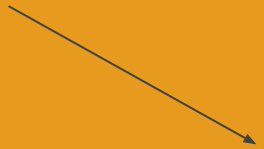
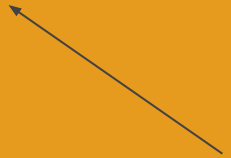
TensorFlow Tutorial

By Yuan Tang at Uptake
terry.tang@uptake.com

Multi-dimensional array of data



TensorFlow



Graph of operations

Agenda

- Motivation
- Overview of Architecture
- Basic Mechanics
- Exercises
- Distributed TensorFlow

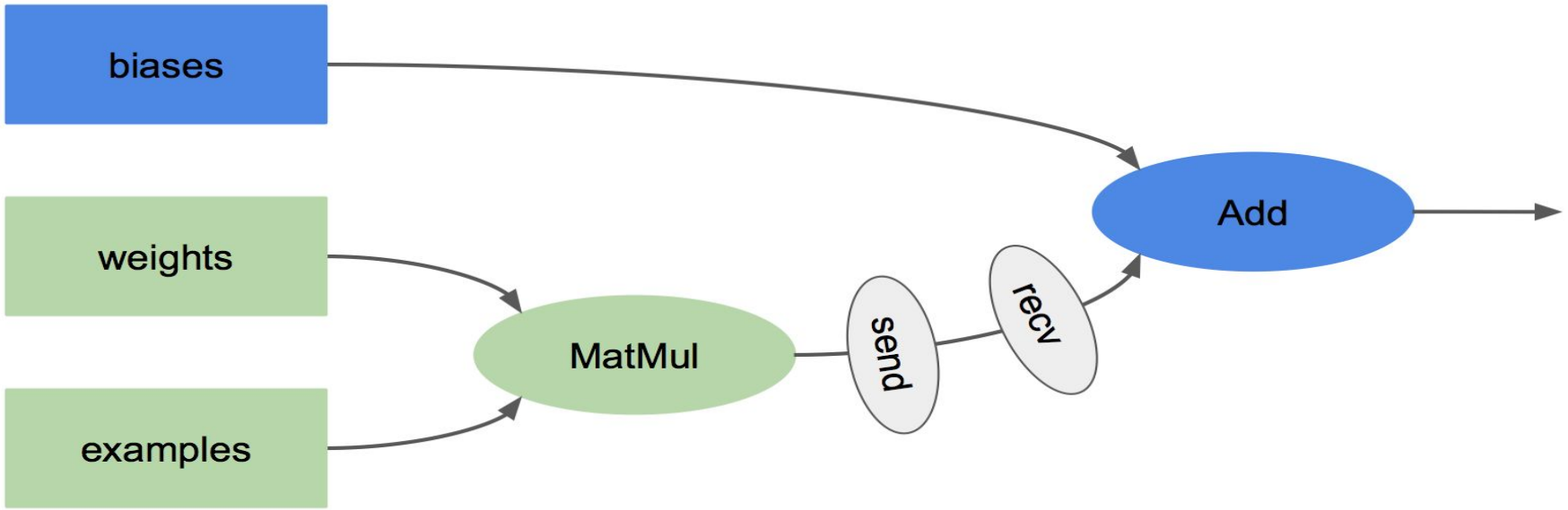
Why TensorFlow?

- General computational platform
- GPU optimized and distributed
- Mobile and Embedded
- Research friendly and production ready

Where is TensorFlow being used?

- Google Photos, Gmail
- Youtube WatchNext
- DeepMind, Stanford

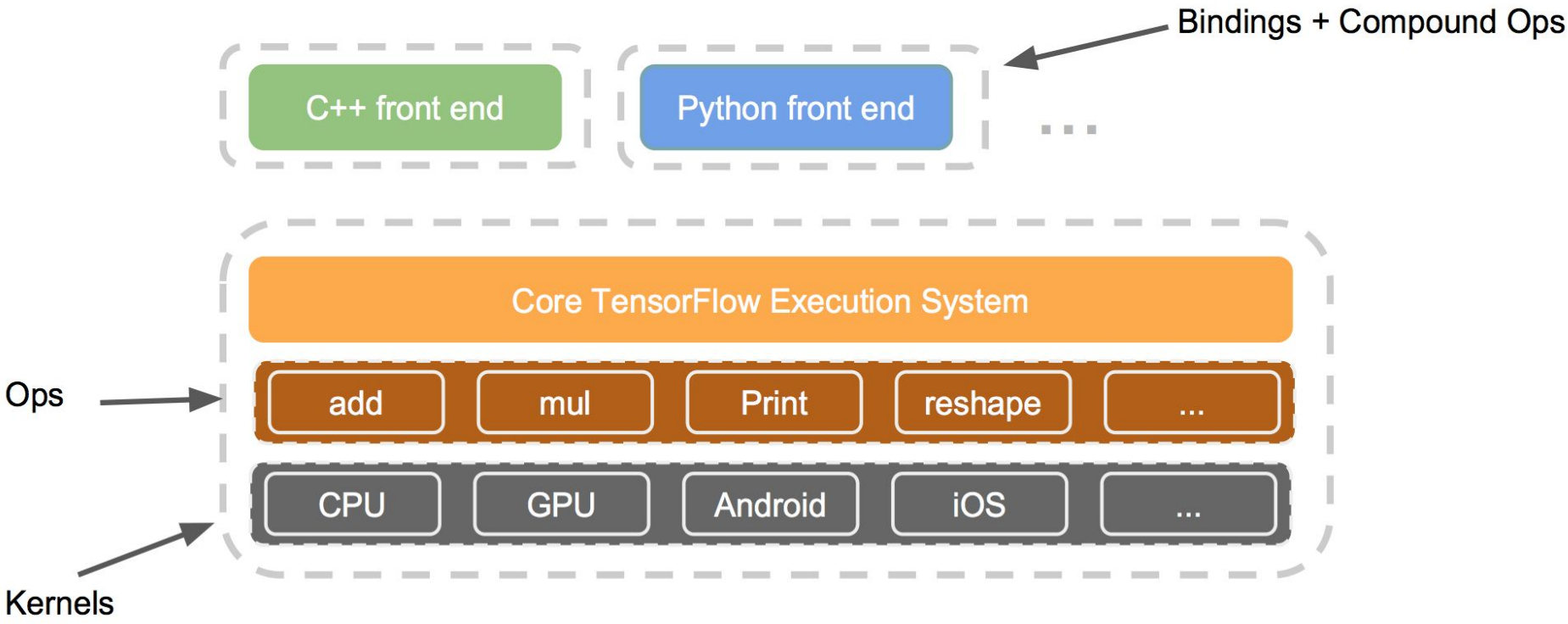
TensorFlow Components



TensorFlow Components - Ops

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

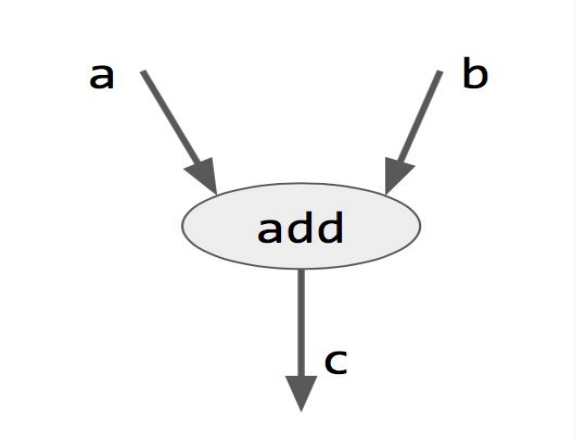
TensorFlow Architecture



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
```



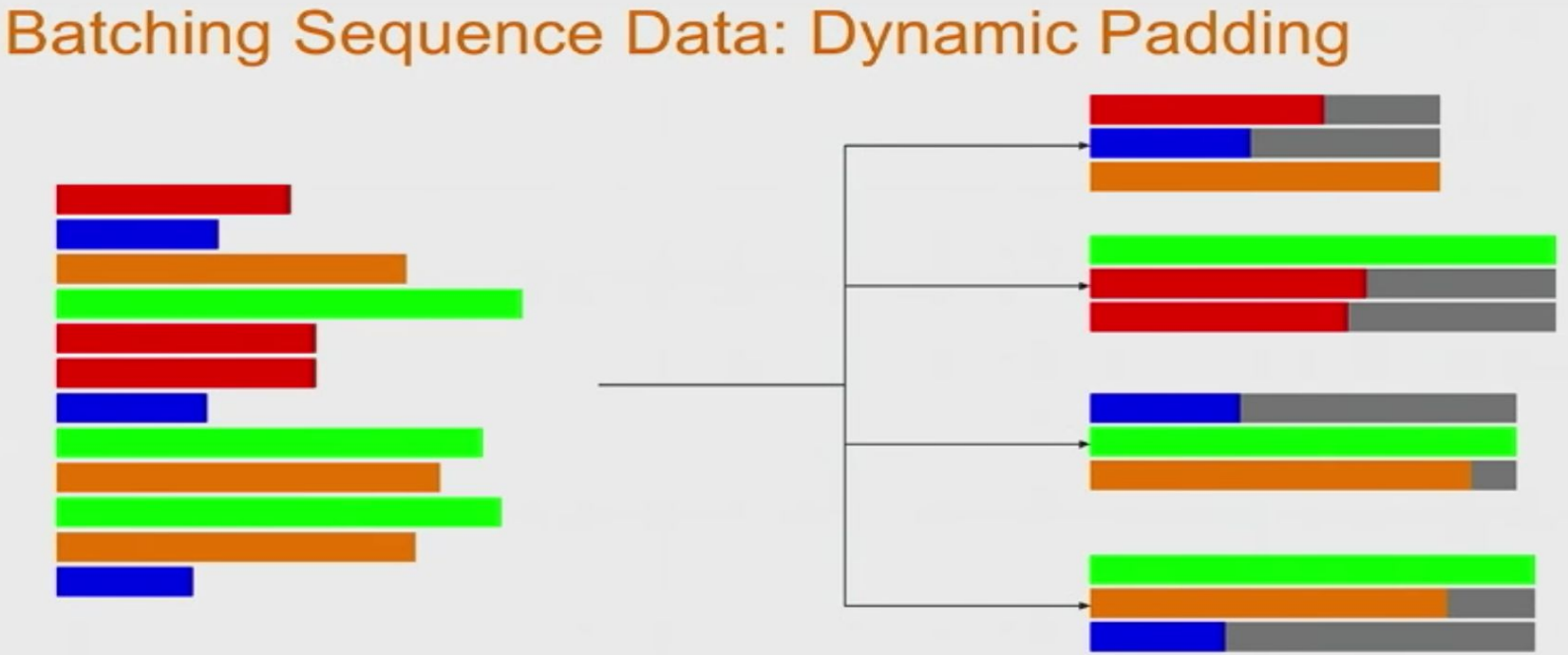
Shape Inference 101

```
>> 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?

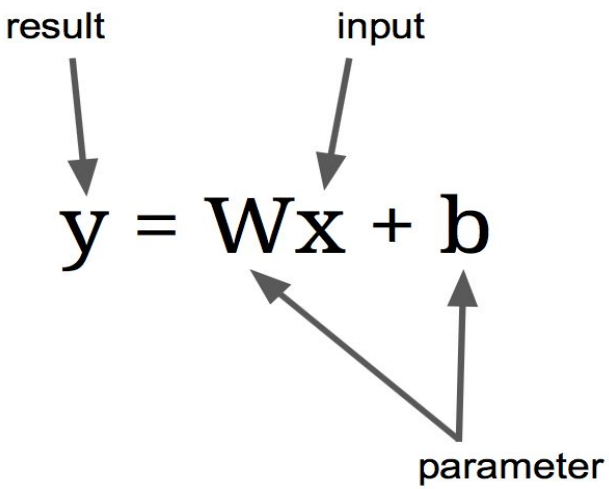


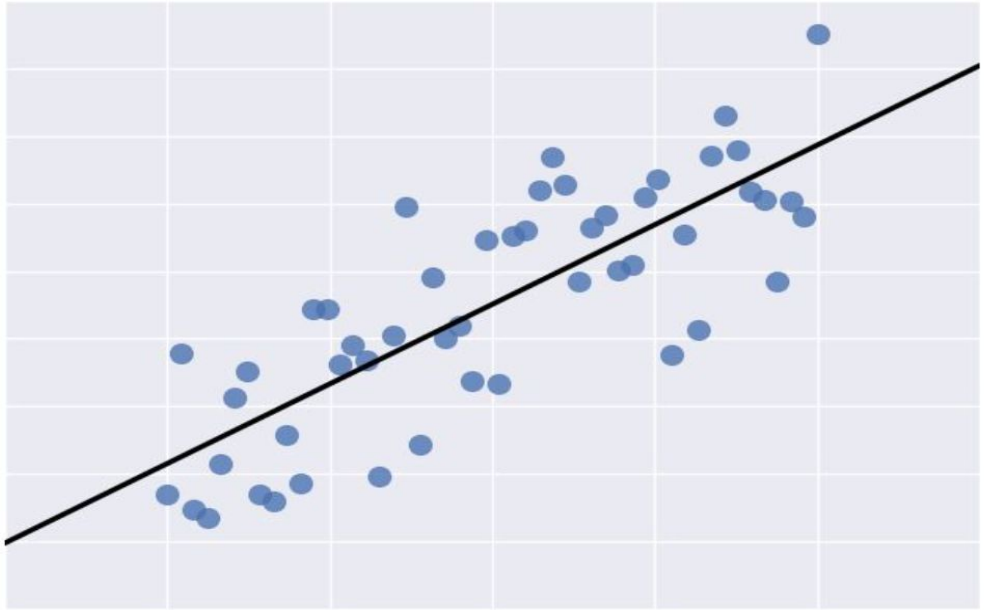
Writing Extensions - Compound Ops

```
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=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

Write a regex to create a tag group

Split on underscores

Data download links

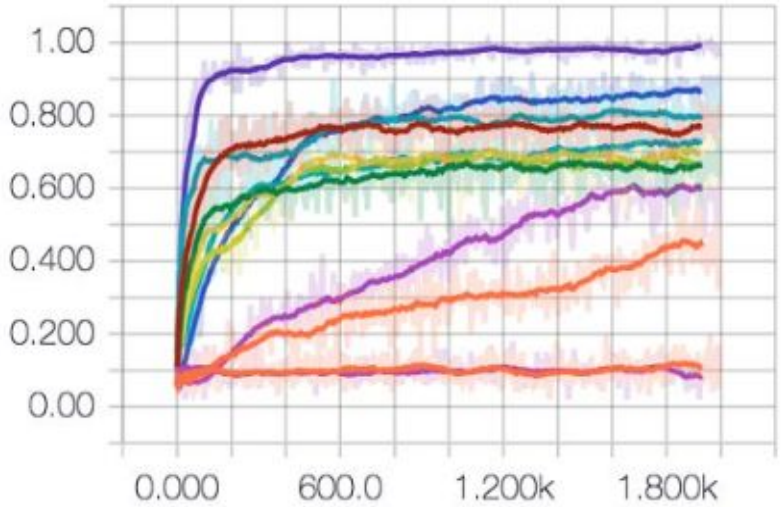
Tooltip sorting method: default

Smoothing

0.6

accuracy

accuracy/accuracy



TensorBoard

EVENTS IMAGES GRAPH HISTOGRAMS

Fit to screen

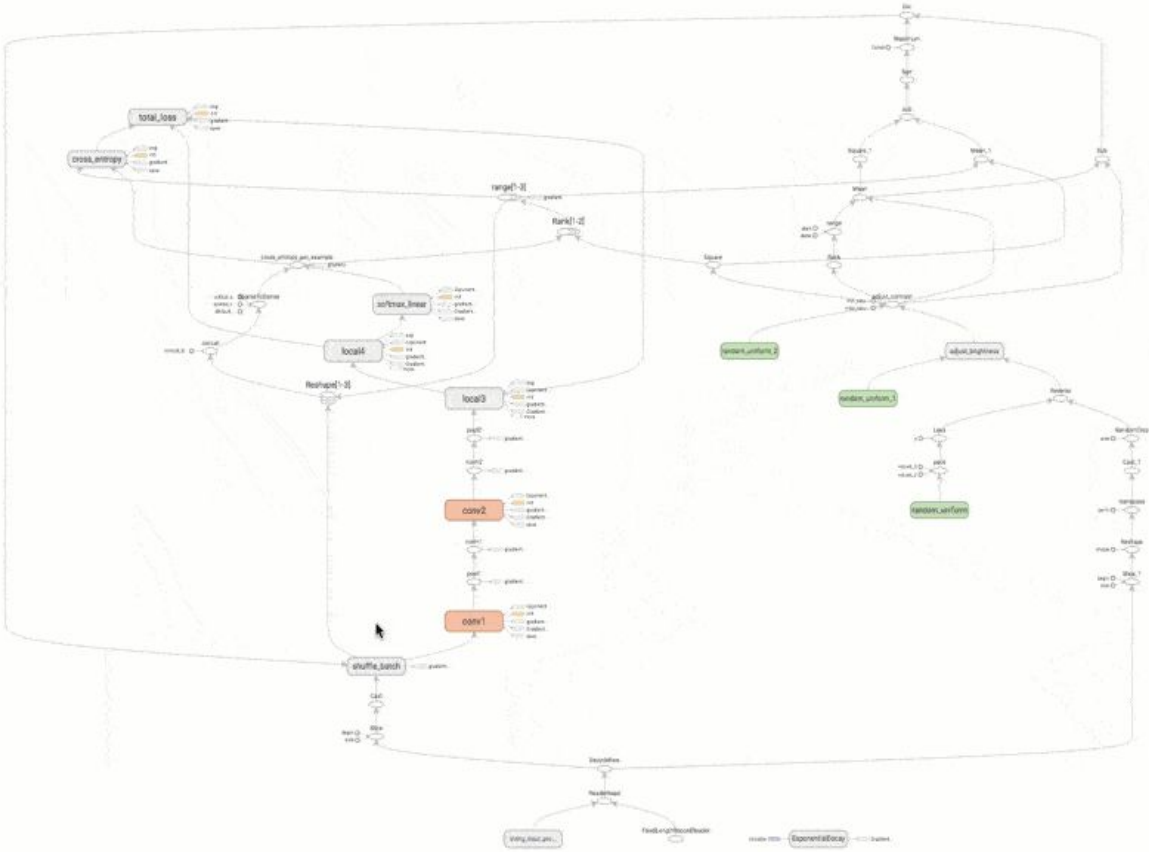
Run cifar-train

Upload Choose File

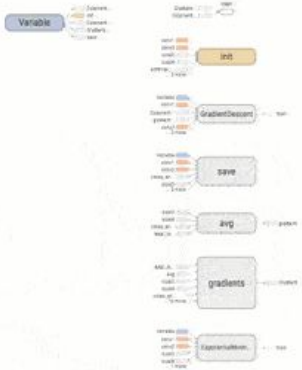
Color Structure

color: same substructure
gray: unique substructure

Main Graph



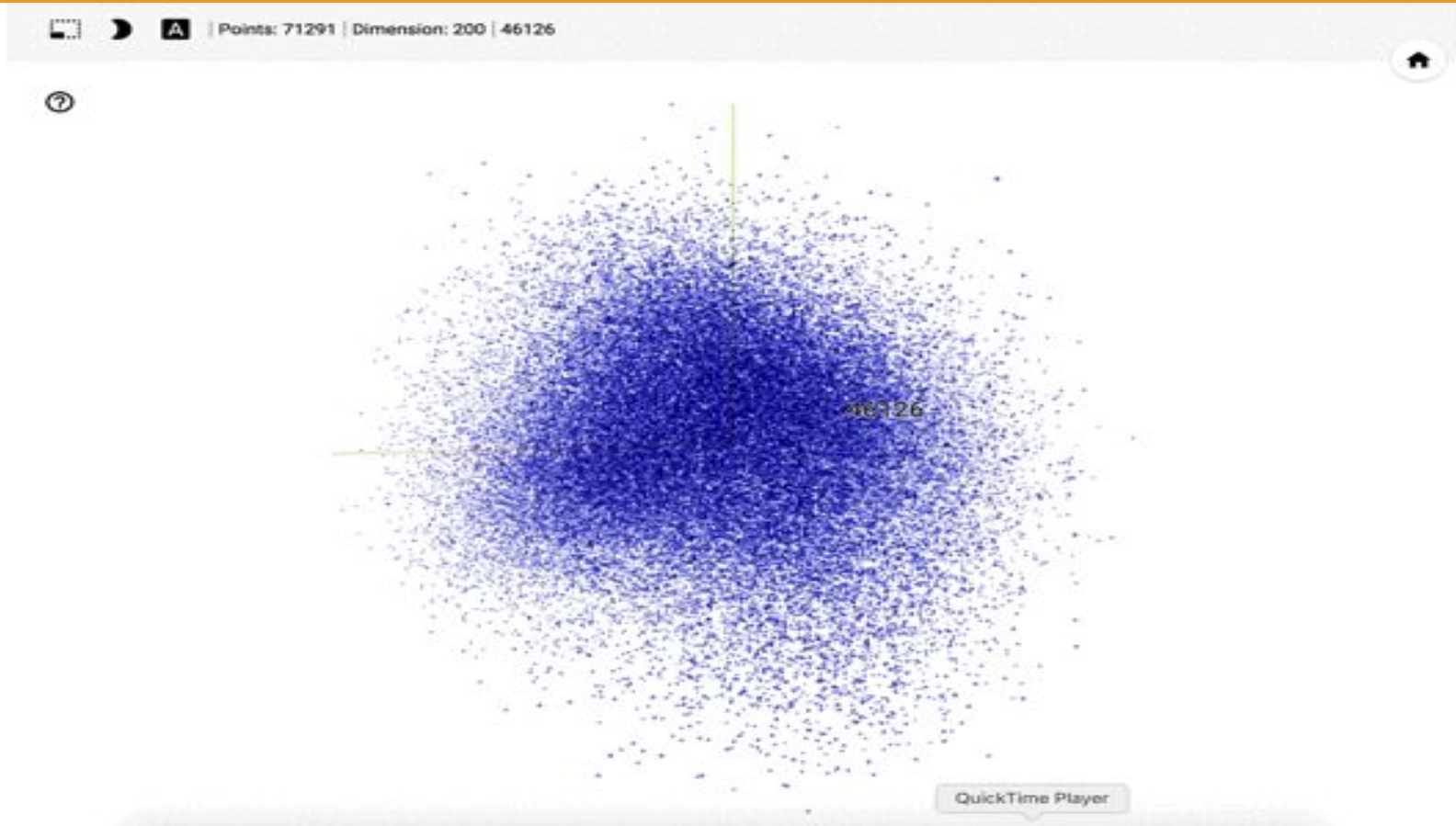
Auxiliary nodes



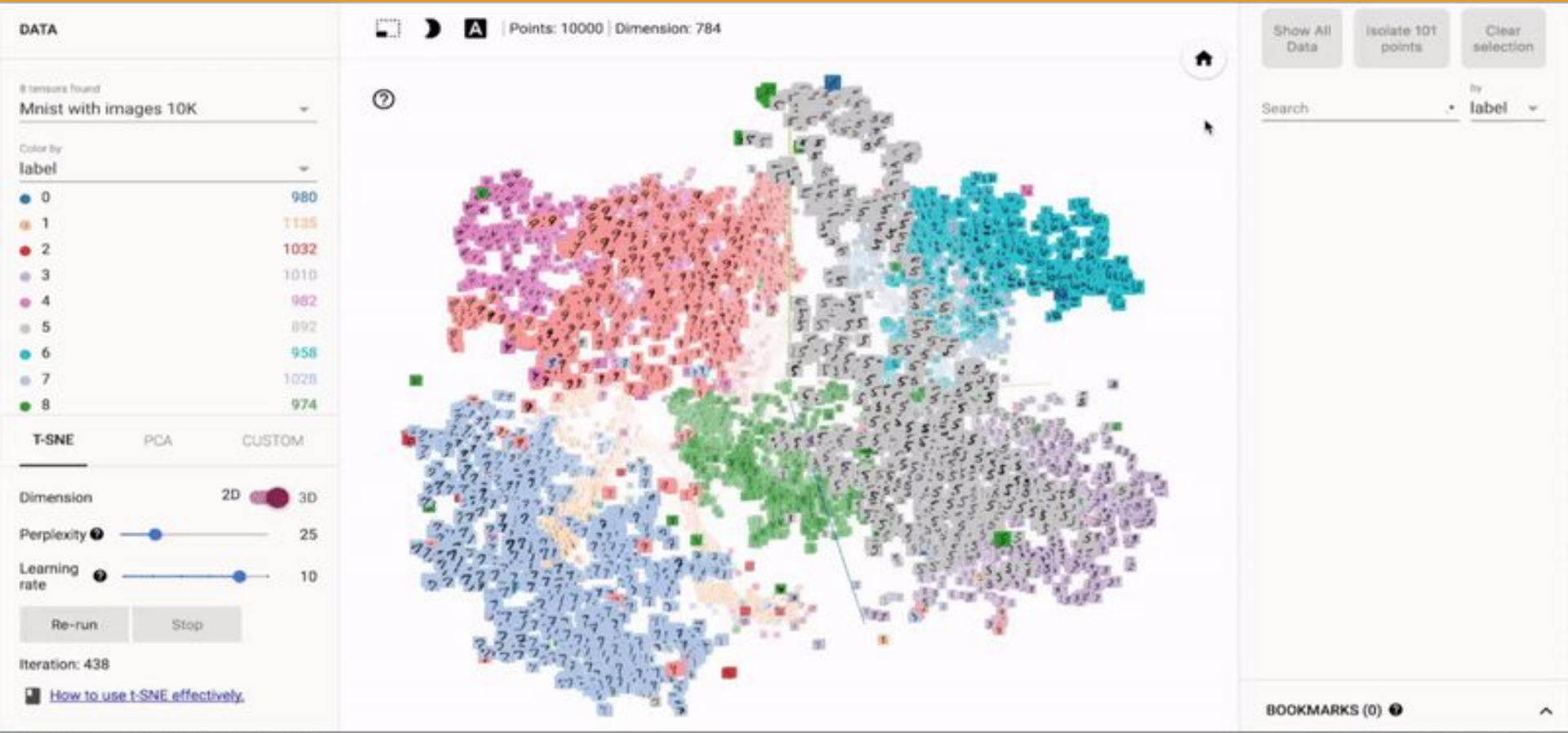
Graph (* = expandable)

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

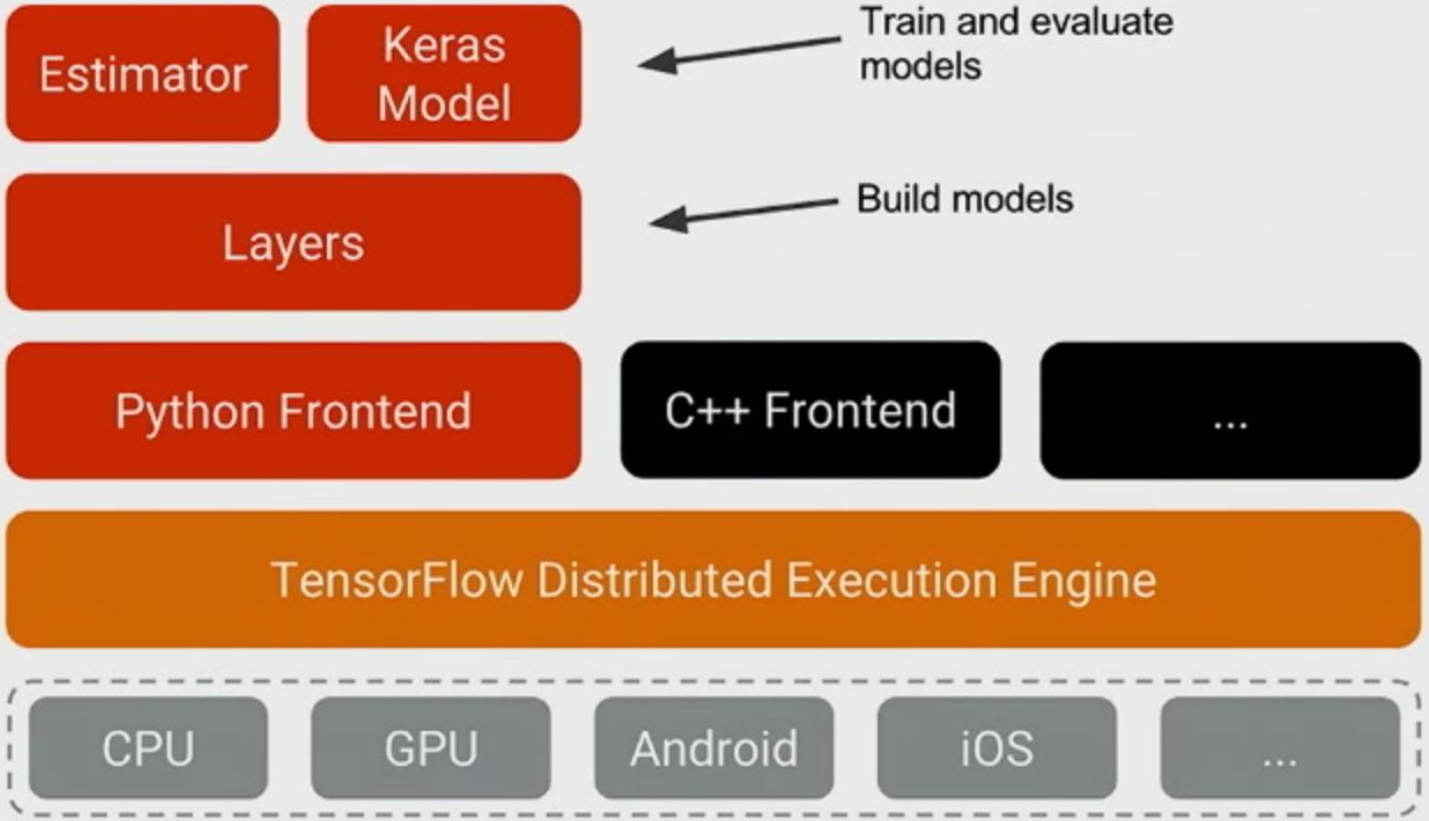
Visualization - Embedding Projector (link)



Visualization - Embedding Projector (link)



High-level APIs


Estimator
Keras Model
Train and evaluate models
Layers
Build models
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...

Keras

- Support two backends
  - Theano
  - TensorFlow
- Focus on fast prototyping
- Used heavily in Kaggle and research

Keras

```
1  from keras.layers import Dense, Activation
2
3  model.add(Dense(output_dim=64, input_dim=100))
4  model.add(Activation('relu'))
5  model.add(Dense(output_dim=10))
6  model.add(Activation('softmax'))
7  model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
8
9  model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)
10 classes = model.predict_classes(X_test, batch_size=32)
11 probs = model.predict_proba(X_test, batch_size=32)
```

Estimator (originally skflow)

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

Deep Neural Networks

```
feature_columns = learn.infer_real_valued_columns_from_input(x_train)
regressor = learn.DNNRegressor(
    feature_columns=feature_columns, hidden_units=[10, 10])

regressor.fit(x_train, y_train, steps=5000, batch_size=1)

y_predicted = list(regressor.predict(scaler.transform(x_test), as_iterable=True))
score = metrics.mean_squared_error(y_predicted, y_test)
```

Custom Model

```
def my_model(features, target):
  """DNN with three hidden layers, and dropout of 0.1 probability."""
  target = tf.one_hot(target, 3, 1, 0)

  features = layers.stack(features, layers.fully_connected, [10, 20, 10],
                          normalizer_fn=layers.dropout,
                          normalizer_params={'keep_prob': 0.9})

  logits = layers.fully_connected(features, 3, activation_fn=None)
  loss = tf.contrib.losses.softmax_cross_entropy(logits, target)

  train_op = tf.contrib.layers.optimize_loss(
      loss, tf.contrib.framework.get_global_step(), optimizer='Adagrad',
      learning_rate=0.1)

  return ({
      'class': tf.argmax(logits, 1),
      'prob': tf.nn.softmax(logits)}, loss, train_op)
```

Custom Model

```
1  x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)
2  x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)
3  x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)
4  x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)
5  x = tf.contrib.layers.relu(x)
6  x = tf.contrib.layers.dropout(x, 0.5)
```

Monitors

```
x_train, x_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=42)

x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train, test_size=0.2, random_state=42)

val_monitor = learn.monitors.ValidationMonitor(
    x_val, y_val, early_stopping_rounds=200)

  classifier = learn.DNNClassifier(
      feature_columns=learn.infer_real_valued_columns_from_input(x_train),
      hidden_units=[10, 20, 10], n_classes=3, model_dir=model_dir,
      config=tf.contrib.learn.RunConfig(save_checkpoints_secs=1))
  classifier.fit(x=x_train, y=y_train, steps=2000, monitors=[val_monitor])
```

FeatureColumn API

```
# Categorical base columns.
gender = sparse_column_with_keys(
  column_name="gender",
  keys=["Female", "Male"])
education = sparse_column_with_hash_bucket(
  "education",
  hash_bucket_size=1000)

# Continuous base columns.
age = real_valued_column("age")
age_buckets = bucketized_column(
  age,
  boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

Run Configuration

```
run_config = tf.contrib.learn.estimators.RunConfig(
    num_cores=3, gpu_memory_fraction=0.6)
  classifier = tf.contrib.learn.DNNClassifier(feature_columns=feature_columns,
                                              hidden_units=[10, 20, 10],
                                              n_classes=3,
                                              config=run_config)
```

Flexible Automatic Input Handling

- Numpy matrix
- Pandas dataframe/series
- Iterators
- HDF5
- TFRecords, TFExamples

Building Blocks

- Estimator (Linear, KMeans, SVM, RandomForest, etc)
- DataFrame
- Layers, Optimizers
- FeatureColumn
- Losses/Metrics
- Monitors/SessionRunHook
- Experiment

Modelling Techniques

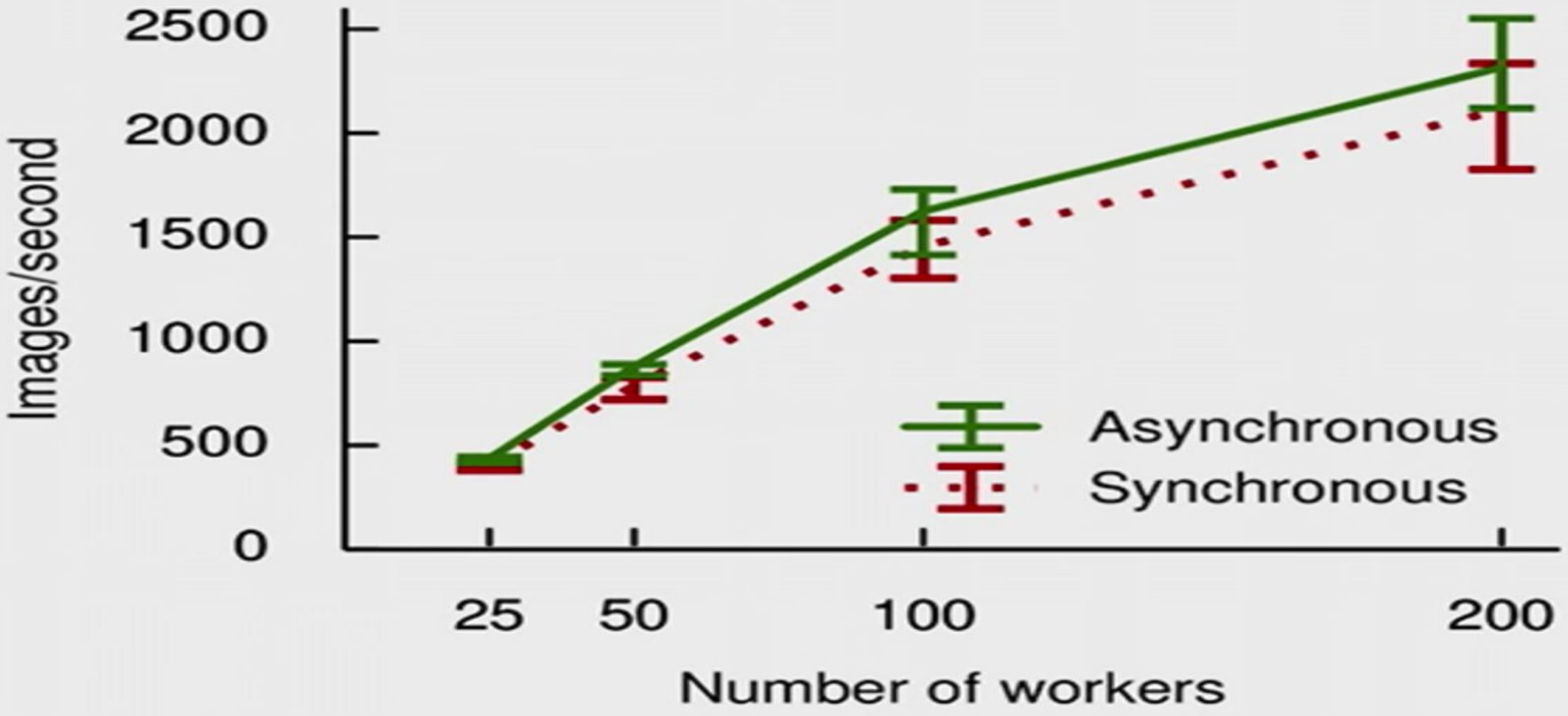
- Early Stopping
- Custom Learning Rate Decay
- Custom Class Weights
- Dropout
- Batch Normalization
- Clip Gradients

Ready for exercises?

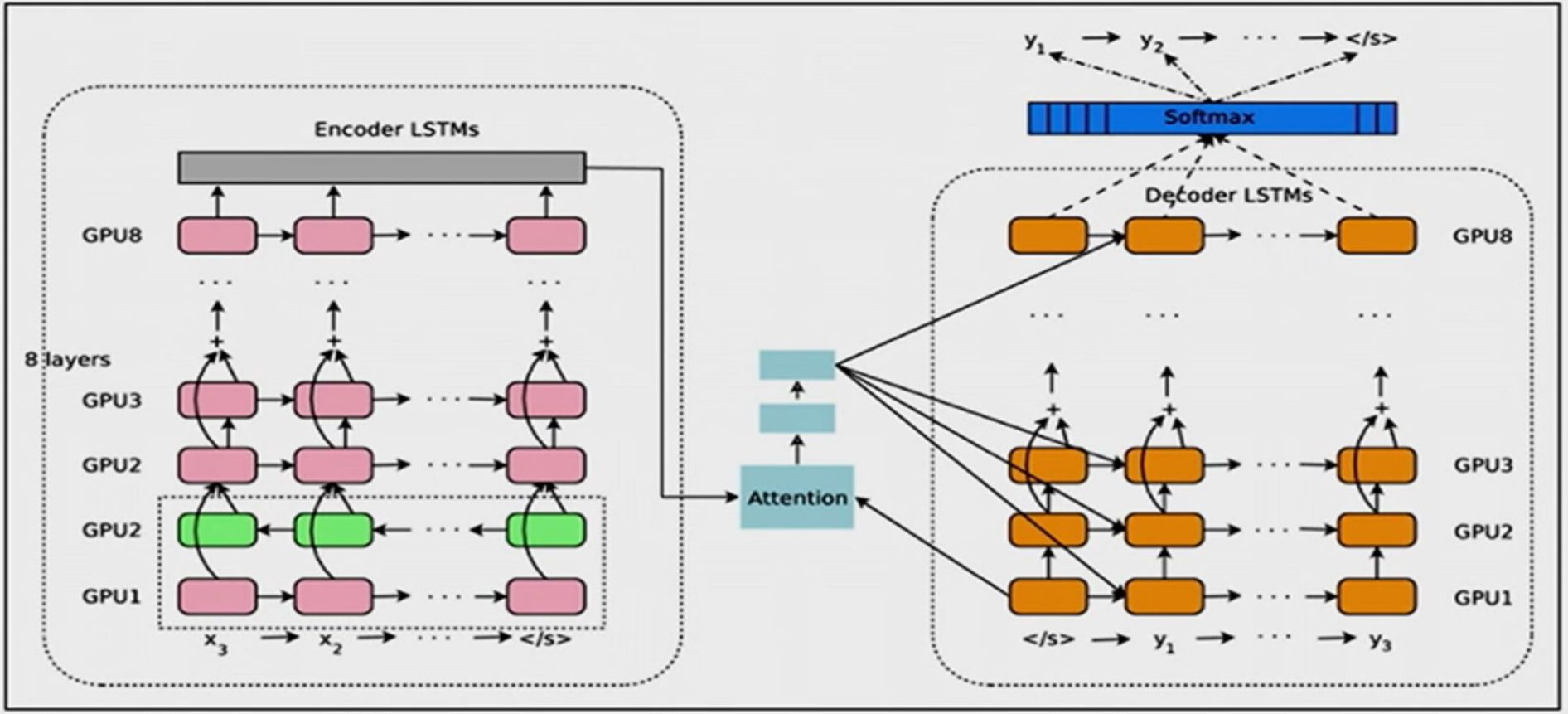
- Link to exercises: https://github.com/terrytangyuan/exercise-ccasa-2017.git

Distributed TensorFlow

Data Parallelism



Model Parallelism



Types of Tasks

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

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)# avoids memory issue
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, ...)
```

Q & A

Thanks!