目錄

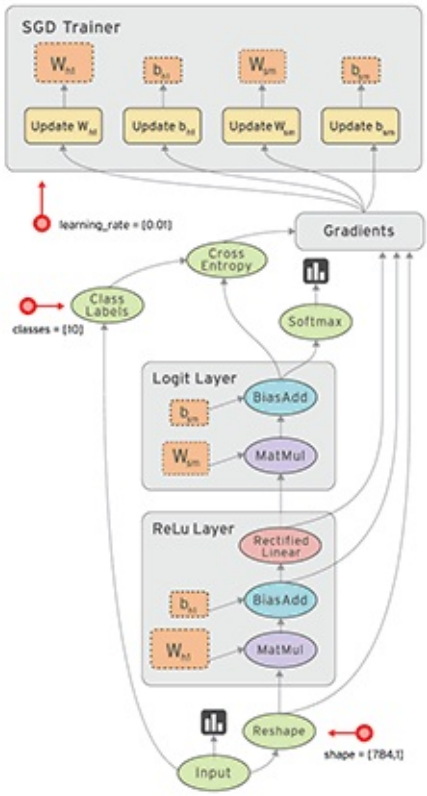
Tensorflow实践

Tensorflow是谷歌在2015年11月开源的机器学习框架，来源于Google内部的深度学习框架DistBelief。由于其良好的架构、分布式架构支持以及简单易用，自开源以来得到广泛的关注。

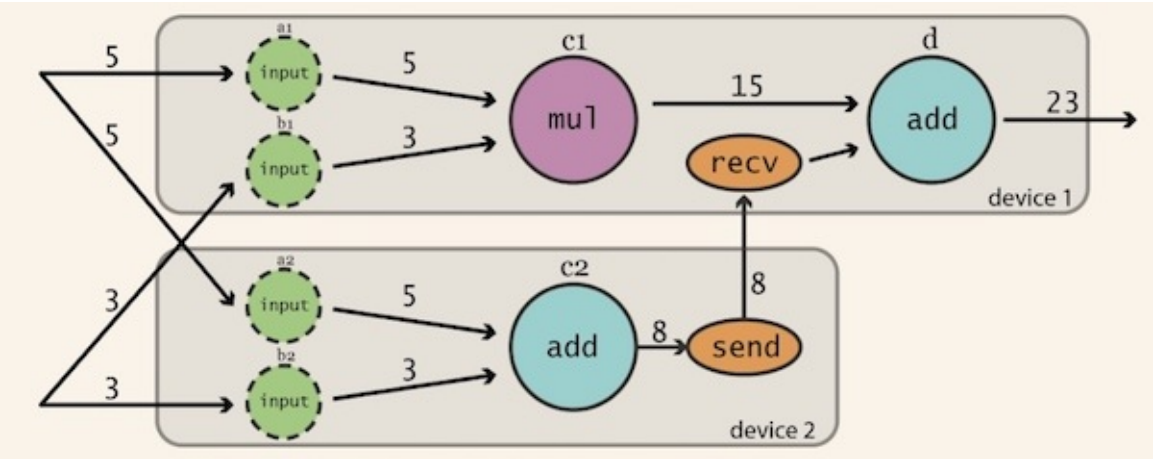
本书整理了Tensorflow的参考指南和实践总结，欢迎关注和参与维护项目。

Tensorflow简介

Tensorflow是一个使用数据流图 (data flow graphs) 技术来进行数值计算的开源软件库。



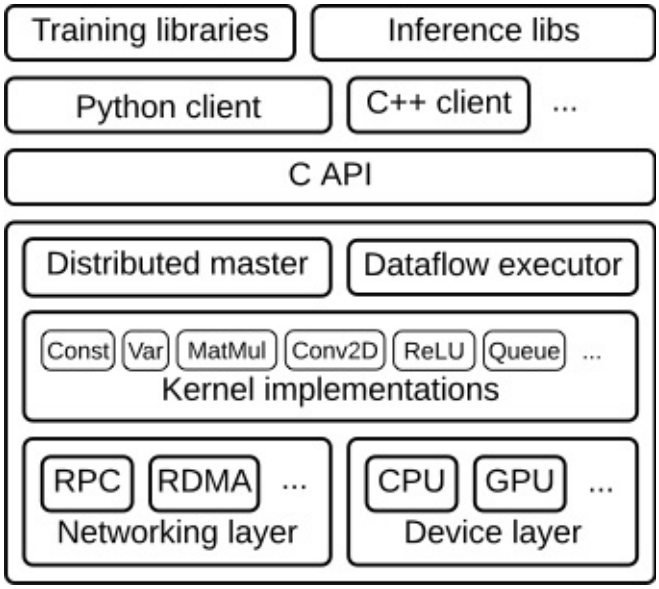
数据流图是是一个有向图，使用结点（一般用圆形或者方形描述，表示一个数学操作或者数据输入的起点和数据输出的终点）和线（表示数字、矩阵或者Tensor张量）来描述数学计算。数据流图可以方便的将各个节点分配到不同的计算设备上完成异步并行计算，适合大规模的机器学习应用。



Tensorflow支持多种异构平台，包括CPU、GPU、移动设备等。TensorFlow内核采用C/C++开发，并提供了C++，Python，Java，Go语言的Client API。其架构灵活，能够支持各种网络模型，具有良好的通用性和可扩展性。

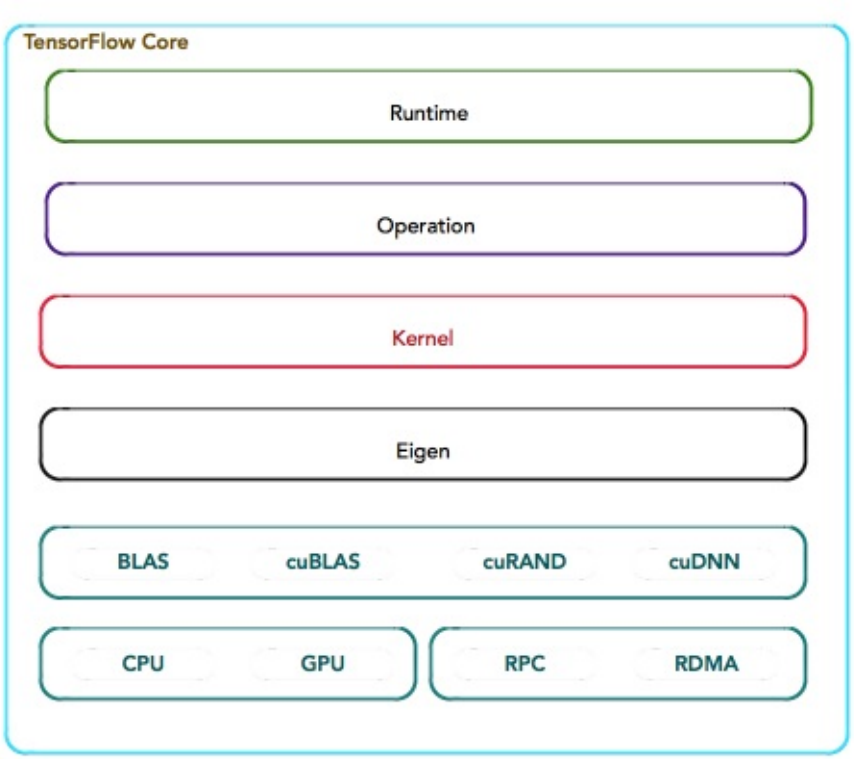
系统概述

Tensorflow以 `C API` 为边界可以分为前端系统（提供编程模型，负责构造计算图）和后端系统（提供运行环境，负责执行计算图）：



如上图所示，系统中主要组件包括

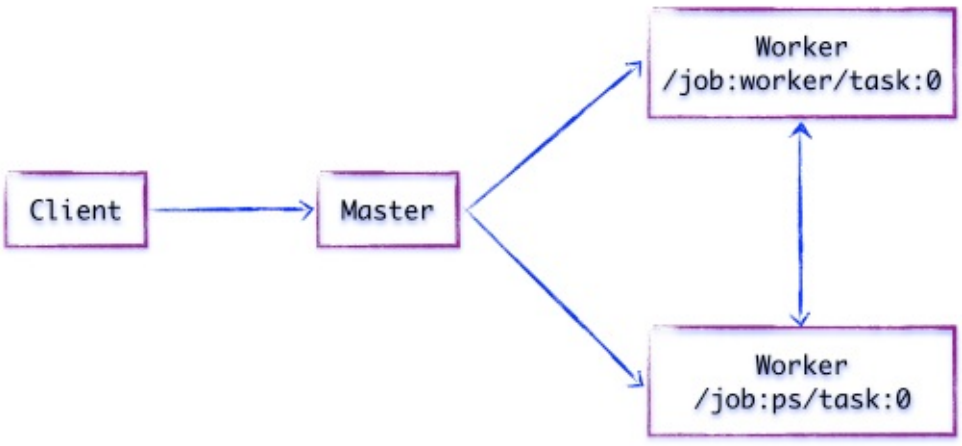
- 支持多语言的客户端，方便用户构造各种复杂的计算图，实现所需的模型设计。客户端以Session为桥梁连接后段的运行时，并启动计算图的执行过程
- 分布式Master负责将计算图拆分为多个子图，以便在不通的进程和设备上并行执行
- Worker Service负责在硬件环境（如CPU或GPU）上调用OP的Kernel实现完成图的计算，并从其他Worker Service接受计算结果或将计算结果发送给其他Worker Services
- Kernel Implements是OP在硬件设备上的特定实现，负责执行OP运算，如数值计算、多维数组操作、控制流、状态管理等（如下图所示）



组件交互

假设存在两个任务

- `/job:ps/task:0`：负责模型参数的存储和更新
- `/job:worker/task:0`：负责模型的训练或推理



组件交互过程为

- 客户端通过TensorFlow的编程接口构造计算图
- 客户端建立Session会话，将Protobuf格式的图定义发送给Distributed Master
- Distributed Master根据 `Session.run` 的 `Fetching` 参数，从计算图中反向遍历，找到所依赖的最小子图；然后将该子图再次分裂为多个字图片段，以便在不同进程

和设备上运行

- Distributed Master将这些子图片段分发给Work Service，并负责任务集的协同
- 随后Work Service启动「本地子图」的执行过程，包括
  - 处理来自Master的请求
  - 调度OP的Kernel实现，执行子图运算
  - 协同任务之间的数据通信

参考文档

- TensorFlow Architecture
- Tensorflow架构与设计

Tensorflow安装

介绍Tensorflow的安装方法，包括

- CPU版安装方法
- GPU版安装方法
- 源码安装方法
- Tensorflow For Go

Tensorflow CPU版本安装

pip安装

最简单的方法使用pip来安装

```
# Python 2.7
pip install --upgrade tensorflow
# Python 3.x
pip3 install --upgrade tensorflow
```

docker

使用镜像 `gcr.io/tensorflow/tensorflow` 启动CPU版Tensorflow：

```
docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow
```

验证安装

```
$ python
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
Hello, TensorFlow!
>>>
```

Tensorflow GPU版本安装

> 注意：从1.2版本开始，Mac OSX不再支持GPU版本（CPU版仍继续支持）。

pip安装

最简单的方法是使用pip安装：

```
# Python 2.7
pip install --upgrade tensorflow-gpu
# Python 3.x
pip3 install --upgrade tensorflow-gpu
```

Docker

首先安装nvidia-docker：

```
# Install nvidia-docker and nvidia-docker-plugin
wget -P /tmp https://github.com/NVIDIA/nvidia-docker/releases/downloa
d/v1.0.1/nvidia-docker_1.0.1-1_amd64.deb
sudo dpkg -i /tmp/nvidia-docker*.deb && rm /tmp/nvidia-docker*.deb

# Test nvidia-smi
nvidia-docker run --rm nvidia/cuda nvidia-smi
```

然后可以使用 `gcr.io/tensorflow/tensorflow:latest-gpu` 镜像启动GPU版Tensorflow：

```
nvidia-docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow:lates
t-gpu
```

CUDA和cuDNN

安装CUDA：

```
# Check for CUDA and try to install.
if ! dpkg-query -W cuda; then
  # The 16.04 installer works with 16.10.
  curl -O http://developer.download.nvidia.com/compute/cuda/repos/ubu
ntu1604/x86_64/cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
  dpkg -i ./cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
  apt-get update
  apt-get install libcupti-dev cuda -y
fi
```

安装cuDNN：

首先到网站https://developer.nvidia.com/cudnn注册，并下载cuDNN v5.1，然后运行命令安装

```
wget https://www.dropbox.com/s/xdak8t60lzk11zb/cudnn-8.0-linux-x64-v5
.1.tgz?dl=1 -O cudnn-8.0-linux-x64-v5.1.tgz
tar zxvf cudnn-8.0-linux-x64-v5.1.tgz
ln -s /usr/local/cuda-8.0 /usr/local/cuda
sudo cp -P cuda/include/cudnn.h /usr/local/cuda/include
sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda/lib64
sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/
libcudnn*
```

安装完成后，可以运行nvidia-smi查看GPU设备的状态

```
$ nvidia-smi
Fri Jun 16 19:33:35 2017
+-----------------------------------------------------------------------------
---------+
| NVIDIA-SMI 375.66                 Driver Version: 375.66
         |
|-------------------------------+----------------------+----------------------
---------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Unc
orr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Co
mpute M. |
|===============================+======================+======================
=========|
|   0  Tesla K80           Off  | 0000:00:04.0     Off |
       0 |
| N/A   74C    P0    80W / 149W |      0MiB / 11439MiB |    100%
 Default |
+-------------------------------+----------------------+----------------------
---------+

+-----------------------------------------------------------------------------
---------+
| Processes:                                                                GP
U Memory |
|  GPU       PID  Type  Process name                                        Us
age      |
|=============================================================================
=========|
|  No running processes found
         |
+-----------------------------------------------------------------------------
---------+
```

验证安装

```
$ python
>>> from tensorflow.python.client import device_lib
>>> print device_lib.list_local_devices()
...
[name: "/cpu:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 9675741273569321173
, name: "/gpu:0"
device_type: "GPU"
memory_limit: 11332668621
locality {
  bus_id: 1
}
incarnation: 7807115828340118187
physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:0
0:04.0"
]
>>>
```

Tensorflow源码安装

以Ubuntu 16.04为例，介绍Tensorflow源码安装的方法。

下载tensorflow源码

```
git clone https://github.com/tensorflow/tensorflow
```

安装bazel

```
echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable
 jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list
curl https://bazel.build/bazel-release.pub.gpg | sudo apt-key add -
sudo apt-get update && sudo apt-get install bazel
```

安装依赖库

```
# Python 2.7
sudo apt-get install python-numpy python-dev python-pip python-wheel
# Python 3.x
sudo apt-get install python3-numpy python3-dev python3-pip python3-wh
eel
```

安装CUDA和cuDNN

参考这里。

编译安装

```
cd tensorflow
./configure
```

安装命令行提示，逐个设置编译选项（可以选择默认值）。

编译CPU版：

```
bazel build --config=opt //tensorflow/tools/pip_package:build_pip_pac
kage
```

编译GPU版：

```
bazel build --config=opt --config=cuda //tensorflow/tools/pip_package
:build_pip_package
```

> 注意，GCC 5需要设置 `--cxxopt="-D_GLIBCXX_USE_CXX11_ABI=0"` 选项。

`bazel build` 会生成一个 `build_pip_package` 命令，用来生成python whl包：

```
# 编译生成python whl包
bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensorf
low_pkg
```

最后，安装生成的包

```
sudo pip install /tmp/tensorflow_pkg/tensorflow-1.2.0-py2-none-any.wh
l
```

Tensorflow For Go

Tensorflow For Go支持Linux和OSX。

安装

下载动态链接库

```
$ TF_TYPE="cpu" # Change to "gpu" for GPU support
$ TARGET_DIRECTORY='/usr/local'
$ curl -L "https://storage.googleapis.com/tensorflow/libtensorflow/li
btensorflow-${TF_TYPE}-$(go env GOOS)-x86_64-1.2.0.tar.gz" | sudo tar
 -C $TARGET_DIRECTORY -xz
# Linux上还需要执行ldconfig
$ sudo ldconfig
```

下载Tensorflow Go库

```
$ go get github.com/tensorflow/tensorflow/tensorflow/go
```

下载完成后，可以用 `go test` 验证安装

```
$ go test github.com/tensorflow/tensorflow/tensorflow/go
```

Go示例

```
$ cat <<EOF | tee tf-hello.go
package main

import (
    tf "github.com/tensorflow/tensorflow/tensorflow/go"
    "github.com/tensorflow/tensorflow/tensorflow/go/op"
    "fmt"
)
```

```
func main() {
    // Construct a graph with an operation that produces a string con
stant.
    s := op.NewScope()
    c := op.Const(s, "Hello from TensorFlow version " + tf.Version())
    graph, err := s.Finalize()
    if err != nil {
        panic(err)
    }

    // Execute the graph in a session.
    sess, err := tf.NewSession(graph, nil)
    if err != nil {
        panic(err)
    }
    output, err := sess.Run(nil, []tf.Output{c}, nil)
    if err != nil {
        panic(err)
    }
    fmt.Println(output[0].Value())
}
EOF

$ export LIBRARY_PATH=/usr/local/lib
$ go run tf-hello.go
2017-06-19 15:38:38.569010: W tensorflow/core/platform/cpu_feature_gu
ard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instr
uctions, but these are available on your machine and could speed up C
PU computations.
2017-06-19 15:38:38.569633: W tensorflow/core/platform/cpu_feature_gu
ard.cc:45] The TensorFlow library wasn't compiled to use AVX instruct
ions, but these are available on your machine and could speed up CPU
computations.
2017-06-19 15:38:38.569640: W tensorflow/core/platform/cpu_feature_gu
ard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instruc
tions, but these are available on your machine and could speed up CPU
 computations.
2017-06-19 15:38:38.569646: W tensorflow/core/platform/cpu_feature_gu
ard.cc:45] The TensorFlow library wasn't compiled to use FMA instruct
ions, but these are available on your machine and could speed up CPU
computations.
```

```
Hello from TensorFlow version 1.2.0
```

C示例

C和Go实际上共用了相同的动态链接库和头文件，安装好Go版本后，C的API可以直接使用，如

```
#include <stdio.h>
#include <tensorflow/c/c_api.h>

int main() {
    printf("Hello from TensorFlow C library version %s\n", TF_Version
());
    return 0;
}
```

编译，运行

```
$ gcc -I/usr/local/include -L/usr/local/lib tf.c -ltensorflow
$ ./a.out
Hello from TensorFlow C library version 1.2.0
```

Tensorflow入门

Tensorflow入门简介。

Hello World

查看版本

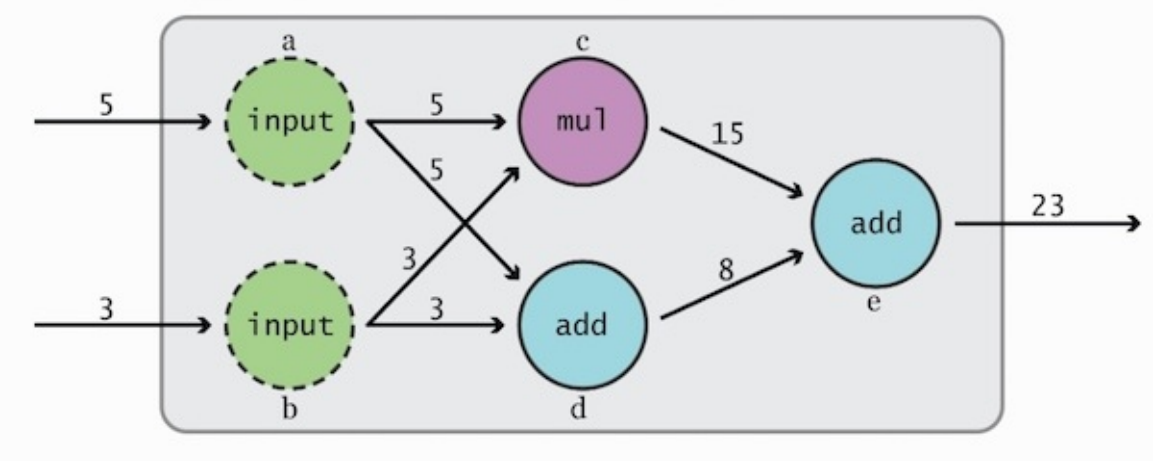
引入tensorflow库并查看版本，确保已安装最新的稳定版（如1.2.0）。

```
from __future__ import print_function, division
import tensorflow as tf

print('Loaded TF version', tf.__version__)
# Output: Loaded TF version 1.2.0
```

为了兼容Python2和Python3的 `print` 和 `division` 函数，建议每个使用到的文件都引入 `from __future__ import print_function, division` 。

简单示例

以一个简单的例子来看一下如何使用tensorflow



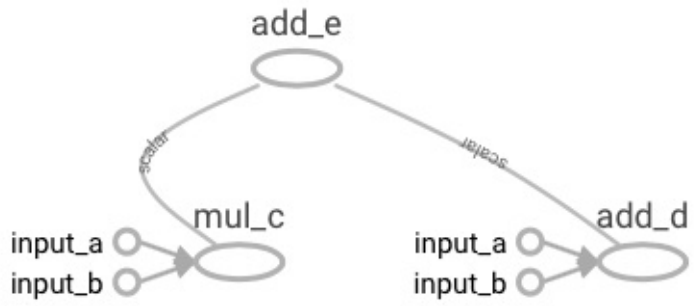
这个示例其实就是计算了 `(5*3)+(5+3)` 的结果

```
# 定义两个常量，值分别为5和3
a = tf.constant(5, name="input_a")
b = tf.constant(3, name="input_b")
# 定义运算操作
c = tf.multiply(a, b, name="mul_c")
d = tf.add(a, b, name="add_d")
e = tf.add(c, d, name="add_e")

# 创建会话，并开始运算
# 注意会话是必需的，会话开始前不会进行任何运算
with tf.Session() as sess:
    print(sess.run(e)) # Output => 23
    writer = tf.summary.FileWriter("./hello_graph", sess.graph)
    writer.flush()
    writer.close()
```

接着，可以启动tensorboard来查看这个Graph（在jupyter notebookt中可以执行 `!tensorboard --logdir="hello_graph"` ）：

```
tensorboard --logdir="hello_graph"
```

打开网页 `http://localhost:6006` 并切换到GRAPHS标签，可以看到生成的Graph：



输入数组示例

对于前面的示例，也可以使用数组作为输入

```
import tensorflow as tf

a = tf.constant([5,3], name="input_a")
b = tf.reduce_prod(a, name="prod_b")
c = tf.reduce_sum(a, name="sum_c")
d = tf.add(c, b, name="add_d")

with tf.Session() as sess:
    print sess.run(d) # => 23
```

基本类型

Tensorflow以图（Graph）来表示计算任务，图中的节点称之为 `op` （即operation）。每个节点包括0个或多个Tensor。为了进行计算，图必须在会话中启动，会话将图的 `op` 分发到CPU、GPU等设备并在执行后返回新的Tensor。

图（Graph）和会话（Session）

如果不指定图，tensorflow会自动创建一个，可以通过 `tf.get_default_graph()` 来获取这个默认图。

```
graph = tf.Graph()
with graph.as_default():
    value1 = tf.constant([1., 2.])
    value2 = tf.Variable([3., 4.])
    result = value1*value2
```

启动图时，需要创建会话，并在改会话中启动：

```
# 使用自定义图
with tf.Session(graph=graph) as sess:
    tf.global_variables_initializer().run()
    print sess.run(result)
    print result.eval()
```

使用默认图

```
sess = tf.Session()

# 在会话中执行图
print(sess.run(product))

# 使用完毕关闭会话
sess.close()
```

会话使用完成后必须关闭，可以调用 `sess.close()` ，也可以使用 `with` 代码块

```
with tf.Session() as sess:
    print(sess.run(product))
```

会话在计算图时会自动检测设备，并在有GPU设备的机器上自动使用GPU。但多个GPU时，Tensorflow只会使用第一个GPU设备，要使用其他设备必须指定:

```
with tf.Session() as sess:
    with tf.device("/gpu:1"):
        print(sess.run(product))
```

在IPython等交互式环境中，可以使用 `tf.InteractiveSession` 代替 `tf.Session` 。这样，可以直接调用 `Tensor.eval()` 和 `Operation.run()` 方法，非常方便。

Tensor

Tensorflow中所有的数据都称之为Tensor，可以是一个变量、数组或者多维数组。
Tensor有几个重要的属性:

- Rank：Tensor的纬数，比如
  - scalar rank=0
  - vector rank=1
  - matrix rank=2
- 类型：数据类型，比如
  - tf.float32
  - tf.uint8
- Shape：Tensor的形状，比如
  - vector shape=[D0]
  - matrix shape=[D0, D1]

Rank与Shape的关系如下表所示

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

常量（Constant）

常量即计算过程中不可变的类型，如

```
a = tf.constant(2)
b = tf.constant(3)

with tf.Session() as sess:
    print sess.run(a+b)  # Output => 5
```

变量（Variable）

变量在计算过程中是可变的，并且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。

如果只想在tf外手动更新变量，那需要声明变量是不可训练的，比如`not_trainable = tf.Variable(0, trainable=False)`。

```
v1 = tf.Variable(10)
v2 = tf.Variable(5)

with tf.Session() as sess:
    # variables must be initialized first.
    tf.global_variables_initializer().run(session=sess)
    print(sess.run(v1+v2)) # Output => 15
```

占位符（Placeholder）

占位符用来给计算图提供输入，常用于传递训练样本。需要在 `Session.run()` 时通过feed绑定。

```
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)

# Define some operations
add = tf.add(a, b)
mul = tf.multiply(a, b)

with tf.Session() as sess:
    print (sess.run(add, feed_dict={a: 2, b: 3}))  # ==> 5
    print (sess.run(mul, feed_dict={a: 2, b: 3}))  # ==> 6
```

数据类型

Tensorflow有着丰富的数据类型，比如 `tf.int32` , `tf.float64` 等，这些类型跟numpy是一致的。

```
import tensorflow as tf
import numpy as np

a = np.array([2, 3], dtype=np.int32)
b = np.array([4, 5], dtype=np.int32)
# Use `tf.add()` to initialize an "add" Operation
c = tf.add(a, b)

with tf.Session() as sess:
    print sess.run(c) # ==> [6 8]
```

`tf.convert_to_tensor(value, dtype=tf.float32)` 是一个非常有用的转换函数，一般用来构造新的Operation。它还可以同时接受python原生类型、numpy数据以及Tensor数据。

数学计算

Tensorflow内置了很多的数学计算操作，包括常见的各种数值计算、矩阵运算以及优化算法等。

```
import tensorflow as tf
# 使用交互式会话方便展示
sess = tf.InteractiveSession()

x = tf.constant([[2, 5, 3, -5],
                 [0, 3,-2,  5],
                 [4, 3, 5,  3],
                 [6, 1, 4,  0]])
y = tf.constant([[4, -7, 4, -3, 4],
                 [6, 4,-7,  4, 7],
                 [2, 3, 2,  1, 4],
                 [1, 5, 5,  5, 2]])

floatx = tf.constant([[2., 5., 3., -5.],
                      [0., 3.,-2.,  5.],
                      [4., 3., 5.,  3.],
                      [6., 1., 4.,  0.]])

print (tf.transpose(x).eval())
print (tf.matmul(x, y).eval())
print (tf.matrix_determinant(tf.to_float(x)).eval())
print (tf.matrix_inverse(tf.to_float(x)).eval())
print (tf.matrix_solve(tf.to_float(x), [[1],[1],[1],[1]]).eval())
```

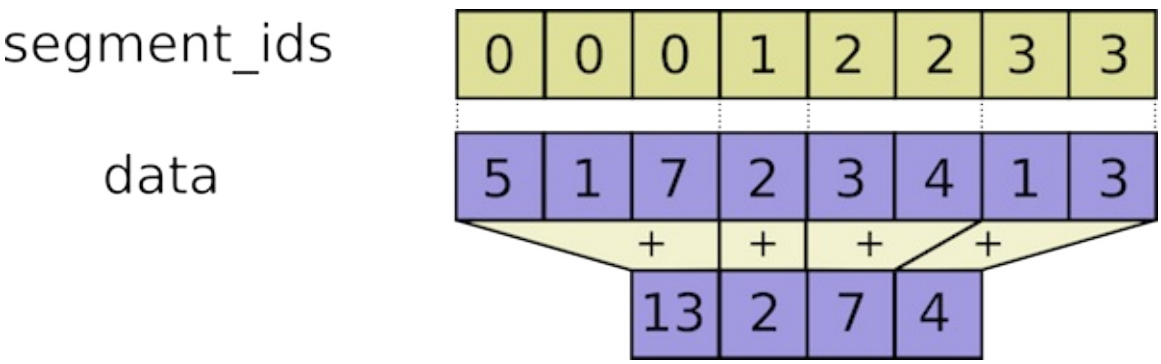
Reduction

Reduction对指定的维度进行操作，并返回降维后的结果：

```
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.constant([[1,  2, 3],
                 [3,  2, 1],
                 [-1,-2,-3]])

boolean_tensor = tf.constant([[True,  False, True],
                 [False, False, True],
                 [True, False, False]])

print (tf.reduce_prod(x).eval()) # => -216
print (tf.reduce_prod(x, reduction_indices=1).eval()) # => [6,6,-6]
print (tf.reduce_min(x, reduction_indices=1).eval()) # => [ 1  1 -3]
print (tf.reduce_max(x, reduction_indices=1).eval()) # => [ 3  3 -1]
print (tf.reduce_mean(x, reduction_indices=1).eval()) # => [ 2  2 -2]

# Computes the "logical and" of elements
print (tf.reduce_all(boolean_tensor, reduction_indices=1).eval()) # =
> [False False False]

# Computes the "logical or" of elements
print (tf.reduce_any(boolean_tensor, reduction_indices=1).eval()) # =
> [ True  True  True]
```

Segmentation

Segmentation根据指定的 `segment_ids` 对输入分段进行计算操作，并返回降维后的结果：



```
import tensorflow as tf
sess = tf.InteractiveSession()

seg_ids = tf.constant([0,1,1,2,2]); # Group indexes : 0|1,2|3,4
x = tf.constant([[2, 5, 3, -5],
                 [0, 3,-2,  5],
                 [4, 3, 5,  3],
                 [6, 1, 4,  0],
                 [6, 1, 4,  0]])

print (tf.segment_sum(x, seg_ids).eval())
print (tf.segment_prod(x, seg_ids).eval())
print (tf.segment_min(x, seg_ids).eval())
print (tf.segment_max(x, seg_ids).eval())
print (tf.segment_mean(x, seg_ids).eval())
```

Sequence

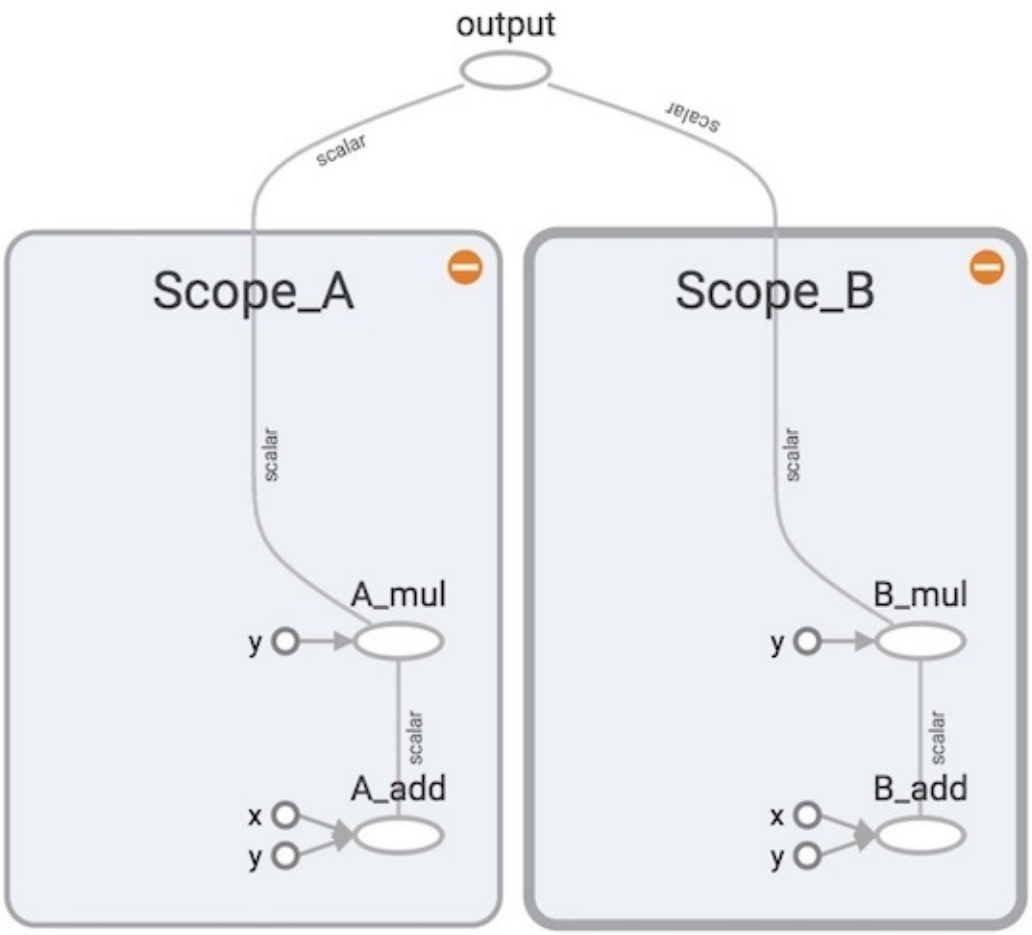
序列比较和索引提取操作。

```
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.constant([[2, 5, 3, -5],
                 [0, 3,-2,  5],
                 [4, 3, 5,  3],
                 [6, 1, 4,  0]])
listx = tf.constant([1,2,5,3,4,5,6,7,8,3,2])
boolx = tf.constant([[True,False], [False,True]])

# 返回各列最小值的索引
print(tf.argmin(x, 0).eval()) # ==>  [1 3 1 0]

# 返回各行最大值的索引
print(tf.argmax(x, 1).eval()) # ==> [1 3 2 0]

# 返回Tensor为True的位置
# ==> [[0 0]
#      [1 1]]
print(tf.where(boolx).eval())

# 返回唯一化数据
print(tf.unique(listx)[0].eval()) # ==> [1 2 5 3 4 6 7 8]
```

Name Scope

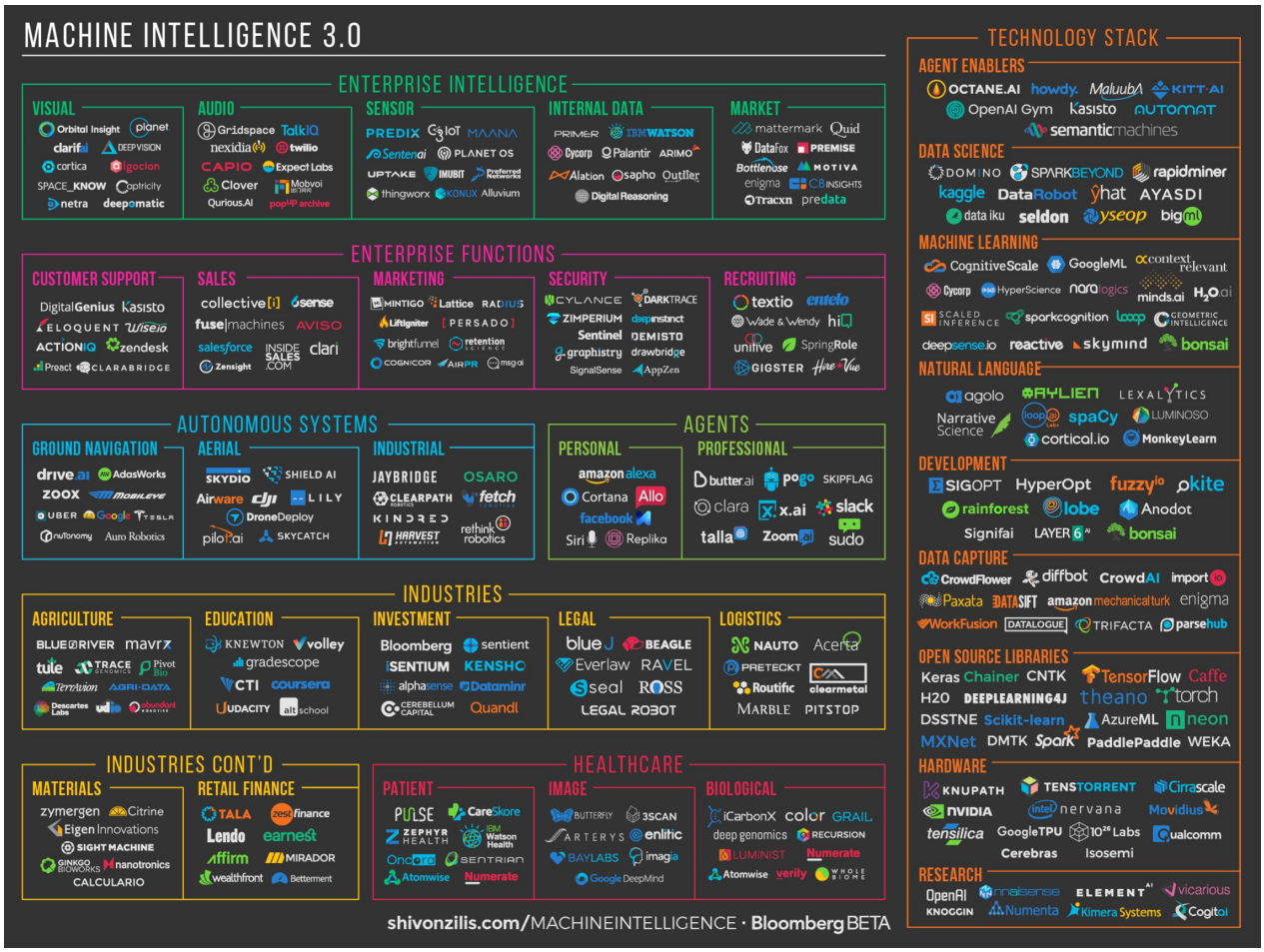
Name scopes可以把复杂操作分成小的命名块，方便组织复杂的图，并方便在TensorBoard展示。

```
import tensorflow as tf

with tf.name_scope("Scope_A"):
    a = tf.add(1, 2, name="A_add")
    b = tf.multiply(a, 3, name="A_mul")

with tf.name_scope("Scope_B"):
    c = tf.add(4, 5, name="B_add")
    d = tf.multiply(c, 6, name="B_mul")

e = tf.add(b, d, name="output")
writer = tf.summary.FileWriter('./name_scope', graph=tf.get_default_g
raph())
writer.close()
```



附录

机器智能全景图



> 注：图片来自shivonzilis。