1. 新手入门
   1. 介绍

**简介**

本章的目的是让你了解和运行 TensorFlow!

在开始之前, 让我们先看一段使用 Python API 撰写的 TensorFlow 示例代码, 让你对将要学习的内容有初步的印象.

这段很短的 Python 程序生成了一些三维数据, 然后用一个平面拟合它.

import tensorflow as tf

import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.

x\_data = np.float32(np.random.rand(2, 100)) # 随机输入

y\_data = np.dot([0.100, 0.200], x\_data) + 0.300

# 构造一个线性模型

#

b = tf.Variable(tf.zeros([1]))

W = tf.Variable(tf.random\_uniform([1, 2], -1.0, 1.0))

y = tf.matmul(W, x\_data) + b

# 最小化方差

loss = tf.reduce\_mean(tf.square(y - y\_data))

optimizer = tf.train.GradientDescentOptimizer(0.5)

train = optimizer.minimize(loss)

# 初始化变量

init = tf.initialize\_all\_variables()

# 启动图 (graph)

sess = tf.Session()

sess.run(init)

# 拟合平面

for step in xrange(0, 201):

sess.run(train)

if step % 20 == 0:

print step, sess.run(W), sess.run(b)

# 得到最佳拟合结果 W: [[0.100 0.200]], b: [0.300]

为了进一步激发你的学习欲望, 我们想让你先看一下 TensorFlow 是如何解决一个经典的机器学习问题的. 在神经网络领域, 最为经典的问题莫过于 MNIST 手写数字分类问题. 我们准备了 两篇不同的教程, 分别面向机器学习领域的初学者和专家. 如果你已经使用其它软件训练过许多 MNIST 模型, 请阅读高级教程 (红色药丸链接). 如果你以前从未听说过 MNIST, 请阅读初级教程 (蓝色药丸链接). 如果你的水平介于这两类人之间, 我们建议你先快速浏览初级教程, 然后再阅读高级教程.

[](http://www.tensorfly.cn/tfdoc/tutorials/mnist_beginners.html)

图片由 CC BY-SA 4.0 授权; 原作者 W. Carter

如果你已经下定决心, 准备学习和安装 TensorFlow, 你可以略过这些文字, 直接阅读 后面的章节. 不用担心, 你仍然会看到 MNIST -- 在阐述 TensorFlow 的特性时, 我们还会使用 MNIST 作为一个样例.

**推荐随后阅读:**

* [下载与安装](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html)
* [基本使用](http://www.tensorfly.cn/tfdoc/get_started/basic_usage.html)
* [TensorFlow 技术指南](http://www.tensorfly.cn/tfdoc/tutorials/mnist_tf.html)

原文：[Introduction](http://tensorflow.org/get_started) 翻译：[@doc001](https://github.com/PFZheng) 校对：[@yangtze](https://github.com/sstruct)

* 1. 下载及安装

# 下载与安装

你可以使用我们提供的二进制包, 或者使用源代码, 安装 TensorFlow.

## 二进制安装

TensorFlow Python API 依赖 Python 2.7 版本.

在 Linux 和 Mac 下最简单的安装方式, 是使用 [pip](https://pypi.python.org/pypi/pip) 安装.

如果在安装过程中遇到错误, 请查阅 [常见问题](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#common_install_problems). 为了简化安装步骤, 建议使用 virtualenv, 教程见 [这里](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#virtualenv_install).

### Ubuntu/Linux

# 仅使用 CPU 的版本

$ pip install https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.5.0-cp27-none-linux\_x86\_64.whl

# 开启 GPU 支持的版本 (安装该版本的前提是已经安装了 CUDA sdk)

$ pip install https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.5.0-cp27-none-linux\_x86\_64.whl

### Mac OS X

在 OS X 系统上, 我们推荐先安装 [homebrew](http://brew.sh/), 然后执行 brew install python, 以便能够使用 homebrew 中的 Python 安装 TensorFlow. 另外一种推荐的方式是在 [virtualenv](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#virtualenv_install) 中安装 TensorFlow.

# 当前版本只支持 CPU

$ pip install https://storage.googleapis.com/tensorflow/mac/tensorflow-0.5.0-py2-none-any.whl

## 基于 Docker 的安装

我们也支持通过 [Docker](http://docker.com/) 运行 TensorFlow. 该方式的优点是不用操心软件依赖问题.

首先, [安装 Docker](http://docs.docker.com/engine/installation/). 一旦 Docker 已经启动运行, 可以通过命令启动一个容器:

$ docker run -it b.gcr.io/tensorflow/tensorflow

该命令将启动一个已经安装好 TensorFlow 及相关依赖的容器.

### 其它镜像

默认的 Docker 镜像只包含启动和运行 TensorFlow 所需依赖库的一个最小集. 我们额外提供了下面的容器, 该容器同样可以通过上述 docker run 命令安装:

* b.gcr.io/tensorflow/tensorflow-full: 镜像中的 TensorFlow 是从源代码完整安装的, 包含了编译和运行 TensorFlow 所需的全部工具. 在该镜像上, 可以直接使用源代码进行实验, 而不需要再安装上述的任何依赖.

## 基于 VirtualEnv 的安装

我们推荐使用 [virtualenv](https://pypi.python.org/pypi/virtualenv) 创建一个隔离的容器, 来安装 TensorFlow. 这是可选的, 但是这样做能使排查安装问题变得更容易.

首先, 安装所有必备工具:

# 在 Linux 上:

$ sudo apt-get install python-pip python-dev python-virtualenv

# 在 Mac 上:

$ sudo easy\_install pip # 如果还没有安装 pip

$ sudo pip install --upgrade virtualenv

接下来, 建立一个全新的 virtualenv 环境. 为了将环境建在 ~/tensorflow 目录下, 执行:

$ virtualenv --system-site-packages ~/tensorflow

$ cd ~/tensorflow

然后, 激活 virtualenv:

$ source bin/activate # 如果使用 bash

$ source bin/activate.csh # 如果使用 csh

(tensorflow)$ # 终端提示符应该发生变化

在 virtualenv 内, 安装 TensorFlow:

(tensorflow)$ pip install --upgrade <$url\_to\_binary.whl>

接下来, 使用类似命令运行 TensorFlow 程序:

(tensorflow)$ cd tensorflow/models/image/mnist

(tensorflow)$ python convolutional.py

# 当使用完 TensorFlow

(tensorflow)$ deactivate # 停用 virtualenv

$ # 你的命令提示符会恢复原样

## 尝试你的第一个 TensorFlow 程序

### (可选) 启用 GPU 支持

如果你使用 pip 二进制包安装了开启 GPU 支持的 TensorFlow, 你必须确保系统里安装了正确的 CUDA sdk 和 CUDNN 版本. 请参间 [CUDA 安装教程](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#install_cuda)

你还需要设置 LD\_LIBRARY\_PATH 和 CUDA\_HOME 环境变量. 可以考虑将下面的命令 添加到 ~/.bash\_profile 文件中, 这样每次登陆后自动生效. 注意, 下面的命令 假定 CUDA 安装目录为 /usr/local/cuda:

export LD\_LIBRARY\_PATH="$LD\_LIBRARY\_PATH:/usr/local/cuda/lib64"

export CUDA\_HOME=/usr/local/cuda

### 运行 TensorFlow

打开一个 python 终端:

$ python

>>> import tensorflow as tf

>>> hello = tf.constant('Hello, TensorFlow!')

>>> sess = tf.Session()

>>> print sess.run(hello)

Hello, TensorFlow!

>>> a = tf.constant(10)

>>> b = tf.constant(32)

>>> print sess.run(a+b)

42

>>>

## 从源码安装

### 克隆 TensorFlow 仓库

$ git clone --recurse-submodules https://github.com/tensorflow/tensorflow

--recurse-submodules 参数是必须得, 用于获取 TesorFlow 依赖的 protobuf 库.

### Linux 安装

#### 安装 Bazel

首先依照 [教程](http://bazel.io/docs/install.html) 安装 Bazel 的依赖. 然后使用下列命令下载和编译 Bazel 的源码:

$ git clone https://github.com/bazelbuild/bazel.git

$ cd bazel

$ git checkout tags/0.1.0

$ ./compile.sh

上面命令中拉取的代码标签为 0.1.0, 兼容 Tensorflow 目前版本. bazel 的HEAD 版本 (即最新版本) 在这里可能不稳定.

将执行路径 output/bazel 添加到 $PATH 环境变量中.

#### 安装其他依赖

$ sudo apt-get install python-numpy swig python-dev

#### 可选: 安装 CUDA (在 Linux 上开启 GPU 支持)

为了编译并运行能够使用 GPU 的 TensorFlow, 需要先安装 NVIDIA 提供的 Cuda Toolkit 7.0 和 CUDNN 6.5 V2.

TensorFlow 的 GPU 特性只支持 NVidia Compute Capability >= 3.5 的显卡. 被支持的显卡 包括但不限于:

* NVidia Titan
* NVidia Titan X
* NVidia K20
* NVidia K40

##### 下载并安装 Cuda Toolkit 7.0

[下载地址](https://developer.nvidia.com/cuda-toolkit-70)

将工具安装到诸如 /usr/local/cuda 之类的路径.

##### 下载并安装 CUDNN Toolkit 6.5

[下载地址](https://developer.nvidia.com/rdp/cudnn-archive)

解压并拷贝 CUDNN 文件到 Cuda Toolkit 7.0 安装路径下. 假设 Cuda Toolkit 7.0 安装 在 /usr/local/cuda, 执行以下命令:

tar xvzf cudnn-6.5-linux-x64-v2.tgz

sudo cp cudnn-6.5-linux-x64-v2/cudnn.h /usr/local/cuda/include

sudo cp cudnn-6.5-linux-x64-v2/libcudnn\* /usr/local/cuda/lib64

##### 配置 TensorFlow 的 Cuba 选项

从源码树的根路径执行:

$ ./configure

Do you wish to bulid TensorFlow with GPU support? [y/n] y

GPU support will be enabled for TensorFlow

Please specify the location where CUDA 7.0 toolkit is installed. Refer to

README.md for more details. [default is: /usr/local/cuda]: /usr/local/cuda

Please specify the location where CUDNN 6.5 V2 library is installed. Refer to

README.md for more details. [default is: /usr/local/cuda]: /usr/local/cuda

Setting up Cuda include

Setting up Cuda lib64

Setting up Cuda bin

Setting up Cuda nvvm

Configuration finished

这些配置将建立到系统 Cuda 库的符号链接. 每当 Cuda 库的路径发生变更时, 必须重新执行上述 步骤, 否则无法调用 bazel 编译命令.

##### 编译目标程序, 开启 GPU 支持

从源码树的根路径执行:

$ bazel build -c opt --config=cuda //tensorflow/cc:tutorials\_example\_trainer

$ bazel-bin/tensorflow/cc/tutorials\_example\_trainer --use\_gpu

# 大量的输出信息. 这个例子用 GPU 迭代计算一个 2x2 矩阵的主特征值 (major eigenvalue).

# 最后几行输出和下面的信息类似.

000009/000005 lambda = 2.000000 x = [0.894427 -0.447214] y = [1.788854 -0.894427]

000006/000001 lambda = 2.000000 x = [0.894427 -0.447214] y = [1.788854 -0.894427]

000009/000009 lambda = 2.000000 x = [0.894427 -0.447214] y = [1.788854 -0.894427]

注意, GPU 支持需通过编译选项 "--config=cuda" 开启.

##### 已知问题

* 尽管可以在同一个源码树下编译开启 Cuda 支持和禁用 Cuda 支持的版本, 我们还是推荐在 在切换这两种不同的编译配置时, 使用 "bazel clean" 清理环境.
* 在执行 bazel 编译前必须先运行 configure, 否则编译会失败并提示错误信息. 未来, 我们可能考虑将 configure 步骤包含在编译过程中, 以简化整个过程, 前提是 bazel 能够提供新的特性支持这样.

### Mac OS X 安装

Mac 和 Linux 需要的软件依赖完全一样, 但是安装过程区别很大. 以下链接用于帮助你 在 Mac OS X 上安装这些依赖:

#### Bazel

参见[本网页](http://bazel.io/docs/install.html)的 Mac OS X 安装指南.

#### SWIG

[Mac OS X 安装教程](http://www.swig.org/Doc3.0/Preface.html#Preface_osx_installation).

注意: 你需要安装[PCRE](ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/), 而不是 PCRE2.

#### Numpy

参见[安装教程](http://docs.scipy.org/doc/numpy/user/install.html).

### 创建 pip 包并安装

$ bazel build -c opt //tensorflow/tools/pip\_package:build\_pip\_package

$ bazel-bin/tensorflow/tools/pip\_package/build\_pip\_package /tmp/tensorflow\_pkg

# .whl 文件的实际名字与你所使用的平台有关

$ pip install /tmp/tensorflow\_pkg/tensorflow-0.5.0-cp27-none-linux\_x86\_64.whl

## 训练你的第一个 TensorFlow 神经网络模型

从源代码树的根路径执行:

$ cd tensorflow/models/image/mnist

$ python convolutional.py

Succesfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.

Succesfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.

Succesfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.

Succesfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.

Extracting data/train-images-idx3-ubyte.gz

Extracting data/train-labels-idx1-ubyte.gz

Extracting data/t10k-images-idx3-ubyte.gz

Extracting data/t10k-labels-idx1-ubyte.gz

Initialized!

Epoch 0.00

Minibatch loss: 12.054, learning rate: 0.010000

Minibatch error: 90.6%

Validation error: 84.6%

Epoch 0.12

Minibatch loss: 3.285, learning rate: 0.010000

Minibatch error: 6.2%

Validation error: 7.0%

...

...

## 常见问题

### GPU 相关问题

如果在尝试运行一个 TensorFlow 程序时出现以下错误:

ImportError: libcudart.so.7.0: cannot open shared object file: No such file or directory

请确认你正确安装了 GPU 支持, 参见 [相关章节](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#install_cuda).

### 在 Linux 上

如果出现错误:

...

"\_\_add\_\_", "\_\_radd\_\_",

^

SyntaxError: invalid syntax

解决方案: 确认正在使用的 Python 版本为 Python 2.7.

### 在 Mac OS X 上

如果出现错误:

import six.moves.copyreg as copyreg

ImportError: No module named copyreg

解决方案: TensorFlow 使用的 protobuf 依赖 six-1.10.0. 但是, Apple 的默认 python 环境 已经安装了 six-1.4.1, 该版本可能很难升级. 这里提供几种方法来解决该问题:

1. 升级全系统的 six:
2. sudo easy\_install -U six
3. 通过 homebrew 安装一个隔离的 python 副本:
4. brew install python
5. 在[virtualenv](http://www.tensorfly.cn/tfdoc/get_started/os_setup.html#virtualenv_install) 内编译或使用 TensorFlow.

如果出现错误:

>>> import tensorflow as tf

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/usr/local/lib/python2.7/site-packages/tensorflow/\_\_init\_\_.py", line 4, in <module>

from tensorflow.python import \*

File "/usr/local/lib/python2.7/site-packages/tensorflow/python/\_\_init\_\_.py", line 13, in <module>

from tensorflow.core.framework.graph\_pb2 import \*

...

File "/usr/local/lib/python2.7/site-packages/tensorflow/core/framework/tensor\_shape\_pb2.py", line 22, in <module>

serialized\_pb=\_b('\n,tensorflow/core/framework/tensor\_shape.proto\x12\ntensorflow\"d\n\x10TensorShapeProto\x12-\n\x03\x64im\x18\x02 \x03(\x0b\x32 .tensorflow.TensorShapeProto.Dim\x1a!\n\x03\x44im\x12\x0c\n\x04size\x18\x01 \x01(\x03\x12\x0c\n\x04name\x18\x02 \x01(\tb\x06proto3')

TypeError: \_\_init\_\_() got an unexpected keyword argument 'syntax'

这是由于安装了冲突的 protobuf 版本引起的, TensorFlow 需要的是 protobuf 3.0.0. 当前 最好的解决方案是确保没有安装旧版本的 protobuf, 可以使用以下命令重新安装 protobuf 来解决 冲突:

brew reinstall --devel protobuf

原文：[Download and Setup](http://tensorflow.org/get_started/os_setup.md) 翻译：[@doc001](https://github.com/PFZheng) 校对：[@yangtze](https://github.com/sstruct)

* 1. 基本用法

# 基本使用

使用 TensorFlow, 你必须明白 TensorFlow:

* 使用图 (graph) 来表示计算任务.
* 在被称之为 会话 (Session) 的上下文 (context) 中执行图.
* 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(arbitrary operation) 赋值或者从其中获取数据.

## 综述

TensorFlow 是一个编程系统, 使用图来表示计算任务. 图中的节点被称之为 op (operation 的缩写). 一个 op 获得 0 个或多个 Tensor, 执行计算, 产生 0 个或多个 Tensor. 每个 Tensor 是一个类型化的多维数组. 例如, 你可以将一小组图像集表示为一个四维浮点数数组, 这四个维度分别是 [batch, height, width, channels].

一个 TensorFlow 图描述了计算的过程. 为了进行计算, 图必须在 会话 里被启动. 会话 将图的 op 分发到诸如 CPU 或 GPU 之类的 设备 上, 同时提供执行 op 的方法. 这些方法执行后, 将产生的 tensor 返回. 在 Python 语言中, 返回的 tensor 是 [numpy](http://www.numpy.org/) ndarray 对象; 在 C 和 C++ 语言中, 返回的 tensor 是tensorflow::Tensor 实例.

## 计算图

TensorFlow 程序通常被组织成一个构建阶段和一个执行阶段. 在构建阶段, op 的执行步骤 被描述成一个图. 在执行阶段, 使用会话执行执行图中的 op.

例如, 通常在构建阶段创建一个图来表示和训练神经网络, 然后在执行阶段反复执行图中的训练 op.

TensorFlow 支持 C, C++, Python 编程语言. 目前, TensorFlow 的 Python 库更加易用, 它提供了大量的辅助函数来简化构建图的工作, 这些函数尚未被 C 和 C++ 库支持.

三种语言的会话库 (session libraries) 是一致的.

### 构建图

构建图的第一步, 是创建源 op (source op). 源 op 不需要任何输入, 例如 常量 (Constant). 源 op 的输出被传递给其它 op 做运算.

Python 库中, op 构造器的返回值代表被构造出的 op 的输出, 这些返回值可以传递给其它 op 构造器作为输入.

TensorFlow Python 库有一个默认图 (default graph), op 构造器可以为其增加节点. 这个默认图对 许多程序来说已经足够用了. 阅读 [Graph 类](http://www.tensorfly.cn/tfdoc/api_docs/python/framework.html#Graph) 文档 来了解如何管理多个图.

import tensorflow as tf

# 创建一个常量 op, 产生一个 1x2 矩阵. 这个 op 被作为一个节点

# 加到默认图中.

#

# 构造器的返回值代表该常量 op 的返回值.

matrix1 = tf.constant([[3., 3.]])

# 创建另外一个常量 op, 产生一个 2x1 矩阵.

matrix2 = tf.constant([[2.],[2.]])

# 创建一个矩阵乘法 matmul op , 把 'matrix1' 和 'matrix2' 作为输入.

# 返回值 'product' 代表矩阵乘法的结果.

product = tf.matmul(matrix1, matrix2)

默认图现在有三个节点, 两个 constant() op, 和一个matmul() op. 为了真正进行矩阵相乘运算, 并得到矩阵乘法的 结果, 你必须在会话里启动这个图.

### 在一个会话中启动图

构造阶段完成后, 才能启动图. 启动图的第一步是创建一个 Session 对象, 如果无任何创建参数, 会话构造器将启动默认图.

欲了解完整的会话 API, 请阅读[Session 类](http://www.tensorfly.cn/tfdoc/api_docs/python/client.html#session-management).

# 启动默认图.

sess = tf.Session()

# 调用 sess 的 'run()' 方法来执行矩阵乘法 op, 传入 'product' 作为该方法的参数.

# 上面提到, 'product' 代表了矩阵乘法 op 的输出, 传入它是向方法表明, 我们希望取回

# 矩阵乘法 op 的输出.

#

# 整个执行过程是自动化的, 会话负责传递 op 所需的全部输入. op 通常是并发执行的.

#

# 函数调用 'run(product)' 触发了图中三个 op (两个常量 op 和一个矩阵乘法 op) 的执行.

#

# 返回值 'result' 是一个 numpy `ndarray` 对象.

result = sess.run(product)

print result

# ==> [[ 12.]]

# 任务完成, 关闭会话.

sess.close()

Session 对象在使用完后需要关闭以释放资源. 除了显式调用 close 外, 也可以使用 "with" 代码块 来自动完成关闭动作.

with tf.Session() as sess:

result = sess.run([product])

print result

在实现上, TensorFlow 将图形定义转换成分布式执行的操作, 以充分利用可用的计算资源(如 CPU 或 GPU). 一般你不需要显式指定使用 CPU 还是 GPU, TensorFlow 能自动检测. 如果检测到 GPU, TensorFlow 会尽可能地利用找到的第一个 GPU 来执行操作.

如果机器上有超过一个可用的 GPU, 除第一个外的其它 GPU 默认是不参与计算的. 为了让 TensorFlow 使用这些 GPU, 你必须将 op 明确指派给它们执行. with...Device 语句用来指派特定的 CPU 或 GPU 执行操作:

with tf.Session() as sess:

with tf.device("/gpu:1"):

matrix1 = tf.constant([[3., 3.]])

matrix2 = tf.constant([[2.],[2.]])

product = tf.matmul(matrix1, matrix2)

...

设备用字符串进行标识. 目前支持的设备包括:

* "/cpu:0": 机器的 CPU.
* "/gpu:0": 机器的第一个 GPU, 如果有的话.
* "/gpu:1": 机器的第二个 GPU, 以此类推.

阅读[使用GPU](http://www.tensorfly.cn/tfdoc/how_tos/using_gpu.html)章节, 了解 TensorFlow GPU 使用的更多信息.

## 交互式使用

文档中的 Python 示例使用一个会话 [Session](http://www.tensorfly.cn/tfdoc/api_docs/python/client.html#Session) 来 启动图, 并调用 [Session.run()](http://www.tensorfly.cn/tfdoc/api_docs/python/client.html#Session.run) 方法执行操作.

为了便于使用诸如 [IPython](http://ipython.org/) 之类的 Python 交互环境, 可以使用 [InteractiveSession](http://www.tensorfly.cn/tfdoc/api_docs/python/client.html#InteractiveSession) 代替 Session 类, 使用 [Tensor.eval()](http://www.tensorfly.cn/tfdoc/api_docs/python/framework.html#Tensor.eval) 和 [Operation.run()](http://www.tensorfly.cn/tfdoc/api_docs/python/framework.html#Operation.run) 方法代替 Session.run(). 这样可以避免使用一个变量来持有会话.

# 进入一个交互式 TensorFlow 会话.

import tensorflow as tf

sess = tf.InteractiveSession()

x = tf.Variable([1.0, 2.0])

a = tf.constant([3.0, 3.0])

# 使用初始化器 initializer op 的 run() 方法初始化 'x'

x.initializer.run()

# 增加一个减法 sub op, 从 'x' 减去 'a'. 运行减法 op, 输出结果

sub = tf.sub(x, a)

print sub.eval()

# ==> [-2. -1.]

## Tensor

TensorFlow 程序使用 tensor 数据结构来代表所有的数据, 计算图中, 操作间传递的数据都是 tensor. 你可以把 TensorFlow tensor 看作是一个 n 维的数组或列表. 一个 tensor 包含一个静态类型 rank, 和 一个 shape. 想了解 TensorFlow 是如何处理这些概念的, 参见 [Rank, Shape, 和 Type](http://www.tensorfly.cn/tfdoc/resources/dims_types.html).

## 变量

[Variables](http://www.tensorfly.cn/tfdoc/how_tos/variables.html) for more details. 变量维护图执行过程中的状态信息. 下面的例子演示了如何使用变量实现一个简单的计数器. 参见 [变量](http://www.tensorfly.cn/tfdoc/how_tos/variables.html) 章节了解更多细节.

# 创建一个变量, 初始化为标量 0.

state = tf.Variable(0, name="counter")

# 创建一个 op, 其作用是使 state 增加 1

one = tf.constant(1)

new\_value = tf.add(state, one)

update = tf.assign(state, new\_value)

# 启动图后, 变量必须先经过`初始化` (init) op 初始化,

# 首先必须增加一个`初始化` op 到图中.

init\_op = tf.initialize\_all\_variables()

# 启动图, 运行 op

with tf.Session() as sess:

# 运行 'init' op

sess.run(init\_op)

# 打印 'state' 的初始值

print sess.run(state)

# 运行 op, 更新 'state', 并打印 'state'

for \_ in range(3):

sess.run(update)

print sess.run(state)

# 输出:

# 0

# 1

# 2

# 3

代码中 assign() 操作是图所描绘的表达式的一部分, 正如 add() 操作一样. 所以在调用 run() 执行表达式之前, 它并不会真正执行赋值操作.

通常会将一个统计模型中的参数表示为一组变量. 例如, 你可以将一个神经网络的权重作为某个变量存储在一个 tensor 中. 在训练过程中, 通过重复运行训练图, 更新这个 tensor.

## Fetch

为了取回操作的输出内容, 可以在使用 Session 对象的 run() 调用 执行图时, 传入一些 tensor, 这些 tensor 会帮助你取回结果. 在之前的例子里, 我们只取回了单个节点 state, 但是你也可以取回多个 tensor:

input1 = tf.constant(3.0)

input2 = tf.constant(2.0)

input3 = tf.constant(5.0)

intermed = tf.add(input2, input3)

mul = tf.mul(input1, intermed)

with tf.Session():

result = sess.run([mul, intermed])

print result

# 输出:

# [array([ 21.], dtype=float32), array([ 7.], dtype=float32)]

需要获取的多个 tensor 值，在 op 的一次运行中一起获得（而不是逐个去获取 tensor）。

## Feed

上述示例在计算图中引入了 tensor, 以常量或变量的形式存储. TensorFlow 还提供了 feed 机制, 该机制 可以临时替代图中的任意操作中的 tensor 可以对图中任何操作提交补丁, 直接插入一个 tensor.

feed 使用一个 tensor 值临时替换一个操作的输出结果. 你可以提供 feed 数据作为 run() 调用的参数. feed 只在调用它的方法内有效, 方法结束, feed 就会消失. 最常见的用例是将某些特殊的操作指定为 "feed" 操作, 标记的方法是使用 tf.placeholder() 为这些操作创建占位符.

input1 = tf.placeholder(tf.types.float32)

input2 = tf.placeholder(tf.types.float32)

output = tf.mul(input1, input2)

with tf.Session() as sess:

print sess.run([output], feed\_dict={input1:[7.], input2:[2.]})

# 输出:

# [array([ 14.], dtype=float32)]

for a larger-scale example of feeds. 如果没有正确提供 feed, placeholder() 操作将会产生错误. [MNIST 全连通 feed 教程](http://www.tensorfly.cn/tfdoc/tutorials/mnist_tf.html) ([source code](https://tensorflow.googlesource.com/tensorflow/+/master/tensorflow/g3doc/tutorials/mnist/fully_connected_feed.py)) 给出了一个更大规模的使用 feed 的例子.

原文：[Basic Usage](http://tensorflow.org/get_started/basic_usage.md) 翻译：[@doc001](https://github.com/PFZheng) 校对：[@yangtze](https://github.com/sstruct)