TensorFlow 内核剖析

编程模型

刘 光聪

技术规划部

2017-08-01

内容

架构概述

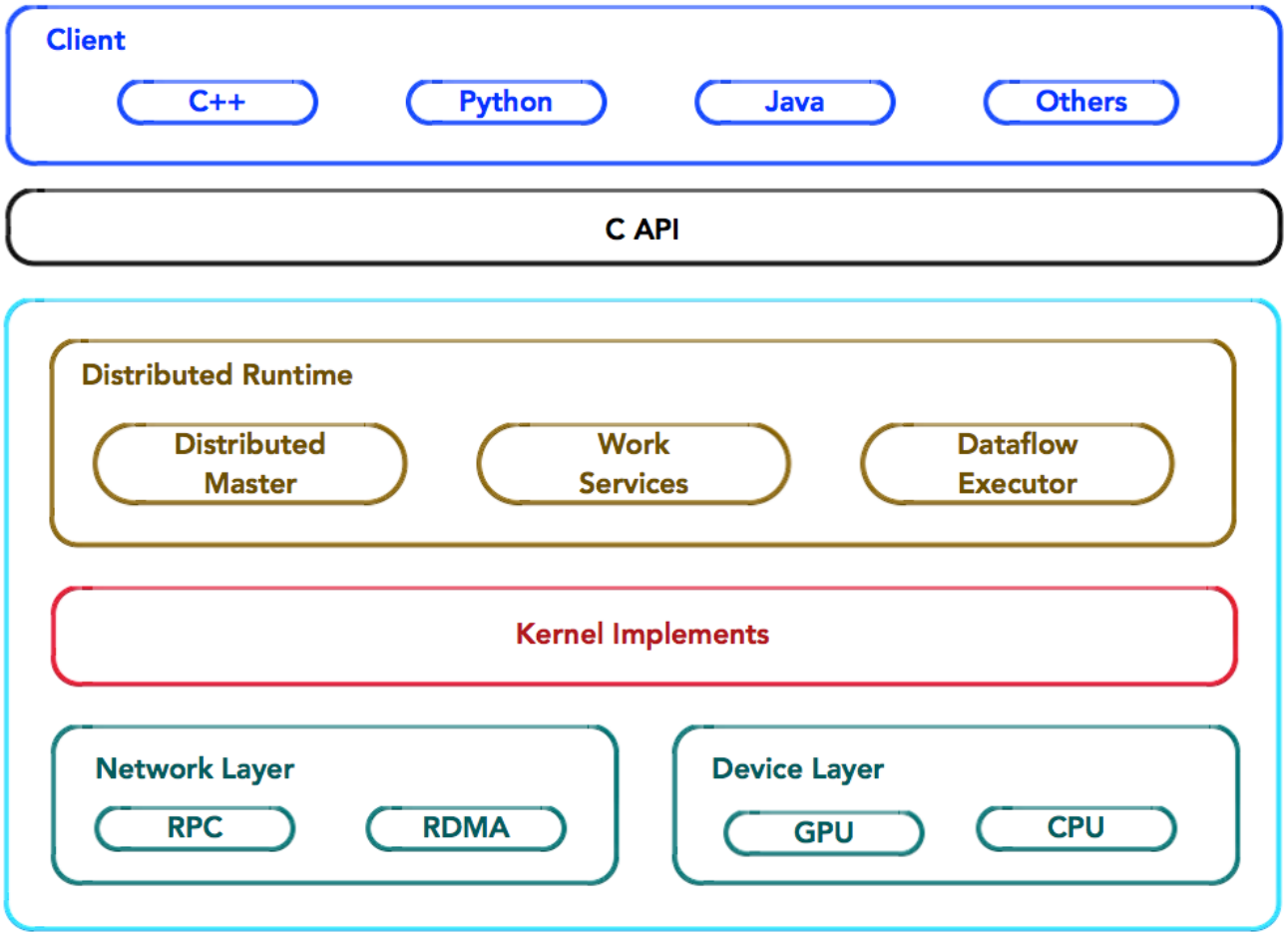
1. 系统架构
2. 图控制

What's TensorFlow

TensorFlow 是一个使用数据流图用于数值计算的开源软件库

- 跨平台：支持多 CPU/GPU 运算；台式机/服务器/移动设备
- 分布式：支持本地/分布式运行模式
- 多语言：支持 Python，C++，Java，Go 等多种编程语言接口
- 通用性：支持构建复杂的网络模型，支持数据/模型并发模式
- 可扩展：支持 OP 扩展，Kernel 扩展，Device 扩展
- 可视化：使用 TensorBoard 可视化计算图
- 自动微分：自动构造反向的计算子图，完成梯度的计算

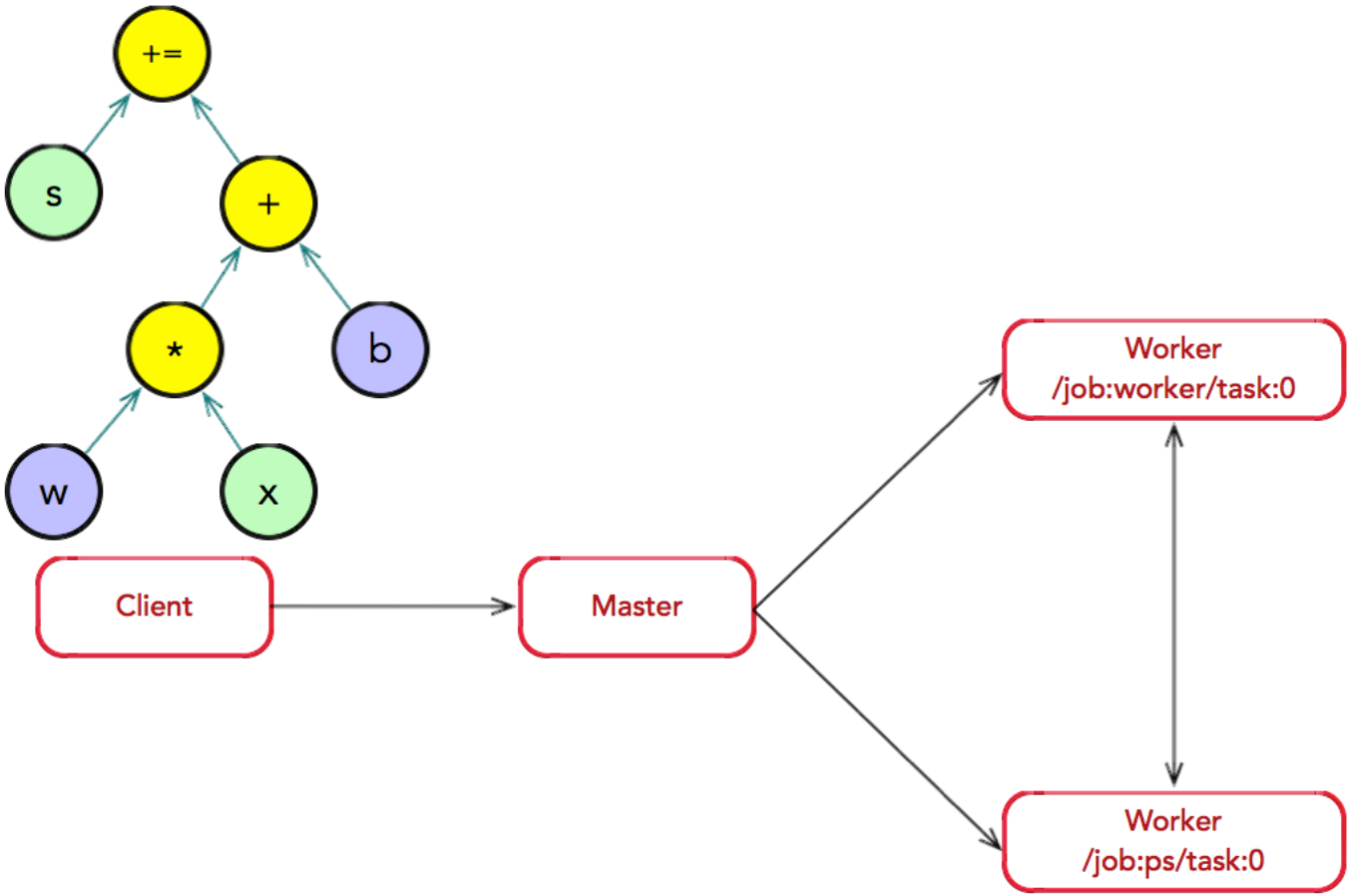
系统架构


Client
C++
Python
Java
Others
C API
Distributed Runtime
Distributed Master
Work Services
Dataflow Executor
Kernel Implements
Network Layer
RPC
RDMA
Device Layer
GPU
CPU

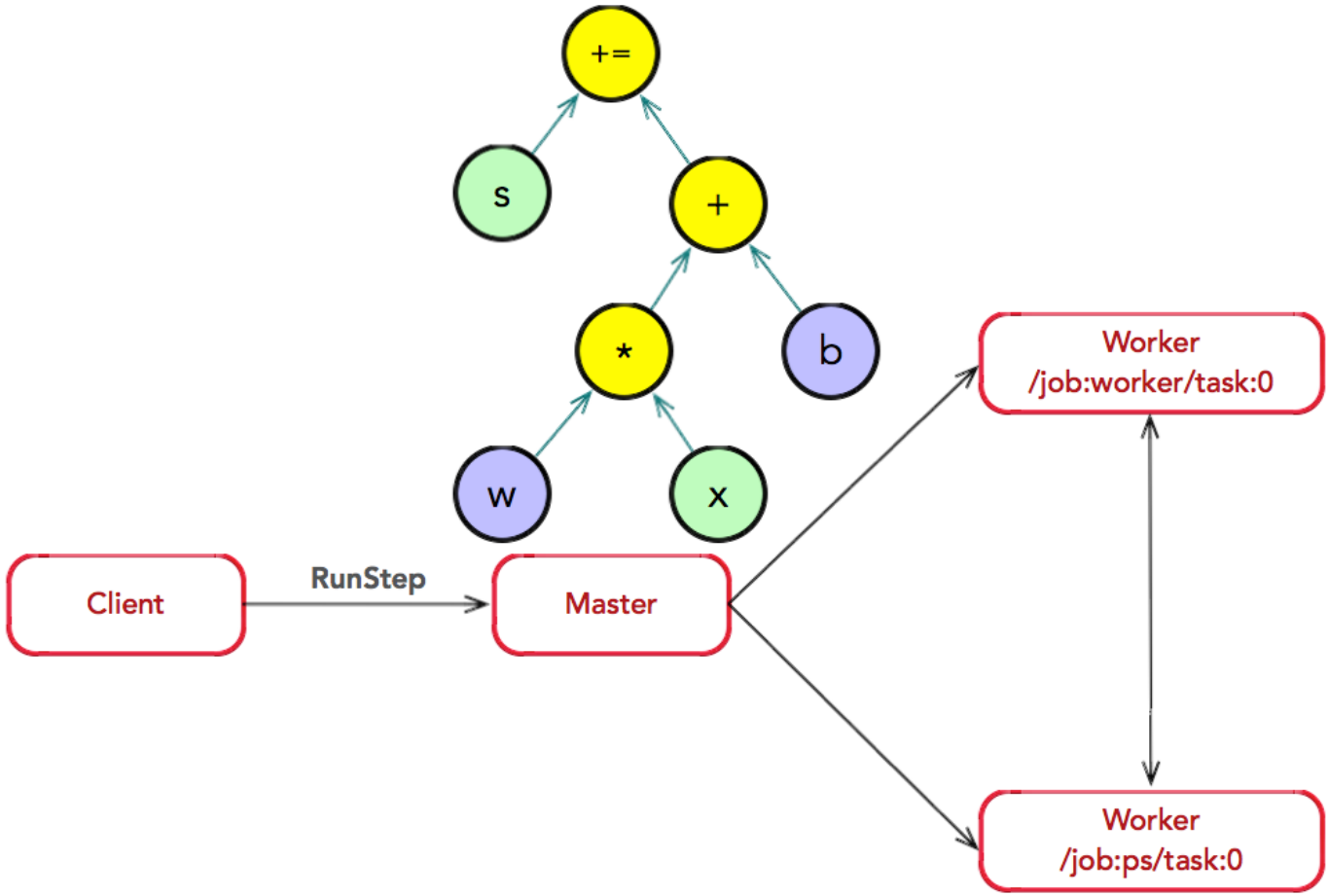
设计原则

- 延迟计算：图的构造与执行分离，并推迟计算图的执行过程
- 原子 OP：OP 是最小的抽象计算单元，支持构造复杂的网络模型
- 抽象设备：支持 CPU, GPU, ASIC 多种异构设备类型
- 抽象任务：基于任务的 PS，支持优化算法和网络模型的扩展

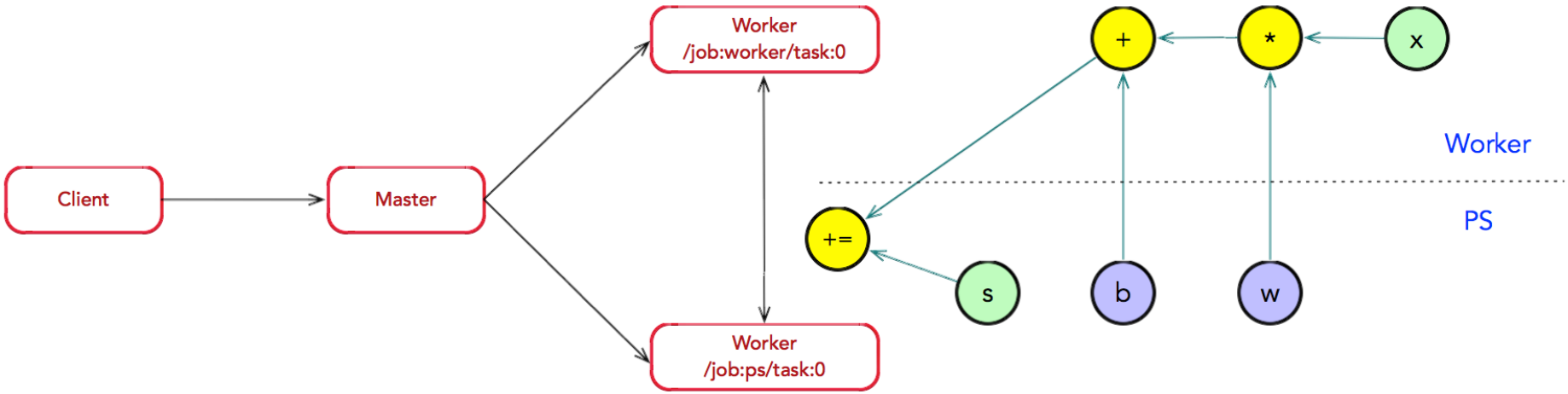
构造计算图


+=
s
+
*
b
w
x
Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

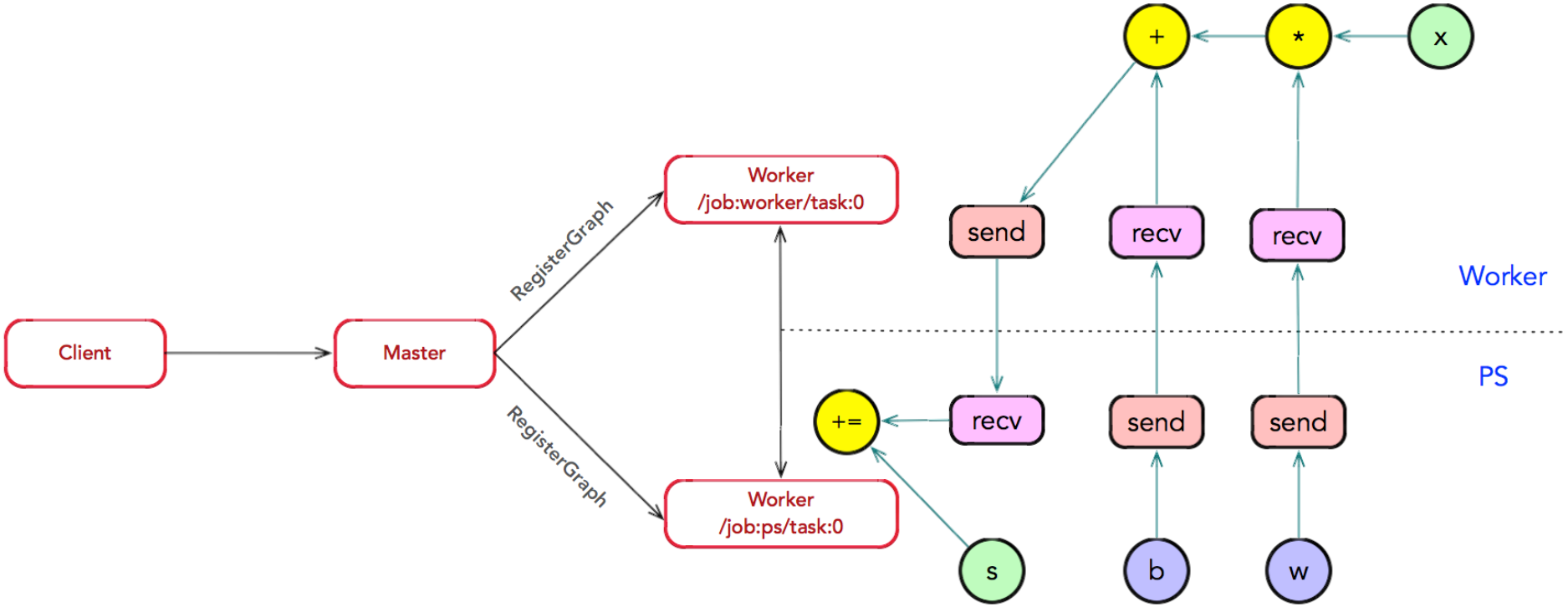
执行计算图


+=
s
+
*
b
w
x
Client
RunStep
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

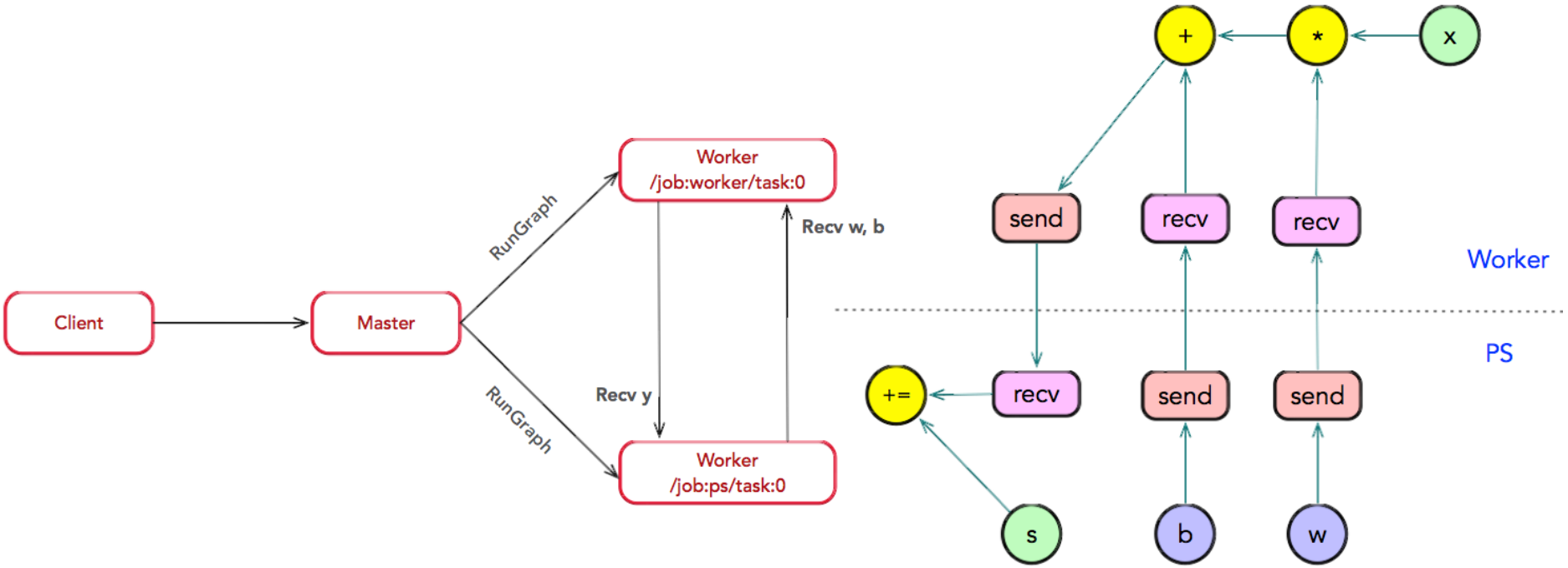
图分解


Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0
+
*
x
+=
s
b
w
Worker
PS

注册图

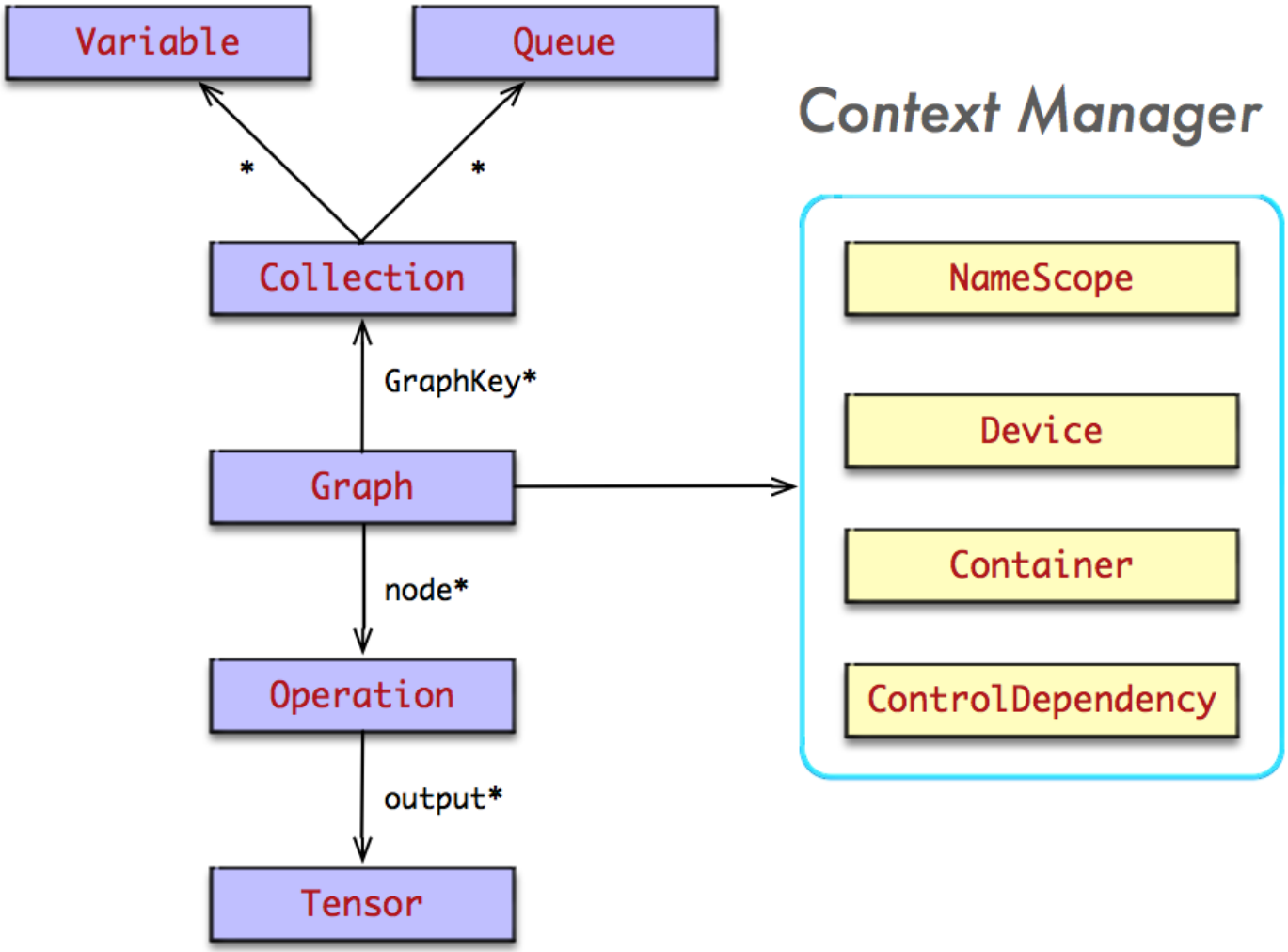

Client
Master
RegisterGraph
RegisterGraph
Worker
/job:worker/task:0
Worker
/job:ps/task:0
+
*
x
send
recv
recv
Worker
PS
+=
recv
send
send
s
b
w

执行图

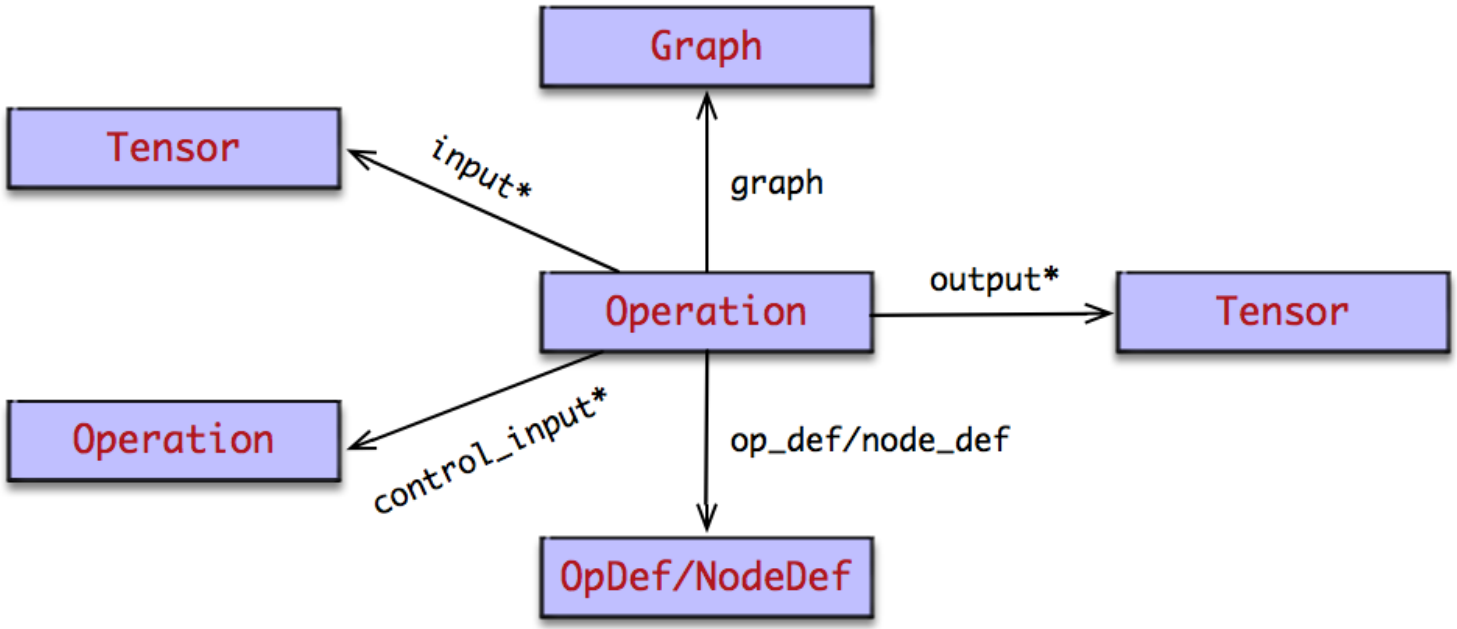

Client
Master
RunGraph
RunGraph
Worker
/job:worker/task:0
Worker
/job:ps/task:0
Recv w, b
Recv y
+
*
x
send
recv
recv
Worker
PS
+=
recv
send
send
s
b
w

编程模型

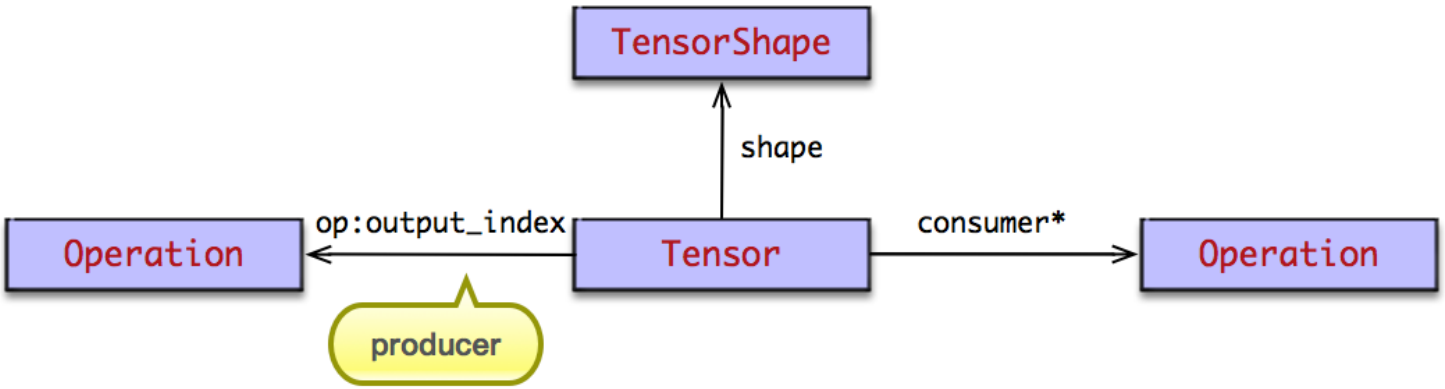
$Graph = Set\{OP\} + Set\{Tensor\}$



OP: 抽象计算


Graph
Tensor
input*
graph
Operation
output*
Tensor
Operation
control_input*
op_def/node_def
OpDef/NodeDef

Tensor: 承载数据

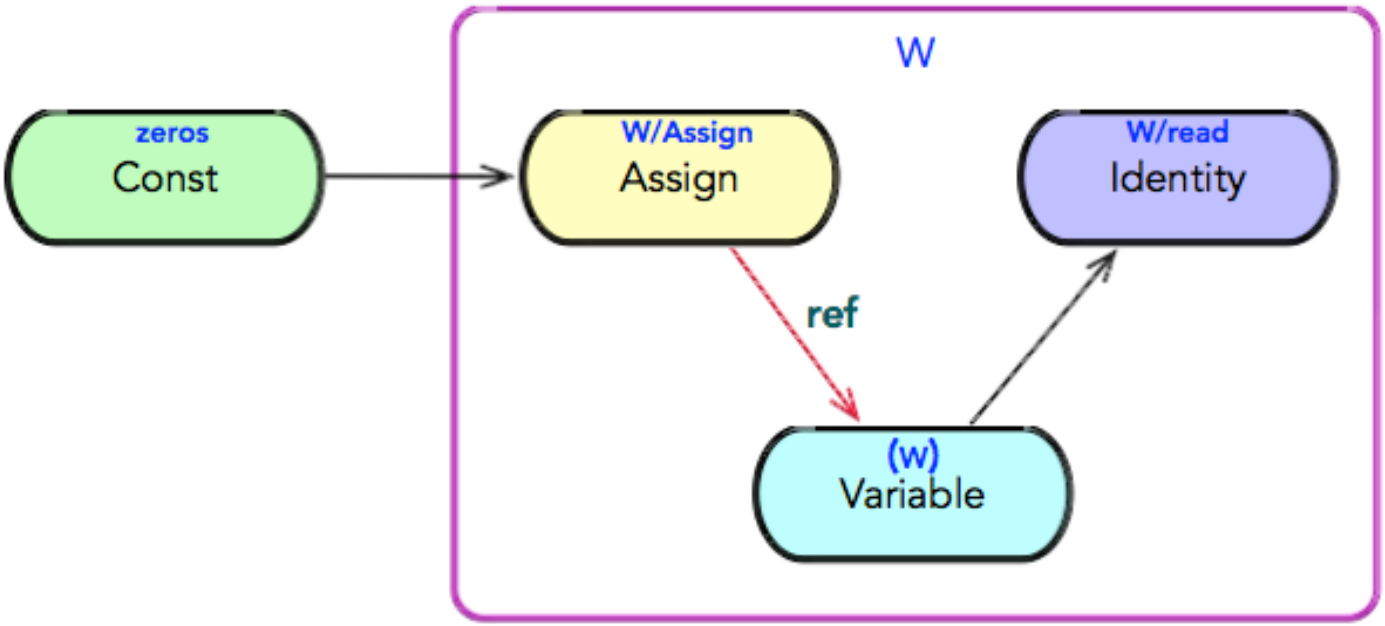

TensorShape
shape
Operation
op:output_index
Tensor
consumer*
Operation
producer

有向边

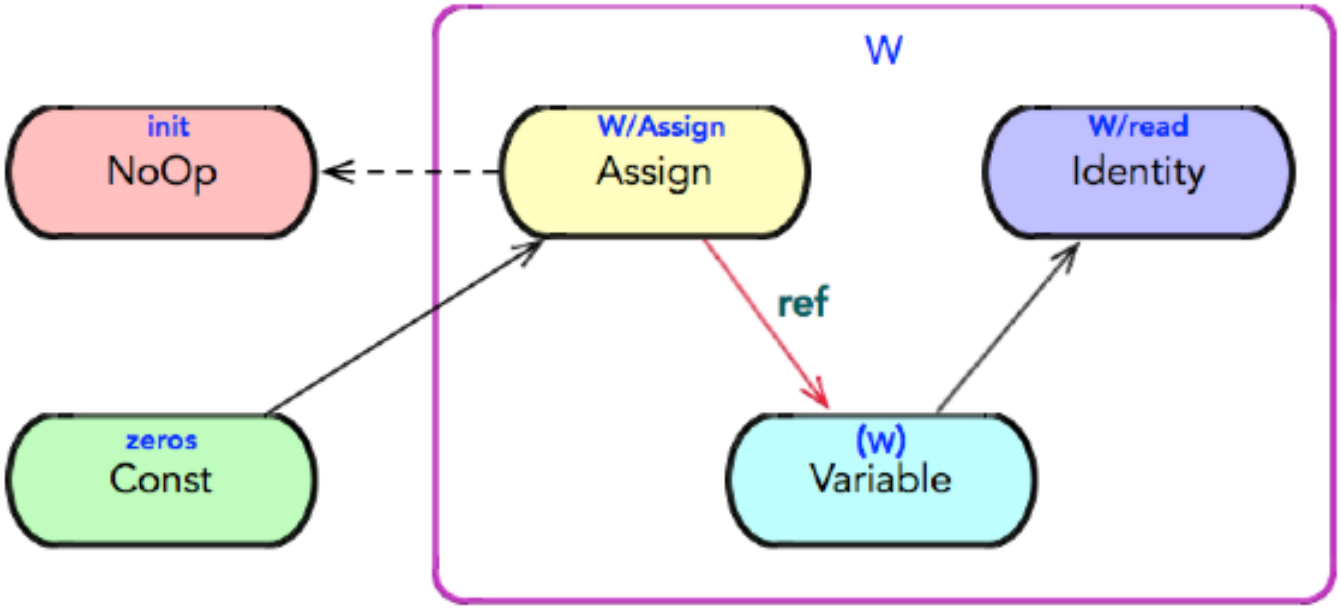
两种类型

- 普通边：承载 Tensor，且表示执行依赖关系
- 控制依赖边：不承载 Tensor，仅表示执行依赖关系

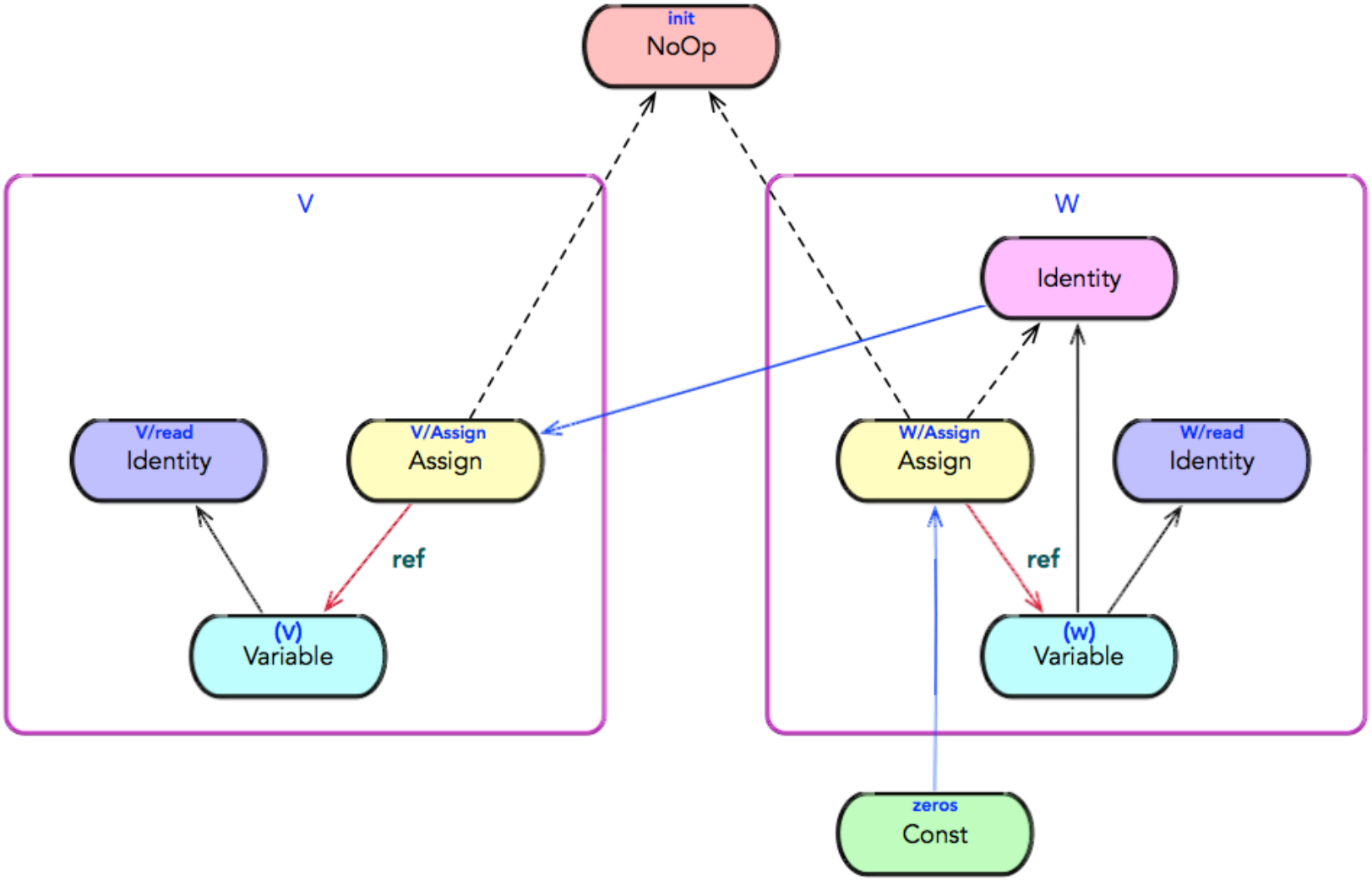
初始化模型



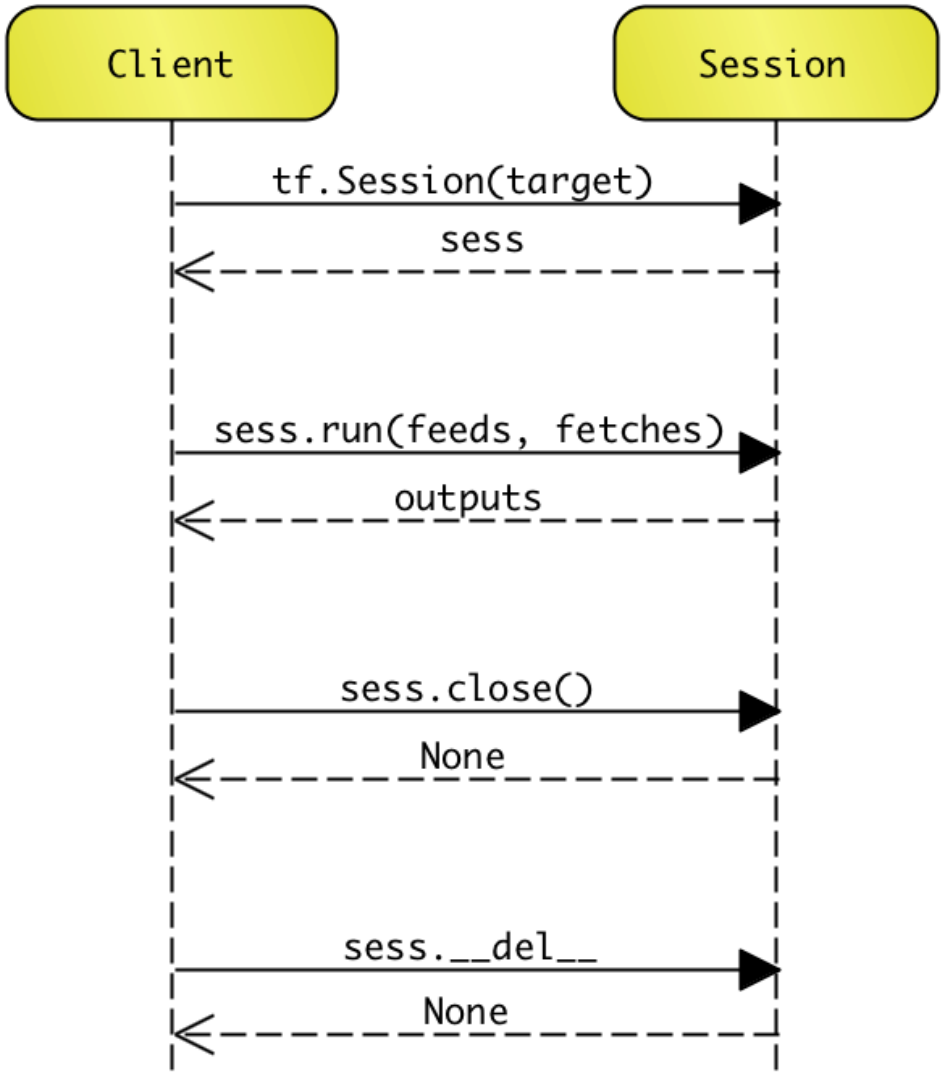
探秘 init_op


W
init
NoOp
W/Assign
Assign
W/read
Identity
ref
zeros
Const
(w)
Variable

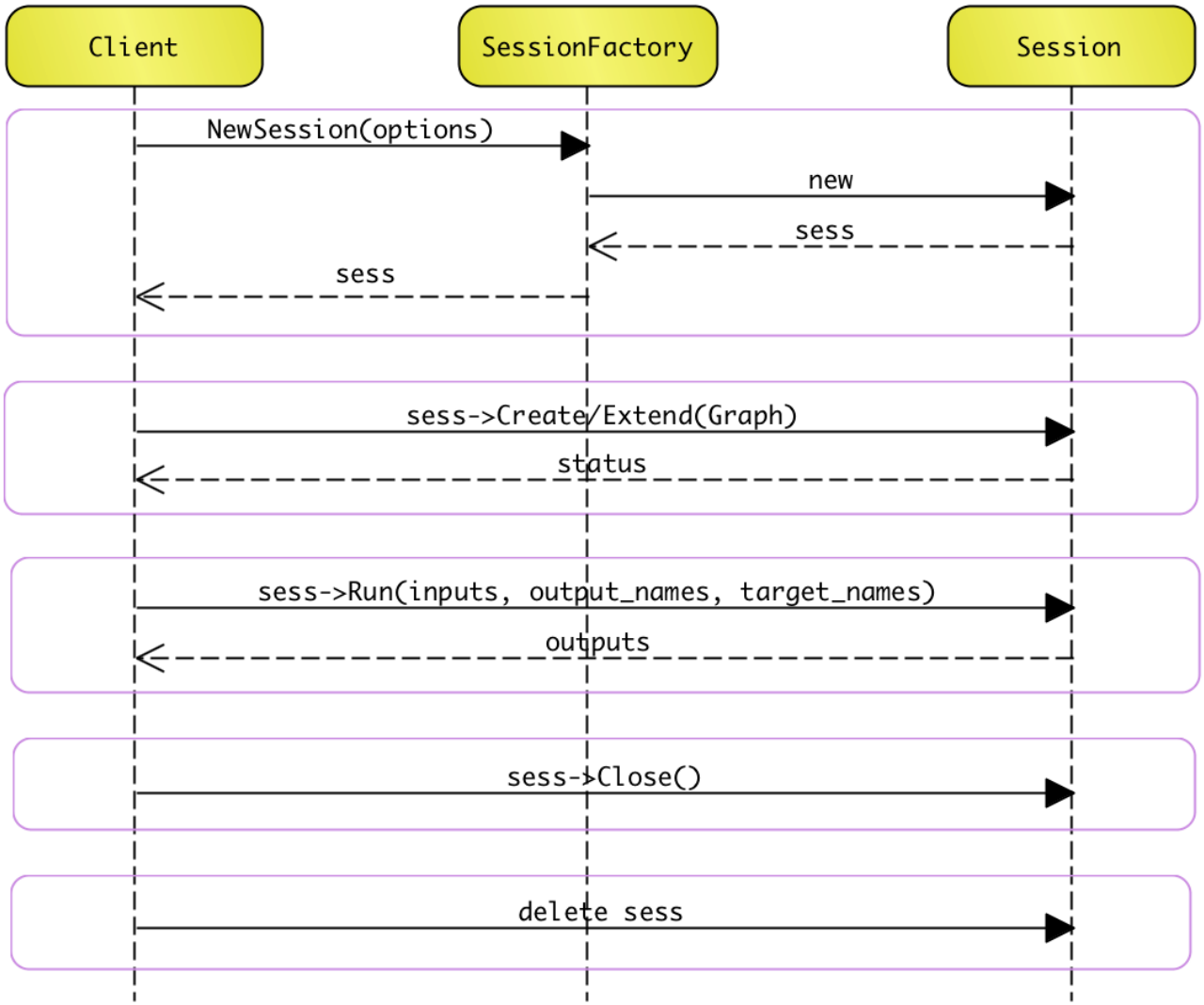
初始化依赖


init
NoOp
V
W
Identity
V/read
Identity
V/Assign
Assign
W/Assign
Assign
W/read
Identity
ref
ref
(V)
Variable
(w)
Variable
zeros
Const

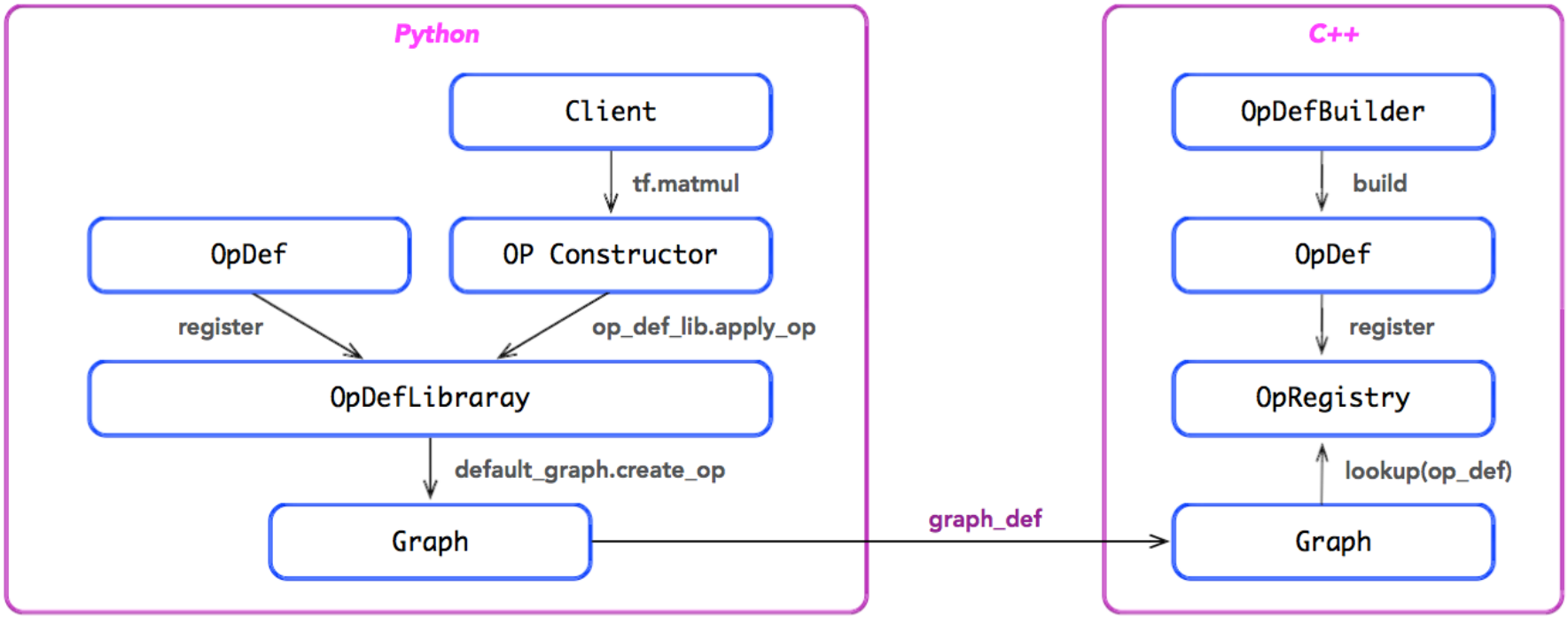
生命周期: Python


Client
Session
tf.Session(target)
sess
sess.run(feeds, fetches)
outputs
sess.close()
None
sess.__del__
None

生命周期: C++


Client
SessionFactory
Session
NewSession(options)
new
sess
sess
sess->Create/Extend(Graph)
status
sess->Run(inputs, output_names, target_names)
outputs
sess->Close()
delete sess

图构造与传递


Python
Client
tf.matmul
OpDef
OP Constructor
register
op_def_lib.apply_op
OpDefLibraray
default_graph.create_op
Graph
graph_def
C++
OpDefBuilder
build
OpDef
register
OpRegistry
lookup(op_def)
Graph

实例: OP 构造器

OP Constructor

```
def zeros_like(tensor, name=None):
  gen_array_ops._zeros_like(tensor, name=name)

tensor = tf.constant([1, 2], name="n1")
zeros = tf.zeros_like(tensor, name="n2")
```

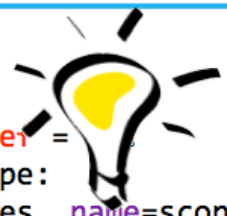
Code Generator

```
def _zeros_like(dtype, shape=None, name=None):
 return _op_def_lib.apply_op("ZerosLike", x=x, name=name)
```

实例：构造 OP

OpDef Repository

```
class OpDefLibrary(object):
  def apply_op(self, op_name, name=None, **keywords):
    inputs, input_types, output_types, attr_protos, op_def =
    with graph.as_default(), ops.name_scope(name) as scope:
      return graph.create_op(op_name, inputs, output_types, name=scope,
                  input_types=input_types, attrs=attr_protos, op_def=op_def)
```

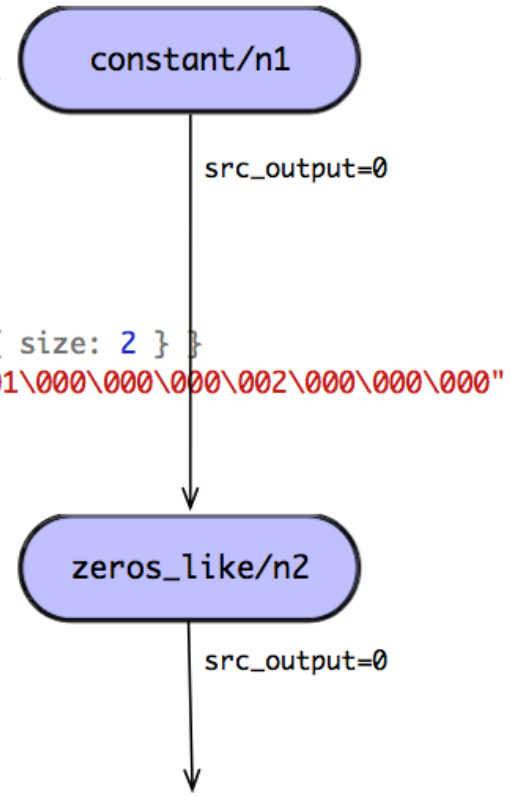


Graph

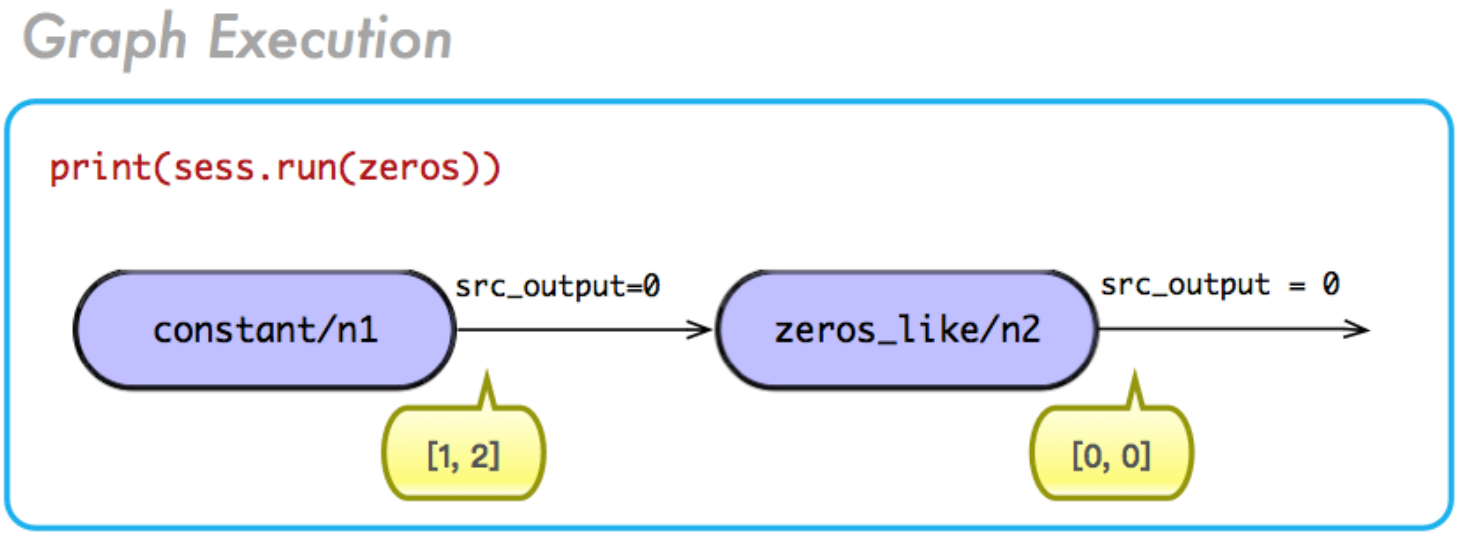
```
class Graph(object):
  def create_op(self, op_type, inputs, dtypes, input_types=None,
                name=None, attrs=None, op_def=None):
    node_def, control_inputs = ...
    return Operation(node_def, self, inputs=inputs, output_types=output_types,
                     control_inputs=control_inputs, input_types=input_types,
                     op_def=op_def)
```

实例：图构造

```
node {
  name: "n1"
  op: "Const"
  attr {
    key: "dtype"
    value { type: DT_INT32 }
  }
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_INT32
        tensor_shape { dim { size: 2 } }
        tensor_content: "\001\000\000\000\002\000\000\000"
      }
    }
  }
}
node {
  name: "n2"
  op: "ZerosLike"
  input: "node1"
  attr {
    key: "T"
    value { type: DT_INT32 }
  }
}
```



实例：图执行


Graph Execution
print(sess.run(zeros))
constant/n1
src_output=0
zeros_like/n2
src_output = 0
[1, 2]
[0, 0]

训练模型

1. 优化算法
2. 工作流

优化器

```
class Optimizer(object):
  """Add operations to minimize loss by updating var_list.
  """
  def minimize(self, loss, var_list):
    grads_and_vars = self.compute_gradients(loss, var_list)
    return self.apply_gradients(grads_and_vars)
```

计算梯度

```
def computea_gradients(loss, var_list):
  grads = gradients(loss, var_list, grad)
  return list(zip(grads, var_list))

def gradients(loss, var_list, grads=1):
  ops_and_grads = {}
  for op in reversed_graph(loss).topological_sort():
    grad = op.grad_fn(grad)
    ops_and_grads[op] = grad
  return [ops_and_grads.get(var) for var in var_list]
```

梯度函数

```
@ops.RegisterGradient("op_name")
def grad_func(op, grad):
  """construct gradient subgraph for an op type.
  Returns:
    A list of gradients, one per each input of op.
  """
  return cons_grad_subgraph(op, grad)
```

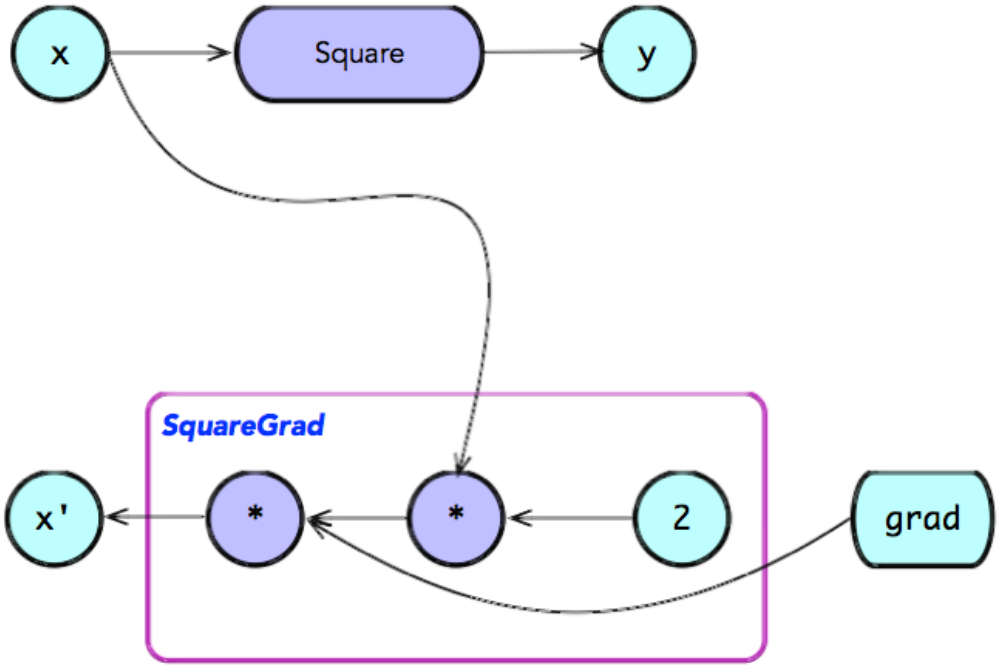
$$(y_1, y_2, ..., y_m) = f(x_1, x_2, ..., x_n)$$

$$(\partial L/\partial x_1, \partial L/\partial x_2, ..., \partial L/\partial x_n) = g\left(x_1, x_2, ..., x_n; \partial L/\partial y_1, \partial L/\partial y_2, ..., \partial L/\partial y_n\right)$$

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i} \text{ for } i = 1, 2, ..., n$$

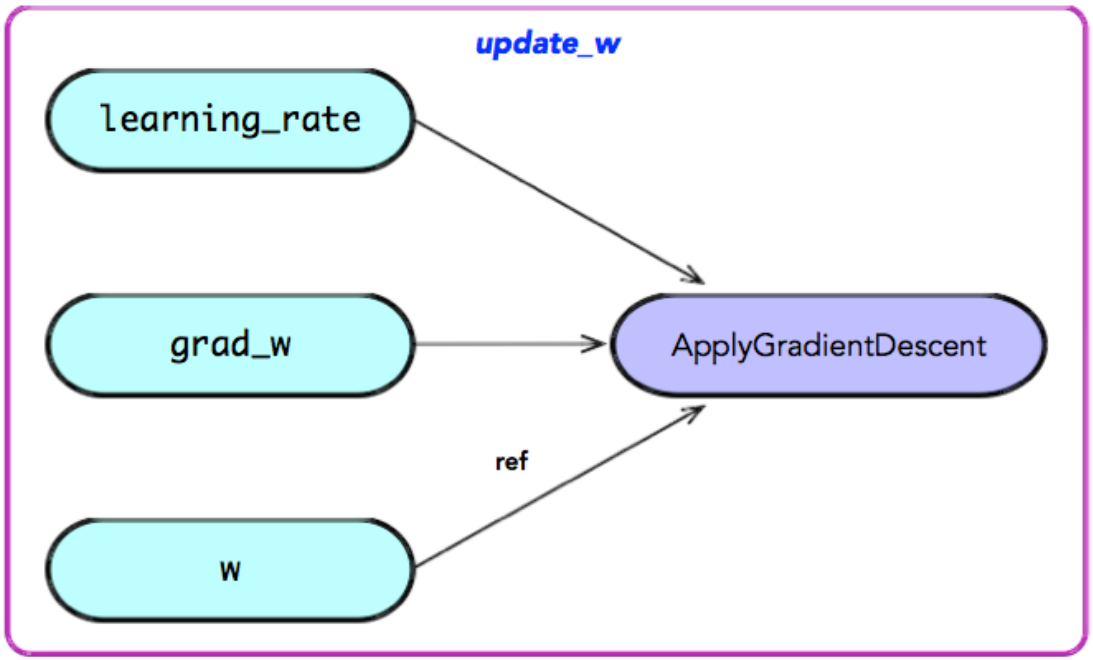
例子：SquareGrad 函数

```
@ops.RegisterGradient("Square")
def SquareGrad(op, grad):
  x = op.inputs[0]
  with ops.control_dependencies([grad.op]):
    return grad * (2.0 * x)
```

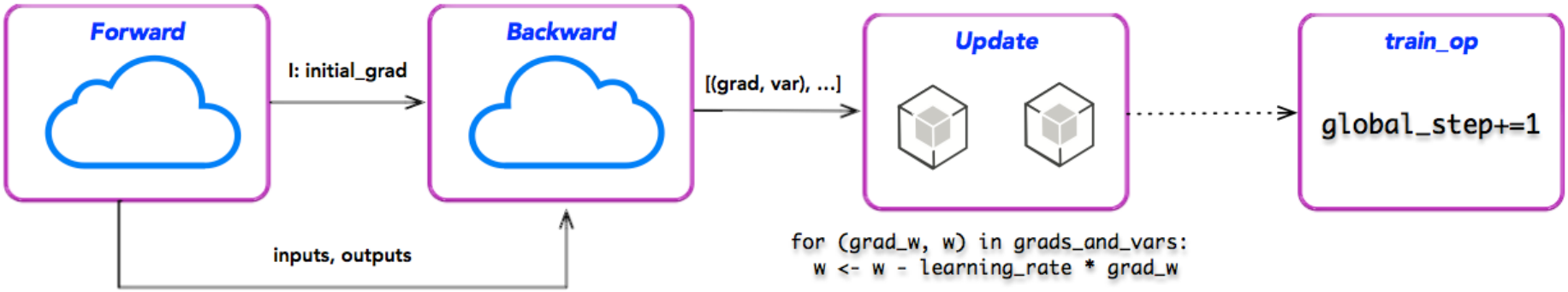


应用梯度

```
def apply_gradients(grads_and_vars, learning_rate):
  for (grad, var) in grads_and_vars:
    apply_gradient_descent(learning_rate, grad, var)
```

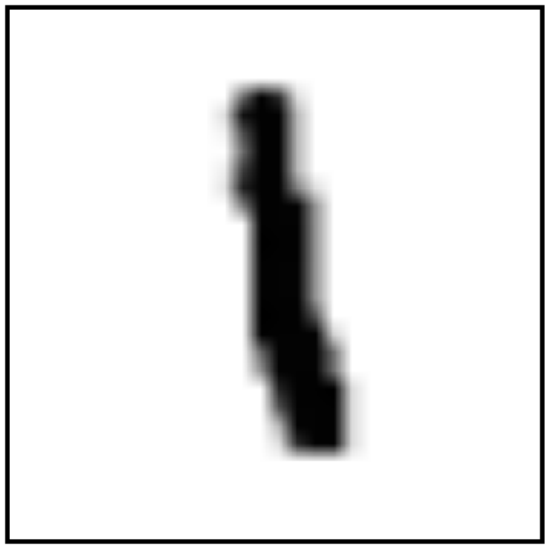


RunStep 过程


Forward
I: initial_grad
Backward
[(grad, var), ...]
Update
train_op
global_step+=1
inputs, outputs
for (grad_w, w) in grads_and_vars:
w <- w - learning_rate * grad_w

实战 Mnist

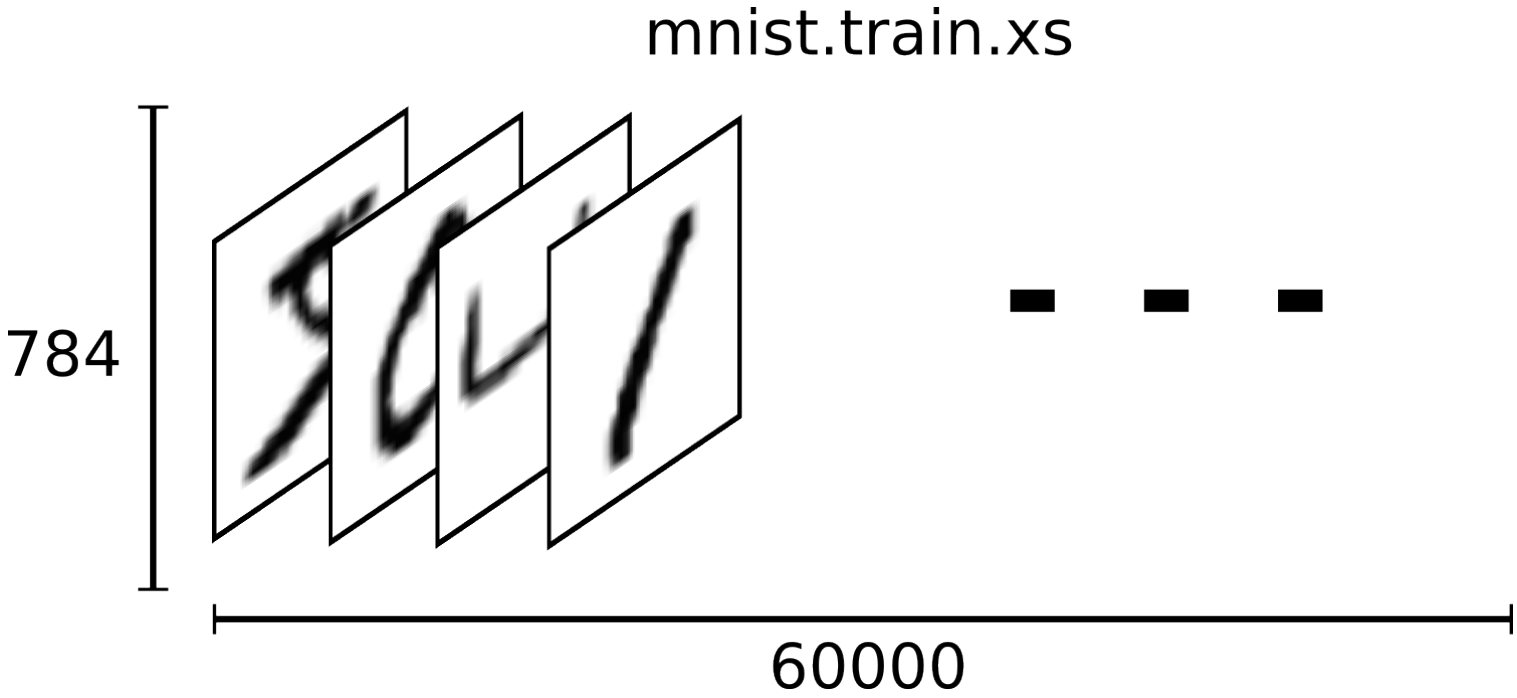
图片：28x28 = 784



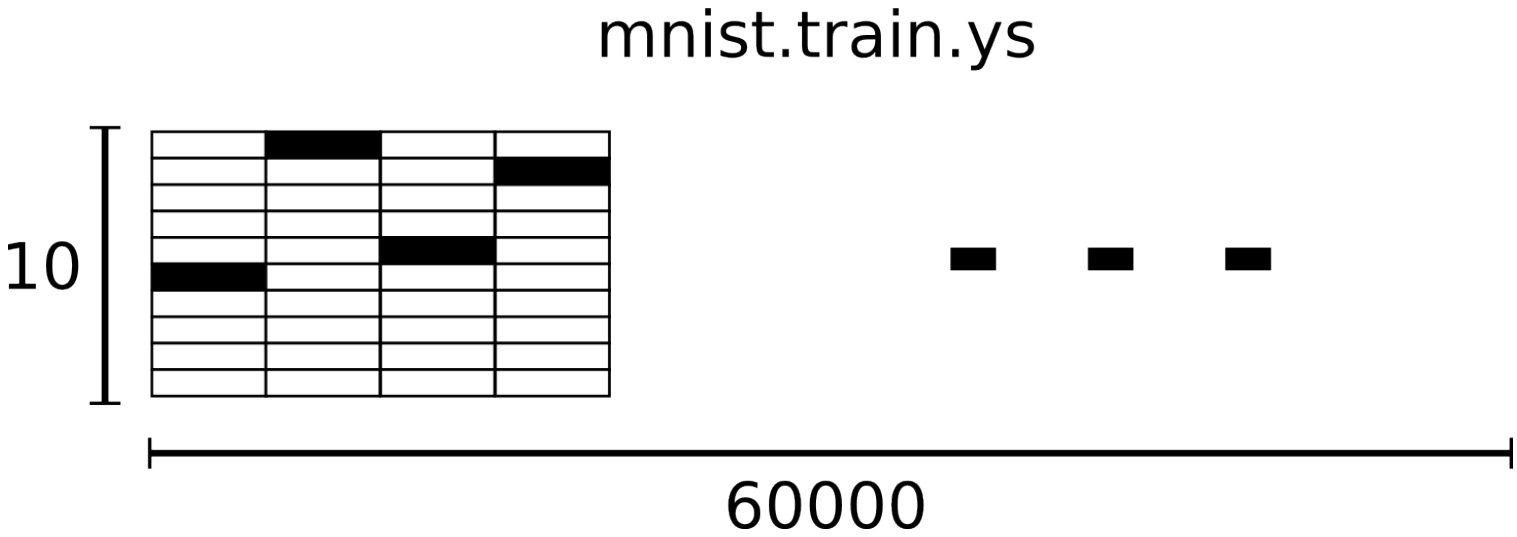
$\approx$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

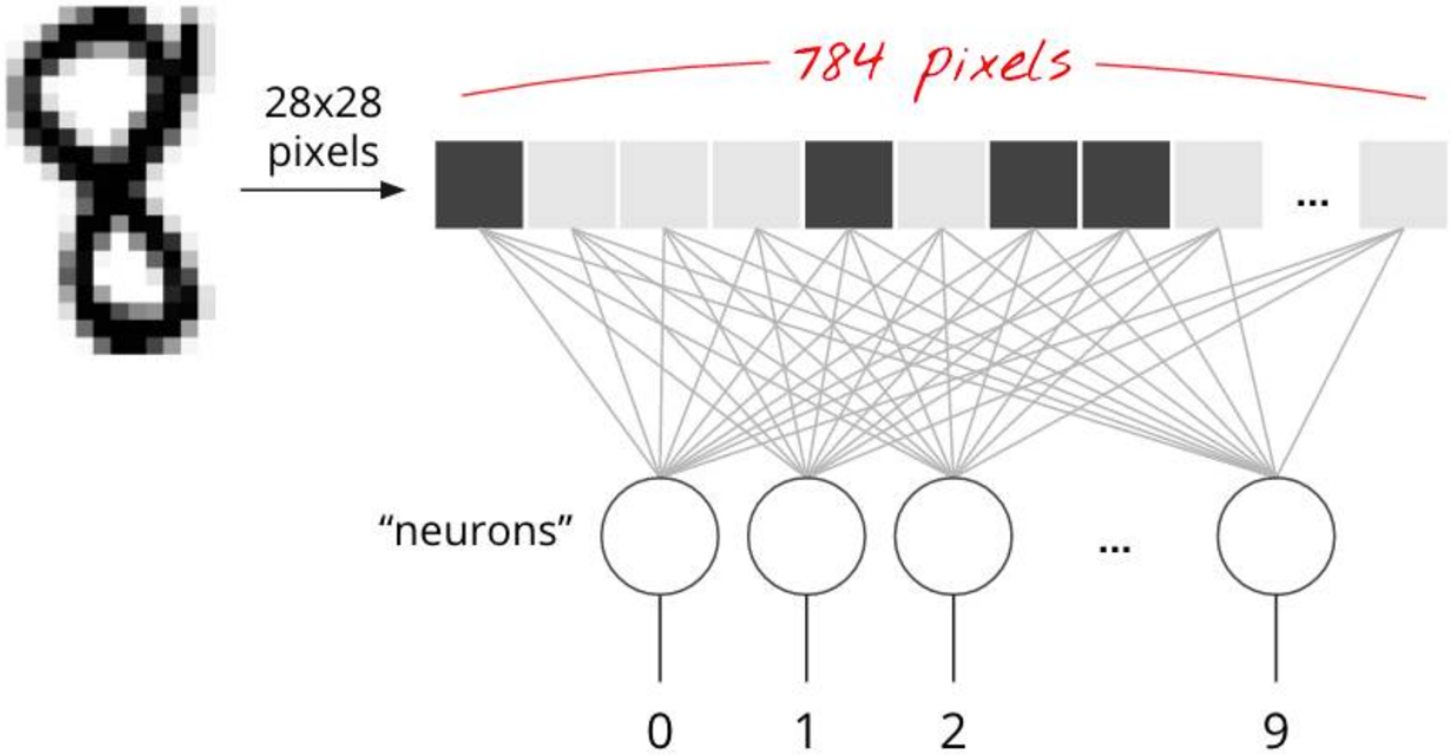
训练数据集输入：[100(*batch_size*), 784]



训练数据集输出：[100(*batch_size*), 10]



单层网络



占位符

```
# inputs
x  = tf.placeholder("float", [None, 784])

# labels
t = tf.placeholder("float", [None, 10])
```

- tf.placeholder 定义了一个占位 OP
- None 表示未确定的样本数目 (batch_size)
- Session.run 时提供 feed_dict 提供一个批次的样本数据

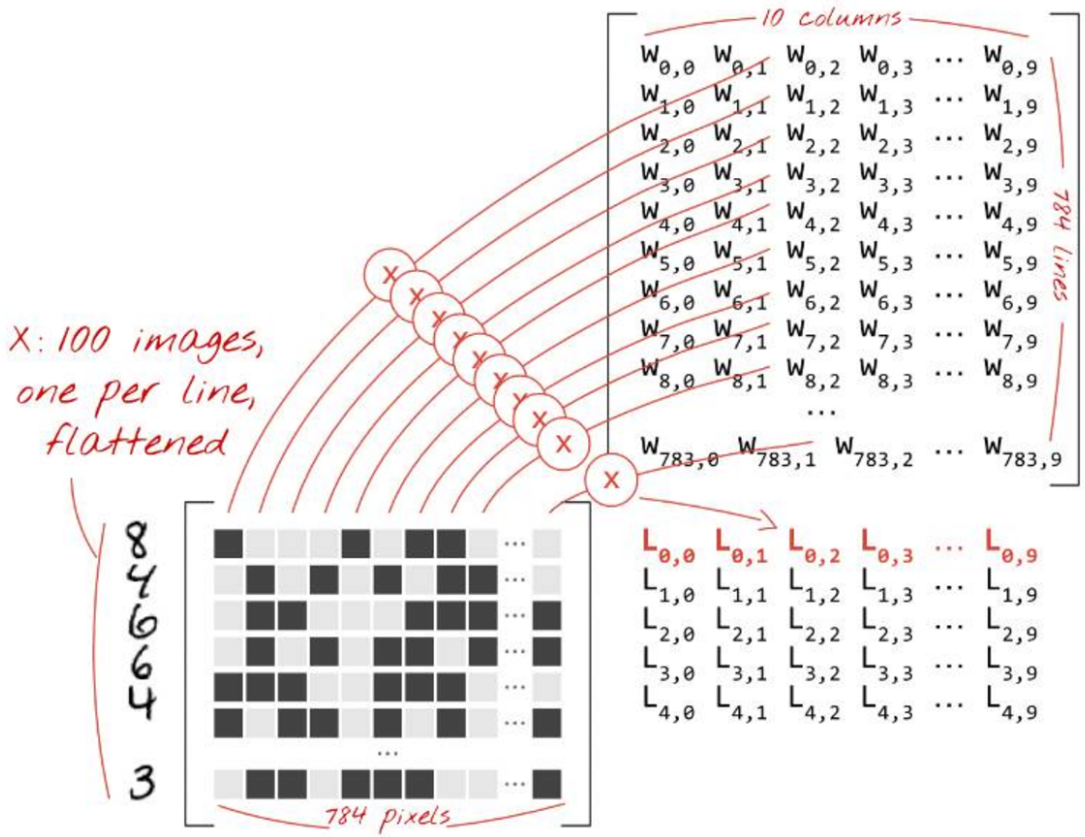
变量

```
# train parameters
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

# init_op
init_op = tf.global_variables_initializer()
```

- 变量：使用 tf.Variable 定义变量，常用于定义模型的权重和偏置
- 初始化：使用初始化 init_op 初始化所有全局变量

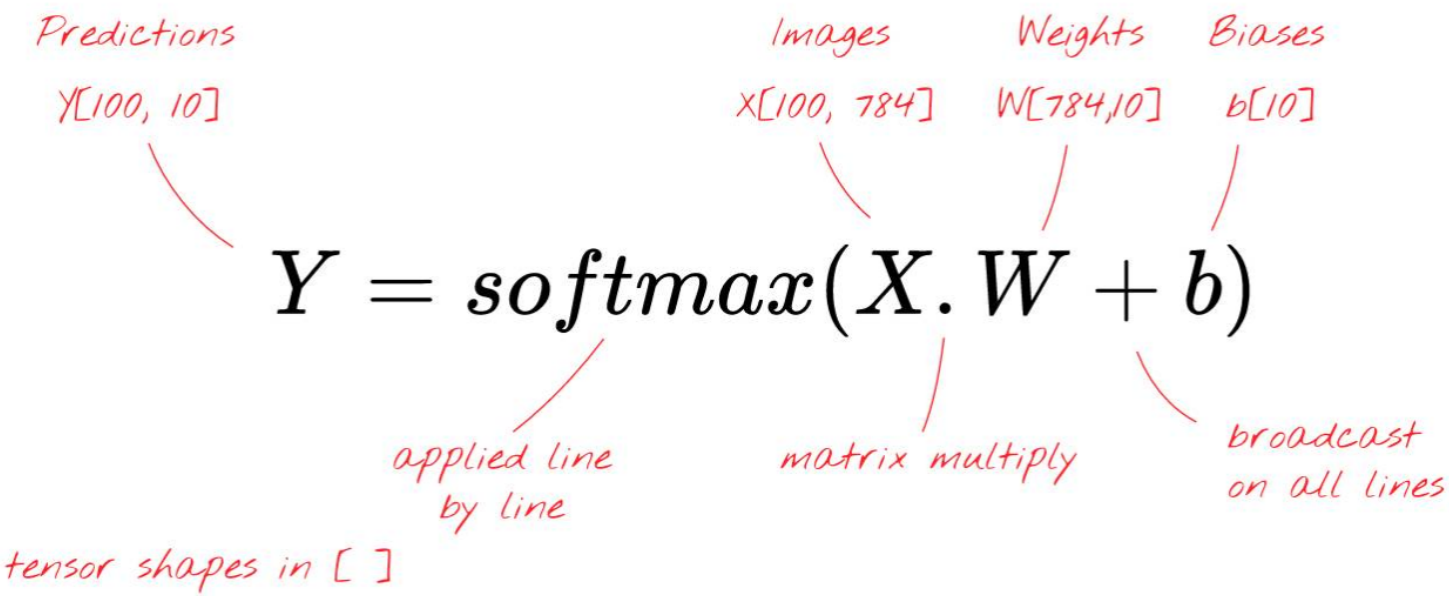
模型：线性和

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```



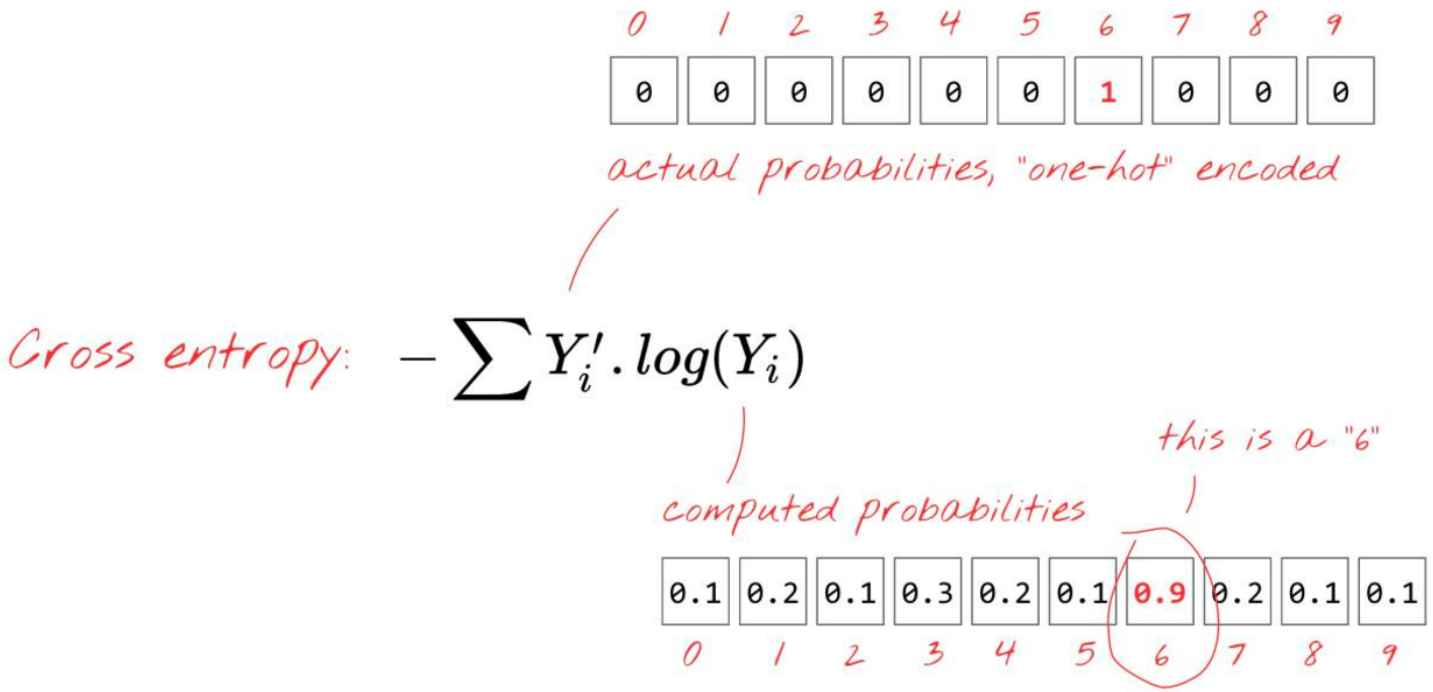
模型: Softmax

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```



损失函数

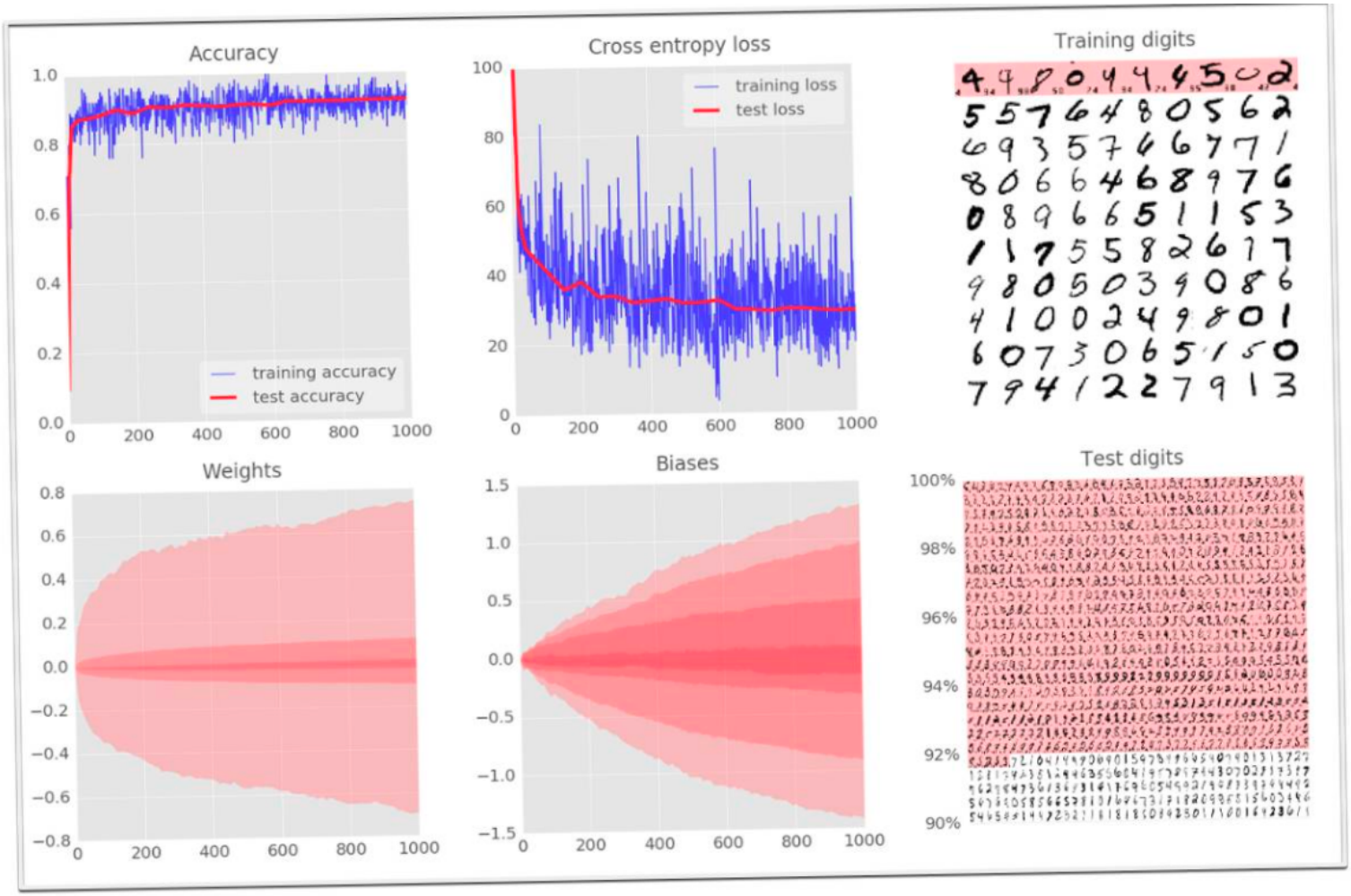
```
cross_entropy = -tf.reduce_sum(t * tf.log(y))
```



执行训练

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(t, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

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

  for step in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, t: batch_ys})

    if step % 10 == 0:
      acc, loss = sess.run([accuracy, cross_entropy],
        feed_dict={x: batch_xs, t: batch_ys})

    if step % 100 == 0:
      acc, loss = sess.run([accuracy, cross_entropy],
        feed_dict={x: mnist.test.images, t: mnist.test.labels})
```

精度：约为 92%


Accuracy
training accuracy
test accuracy
Cross entropy loss
training loss
test loss
Training digits
Weights
Biases
Test digits
92%

文献

文献

- TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Google Inc.
- TensorFlow: A System for Large-Scale Machine Learning, Google Inc.
- tensorflow.org

Q&A

联系我

- Github: https://github.com/horance-liu
- 简书: http://www.jianshu.com/users/49d1f3b7049e

致谢

Thanks