

TensorFlow计算图浅介

吴建军

目录

- TensorFlow概述
- HelloWorld例子
- 基本类型与结构
- TensorBoard
- 计算图的存取
- 进阶问题一览

wujianjunml@outlook.cn

TensorFlow概述

wujianjunml@outlook.cn

TensorFlow概述

- TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms



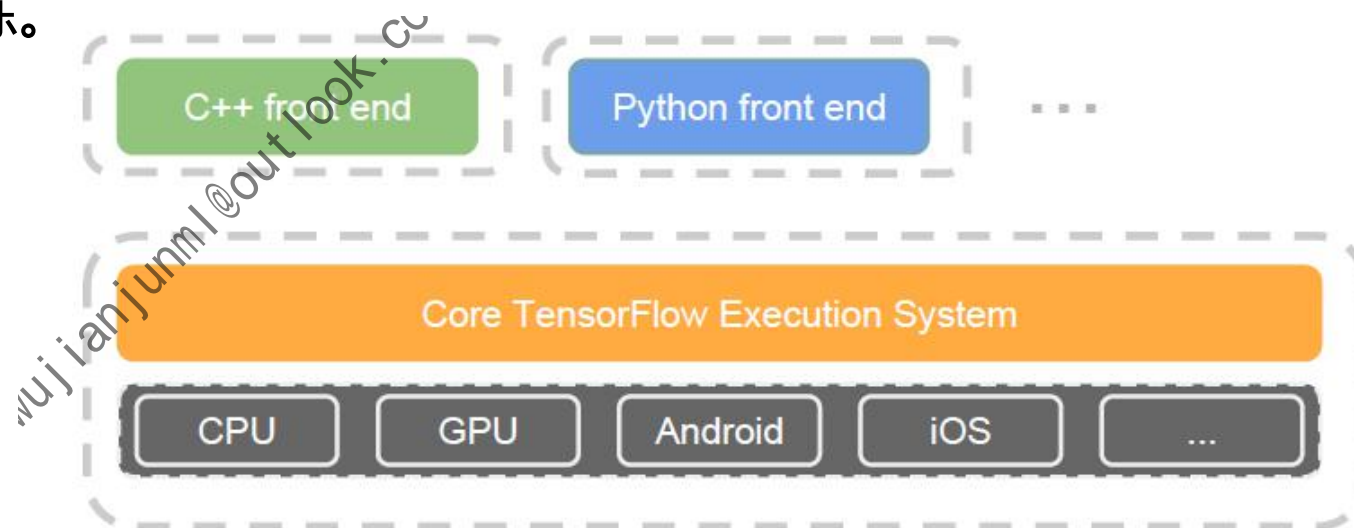
TensorFlow

TensorFlow is an end-to-end open source platform for machine learning

- 2015年11月由Google开源。
- 2016年4月发布0.8版本。
- 2017年2月发布1.0版本。
- 2018年6月Keras成Tensorflow的一部分。
- 预计在2019年底发布正式的2.0版本。

TensorFlow概述

- Tensorflow核心由C++编写，同时提供多种前端语言。
- Tensorflow可以运行在多种设备上。
- Tensorflow可以支持分布式训练。



- Tensorflow使用SWIG联通前端多语言编程环境与后端C/C++实现系统。
- SWIG是一种让脚本语言调用C/C++接口的工具，支持Perl, Python, Ruby等很多脚本。并且对C++的继承、多态、模板，STL等都有较好的支持。

TensorFlow概述

- Tensorflow缺点多多(入门难度大, 系统庞大复杂等), 但仍然是最流行的深度学习框架。

pytorch / pytorch

Watch 1,330 Star 31,736 Fork 7,841

Code Issues 3,298 Pull requests 891 Projects 5 Wiki Security Insights

Tensors and Dynamic neural networks in Python with strong GPU acceleration <https://pytorch.org>

neural-network autograd gpu numpy deep-learning tensor python machine-learning

20,888 commits 3,509 branches 23 releases 1,171 contributors View license

tensorflow / tensorflow

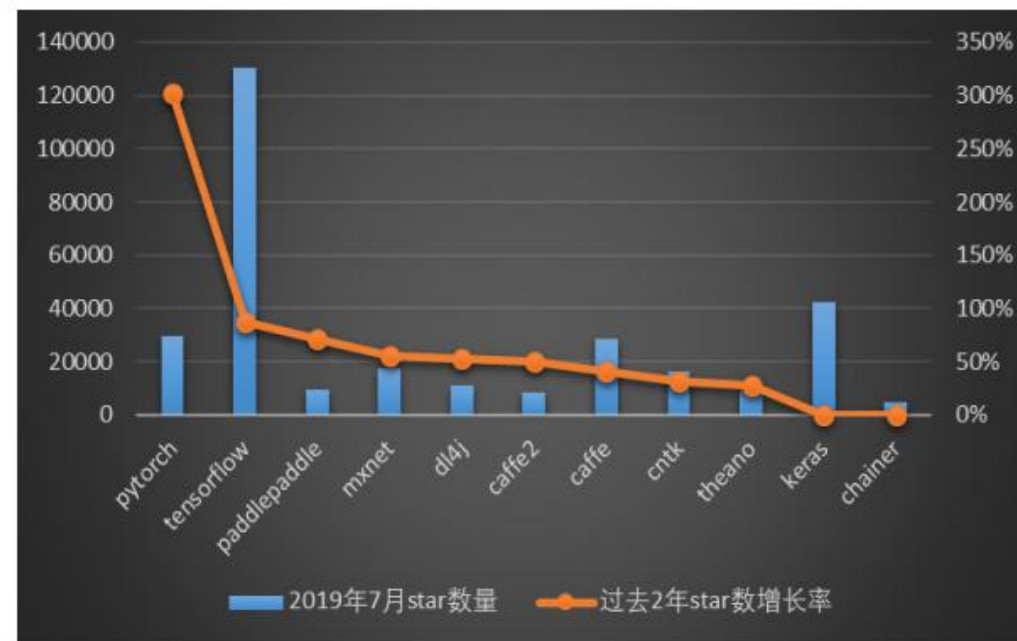
Watch 8,584 Star 134,474 Fork 77,444

Code Issues 2,461 Pull requests 217 Projects 1 Security Insights

An Open Source Machine Learning Framework for Everyone <https://tensorflow.org>

tensorflow machine-learning python deep-learning deep-neural-networks neural-network ml distributed

65,521 commits 44 branches 93 releases 2,189 contributors Apache-2.0



HelloWorld例子

HelloWorld例子

- TensorFlow使用graph来表示计算任务，图中的节点称为operation，边称为tensor。
- 在会话 (Session)中执行图的计算，会话将计算分发到CPU、GPU等设备执行。
- 使用tensor表示数据。
- 通过变量 (Variable) 维护状态。
- 使用feed 和fetch可以为任意的操作赋值或者从其中获取数据。

HelloWorld例子

- Tensorflow采用图的方式表达运算:

```
import tensorflow as tf
```

```
b = tf.Variable(tf.zeros([100]))
```

```
W = tf.Variable(tf.random_uniform([784,100],-1,1))
```

```
x = tf.placeholder(name="x")
```

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

```
C = [...]
```

```
# 100-d vector, init to zeroes
```

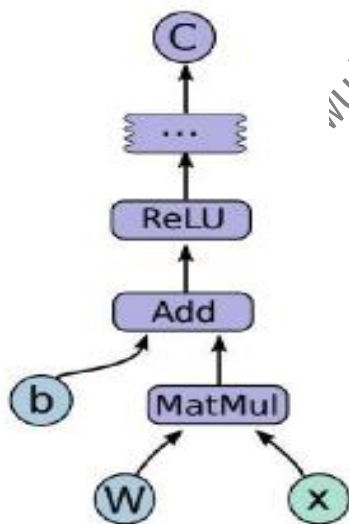
```
# 784x100 matrix w/rnd vals
```

```
# Placeholder for input
```

```
# Relu(Wx+b)
```

```
# Cost computed as a function
```

```
# of Relu
```



$$C = \text{ReLU}(W \cdot x + b)$$

HelloWorld例子

- 首先，我们介绍一个例子。
- 下面的代码使用tensorflow训练一个线性回归模型：

```
# 1. 首先，拿到数据，这里是随机生成的
n_samples = 100
X_data = np.linspace(-1, 1, n_samples)
noise = np.random.normal(0, 0.5, n_samples)
y_data = 5 * X_data + noise
X_data = np.reshape(X_data, (n_samples, 1)) # 注意这里一定要记得 reshape , tensorflow 中任何数据必须明确给出 shape
y_data = np.reshape(y_data, (n_samples, 1))
# 2. 然后，定义计算图
```

HelloWorld例子

```
g = tf.Graph()
with g.as_default():
    # a. 网络输入
    x = tf.placeholder(tf.float32, shape=(None, 1))
    y = tf.placeholder(tf.float32, shape=(None, 1))
    # b. 预测结构
    w = tf.get_variable("weight", (1, 1), initializer = tf.random_normal_initializer())
    b = tf.get_variable("bais", (1,1), initializer = tf.constant_initializer(0.0))
    y_pred = tf.add(tf.matmul(x, w), b)
    # c. loss计算
    loss = tf.reduce_sum(tf.square(y - y_pred))
    tf.summary.scalar("loss", loss) # 这是在添加可视化的节点
    # d. 优化节点
    opt = tf.train.GradientDescentOptimizer(0.001)
    operation = opt.minimize(loss)
    # e. 通过以下三个节点结束图的定义
    summary = tf.summary.merge_all()
    init = tf.initialize_all_variables()
    saver = tf.train.Saver()
```

HelloWorld例子

3.最后，运行计算图

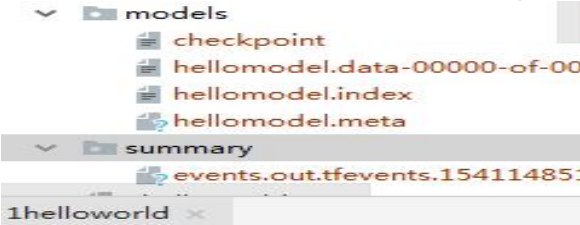
```
with tf.Session(graph=g) as sess:
    # a. 初始化
    sess.run(init) # 第一步一定是运行初始化
    writer = tf.summary.FileWriter("./summary/", sess.graph) # 负责写入可视化信息的 writer
    # b. 开始迭代训练
    random_permutation = np.arange(0, n_samples, 1)
    batch_num = 32
    num_step = int(n_samples / batch_num)
    for epoch in range(500): # 迭代 500 轮
        np.random.shuffle(random_permutation) # 每一轮训练时，注意随机打散数据
        loss_val_list = []
        for step in range(num_step):
            # 随机抽取一个batch的数据，一般数据量很大，不能全部装进显存，所以一个batch一个batch地 feed 到模型
            if step == num_step - 1:
                batch_data = X_data[random_permutation[step * batch_num:],:]
                batch_label = y_data[random_permutation[step * batch_num:],:]
            else:
                batch_data = X_data[random_permutation[step*batch_num:(step + 1)*batch_num],:]
                batch_label = y_data[random_permutation[step*batch_num:(step + 1)*batch_num],:]
            sess.run(operation, feed_dict = {x: batch_data, y: batch_label}) # 更新模型参数
        loss_val_list.append(sess.run(loss, feed_dict = {x: X_data, y: y_data})) # 计算当前loss
```

HelloWorld例子

```
loss_val = np.mean(loss_val_list)
result = sess.run(summary, feed_dict={x: X_data, y: y_data}) # 计算 summary
writer.add_summary(result, epoch)
print("loss_val = %s" % (loss_val))
# 训练结束后, 打印结果
print(sess.run(w))
print(sess.run(b))
saver.save(sess, './models/hellomodel') # 保存模型
```

- 我们看看运行这段程序后的输出

- 可以看到最后基本还原了模型。



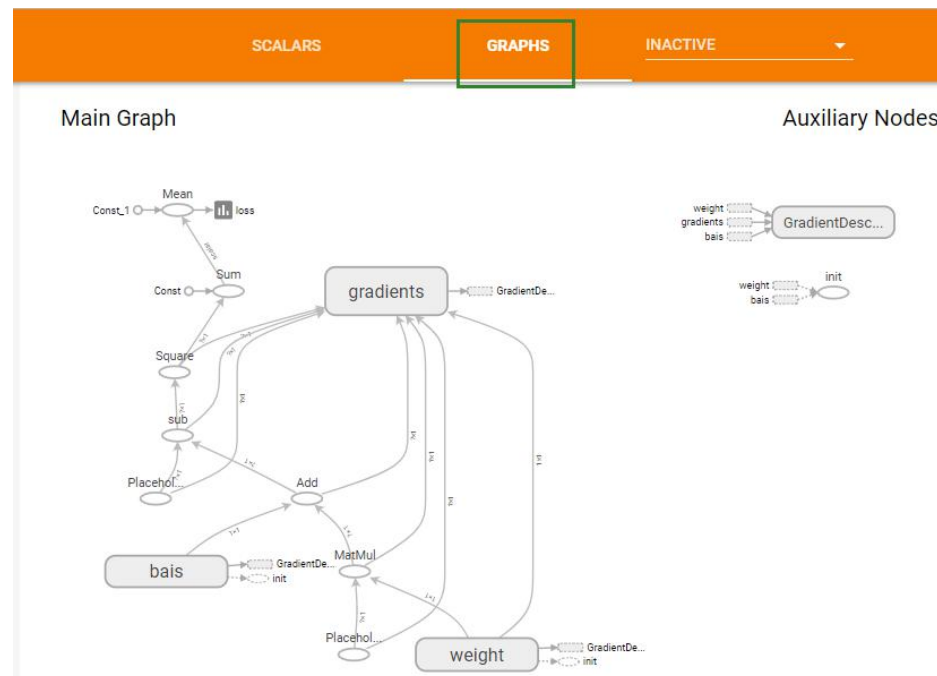
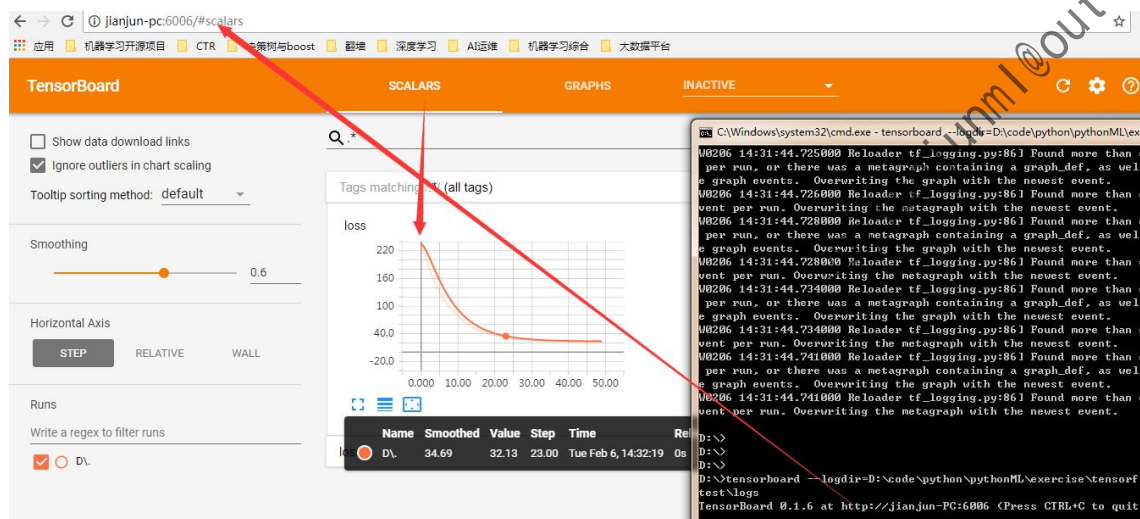
```
1helloworld x
loss_val = 23.701471
loss_val = 23.692392
loss_val = 23.691881
loss_val = 23.691446
loss_val = 23.69201
loss_val = 23.695541
loss_val = 23.69114
loss_val = 23.692614
loss_val = 23.691292
loss_val = 23.691545
loss_val = 23.692804
loss_val = 23.691696
[[4.966921]]
[[-0.00787007]]
```

HelloWorld例子

- 接着我们在看看可视化输出，在命令行中执行命令：

```
d:\>tensorboard --logdir=D:\code\python\pythonML\ml\tensorflow_test\basic_test\summmary
```

- 然后在浏览器中输入提示的地址，就可以看见下面的图形化界面：

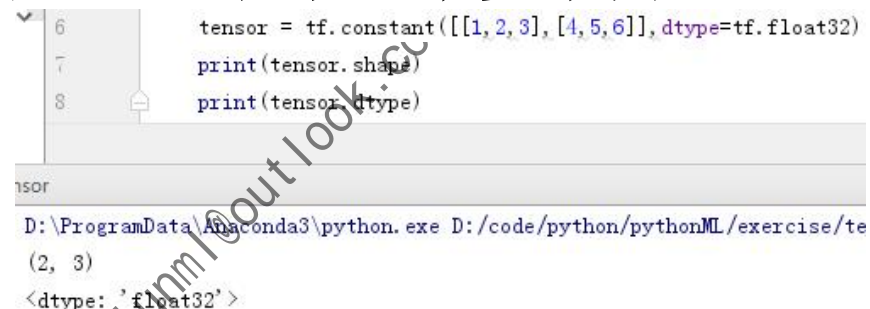


基本类型与结构

基本类型与结构

- tf.Tensor

- 张量(Tensor)是tensorflow的基础，表示一个多维向量。



The screenshot shows a Jupyter Notebook cell with the following code:

```
tensor = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)
print(tensor.shape)
print(tensor.dtype)
```

The output of the cell is:

```
tensor
D:\ProgramData\Anaconda3\python.exe D:/code/python/pythonML/exercise/te
(2, 3)
<dtype: 'float32'>
```

- tf.Operation

- 一个Operation就是Graph中的一个计算节点。其接收零个或者多个Tensor对象作为输入，然后产生零个或者多个Tensor对象作为输出。下面的操作都将产生对应的tf.Operation，并在计算图中增加对应的节点：
 - tf.placeholder, tf.constant, tf.ones, tf.Variable, tf.assign, tf.add, tf.fill
 - tf.train.Saver(), tf.train.Optimizer.minimize,
 - tf.summary.scalar, tf.summary.merge_all,

基本类型与结构

- `tf.Variable`

- `Variable`是代表一个可修改的张量，在训练过程中会自动更新或优化，常用于模型参数，在定义时需要指定初始值。
- 创建变量有两种方式`tf.get_variable()` , `tf.Variable()`。注意`Variable`是个class。

```
29 g = tf.Graph()
30 with g.as_default():
31     counter = tf.Variable(10, name="counter")
32     constant = tf.Variable([1, 2, 3], name="constant")
33     weights = tf.Variable(tf.truncated_normal([3, 2], stddev=1.0, name='weights'))
34     biases = tf.Variable(tf.zeros([3]), name='biases')
35     init_op = tf.initialize_all_variables()
36 with tf.Session(graph=g) as sess:
37     sess.run(init_op)
38     print(sess.run(counter))
39     print(sess.run(constant))
40     print(sess.run(weights))
41     print(sess.run(biases))
```

varibael_test

```
10
[1 2 3]
[[-0.99679381  0.66337514]
 [ 0.35385132 -0.29085767]
 [ 0.41075942 -0.85163862]]
[ 0.  0.  0.]
```

```
44 g = tf.Graph()
45 with g.as_default():
46     my_variable = tf.get_variable("my_variable", [2, 3], dtype = tf.int32, initializer=tf.zeros_initializer)
47     other_variable = tf.get_variable("other_variable", dtype=tf.float32, initializer = tf.constant([23.0, 42.0]))
48     init_op = tf.initialize_all_variables()
49 with tf.Session(graph=g) as sess:
50     sess.run(init_op)
51     print(sess.run(my_variable))
52     print(sess.run(other_variable))
```

varibael_test

```
[[0 0 0]
 [0 0 0]]
[ 23.  42.]
```

基本类型与结构

- tf.Graph

- 一旦import Tensorflow成功后，立即就有一个默认图生成，也可以通过tf.Graph()函数来生成新的计算图，使用tf.Graph.as_default()的上下文管理器，它能够在这个上下文里面覆盖默认的图。

- Graph 有如下几个重要函数：

- device: 返回一个上下文管理器指定新建操作默认使用的设备。

```
g = tf.Graph()
# 指定计算运行的设备
with g.device('/gpu:0'):
    result = a + b
```

```
if __name__ == "__main__":
    c = tf.constant(3.0)
    print(c.graph == tf.get_default_graph())

    g = tf.Graph()
    with g.as_default():
        print(g == tf.get_default_graph())
        c = tf.constant(3.0)
        print(c.graph == g)
```

basicExercise (1)

```
D:\ProgramData\Anaconda3\python.exe D:/code/py
True
True
True
```

基本类型与结构

- tf.Session

- Session提供了Operation执行和Tensor求值的环境。

- Session有如下几个重要函数：

- run(fetches, feed_dict=None, options=None, run_metadata=None)

```
if __name__ == '__main__':
    g = tf.Graph()
    with g.as_default():
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
        c = tf.matmul(a, b, name='c') # c is a tf.Tensor
        c_operation = g.get_operation_by_name('c') # c_operation is a tf.Operation

    with tf.Session(graph=g) as sess:
        print(sess.run(c)) # [[ 22.  28.], [ 49.  64.]], run 一个 tensor 返回一个 numpy
        print(c_operation.run()) # None, run 一个 Operation 则返回 None
        print(c_operation.outputs[0]) # Tensor("c:0", shape=(2, 2), dtype=float32)
```

5_session x

```
[[22. 28.]
 [49. 64.]]
None
Tensor("c:0", shape=(2, 2), dtype=float32)
```

基本类型与结构

- collection
 - 可以通过collection来管理不同类别的资源。
 - `add_to_collection(name, value)`
往指定name的collection中添加值。
 - `get_collection(name, scope=None)`
从指定name的collection中返回值。

```
g = tf.Graph()
with g.as_default():
    w1 = tf.Variable([1, 2, 3], dtype=tf.float32, name='w1')
    tf.add_to_collection('c1', w1)
    tf.add_to_collection('c1', w1)
    params = tf.get_collection('c1') # 返回的是 collection 的 copy
    one = tf.constant([1, 1, 1], dtype=tf.float32)
    new_value = tf.add(params[0], one)
    update = tf.assign(params[0], new_value) # 对集合中某个 Variable 赋值
    init = tf.global_variables_initializer()

with tf.Session(graph=g) as sess:
    sess.run(init)
    params = tf.get_collection('c1') # 返回的是 collection 的 copy
    print([e.name for e in params]) # 注意 w1 重复了两次
    print(sess.run(params[0])) # 更新前的值
    sess.run(update) # 更新
    print(sess.run(params[0]))
    del params[:] # 删除的是 copy, 不会删除图中的 variable
    print(sess.run(tf.get_collection('c1')[0])) #
    print(sess.run(tf.get_collection('c1')[1]))
```

collection

```
['w1:0' 'w1:0']
[ 1.  2.  3.]
[ 2.  3.  4.]
[ 2.  3.  4.]
[ 2.  3.  4.]
```

TensorBoard

wujianjunm1@outlook.com

TensorBoard

- TensorBoard是tensorflow的可视化工具，可以将graph中的summary做图形化展示。
- tf.summary
 - A summary node is an operation that will write information about particular values of interest to a predefined directory, in a format understood by the TensorBoard server.
 - tf.summary.scalar返回一个字符串类型的标量tensor，字符串取值就是给定tensor的取值的protocol buffer。
- tf.summary.FileWriter负责将summaries 写入磁盘文件。
 - 其有一个常用函数：add_summary将summary的protocol buffer打包写到event file中。

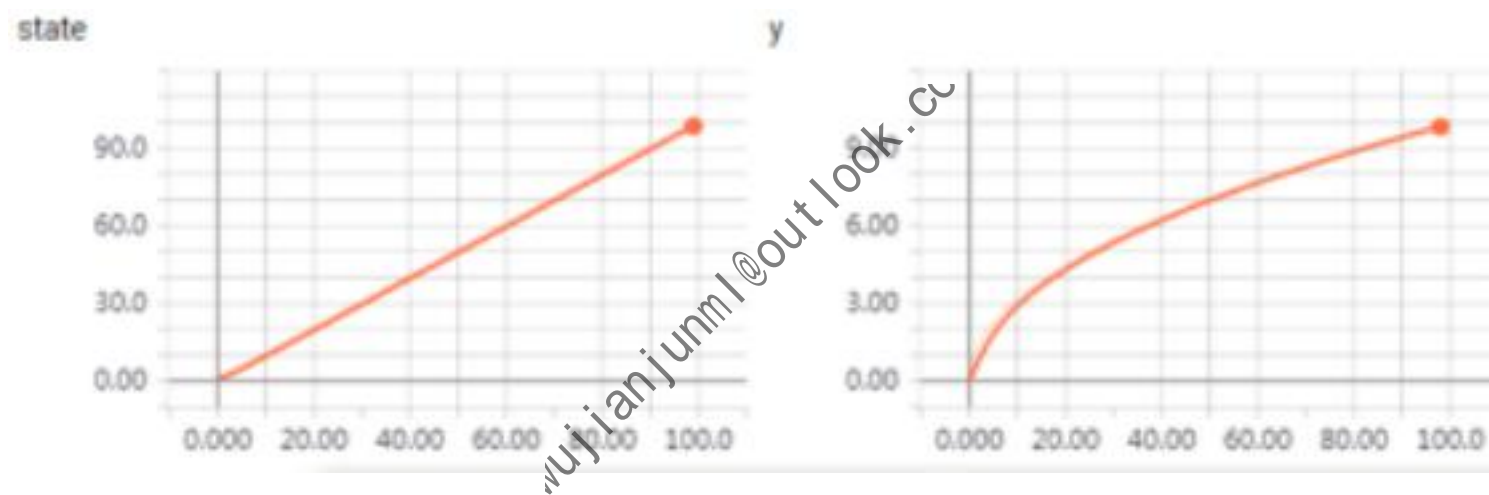
TensorBoard

- 例子如下:

```
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(tf.float32, shape=())
    y = tf.sqrt(x)
    y_summary = tf.summary.scalar("y", y) # 第一个 summary, summary 的输出也是 tensor, 只不过是 string 类型的而已
    state = tf.Variable(0, name="counter")
    one = tf.constant(1)
    new_value = tf.add(state, one)
    update = tf.assign(state, new_value)
    state_summary = tf.summary.scalar("state", state) # 第二个 summary
    init_op = tf.initialize_all_variables()
with tf.Session(graph=g) as sess:
    sess.run(init_op)
    writer = tf.summary.FileWriter("summarytest/", sess.graph)
    for i in range(100):
        sess.run(update)
        result = sess.run(state_summary)
        writer.add_summary(result, global_step=i)
        result = sess.run(y_summary, feed_dict={x: i}) # 任何时候 run 任何 tensor 都必须 feed 值给 placeholder
        writer.add_summary(result, global_step=i)
```

TensorBoard

- Tensorboard显示如下:



计算图的存取

计算图的存取

- 通常，Tensorflow模型分为两部分存储，一部分是网络结构，另外一部分是变量取值。
- 保存时首先在graph中创建一个tf.train.Saver节点，然后在session中调用其save方法。

```
import tensorflow as tf
w1 = tf.Variable(tf.random_normal(shape=[2]), name='w1')
w2 = tf.Variable(tf.random_normal(shape=[5]), name='w2')
saver = tf.train.Saver()
sess = tf.Session()
sess.run(tf.global_variables_initializer())
saver.save(sess, 'my_test_model')
```

- 运行上面的程序将输出下面4个文件：
 - ◆ my_test_model.data-00000-of-00001, 包含网络中变量取值。
 - ◆ my_test_model.index, identifying the checkpoint。
 - ◆ my_test_model.meta, 包含网络结构。
 - ◆ checkpoint, a protocol buffer with a list of recent checkpoints。

计算图的存取

- 保存好模型，使用时需要首先利用meta文件还原网络结构，然后给网络结构中的变量赋予训练得到的值。

```
with tf.Session() as sess:  
    saver = tf.train.import_meta_graph('my-model-1000.meta')  
    saver.restore(sess, tf.train.latest_checkpoint('./'))  
    print(sess.run('w1:0'))
```

- Remember, `import_meta_graph` appends the network defined in .meta file to the current graph. So, this will create the graph/network for you but we still need to load the value of the parameters that we had trained on this graph.

计算图的存取

- 下面举个例子，首先是训练模型的代码。

```
test
s_test
flow_test
sic_test
models
  checkpoint
  tmodel-20.data-0000
  tmodel-20.index
  tmodel-20.meta
  tmodel-40.data-0000
  tmodel-40.index
  tmodel-60.data-0000
  tmodel-60.index
  tmodel-80.data-0000
  tmodel-80.index
  tmodel-100.data-0000
  tmodel-100.index
summary
1helloworld.py
2tensor.py
```

```
4 g = tf.Graph()
5 with g.as_default():
6     state = tf.Variable(0.0, name="counter")
7     one = tf.constant(1.0)
8     update = tf.assign(state, tf.add(state, one))
9     x = tf.placeholder(tf.float32, shape=(1), name="x") # 网络的输入必须以规范的名字命名!!!
10    y = tf.multiply(x, state, name="y") # 网络的输出必须以规范的名字命名!!!
11    init_op = tf.initialize_all_variables()
12    saver = tf.train.Saver()
13    with tf.Session(graph=g) as sess:
14        sess.run(init_op)
15        for i in range(101):
16            sess.run(update) # 更新模型参数，这里直接加一
17            print(sess.run(y, feed_dict={x:i}))
18            # 每隔20轮保存模型
19            if i % 20 == 0 and i > 0:
20                if i == 20: # 第一次保存时，保存meta
21                    saver.save(sess, './models/tmodel', global_step=i)
22                else: # 以后每次保存时，不必重复保存meta
23                    saver.save(sess, './models/tmodel', global_step=i, write_meta_graph=False)
```

计算图的存取

- 然后加载模型做预测。

```
3 class Model:
4     def __init__(self):
5         self.g = tf.Graph()
6         with self.g.as_default():
7             self.saver = tf.train.import_meta_graph('./models/tmodel-20.meta') # 首先还原图结构,
8             self.x = tf.get_default_graph().get_tensor_by_name("x:0")
9             self.y = tf.get_default_graph().get_tensor_by_name("y:0")
10            # 追加图节点
11            c = tf.constant(111, dtype=tf.float32)
12            self.w = tf.add(self.y, c)
13        self.sess = tf.Session(graph=self.g)
14        with self.sess.as_default() as sess:
15            self.saver.restore(sess, tf.train.latest_checkpoint('./models/')) # 然后给图中变量赋值, 采用最新模型的取值
16
17    def predictbyLastestModel(self, v):
18        return self.sess.run(self.y, feed_dict={self.x: v}), self.sess.run(self.w, feed_dict={self.x: v})
19
20    def predictByModel20(self, v):
21        self.saver.restore(self.sess, './models/tmodel-20') # 重新采用指定版本模型的值
22        return self.sess.run(self.y, feed_dict={self.x: v}), self.sess.run(self.w, feed_dict={self.x: v})
23
24 if __name__ == "__main__":
25     model = Model()
26     print(model.predictbyLastestModel(100), model.predictByModel20(100))
27     print(model.predictbyLastestModel(200), model.predictByModel20(200))
28
29 Model > __init__ > with self.sess....
read_test1
(10100.0, 10211.0) (2100.0, 2211.0)
(4200.0, 4311.0) (4200.0, 4311.0)
```

进阶问题一览

进阶问题一览

- 变量共享：几种scope的异同。
- 核心API：网络层与优化器。
- 多GPU训练与分布式训练。
- IO操作。
- 性能调优。
- 网络结构调优
- etc.

wujianjunml@outlook.cn

Q&A

wujianjunm1@outlook.com