

深度学习利器

TensorFlow 程序设计

武维 著



Geekbang | InfoQ
极客邦科技

序言

在工业界，TensorFlow 比其他框架更具有优势。TensorFlow 支持异构设备的分布式计算，使得上千万、上亿数据量的模型能够有效地利用机器资源进行训练。TensorFlow 支持卷积神经网络、循环神经网络，这些都是在计算机视觉、语音识别、自然语言处理方面最流行的深度神经网络。TensorFlow 支持从研究团队快速迁移学习模型到生产团队，实现了研究团队发布模型，生产团队验证模型，构建起了模型研究到生产实践的桥梁。TensorFlow 支持直接面向终端用户的移动端（Android 系统）以及一些智能产品的嵌入式开发。另外，TensorFlow 有出色的版本管理和详细的官方文档。

本书 TensorFlow 程序设计中的关键技术主要包括以下几个方面：

1. TensorFlow 编程基础及实践：主要包括 TensorFlow 变量、TensorFlow 应用架构、TensorFlow 可视化技术、GPU 使用、以及

HDFS集成使用等。

2. TensorFlow系统架构及C/C++编程API：主要包括Client, Master, Worker, Kernel的相关系统组件及运行方式，以及采用C++ API去训练模型，提供更好的运算性能及更好地控制GPU内存的分配。
3. 分布式TensorFlow技术：主要包括分布式TensorFlow编程API；分布式TensorFlow MNIST模型；梯度向降法在分布式TensorFlow中的性能分析，包括Async-SGD, Sync-SGD， Sync-SGDwith backups算法。
4. TensorFlow与卷积神经网络：主要包括卷积神经网络的特征图、卷积核，池化操作等关键技术；使用TensorFlow API构建卷积神经网络；TensorFlow Cifar10, InceptionV3及Vgg19模型的架构和代码。
5. TensorFlow与自然语言处理模型：主要包括Word2Vec数学原理；近义词模型；循环神经网络（Recurrent Neural Network, RNN）技术原理；长短期记忆网络（Long Short-Term Memory, LSTM）技术原理；TensorFlow语言预测模型；TensorFlow的机器翻译模型。

TensorFlow 在智能终端中的应用：基于看花识名 APP，讲解了 TensorFlow 在 Android 智能终端中的应用。主要包括：模型训练；模型参数量化处理；TensorFlowAndroid 开发环境的构建及相关开发 A P I 。

作者简介

武维（邮箱：3381209@qq.com），博士，系统架构师，主要从事大数据，深度学习，云计算等领域的研发工作。

助力人工智能落地

2018.01.13 – 01.14 · 北京国际会议中心

近年来，获得投资界助力的AI市场发展迅猛，人工智能正在渗透到各行各业。过了“尝鲜期”，大多数企业开始发力AI落地：

- 1 如何用机器学习实现2亿月活跃用户？
- 2 如何基于人工智能为用户精准推荐他们最喜欢的商品？
- 3 如何通过深度学习网络结构使准确度提升 11% ?
- 4 如何优化风控模型来解决智能金融带来的安全问题？
- 5 如何通过机器学习等 AI 技术来提高运营效率？
- 6 如何解决VR直播中高码率带来的“三高”问题？

为了帮助企业摆脱“落地难”的困扰，InfoQ中国为大家梳理了整个AI产业生态链，并瞄准全球顶尖AI落地案例策划了AICon全球人工智能技术大会。大会将精选30+国内外AI技术专家共享他们的落地痛点及填坑经验。

演讲嘉宾



颜水成
360人工智能研究院
院长及首席科学家



洪亮劼
Etsy
数据科学主管



张浩
饿了么
技术副总裁



尹大朏
摩拜单车
首席科学家



裴少芳
iTutorGroup
大数据部总监
张瑞
知乎
机器学习团队负责人



杨骥
国美在线
大数据中心副总监



胡时伟
第四范式
首席架构师



胡南炜
微博
机器学习计算和
服务平台负责人



吴甘沙
驭势科技
联合创始人&CEO



陈伟
搜狗
语音交互技术中心
研发总监



张重阳
微信小程序
商业技术高级研究员

8折 限时购票，立减720元
团 购 享 受 更 多 优 惠



TensorFlow 实战



郑泽宇

才云科技联合创始人/首席大数据科学家
前谷歌高级工程师

课程大纲

深度学习与TensorFlow简介
深层神经网络解决MNIST问题
神经网络优化方法
卷积神经网络
循环神经网络
TensorFlow加速



现在报名，立减 **300** 元
加小助手，咨询课程

目录

- 05 TensorFlow 编程基础及实践
- 16 TensorFlow 系统架构及 C/C++ 编程 API
- 27 分布式 TensorFlow 技术
- 38 TensorFlow 与卷积神经网络
- 51 TensorFlow 与自然语言处理模型
- 64 TensorFlow 在智能终端中的应用



AI前线 (ID: ai-front)

关注技术落地
探寻 AI 应用场景

每周一节技术分享公开课
助力你全面拥抱人工智能技术

InfoQ



关注AI前线公众号

TensorFlow 编程基础及实践



深度学习及 TensorFlow 简介

深度学习目前已经被应用到图像识别，语音识别，自然语言处理，机器翻译等场景并取得了很好的行业应用效果。至今已有数种深度学习框架，如 TensorFlow、Caffe、Theano、Torch、MXNet，这些框架都能够支持深度神经网络、卷积神经网络、深度信念网络和递归神经网络等模型。TensorFlow 最初由 Google Brain 团队的研究员和工程师研发，目前已成为 GitHub 上最受欢迎的机器学习项目。

TensorFlow 开源一周年以来，已有 500+contributors，以及 11000+

个 commits。目前采用 TensorFlow 平台，在生产环境下进行深度学习的公司有 ARM、Google、UBER、DeepMind、京东等公司。目前谷歌已把 TensorFlow 应用到很多内部项目，如谷歌语音识别，GMail，谷歌图片搜索等。TensorFlow 主要特性如下所述。

- 使用灵活：TensorFlow 是一个灵活的神经网络学习平台，采用图计算模型，支持 High-Level 的 API，支持 Python、C++、Go、Java 接口。
- 跨平台：TensorFlow 支持 CPU 和 GPU 的运算，支持台式机、服务器、移动平台的计算。并从 r0.12 版本支持 Windows 平台。
- 产品化：TensorFlow 支持从研究团队快速迁移学习模型到生产团队。实现了研究团队发布模型，生产团队验证模型，构建起了模型研究到生产实践的桥梁。
- 高性能：TensorFlow 中采用了多线程，队列技术以及分布式训练模型，实现了在多 CPU、多 GPU 的环境下分布式训练模型。

本文主要介绍 TensorFlow 一些关键技术的使用实践，包括 TensorFlow 变量、TensorFlow 应用架构、TensorFlow 可视化技术、GPU 使用，以及 HDFS 集成使用。

TensorFlow 变量

TensorFlow 中的变量在使用前需要被初始化，在模型训练中或训练完成后可以保存或恢复这些变量值。下面介绍如何创建变量，初始化变量，保存变量，恢复变量以及共享变量。

```
# 创建模型的权重及偏置
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
# 指定变量所在设备为CPU:0
with tf.device("/cpu:0"):
    v = tf.Variable(...)
# 初始化模型变量
```

```

init_op = tf.global_variables_initializer()
sess=tf.Session()
sess.run(init_op)

#保存模型变量，由三个文件组成model.data, model.index, model.meta
saver = tf.train.Saver()
saver.restore(sess, "/tmp/model")

#恢复模型变量
saver.restore(sess, "/tmp/model")

```

在复杂的深度学习模型中，存在大量的模型变量，并且期望能够一次性地初始化这些变量。TensorFlow 提供了 `tf.variable_scope` 和 `tf.get_variable` 两个 API，实现了共享模型变量。`tf.get_variable(<name>, <shape>, <initializer>)`：表示创建或返回指定名称的模型变量，其中 `name` 表示变量名称，`shape` 表示变量的维度信息，`initializer` 表示变量的初始化方法。`tf.variable_scope(<scope_name>)`：表示变量所在的命名空间，其中 `scope_name` 表示命名空间的名称。共享模型变量使用示例如下：

```

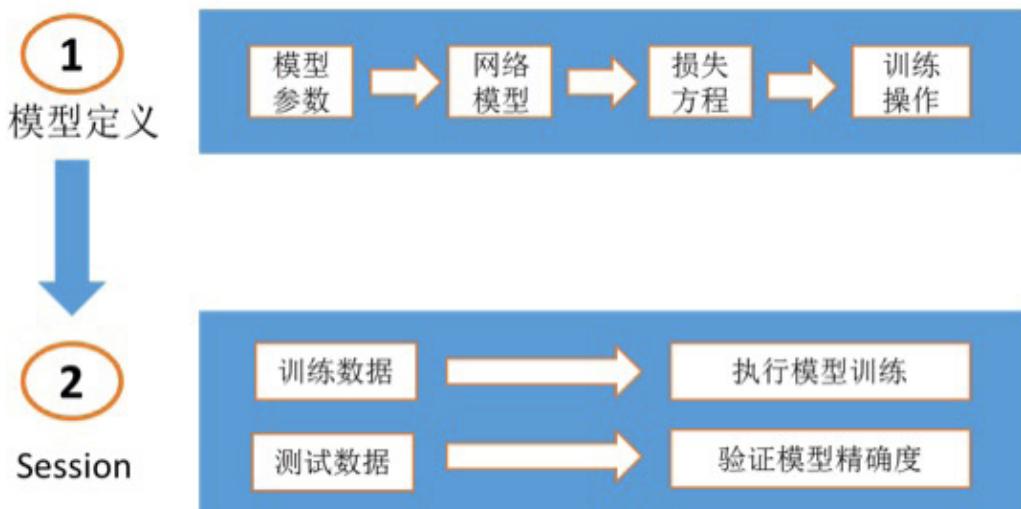
#定义卷积神经网络运算规则，其中weights和biases为共享变量
def conv_relu(input, kernel_shape, bias_shape):
    # 创建变量"weights".
    weights = tf.get_variable("weights", kernel_shape, initializer=tf.random_
normal_initializer())
    # 创建变量 "biases".
    biases = tf.get_variable("biases", bias_shape, initializer=tf.constant_
initializer(0.0))
    conv = tf.nn.conv2d(input, weights, strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv + biases)

#定义卷积层，conv1和conv2为变量命名空间
with tf.variable_scope("conv1"):
    # 创建变量 "conv1/weights", "conv1/biases".
    relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])
    with tf.variable_scope("conv2"):
        # 创建变量 "conv2/weights", "conv2/biases".
        relu1 = conv_relu(relu1, [5, 5, 32, 32], [32])

```

TensorFlow 应用架构

TensorFlow 的应用架构主要包括模型构建，模型训练，及模型评估三个方面。模型构建主要指构建深度学习神经网络，模型训练主要指在 TensorFlow 会话中对训练数据执行神经网络运算，模型评估主要指根据测试数据评估模型精确度。如下图所示：



网络模型，损失方程，模型训练操作定义示例如下：

```

#两个隐藏层，一个logits输出层
hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
logits = tf.matmul(hidden2, weights) + biases

#损失方程，采用softmax交叉熵算法
cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits( logits, labels,
name='xentropy')

loss = tf.reduce_mean(cross_entropy, name='xentropy_mean')

#选定优化算法及定义训练操作
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
global_step = tf.Variable(0, name='global_step', trainable=False)
train_op = optimizer.minimize(loss, global_step=global_step)

模型训练及模型验证示例如下：

#加载训练数据，并执行网络训练
  
```

```

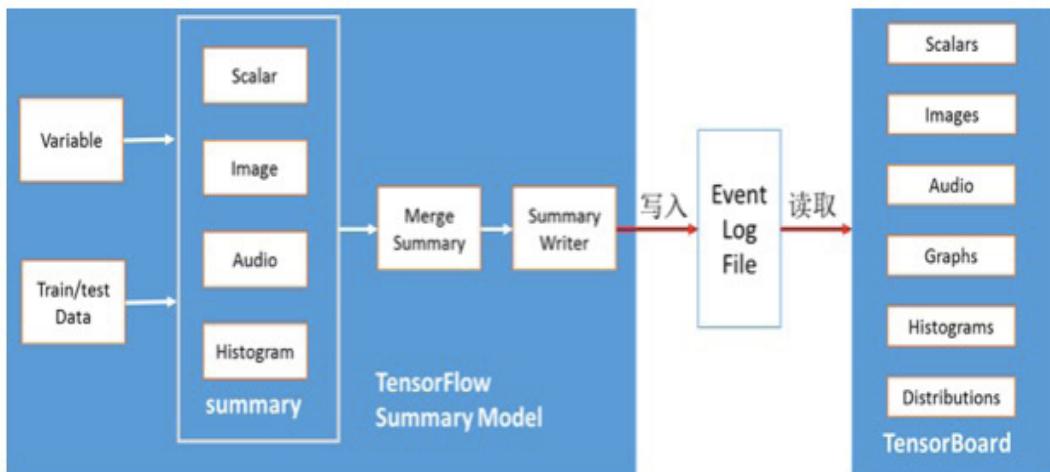
for step in xrange(FLAGS.max_steps):
    feed_dict = fill_feed_dict(data_sets.train, images_placeholder, labels_placeholder)
    _, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)

#加载测试数据，计算模型精确度
for step in xrange(steps_per_epoch):
    feed_dict = fill_feed_dict(data_set, images_placeholder, labels_placeholder)
    true_count += sess.run(eval_correct, feed_dict=feed_dict)

```

TensorFlow 可视化技术

大规模的深度神经网络运算模型是非常复杂的，并且不容易理解运算过程。为了易于理解、调试及优化神经网络运算模型，数据科学家及应用开发人员可以使用 TensorFlow 可视化组件：TensorBoard。TensorBoard 主要支持 TensorFlow 模型可视化展示及统计信息的图表展示。TensorBoard 应用架构如下：



TensorFlow 可视化技术主要分为两部分：TensorFlow 摘要模型及 TensorBoard 可视化组件。在摘要模型中，需要把模型变量或样本数据转换为 TensorFlow summary 操作，然后合并 summary 操作，最后通过 Summary Writer 操作写入 TensorFlow 的事件日志。TensorBoard 通过读取事件日志，进行相关摘要信息的可视化展示，主要包括：Scalar 图、图片

数据可视化、声音数据展示、图模型可视化，以及变量数据的直方图和概率分布图。TensorFlow 可视化技术的关键流程如下所示：

```
#定义变量及训练数据的摘要操作
tf.summary.scalar('max', tf.reduce_max(var))
tf.summary.histogram('histogram', var)
tf.summary.image('input', image_shaped_input, 10)

#定义合并变量操作，一次性生成所有摘要数据
merged = tf.summary.merge_all()

#定义写入摘要数据到事件日志的操作
train_writer = tf.train.SummaryWriter(FLAGS.log_dir + '/train', sess.graph)
test_writer = tf.train.SummaryWriter(FLAGS.log_dir + '/test')

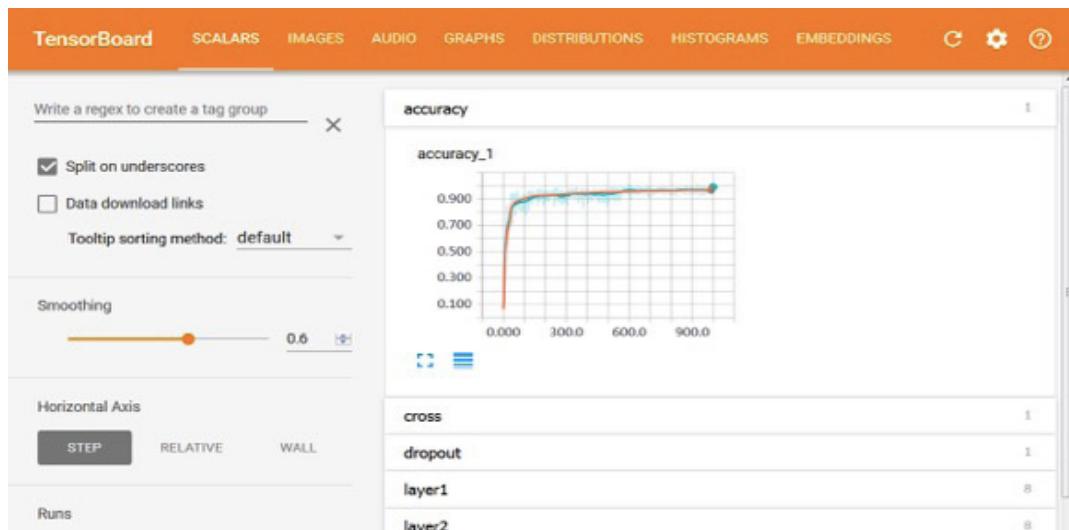
#执行训练操作，并把摘要信息写入到事件日志
summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(True))

train_writer.add_summary(summary, i)

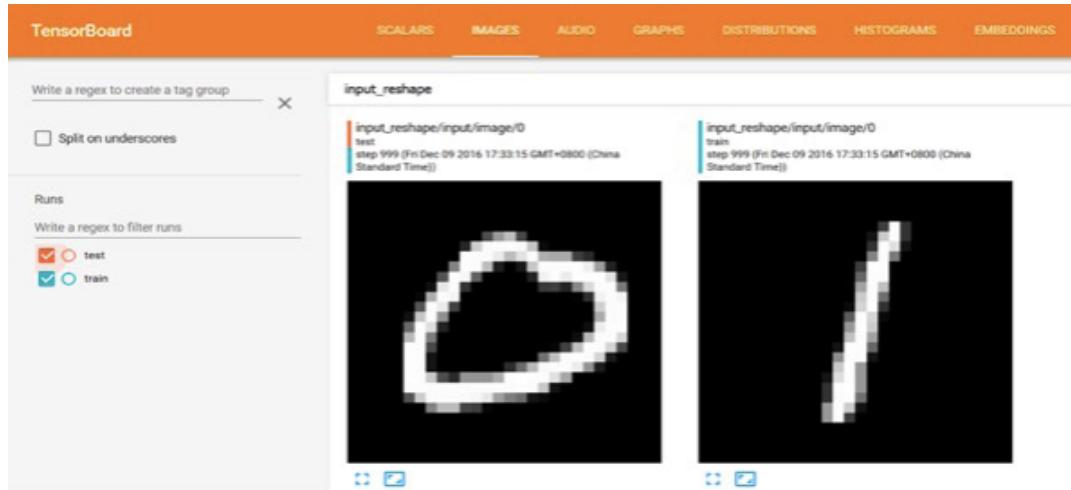
#下载示例code，并执行模型训练
python mnist_with_summaries.py

#启动TensorBoard，TensorBoard的UI地址为http://ip_address:6006
tensorboard --logdir=/path/to/log-directory
```

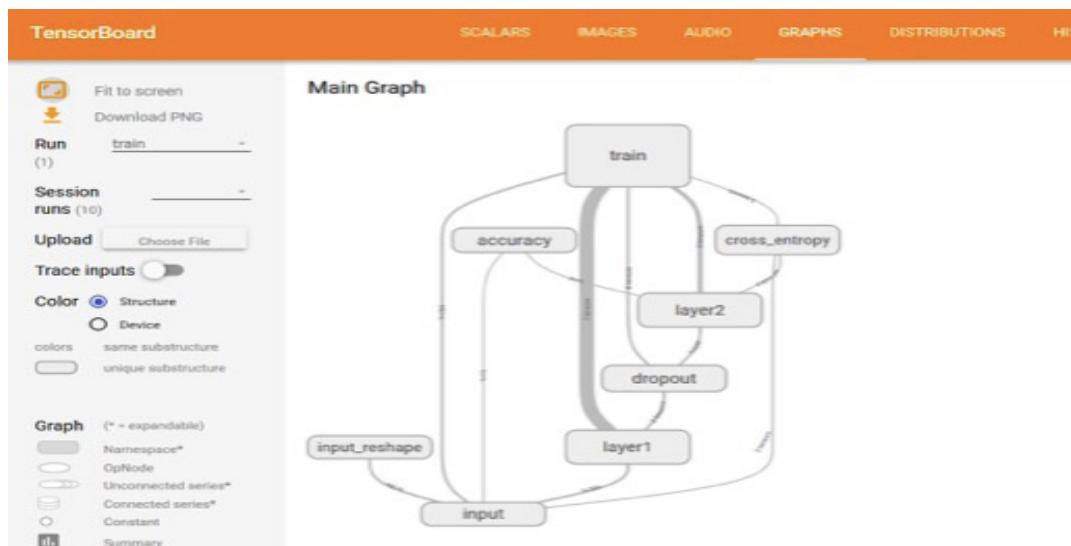
TensorBoard Scalar 图如下所示，其中横坐标表示模型训练的迭代次数，纵坐标表示该标量值，例如模型精确度，熵值等。TensorBoard 支持这些统计值的下载。



TensorFlow Image 摘要信息如下图所示，该示例中显示了测试数据和训练数据中的手写数字图片。



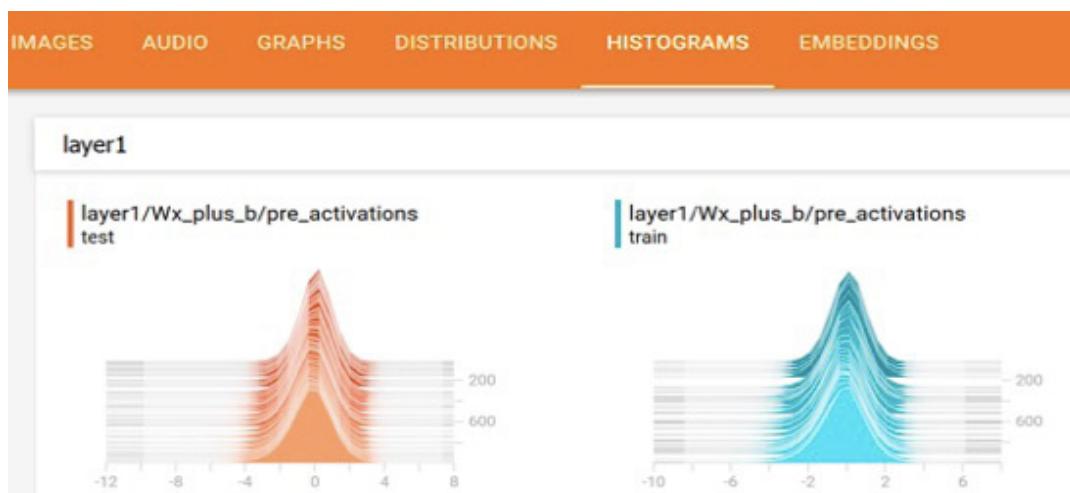
TensorFlow 图模型如下图所示，可清晰地展示模型的训练流程，其中的每个方框表示变量所在的命名空间。包含的命名空间有 input（输入数据），input_reshape（矩阵变换，用于图形化手写数字），layer1（隐含层 1），layer2（隐含层 2），dropout（丢弃一些神经元，防止过拟合），accuracy（模型精确度），cross_entropy（目标函数值，交叉熵），train（训练模型）。例如，input 命名空间操作后的 tensor 数据会传递给 input_reshape，train，accuracy，layer1，cross_entropy 命名空间中的操作。



TensorFlow 变量的概率分布如下图所示，其中横坐标为迭代次数，纵坐标为变量取值范围。图表中的线表示概率百分比，从高到底为 [maximum, 93%, 84%, 69%, 50%, 31%, 16%, 7%, minimum]。例如，图表中从高到底的第二条线为 93%，对应该迭代下有 93% 的变量权重值小于该线对应的目标值。



上述 TensorFlow 变量概率分布对应的直方图如下图所示：



TensorFlow GPU 使用

GPU 设备已经广泛地应用于图像分类，语音识别，自然语言处理，

机器翻译等深度学习领域，并实现了开创性的性能改进。与单纯使用 CPU 相比，GPU 具有数以千计的计算核心，可实现 10-100 倍的性能提升。TensorFlow 支持 GPU 运算的版本为 tensorflow-gpu，并且需要先安装相关软件：GPU 运算平台 CUDA 和用于深度神经网络运算的 GPU 加速库 CuDNN。在 TensorFlow 中，CPU 或 GPU 的表示方式如下所示：

- “/cpu:0”：表示机器中第一个CPU。
- “/gpu:0”：表示机器中第一个GPU卡。
- “/gpu:1”：表示机器中第二个GPU卡。

TensorFlow 中所有操作都有 CPU 和 GPU 运算的实现，默认情况下 GPU 运算的优先级比 CPU 高。如果 TensorFlow 操作没有指定在哪个设备上进行运算，默认会优选采用 GPU 进行运算。下面介绍如何在 TensorFlow 使用 GPU：

```
# 定义使用gpu0执行a*b的矩阵运算，其中a, b, c都在gpu0上执行
with tf.device('/gpu:0'):
    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)

# 通过log_device_placement指定在日志中输出变量和操作所在的设备
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
print sess.run(c)
```

本实验环境下只有一个 GPU 卡，设备的 Device Mapping 及变量操作所在设备位置如下：

```
#设备的Device Mapping
/job:localhost/relica:0/task:0/gpu:0 -> device: 0, name: Tesla K20c, pci bus
id: 0000:81:00.0

#变量操作所在设备位置
a: (Const): /job:localhost/relica:0/task:0/gpu:0
b: (Const): /job:localhost/relica:0/task:0/gpu:0
(MatMul)/job:localhost/relica:0/task:0/gpu:0
```

默认配置下，TensorFlow Session 会占用 GPU 卡上所有内存。但 TensorFlow 提供了两个 GPU 内存优化配置选项。config.gpu_options.

allow_growth: 根据程序运行情况，分配 GPU 内存。程序开始的时候分配比较少的内存，随着程序的运行，增加内存的分配，但不会释放已经分配的内存。config.gpu_options.per_process_gpu_memory_fraction: 表示按照百分比分配 GPU 内存，例如 0.4 表示分配 40% 的 GPU 内存。示例代码如下：

```
#定义TensorFlow配置
config = tf.ConfigProto()
#配置GPU内存分配方式
#config.gpu_options.allow_growth = True
#config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config, ...)
```

TensorFlow 与 HDFS 集成使用

HDFS 是一个高度容错性的分布式系统，能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。TensorFlow 与 HDFS 集成示例如下：

```
#配置JAVA和HADOOP环境变量
source $HADOOP_HOME/libexec/hadoop-config.sh
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64/server
#执行TensorFlow运行模型
CLASSPATH=$(($HADOOP_HDFS_HOME/bin/hadoop classpath --glob)) python tensorflow_
model.py
#在TensorFlow模型中定义文件的读取队列
filename_queue = tf.train.string_input_producer(["hdfs://namenode:8020/path/to/
file1.csv", "hdfs://namenode:8020/path/to/file2.csv"])
#从文件中读取一行数据，value为所对应的行数据
reader = tf.TextLineReader()
key, value = reader.read(filename_queue)
# 把读取到的value值解码成特征向量，record_defaults定义解码格式及对应的数据类型
record_defaults = [[1], [1], [1], [1], [1]]
col1, col2, col3, col4, col5 = tf.decode_csv(value, record_defaults=record_
defaults)
features = tf.pack([col1, col2, col3, col4])
with tf.Session() as sess:
```

```
# 定义同步对象，并启动相应线程把HDFS文件名插入到队列
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord)
for i in range(1200):
    # 从文件队列中读取一行数据
    example, label = sess.run([features, col5])
# 请求停止队列的相关线程（包括进队及出队线程）
coord.request_stop()
# 等待队列中相关线程结束（包括进队及出队线程）
coord.join(threads)
```

TensorFlow 系统架构及 C/C++ 编程 API



2015 年 11 月 9 日谷歌开源了人工智能平台 TensorFlow，同时成为 2015 年最受关注的开源项目之一。经历了从 v0.1 到 v0.12 的 12 个版本迭代后，谷歌于 2017 年 2 月 15 日发布了 TensorFlow 1.0 版本，并同时在美国加州山景城举办了首届 TensorFlow Dev Summit 会议。

TensorFlow 1.0 及 Dev Summit (2017) 回顾

和以往版本相比，TensorFlow 1.0 的特性改进主要体现在以下几个方面：

- 速度更快：TensorFlow 1.0 版本采用了 XLA 的编译技术，改进了

TensorFlow的运行性能及内存利用。从Benchmark问题的测试结果来看，对单机Inception v3模型，实现了在单机8 GPUs上7.3倍的运算加速；对分布式Inception v3模型，实现了在多机64 GPUs上58倍的运算加速。

- 更加灵活：该版本除了支持tf.layers, tf.metrics及tf.losses模型的High-Level API外，实现了对keras (high-level neural networks library) API的全面兼容。
- 更产品化：TensorFlow Python API在v1.0版本中趋于稳定，为产品兼容性打下坚实基础。

在TensorFlow 1.0 版本发布的当天，谷歌公司还举办了TensorFlow 2017 DEV Summit。该日程主要包括以下几个方面的主题演讲：

- XLA (TensorFlow, Compiled)编译技术：介绍采用XLA技术最小化图计算执行时间和最大化利用计算资源，用于减少数据训练和模型结果推断时间。
- Hands-on TensorBoard可视化技术：介绍了如何使用TensorBoard，以及TensorFlow图模型、训练数据的可视化等。
- TensorFlow High-Level API：介绍了使用Layers, Estimators, and Canned Estimators High-Level API定义训练模型。
- Integrating Keras & TensorFlow：介绍了如何在TensorFlow中使用Keras API进行模型定义及训练。
- TensorFlow at DeepMind：介绍了在DeepMind中使用TensorFlow平台的典型案例，包括AlphaGo等应用。
- Skin Cancer Image Classification：介绍了斯坦福医学院使用TensorFlow分类皮肤癌照片，用于医学诊断。
- Mobile and Embedded TensorFlow：介绍了如何把TensorFlow模型运行在移动终端、嵌入式设备，包括安卓，iOS等系统。
- Distributed TensorFlow：系统性地介绍了分布式TensorFlow的相关技术，以及如何应用于大规模模型训练。

- TensorFlow Ecosystem: 讲解了TensorFlow的生态系统，包括生成训练数据，分布式运行TensorFlow和serving models的产品化流程。
- Serving Models in Production with TensorFlow Serving: 系统性讲解了如何在生产环境中应用TensorFlow Serving模型。
- ML Toolkit: 介绍了TensorFlow的机器学习库，如线性回归，KMeans等算法模型的使用。
- Sequence Models and the RNN API: 介绍了如何构建高性能的sequence-to-sequence模型，以及相关API。
- Wide & Deep Learning: 介绍了如何结合Wide模型和Deep模型构建综合训练模型。
- Magenta, Music and Art Generation: 使用增强型深度学习模型生成音乐声音和艺术图片。
- Case Study, TensorFlow in Medicine - Retinal Imaging: 使用TensorFlow机器学习平台对医学视网膜图片进行分类，辅助医学诊断。

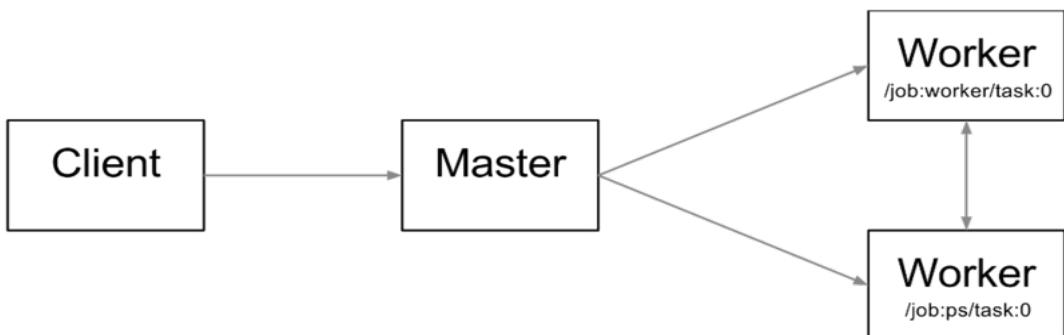
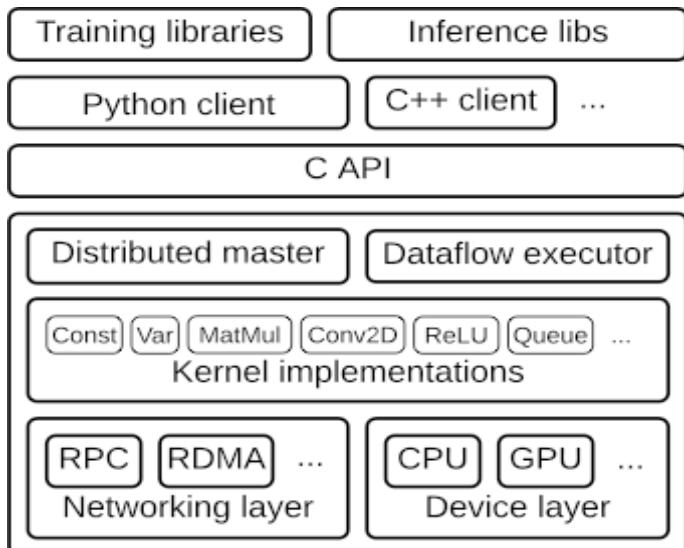
TensorFlow 系统架构

TensorFlow 作为分布式机器学习平台，主要架构如下图所示。RPC 和 RDMA 为网络层，主要负责传递神经网络算法参数。CPU 和 GPU 为设备层，主要负责神经网络算法中具体的运算操作。Kernel 为 TensorFlow 中算法操作的具体实现，如卷积操作，激活操作等。Distributed Master 用于构建子图；切割子图为多个分片，不同的子图分片运行在不同的设备上；Master 还负责分发子图分片到 Executor/Work 端。Executor/Work 在设备（CPUs, GPUs, etc.）上，调度执行子图操作；并负责向其它 Worker 发送和接收图操作的运行结果。C API 把 TensorFlow 分割为前端和后端，前端（Python/C++/Java Client）基于 C API 触发 TensorFlow 后端程序运行。Training libraries 和 Inference libs 是模型训练和推导的库函数，为用户开

发应用模型使用。

下图 为 Client、Master 及 Worker 的内部工作原理。"/job:worker/task:0" 和 "/job:ps/task:0" 表示 worker 中的执行服务。"job:ps" 表示参数服务器，用于存储及更新模型参数。"job:worker" 用于优化模型参数，并发参数发送到参数服务器上。

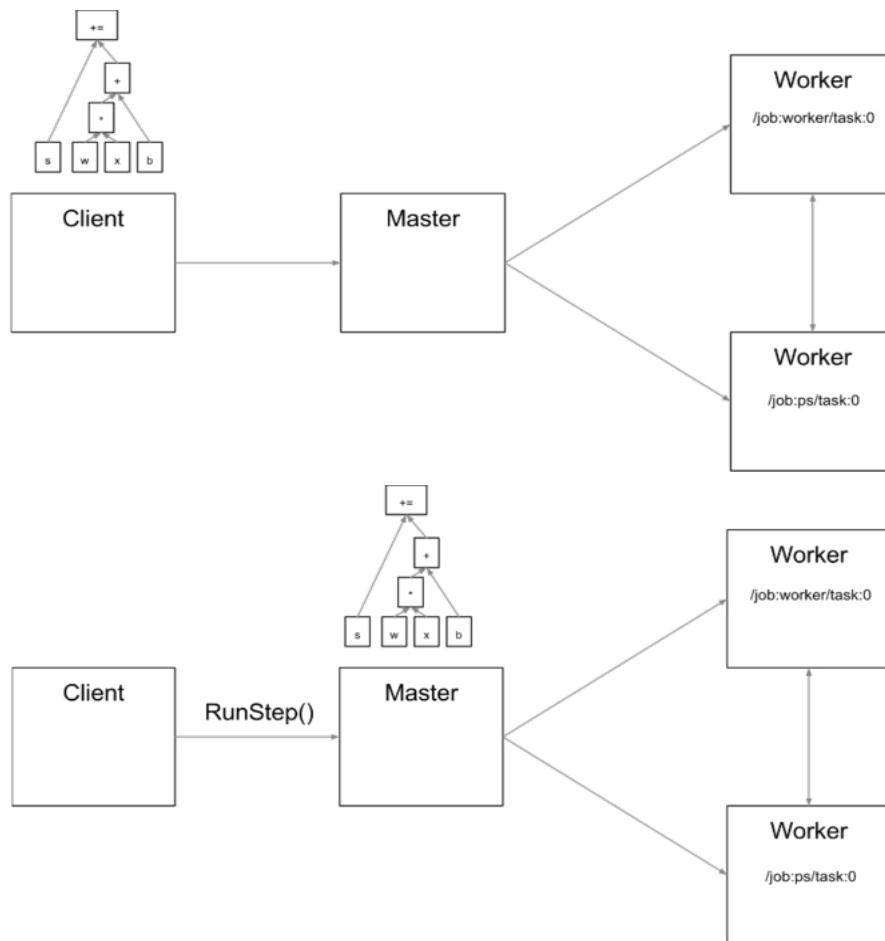
Distributed Master 和 Worker Service 只存在于分布式 TensorFlow 中。单机版本的 TensorFlow 实现了 Local 的 Session，通过本地进程的内部通讯实现上述功能。



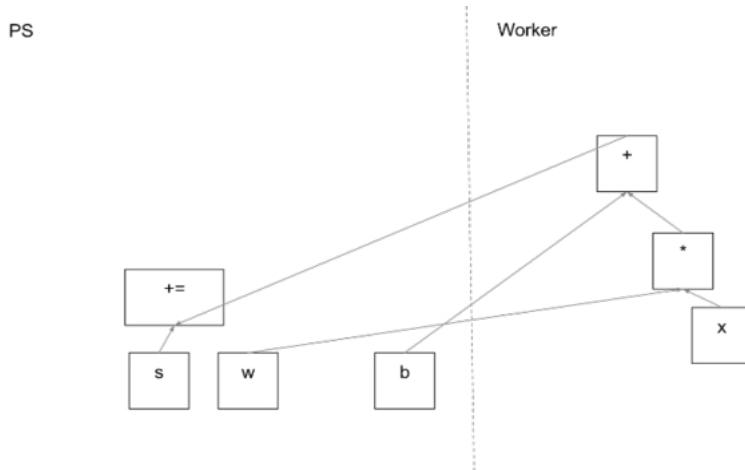
用户编写 TensorFlow 应用程序生成计算图，Client 组件会创建 Session，并通过序列化技术，发送图定义到 Distributed Master 组件。下图中，Client 创建了一个 $s = w * x + b$ 的图计算模型。

当 Client 触发 Session 运算的时候，Master 构建将要运行的子图。并根据设备情况，切割子图为多个分片。下面为 Master 构建的运行子图。

接着切割子图，把模型参数分组在参数服务器上，图计算操作分组在

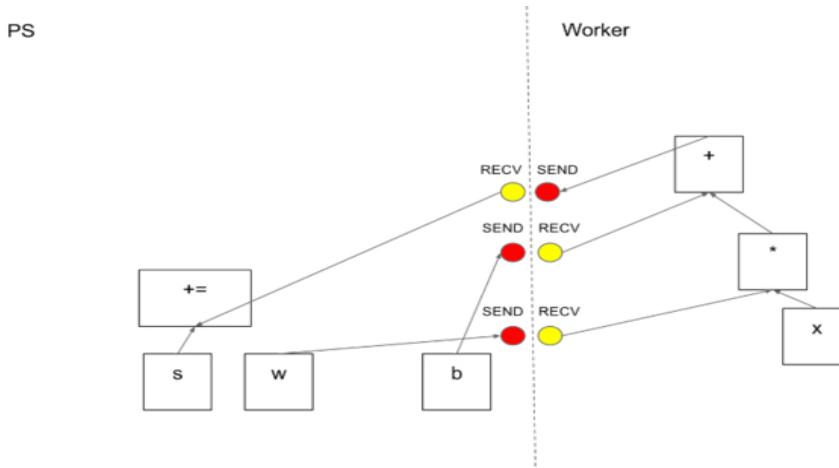


运算 Worker 上。下图为一种可行的图切割策略：

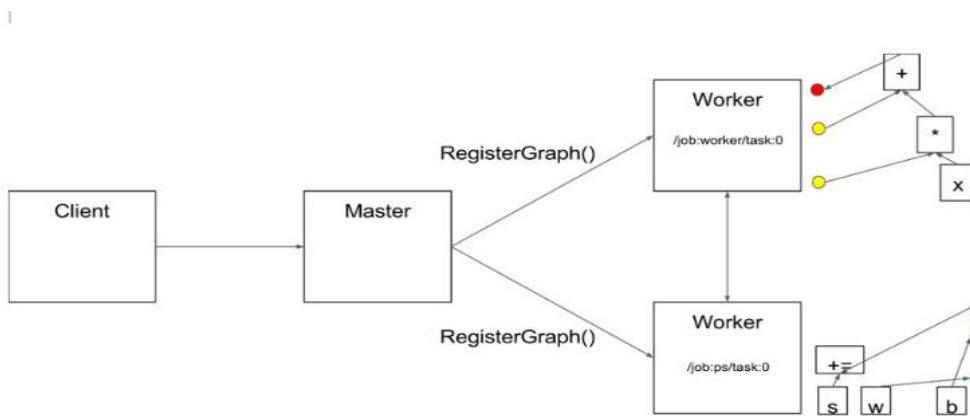


Distributed Master 会根据模型参数的分区情况进行切割边，在 Task 间插入发送和接收 Tensor 信息的通信节点，如下图所示。

接着 Distributed Master 通过 RegisterGraph 方法发送子图分片给



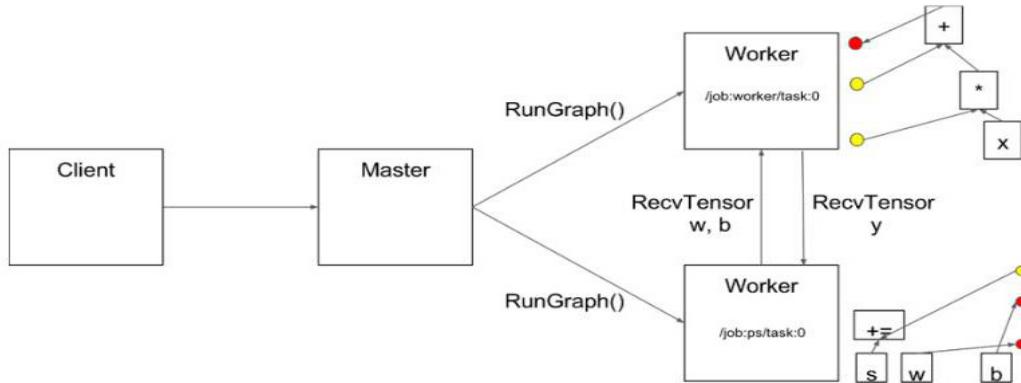
Task, 如下图所示。



Master 通过 `RunGraph` 触发子图运算，Worker 会使用 GPU/CPU 运算设备执行 TensorFlow Kernel 运算。在本节点的 CPU 和 GPU 之间，使用 `cudaMemcpyAsync` 传输数据；在本节点 GPU 和 GPU 之间，使用 peer-to-peer DMA 传输数据，避免通过 CPU 复制数据。TensorFlow 使用 gRPC (TCP) 和 RDMA (Converged Ethernet) 技术，实现 Worker 间的数据通信及传输，如下图所示。

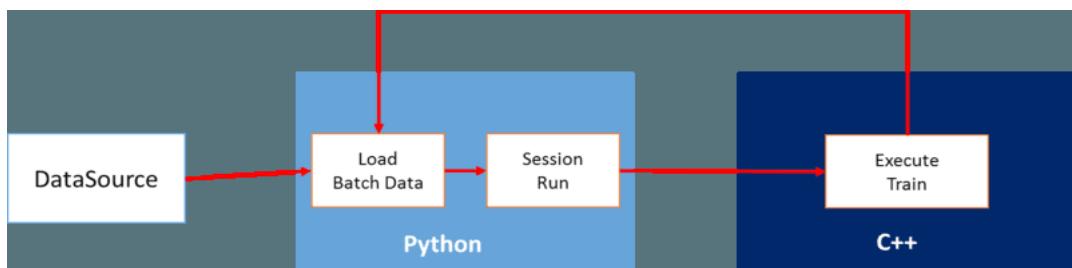
高性能程序设计

TensorFlow 内核采用 C/C++ 开发，并提供了 C++, Python, Java, Go 语言的 Client API。特别是 Python API，是目前主流的 TensorFlow 模型开发接口。但为什么还需要采用 C++ API 去训练模型呢？本文基于如



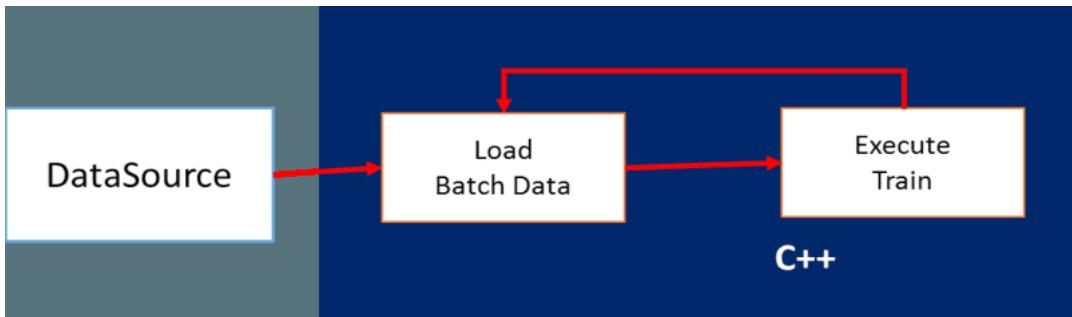
下两点考虑，首先当我们采用 Python API 去训练模型的时候，需要不断地用 Python API 调用 C/C++ 底层接口，重复的接口调用一定程度上影响了程序的执行性能。更为重要的是，在 GPU 上训练模型的时候需要大量的内存交换；如果采用 C++ API 去训练模型，可提供更好的运算性能及更好地控制 GPU 内存的分配。

下图为 Python API 的运算架构：在模型训练的每次迭代中，程序通过 Python API 读取 Batch Data，然后通过 TensorFlow Session Run 接口，传递数据给 C++，并触发神经网络训练。如下图所示：

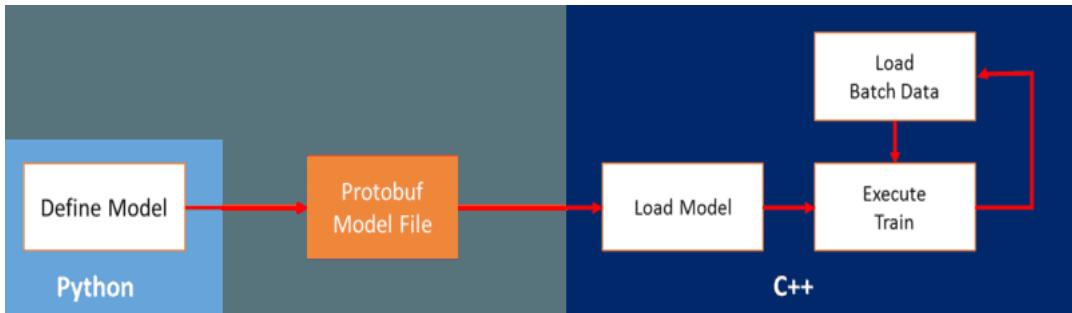


下图为 C++ API 的运算架构：在模型训练的每次迭代中，通过 C++ API 读取 Batch Data 后，直接触发模型训练。减少了不同语言间 API 接口的循环调用及数据传输。如下图所示。

为了采用 C++ API 进行模型训练，我们首先需要编写训练模型，这个编写过程可以采用 Python 语言来完成。我们首先采用 Python API 编写训练模型，然后把图模型转换为 Protobuf 的序列化文件。接着通过 C++ API 加载该模型文件，创建 TensorFlow Session，初始化模型变量，以及



加载训练数据并执行神经网络训练。程序架构如下图所示：



下面为使用 Python API 定义训练模型的示例：

```

with tf.Session() as sess:
    # 定义 Placeholder Tensor 接入训练数据
    x = tf.placeholder(tf.float32, [None, 32], name="x")
    y = tf.placeholder(tf.float32, [None, 8], name="y")
    # 定义训练模型
    w1 = tf.Variable(tf.truncated_normal([32, 16], stddev=0.1))
    b1 = tf.Variable(tf.constant(0.0, shape=[16]))
    w2 = tf.Variable(tf.truncated_normal([16, 8], stddev=0.1))
    b2 = tf.Variable(tf.constant(0.0, shape=[8]))
    a = tf.nn.tanh(tf.nn.bias_add(tf.matmul(x, w1), b1))
    y_out = tf.nn.tanh(tf.nn.bias_add(tf.matmul(a, w2), b2), name="y_out")
    cost = tf.reduce_sum(tf.square(y-y_out), name="cost")
    optimizer = tf.train.AdamOptimizer().minimize(cost, name="train")
    # 定义变量初始化操作
    init = tf.initialize_variables(tf.all_variables(), name='init_all_vars_op')
    # 把图模型转换为 Protobuf 文件
    tf.train.write_graph(sess.graph_def, '.', 'mlp.pb', as_text=False)
  
```

下面为使用 C++ API 加载 Protobuf 图模型，并执行训练的示例：

```
#include "tensorflow/core/public/session.h"
#include "tensorflow/core/graph/default_device.h"
using namespace tensorflow;

int main(int argc, char* argv[]) {
    //Protobuf模型文件名
    std::string graph_definition = "mlp.pb";
    //Tensorflow Session
    Session* session;
    //定义图模型对象
    GraphDef graph_def;
    SessionOptions opts;
    //存储Session会话的运行结果
    std::vector<Tensor> outputs;
    #加载Protobuf模型文件到图模型对象中
    TF_CHECK_OK(ReadBinaryProto(Env::Default(), graph_definition, &graph_def));
    // 默认在gpu 0上执行模型的训练操作
    graph::SetDefaultDevice("/gpu:0", &graph_def);
    //设定GPU显存使用参数
    opts.config.mutable_gpu_options()->set_per_process_gpu_memory_fraction(0.5);
    opts.config.mutable_gpu_options()->set_allow_growth(true);
    //创建TensorFlow会话
    TF_CHECK_OK(NewSession(opts, &session));
    // 加载图对象到会话中
    TF_CHECK_OK(session->Create(graph_def));
    // 执行模型参数初始化操作
    TF_CHECK_OK(session->Run({}, {}, {"init_all_vars_op"}, nullptr));
    //定义模型输入数据，包括数据类型和维度信息
    Tensor x(DT_FLOAT, TensorShape({100, 32}));
    Tensor y(DT_FLOAT, TensorShape({100, 8}));
    //把Tensor转换为矩阵，并初始化Tensor数据
    auto _XTensor = x.matrix<float>();
    auto _YTensor = y.matrix<float>();
    _XTensor.setRandom();
    _YTensor.setRandom();
```

```

for (int i = 0; i < 10; ++i) {

    //执行模型的训练操作，{{"x", x}, {"y", y}}表示输入数据Tensor名称和Tensor对象；{"cost"}表示要获取输出值的操作名称；&outputs表示执行"cost"操作后返回的Tensor对象

    TF_CHECK_OK(session->Run({{"x", x}, {"y", y}}, {"cost"}, {}, &outputs));

    //获取执行“cost”操作后的运算结果

    float cost = outputs[0].scalar<float>()(0);

    std::cout << "Cost: " << cost << std::endl;

    //执行"train"操作

    TF_CHECK_OK(session->Run({{"x", x}, {"y", y}}, {}, {"train"}, nullptr));

// Train

outputs.clear();

}

//关闭Session及删除Session对象

session->Close();

delete session;

return 0;

}

```

当 C++ 程序写好后，编译时候需要链接的头文件，开源已经帮我们整理好了，存放于目录 /usr/lib/python2.7/site-packages/tensorflow/include 下。编译和运行的时候需要链接 libtensorflow_cc.so，可以按照下面的方式编译该库文件：bazel build -c opt //tensorflow:libtensorflow_cc.so --copt=-m64 --linkopt=-m64 --spawn_strategy=standalone --genrule_strategy=standalone --verbose_failures。具体可参考 TensorFlow 源代码的官方编译文档。

总结

本文首先回顾了 TensorFlow 1.0 主要新特性及 TensorFlow 2017 Dev Summit 的主要议程。到目前为止 TensorFlow 的 GitHub Star 排名为 51000+，Fork 排名已达 24000+，有 15000+ commits。并随着 TensorFlow 新版本的不断发布以及新特性的不断增加，TensorFlow 使用更加灵活，运行速度更快，使用方式更产品化，已成为目前主流的深度学习平台之一。

接着介绍了 TensorFlow 的系统架构，包括 Client, Master, Worker, Kernel 的相关概念及运行方式，是一种适合大规模分布式训练的机器学习平台。从上述系统架构中可以看到，TensorFlow 内核采用 C/C++ 开发，当采用 Python API 去训练模型的时候，需要不断地用 Python 调用 C/C++ 底层接口，重复的接口调用一定程度上影响了程序的执行性能。如果有最求高性能运算的朋友，可以尝试用下本文高性能运算章节推荐的方法。

分布式 TensorFlow 技术



TensorFlow 发展及使用简介

2015 年 11 月 9 日谷歌开源了人工智能系统 TensorFlow，同时成为 2015 年最受关注的开源项目之一。TensorFlow 的开源大大降低了深度学习在各个行业中的应用难度。TensorFlow 的近期里程碑事件主要有：

- 2016年11月09日：TensorFlow开源一周年。
- 2016年09月27日：TensorFlow支持机器翻译模型。
- 2016年08月30日：TensorFlow支持使用TF-Slim接口定义复杂模型。

- 2016年08月24日：TensorFlow支持自动学习生成文章摘要模型。
- 2016年06月29日：TensorFlow支持Wide & Deep Learning。
- 2016年06月27日：TensorFlow v0.9发布，改进了移动设备的支持。
- 2016年05月12日：发布SyntaxNet，最精确的自然语言处理模型。
- 2016年04月29日：DeepMind模型迁移到TensorFlow。
- 2016年04月14日：发布了分布式TensorFlow。

TensorFlow是一种基于图计算的开源软件库，图中节点表示数学运算，图中的边表示多维数组（Tensor）。TensorFlow是跨平台的深度学习框架，支持CPU和GPU的运算，支持台式机、服务器、移动平台的计算，并从r0.12版本开始支持Windows平台。Tensorflow提供了各种安装方式，包括Pip安装，Virtualenv安装，Anaconda安装，docker安装，源代码安装。本文主要介绍Pip的安装方式，Pip是一个Python的包安装及管理工具。Linux系统下，使用Pip的安装流程如下：

```
yum install python-pip python-dev  
  
export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/  
tensorflow-0.12.0rc0-cp27-none-linux_x86_64.whl  
  
pip install --upgrade $TF_BINARY_URL
```

安装完毕后，TensorFlow会安装到 /usr/lib/python2.7/site-packages/tensorflow目录下。使用TensorFlow之前，我们需要先熟悉下常用API。

`tf.random_uniform([1], -1.0, 1.0)`: 构建一个tensor，该tensor的shape为[1]，该值符合[-1, 1]的均匀分布。其中[1]表示一维数组，里面包含1个元素。

`tf.Variable(initial_value=None)`: 构建一个新变量，该变量会加入到TensorFlow框架中的图集合中。

`tf.zeros([1])`: 构建一个tensor，该tensor的shape为[1]，里面所有元素为0。

`tf.square(x, name=None)`: 计算tensor的平方值。

`tf.reduce_mean(input_tensor)`: 计算input_tensor中所有元素的均值。

tf.train.GradientDescentOptimizer(0.5): 构建一个梯度下降优化器，0.5为学习速率。学习率决定我们迈向（局部）最小值时每一步的步长，设置的太小，那么下降速度会很慢，设的太大可能出现直接越过最小值的现象。所以一般调到目标函数的值在减小而且速度适中的情况。

optimizer.minimize(loss): 构建一个优化算子操作。使用梯度下降法计算损失方程的最小值。loss 为需要被优化的损失方程。

tf.initialize_all_variables(): 初始化所有 TensorFlow 的变量。

tf.Session(): 创建一个 TensorFlow 的 session，在该 session 种会运行 TensorFlow 的图计算模型。

sess.run(): 在 session 中执行图模型的运算操作。如果参数为 tensor 时，可以用来求 tensor 的值。

下面为使用 TensorFlow 中的梯度下降法构建线性学习模型的使用示例：

```
#导入TensorFlow python API库
import tensorflow as tf
import numpy as np
#随机生成100点 (x, y)
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3
#构建线性模型的tensor变量W, b
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b
#构建损失方程, 优化器及训练模型操作train
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
#构建变量初始化操作init
init = tf.initialize_all_variables()
#构建TensorFlow session
sess = tf.Session()
```

```
# 初始化所有TensorFlow变量
sess.run(init)

# 训练该线性模型，每隔20次迭代，输出模型参数

for step in range(201):
    sess.run(train)

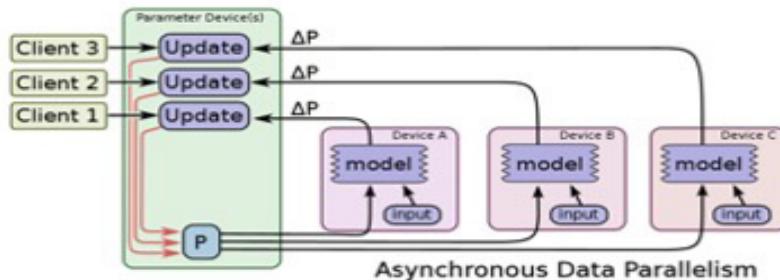
    if step % 20 == 0:
        print(step, sess.run(w), sess.run(b))
```

分布式 TensorFlow 应用架构

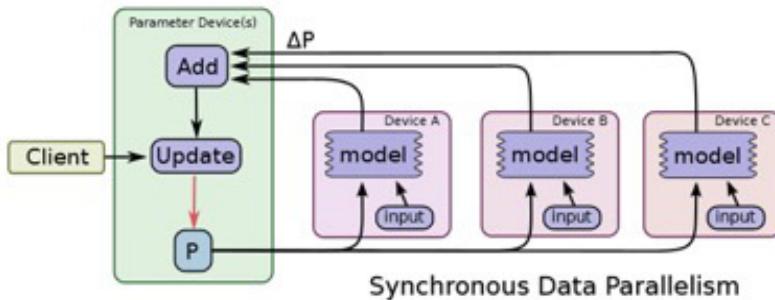
2016 年 4 月 14 日，Google 发布了分布式 TensorFlow，能够支持在几百台机器上并行训练。分布式的 TensorFlow 由高性能的 gRPC 库作为底层技术支持。TensorFlow 集群由一系列的任务组成，这些任务执行 TensorFlow 的图计算。每个任务会关联到 TensorFlow 的一个服务，该服务用于创建 TensorFlow 会话及执行图计算。TensorFlow 集群也可以划分为一个或多个作业，每个作业可以包含一个或多个任务。在一个 TensorFlow 集群中，通常一个任务运行在一个机器上。如果该机器支持多 GPU 设备，可以在该机器上运行多个任务，由应用程序控制任务在哪个 GPU 设备上运行。

常用的深度学习训练模型为数据并行化，即 TensorFlow 任务采用相同的训练模型在不同的小批量数据集上进行训练，然后在参数服务器上更新模型的共享参数。TensorFlow 支持同步训练和异步训练两种模型训练方式。

异步训练即 TensorFlow 上每个节点上的任务为独立训练方式，不需要执行协调操作，如下图所示：



同步训练为 TensorFlow 上每个节点上的任务需要读入共享参数，执行并行化的梯度计算，然后将所有共享参数进行合并，如下图所示：



分布式 TensorFlow 应用开发 API 主要包括：

`tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})`: 创建 TensorFlow 集群描述信息，其中 ps, worker 为作业名称，ps_hosts, worker_hosts 为该作业的任务所在节点的地址信息。示例如下：

```
cluster = tf.train.ClusterSpec({"worker": ["worker0.example.com:2222", "worker1.example.com:2222", "worker2.example.com:2222"], "ps": ["ps0.example.com:2222", "ps1.example.com:2222"]})
```

`tf.train.Server(cluster, job_name, task_index)`: 创建一个 TensorFlow 服务，用于运行相应作业上的计算任务，运行的任务在 task_index 指定的机器上启动。

`tf.device(device_name_or_function)`: 设定在指定的设备上执行 Tensor 运算，示例如下：

```
# 指定在 task0 所在的机器上执行 Tensor 的操作运算
```

```
with tf.device("/job:ps/task:0"):
    weights_1 = tf.Variable(...)
    biases_1 = tf.Variable(...)
```

分布式 TensorFlow MNIST 模型训练

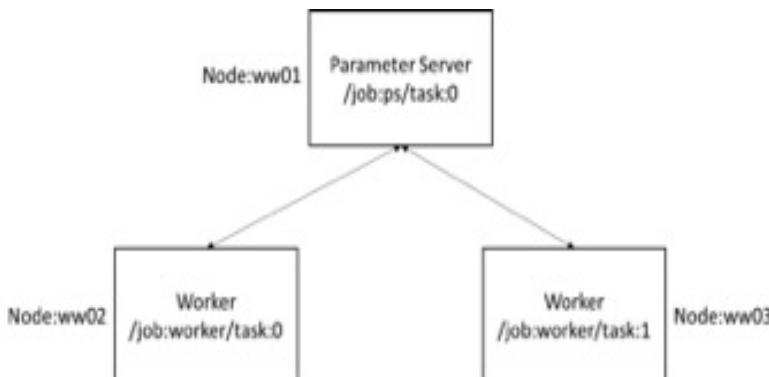
MNIST 是一个手写数字的图片数据库，可从网站 <http://yann.lecun.com/exdb/mnist/> 下载相关数据，其中的每一张图片为 0 到 9 之间的手写数字灰度图片，大小为 28*28 像素，如下图所示：



MNIST 数据集主要包含训练样本 60000 个，测试样本 10000 个。图像数据主要为图片的像素数据，图像数据标签主要表示该图片的类别。由以下四个文件组成：

- train-images-idx3-ubyte.gz （训练图像数据60000个）
- train-labels-idx1-ubyte.gz （训练图像数据标签60000个）
- t10k-images-idx3-ubyte.gz （测试图像数据10000个）
- t10k-labels-idx1-ubyte.gz （测试图像数据标签10000个）

本文采用如下的结构对 MNIST 数据集进行分布式训练，由三个节点组成。ww01 节点为 Parameter Server，ww02 节点为 Worker0，ww03 节点为 Worker1。其中 Parameter Server 执行参数更新任务，Worker0，Worker1 执行图模型训练计算任务，如下图所示。分布式 MNIST 训练模型在执行十万次迭代后，收敛精度达到 97.77%。



在 ww01 节点执行如下命令，启动参数服务 /job:ps/task:0：

```
python asyncmnist.py --ps_hosts=ww01:2222 --worker_
hosts=ww02:2222,ww03:2222 --job_name=ps --task_index=0
```

在 ww02 节点执行如下命令，启动模型运算 /job:worker/task:0：

```
python asyncmnist.py --ps_hosts=ww01:2222 --worker_
hosts=ww02:2222,ww03:2222 --job_name=worker --task_index=0
```

在 ww03 节点执行如下命令，启动模型运算 /job:worker/task:1：

```
python asyncmnist.py --ps_hosts=ww01:2222 --worker_
hosts=ww02:2222,ww03:2222 --job_name=worker --task_index=1
```

分布式 MNIST 的训练模型如下：

```
import math

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

# TensorFlow集群描述信息，ps_hosts表示参数服务节点信息，worker_hosts表示worker节点信息

tf.app.flags.DEFINE_string("ps_hosts", "", "Comma-separated list of hostname:port
pairs")

tf.app.flags.DEFINE_string("worker_hosts", "", "Comma-separated list of
hostname:port pairs")

# TensorFlow Server模型描述信息，包括作业名称，任务编号，隐含层神经元数量，MNIST数据
目录以及每次训练数据大小（默认一个批次为100个图片）

tf.app.flags.DEFINE_string("job_name", "", "One of 'ps', 'worker'")

tf.app.flags.DEFINE_integer("task_index", 0, "Index of task within the job")

tf.app.flags.DEFINE_integer("hidden_units", 100, "Number of units in the hidden
layer of the NN")

tf.app.flags.DEFINE_string("data_dir", "MNIST_data", "Directory for storing mnist
data")

tf.app.flags.DEFINE_integer("batch_size", 100, "Training batch size")

FLAGS = tf.app.flags.FLAGS

#图片像素大小为28*28像素

IMAGE_PIXELS = 28

def main(_):
```

```
#从命令行参数中读取TensorFlow集群描述信息
ps_hosts = FLAGS.ps_hosts.split(",")
worker_hosts = FLAGS.worker_hosts.split(",")

# 创建TensorFlow集群描述对象
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

# 为本地执行Task，创建TensorFlow本地Server对象。
server = tf.train.Server(cluster, job_name=FLAGS.job_name, task_index=FLAGS.
task_index)

#如果是参数服务，直接启动即可
if FLAGS.job_name == "ps":
    server.join()

elif FLAGS.job_name == "worker":
    #分配操作到指定的worker上执行，默认为该节点上的cpu0
    with tf.device(tf.train.replica_device_setter(worker_device="/job:worker/
task:%d" % FLAGS.task_index, cluster=cluster)):

        # 定义TensorFlow隐含层参数变量，为全连接神经网络隐含层
        hid_w = tf.Variable(tf.truncated_normal([IMAGE_PIXELS * IMAGE_PIXELS,
FLAGS.hidden_units], stddev=1.0 / IMAGE_PIXELS), name="hid_w")
        hid_b = tf.Variable(tf.zeros([FLAGS.hidden_units]), name="hid_b")

        # 定义TensorFlow softmax回归层的参数变量
        sm_w = tf.Variable(tf.truncated_normal([FLAGS.hidden_units, 10],
stddev=1.0 / math.sqrt(FLAGS.hidden_units)), name="sm_w")
        sm_b = tf.Variable(tf.zeros([10]), name="sm_b")

        #定义模型输入数据变量（x为图片像素数据，y_为手写数字分类）
        x = tf.placeholder(tf.float32, [None, IMAGE_PIXELS * IMAGE_PIXELS])
        y_ = tf.placeholder(tf.float32, [None, 10])

        #定义隐含层及神经元计算模型
        hid_lin = tf.nn.xw_plus_b(x, hid_w, hid_b)
        hid = tf.nn.relu(hid_lin)

        #定义softmax回归模型，及损失方程
        y = tf.nn.softmax(tf.nn.xw_plus_b(hid, sm_w, sm_b))
        loss = -tf.reduce_sum(y_ * tf.log(tf.clip_by_value(y, 1e-10, 1.0)))

        #定义全局步长，默认值为0
        global_step = tf.Variable(0)
```

```

#定义训练模型，采用Adagrad梯度下降法

train_op = tf.train.AdagradOptimizer(0.01).minimize(loss, global_
step=global_step)

#定义模型精确度验证模型，统计模型精确度

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

#对模型定期做checkpoint，通常用于模型回复

saver = tf.train.Saver()

#定义收集模型统计信息的操作

summary_op = tf.merge_all_summaries()

#定义操作初始化所有模型变量

init_op = tf.initialize_all_variables()

#创建一个监管程序，用于构建模型检查点以及计算模型统计信息。

sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0), logdir="/tmp/
train_logs", init_op=init_op, summary_op=summary_op, saver=saver, global_
step=global_step, save_model_secs=600)

#读入MNIST训练数据集

mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

#创建TensorFlow session对象，用于执行TensorFlow图计算

with sv.managed_session(server.target) as sess:

    step = 0

    while not sv.should_stop() and step < 1000:

        # 读入MNIST的训练数据，默认每批次为100个图片

        batch_xs, batch_ys = mnist.train.next_batch(FLAGS.batch_size)

        train_feed = {x: batch_xs, y_: batch_ys}

        #执行分布式TensorFlow模型训练

        _, step = sess.run([train_op, global_step], feed_dict=train_feed)

        #每隔100步长，验证模型精度

        if step % 100 == 0:

            print "Done step %d" % step

            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.
test.labels}))

    # 停止TensorFlow Session

```

```

    sv.stop()

if __name__ == "__main__":
    tf.app.run()

```

梯度下降法在分布式 TensorFlow 中的性能比较分析

2016 年 谷歌 在 ICLR (the International Conference on Learning Representations) Workshop 上发表了论文 REVISITING DISTRIBUTED SYNCHRONOUS SGD。基于 ImageNet 数据集，该论文对异步随机梯度下降法 (Async-SGD) 和同步随机梯度下降法 (Sync-SGD) 进行了比较分析。

Dean 在 2012 年提出了分布式随机梯度下降法，模型参数可以分布式地存储在不同的服务器上，称之为参数服务器 (Parameter Server, PS)，以及 Worker 节点可以并发地处理训练数据并且能够和参数服务通信获取模型参数。异步随机梯度下降法 (Async-SGD)，主要由以下几个步骤组成：

- 针对当前批次的训练数据，从参数服务器获取模型的最新参数。
- 基于上述获取到的模型参数，计算损失方程的梯度。

将上述计算得到的梯度发送回参数服务器，并相应地更新模型参数。

同步随机梯度下降法 (Sync-SGD) 与 Sync-SGD 的主要差异在于参数服务器将等待所有 Worker 发送相应的梯度值，并聚合这些梯度值，最后把更新后的梯度值发送回节点。

Async-SGD 的主要问题是每个 Worker 节点计算的梯度值发送回参数服务器会有参数更新冲突，一定程度影响算法的收敛速度。Sync-SGD 算法能够保证在数据集上执行的是真正的随机梯度下降法，消除掉了参数的更新冲突。但同步随机梯度下降法同时带来的问题是训练数据的批量数据会比较大，参数服务器上参数的更新时间依赖于最慢的 worker 节点。

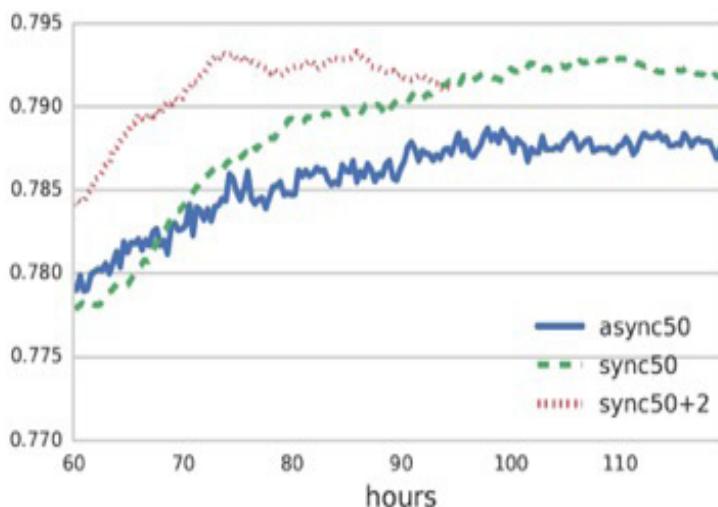
为了解决有些 worker 节点比较慢的问题，我们可以使用多一点的 Worker 节点，这样 Worker 节点数变为 $N+N*5\%$ ， N 为集群 Worker 节点数。Sync-SGD 可以设定为在接受到 N 个 Worker 节点的参数后，可以直接更新参数服务器上的模型参数，进入下一个批次的模型训练。慢节点上训练

出来的参数是会被丢弃掉。我们称这种方法为 Sync-SGD with backups。

2015 年，Abadi 使用 TensorFlow 的 Async-SGD, Sync-SGD, Sync-SGD with backups 训练模型对 ImageNet 的 Benchmark 问题进行了实验分析。要对该训练数据进行 1000 种图片的分类训练，实验环境为 50 到 200 个的 worker 节点，每个 worker 节点上运行 k40 GPU。使用分布式 TensorFlow 后大大缩短了模型训练时间，Async-SGD 算法实验结果如下，其中 200 个节点的训练时间比采用 25 个节点的运算时间缩短了 8 倍，如下图所示。

Number of workers	Test Accuracy (%)	Time (hrs)	speedup relative to 25
25	78.94	184.9	1
50	78.83	97.67	1.89
100	78.44	51.97	3.56
200	78.04	22.94	8.06

下图为 50 个 Worker 节点的 Async-SGD, Sync-SGD, Sync-SGD with backups 模型训练结果的比较。



从结果中可以看出增加 2 backup 节点，Sync-SGD with backups 模型可以快速提升模型训练速度。同时 Sync-SGD 模型比 Async-SGD 模型大概提升了 25% 的训练速度，以及 0.48% 的精确度。随着数据集的增加，分布式训练的架构变得越来越重要。而分布式 TensorFlow 正是解决该问题的利器，有效地提升了大规模模型训练的效率，提供了企业级的深度学习解决方案。

TensorFlow 与卷积神经网络



前言

图像识别技术越来越多地渗透到我们的日常生活中，人可以很快地识别图像类型，比如，很容易地识别一个图片是狮子还是其它动物，可以很容易地对人脸进行识别。但是对于机器来说，去识别一个图片是什么，是一个非常困难的问题。但在过去的几年中，图像识别技术取得了巨大的进展，在一些固定领域可以达到，甚至超越人类的识别精度，该技术称为深度卷积神经网络（Deep Convolutional Neural Network）。

目前，学术界主要通过 ImageNet 的 Benchmark 问题，去验证图像识

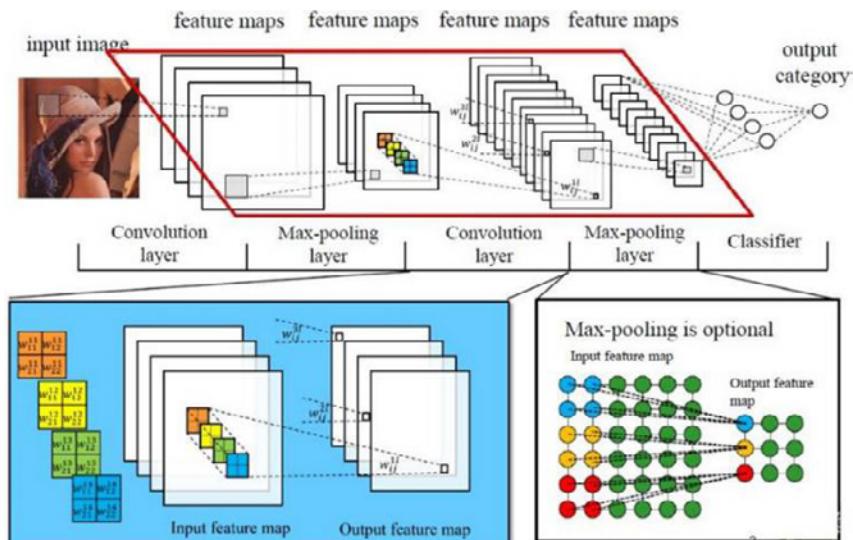
别技术的发展程度，卷积神经网络模型包括：QuocNet, AlexNet, Inception (GoogLeNet), BN-Inception-v2，以及最新的 Inception-v3 模型。其中，AlexNet 的 top-5 的错误率为 15.3%；Inception (GoogLeNet) 降到 6.67%；BN-Inception-v2 降到 4.9%；Inception-v3 降到 3.46%。

如果用户有业务图片数据，如何利用开源现有的模型进行训练呢？如何进行花图片识别，人物图片识别，车辆图片识别，医学图片识别呢？本文主要介绍 TensorFlow 开源模型 Cifar10, Inception V3, Vgg19 的主要架构和代码。如果用户需要对业务图片识别，可再已有模型的基础上持续改进，进行训练及调优，加速研发，满足业务需求。

卷积神经网络回顾

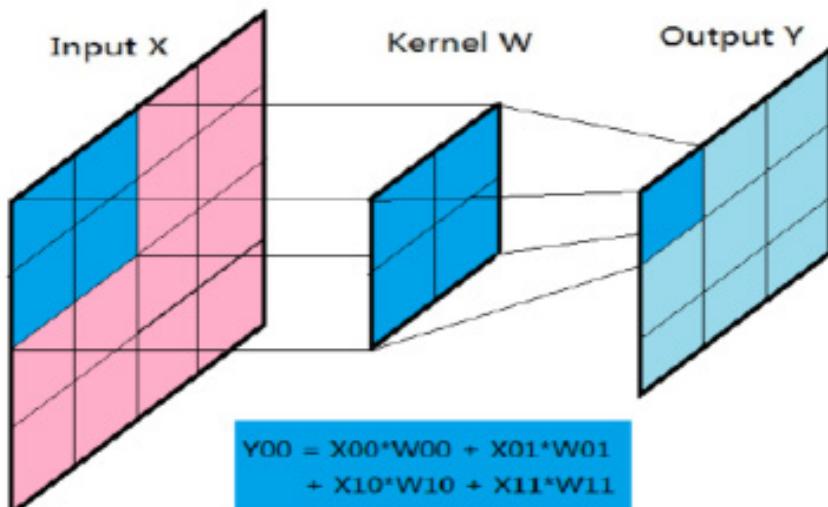
卷积神经网络是基于人工神经网络的深度机器学习方法，成功应用于图像识别领域。CNN 采用了局部连接和权值共享，保持了网络的深层结构，同时又减少了网络参数，使模型具有良好的泛化能力又较容易训练，CNN 的训练算法是梯度下降的错误反向传播（Back Propagate，BP）算法的一种变形。

卷积神经网络通常采用若干个卷积和子采样层的叠加结构作为特征抽取器。卷积层与子采样层不断将特征图缩小，但是特征图的数量往往增多。



特征抽取器后面接一个分类器，分类器通常由一个多层次感知机构成。在特征抽取器的末尾，我们将所有的特征图展开并排列成为一个向量，称为特征向量，该特征向量作为后层分类器的输入，如上图所示。

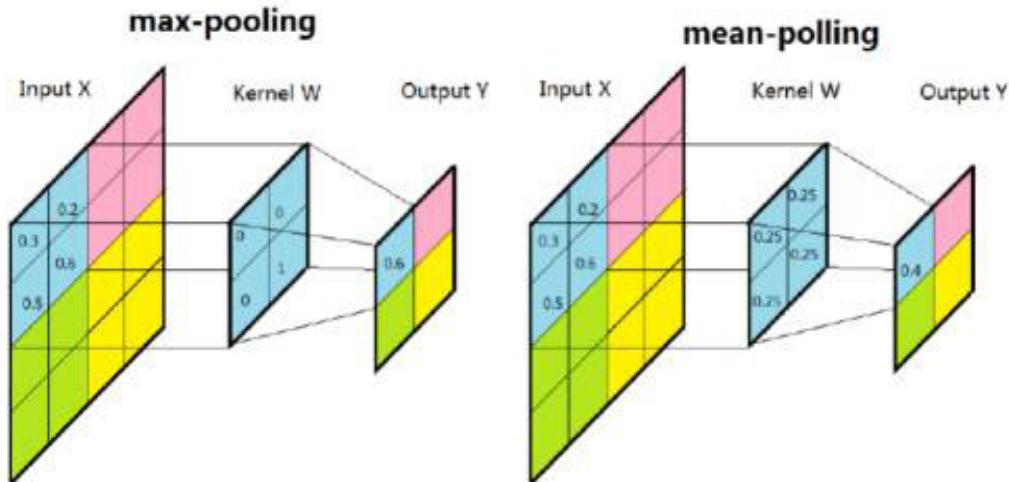
卷积过程有三个二维矩阵参与，它们分别是两个特征图和一个卷积核：原图 inputX、输出图 outputY、卷积核 kernelW。卷积过程可以理解为卷积核 kernelW 覆盖在原图 inputX 的一个局部的面上，kernelW 对应位置的权重乘于 inputX 对应神经元的输出，对各项乘积求和并赋值到 outputY 矩阵的对应位置。卷积核在 inputX 图中从左向右，从上至下每次移动一个位置，完成整张 inputX 的卷积过程，如下图所示。



子采样有两种方式，一种是均值子采样，一种是最大值子采样，如下图所示。

在最大值子采样中的卷积核中，只有一个值为 1，其他值为 0，保留最强输入值，卷积核在原图上的滑动步长为 2，相当于把原图缩减到原来的 1/4。均值子采样卷积核中的每个权重为 0.25，保留的是输入图的均值数据。

卷积核的本质是神经元之间相互连接的权重，而且该权重被属于同一特征图的神经元所共享。在实际的网络训练过程中，输入神经元组成的特征图被切割成卷积核大小的小图。每个小图通过卷积核与后层特征图的一



个神经元连接。一个特征图上的所有小图和后层特征图中某个神经元的连接使用的是相同的卷积核，也就是同特征图的神经元共享了连接权重。

TensorFlow API 构建卷积神经网络

在 TensorFlow 中，卷积神经网络（Convolutional neural networks, CNNs）主要包含三种类型的组件，主要 API 如下：

卷积层（Convolutional Layer），构建一个 2 维卷积层，常用的参数有：

```
conv = tf.layers.conv2d(
    inputs=pool,
    filters=64,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
```

inputs 表示输入要的 Tensor，filters 表示卷积核的数量，kernel_size 表示卷积核的大小，padding 表示卷积的边界处理方式，有 valid 和 same 两种方式，valid 方式不会在原有输入的基础上添加新的像素，same 表示需要对 input 的边界数据进行填存，具体计算公式参见 https://www.tensorflow.org/api_docs/python/tf/nn/convolution。

activation 表示要采用的激活函数。

池化层（Pooling Layer，max_pooling2d 或 average_pooling2d），用

于构建 2 维池化，常用的参数有：

```
tf.layers.max_pooling2d(  
    inputs=conv,  
    pool_size=[2, 2],  
    strides=2)
```

inputs 表示要被池化的输入 Tensor，pool_size 表示池化窗口大小，strides 表示进行池化操作的步长。

全链接层（Dense Layer, dense），主要对特性向量执行分类操作。执行全链接操作前，需要对池化后的特性向量，执行展开操作，转换成 [batch_size, features] 的形式，如下所示：

```
tf.reshape(pool, [-1, 7 * 7 * 64]), -1 表示 BatchSize,
```

全链接层主要参数如下所示：

```
tf.layers.dense(inputs=pool2_flat,  
    units=1024,  
    activation=tf.nn.relu)
```

inputs 表示输入层，units 表示输出层的 tensor 的形状为 [batchsize, units]，activation 表示要采用的激活函数。

使用 TensorFlow API 构建卷积神经网络的示例代码，如下所示：

```
# 输入层  
  
# 改变输入数据维度为 4-D tensor: [batch_size, width, height, channels]  
# 图像数据为 28x28 像素大小，并且为单通道  
  
input_layer = tf.reshape(features, [-1, 28, 28, 1])  
  
# 卷积层1  
  
# 卷积核大小为5x5，卷积核数量为32， 激活方法使用RELU  
  
# 输入Tensor维度: [batch_size, 28, 28, 1]  
# 输出Tensor维度: [batch_size, 28, 28, 32]  
  
conv1 = tf.layers.conv2d(  
    inputs=input_layer,  
    filters=32,  
    kernel_size=[5, 5],  
    padding="same",  
    activation=tf.nn.relu)
```

```

# 池化层1

# 采用2x2维度的最大化池化操作，步长为2

# 输入Tensor维度: [batch_size, 28, 28, 32]

# 输出Tensor维度: [batch_size, 14, 14, 32]

pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

#卷积层2

#卷积核大小为5x5，卷积核数量为64， 激活方法使用RELU.

#输入Tensor维度: [batch_size, 14, 14, 32]

#输出Tensor维度: [batch_size, 14, 14, 64]

conv2 = tf.layers.conv2d(

    inputs=pool1,

    filters=64,

    kernel_size=[5, 5],

    padding="same",

    activation=tf.nn.relu)

#池化层2

#采用2x2维度的最大化池化操作，步长为2

#输入Tensor维度: [batch_size, 14, 14, 64]

#输出Tensor维度: [batch_size, 7, 7, 64]

pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

# 展开并列池化层输出Tensor为一个向量

#输入Tensor维度: [batch_size, 7, 7, 64]

#输出Tensor维度: [batch_size, 7 * 7 * 64]

pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])

# 全链接层

# 该全链接层具有1024神经元

#输入Tensor维度: [batch_size, 7 * 7 * 64]

#输出Tensor维度: [batch_size, 1024]

dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu)

#对全链接层的数据加入dropout操作，防止过拟合

#40%的数据会被dropout，

dropout = tf.layers.dropout(

    inputs=dense, rate=0.4, training=mode == learn.ModeKeys.TRAIN)

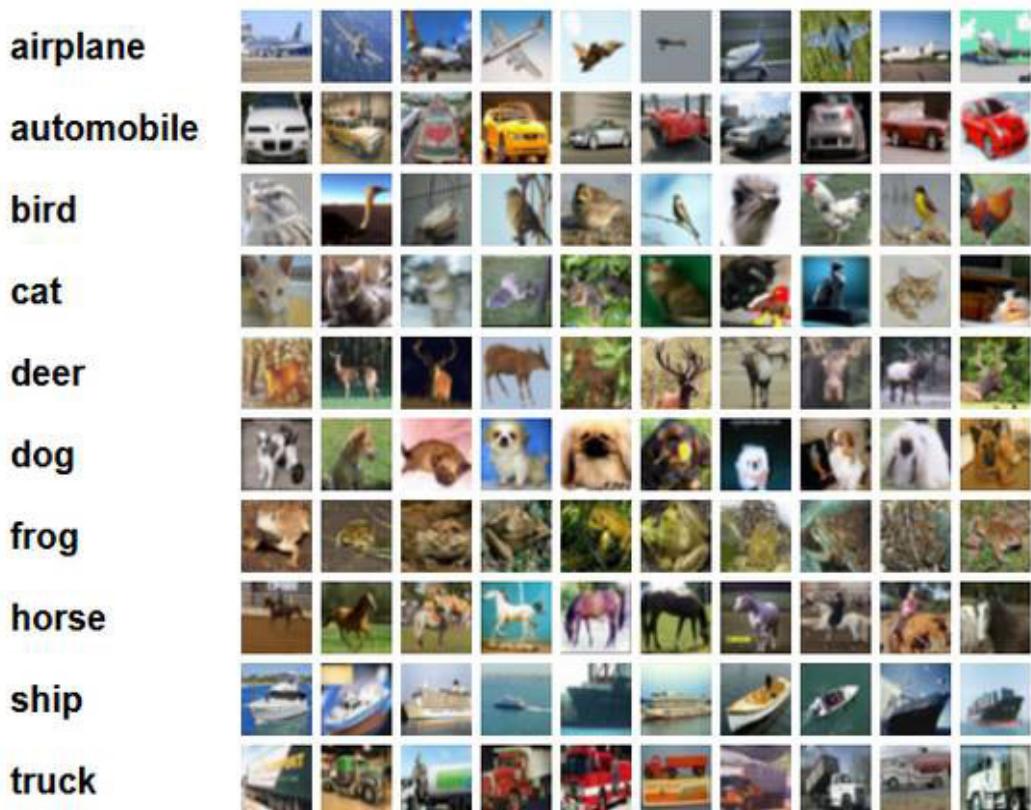
# Logits层，对dropout层的输出Tensor，执行分类操作

```

```
#输入Tensor维度: [batch_size, 1024]
#输出Tensor维度: [batch_size, 10]
logits = tf.layers.dense(inputs=dropout, units=10)
```

TensorFlow Cifar10 模型

CIFAR-10, <http://www.cs.toronto.edu/~kriz/cifar.html>, 是图片识别的 benchmark 问题，主要对 RGB 为 32*32 的图像进行 10 分类，类别包括：airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck。其中包括 50000 张训练图片，10000 张测试图片。



TensorFlow Cifar10, <https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10>, 模型包含 1,068,298 个参数，单个图片的推导包含 19.5M 个乘法 / 加法运算。该模型在 GPU 上运行几个小时后，测试精确度会达到 86%。模型特性主要包括：

- 核心数学组件：卷积操作，RELU激活算子，池化操作，局部响应归一化操作。
- 可视化展示：展示训练过程的loss值，梯度，以及参数分布情况等。
- 滑动平均：使用参数的滑动平均值执行评估操作。
- 预处理队列：通过队列对训练数据进行预处理，用于减少读取数据的延迟，加快数据的预处理。

该模型的代码结构如下：

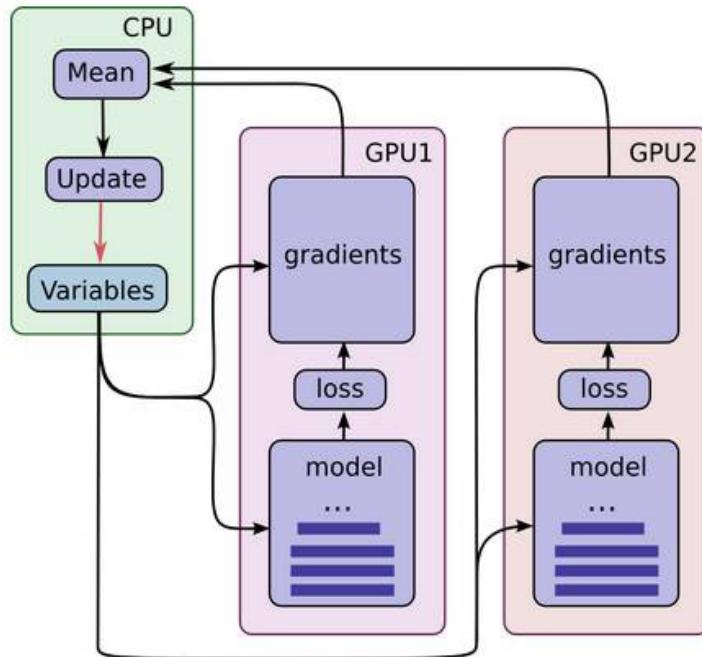
- cifar10_input.py：负责加载训练数据。
- cifar10.py：负责构建cifar10模型。
- cifar10_train.py：负责在单设备（CPU/GPU）上进行训练。
- cifar10_multi_gpu_train.py：负责在多GPU上进行训练。
- cifar10_eval.py：负责对模型进行评估。

该模型的 Graph 结构如下：



下图为 Cifar10 的多 GPU 模型架构，每个 GPU 型号最好相同，具备

足够的内存能运行整个 Cifar10 模型。



该架构会复制 Cifar10 模型到每个 GPU 上，每个 GPU 上训练完一个 Batch 的数据后，在 CPU 端对梯度执行同步操作（求均值），更新训练参数，然后把模型参数发送给每个 GPU，进行下一个 Batch 数据的训练。

TensorFlow Inception V3 模型

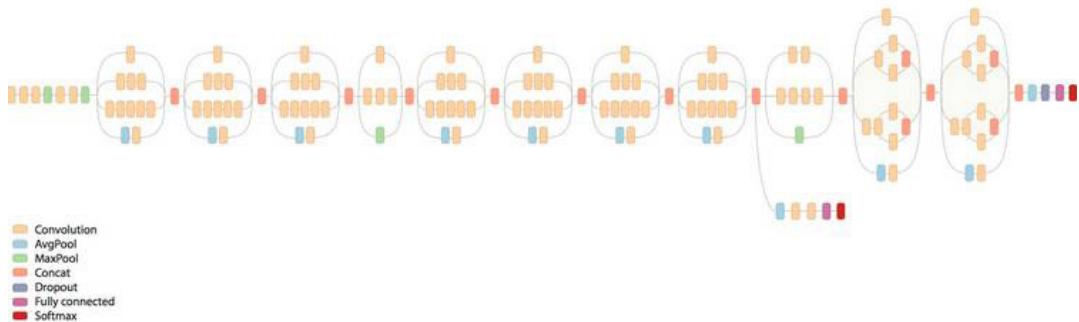
Inception V3, <http://arxiv.org/abs/1512.00567>, 模型包含 25 million 个模型参数，对单个图片的推导包含了 5 billion 的乘法 / 加法运算。top-1 的误差率降到了 21.2%，top-5 的误差率降到了 5.6%。该模型网络结构如下图所示。

由于 ImageNet 的训练数据比较大，下面主要介绍如何使用 Flower 的数据进行训练，该数据集有 5 种类别（daisy, dandelion, roses, sunflowers, tulips）的花，大概有几千张图片。

首先我们需要下载 TensorFlow Inception V3 模型，如下所示：

```
git pull https://github.com/tensorflow/models
```

对花数据进行训练的代码结构，如下所示：



- `data/download_and_preprocess_flowers.sh`: 下载花的数据，并转换为TFRecord格式。
- `slim/inception_model.py`: inception V3模型。
- `flowers_train.py`: 执行单机多GPU模型的训练。
- `flowers_eval.py`: 对训练的准确度进行评估。

```
#进入Inception V3程序目录
cd models/inception

#设定Flower数据的存储路径
FLOWERS_DATA_DIR=/tmp/flowers-data/

#编译程序
bazel build //inception:download_and_preprocess_flowers

#执行下载和转换TFRecord操作
bazel-bin/inception/download_and_preprocess_flowers "${FLOWERS_DATA_DIR}"
```

转换好的训练数据包括: `train-00000-of-00002`, `train-00001-of-00002`

转换好的验证数据包括: `validation-00000-of-00002`, `validation-00001-of-00002`

基于训练好的Inception V3模型，继续训练花的数据:

```
#设定Inception V3模型下载路径
INCEPTION_MODEL_DIR=$HOME/inception-v3-model

mkdir -p ${INCEPTION_MODEL_DIR}
cd ${INCEPTION_MODEL_DIR}

#下载训练好的Inception模型
curl -O http://download.tensorflow.org/models/image/imagenet/
inception-v3-2016-03-01.tar.gz

tar xzf inception-v3-2016-03-01.tar.gz

#编译程序
```

```

bazel build //inception:flowers_train

#设定要加载的模型路径

MODEL_PATH="${INCEPTION_MODEL_DIR}/inception-v3/model.ckpt-157585"

#设定训练数据路径

FLOWERS_DATA_DIR=/tmp/flowers-data/

#执行模型训练

bazel-bin/inception/flowers_train \
    --train_dir="${TRAIN_DIR}" \
    --data_dir="${FLOWERS_DATA_DIR}" \
    --pretrained_model_checkpoint_path="${MODEL_PATH}" \
    --fine_tune \
    --initial_learning_rate=0.001 \
    --input_queue_memory_factor=1

```

采用单 GPU，训练 1000 次迭代后，模型 loss 值降到 1.04，如下所示：

```

2017-06-11 10:27:43.211374: step 960, loss = 1.05 (20.7 examples/sec; 1.549 sec/batch)
2017-06-11 10:27:58.776483: step 970, loss = 1.11 (20.2 examples/sec; 1.583 sec/batch)
2017-06-11 10:28:14.302368: step 980, loss = 1.12 (20.6 examples/sec; 1.550 sec/batch)
2017-06-11 10:28:29.993650: step 990, loss = 0.99 (20.2 examples/sec; 1.585 sec/batch)
2017-06-11 10:28:45.559217: step 1000, loss = 1.04 (20.7 examples/sec; 1.547 sec/batch)

```

TensorFlow Vgg19 模型

VGG 网络与 AlexNet 类似，也是一种 CNN，VGG 在 2014 年的 ILSVRC localization and classification 两个问题上分别取得了第一名和第二名。VGG 网络非常深，通常有 16 - 19 层，卷积核大小为 3×3 ，16 和 19 层的区别主要在于后面三个卷积部分卷积层的数量。可以看到 VGG 的前几层为卷积和 maxpool 的交替，后面紧跟三个全连接层，激活函数采用 Relu，训练采用了 dropout。VGG 中各模型配置如下，其中 VGG19 的 top-1 的训练精度可达到 71.1%，top-5 的训练精度可达到 89.8%。模型结构示例如下。

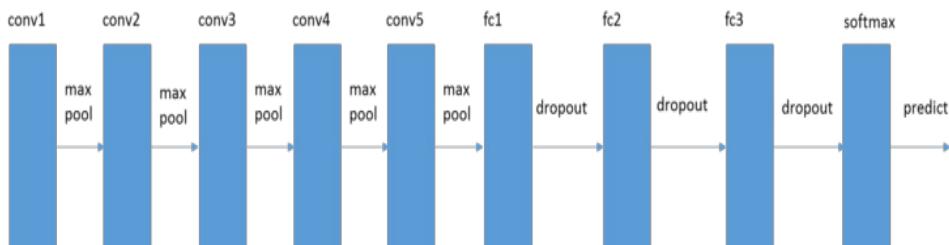
TensorFlow Vgg19 的模型示例如下，<https://github.com/tensorflow/models/blob/master/slim/nets/vgg.py>：

```

#卷积操作和池化操作

net = slim.repeat(inputs, 2, slim.conv2d, 64, [3, 3], scope='conv1')

```



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

```
net = slim.max_pool2d(net, [2, 2], scope='pool1')
net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], scope='conv2')
net = slim.max_pool2d(net, [2, 2], scope='pool2')
net = slim.repeat(net, 4, slim.conv2d, 256, [3, 3], scope='conv3')
net = slim.max_pool2d(net, [2, 2], scope='pool3')
net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv4')
net = slim.max_pool2d(net, [2, 2], scope='pool4')
net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv5')
net = slim.max_pool2d(net, [2, 2], scope='pool5')
net = slim.conv2d(net, 4096, [7, 7], padding=fc_conv_padding, scope='fc6')
#dropout操作，防止过拟合
net = slim.dropout(net, dropout_keep_prob, is_training=is_training,
                   scope='dropout6')
net = slim.conv2d(net, 4096, [1, 1], scope='fc7')
net = slim.dropout(net, dropout_keep_prob, is_training=is_training,
                   scope='dropout7')
net = slim.conv2d(net, num_classes, [1, 1],
                  activation_fn=None,
                  normalizer_fn=None,
                  scope='fc8')
```

总结

本文首先回顾了深度卷积神经网络的特征图、卷积核，池化操作，全链接层等基本概念。接着介绍了使用 TensorFlow API 构建卷积神经网络，主要包括卷积操作 API，池化操作 API 以及全链接操作 API。针对图片识别，讲解了 TensorFlow Benchmark 模型（Cifar10，Inception V3 及 Vgg19）的架构和代码。如果有用户需要对自己的业务图片进行识别，可再已有模型的基础上持续改进，进行训练及调优，加速研发。

TensorFlow 与自然语言处理模型



前言

自然语言处理（简称 NLP），是研究计算机处理人类语言的一门技术，NLP 技术让计算机可以基于一组技术和理论，分析、理解人类的沟通内容。传统的自然语言处理方法涉及到了很多语言学本身的知识，而深度学习，是表征学习（representation learning）的一种方法，在机器翻译、自动问答、文本分类、情感分析、信息抽取、序列标注、语法解析等领域都有广泛的应用。

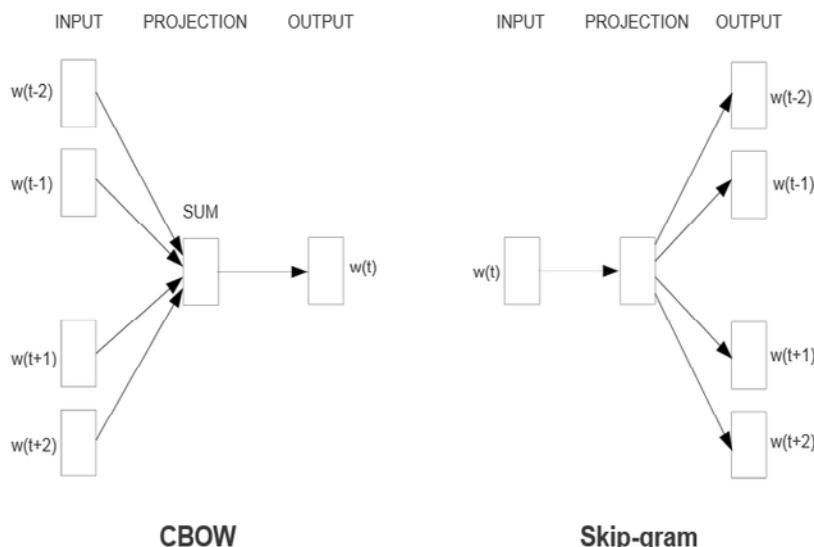
2013 年末谷歌发布的 word2vec 工具，将一个词表示为词向量，将文

字数字化，有效地应用于文本分析。2016 年谷歌开源自动生成文本摘要模型及相关 TensorFlow 代码。2016/2017 年，谷歌发布 / 升级语言处理框架 SyntaxNet，识别率提高 25%，为 40 种语言带来文本分割和词态分析功能。2017 年谷歌官方开源 tf-seq2seq，一种通用编码器 / 解码器框架，实现自动翻译。本文主要结合 TensorFlow 平台，讲解 TensorFlow 词向量生成模型（Vector Representations of Words）；使用 RNN、LSTM 模型进行语言预测；以及 TensorFlow 自动翻译模型。

Word2Vec 数学原理简介

我们将自然语言交给机器学习来处理，但机器无法直接理解人类语言。那么首先要做的是要将语言数学化，Hinton 于 1986 年提出 Distributed Representation 方法，通过训练将语言中的每一个词映射成一个固定长度的向量。所有这些向量构成词向量空间，每个向量可视为空间中的一个点，这样就可以根据词之间的距离来判断它们之间的相似性，并且可以把其应用扩展到句子、文档及中文分词。

Word2Vec 中用到两个模型，CBOW 模型 (Continuous Bag-of-Words model) 和 Skip-gram 模型 (Continuous Skip-gram Model)。模型示例如下，是三层结构的神经网络模型，包括输入层，投影层和输出层。



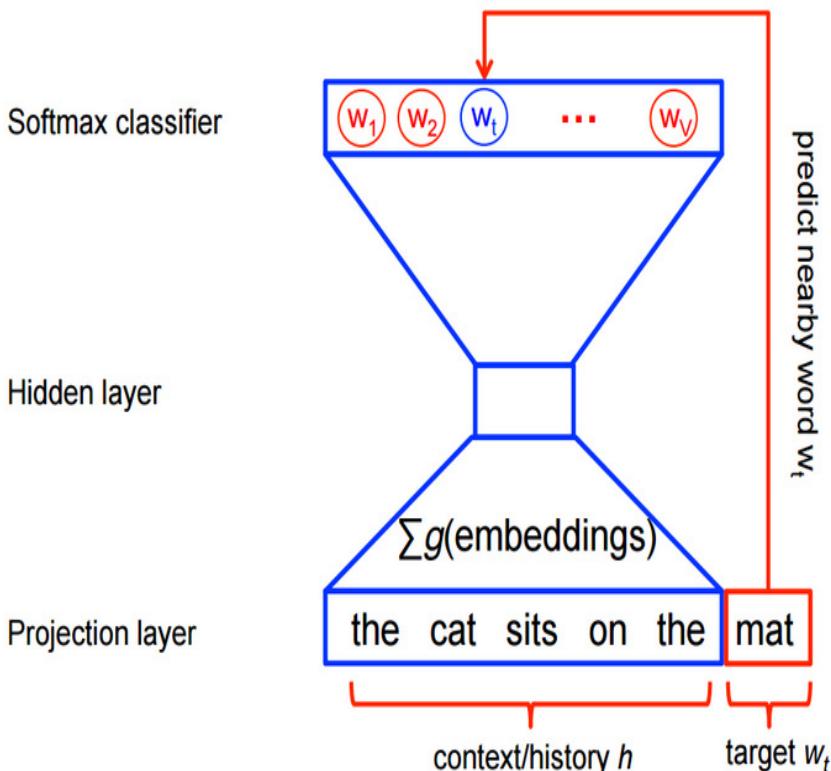
$$\begin{aligned}
 P(w_t|h) &= \text{softmax(score}(w_t, h)) \\
 &= \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\}}
 \end{aligned}$$

其中 $\text{score}(w_t, h)$, 表示在的上下文环境下, 预测结果是的概率得分。

上述目标函数, 可以转换为极大化似然函数, 如下所示:

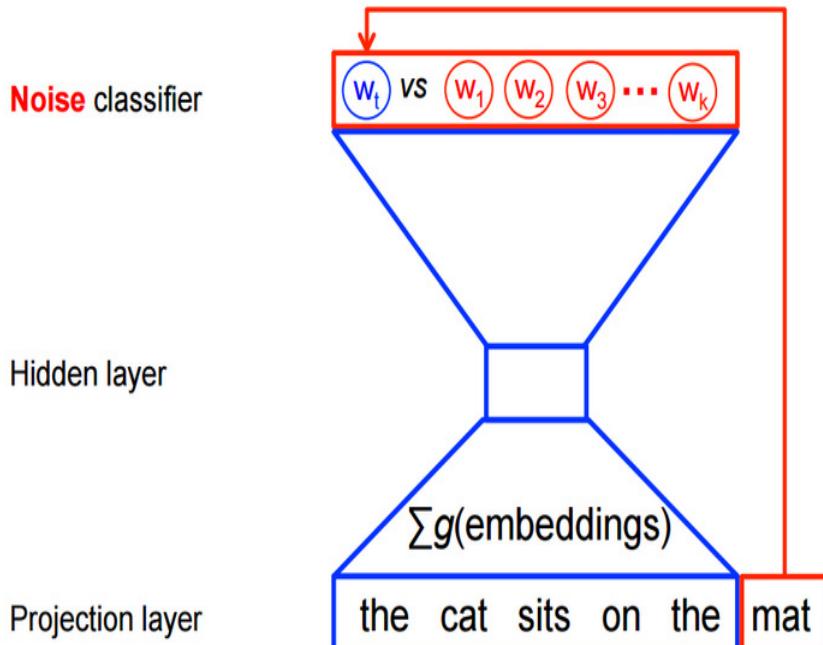
$$\begin{aligned}
 J_{\text{ML}} &= \log P(w_t|h) \\
 &= \text{score}(w_t, h) - \log \left(\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\} \right)
 \end{aligned}$$

求解上述概率模型的计算成本是非常高昂的, 需要在神经网络的每一次训练过程中, 计算每个词在他的上下文环境中出现的概率得分, 如下所示:



然而在使用 word2vec 方法进行特性学习的时候, 并不需要计算全概

率模型。在 CBOW 模型和 skip-gram 模型中，使用了逻辑回归（logistic regression）二分类方法进行的预测。如下图 CBOW 模型所示，为了提高模型的训练速度和改善词向量的质量，通常采用随机负采样（Negative Sampling）的方法，噪音样本 $w_1, w_2, w_3, \dots, w_k$ 为选中的负采样。



TensorFlow 近义词模型

本章讲解使用 TensorFlow word2vec 模型寻找近义词，输入数据是一大段英文文章，输出是相应词的近义词。比如，通过学习文章可以得到和 five 意思相近的词有：four, three, seven, eight, six, two, zero, nine。通过对大段英文文章的训练，当神经网络训练到 10 万次迭代，网络 Loss 值减小到 4.6 左右的时候，学习得到的相关近似词，如下图所示：

```

Nearest to of: in, and, michelob, including, ursus, nn, operatorname, dasyprocta,
Nearest to two: four, three, five, six, one, seven, eight, callithrix,
Nearest to new: worker, andamanese, brew, it, convenience, microsite, rss, dasyprocta,
Nearest to b: d, agouti, transmissions, yankees, UNK, dasyprocta, w, andamanese,
Nearest to or: and, than, dasyprocta, operatorname, ursus, callithrix, mishnayot, senex,
Nearest to who: he, she, they, also, and, never, southwards, matthers,
Nearest to five: four, three, seven, eight, six, two, zero, nine,
Nearest to he: it, she, they, who, there, we, but, naples,
Nearest to s: ursus, dasyprocta, agave, his, kapoor, agouti, zero, five,
Nearest to time: year, abet, pettigrew, thaler, offenses, branden, pontificia, presence,
Nearest to not: generally, still, it, you, they, never, now, to,
Nearest to into: from, through, lexical, under, with, batll, taiwanese, and,
Nearest to zero: eight, seven, five, four, nine, six, agouti, three,
Nearest to th: six, seven, nine, aberdeenshire, albury, eight, five, cebus,
Nearest to often: sometimes, usually, also, always, commonly, now, widely, never,
Nearest to with: in, between, operatorname, through, without, microcebus, or, for,

```

下面为 TensorFlow word2vec API 使用说明。

构建词向量变量，`vocabulary_size` 为字典大小，`embedding_size` 为词向量大小

```
embeddings = tf.Variable(tf.random_uniform([vocabulary_size, embedding_size],
-1.0, 1.0))
```

定义负采样中逻辑回归的权重和偏置

```
nce_weights = tf.Variable(tf.truncated_normal
([vocabulary_size, embedding_size], stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
```

定义训练数据的接入

```
train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
```

定义根据训练数据输入，并寻找对应的词向量

```
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
```

基于负采样方法计算 Loss 值

```
loss = tf.reduce_mean( tf.nn.nce_loss
(weights=nce_weights, biases=nce_biases, labels=train_labels,
inputs=embed, num_sampled=num_sampled, num_classes=vocabulary_size))
```

定义使用随机梯度下降法执行优化操作，最小化 loss 值

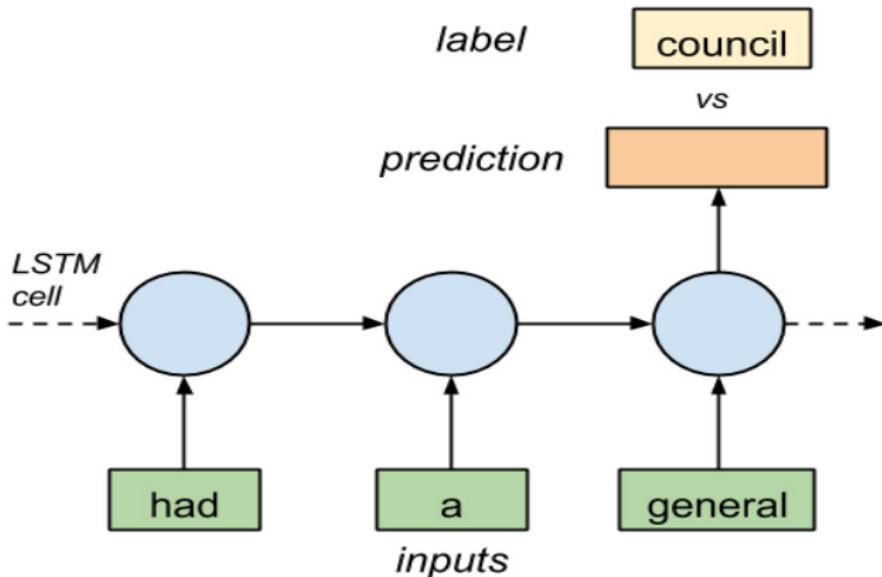
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1.0).minimize(loss)
```

通过 TensorFlow Session Run 的方法执行模型训练

```
for inputs, labels in generate_batch(...):
    feed_dict = {train_inputs: inputs, train_labels: labels}
    _, cur_loss = session.run([optimizer, loss], feed_dict=feed_dict)
```

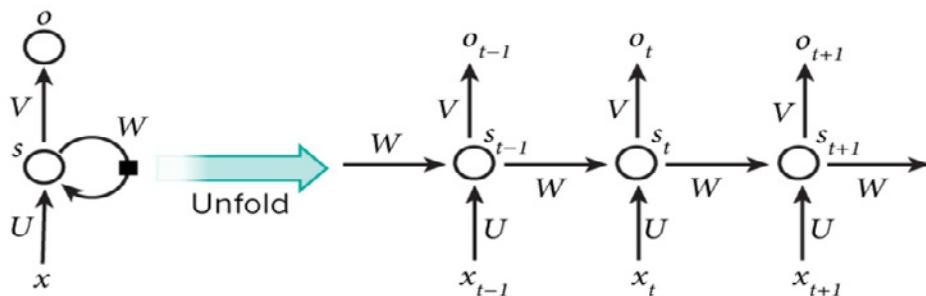
TensorFlow 语言预测模型

本章主要回顾 RNN、LSTM 技术原理，并基于 RNN/LSTM 技术训练语言模型。也就是给定一个单词序列，预测最有可能出现的下一个单词。例如，给定 [had, a, general] 3 个单词的 LSTM 输入序列，预测下一个单词是什么？如下图所示：



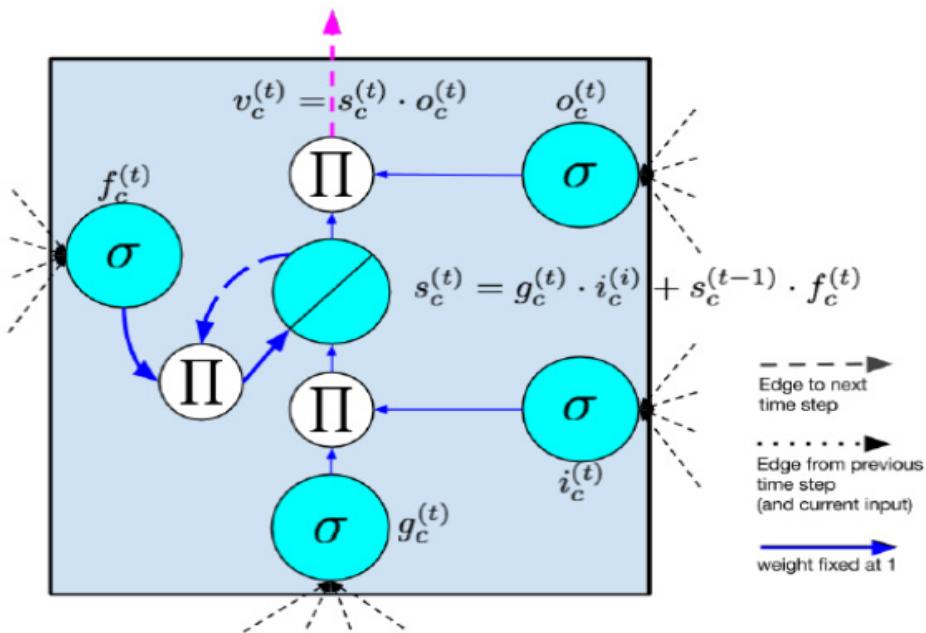
RNN技术原理

循环神经网络（Recurrent Neural Network, RNN）是一类用于处理序列数据的神经网络。和卷积神经网络的区别在于，卷积网络是适用于处理网格化数据（如图像数据）的神经网络，而循环神经网络是适用于处理序列化数据的神经网络。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNN之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。如下图所示：



LSTM技术原理

RNN 有一问题，反向传播时，梯度也会呈指数倍数的衰减，导致经过许多阶段传播后的梯度倾向于消失，不能处理长期依赖的问题。虽然 RNN 理论上可以处理任意长度的序列，但实习应用中，RNN 很难处理长度超过 10 的序列。为了解决 RNN 梯度消失的问题，提出了 Long Short-Term Memory 模块，通过门的开关实现序列上的记忆功能，当误差从输出层反向传播回来时，可以使用模块的记忆元记下来。所以 LSTM 可以记住比较长时间内的信息。常见的 LSTM 模块如下图所示：



$$s^t = g^t \odot i^t + f^t \odot s^{t-1}$$

output gate 类似于 input gate 同样会产生一个 0-1 向量来控制 Memory Cell 到输出层的输出，如下公式所示：

$$v^t = s^t \odot o^t$$

三个门协作使得 LSTM 存储块可以存取长期信息，比如说只要输入门

保持关闭，记忆单元的信息就不会被后面时刻的输入所覆盖。

使用 TensorFlow 构建单词预测模型

首先下载 PTB 的模型数据，该数据集大概包含 10000 个不同的单词，并对不常用的单词进行了标注。

首先需要对样本数据集进行预处理，把每个单词用整数标注，即构建词典索引，如下所示：

读取训练数据：

```
data = _read_words(filename)

#按照单词出现频率，进行排序

counter = collections.Counter(data)

count_pairs = sorted(counter.items(), key=lambda x: (-x[1], x[0]))

#构建词典及词典索引

words, _ = list(zip(*count_pairs))

word_to_id = dict(zip(words, range(len(words))))
```

接着读取训练数据文本，把单词序列转换为单词索引序列，生成训练数据，如下所示：

读取训练数据单词，并转换为单词索引序列：

```
data = _read_words(filename) data = [word_to_id[word] for word in data if word in
word_to_id]
```

生成训练数据的 data 和 label，其中 epoch_size 为该 epoch 的训练迭代次数，num_steps 为 LSTM 的序列长度：

```
i = tf.train.range_input_producer(epoch_size, shuffle=False).dequeue()

x = tf.strided_slice(data, [0, i * num_steps], [batch_size, (i + 1) * num_
steps])

x.set_shape([batch_size, num_steps])

y = tf.strided_slice(data, [0, i * num_steps + 1], [batch_size, (i + 1) * num_
steps + 1])

y.set_shape([batch_size, num_steps])
```

构建 LSTM Cell，其中 size 为隐藏神经元的数量：

```
lstm_cell = tf.contrib.rnn.BasicLSTMCell(size,
```

```
forget_bias=0.0, state_is_tuple=True)
```

如果为训练模式，为保证训练鲁棒性，定义 dropout 操作：

```
attn_cell = tf.contrib.rnn.DropoutWrapper(lstm_cell,
output_keep_prob=config.keep_prob)
```

根据层数配置，定义多层 RNN 神经网络：

```
cell = tf.contrib.rnn.MultiRNNCell( [ attn_cell for _ in range(config.num_layers)],
state_is_tuple=True)
```

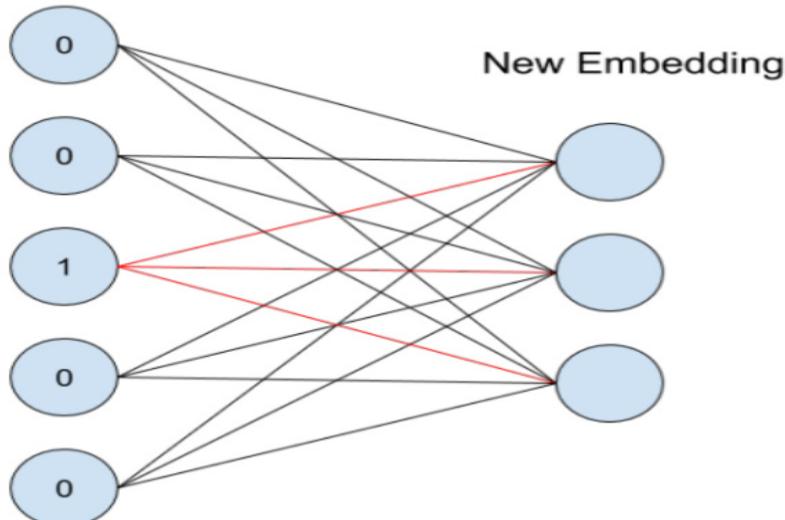
根据词典大小，定义词向量：

```
embedding = tf.get_variable("embedding",
[vocab_size, size], dtype=data_type())
```

根据单词索引，查找词向量，如下图所示。从单词索引找到对应的 One-hot encoding，然后红色的 weight 就直接对应了输出节点的值，也就是对应的 embedding 向量。

```
inputs = tf.nn.embedding_lookup(embedding, input_.input_data)
```

One-hot Embedding



定义 RNN 网络，其中 state 为 LSTM Cell 的状态，cell_output 为 LSTM Cell 的输出：

```
for time_step in range(num_steps):
if time_step > 0: tf.get_variable_scope().reuse_variables()
```

```
(cell_output, state) = cell(inputs[:, time_step, :], state)
outputs.append(cell_output)
```

定义训练的 loss 值就，如下公式所示。

$$\text{loss} = -\frac{1}{N} \sum_{i=1}^N \ln p_{\text{target}_i}$$

```
softmax_w = tf.get_variable("softmax_w", [size, vocab_size], dtype=data_type())
softmax_b = tf.get_variable("softmax_b", [vocab_size], dtype=data_type())
logits = tf.matmul(output, softmax_w) + softmax_b
```

Loss 值：

```
loss = tf.contrib.legacy_seq2seq.sequence_loss_by_example([logits],
[tf.reshape(input_.targets, [-1])], [tf.ones([batch_size * num_steps],
dtype=data_type())])
```

定义梯度及优化操作：

```
cost = tf.reduce_sum(loss) / batch_size
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(tf.gradients(cost, tvars), config.max_grad_norm)
optimizer = tf.train.GradientDescentOptimizer(self._lr)
```

单词困惑度 eloss：

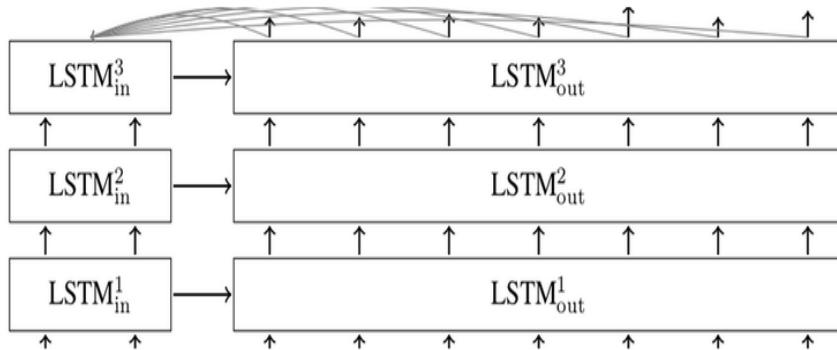
```
perplexity = np.exp(costs / iters)
```

TensorFlow 语言翻译模型

本节主要讲解使用 TensorFlow 实现 RNN、LSTM 的语言翻译模型。基础的 sequence-to-sequence 模型主要包含两个 RNN 网络，一个 RNN 网络用于编码 Sequence 的输入，另一个 RNN 网络用于产生 Sequence 的输出。基础架构如下图所示

上图中的每个方框表示 RNN 中的一个 Cell。在上图的模型中，每个输入会被编码成固定长度的状态向量，然后传递给解码器。2014 年，Bahdanau 在论文“Neural Machine Translation by Jointly Learning to Align

and Translate”中引入了 Attention 机制。Attention 机制允许解码器在每一步输出时参与到原文的不同部分，让模型根据输入的句子以及已经产生的内容来影响翻译结果。一个加入 attention 机制的多层 LSTM sequence-to-sequence 网络结构如下图所示：



针对上述 sequence-to-sequence 模型，TensorFlow 封装成了可以直接调用的函数 API，只需要几百行的代码就能实现一个初级的翻译模型。`tf.nn.seq2seq` 文件共实现了 5 个 seq2seq 函数：

1. `basic_rnn_seq2seq`: 输入和输出都是embedding的形式；encoder和decoder用相同的RNN cell，但不共享权值参数；
2. `tied_rnn_seq2seq`: 同`basic_rnn_seq2seq`，但encoder和decoder共享权值参数；
3. `embedding_rnn_seq2seq`: 同`basic_rnn_seq2seq`，但输入和输出改为id的形式，函数会在内部创建分别用于encoder和decoder的embedding矩阵；
4. `embedding_tied_rnn_seq2seq`: 同`tied_rnn_seq2seq`，但输入和输出改为id形式，函数会在内部创建分别用于encoder和decoder的embedding矩阵；
5. `embedding_attention_seq2seq`: 同`embedding_rnn_seq2seq`，但多了attention机制。

`embedding_rnn_seq2seq` 函数接口使用说明如下：

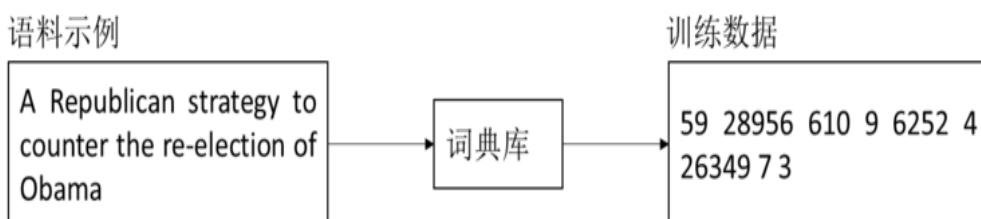
- `encoder_inputs`: encoder的输入；

- decoder_inputs: decoder的输入；
- cell: RNN_Cell的实例；
- num_encoder_symbols, num_decoder_symbols: 分别是编码和解码的大小；
- embedding_size: 词向量的维度；
- output_projection: decoder的output向量投影到词表空间时，用到的投影矩阵和偏置项；
- feed_previous: 若为True, 只有第一个decoder的输入符号有用，所有的decoder输入都依赖于上一步的输出。

```
outputs, states = embedding_rnn_seq2seq(
    encoder_inputs, decoder_inputs, cell,
    num_encoder_symbols, num_decoder_symbols,
    embedding_size, output_projection=None,
    feed_previous=False)
```

TensorFlow 官方提供了英语到法语的翻译示例，采用的是 statmt 网站提供的语料数据，主要包含: giga-fren.release2.fixed.en (英文语料, 3.6G) 和 giga-fren.release2.fixed.fr (法文语料, 4.3G)。该示例的代码结构如下所示:

- seq2seq_model.py: seq2seq的TensorFlow模型；
- 采用了embedding_attention_seq2seq用于创建seq2seq模型。
- data_utils.py: 对语料数据进行数据预处理，根据语料数据生成词典库；并基于词典库把要翻译的语句转换成用用词ID表示的训练序列。如下图所示：



- translate.py: 主函数入口，执行翻译模型的训练
执行模型训练：

```
python translate.py  
--data_dir [your_data_directory] --train_dir [checkpoints_directory]  
--en_vocab_size=40000 --fr_vocab_size=40000
```

总结

随着 TensorFlow 新版本的不断发布以及新模型的不断增加，TensorFlow 已成为主流的深度学习平台。本文主要介绍了 TensorFlow 在自然语言处理领域的相关模型和应用。首先介绍了 Word2Vec 数学原理，以及如何使用 TensorFlow 学习词向量；接着回顾了 RNN、LSTM 的技术原理，讲解了 TensorFlow 的语言预测模型；最后实例分析了 TensorFlow sequence-to-sequence 的机器翻译 API 及官方示例。

TensorFlow 在智能终端中的应用



前言

深度学习在图像处理、语音识别、自然语言处理领域的应用取得了巨大成功，但是它通常在功能强大的服务器端进行运算。如果智能手机通过网络远程连接服务器，也可以利用深度学习技术，但这样可能会很慢，而且只有在设备处于良好的网络连接环境下才行，这就需要把深度学习模型迁移到智能终端。

由于智能终端 CPU 和内存资源有限，为了提高运算性能和内存利用率，需要对服务器端的模型进行量化处理并支持低精度算法。TensorFlow

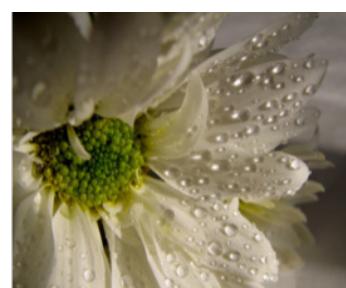
版本增加了对 Android、iOS 和 Raspberry Pi 硬件平台的支持，允许它在这些设备上执行图像分类等操作。这样就可以创建在智能手机上工作并且不需要云端每时每刻都支持的机器学习模型，带来了新的 APP。

本文主要基于看花识名 APP 应用，讲解 TensorFlow 模型如何应用于 Android 系统；在服务器端训练 TensorFlow 模型，并把模型文件迁移到智能终端；TensorFlow Android 开发环境构建以及应用开发 API。

看花识名 APP

使用 AlexNet 模型、Flowers 数据以及 Android 平台构建了“看花识名”APP。TensorFlow 模型对五种类型的花数据进行训练。如下图所示：

Daisy：雏菊



Dandelion：蒲公英



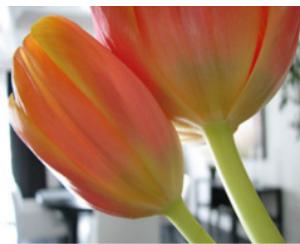
Roses：玫瑰



Sunflowers: 向日葵



Tulips: 郁金香



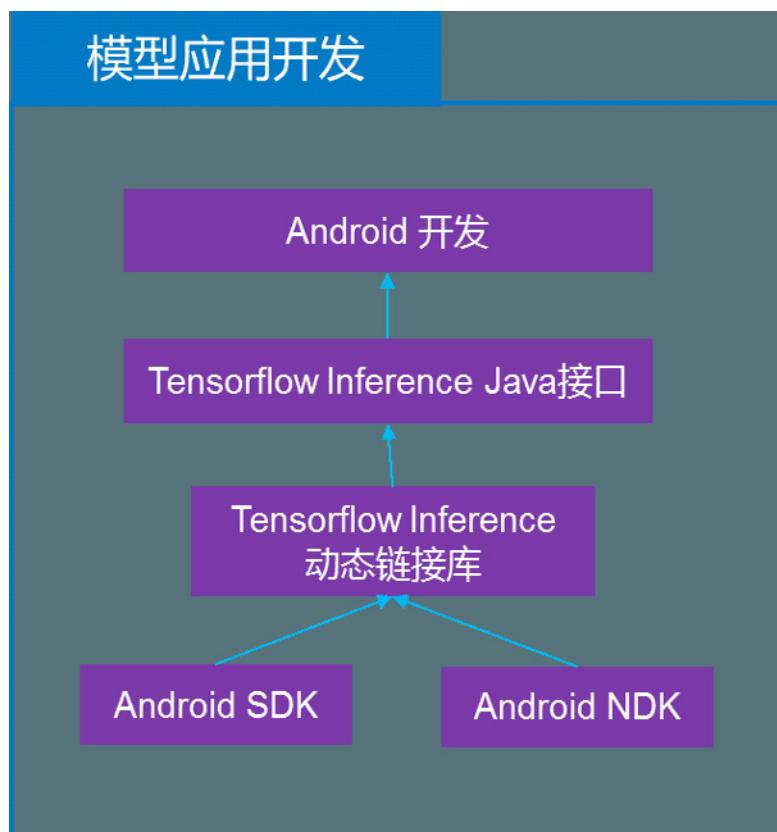
在服务器上把模型训练好后，把模型文件迁移到 Android 平台，在手机上安装 APP。使用效果如下图所示，界面上端显示的是模型识别的置信度，界面中间是要识别的花：



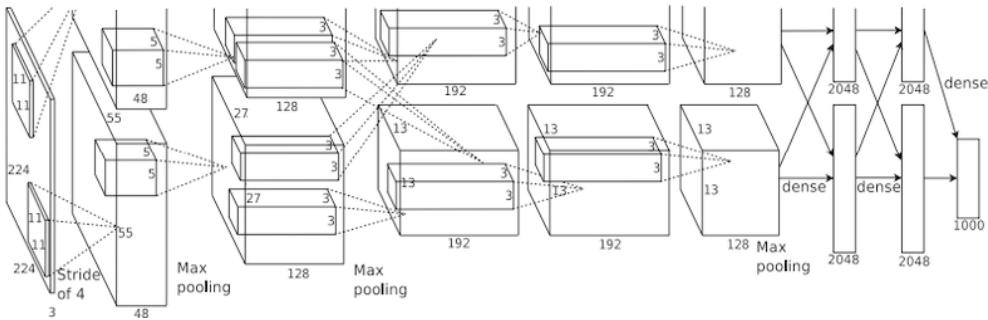
TensorFlow 模型如何应用于看花识名 APP 中，主要包括以下几个关键步骤：模型选择和应用、模型文件转换以及 Android 开发。如下图所示。

模型训练及模型文件

本章采用 AlexNet 模型对 Flowers 数据进行训练。AlexNet 在 2012 取



得了 ImageNet 最好成绩，top 5 准确率达到 80.2%。这对于传统的机器学习分类算法而言，已经相当出色。模型结构如下：



本文采用 TensorFlow 官方 Slim (<https://github.com/tensorflow/models/tree/master.slim>) AlexNet 模型进行训练。

首先下载 Flowers 数据，并转换为 TFRecord 格式：

```
DATA_DIR=/tmp/data/flowers
python download_and_convert_data.py --dataset_name=flowers
--dataset_dir="${DATA_DIR}"
```

执行模型训练，经过 36618 次迭代后，模型精度达到 85%

```
TRAIN_DIR=/tmp/data/train
python train_image_classifier.py --train_dir=${TRAIN_DIR}
--dataset_dir=${DATASET_DIR} --dataset_name=flowers
--dataset_split_name=train --model_name=alexnet_v2
--preprocessing_name=vgg
```

生成 Inference Graph 的 PB 文件

```
python export_inference_graph.py --alsologtostderr
--model_name=alexnet_v2 --dataset_name=flowers --dataset_dir=${DATASET_DIR}
--output_file=alexnet_v2_inf_graph.pb
```

结合 CheckPoint 文件和 Inference GraphPB 文件，生成 Freeze Graph 的 PB 文件

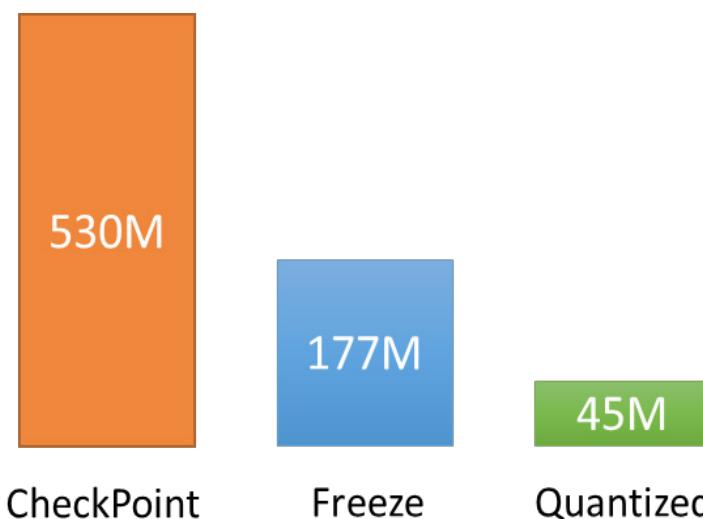
```
python freeze_graph.py --input_graph=alexnet_v2_inf_graph.pb
--input_checkpoint= ${TRAIN_DIR}/model.ckpt-36618 --input_binary=true
--output_graph=frozen_alexnet_v2.pb --output_node_names=alexnet_v2/fc8/squeezed
```

对 Freeze Graph 的 PB 文件进行数据量化处理，减少模型文件的大小，生成的 quantized_alexnet_v2_graph.pb 为智能终端中应用的模型文件

```
bazel-bin/tensorflow/tools/graph_transforms/transform_graph
```

```
--in_graph=frozen_alexnet_v2.pb --outputs="alexnet_v2/fc8/squeezed"
--out_graph=quantized_alexnet_v2_graph.pb --transforms='add_default_attributes
strip_unused_nodes(type=float, shape="1,224,224,3") remove_nodes(op=Identity,
op=CheckNumerics) fold_constants(ignore_errors=true) fold_batch_norms
fold_old_batch_norms quantize_weights quantize_nodes
strip_unused_nodes sort_by_execution_order'
```

为了减少智能终端上模型文件的大小，TensorFlow 中常用的方法是对模型文件进行量化处理，本文对 AlexNet CheckPoint 文件进行 Freeze 和 Quantized 处理后的文件大小变化如下图所示：

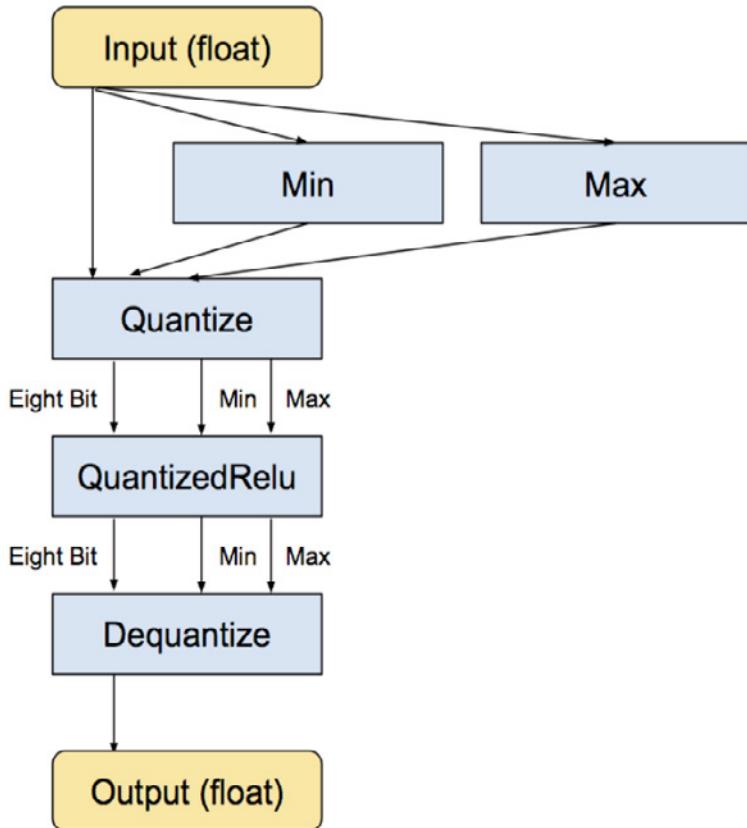


量化操作的主要思想是在模型的 Inference 阶段采用等价的 8 位整数操作代替 32 位的浮点数操作，替换的操作包括：卷积操作、矩阵相乘、激活函数、池化操作等。量化节点的输入、输出为浮点数，但是内部运算会通过量化计算转换为 8 位整数（范围为 0 到 255）的运算，浮点数和 8 位量化整数的对应关系示例如右图所示。

量化 Relu 操作的基本

Quantized		Float
0		-10.0
255		30.0
128		10.0

思想如下图所示：



TensorFlow Android 应用开发环境构建

在 Android 系统上使用 TensorFlow 模型做 Inference 依赖于两个文件 libtensorflow_inference.so 和 libandroid_tensorflow_inference_java.jar。这两个文件可以通过下载 TensorFlow 源代码后，采用 bazel 编译出来，如下所示。

下载 TensorFlow 源代码。

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

下载安装 [Android NDK](#)。

下载安装 [Android SDK](#)。

配置 tensorflow/WORKSPACE 中 android 开发工具路径：

```
android_sdk_repository(name = "androidsdk", api_level = 23, build_tools_version
```

```
= "25.0.2", path = "/opt/android",)
android_ndk_repository(name="androidndk", path="/opt/android/android-ndk-r12b",
api_level=14)
```

编译 libtensorflow_inference.so:

```
bazel build -c opt //tensorflow/contrib/android:libtensorflow_inference.so
--crosstool_top=/external:android/crosstool --host_crosstool_top=
@bazel_tools//tools/cpp:toolchain --cpu=armeabi-v7a
```

编译 libandroid_tensorflow_inference_java.jar:

```
bazel build //tensorflow/contrib/android:android_tensorflow_inference_java
```

TensorFlow 提供了 [Android 开发的示例框架](#)，下面基于 AlexNet 模型的看花识名 APP 做一些相应源码的修改，并编译生成 Android 的安装包：

基于 AlexNet 模型，修改 Inference 的输入、输出的 Tensor 名称：

```
private static final String INPUT_NAME = "input";
private static final String OUTPUT_NAME
= "alexnet_v2/fc8/squeezed";
```

放置 quantized_alexnet_v2_graph.

pb 和对应的 labels.txt 文件到 assets 目录下，并修改 Android 文件路径：

```
private static final String MODEL_FILE
= "file:///android_asset/quantized_
alexnet_v2_graph.pb";
private static final String LABEL_FILE =
"file:///android_asset/labels.txt";
```

编译生成安装包：

```
bazel build -c opt //tensorflow/examples/
android:tensorflow_demo
```

拷贝 tensorflow_demo.apk 到手机上，并执行安装，太阳花识别效果如右图所示。



TensorFlow 移动端应用开发 API

在 Android 系统中执行 TensorFlow Inference 操作，需要调用 libandroid_tensorflow_inference_java.jar 中的 JNI 接口，主要接口如下：

构建 TensorFlow Inference 对象，构建该对象时候会加载 TensorFlow 动态链接库 libtensorflow_inference.so 到系统中；参数 assetManager 为 android asset 管理器；参数 modelFilename 为 TensorFlow 模型文件在 android_asset 中的路径。

```
TensorFlowInferenceInterface inferenceInterface = new  
TensorFlowInferenceInterface(assetManager, modelFilename);
```

向 TensorFlow 图中加载输入数据，本 App 中输入数据为摄像头截取到的图片；参数 inputName 为 TensorFlow Inference 中的输入数据 Tensor 的名称；参数 floatValues 为输入图片的像素数据，进行预处理后的浮点值；[1,inputSize,inputSize,3] 为裁剪后图片的大小，比如 1 张 224*224*3 的 RGB 图片。

```
inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
```

执行模型推理；outputNames 为 TensorFlow Inference 模型中要运算 Tensor 的名称，本 APP 中为分类的 Logist 值。

```
inferenceInterface.run(outputNames);
```

获取模型 Inference 的运算结果，其中 outputName 为 Tensor 名称，参数 outputs 存储 Tensor 的运算结果。本 APP 中，outputs 为计算得到的 Logist 浮点数组。

```
inferenceInterface.fetch(outputName, outputs);
```

总结

本文基于看花识名 APP，讲解了 TensorFlow 在 Android 智能终端中的应用技术。首先回顾了 AlexNet 模型结构，基于 AlexNet 的 slim 模型对 Flowers 数据进行训练；对训练后的 CheckPoint 数据，进行 Freeze 和 Quantized 处理，生成智能终端要用的 Inference 模型。然后介绍

了 TensorFlow Android 应用开发环境的构建，编译生成 TensorFlow 在 Android 上的动态链接库以及 java 开发包；文章最后介绍了 Inference API 的使用方式。

参考文献

- <http://www.tensorflow.org>



EGO会员招募季第三季正式开启

EGO旨在组建全球最具影响力的技术领导者社交网络

联结杰出的技术领导者学习和成长



申请加入
扫码联系E小欧

版权声明

InfoQ 中文站出品

深度学习利器：TensorFlow程序设计

©2017北京极客邦科技有限公司

本书版权为北京极客邦科技有限公司所有，未经出版者预先的书面许可，不得以任何方式复制或者抄袭本书的任何部分，本书任何部分不得用于再印刷，存储于可重复使用的系统，或者以任何方式进行电子、机械、复印和录制等形式传播。

本书提到的公司产品或者使用到的商标为产品公司所有。

如果读者要了解具体的商标和注册信息，应该联系相应的公司。

出版：北京极客邦科技有限公司

北京市朝阳区洛娃大厦C座1607

欢迎共同参与 InfoQ 中文站的内容建设工作，包括原创投稿和翻译，请联系 editors@cn.infoq.com。

网址：www.infoq.com.cn