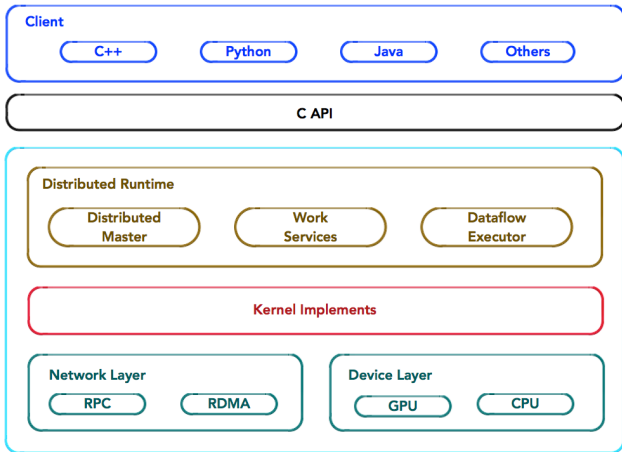


内容

- 1 架构概述
- 2 编程模型
- 3 训练模型
- 4 实战 Mnist
- 5 文献

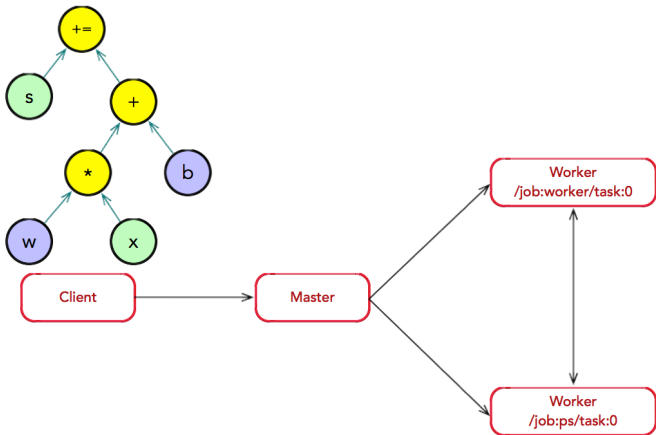
系统架构



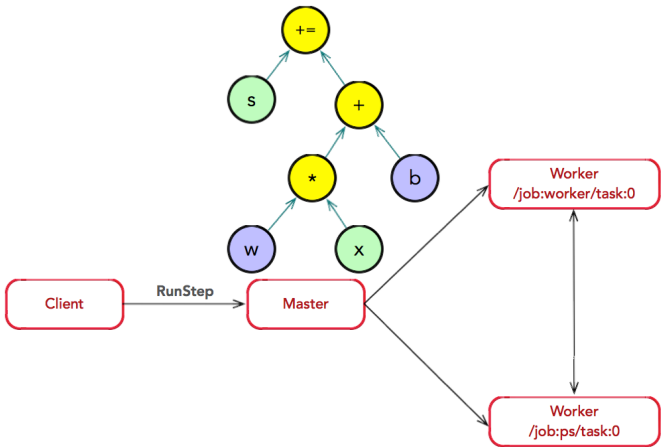
设计原则

- **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

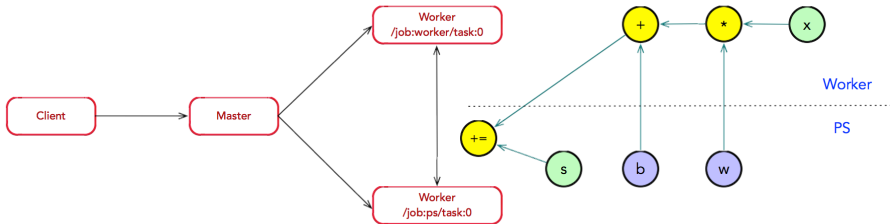
构造计算图



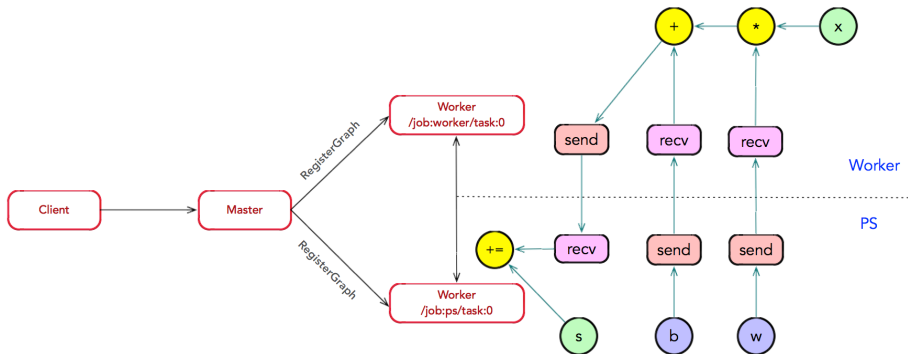
执行计算图



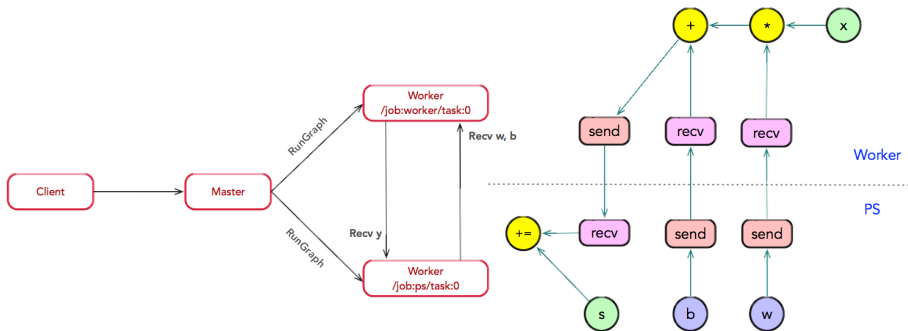
图分解



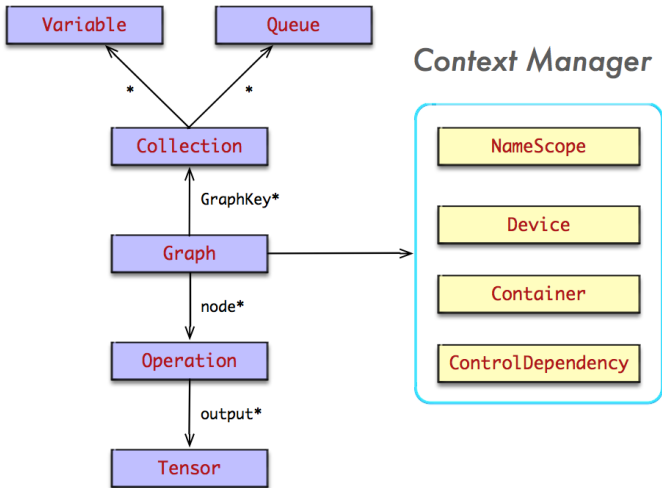
注册图



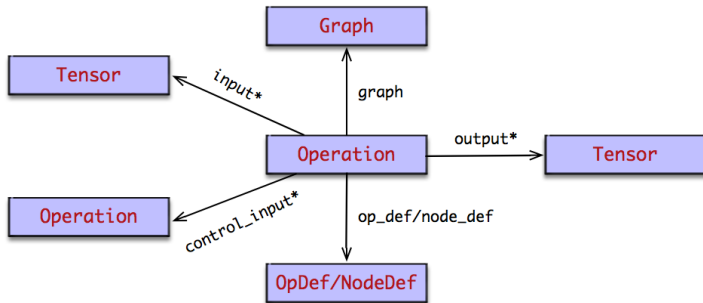
执行图



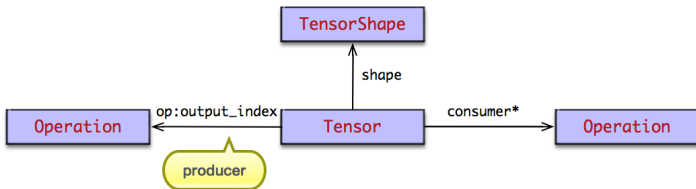
$$Graph = Set\{OP\} + Set\{Tensor\}$$



OP: 抽象计算



Tensor: 承载数据

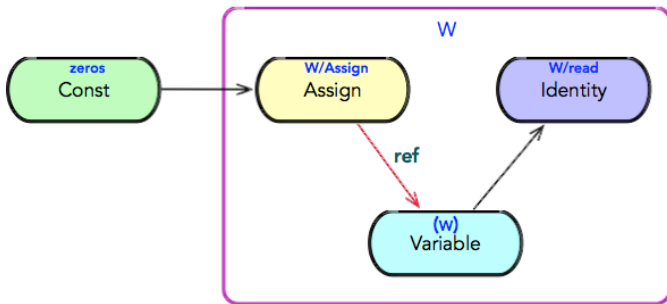


有向边

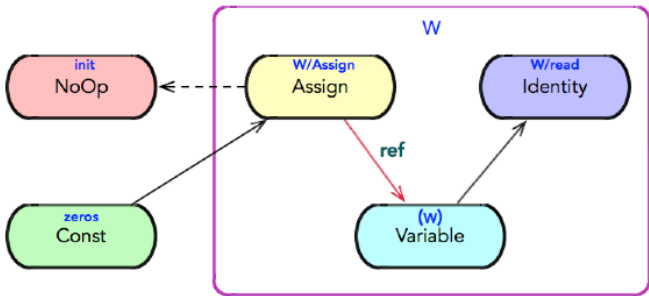
两种类型

- **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

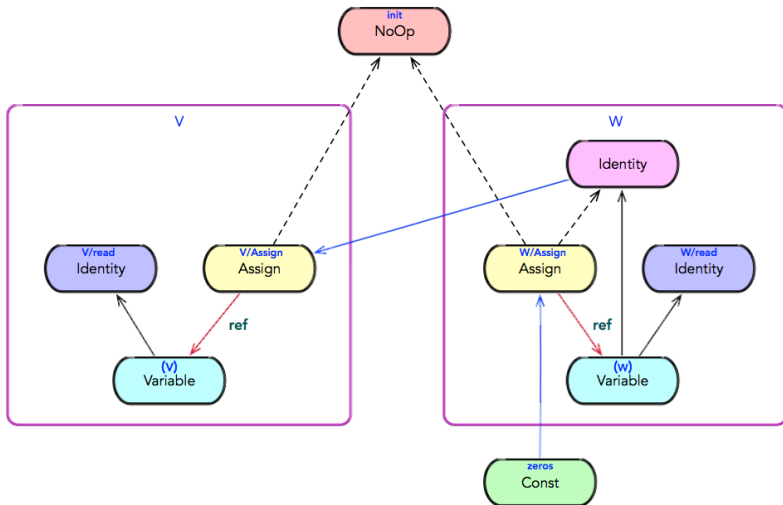
初始化模型



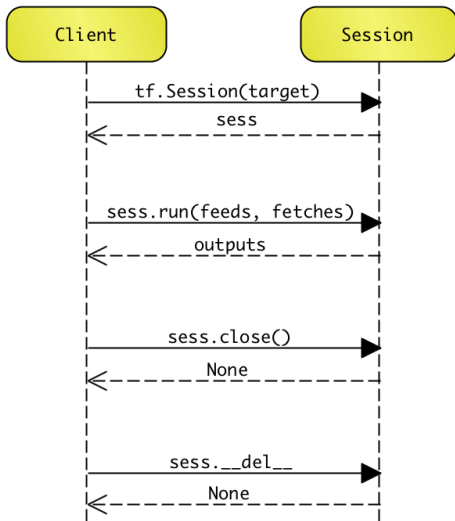
探秘 init_op



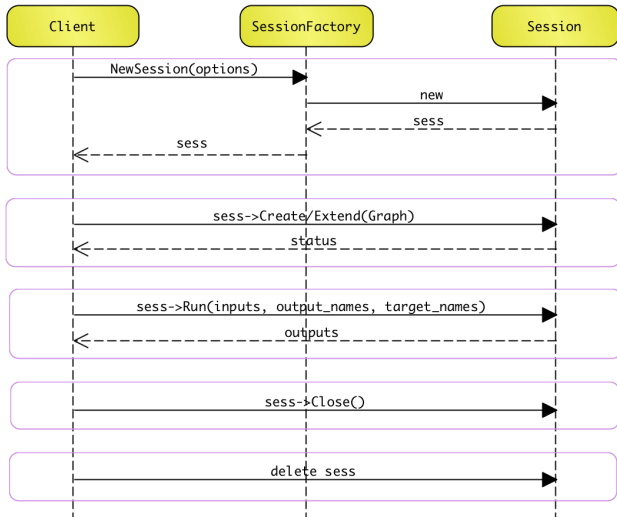
初始化依赖



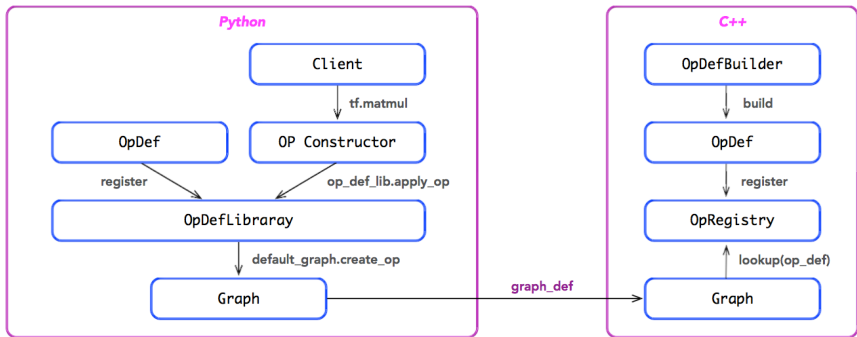
生命周期: Python



生命周期: C++



图构造与传递



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)
```

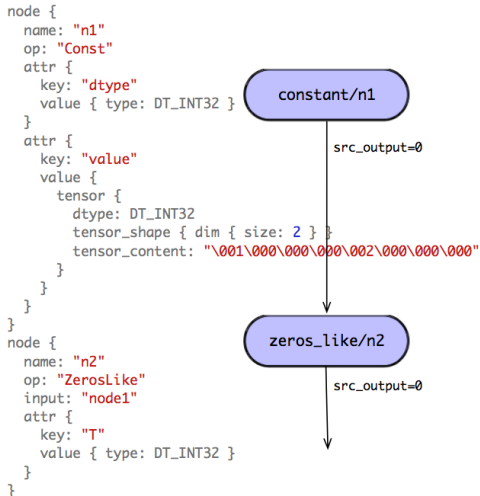


Graph

```
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)
```



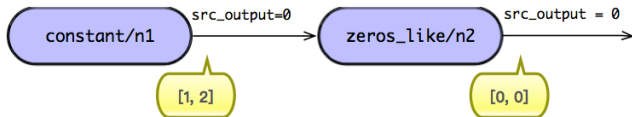
实例：图构造



实例：图执行

Graph Execution

```
print(sess.run(zeros))
```



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)
```


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)
```

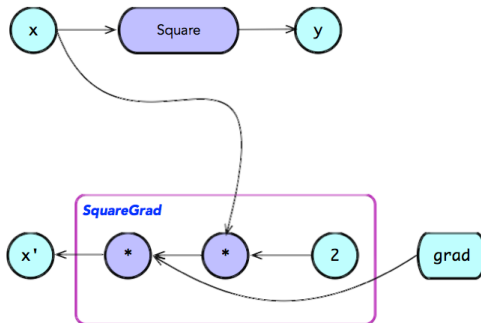
$$(y_1, y_2, \dots, y_m) = f(x_1, x_2, \dots, x_n)$$

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

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

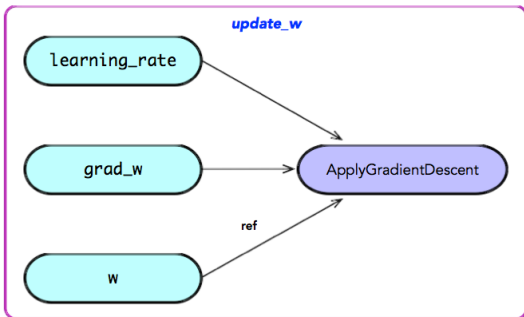
例子：SquareGrad 函数

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

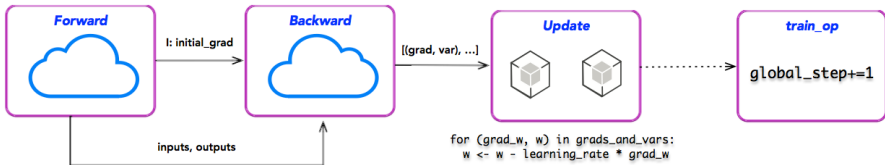


应用梯度

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



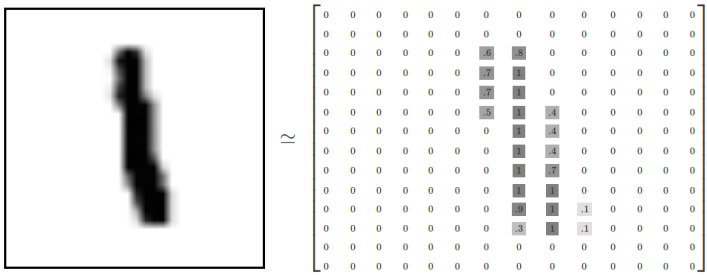
RunStep 过程



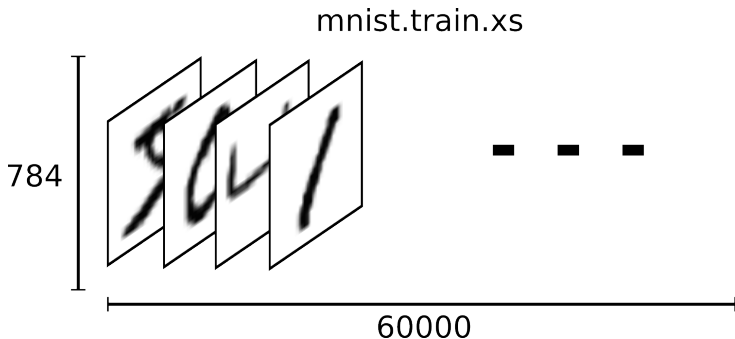
实战 Mnist

- 1 提出问题
- 2 单层感知器
- 3 多层感知器
- 4 调整超参
- 5 卷积神经网络

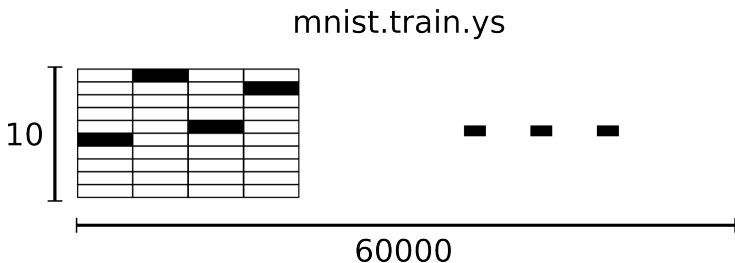
图片：28x28 = 784



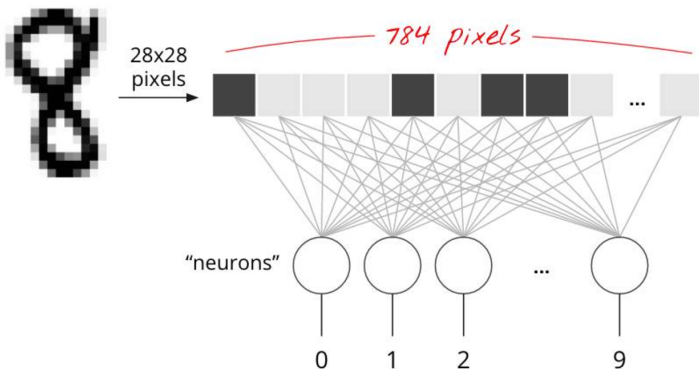
训练数据集输入: $[100(batch_size), 784]$



训练数据集输出：[100(batch_size), 10]



单层网络模型



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

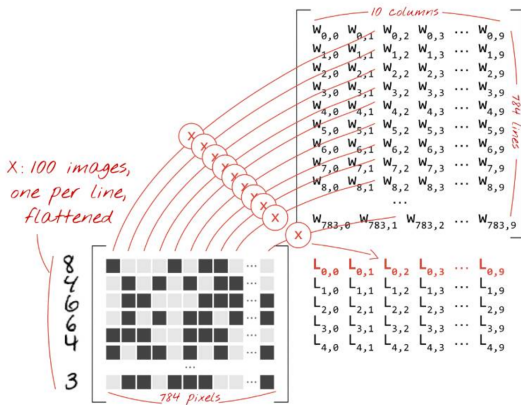
```
# train parameters
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

# 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

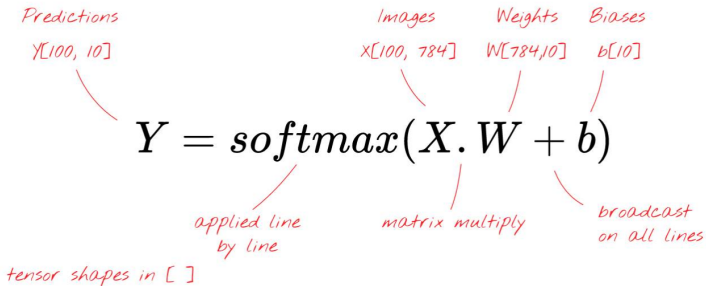
模型：线性

```
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))
```

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy: $-\sum Y'_i \cdot \log(Y_i)$

computed probabilities

0.1	0.2	0.1	0.3	0.2	0.1	0.9	0.2	0.1	0.1
0	1	2	3	4	5	6	7	8	9

this is a "6"

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))

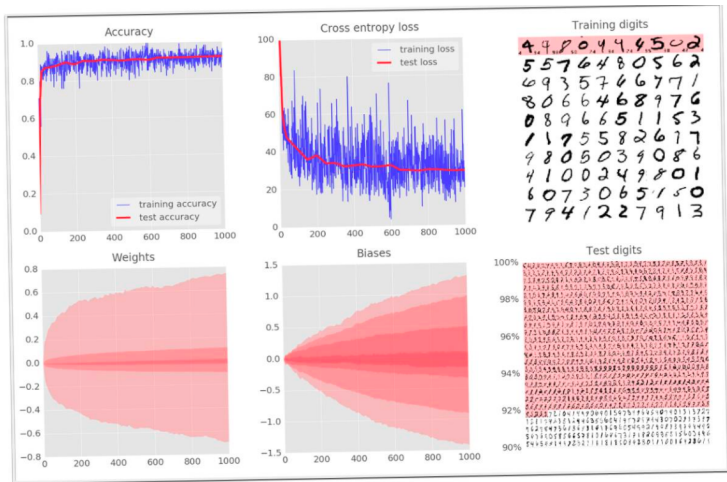
with tf.Session() as sess:
    sess.run(init_op)

    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%



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](https://www.tensorflow.org)

致谢

Q&A



联系我

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

致谢

Thanks