_fn=..., steps=20)
classifier.predict(input_fn=...)
classifier.export_savedmodel(...)
```

Canned Estimator

Existing:

- Linear Estimators
- DNN Estimators

On the roadmap:

- Dynamic RNN & State Saving RNN Estimators
- Time Series Estimators
- Clustering
- Generative models
- Tree-based models

Custom Estimator - Example

Link to code

Building Blocks

- Layers
- Optimizers
- Losses
- Metrics
- SessionRunHook
- Experiment

Modelling Techniques

- Early Stopping
- Custom Learning Rate Decay
- Custom Class Weights
- Dropout
- Batch Normalization
- Clip Gradients

Datasets API

- Set of transformations that compose together to build complex data pipelines
- Create and manipulate different datasets and iterators in the same program to parameterize the behavior

Datasets API - Dataset Interface

```
# Read records from a list of files.
dataset = TFRecordDataset(["file1.tfrecord", "file2.tfrecord", …])
# Parse string values into tensors.
dataset = dataset.map(lambda record: tf.parse_single_example(record, …))
# Randomly shuffle using a buffer of 10000 examples.
dataset = dataset.shuffle(10000)
# Repeat for 100 epochs.
dataset = dataset.repeat(100)
# Combine 128 consecutive elements into a batch.
dataset = dataset.batch(128)
```

Datasets API - Iterator Interface

```
dataset = …
# A one-shot iterator automatically initializes itself on first use.
iterator = dataset.make_one_shot_iterator()
# The return value of get_next() matches the dataset element type.
images, labels = iterator.get_next()
train_op = model_and_optimizer(images, labels)
# Loop until all elements have been consumed.
try:
  while True:
    sess.run(train_op)
except tf.errors.OutOfRangeError:
  pass
```

Datasets API - Integration with input_fn

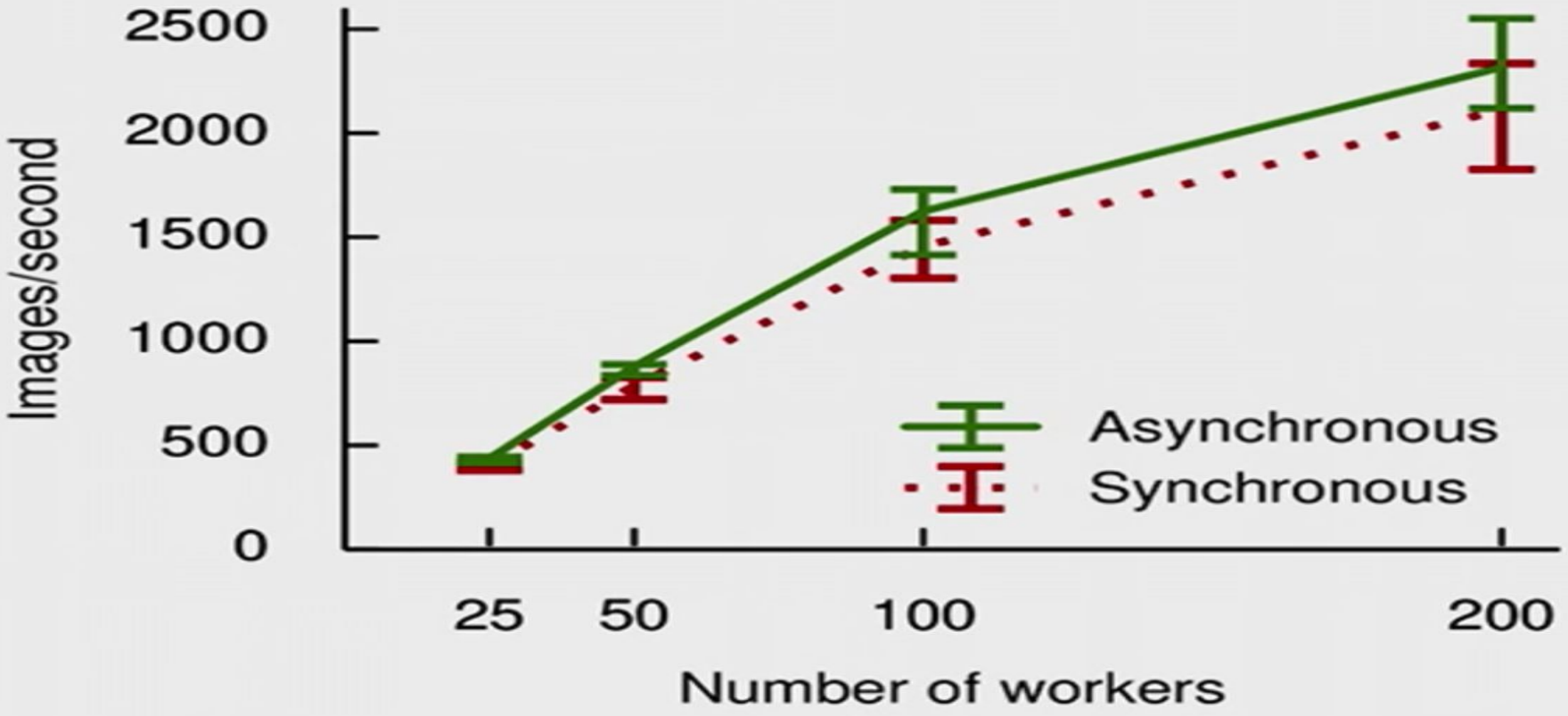
```
def input_fn():
  dataset = …
  # A one-shot iterator automatically initializes itself on first use.
  iterator = dataset.make_one_shot_iterator()
  # The return value of get_next() matches the dataset element type.
  images, labels = iterator.get_next()
  return images, labels
# The input_fn can be used as a regular Estimator input function.
estimator = tf.estimator.Estimator(…)
estimator.train(train_input_fn=input_fn, …)
```

Datasets API - Performance Tuning

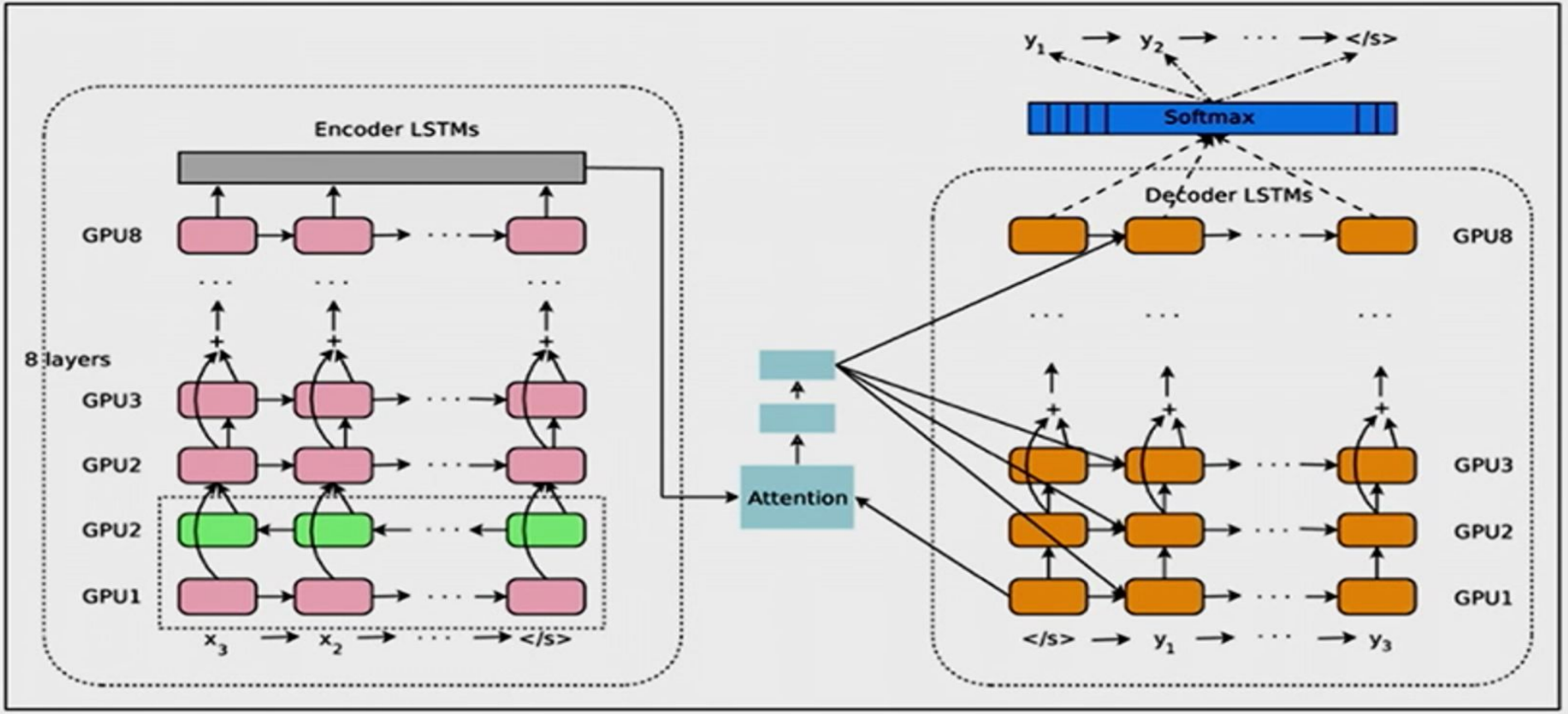
```
# Use interleave() and prefetch() to read many files concurrently.
files = Dataset.list_files("*.tfrecord")
dataset = files.interleave(lambda x: TFRecordDataset(x).prefetch(100),
                           cycle_length=8)
# Use num_parallel_calls to parallelize map().
dataset = dataset.map(lambda record: tf.parse_single_example(record, …),
                      num_parallel_calls=64)
```

Distributed TensorFlow

Data Parallelism

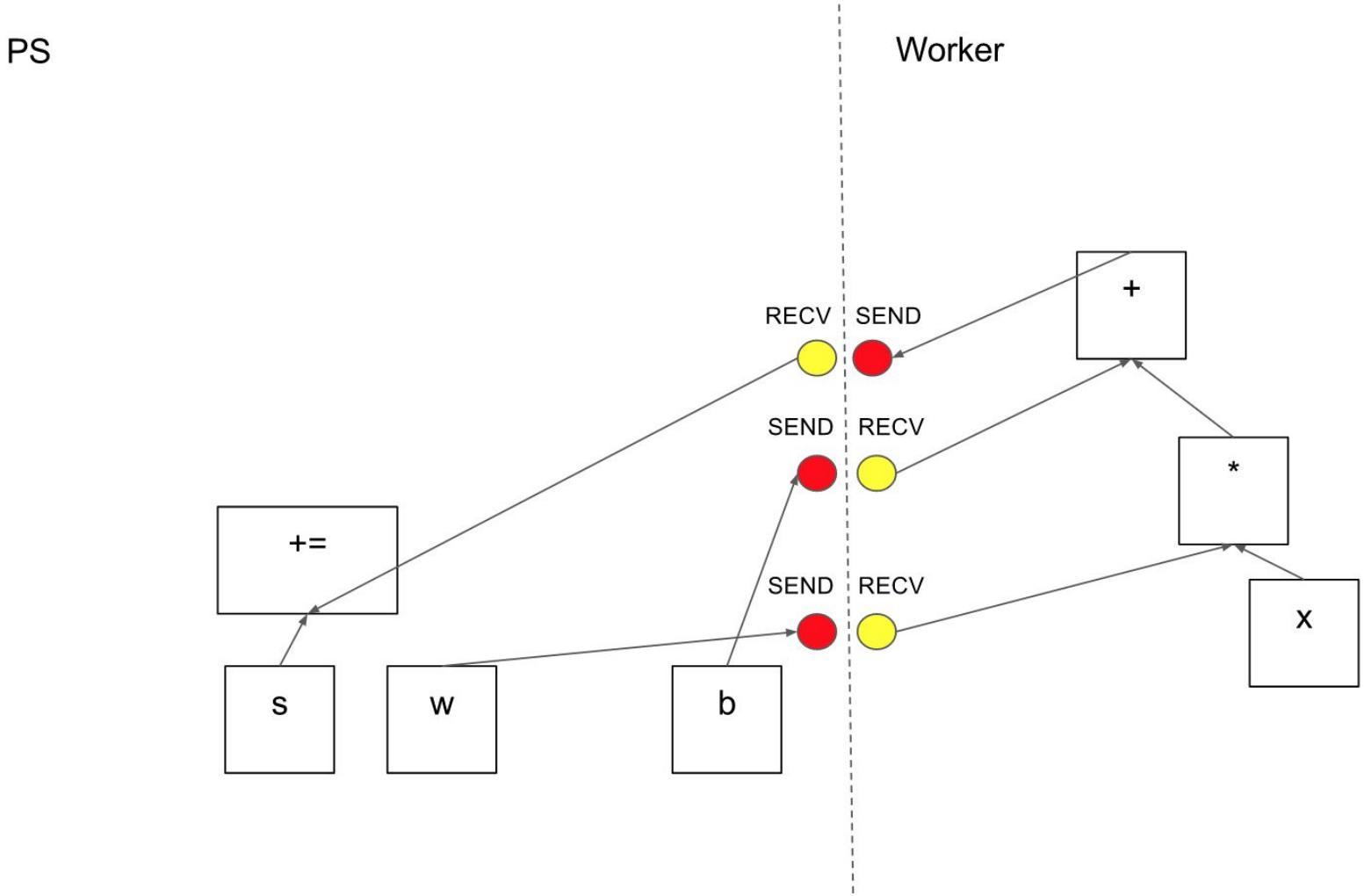


Model Parallelism



Types of Tasks

- Parameter server tasks
  - Variables
  - Update Ops
- Worker tasks
  - Pre-processing
  - Loss calculation
  - Backpropogation


PS
Worker
RECV
SEND
SEND
RECV
SEND
RECV
+
*
x
+=
s
w
b

Device Replacement

```
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
server = tf.train.Server(cluster,
                         job_name=FLAGS.job_name,
                         task_index=FLAGS.task_index)
if FLAGS.job_name == "ps":
    server.join() # listens requests
elif FLAGS.job_name == "worker":
    with tf.device(tf.train.replica_device_setter(
        worker_device="/job:worker/task:%d" % FLAGS.task_index,
        cluster=cluster)):
        # Build the model...
```

In-graph Replication (single tf.Graph)

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

Between-graph Replication (multiple clients)

```
with tf.device("/job:ps/task:0/cpu:0"):
     W = tf.Variable(...)
     b = tf.Variable(...)
with tf.device("/job:worker/task:0/gpu:0"):
     result = tf.matmul(input, W) + b
     loss = fn(result)
```

```
with tf.device("/job:ps/task:0/cpu:0"):
     W = tf.Variable(...)
     b = tf.Variable(...)
with tf.device("/job:worker/task:1/gpu:0"):
     result = tf.matmul(input, W) + b
     loss = fn(result)
```

Variable Placement - Partitioned Variables

```
greedy = tf.contrib.training.GreedyLoadBalancingStrategy(...)
with tf.device(tf.train.replica_device_setter(
    ps_tasks=3, ps_strategy=greedy)):
    embedding = tf.get_variable(
        "embedding", [100000000, 20],
        partitioner=tf.fixed_size_partitioner(3))
```

Fault Tolerance

```
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):
    weights = tf.get_variable("weights", [784, 100])
    biases = tf.get_variable("biases", [100])
    ...

saver = tf.train.Saver(sharded=True)
with tf.Session(server.target) as sess:
    while True:
        ...
        if is_chief and step % 1000 == 0:
            saver.save(sess, "hdfs://...")
```

Fault Tolerance

```
server = tf.train.Server(...)
is_chief = FLAGS.task_index == 0

with tf.train.MonitoredTrainingSession(server.target, is_chief) as sess:
    while not sess.should_stop():
        sess.run(train_op)
```

Experiment and Estimator

```
def experiment_fn(config, params):
    features = [tf.layers.embedding_column(...),
                tf.layers.bucketized_column(...)]
    return Experiment(
        train_input_fn=..., eval_input_fn=...,
        estimator=DNNClassifier(
            hidden_units=[10, 20], feature_columns=features,
            config, params))

learn_runner.run(experiment_fn, config, ...)
```

References

- R Interface to TensorFlow
- TensorFlow Estimators KDD'17 Paper
- TensorFlow OSDI'16 Paper
- TensorFlow Datasets API
- Code Presented in The Slides
- My Personal Website

Ready for exercises?

- Link to exercises: https://github.com/terrytangyuan/tensorflow-basic-exercises

Thanks!