

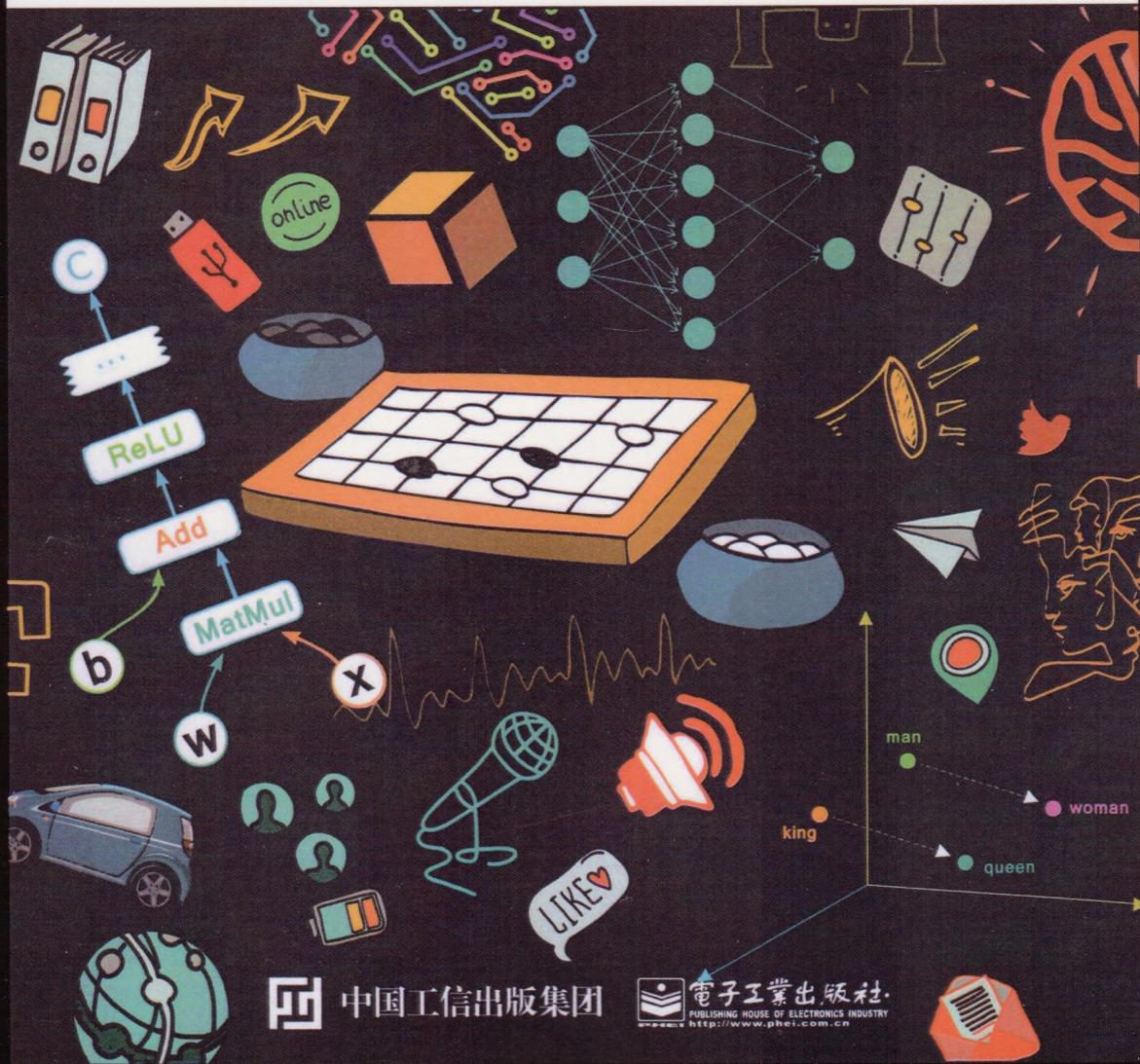
Google TensorFlow 工程研发总监力荐  
一本书掌握谷歌深度学习框架

Broadview®  
[www.broadview.com.cn](http://www.broadview.com.cn)

# TensorFlow

## 实战

黄文坚 唐源 著



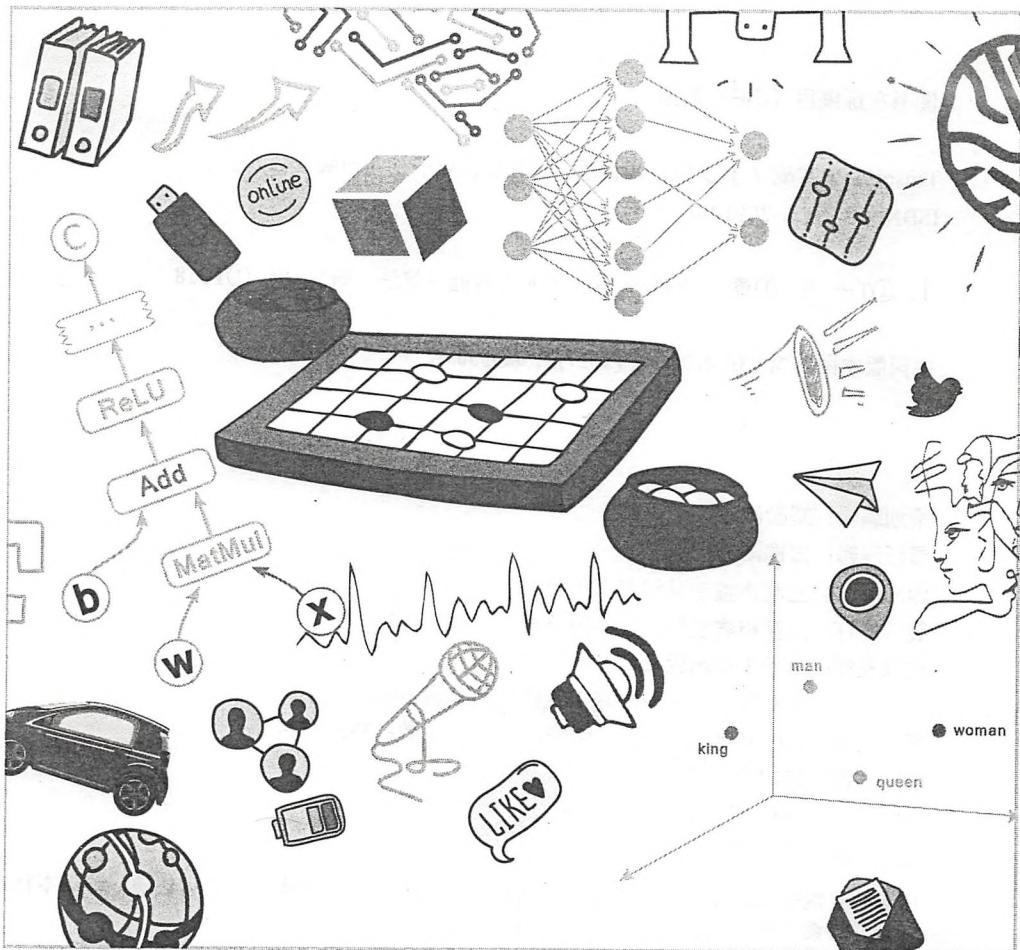
中国工信出版集团



电子工业出版社  
<http://www.phei.com.cn>

# TensorFlow 实战

黄文坚 唐源 著



電子工業出版社  
Publishing House of Electronics Industry  
北京•BEIJING

## 内 容 简 介

本书介绍了 TensorFlow 的基础原理和应用，并侧重于结合实际例子讲解使用 TensorFlow 的方法。TensorFlow 目前最主要的应用是在机器学习和深度学习领域，本书讲解了全连接神经网络、卷积神经网络、循环神经网络、深度强化学习等常见的深度学习模型，还介绍了 TensorBoard、单机多 GPU 并行、分布式并行，TF Learn 和其他 TensorFlow 辅助组件。

希望快速上手 TensorFlow、了解深度学习技术及其应用实践的人士，以及机器学习、分布式计算领域的学生、从业者。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目 (CIP) 数据

TensorFlow 实战 / 黄文坚, 唐源著. —北京：电子工业出版社，2017.2

ISBN 978-7-121-30912-0

I. ①T… II. ①黄… ②唐… III. ①人工智能—算法—研究 IV. ①TP18

中国版本图书馆 CIP 数据核字(2017)第 024533 号

策划编辑：郑柳洁

责任编辑：郑柳洁

印 刷：三河市鑫金马印装有限公司

装 订：三河市鑫金马印装有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：720×1000 1/16 印张：19.5 字数：335 千字

版 次：2017 年 2 月第 1 版

印 次：2017 年 2 月第 2 次印刷

定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888, 88258888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：010-51260888-819, [faq@phei.com.cn](mailto:faq@phei.com.cn)。

# 好评袭来

“AI and Machine Learning are going to be a key part of our future. We made TensorFlow open source to bring these technologies to everyone and help move the world forward. This book is a great example of the TensorFlow community giving back to multiply everyone's efforts.”

Engineering Director of TensorFlow, Rajat Monga

TensorFlow 的开源对整个学术界及工业界都产生了巨大的影响，可以比做机器学习的 Hadoop。本书涵盖了从多层次感知机、CNN、RNN 到强化学习等一系列模型的 TensorFlow 实现；在详尽地介绍算法和模型的细节的同时穿插实际的代码，对帮助读者快速建立算法和代码的联系大有助益；对入门 TensorFlow 和深度学习的研究者来说是一份非常好的学习材料。

360 首席科学家，颜水成

TensorFlow 是基于 Computation Graph 的机器学习框架，支持 GPU 和分布式，是目前最有影响力的开源深度学习系统。TensorFlow 的工程实现非常优秀，拓展也非常灵活，对机器学习尤其是深度学习的推广大有裨益。本书结合了大量的实际例子，清晰地讲解了

如何使用 TensorFlow 构筑常见的深度学习模型，可通读也可作为工具书查阅。在本书上市前，国内还没有介绍 TensorFlow 的技术书籍，推荐对 TensorFlow 或深度学习感兴趣的人士阅读此书。

北京大学计算机系教授 网络与信息系统研究所所长，崔斌

深度学习乃至人工智能正逐渐在 FinTech 领域发挥巨大的作用，其应用包括自动报告生成、金融智能搜索、量化交易和智能投顾。而 TensorFlow 为金融业方便地使用深度学习提供了可能。本书介绍了通过 TensorFlow 实现各类神经网络的案例，非常适合初学者快速入门。

PPmoney CTO，康德胜

TensorFlow 是 Google 开源的一套深度学习框架，已发展成为最主流的深度学习框架，目前在市面上没有看到关于 TensorFlow 的中文书籍出版。本书一方面一步步地介绍了 TensorFlow 的使用方法，使得没有使用过的人可以很快上手使用；另一方面，讲解了诸如卷积神经网络、循环神经网络、强化学习、自编码器等深度学习知识，使得不懂深度学习的人也可以入门。本书在介绍基本知识和原理的同时，用实例进行讲解，比较适合初学者学习使用 TensorFlow 及深度学习知识。

格灵深瞳 CTO，邓亚峰

《TensorFlow 实战》由浅入深，透过大量的代码实例，为读者揭开深度学习的层层面纱，加深理论理解的同时，也更好地联系了实际应用。

小米图像算法资深工程师，万韶华

# 前言

AlphaGo 在 2017 年年初化身 Master，在弈城和野狐等平台上连胜中日韩围棋高手，其中包括围棋世界冠军井山裕太、朴廷桓、柯洁等，还有棋圣聂卫平，总计取得 60 连胜，未尝败绩。遥想 2016 年 3 月，当时 AlphaGo 挑战李世石还一度不被看好，到今日已经可以完胜各位高手。AlphaGo 背后神秘的推动力就是 TensorFlow——Google 于 2015 年 11 月开源的机器学习及深度学习框架。DeepMind 宣布全面迁移到 TensorFlow 后，AlphaGo 的算法训练任务就全部放在了 TensorFlow 这套分布式框架上。

TensorFlow 在 2015 年年底一出现就受到了极大的关注，在一个月内获得了 GitHub 上超过一万颗星的关注，目前在所有的机器学习、深度学习项目中排名第一，甚至在所有的 Python 项目中也排名第一。本书将重点从实用的层面，为读者讲解如何使用 TensorFlow 实现全连接神经网络、卷积神经网络、循环神经网络，乃至 Deep Q-Network。同时结合 TensorFlow 原理，以及深度学习的部分知识，尽可能让读者通过学习本书做出实际项目和成果。

本书各章节间没有太强的依赖关系，如果读者对某一章感兴趣，可以直接阅读。本书使用 TensorFlow 1.0.0-rc0 作为示例讲解，应该与最新版的 TensorFlow 兼容绝大部分代码，可能存在少数接口的更新，读者可参阅提示信息。书中大部分代码是 Python 代码，这也是 TensorFlow 支持的最全、最完整的接口语言。

本书的前两章介绍了 TensorFlow 的基础知识和概念。第 3 章和第 4 章介绍了简单的

示例及全连接神经网络。第 5 章和第 6 章介绍了基础的卷积神经网络，以及目前比较经典的 AlexNet、VGGNet、Inception Net 和 ResNet。第 7 章介绍了 Word2Vec、RNN 和 LSTM。第 8 章介绍了强化学习，以及基于深度学习的策略网络和估值网络。第 9 章介绍了 TensorBoard、单机多 GPU 并行，以及分布式并行。

第 10 章介绍了 TensorFlow 里面的 contrib.learn 模块，包含许多类型的深度学习及流行的机器学习算法的使用方法，也解析了这个模块的分布式 Estimator 的基本架构，以及如何使用 Estimator 快速搭建自己的分布式机器学习模型架构，进行模型的训练和评估，也介绍了如何使用监督器更好地监测和跟踪模型的训练及使用 DataFrame 读取不同的数据格式。第 11 章介绍了 Contrib 模块，这个模块里提供了许多机器学习需要的功能，包括统计分布、机器学习层、优化函数、指标，等等。本章将简单介绍其中的一些功能让大家了解 TensorFlow 的涵盖范围，并感受到社区的积极参与和贡献度。第 10 章和第 11 章使用了 TensorFlow 0.11.0-rc0 版本作为示例讲解。

作者在写作本书时，获得了亲人、同事、好友的帮助，在此非常感谢你们的支持。

## 作 者

# 目录

<b>1</b>	<b>TensorFlow 基础</b>	<b>1</b>
1.1	TensorFlow 概要 .....	1
1.2	TensorFlow 编程模型简介 .....	4
<b>2</b>	<b>TensorFlow 和其他深度学习框架的对比</b>	<b>18</b>
2.1	主流深度学习框架对比 .....	18
2.2	各深度学习框架简介 .....	20
<b>3</b>	<b>TensorFlow 第一步</b>	<b>39</b>
3.1	TensorFlow 的编译及安装 .....	39
3.2	TensorFlow 实现 Softmax Regression 识别手写数字 .....	46
<b>4</b>	<b>TensorFlow 实现自编码器及多层感知机</b>	<b>55</b>
4.1	自编码器简介 .....	55
4.2	TensorFlow 实现自编码器 .....	59
4.3	多层感知机简介 .....	66

4.4 TensorFlow 实现多层感知机 .....	70
<b>5 TensorFlow 实现卷积神经网络</b>	<b>74</b>
5.1 卷积神经网络简介 .....	74
5.2 TensorFlow 实现简单的卷积网络 .....	80
5.3 TensorFlow 实现进阶的卷积网络 .....	83
<b>6 TensorFlow 实现经典卷积神经网络</b>	<b>95</b>
6.1 TensorFlow 实现 AlexNet .....	97
6.2 TensorFlow 实现 VGGNet .....	108
6.3 TensorFlow 实现 GoogleInceptionNet .....	119
6.4 TensorFlow 实现 ResNet .....	143
6.5 卷积神经网络发展趋势 .....	156
<b>7 TensorFlow 实现循环神经网络及 Word2Vec</b>	<b>159</b>
7.1 TensorFlow 实现 Word2Vec .....	159
7.2 TensorFlow 实现基于 LSTM 的语言模型 .....	173
7.3 TensorFlow 实现 BidirectionalLSTMClassifier .....	188
<b>8 TensorFlow 实现深度强化学习</b>	<b>195</b>
8.1 深度强化学习简介 .....	195
8.2 TensorFlow 实现策略网络 .....	201
8.3 TensorFlow 实现估值网络 .....	213
<b>9 TensorBoard、多 GPU 并行及分布式并行</b>	<b>233</b>
9.1 TensorBoard .....	233
9.2 多 GPU 并行 .....	243
9.3 分布式并行 .....	249

---

<b>10</b>	<b>TF.Learn 从入门到精通</b>	<b>259</b>
10.1	分布式 Estimator .....	259
10.2	深度学习 Estimator .....	267
10.3	机器学习 Estimator .....	272
10.4	DataFrame .....	278
10.5	监督器 Monitors .....	279
<b>11</b>	<b>TF.Contrib 的其他组件</b>	<b>283</b>
11.1	统计分布 .....	283
11.2	Layer 模块 .....	285
11.3	性能分析器 tfprof .....	293
<b>参考文献</b>		<b>297</b>

---



# TensorFlow 基础

## 1.1 TensorFlow 概要

Google 第一代分布式机器学习框架 DistBelief<sup>1</sup>，在内部大规模使用后并没有选择开源。而后第二代分布式机器学习系统 TensorFlow<sup>2</sup>终于选择于 2015 年 11 月在 GitHub 上开源，且在 2016 年 4 月补充了分布式版本，并于 2017 年 1 月发布了 1.0 版本的预览，API 接口趋于稳定。目前 TensorFlow 仍处于快速开发迭代中，有大量新功能及性能优化在持续研发。TensorFlow 最早由 Google Brain 的研究员和工程师开发，设计初衷是加速机器学习的研究，并快速地将研究原型转化为产品。Google 选择开源 TensorFlow 的原因也非常简单：第一是希望通过社区的力量，让大家一起完善 TensorFlow。之前 Google 内部 DistBelief 及 TensorFlow 的用户就贡献了非常多的意见和反馈，使得产品质量得到了快速提升；第二是回馈社区，Google 希望让这个优秀的工具得到更多的应用，从整体上提高学术界乃至工业界使用深度学习的效率。除了 TensorFlow，Google 也开源过大量成功的项目，包括大名鼎鼎的移动操作系统 Android、浏览器 Chromium、编程语言 Go、JavaScript 引擎 V8、数据交换框架 Protobuf、编译工具 Bazel、OCR 工具 Tesseract 等共计数百个高质量的项目。

TensorFlow 的官方网址：[www.tensorflow.org](http://www.tensorflow.org)

GitHub 网址：[github.com/tensorflow/tensorflow](https://github.com/tensorflow/tensorflow)

模型仓库网址: [github.com/tensorflow/models](https://github.com/tensorflow/models)

TensorFlow 既是一个实现机器学习算法的接口,同时也是执行机器学习算法的框架。它前端支持 Python、C++、Go、Java 等多种开发语言,后端使用 C++、CUDA 等写成。TensorFlow 实现的算法可以在众多异构的系统上方便地移植,比如 Android 手机、iPhone、普通的 CPU 服务器,乃至大规模 GPU 集群,如图 1-1 所示。除了执行深度学习算法,TensorFlow 还可以用来实现很多其他算法,包括线性回归、逻辑回归、随机森林等。TensorFlow 建立的大规模深度学习模型的应用场景也非常广,包括语音识别、自然语言处理、计算机视觉、机器人控制、信息抽取、药物研发、分子活动预测等,使用 TensorFlow 开发的模型也在这些领域获得了最前沿的成果。

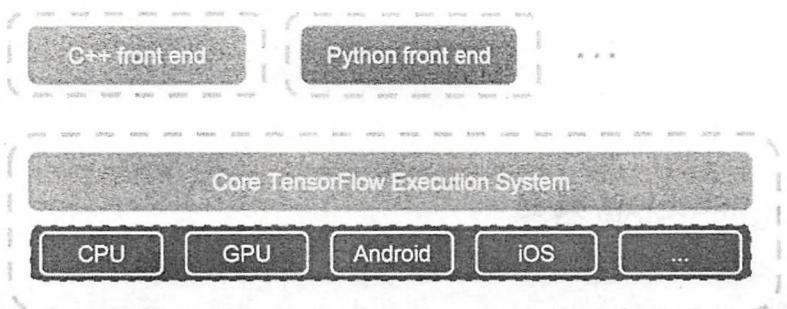


图 1-1 TensorFlow 基础架构

为了研究超大规模的深度神经网络,Google 在 2011 年启动了 Google Brain 项目,同时开发了第一代的分布式机器学习框架 DistBelief。有超过 50 个 Google 的团队在他们的产品中使用了 DistBelief,比如 Google Search 中的搜索结果排序、Google Photos 中的图片标注、Google Translate 中的自然语言处理等,都依赖于 DistBelief 建立的深度学习模型。Google 基于使用 DistBelief 时的经验及训练大规模分布式神经网络的需求,开发了 TensorFlow——第二代分布式机器学习算法实现框架和部署系统。Google 将著名的 Inception Net 从 DistBelief 移植到 TensorFlow 后,获得了 6 倍的训练速度提升。目前,在 Google 内部使用 TensorFlow 的项目呈爆炸性的增长趋势,在 2016 年已经有超过 2000 个项目使用了 TensorFlow 建立的深度学习模型,而且这个数字还在高速增长中,如图 1-2 所示。

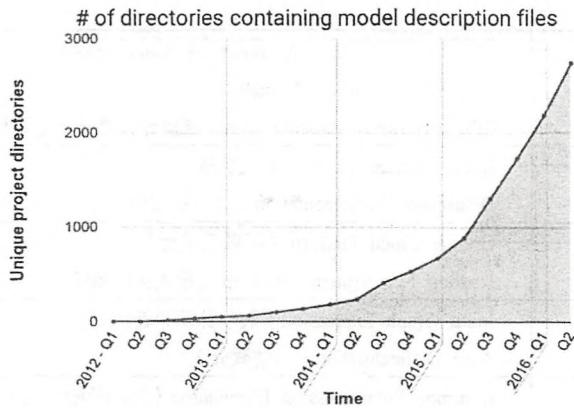


图 1-2 TensorFlow 在 Google 的使用趋势

TensorFlow 使用数据流式图来规划计算流程，它可以将计算映射到不同的硬件和操作系统平台。凭借着统一的架构，TensorFlow 可以方便地部署到各种平台，大大简化了真实场景中应用机器学习的难度。使用 TensorFlow 我们不需要给大规模的模型训练和小规模的应用部署开发两套不同的系统，避免了同时维护两套程序的成本，TensorFlow 给训练和预测的共同部分提供了一个恰当的抽象。TensorFlow 的计算可以表示为有状态的数据流式图，对于大规模的神经网络训练，TensorFlow 可以让用户简单地实现并行计算，同时使用不同的硬件资源进行训练，同步或异步地更新全局共享的模型参数和状态。将一个串行的 TensorFlow 算法改造成并行的成本也是非常低的，通常只需要对小部分代码进行改写。相比于 DistBelief，TensorFlow 的计算模型更简洁灵活，计算性能显著提升，同时支持更多的异构计算系统。大量 Google 内部的 DistBelief 用户转向了 TensorFlow，他们使用 TensorFlow 进行各种研究和产品开发，包括在手机上跑计算机视觉模型，或是训练有数百亿参数、数千亿数据的神经网络模型。虽然绝大多数的 TensorFlow 应用都在机器学习及深度学习领域，但 TensorFlow 抽象出的数据流式图也可以应用在通用数值计算和符号计算上，比如分形图计算或者偏微分方程数值求解。表 1-1 所示为 TensorFlow 的主要技术特性。

表 1-1 TensorFlow 的主要技术特性

编程模型	Dataflow-like model (数据流模型)
语言	Python、C++、Go、Rust、Haskell、Java（还有非官方的 Julia、JavaScript、R 的支持）
部署	Code once, run everywhere (一次编写，各处运行)

续表

计算资源	CPU (Linux、Mac、Windows、Android、iOS) GPU (Linux、Mac、Windows) TPU (Tensor Processing Unit, 张量计算单元, 主要用作推断)
实现方式	Local Implementation (单机实现) Distributed Implementation (分布式实现)
平台支持	Google Cloud Platform (谷歌云平台) Hadoop File System (Hadoop 分布式文件系统)
数学表达	Math Graph Expression (数学计算图表达) Auto Differentiation (自动微分)
优化	Common Subexpression Elimination (共同子图消除) Asynchronous Kernel Optimization (异步核优化) Communication Optimization (通信优化) Model Parallelism (模型并行) Data Parallelism (数据并行) Pipeline (流水线)

## 1.2 TensorFlow 编程模型简介

### 1.2.1 核心概念

TensorFlow 中的计算可以表示为一个有向图 (directed graph), 或称计算图 (computation graph), 其中每一个运算操作 (operation) 将作为一个节点 (node), 节点与节点之间的连接称为边 (edge)。这个计算图描述了数据的计算流程, 它也负责维护和更新状态, 用户可以对计算图的分支进行条件控制或循环操作。用户可以使用 Python、C++、Go、Java 等几种语言设计这个数据计算的有向图。计算图中每一个节点可以有任意多个输入和任意多个输出, 每一个节点描述了一种运算操作, 节点可以算是运算操作的实例化 (instance)。在计算图的边中流动 (flow) 的数据被称为张量 (tensor), 故得名 TensorFlow。而 tensor 的数据类型, 可以是事先定义的, 也可以根据计算图的结构推断得到。有一类特殊的边中没有数据流动, 这种边是依赖控制 (control dependencies), 作用是让它的起始节点执行完之后再执行目标节点, 用户可以使用这样的边进行灵活的条件控制, 比如限制内存使用的最高峰值。下面是用 Python 设计并执行计算图的示例。计算图示例如图 1-3 所示。

```

import tensorflow as tf
b=tf.Variable(tf.zeros([100]))      # 生成 100 维的向量，初始化为 0
W=tf.Variable(tf.random_uniform([784,100],-1,1)) # 生成 784x100 的随机矩阵 W
x=tf.placeholder(name="x")          # 输入的 Placeholder
relu=tf.nn.relu(tf.matmul(W, x)+b)  # ReLU(Wx+b)
C=[...]
s=tf.Session()
for step in range(0, 10):
    input=...construct 100-D input array... # 为输入创建一个 100 维的向量
    result=s.run(C, feed_dict={x: input})   # 获取 Cost，供给输入 x
    print(step, result)

```

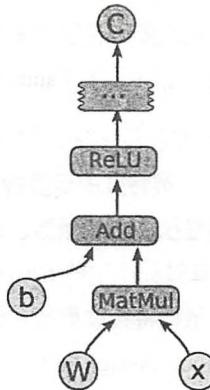


图 1-3 计算图示例

一个运算操作代表了一种类型的抽象运算，比如矩阵乘法或者向量加法。运算操作可以有自己的属性，但是所有属性必须被预先设置，或者能在创建计算图时被推断出来。通过设置运算操作的属性可以用来支持不同的 tensor 元素类型，比如让向量加法支持浮点数（float）或者整数（int）。运算核（kernel）是一个运算操作在某个具体硬件（比如在 CPU 或者 GPU 中）的实现。在 TensorFlow 中，可以通过注册机制加入新的运算操作或者运算核。表 1-2 所示为部分 TensorFlow 内建的运算操作。

表 1-2 TensorFlow 内建的运算操作

类    型	示    例
标量运算	Add、Sub、Mul、Div、Exp、Log、Greater、Less、Equal

续表

类 型	示 例
向量运算	Concat、Slice、Split、Constant、Rank、Shape、Shuffle
矩阵运算	MatMul、MatrixInverse、MatrixDeterminant
带状态的运算	Variable、Assign、AssignAdd
神经网络组件	SoftMax、Sigmoid、ReLU、Convolution2D、MaxPooling
储存、恢复	Save、Restore
队列及同步运算	Enqueue、Dequeue、MutexAcquire、MutexRelease
控制流	Merge、Switch、Enter、Leave、NextIteration

Session 是用户使用 TensorFlow 时的交互式接口。用户可以通过 Session 的 Extend 方法添加新的节点和边, 用以创建计算图, 然后就可以通过 Session 的 Run 方法执行计算图: 用户给出需要计算的节点, 同时提供输入数据, TensorFlow 就会自动寻找所有需要计算的节点并按依赖顺序执行它们。对绝大多数的用户来说, 他们只会创建一次计算图, 然后反复地执行整个计算图或是其中的一部分子图 (sub-graph)。

在大多数运算中, 计算图会被反复执行多次, 而数据也就是 tensor 并不会被持续保留, 只是在计算图中过一遍。Variable 是一类特殊的运算操作, 它可以将一些需要保留的 tensor 储存在内存或显存中, 比如神经网络模型中的系数。每一次执行计算图后, Variable 中的数据 tensor 将会被保存, 同时在计算过程中这些 tensor 也可以被更新, 比如神经网络每一次 mini-batch 训练时, 神经网络的系数将会被更新并保存。使用 Variable, 可以在计算图中实现一些特殊的操作, 比如 Assign、AssignAdd ( $+=$ ) 或 AssignMul ( $\*=$ )。

## 1.2.2 实现原理

TensorFlow 有一个重要组件 client, 顾名思义, 就是客户端, 它通过 Session 的接口与 master 及多个 worker 相连。其中每一个 worker 可以与多个硬件设备 (device) 相连, 比如 CPU 或 GPU, 并负责管理这些硬件。而 master 则负责指导所有 worker 按流程执行计算图。TensorFlow 有单机模式和分布式模式两种实现, 其中单机指 client、master、worker 全部在一台机器上的同一个进程中; 分布式的版本允许 client、master、worker 在不同机器的不同进程中, 同时由集群调度系统统一管理各项任务。图 1-4 所示为单机版和分布式版本的示例图。

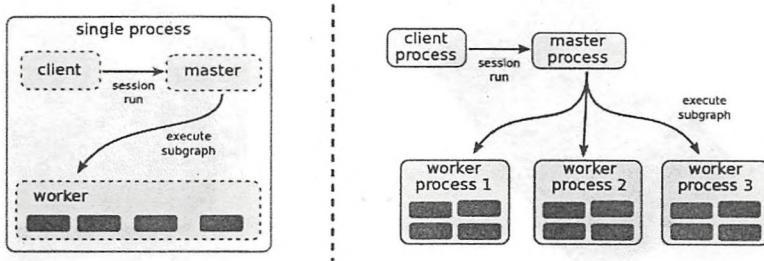


图 1-4 TensorFlow 单机版本和分布式版本的示例图

TensorFlow 中每一个 worker 可以管理多个设备,每一个设备的 name 包含硬件类别、编号、任务号(单机版本没有),示例如下。

单机模式: /job:localhost/device:cpu:0

分布式模式: /job:worker/task:17/device:gpu:3

TensorFlow 为 CPU 和 GPU 提供了管理设备的对象接口,每一个对象负责分配、释放设备的内存,以及执行节点的运算核。TensorFlow 中的 tensor 是多维数组,数据类型支持 8 位至 64 位的 int,以及 IEEE 标准的 float、double 和复数型,同时还支持任意字符串。每一个设备有单独的 allocator 负责储存各种数据类型的 tensor,同时 tensor 的引用次数也会被记录,当引用数为 0 时,内存将被释放。如图 1-5 所示,TensorFlow 支持的设备包括 x86 架构 CPU、手机上的 ARM CPU、GPU、TPU (Tensor Processing Unit, Google 专门为大规模深度学习计算定制的芯片,但目前还没有公开发布的计划),例如 AlphaGo 在与李世石比赛时就大量使用了 TPU 集群的计算资源。

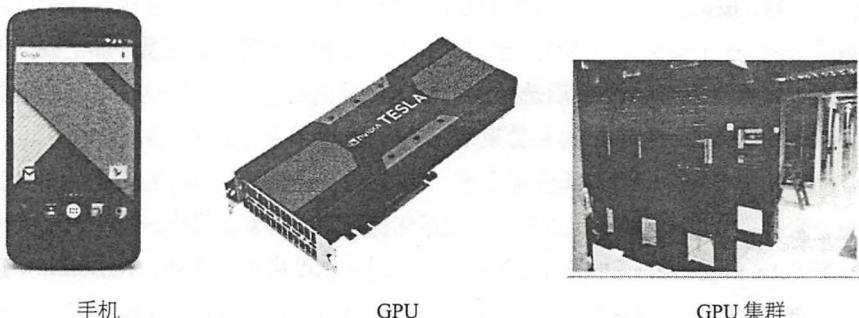
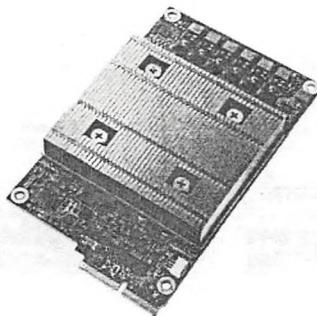
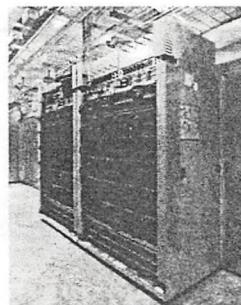


图 1-5 TensorFlow 支持的设备



TPU



TPU 集群

图 1-5 TensorFlow 支持的设备（续）

在只有一个硬件设备的情况下，计算图会按依赖关系被顺序执行。当一个节点的所有上游依赖都被执行完时（依赖数为 0），这个节点就会被加入 ready queue 以等待执行。同时，它下游所有节点的依赖数减 1，实际上这就是标准的计算拓扑序的方式。当有多个设备时，情况就变得复杂了，难点有二：

- (1) 每一个节点该让什么硬件设备执行。
- (2) 如何管理节点间的数据通信。

对第 1 个问题，TensorFlow 设计了一套为节点分配设备的策略。这个策略首先需要计算一个代价模型，这个代价模型估算每一个节点的输入、输出 tensor 的大小，以及所需要的计算时间。代价模型一部分由人工经验制定的启发式规则得到，另一部分则是由对一小部分数据进行实际运算而测量得到的。接下来，分配策略会模拟执行整个计算图，首先会从起点开始，按拓扑序执行。在模拟执行一个节点时，会把每一个能执行这个节点的设备都测试一遍，这个测试会考虑代价模型对这个节点的计算时间的估算，加上数据传到这个设备上所需要的通信时间，最后选择一个综合时间最短的设备作为这个节点的运算设备。可以看到这个策略是一个简单的贪婪策略，它不能确保找到全局最优解，但是可以用较快的速度找到一个不错的节点运算分配方案。同时除了运行时间，内存的最高使用峰值也会被考虑进来。目前 TensorFlow 的节点分配策略仍在不断研发、优化，将来可能使用一个强化学习（Reinforcement Learning）的神经网络来辅助决策。此外，TensorFlow 还允许用户对节点的分配设置限制条件，比如“只给这个节点分配 GPU 类型的设备”，“只给这个节点分配/job:worker/task:17 上的设备”，“这个节点分配的设备必须和 variable13 一致”。对于这些限制条件，TensorFlow 会先计算每个节点可以使用的设备，再使用并查集

( union-find ) 算法找到必须使用同一个设备的节点。

我们再来看第 2 个问题，当给节点分配设备的方案被确定，整个计算图就会被划分为许多子图，使用同一个设备并且相邻的节点会被划分到同一个子图。然后计算图中从  $x$  到  $y$  的边，会被取代为一个发送端的发送节点( send node )、一个接收端的接收节点( receive node )，以及从发送节点到接收节点的边，如图 1-6 所示。

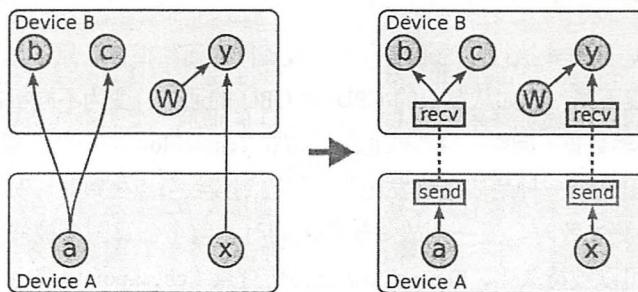


图 1-6 TensorFlow 的通信机制

这样就把数据通信的问题转变为发送节点和接收节点的实现问题，用户不需要为不同的硬件环境实现通信方法。同时两个子图之间可能会有多个接收节点，如果这些接收节点接收的都是同一个 tensor，那么所有这些接收节点会被自动合并为一个，避免了数据的反复传输或者重复占用设备内存。总结一下，TensorFlow 的通信机制很优秀，发送节点和接收节点的设计简化了底层的通信模式，用户无须设计节点之间的通信流程，可以让同一套代码，自动扩展到不同硬件环境并处理复杂的通信流程。图 1-7 所示为 CPU 与 GPU 之间通信的流程。

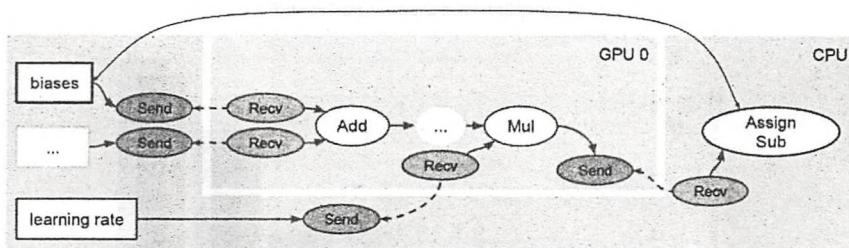


图 1-7 CPU 与 GPU 的通信过程

同时，从单机单设备的版本改造为单机多设备的版本也非常容易，下面的代码只添加了加粗的这一行，就实现了从一块 GPU 训练到多块 GPU 训练的改造。

```

for i in range(8):
    for d in range(4):
        with tf.device("/gpu:%d" % d):
            input = x[i] if d == 0 else m[d-1]
            m[d], c[d] = LSTMCell(input, mprev[d], cprev[d])
            mprev[d] = m[d]
            cprev[d] = c[d]

```

TensorFlow 分布式执行时的通信和单机设备间的通信很像，只不过是对发送节点和接收节点的实现不同：比如从单机的 CPU 到 GPU 的通信，变为不同机器之间使用 TCP 或者 RDMA 传输数据。同时，容错性也是分布式 TensorFlow 的一个特点。故障会在两种情况下被检测出来，一种是信息从发送节点传输到接收节点失败时，另一种是周期性的 worker 心跳检测失败时。当一个故障被检测到时，整个计算图会被终止并重启。其中 Variable node 可以被持久化，TensorFlow 支持检查点（checkpoint）的保存和恢复，每一个 Variable node 都会链接一个 Save node，每隔几轮迭代就会保存一次数据到持久化的储存系统，比如一个分布式文件系统。同样，每一个 Variable node 都会连接一个 Restore node，在每次重启时会被调用并恢复数据。这样在发生故障并重启后，模型的参数将得以保留，训练将从上一个 checkpoint 恢复而不需要完全从头再来。

图 1-8 所示为使用 TensorFlow 在 GPU 集群进行分布式训练的性能对比图，在 GPU 数量小于 16 时，基本没有性能损耗。直到 50 块 GPU 时，依然可以获得 80% 的效率，也就是 40 倍于单 GPU 的提速。在 100 块 GPU 时，最终可以获得 56 倍的提速，也就是 56% 的使用效率，可以看到 TensorFlow 在大规模分布式系统上有相当高的并行效率。

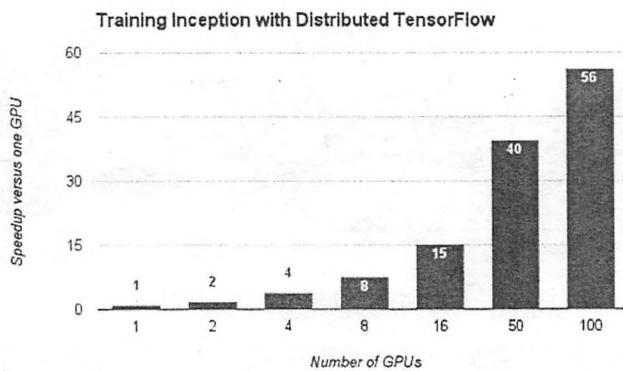


图 1-8 TensorFlow 分布式训练性能

### 1.2.3 拓展功能

在深度学习乃至机器学习中，计算 cost function 的梯度都是最基本的需求，因此 TensorFlow 也原生支持自动求导。比如一个 tensor  $C$  在计算图中有一组依赖的 tensor  $\{X_k\}$ ，那么在 TensorFlow 中可以自动求出  $\{dC/dX_k\}$ 。这个求解梯度的过程也是通过在计算图拓展节点的方式实现的，只不过求梯度的节点对用户是透明的。

如图 1-9 所示，当 TensorFlow 计算一个 tensor  $C$  关于 tensor  $I$  的梯度时，会先寻找从  $I$  到  $C$  的正向路径，然后从  $C$  回溯到  $I$ ，对这条回溯路径上的每一个节点增加一个对应求解梯度的节点，并根据链式法则（chain rule）计算总的梯度，这就是大名鼎鼎的反向传播（back propagation, BP）算法。这些新增的节点会计算梯度函数（gradient function），比如  $[db, dW, dx] = \text{tf.gradients}(C, [b, W, x])$ 。

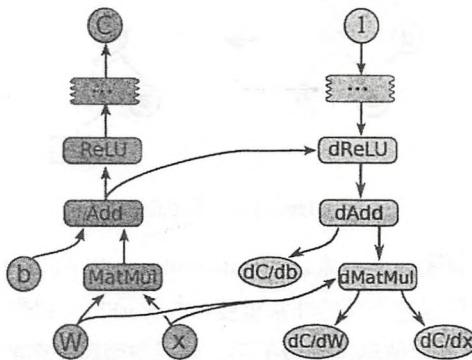


图 1-9 TensorFlow 自动求导示例

自动求导虽然对用户很方便，但伴随而来的是 TensorFlow 对计算的优化（比如为节点分配设备的策略）变得很麻烦，尤其是内存使用的问题。在正向执行计算图，也就是进行推断（inference）时，因为确定了执行顺序，使用经验的规则是比较容易取得好效果的，tensor 在产生后会迅速地被后续节点使用掉，不会持续占用内存。然而在进行反向传播计算梯度时，经常需要用到计算图开头的 tensor，这些 tensor 可能会占用大量的 GPU 显存，也限制了模型的规模。目前 TensorFlow 仍在持续改进这些问题，包括使用更好的优化方法；重新计算 tensor，而不是保存 tensor；将 tensor 从 GPU 显存移到 CPU 控制的主内存中。

TensorFlow 还支持单独执行子图，用户可以选择计算图的任意子图，并沿某些边输入数据，同时从另一些边获取输出结果。TensorFlow 用节点名加 port 的形式指定数据，

例如 `bar:0` 表示名为 `bar` 的节点的第 1 个输出。在调用 `Session` 的 `Run` 方法执行子图时，用户可以选择一组输入数据的映射，比如 `name:port -> tensor`；同时用户必须指定一组输出数据，比如 `name[:port]`，来选择执行哪些节点，如果 `port` 也被选择，那么这些 `port` 输出的数据将会作为 `Run` 函数调用的结果返回。然后整个计算图会根据输入和输出进行调整，输入数据的节点会连接一个 `feed node`，输出数据的节点会连接一个 `fetch node`。TensorFlow 会根据输出数据自动推导出哪些节点需要被执行。如图 1-10 所示，我们选择用一些数据替换掉 `b`，那么 `feed node` 会替代 `b` 连接到 `c`。同时，我们只需要获取 `f:0` 的结果，所以一个 `fetch node` 便会连接到 `f`，这样 `d` 和 `e` 就不会被执行，所以最终需要执行的节点便只有 `a`、`c` 和 `f`。

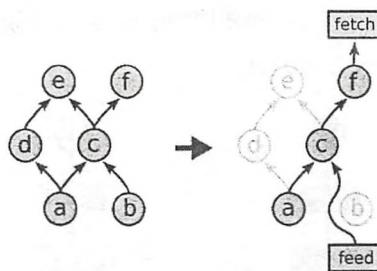


图 1-10 TensorFlow 子图的执行示例

TensorFlow 支持计算图的控制流，比如 `if-condition` 和 `while-loop`，因为大部分机器学习算法需要反复迭代，所以这个功能非常重要。TensorFlow 提供 `Switch` 和 `Merge` 两种 operator，可以根据某个布尔值跳过某段子图，然后把两段子图的结果合并，实现 `if-else` 的功能。同时还提供了 `Enter`、`Leave` 和 `NextIteration` 用来实现循环和迭代。在使用高阶语言（比如 Python）的 `if-else`、`while`、`for` 控制计算流程时，这些控制流会被自动编译为上述那些 operator，方便了用户。Loop 中的每一次循环会有唯一的 `tag`，它的执行结果会输出成 `frame`，这样用户可以方便地查询结果日志。同时，TensorFlow 的控制流支持分布式，每一轮循环中的节点可能分布在不同机器的不同设备上。分布式控制流的实现方式也是依靠计算图的改写，循环内的节点会被划分到不同的小的子图，每一个子图会连接控制节点，实现自己的循环，同时将循环终止等信号发送到其他子图。同时，控制流也提供了对计算图中隐含的梯度计算节点的支持，TensorFlow 会将控制流中的自动求导功能隐藏到底层，用户不需要考虑这部分功能的逻辑设计。

TensorFlow 的数据输入除了通过 `feed node`，也有特殊的 `input node` 可以让用户直接输入文件系统的路径，例如一个 Google Cloud Platform 的文件路径。如果从 `feed node`

输入数据，那么数据必须从 client 读取，并通过网络传到分布式系统的其他节点，这样会有较大的网络开销，直接使用文件路径，可以让 worker 节点读取本地的文件，提高效率。

队列（queue）也是 TensorFlow 任务调度的一个重要特性，这个特性可以让计算图的不同节点异步地执行。使用队列的一个目的是当一个 batch 的数据运算时，提前从磁盘读取下一个 batch 的数据，减少磁盘 I/O 阻塞时间。同时还可以异步地计算许多梯度，再组合成一个更复杂的整体梯度。除了传统的先进先出（FIFO）队列，TensorFlow 还实现了洗牌队列（shuffling queue），用以满足某些机器学习算法对随机性的要求，这对于损失函数优化或者模型收敛会有帮助。

容器（Container）是 TensorFlow 中一种特殊的管理长期变量的机制，例如 Variable 对象就储存在容器中。每一个进程会有一个默认的容器一直存在，直到进程结束。使用容器甚至可以允许不同计算图的不同 Session 之间共享一些状态值。

#### 1.2.4 性能优化

在 TensorFlow 中有很多高度抽象的运算操作，这些运算操作可能是由很多层复杂的计算组合而成的，当有多个高阶运算操作同时存在时，它们的前几层可能是完全一致的重复计算（输入及运算内容均一致）。这时，TensorFlow 会自动识别这些重复计算，同时改写计算图，只执行一次重复的计算，然后把这个高阶运算操作后续的计算连接到这些共有的计算上，避免冗余计算。

同时，巧妙地安排运算的顺序也可以极大地改善数据传输和内存占用的问题。比如适当调整顺序以错开某些数据同时存在于内存的时间，对于显存容量比较小的 GPU 来说至关重要。TensorFlow 也会精细地安排接收节点的执行时间，如果接收节点过早地接收数据，那么数据会堆积在设备的内存中，所以 TensorFlow 设计了策略让接收节点在刚好需要数据来计算时才开始接收数据。

TensorFlow 也提供异步计算的支持，这样线程执行时就无须一直等待某个节点计算完成。有一些节点，比如 receive、enqueue、dequeue 就是异步的实现，这些节点不必因等待 I/O 而阻塞一个线程继续执行其他任务。

TensorFlow 同时支持几种高度优化的第三方计算库。

线性代数计算库：

- Eigen<sup>3</sup>

矩阵乘法计算库：

- BLAS<sup>4</sup>
- cuBLAS (CUDA BLAS)<sup>5</sup>

深度学习计算库：

- cuda-convnet<sup>6</sup>
- cuDNN<sup>7</sup>

很多机器学习算法在数字精度较低时依然可以正常工作，TensorFlow 也支持对数据进行压缩。比如将 32-bit 浮点数有损地压缩为 16-bit 浮点数，这样做可以降低在不同机器、不同设备之间传输数据时的网络开销，提高数据通信效率。

TensorFlow 提供了三种不同的加速神经网络训练的并行计算模式。

(1) 数据并行：通过将一个 mini-batch 的数据放在不同设备上计算，实现梯度计算的并行化。例如，将有 1000 条样本的 mini-batch 拆分成 10 份 100 条样本的数据并行计算，完成后将 10 份梯度数据合并得到最终梯度并更新到共享的参数服务器 (parameter server)。这样的操作会产生许多完全一样的子图的副本，在 client 上可以用一个线程同步控制这些副本运算的循环。TensorFlow 中的数据并行如图 1-11 所示。

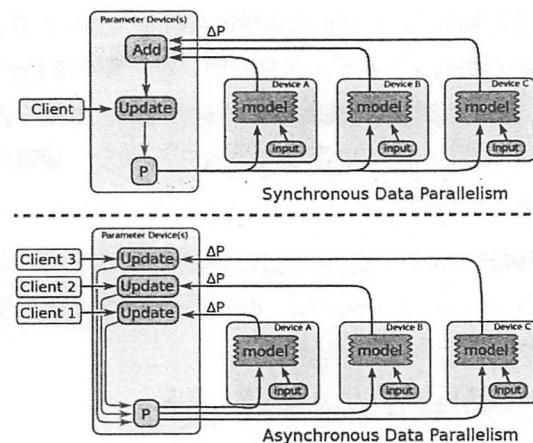


图 1-11 TensorFlow 中的数据并行

上述这个操作还可以改成异步的，使用多个线程控制梯度计算，每一个线程计算完后异步地更新模型参数。同步的方式相当于用了一个较大的 mini-batch（当然其中的批标准化（batch normalization）还是独立的），其优点是没有梯度干扰，缺点是容错性差，一台机器出现问题后可能需要重跑。异步的方式的优点是有一定的容错性，但是因为梯度干扰的问题，导致每一组梯度的利用效率都下降了。另外一种就是混合式，比如两组异步的，其中每组有 50 份同步的训练。从图 1-12 中可以看到，一般来说，同步训练的模型精度较好。

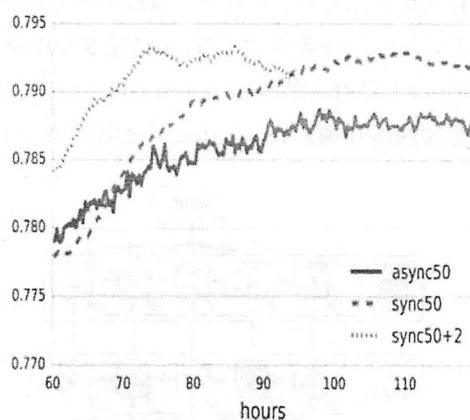


图 1-12 数据并行中同步、异步训练的性能对比

图 1-13 所示为使用 10 块 GPU 和 50 块 GPU 训练 Inception 时的对比图，要达到相同的精度，50 块 GPU 需要的时间只是 10 块的 1/4 左右。

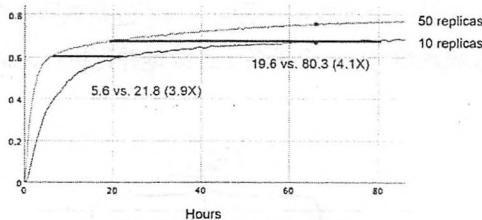


图 1-13 10 块 GPU 和 50 块 GPU 的训练效率对比

相比于模型并行，数据并行的计算性能损耗非常小，尤其是对于 sparse 的 model。因为不同 mini-batch 之间干扰的概率很小，所以经常可以同时进行很多份（replicas）数据并行，甚至可高达上千份。表 1-3 所示为 Google 的各个项目中使用的数据并行的份数或者 GPU 数目。

表 1-3 Google 使用数据并行的项目

RankBrain	500 份数据并行
ImageNet Inception Model	50 块 GPU, 40 倍提速
SmartReply	16 份数据并行, 每一份包含多块 GPU
Language model on “One Billion Word”	32 块 GPU

(2) 模型并行: 将计算图的不同部分放在不同的设备上运算, 可以实现简单的模型并行, 其目标在于减少每一轮训练迭代的时间, 不同于数据并行同时进行多份数据的训练。模型并行需要模型本身有大量可以并行, 且互相不依赖或者依赖程度不高的子图。在不同的硬件环境上性能损耗不同, 比如在单核的 CPU 上使用 SIMD 是没有额外开销的, 在多核 CPU 上使用多线程也基本上没有额外开销, 在多 GPU 上的限制主要在 PCIe 的带宽, 在多机之间的限制则主要在网络开销。TensorFlow 中的模型并行如图 1-14 所示。

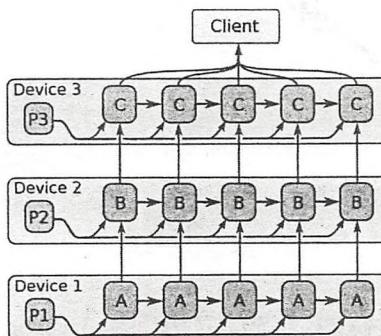


图 1-14 TensorFlow 中的模型并行

(3) 流水线并行: 和异步的数据并行很像, 只不过是在同一个硬件设备上实现并行。大致思路是将计算做成流水线, 在一个设备上连续地并行执行, 提高设备的利用率, 如图 1-15 所示。

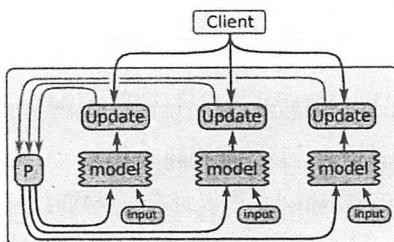
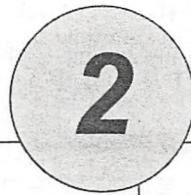


图 1-15 TensorFlow 中的流水线并行

未来，TensorFlow 会支持把任意子图独立出来，封装成一个函数，并让不同的前端语言（比如 Python 或者 C++）来调用。这样将设计好的子图发布在开源社区中，大家的工作就可以方便地被分享了。TensorFlow 也计划推出优化计算图执行的 just-in-time 编译器（目前在 TensorFlow 1.0.0-rc0 中已有试验性的 XLA 组件，可提供 JIT 及 AOT 编译优化），期望可以自动推断出 tensor 的类型、大小，并自动生成一条高度优化过的流水线。同时，TensorFlow 还会持续优化为运算节点分配硬件设备的策略，以及节点执行排序的策略。



# 2

# TensorFlow 和其他深度 学习框架的对比

## 2.1 主流深度学习框架对比

深度学习研究的热潮持续高涨，各种开源深度学习框架也层出不穷，其中包括 TensorFlow、Caffe<sup>8</sup>、Keras<sup>9</sup>、CNTK<sup>10</sup>、Torch7<sup>11</sup>、MXNet<sup>12</sup>、Leaf<sup>13</sup>、Theano<sup>14</sup>、DeepLearning4<sup>15</sup>、Lasagne<sup>16</sup>、Neon<sup>17</sup>，等等。然而 TensorFlow 却杀出重围，在关注度和用户数上都占据绝对优势，大有一统江湖之势。表 2-1 所示为各个开源框架在 GitHub 上的数据统计（数据统计于 2017 年 1 月 3 日），可以看到 TensorFlow 在 star 数量、fork 数量、contributor 数量这三个数据上都完胜其他对手。究其原因，主要是 Google 在业界的号召力确实强大，之前也有许多成功的开源项目，以及 Google 强大的人工智能研发水平，都让大家对 Google 的深度学习框架充满信心，以至于 TensorFlow 在 2015 年 11 月刚开源的第一个月就积累了 10000+ 的 star。其次，TensorFlow 确实在很多方面拥有优异的表现，比如设计神经网络结构的代码的简洁度，分布式深度学习算法的执行效率，还有部署的便利性，都是其得以胜出的亮点。如果一直关注着 TensorFlow 的开发进度，就会发现基本上每星期 TensorFlow 都会有 1 万行以上的代码更新，多则数万行。产品本身优异的质量、快速的

迭代更新、活跃的社区和积极的反馈，形成了良性循环，可以想见 TensorFlow 未来将继续在各种深度学习框架中独占鳌头。

表 2-1 各个开源框架在 GitHub 上的数据统计

框架	机构	支持语言	Stars	Forks	Contributors
TensorFlow	Google	Python/C++/Go/...	41628	19339	568
Caffe	BVLC	C++/Python	14956	9282	221
Keras	fchollet	Python	10727	3575	322
CNTK	Microsoft	C++	9063	2144	100
MXNet	DMLC	Python/C++/R/...	7393	2745	241
Torch7	Facebook	Lua	6111	1784	113
Theano	U. Montreal	Python	5352	1868	271
Deeplearning4J	DeepLearning4J	Java/Scala	5053	1927	101
Leaf	AutumnAI	Rust	4562	216	14
Lasagne	Lasagne	Python	2749	761	55
Neon	NervanaSystems	Python	2633	573	52

观察表 2-1 还可以发现，Google、Microsoft、Facebook 等巨头都参与了这场深度学习框架大战，此外，还有毕业于伯克利大学的贾扬清主导开发的 Caffe，蒙特利尔大学 Lisa Lab 团队开发的 Theano，以及其他个人或商业组织贡献的框架。另外，可以看到各大主流框架基本都支持 Python，目前 Python 在科学计算和数据挖掘领域可以说是独领风骚。虽然有来自 R、Julia 等语言的竞争压力，但是 Python 的各种库实在是太完善了，Web 开发、数据可视化、数据预处理、数据库连接、爬虫等无所不能，有一个完美的生态环境。仅在数据挖掘工具链上，Python 就有 NumPy、SciPy、Pandas、Scikit-learn、XGBoost 等组件，做数据采集和预处理都非常方便，并且之后的模型训练阶段可以和 TensorFlow 等基于 Python 的深度学习框架完美衔接。

表 2-2 和图 2-1 所示为对主流的深度学习框架 TensorFlow、Caffe、CNTK、Theano、Torch 在各个维度的评分，本书 2.2 节会对各个深度学习框架进行比较详细的介绍。

表 2-2 主流深度学习框架在各个维度的评分

	模型设计	接口	部署	性能	架构设计	总体评分
TensorFlow	80	80	90	90	100	88
Caffe	60	60	90	80	70	72

续表

	模型设计	接 口	部 署	性 能	架构设计	总体评分
CNTK	50	50	70	100	60	66
Theano	80	70	40	50	50	58
Torch	90	70	60	70	90	76
MXNet	70	100	80	80	90	84
DeepLearning4J	60	70	80	80	70	72

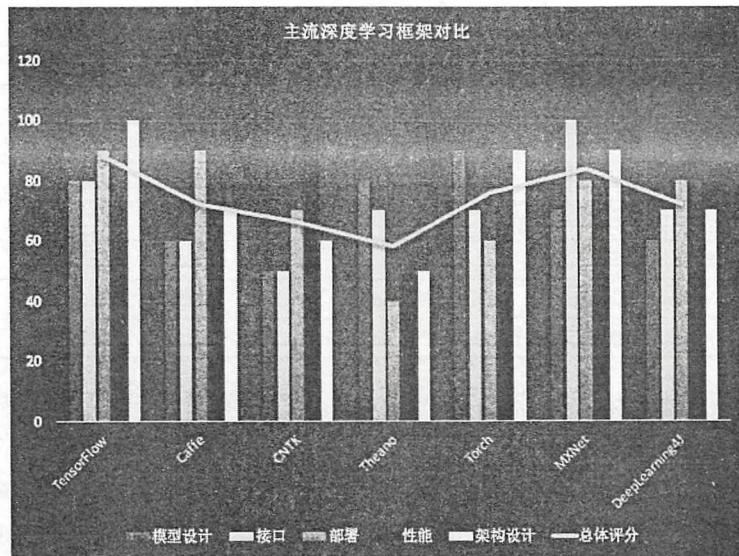


图 2-1 主流深度学习框架对比图

## 2.2 各深度学习框架简介

在本节，我们先来看看目前各流行框架的异同，以及各自的特点和优势。

### 2.2.1 TensorFlow

TensorFlow 是相对高阶的机器学习库，用户可以方便地用它设计神经网络结构，而不必为了追求高效率的实现亲自写 C++ 或 CUDA<sup>18</sup> 代码。它和 Theano 一样都支持自动求导，用户不需要再通过反向传播求解梯度。其核心代码和 Caffe 一样是用 C++ 编写的，使用 C++ 简化了线上部署的复杂度，并让手机这种内存和 CPU 资源都紧张的设备可以运行

复杂模型（Python 则会比较消耗资源，并且执行效率不高）。除了核心代码的 C++ 接口，TensorFlow 还有官方的 Python、Go 和 Java 接口，是通过 SWIG ( Simplified Wrapper and Interface Generator ) 实现的，这样用户就可以在一个硬件配置较好的机器中用 Python 进行实验，并在资源比较紧张的嵌入式环境或需要低延迟的环境中用 C++ 部署模型。SWIG 支持给 C/C++ 代码提供各种语言的接口，因此其他脚本语言的接口未来也可以通过 SWIG 方便地添加。不过使用 Python 时有一个影响效率的问题是，每一个 mini-batch 要从 Python 中 feed 到网络中，这个过程在 mini-batch 的数据量很小或者运算时间很短时，可能会带来影响比较大的延迟。现在 TensorFlow 还有非官方的 Julia、Node.js、R 的接口支持，地址如下。

Julia: [github.com/malmaud/TensorFlow.jl](https://github.com/malmaud/TensorFlow.jl)

Node.js: [github.com/node-tensorflow/node-tensorflow](https://github.com/node-tensorflow/node-tensorflow)

R: [github.com/rstudio/tensorflow](https://github.com/rstudio/tensorflow)

TensorFlow 也有内置的 TF.Learn 和 TF.Slim 等上层组件可以帮助快速地设计新网络，并且兼容 Scikit-learn estimator 接口，可以方便地实现 evaluate、grid search、cross validation 等功能。同时 TensorFlow 不只局限于神经网络，其数据流式图支持非常自由的算法表达，当然也可以轻松实现深度学习以外的机器学习算法。事实上，只要可以将计算表示成计算图的形式，就可以使用 TensorFlow。用户可以写内层循环代码控制计算图分支的计算，TensorFlow 会自动将相关的分支转为子图并执行迭代运算。TensorFlow 也可以将计算图中的各个节点分配到不同的设备执行，充分利用硬件资源。定义新的节点只需要写一个 Python 函数，如果没有对应的底层运算核，那么可能需要写 C++ 或者 CUDA 代码实现运算操作。

在数据并行模式上，TensorFlow 和 Parameter Server 很像，但 TensorFlow 有独立的 Variable node，不像其他框架有一个全局统一的参数服务器，因此参数同步更自由。TensorFlow 和 Spark 的核心都是一个数据计算的流式图，Spark 面向的是大规模的数据，支持 SQL 等操作，而 TensorFlow 主要面向内存足以装载模型参数的环境，这样可以最大化计算效率。

TensorFlow 的另外一个重要特点是它灵活的移植性，可以将同一份代码几乎不经过修改就轻松地部署到有任意数量 CPU 或 GPU 的 PC、服务器或者移动设备上。相比于 Theano，TensorFlow 还有一个优势就是它极快的编译速度，在定义新网络结构时，Theano

通常需要长时间的编译，因此尝试新模型需要比较大的代价，而 TensorFlow 完全没有这个问题。TensorFlow 还有功能强大的可视化组件 TensorBoard，能可视化网络结构和训练过程，对于观察复杂的网络结构和监控长时间、大规模的训练很有帮助。TensorFlow 针对生产环境高度优化，它产品级的高质量代码和设计都可以保证在生产环境中稳定运行，同时一旦 TensorFlow 广泛地被工业界使用，将产生良性循环，成为深度学习领域的事实标准。

除了支持常见的网络结构[卷积神经网络（Convolutional Neural Network, CNN）、循环神经网络（Recurrent Neural Network, RNN）]外，TensorFlow 还支持深度强化学习乃至其他计算密集的科学计算（如偏微分方程求解等）。TensorFlow 此前不支持 symbolic loop，需要使用 Python 循环而无法进行图编译优化，但最近新加入的 XLA 已经开始支持 JIT 和 AOT，另外它使用 bucketing trick 也可以比较高效地实现循环神经网络。TensorFlow 的一个薄弱地方可能在于计算图必须构建为静态图，这让很多计算变得难以实现，尤其是序列预测中经常使用的 beam search。

TensorFlow 的用户能够将训练好的模型方便地部署到多种硬件、操作系统平台上，支持 Intel 和 AMD 的 CPU，通过 CUDA 支持 NVIDIA 的 GPU（最近也开始通过 OpenCL 支持 AMD 的 GPU，但没有 CUDA 成熟），支持 Linux 和 Mac，最近在 0.12 版本中也开始尝试支持 Windows。在工业生产环境中，硬件设备有些是最新款的，有些是用了几年的老机型，来源可能比较复杂，TensorFlow 的异构性让它能够全面地支持各种硬件和操作系统。同时，其在 CPU 上的矩阵运算库使用了 Eigen 而不是 BLAS 库，能够基于 ARM 架构编译和优化，因此在移动设备（Android 和 iOS）上表现得很好。

TensorFlow 在最开始发布时只支持单机，而且只支持 CUDA 6.5 和 cuDNN v2，并且没有官方和其他深度学习框架的对比结果。在 2015 年年底，许多其他框架做了各种性能对比评测，每次 TensorFlow 都会作为较差的对照组出现。那个时期的 TensorFlow 真的不快，性能上仅和普遍认为很慢的 Theano 比肩，在各个框架中可以算是垫底。但是凭借 Google 强大的开发实力，很快支持了新版的 cuDNN（目前支持 cuDNN v5.1），在单 GPU 上的性能追上了其他框架。表 2-3 所示为 <https://github.com/soumith/convnet-benchmarks> 给出的各个框架在 AlexNet 上单 GPU 的性能评测。

表 2-3 各深度学习框架在 AlexNet 上的性能对比

Library (库)	Class (类)	总时间 (ms)	前馈时间 (ms)	反馈时间 (ms)
CuDNN[R4]-fp16 (Torch)	cudnn.SpatialConvolution	71	25	46
Nervana-neon-fp16	ConvLayer	78	25	52
CuDNN[R4]-fp32 (Torch)	cudnn.SpatialConvolution	81	27	53
TensorFlow	conv2d	81	26	55
Nervana-neon-fp32	ConvLayer	87	28	58
fbfft (Torch)	fbnn.SpatialConvolution	104	31	72
Chainer	Convolution2D	177	40	136
cudaconvnet2*	ConvLayer	177	42	135
CuDNN[R2] *	cudnn.SpatialConvolution	231	70	161
Caffe (native)	ConvolutionLayer	324	121	203
Torch-7 (native)	SpatialConvolutionMM	342	132	210
CL-nn (Torch)	SpatialConvolutionMM	963	388	574
Caffe-CLGreenTea	ConvolutionLayer	1442	210	1232

目前在单 GPU 的条件下，绝大多数深度学习框架都依赖于 cuDNN，因此只要硬件计算能力或者内存分配差异不大，最终训练速度不会相差太大。但是对于大规模深度学习来说，巨大的数据量使得单机很难在有限的时间完成训练。这时需要分布式计算使 GPU 集群乃至 TPU 集群并行计算，共同训练出一个模型，所以框架的分布式性能是至关重要的。TensorFlow 在 2016 年 4 月开源了分布式版本，使用 16 块 GPU 可达单 GPU 的 15 倍提速，在 50 块 GPU 时可达到 40 倍提速，分布式的效率很高。目前原生支持的分布式深度学习框架不多，只有 TensorFlow、CNTK、DeepLearning4J、MXNet 等。不过目前 TensorFlow 的设计对不同设备间的通信优化得不是很好，其单机的 reduction 只能用 CPU 处理，分布式的通信使用基于 socket 的 RPC，而不是速度更快的 RDMA，所以其分布式性能可能还没有达到最优。

Google 在 2016 年 2 月开源了 TensorFlow Serving<sup>19</sup>，这个组件可以将 TensorFlow 训练好的模型导出，并部署成可以对外提供预测服务的 RESTful 接口，如图 2-2 所示。有了这个组件，TensorFlow 就可以实现应用机器学习的全流程：从训练模型、调试参数，到打包模型，最后部署服务，名副其实是一个从研究到生产整条流水线都齐备的框架。这里引用 TensorFlow 内部开发人员的描述：“TensorFlow Serving 是一个为生产环境而设计的高性能的机器学习服务系统。它可以同时运行多个大规模深度学习模型，支持模型生命周期管理、算法实验，并可以高效地利用 GPU 资源，让 TensorFlow 训练好的模型更快捷方

便地投入到实际生产环境”。除了 TensorFlow 以外的其他框架都缺少为生产环境部署的考虑，而 Google 作为广泛在实际产品中应用深度学习的巨头可能也意识到了这个机会，因此开发了这个部署服务的平台。TensorFlow Serving 可以说是一副王牌，将会帮 TensorFlow 成为行业标准做出巨大贡献。

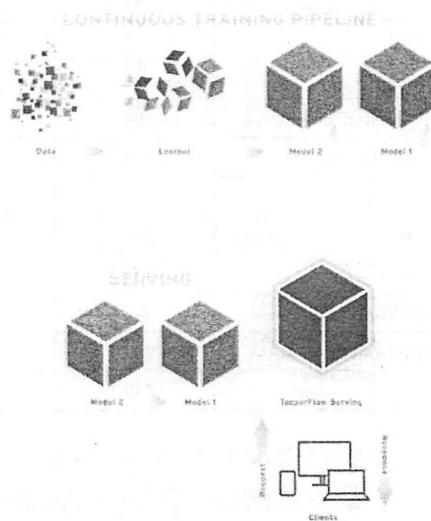


图 2-2 TensorFlow Serving 架构

TensorBoard 是 TensorFlow 的一组 Web 应用，用来监控 TensorFlow 运行过程，或可视化 Computation Graph。TensorBoard 目前支持 5 种可视化：标量( scalars )、图片( images )、音频( audio )、直方图( histograms )和计算图( Computation Graph )。TensorBoard 的 Events Dashboard 可以用来持续地监控运行时的关键指标，比如 loss、学习速率( learning rate )或是验证集上的准确率( accuracy )；Image Dashboard 则可以展示训练过程中用户设定保存的图片，比如某个训练中间结果用 Matplotlib 等绘制( plot )出来的图片；Graph Explorer 则可以完全展示一个 TensorFlow 的计算图，并且支持缩放拖曳和查看节点属性。TensorBoard 的可视化效果如图 2-3 和图 2-4 所示。

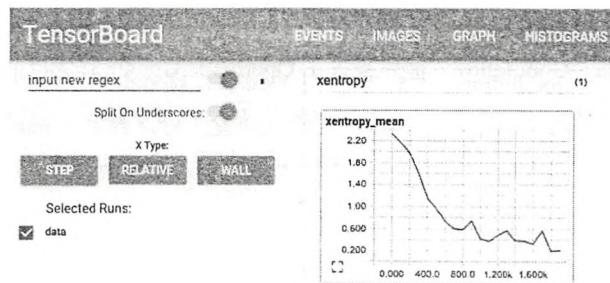


图 2-3 TensorBoard 的 loss 标量的可视化

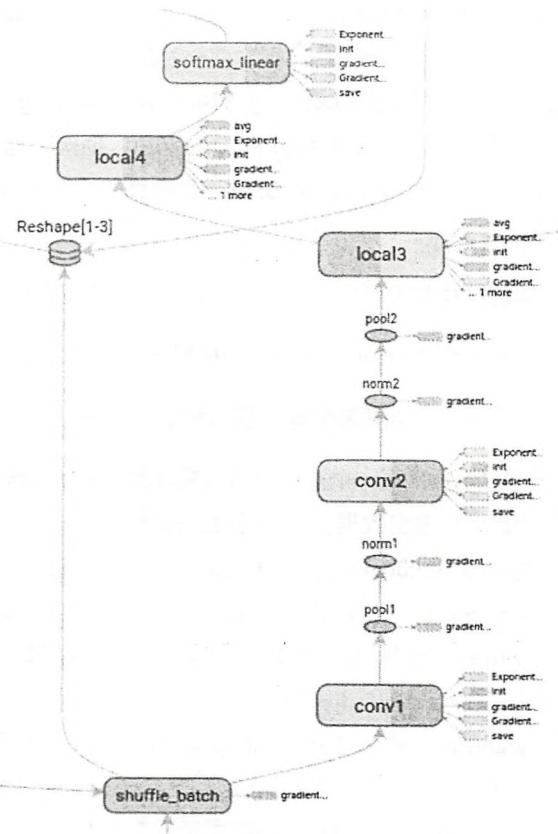


图 2-4 TensorBoard 的模型结构可视化

TensorFlow 拥有产品级的高质量代码，有 Google 强大的开发、维护能力的加持，整体架构设计也非常优秀。相比于同样基于 Python 的老牌对手 Theano，TensorFlow 更成熟、

更完善，同时 Theano 的很多主要开发者都去了 Google 开发 TensorFlow（例如书籍 *Deep Learning* 的作者 Ian Goodfellow，他后来去了 OpenAI）。Google 作为巨头公司有比高校或者个人开发者多得多的资源投入到 TensorFlow 的研发，可以预见，TensorFlow 未来的发展将会是飞速的，可能会把大学或者个人维护的深度学习框架远远甩在身后。

## 2.2.2 Caffe

官方网址：[caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/)

GitHub：[github.com/BVLC/caffe](https://github.com/BVLC/caffe)

Caffe 全称为 Convolutional Architecture for Fast Feature Embedding，是一个被广泛使用的开源深度学习框架（在 TensorFlow 出现之前一直是深度学习领域 GitHub star 最多的项目），目前由伯克利视觉学中心（Berkeley Vision and Learning Center, BVLC）进行维护。Caffe 的创始人是加州大学伯克利的 Ph.D. 贾扬清，他同时也是 TensorFlow 的作者之一，曾工作于 MSRA、NEC 和 Google Brain，目前就职于 Facebook FAIR 实验室。Caffe 的主要优势包括如下几点。

- (1) 容易上手，网络结构都是以配置文件形式定义，不需要用代码设计网络。
- (2) 训练速度快，能够训练 state-of-the-art 的模型与大规模的数据。
- (3) 组件模块化，可以方便地拓展到新的模型和学习任务上。

Caffe 的核心概念是 Layer，每一个神经网络的模块都是一个 Layer。Layer 接收输入数据，同时经过内部计算产生输出数据。设计网络结构时，只需要把各个 Layer 拼接在一起构成完整的网络（通过写 protobuf 配置文件定义）。比如卷积的 Layer，它的输入就是图片的全部像素点，内部进行的操作是各种像素值与 Layer 参数的 convolution 操作，最后输出的是所有卷积核 filter 的结果。每一个 Layer 需要定义两种运算，一种是正向（forward）的运算，即从输入数据计算输出结果，也就是模型的预测过程；另一种是反向（backward）的运算，从输出端的 gradient 求解相对于输入的 gradient，即反向传播算法，这部分也就是模型的训练过程。实现新 Layer 时，需要将正向和反向两种计算过程的函数都实现，这部分计算需要用户自己写 C++ 或者 CUDA（当需要运行在 GPU 时）代码，对普通用户来说还是非常难上手的。正如它的名字 Convolutional Architecture for Fast Feature Embedding 所描述的，Caffe 最开始设计时的目标只针对于图像，没有考虑文本、语音或者时间序列的数据，因此 Caffe 对卷积神经网络的支持非常好，但对时间序列 RNN、LSTM

等支持得不是特别充分。同时，基于 Layer 的模式也对 RNN 不是非常友好，定义 RNN 结构时比较麻烦。在模型结构非常复杂时，可能需要写非常冗长的配置文件才能设计好网络，而且阅读时也比较费力。

Caffe 的一大优势是拥有大量的训练好的经典模型（AlexNet、VGG、Inception）乃至其他 state-of-the-art（ResNet 等）的模型，收藏在它的 Model Zoo ([github.com/BVLC/caffe/wiki/Model-Zoo](https://github.com/BVLC/caffe/wiki/Model-Zoo))。因为知名度较高，Caffe 被广泛地应用于前沿的工业界和学术界，许多提供源码的深度学习的论文都是使用 Caffe 来实现其模型的。在计算机视觉领域 Caffe 应用尤其多，可以用来做人脸识别、图片分类、位置检测、目标追踪等。虽然 Caffe 主要是面向学术圈和研究者的，但它的程序运行非常稳定，代码质量比较高，所以也很适合对稳定性要求严格的生产环境，可以算是第一个主流的工业级深度学习框架。因为 Caffe 的底层是基于 C++ 的，因此可以在各种硬件环境编译并具有良好的移植性，支持 Linux、Mac 和 Windows 系统，也可以编译部署到移动设备系统如 Android 和 iOS 上。和其他主流深度学习库类似，Caffe 也提供了 Python 语言接口 pycaffe，在接触新任务，设计新网络时可以使用其 Python 接口简化操作。不过，通常用户还是使用 Protobuf 配置文件定义神经网络结构，再使用 command line 进行训练或者预测。Caffe 的配置文件是一个 JSON 类型的.prototxt 文件，其中使用许多顺序连接的 Layer 来描述神经网络结构。Caffe 的二进制可执行程序会提取这些.prototxt 文件并按其定义来训练神经网络。理论上，Caffe 的用户可以完全不写代码，只是定义网络结构就可以完成模型训练了。Caffe 完成训练之后，用户可以把模型文件打包制作成简单易用的接口，比如可以封装成 Python 或 MATLAB 的 API。不过在.prototxt 文件内部设计网络节构可能会比较受限，没有像 TensorFlow 或者 Keras 那样在 Python 中设计网络结构方便、自由。更重要的是，Caffe 的配置文件不能用编程的方式调整超参数，也没有提供像 Scikit-learn 那样好用的 estimator 可以方便地进行交叉验证、超参数的 Grid Search 等操作。Caffe 在 GPU 上训练的性能很好( 使用单块 GTX 1080 训练 AlexNet 时一天可以训练上百万张图片 )，但是目前仅支持单机多 GPU 的训练，没有原生支持分布式的训练。庆幸的是，现在有很多第三方的支持，比如雅虎开源的 CaffeOnSpark，可以借助 Spark 的分布式框架实现 Caffe 的大规模分布式训练。

### 2.2.3 Theano

官方网址：<http://wwwdeeplearningnetsoftwaretheano/>

GitHub：[github.com/Theano/Theano](https://github.com/Theano/Theano)

Theano 诞生于 2008 年，由蒙特利尔大学 Lisa Lab 团队开发并维护，是一个高性能的符号计算及深度学习库。因其出现时间早，可以算是这类库的始祖之一，也一度被认为是深度学习研究和应用的重要标准之一。Theano 的核心是一个数学表达式的编译器，专门为处理大规模神经网络训练的计算而设计。它可以将用户定义的各种计算编译为高效的底层代码，并链接各种可以加速的库，比如 BLAS、CUDA 等。Theano 允许用户定义、优化和评估包含多维数组的数学表达式，它支持将计算装载到 GPU（Theano 在 GPU 上性能不错，但是 CPU 上较差）。与 Scikit-learn 一样，Theano 也很好地整合了 NumPy，对 GPU 的透明让 Theano 可以较为方便地进行神经网络设计，而不必直接写 CUDA 代码。Theano 的主要优势如下。

- (1) 集成 NumPy，可以直接使用 NumPy 的 ndarray，API 接口学习成本低。
- (2) 计算稳定性好，比如可以精准地计算输出值很小的函数（像  $\log(1+x)$ ）。
- (3) 动态地生成 C 或者 CUDA 代码，用以编译成高效的机器代码。

因为 Theano 非常流行，有许多人为它编写了高质量的文档和教程，用户可以方便地查找 Theano 的各种 FAQ，比如如何保存模型、如何运行模型等。不过 Theano 更多地被当作一个研究工具，而不是当作产品来使用。虽然 Theano 支持 Linux、Mac 和 Windows，但是没有底层 C++ 的接口，因此模型的部署非常不方便，依赖于各种 Python 库，并且不支持各种移动设备，所以几乎没有在工业生产环境的应用。Theano 在调试时输出的错误信息非常难以看懂，因此 DEBUG 时非常痛苦。同时，Theano 在生产环境使用训练好的模型进行预测时性能比较差，因为预测通常使用服务器 CPU（生产环境服务器一般没有 GPU，而且 GPU 预测单条样本延迟高反而不如 CPU），但是 Theano 在 CPU 上的执行性能比较差。

Theano 在单 GPU 上执行效率不错，性能和其他框架类似。但是运算时需要将用户的 Python 代码转换成 CUDA 代码，再编译为二进制可执行文件，编译复杂模型的时间非常久。此外，Theano 在导入时也比较慢，而且一旦设定了选择某块 GPU，就无法切换到其他设备。目前，Theano 在 CUDA 和 cuDNN 上不支持多 GPU，只在 OpenCL 和 Theano 自己的 gputarray 库上支持多 GPU 训练，速度暂时还比不上 CUDA 的版本，并且 Theano 目前还没有分布式的实现。不过，Theano 在训练简单网络（比如很浅的 MLP）时性能可能比 TensorFlow 好，因为全部代码都是运行时编译，不需要像 TensorFlow 那样每次 feed mini-batch 数据时都得通过低效的 Python 循环来实现。

Theano 是一个完全基于 Python ( C++/CUDA 代码也是打包为 Python 字符串 ) 的符号计算库。用户定义的各种运算，Theano 可以自动求导，省去了完全手工写神经网络反向传播算法的麻烦，也不需要像 Caffe 一样为 Layer 写 C++ 或 CUDA 代码。Theano 对卷积神经网络的支持很好，同时它的符号计算 API 支持循环控制（内部名 scan），让 RNN 的实现非常简单并且高性能，其全面的功能也让 Theano 可以支持大部分 state-of-the-art 的网络。Theano 派生出了大量基于它的深度学习库，包括一系列的上层封装，其中有大名鼎鼎的 Keras，Keras 对神经网络抽象得非常合适，以至于可以随意切换执行计算的后端（目前同时支持 Theano 和 TensorFlow）。Keras 比较适合在探索阶段快速地尝试各种网络结构，组件都是可插拔的模块，只需要将一个个组件（比如卷积层、激活函数等）连接起来，但是设计新模块或者新的 Layer 就不太方便了。除 Keras 外，还有学术界非常喜爱的 Lasagne，同样也是 Theano 的上层封装，它对神经内网络的每一层的定义都非常严谨。另外，还有 scikit-neuralnetwork、nolearn 这两个基于 Lasagne 的上层封装，它们将神经网络抽象为兼容 Scikit-learn 接口的 classifier 和 regressor，这样就可以方便地使用 Scikit-learn 中经典的 fit、transform、score 等操作。除此之外，Theano 的上层封装库还有 blocks、deepy、pyLearn2 和 Scikit-theano，可谓是一个庞大的家族。如果没有 Theano，可能根本不会出现这么多好用的 Python 深度学习库。同样，如果没有 Python 科学计算的基石 NumPy，就不会有 SciPy、Scikit-learn 和 Scikit-image，可以说 Theano 就是深度学习界的 NumPy，是其他各类 Python 深度学习库的基石。虽然 Theano 非常重要，但是直接使用 Theano 设计大型的神经网络还是太繁琐了，用 Theano 实现 Google Inception 就像用 NumPy 实现一个支持向量机（SVM）。且不说很多用户做不到用 Theano 实现一个 Inception 网络，即使能做到但是否有必要花这个时间呢？毕竟不是所有一切都是基础科学工作者，大部分使用场景还是在工业应用中。所以简单易用是一个很重要的特性，这也就是其他上层封装库的价值所在：不需要总是从最基础的 tensor 粒度开始设计网络，而是从更上层的 Layer 粒度设计网络。

## 2.2.4 Torch

官方网址：<http://torch.ch/>

GitHub：[github.com/torch/torch7](https://github.com/torch/torch7)

Torch 给自己的定位是 LuaJIT 上的一个高效的科学计算库，支持大量的机器学习算法，同时以 GPU 上的计算优先。Torch 的历史非常悠久，但真正得到发扬光大是在 Facebook 开源了其深度学习的组件之后，此后包括 Google、Twitter、NYU、IDIAP、Purdue 等组织

都大量使用 Torch。Torch 的目标是让设计科学计算算法变得便捷，它包含了大量的机器学习、计算机视觉、信号处理、并行运算、图像、视频、音频、网络处理的库，同时和 Caffe 类似，Torch 拥有大量的训练好的深度学习模型。它可以支持设计非常复杂的神经网络的拓扑图结构，再并行化到 CPU 和 GPU 上，在 Torch 上设计新的 Layer 是相对简单的。它和 TensorFlow 一样使用了底层 C++ 加上层脚本语言调用的方式，只不过 Torch 使用的是 Lua。Lua 的性能是非常优秀的（该语言经常被用来开发游戏），常见的代码可以通过透明的 JIT 优化达到 C 的性能的 80%；在便利性上，Lua 的语法也非常简单易读，拥有漂亮和统一的结构，易于掌握，比写 C/C++ 简洁很多；同时，Lua 拥有一个非常直接的调用 C 程序的接口，可以简便地使用大量基于 C 的库，因为底层核心是 C 写的，因此也可以方便地移植到各种环境。Lua 支持 Linux、Mac，还支持各种嵌入式系统（iOS、Android、FPGA 等），只不过运行时还是必须有 LuaJIT 的环境，所以工业生产环境的使用相对较少，没有 Caffe 和 TensorFlow 那么多。

为什么不简单地使用 Python 而是使用 LuaJIT 呢？官方给出了以下几点理由。

（1）LuaJIT 的通用计算性能远胜于 Python，而且可以直接在 LuaJIT 中操作 C 的 pointers。

（2）Torch 的框架，包含 Lua 是自洽的，而完全基于 Python 的程序对不同平台、系统移植性较差，依赖的外部库较多。

（3）LuaJIT 的 FFI 拓展接口非常易学，可以方便地链接其他库到 Torch 中。Torch 中还专门设计了 N-Dimension array type 的对象 Tensor，Torch 中的 Tensor 是一块内存的视图，同时一块内存可能有许多视图（Tensor）指向它，这样的设计同时兼顾了性能（直接面向内存）和便利性。同时，Torch 还提供了不少相关的库，包括线性代数、卷积、傅里叶变换、绘图和统计等，如图 2-5 所示。

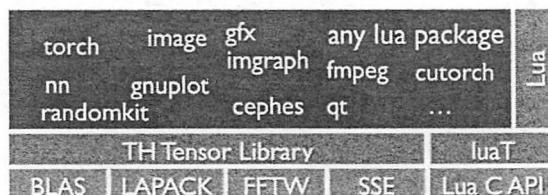


图 2-5 Torch 提供的各种数据处理的库

Torch 的 nn 库支持神经网络、自编码器、线性回归、卷积网络、循环神经网络等，同时支持定制的损失函数及梯度计算。Torch 因为使用了 LuaJIT，因此用户在 Lua 中做数

据预处理等操作可以随意使用循环等操作，而不必像在 Python 中那样担心性能问题，也不需要学习 Python 中各种加速运算的库。不过，Lua 相比 Python 还不是那么主流，对大多数用户有学习成本。Torch 在 CPU 上的计算会使用 OpenMP、SSE 进行优化，GPU 上使用 CUDA、cutorch、cunn、cuDNN 进行优化，同时还有 cuda-convnet 的 wrapper。Torch 有很多第三方的扩展可以支持 RNN，使得 Torch 基本支持所有主流的网络。和 Caffe 类似的是，Torch 也是主要基于 Layer 的连接来定义网络的。Torch 中新的 Layer 依然需要用户自己实现，不过定义新 Layer 和定义网络的方式很相似，非常简便，不像 Caffe 那么麻烦，用户需要使用 C++ 或者 CUDA 定义新 Layer。同时，Torch 属于命令式编程模式，不像 Theano、TensorFlow 属于声明性编程（计算图是预定义的静态的结构），所以用它实现某些复杂操作（比如 beam search）比 Theano 和 TensorFlow 方便很多。

### 2.2.5 Lasagne

官网网址：<http://lasagne.readthedocs.io/>

GitHub：[github.com/Lasagne/Lasagne](https://github.com/Lasagne/Lasagne)

Lasagne 是一个基于 Theano 的轻量级的神经网络库。它支持前馈神经网络，比如卷积网络、循环神经网络、LSTM 等，以及它们的组合；支持许多优化方法，比如 Nesterov momentum、RMSprop、ADAM 等；它是 Theano 的上层封装，但又不像 Keras 那样进行了重度的封装，Keras 隐藏了 Theano 中所有的方法和对象，而 Lasagne 则是借用了 Theano 中很多的类，算是介于基础的 Theano 和高度抽象的 Keras 之间的一个轻度封装，简化了操作同时支持比较底层的操作。Lasagne 设计的六个原则是简洁、透明、模块化、实用、聚焦和专注。

### 2.2.6 Keras

官方网址：[keras.io](https://keras.io)

GitHub：[github.com/fchollet/keras](https://github.com/fchollet/keras)

Keras 是一个崇尚极简、高度模块化的神经网络库，使用 Python 实现，并可以同时运行在 TensorFlow 和 Theano 上。它旨在让用户进行最快速的原型实验，让想法变为结果的这个过程最短。Theano 和 TensorFlow 的计算图支持更通用的计算，而 Keras 则专精于深度学习。Theano 和 TensorFlow 更像是深度学习领域的 NumPy，而 Keras 则是这个领域的

Scikit-learn。它提供了目前为止最方便的 API，用户只需要将高级的模块拼在一起，就可以设计神经网络，它大大降低了编程开销（code overhead）和阅读别人代码时的理解开销（cognitive overhead）。它同时支持卷积网络和循环网络，支持级联的模型或任意的图结构的模型（可以让某些数据跳过某些 Layer 和后面的 Layer 对接，使得创建 Inception 等复杂网络变得容易），从 CPU 上计算切换到 GPU 加速无须任何代码的改动。因为底层使用 Theano 或 TensorFlow，用 Keras 训练模型相比于前两者基本没有什么性能损耗（还可以享受前两者持续开发带来的性能提升），只是简化了编程的复杂度，节约了尝试新网络结构的时间。可以说模型越复杂，使用 Keras 的收益就越大，尤其是在高度依赖权值共享、多模型组合、多任务学习等模型上，Keras 表现得非常突出。Keras 所有的模块都是简洁、易懂、完全可配置、可随意插拔的，并且基本上没有任何使用限制，神经网络、损失函数、优化器、初始化方法、激活函数和正则化等模块都是可以自由组合的。Keras 也包括绝大部分 state-of-the-art 的 Trick，包括 Adam、RMSProp、Batch Normalization、PReLU、ELU、LeakyReLU 等。同时，新的模块也很容易添加，这让 Keras 非常适合最前沿的研究。Keras 中的模型也都是在 Python 中定义的，不像 Caffe、CNTK 等需要额外的文件来定义模型，这样就可以通过编程的方式调试模型结构和各种超参数。在 Keras 中，只需要几行代码就能实现一个 MLP，或者十几行代码实现一个 AlexNet，这在其他深度学习框架中基本是不可能完成的任务。Keras 最大的问题可能是目前无法直接使用多 GPU，所以对大规模的数据处理速度没有其他支持多 GPU 和分布式的框架快。Keras 的编程模型设计和 Torch 很像，但是相比 Torch，Keras 构建在 Python 上，有一套完整的科学计算工具链，而 Torch 的编程语言 Lua 并没有这样一条科学计算工具链。无论从社区人数，还是活跃度来看，Keras 目前的增长速度都已经远远超过了 Torch。

### 2.2.7 MXNet

官网网址：[mxnet.io](http://mxnet.io)

GitHub：[github.com/dmlc/mxnet](https://github.com/dmlc/mxnet)

MXNet 是 DMLC (Distributed Machine Learning Community) 开发的一款开源的、轻量级、可移植的、灵活的深度学习库，它让用户可以混合使用符号编程模式和指令式编程模式来最大化效率和灵活性，目前已经是 AWS 官方推荐的深度学习框架。MXNet 的很多作者都是中国人，其最大的贡献组织为百度，同时很多作者来自 cxxnet、minerva 和 purine2 等深度学习项目，可谓博采众家之长。它是各个框架中率先支持多 GPU 和分布式

的，同时其分布式性能也非常高。MXNet 的核心是一个动态的依赖调度器，支持自动将计算任务并行化到多个 GPU 或分布式集群（支持 AWS、Azure、Yarn 等）。它上层的计算图优化算法可以让符号计算执行得非常快，而且节约内存，开启 mirror 模式会更加省内存，甚至可以在某些小内存 GPU 上训练其他框架因显存不够而训练不了的深度学习模型，也可以在移动设备（Android、iOS）上运行基于深度学习的图像识别等任务。此外，MXNet 的一个很大的优点是支持非常多的语言封装，比如 C++、Python、R、Julia、Scala、Go、MATLAB 和 JavaScript 等，可谓非常全面，基本主流的脚本语言全部都支持了。在 MXNet 中构建一个网络需要的时间可能比 Keras、Torch 这类高度封装的框架要长，但是比直接用 Theano 等要快。MXNet 的各级系统架构（下面为硬件及操作系统底层，逐层向上为越来越抽象的接口）如图 2-6 所示。

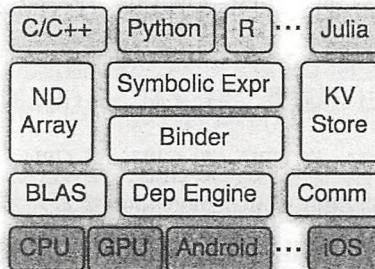


图 2-6 MXNet 系统架构

### 2.2.8 DIGITS

官方网址：[developer.nvidia.com/digits](http://developer.nvidia.com/digits)

GitHub：[github.com/NVIDIA/DIGITS](https://github.com/NVIDIA/DIGITS)

DIGITS（Deep Learning GPU Training System）不是一个标准的深度学习库，它可以算是一个 Caffe 的高级封装（或者 Caffe 的 Web 版培训系统）。因为封装得非常重，以至于你不需要（也不能）在 DIGITS 中写代码，即可实现一个深度学习的图片识别模型。在 Caffe 中，定义模型结构、预处理数据、进行训练并监控训练过程是相对比较烦琐的，DIGITS 把所有这些操作都简化为在浏览器中执行。它可以算作 Caffe 在图片分类上的一个漂亮的用户可视化界面（GUI），计算机视觉的研究者或者工程师可以非常方便地设计深度学习模型、测试准确率，以及调试各种超参数。同时使用它也可以生成数据和训练结果的可视化统计报表，甚至是网络的可视化结构图。训练好的 Caffe 模型可以被 DIGITS

直接使用，上传图片到服务器或者输入 url 即可对图片进行分类。

## 2.2.9 CNTK

官方网址: [cntk.ai](http://cntk.ai)

GitHub: [github.com/Microsoft/CNTK](https://github.com/Microsoft/CNTK)

CNTK (Computational Network Toolkit) 是微软研究院 (MSR) 开源的深度学习框架。它最早由 *start the deep learning craze* 的演讲人创建，目前已经发展成一个通用的、跨平台的深度学习系统，在语音识别领域的使用尤其广泛。CNTK 通过一个有向图将神经网络描述为一系列的运算操作，这个有向图中子节点代表输入或网络参数，其他节点代表各种矩阵运算。CNTK 支持各种前馈网络，包括 MLP、CNN、RNN、LSTM、Sequence-to-Sequence 模型等，也支持自动求解梯度。CNTK 有丰富的细粒度的神经网络组件，使得用户不需要写底层的 C++ 或 CUDA，就能通过组合这些组件设计新的复杂的 Layer。CNTK 拥有产品级的代码质量，支持多机、多 GPU 的分布式训练。

CNTK 设计是性能导向的，在 CPU、单 GPU、多 GPU，以及 GPU 集群上都有非常优异的表现。同时微软最近推出的 1-bit compression 技术大大降低了通信代价，让大规模并行训练拥有了很高的效率。CNTK 同时宣称拥有很高的灵活度，它和 Caffe 一样通过配置文件定义网络结构，再通过命令行程序执行训练，支持构建任意的计算图，支持 AdaGrad、RmsProp 等优化方法。它的另一个重要特性就是拓展性，CNTK 除了内置的大量运算核，还允许用户定义他们自己的计算节点，支持高度的定制化。CNTK 在 2016 年 9 月发布了对强化学习的支持，同时，除了通过写配置文件的方式定义网络结构，CNTK 还将支持其他语言的绑定，包括 Python、C++ 和 C#，这样用户就可以用编程的方式设计网络结构。CNTK 与 Caffe 一样也基于 C++ 并且跨平台，大部分情况下，它的部署非常简单。PC 上支持 Linux、Mac 和 Windows，但是它目前不支持 ARM 架构，限制了其在移动设备上的发挥。图 2-7 所示为 CNTK 目前的总体架构图。

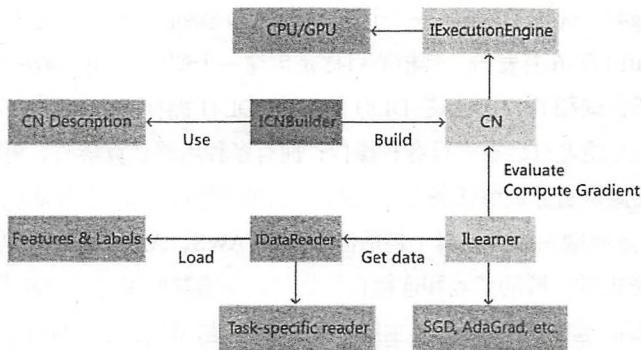


图 2-7 CNTK 的总体架构图

CNTK 原生支持多 GPU 和分布式，从官网公布的对比评测来看，性能非常不错。在多 GPU 方面，CNTK 相对于其他的深度学习库表现得更突出，它实现了 1-bit SGD 和自适应的 mini-batching。图 2-8 所示为 CNTK 官网公布的在 2015 年 12 月的各个框架的性能对比。在当时，CNTK 是唯一支持单机 8 块 GPU 的框架，并且在分布式系统中可以超越 8 块 GPU 的性能。

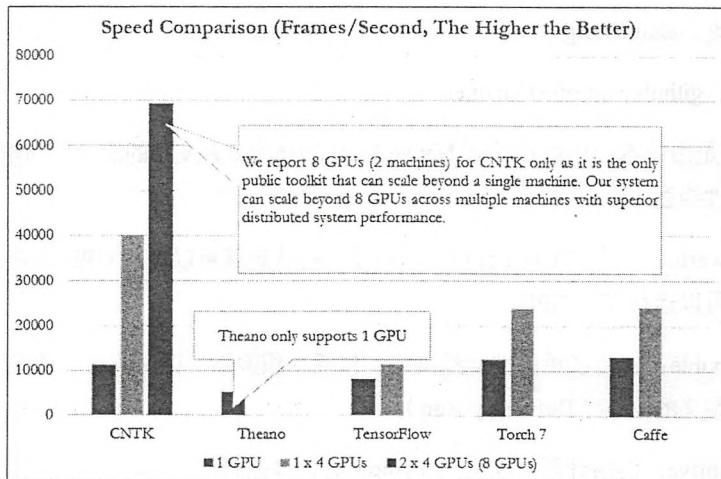


图 2-8 CNTK 与各个框架的性能对比

### 2.2.10 Deeplearning4J

官方网址：<http://deeplearning4j.org/>

GitHub：[github.com/deeplearning4j/deeplearning4j](https://github.com/deeplearning4j/deeplearning4j)