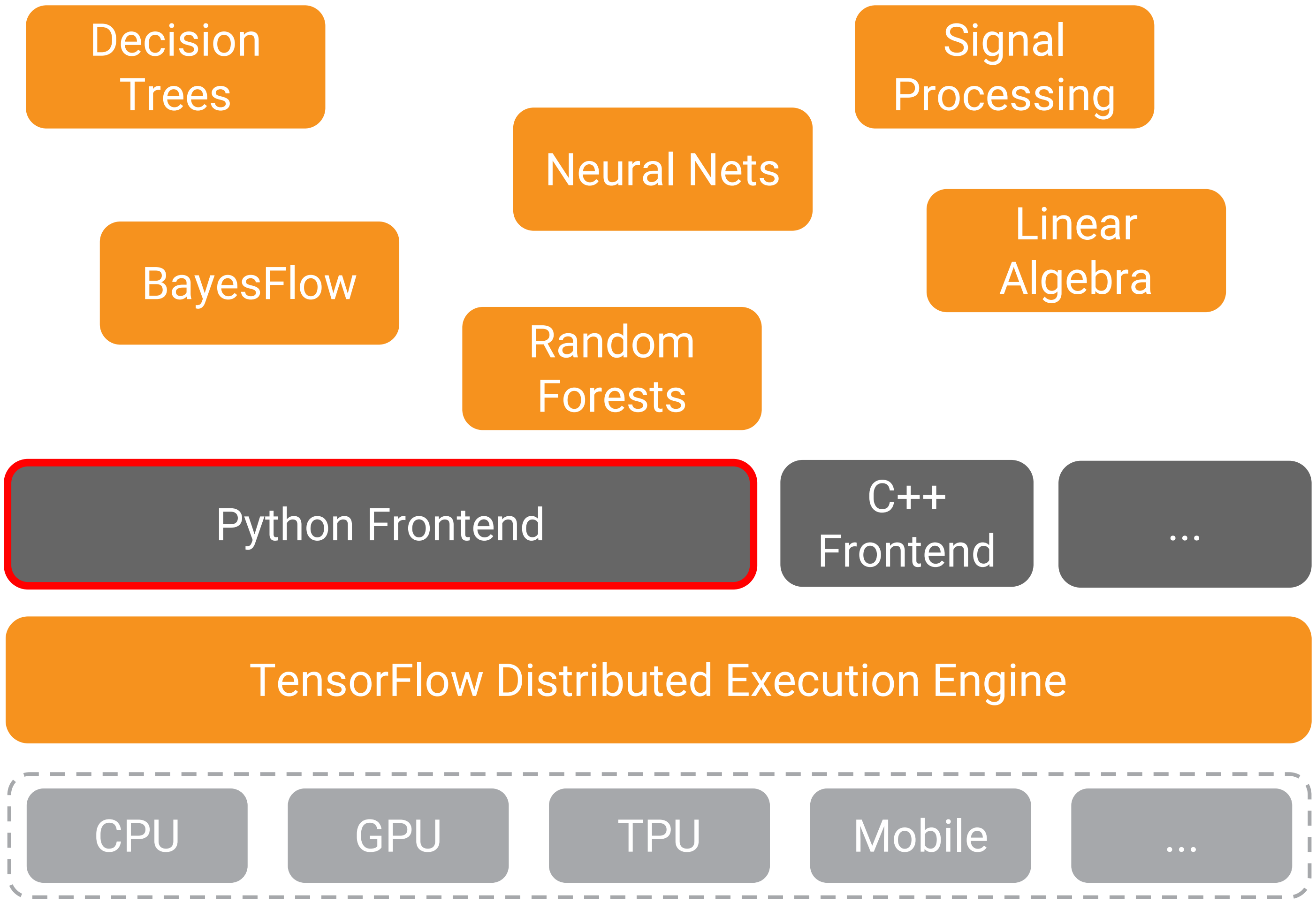
TensorFlow

Research at Scale

Rajat Monga

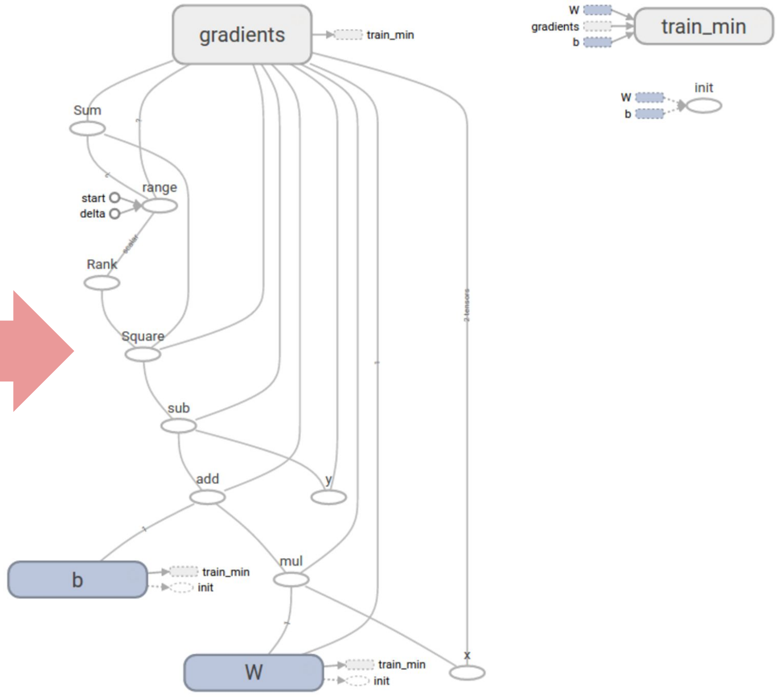

Decision Trees
Signal Processing
Neural Nets
Linear Algebra
BayesFlow
Random Forests
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
TPU
Mobile
...

Graphs

```
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], tf.float32)
b = tf.Variable([-.3], tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x:x_train, y:y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss  = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```



What if...

You can call TensorFlow ops directly from Python?

Eager Execution

As simple as possible

Boilerplate

```python
x = tf.placeholder(tf.float32, shape=[1, 1])
m = tf.matmul(x, x)

print(m)
# Tensor("MatMul:0", shape=(1, 1), dtype=float32)

with tf.Session() as sess:
  m_out = sess.run(m, feed_dict={x: [[2.]]})
print(m_out)
# [[4.]]
```

Code like this...

~~Boilerplate~~

```python
x = [[2.]]
m = tf.matmul(x, x)

print(m)
# tf.Tensor([[4.]], dtype=float32, shape=(1,1))
```

Becomes this

Instant Errors

```
x = tf.gather([0, 1, 2], 7)

InvalidArgumentError: indices = 7 is not in [0, 3) [Op:Gather]
```

Python Control Flow

```python
a = tf.constant(6)
while not tf.equal(a, 1):
  if tf.equal(a % 2, 0):
    a = a / 2
  else:
    a = 3 * a + 1
  print(a)
```

```
# Outputs
tf.Tensor(3, dtype=int32)
tf.Tensor(10, dtype=int32)
tf.Tensor(5, dtype=int32)
tf.Tensor(16, dtype=int32)
tf.Tensor(8, dtype=int32)
tf.Tensor(4, dtype=int32)
tf.Tensor(2, dtype=int32)
tf.Tensor(1, dtype=int32)
```

Gradients

- Operations executed are recorded on a tape
- Tape is played back to compute gradients

Gradients

```
def square(x):
  return tf.multiply(x, x)  # Or x * x

grad = tfe.gradients_function(square)



print(square(3.))    # tf.Tensor(9., dtype=tf.float32
print(grad(3.))      # [tf.Tensor(6., dtype=tf.float32))]
```

Gradients

```
def square(x):
  return tf.multiply(x, x)  # Or x * x

grad = tfe.gradients_function(square)
gradgrad = tfe.gradients_function(lambda x: grad(x)[0])


print(square(3.))    # tf.Tensor(9., dtype=tf.float32)
print(grad(3.))      # [tf.Tensor(6., dtype=tf.float32)]
print(gradgrad(3.))  # [tf.Tensor(2., dtype=tf.float32))]
```

Custom Gradients

```
def log1pexp(x):
  return tf.log(1 + tf.exp(x))
grad_log1pexp = tfe.gradients_function(log1pexp)

print(grad_log1pexp(0.))
```

Works fine, prints [0.5]

Custom Gradients

```
def log1pexp(x):
  return tf.log(1 + tf.exp(x))
grad_log1pexp = tfe.gradients_function(log1pexp)

print(grad_log1pexp(100.))
```

[nan] due to numeric instability

Custom Gradients

```
@tfe.custom_gradient
def log1pexp(x):
  e = tf.exp(x)
  def grad(dy):
    return dy * (1 - 1 / (1 + e))
  return tf.log(1 + e), grad
grad_log1pexp = tfe.gradients_function(log1pexp)

# Gradient at x = 0 works as before.
print(grad_log1pexp(0.))   # [0.5]
# And now gradient computation at x=100 works as well.
print(grad_log1pexp(100.))  # [1.0]
```

Using GPUs

`tf.device()` for manual placement

```
with tf.device(“/gpu:0”):
  x = tf.random_uniform([10, 10])
  y = tf.matmul(x, x)
  # x and y reside in GPU memory
```

It’s not *that* different

A Collection of Operations

TensorFlow = Operation Kernels + Composition

- Session: One way to compose operations
- Eager execution: Compose using Python

Building Models

The same APIs as graph building (`tf.layers`, `tf.train.Optimizer`, `tf.data` etc.)

```
model = tf.layers.Dense(units=1, use_bias=True)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
```

Building Models

```
model = tf.layers.Dense(units=1, use_bias=True)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)

# Define a loss function
def loss(x, y):
  return tf.reduce_mean(tf.square(y - model(x)))
```

Training Models

Compute and apply gradients

```
for (x, y) in get_next_batch():
  optimizer.apply_gradients(grad_fn(x, y))
```

Training Models

Compute and apply gradients

```
grad_fn = tfe.implicit_gradients(loss)

for (x, y) in get_next_batch():
  optimizer.apply_gradients(grad_fn(x, y))
```

No more graphs then?

Graphs are

Optimizable

- Automatic buffer reuse
- Constant folding
- Inter-op parallelism
- Automatic trade-off between compute and memory

Graphs are

Deployable

- TensorFlow Serving
- Mobile
- Any other C++/Java/other program

  Without loss in translation between runtimes

Graphs are

Transformable

- Carve out subgraphs to offload to accelerators
- Train with quantization in mind

Imperative to declarative and back

- **Write model definition code once**
  The exact same code can execute operations in one Python process and construct graphs in another (see examples)

- **Checkpoints are compatible**
  Train eagerly, checkpoint, load in a graph, or vice-versa

- **Future:**
  Within the same Python process, selectively "compile" portions of your computations into graphs and execute

Start with eager

```
optimizer = tf.train.AdagradOptimizer(0.01)
for _ in xrange(num_iters):
  (images, labels) = iterator.next()
  optimizer.minimize(model_loss)
```

Run distributed

```
optimizer = tf.train.AdagradOptimizer(0.01)

step = tf.train.get_or_create_global_step()
train_op = optimizer.minimize(model_loss, global_step=step)
```

Same model spec

```
hooks = [tf.train.StopAtStepHook(last_step=num_iters)]
with tf.train.MonitoredTrainingSession(hooks=hooks, ...) as mon_sess:
  while not mon_sess.should_stop():
    mon_sess.run(train_op)
```

Or even on TPUs

```
def model_fn():
  optimizer = tf.train.AdagradOptimizer(0.01)
  optimizer = tpu.CrossShardOptimizer(optimizer)
  step = tf.train.get_or_create_global_step()
  train_op = optimizer.minimize(model_loss, global_step=step)
  return tf.estimator.EstimatorSpec(train_op=train_op, ...)
```

Same model spec

```
estimator = tf.tpu_estimator.TPUEstimator(model_fn=model_fn, ...)
```

Thank you!

Rajat Monga