9 种深度学习框架的综评

Jiechao Cheng
Nov. 28th, 2016

本文对目前比较热门的几种机器学习框架 (Caffe、CNTK、GraphLab、GraphX、MXNet、Petuum、TensorFlow、Theano、Torch) ，分别从网络支持、系统架构、调度方式、计算方式、一致性协议、部署模式、接口、并行性、性能、容错、代码复用性、短板、总体概述等方面展开描述，并进行主观评分 (满分 5 pts)。

1.网络支持

Caffe (3.5 pts)

Caffe 可能是第一个主流的工业级深度学习工具，它开始于 2013 年底,具有出色的卷积神经网络实现。在计算机视觉领域 Caffe 依然是最流行的工具包，它有很多扩展，但是由于一些遗留的架构问题，它对递归网络和语言建模的支持很差。此外，在 Caffe 中图层需要使用 C++定义，而网络则使用 Protobuf 定义。

CNTK (4 pts)

CNTK 由深度学习热潮的发起演讲人创建，目前已经发展成一个通用的、平台独立的深度学习系统。在 CNTK (类似 TensorFlow 和 Theano) 中，网络会被指定为向量运算的符号图，运算的组合会形成层。CNTK 通过细粒度的构件块让用户不需要使用低层次语言（类似 Caffe）就能创建新的、复杂的层类型。

GraphLab (4 pts)

GraphLab 的开发基于最新的架构包括 Convolution Layer、Max、Sum、Average Pooling 和 Dropout。用户可以利用 API 开发定制化的神经网络，以及一些应用包括图像分类、对象侦测和图像类比；支持的深度学习网络有 CNN、RNN 等。最新发布的 GraphLab Create 1.1 及 1.2 使深度学习变得超级简单。你不需要在选择模型和调参上成为行家，就可以玩转神经网络；基于输入数据，相关函数选择一个网络架构并设置合理的参数值。

GraphX (3.5 pts)

GraphX 支持的神经网络目前主要是 CNN。GraphX 是从表到图、允许图与表的交互，融合了图并行以及数据并行的优势，是 Spark 中用于图 (如社交网络) 和图并行计算 (如 PageRank、协同过滤) 的 API，GraphX 可以认为是 GraphLab (C++) 和 Pregel (C++) 在 Spark (Scala) 上的重写及优化。

MXNet (4.5 pts)

MXNet 是多功能的机器学习(ML)库，促进了 ML 算法的发展，特别是深神经网络。MXNet 嵌入在宿主语言，混合了声明式符号表达式和命令式张量计算。它提供自动获得渐变分化。MXNet 是计算和记忆高效的框架，可以在各种异构系统上运行，从移动设备到分布式 GPU 集群。

Petuum (5 pts)

Petuum 已被应用于解决具体的工业界问题，在大规模硬件系统上得到了部署，有基于深度学习的图像分析和网络分析应用，支持大部分先进的网络。Petuum 平台有两个主要组成部分：分布式键值存储系统 Bösen 和动态参数更新调度器 Strads。Bösen 使用的一致性协议是受限异步，既可以获得与完全异步相似的性能，又可以保证机器学习算法的近似正确性。Strads 对机器学习模型参数的更新进行细粒度调度，根据参数的优先级自动调整更新次序，并根据参数的相关性防止不安全的并行。

TensorFlow (4.5 pts)

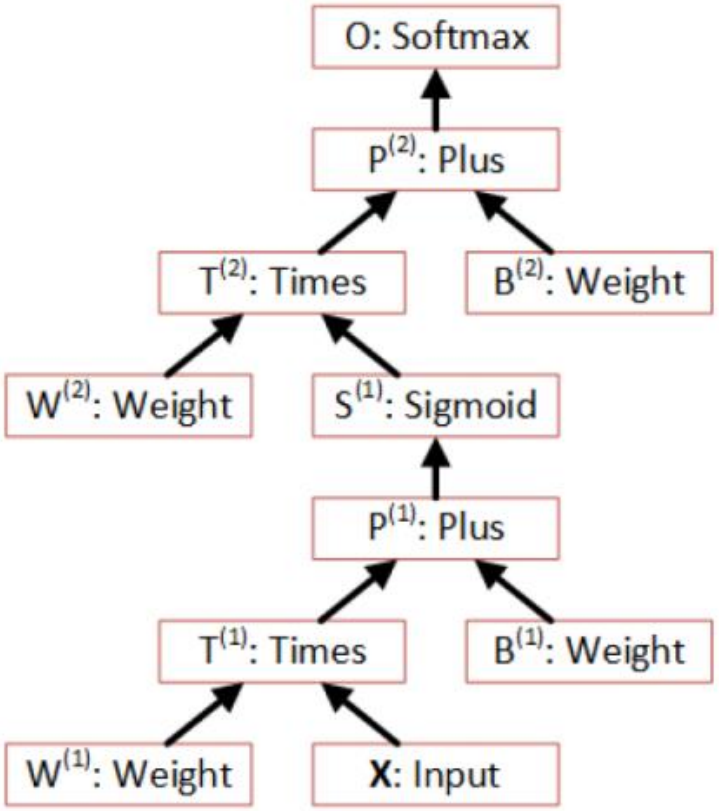
TensorFlow 是一个理想的 RNN (递归神经网络) API 和实现，TensorFlow 使用了向量运算的符号图方法，使得新网络的指定变得相当容易，但 TensorFlow 并不支持双向 RNN 和 3D 卷积，同时公共版本的图定义也不支持循环和条件控制，这使得 RNN 的实现并不理想，因为必须要使用 Python 循环且无法进行图编译优化。TensorFlow 建模的灵活性比较差，每个计算流必须构建成静态的图，所以导致一些计算效果差，比如柱型搜索 (常用于序列评测)。

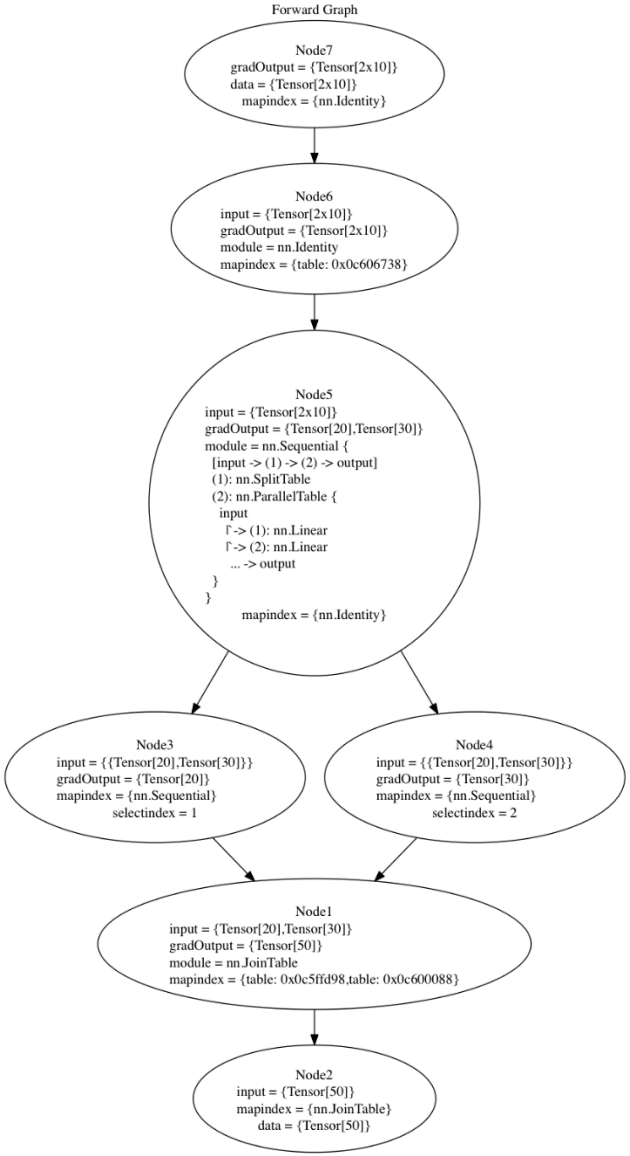
Theano (4.5 pts)

Theano 支持大部分先进的网络，支持高水平的框架。它引领了符号图在编程网络中使用的趋势。Theano 的符号 API 支持循环控制，让 RNN 的实现更加容易且高效。

Torch (5 pts)

Torch 对卷积网络的支持非常好。在 TensorFlow 和 Theano 中时域卷积可以通过 conv2d 来实现，但这样做有点取巧；Torch 通过时域卷积的本地接口使得它的使用非常直观。Torch 通过很多非官方的扩展支持大量的 RNN，同时网络的定义方法也有很多种。但 Torch 本质上是以图层的方式定义网络的，这种粗粒度的方式使得它对新图层类型的扩展不够灵活，需要用户自行实现完整的向前、向后和渐变输入更新。与 Caffe 相比，在 Torch 中定义新图层非常容易，不需要使用 C++编程，而且新图层和网络定义方式之间的区别最小；但 Caffe 中图形由 C++定义，而网络通过 Protobuf 定义。另外，Torch 由于是命令式，比 TensorFlow 和 Theano (声明式) 更灵活，因此，这也让 Torch 中的一些操作 (如柱型搜索) 更容易。


O: Softmax
P(2): Plus
T(2): Times
B(2): Weight
W(2): Weight
S(1): Sigmoid
P(1): Plus
T(1): Times
B(1): Weight
W(1): Weight
X: Input

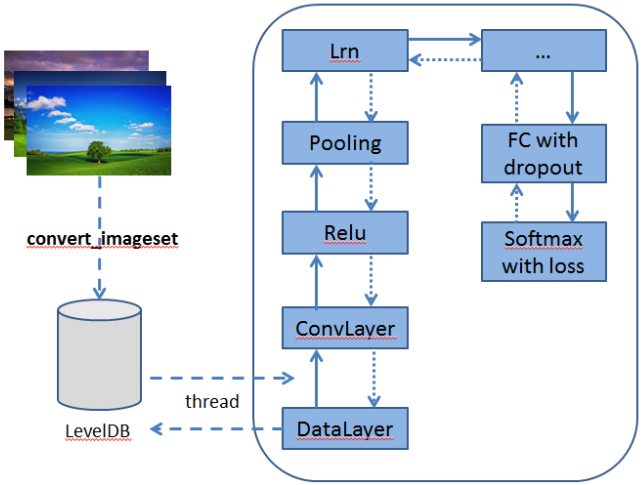

Forward Graph
Node7
gradOutput = {Tensor[2x10]}
data = {Tensor[2x10]}
mapindex = {nn.Identity}
Node6
input = {Tensor[2x10]}
gradOutput = {Tensor[2x10]}
module = nn.Identity
mapindex = {table: 0x0c606738}
Node5
input = {Tensor[2x10]}
gradOutput = {Tensor[20],Tensor[30]}
module = nn.Sequential {
[input -> (1) -> (2) -> output]
(1): nn.SplitTable
(2): nn.ParallelTable {
input
|`-> (1): nn.Linear
|`-> (2): nn.Linear
... -> output
}
}
mapindex = {nn.Identity}
Node3
input = {{Tensor[20],Tensor[30]}}
gradOutput = {Tensor[20]}
mapindex = {nn.Sequential}
selectindex = 1
Node4
input = {{Tensor[20],Tensor[30]}}
gradOutput = {Tensor[30]}
mapindex = {nn.Sequential}
selectindex = 2
Node1
input = {Tensor[20],Tensor[30]}
gradOutput = {Tensor[50]}
module = nn.JoinTable
mapindex = {table: 0x0c5ffd98,table: 0x0c600088}
Node2
input = {Tensor[50]}
mapindex = {nn.JoinTable}
data = {Tensor[50]}

上图：CNTK/Theano/TensorFlow 图模型，下图：Caffe/Torch 图模型

2.系统架构

Caffe (3.5 pts)

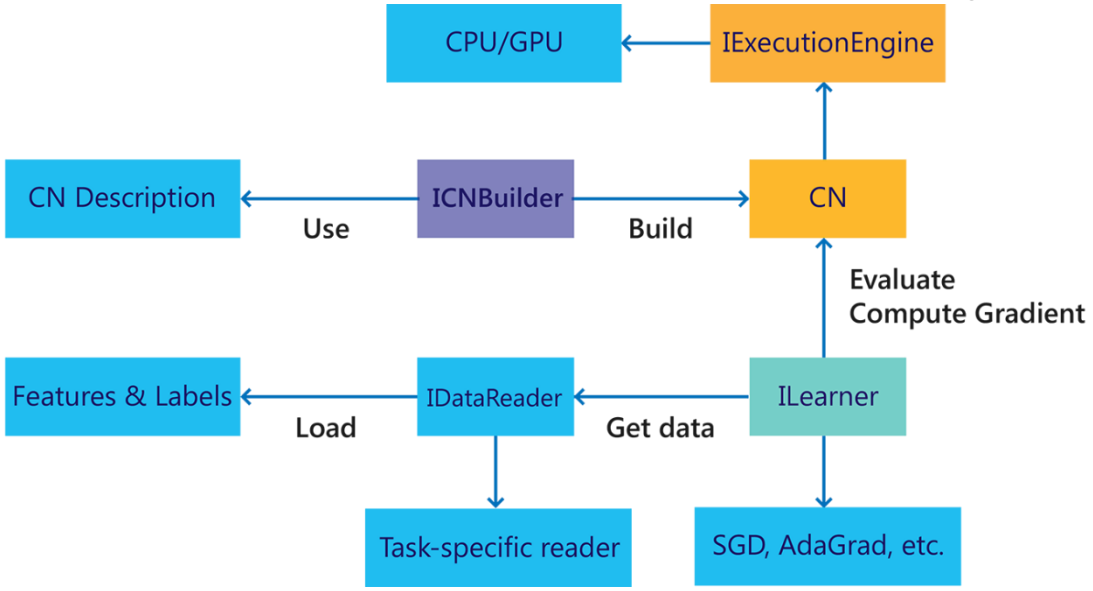
Caffe 的架构在现在看来算是平均水准，它的主要痛点是使用 C++进行分层设计，而模型需要使用 protobuf 接口定义。另外，如果想要支持 CPU 和 GPU，用户还必须实现额外的函数，例如 Forward_GPU 和 Backward_GPU；对于自定义的层类型，还必须为其分配一个 int 类型的 id，并将其添加到 proto 文件中。Caffe 有 pycaffe 接口，但仅仅是命令行接口的替换，模型还是由 protobuf 定义。Caffe 适合工业界和大规模互联网媒体需求的 CUDA GPU 计算，在单个 K40 或 Titan GPU 上每天处理超过 4000 万幅图像 (≈2.5 毫秒/张)。Caffe 通过从实际执行中将模型表示分离出来，允许实验和多平台之间无缝切换，便于从成型机的开发和部署移植到云环境。它在研究项目，大规模工业应用，视觉、语音和多媒体启动原型中影响很大。



Caffe 系统架构图

CNTK (5 pts)

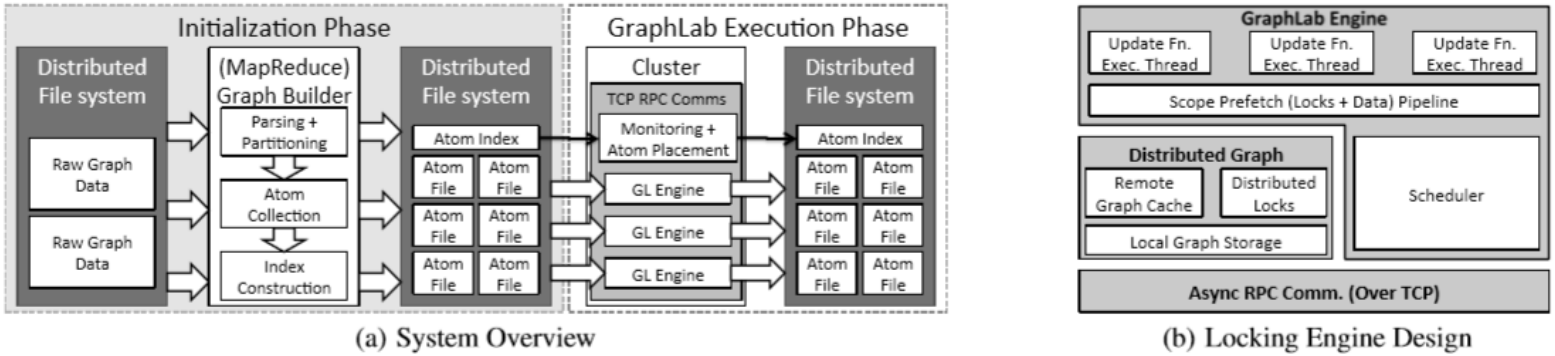
CNTK 通过细粒度的构件块让用户不需要使用低层次的语言就能创建新的、复杂的层类型。同时它实现了跨多 GPU 和服务器自动分化和并行化的随机梯度下降 (SGD，误差反向传播) 学习。架构图如下所示，最左边是输入的原始数据， CN Description 是网络描述，可以理解为给予使用样本用户的指导性意见，而 Features&Labels 中的 Features 就是样本，Labels 可以理解为样本上标明真假的标签。中间的是 ICNBuilder 是学习行为的驱使，IDataReader 是仔细观察并且检验所学内容的行为的驱使。最右边的 ILeaner 是学习方法，CN 是学习的成果，IExecutionEngine 是最终拿真正东西进行检验的行为。



CNTK 系统架构图

GraphLab (4 pts)

图 a 提供了一个高级概述的 GraphLab 系统。用户通过构建原子图表示分布式文件系统 (DFS) 开始。如果使用散列分区，构建过程是通过每个顶点和边表现映射的 Map-Reduce 过程，并且每一个 reducer 积累原子文件。这种原子日志格式允许未来的变化附加到图而不必重新处理所有数据。图 b 提供了一个高级概述的 GraphLab 锁定引擎实现过程。GraphLab 在群集上启动时，每台机器上执行 GraphLab 程序的一个实例。GraphLab 处理过程是对称的，每台机器互相直接使用自定义异步 RPC 协议通过 TCP/IP 沟通，第一进程的额外责任是成为主机器或监视机器。在主程序启动时，基于原子指数计算原子的位置，随后所有进程被分配执行原子的并行加载。每个进程负责在本地图存储内管理分布式图的分区，并提供分布式锁，缓存是用来提供对远程图形数据的访问。

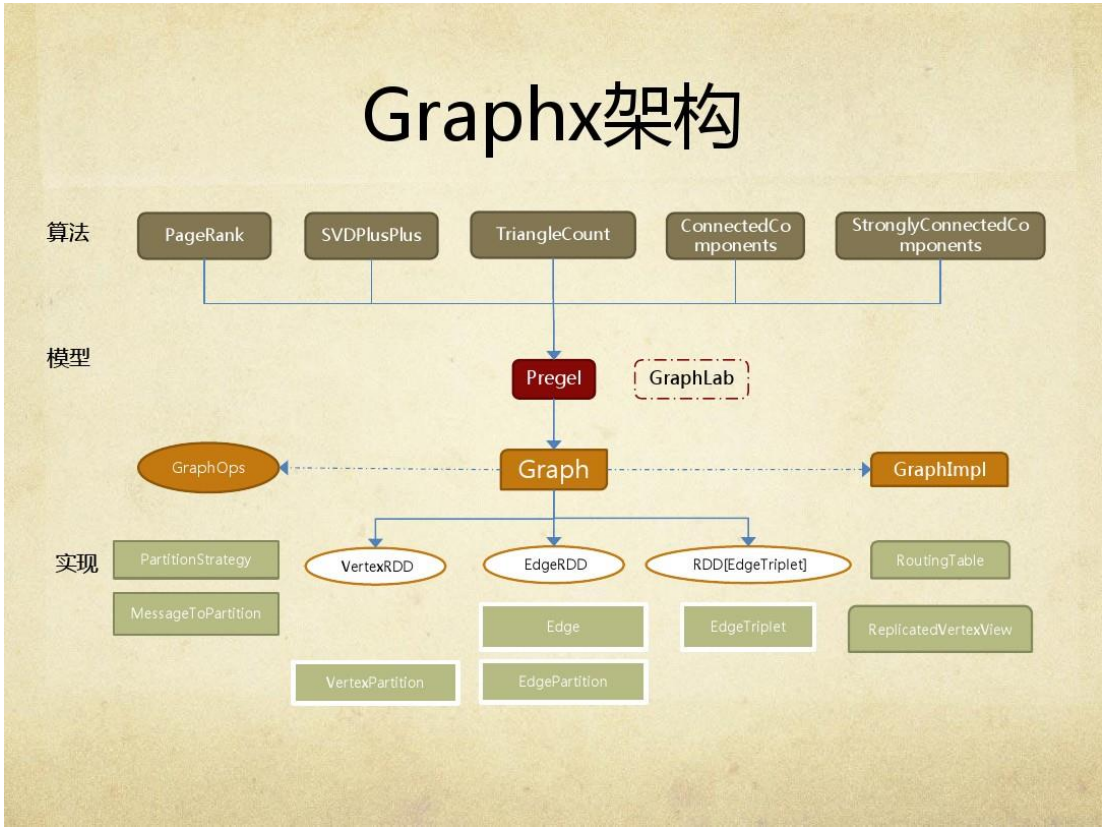


GraphLab 系统架构图

GraphX (4 pts)

GraphX 代表了熟悉的组合图形抽象，足以表达现有的图形 API，但只能使用较少的基本数据流运算符 (例如，加入、映射、分组)。为在专业图形系统中实现高性能奇偶校验，GraphX 重写了具体的图优化，分布式联接优化和实例化视图维护。GraphX 项目的目的就是将 graph-parallel 和 data-parallel 统

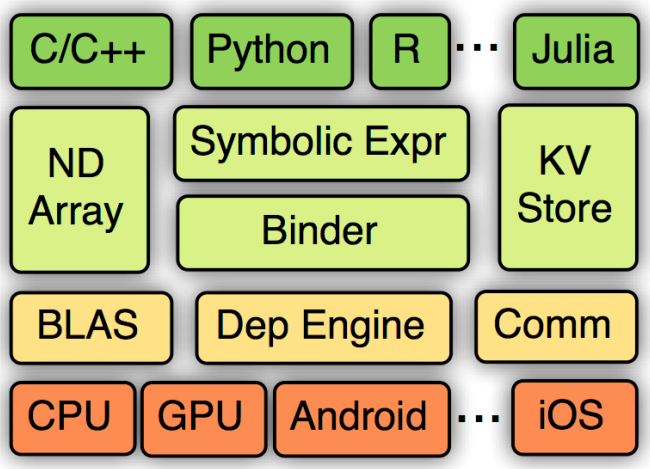
一到一个系统中，这个系统拥有一个唯一的组合 API。GraphX 允许用户将数据当做一个图和一个集合 (RDD)，而不需要数据移动或者复制。通过将最新的进展整合进 graph-parallel 系统，GraphX 能够优化图操作的执行。



GraphX 系统架构图

MXNet (5 pts)

MXNet 允许使用者将符号编程和命令式编程相结合，以追求效率和生产力的最大化。在命令式编程上 MXNet 提供张量运算，而声明式编程中 MXNet 支持符号表达式。用户可以自由的混合它们来快速实现自己的想法。相关架构图如下所示，从上到下分别为各种主语言的嵌入，编程接口 (矩阵运算、符号表达式、分布式通讯)，两种编程模式的统一系统实现，以及各硬件的支持。

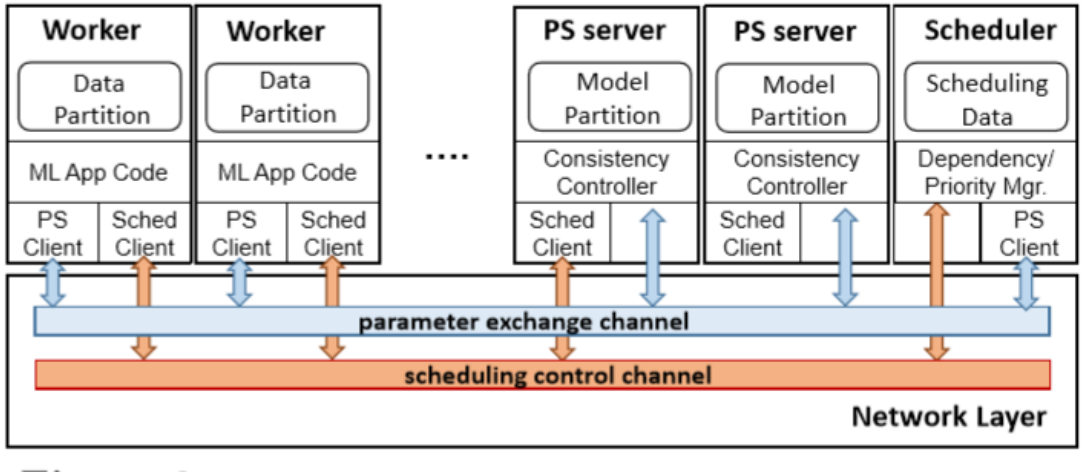


MXNet 系统架构图

Petuum (4 pts)

如图所示，Petuum 系统有三个部分组成：scheduler、workers、parameter server。Petuum 平台由基于受限异步一致性协议的分布式键值存储系统 Bösen (数据并行) 和动态参数更新调度器 Strads (模型并行)两个主要部分组成。与其他的参数服务器并没有大的差别，但模块化设计更加良好，比如一致性模

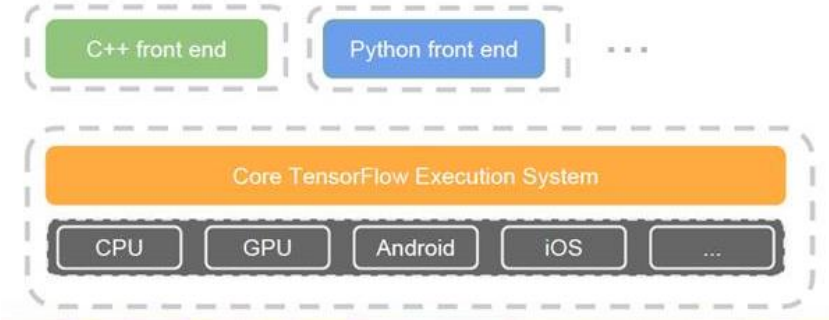
型，调度这些重要功能都放入单独组件。比较遗憾的是 Petuum/Bosen 也没有在容错设计上有所考虑，这跟 Eric Xing 宣称的原型系统也相吻合，因此跟 Paracel 类似，Bosen 目前主要适用于几十台机器的集群，在更大集群上处理有风险。



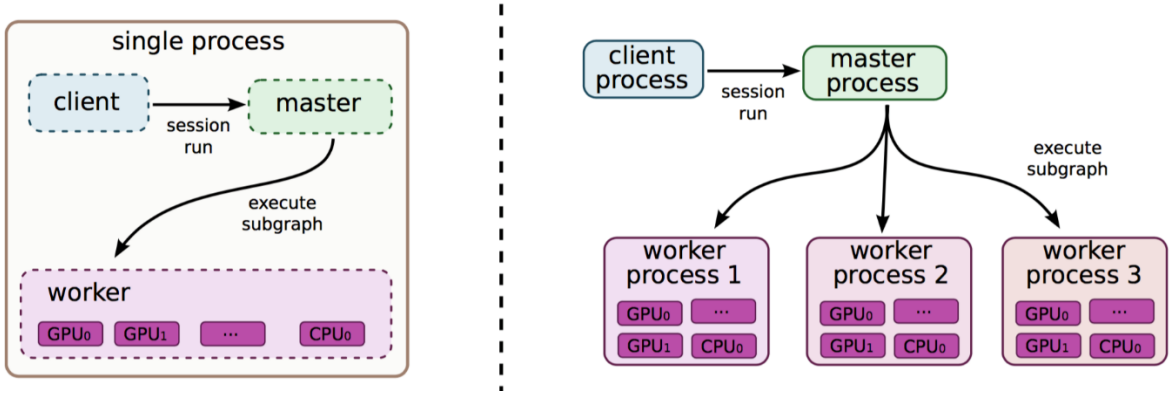
Petuum 系统架构图

TensorFlow (5 pts)

TensorFlow 的架构清晰，采用了模块化设计，支持多种前端和执行平台。TensorFlow 使用数据流图来表示计算，共享状态，并改变状态等操作。在跨越许多机器的一个集群中、跨多个计算设备的一个机器内，包括多核 CPUs，通用 GPUs 和称为张量处理单元 (TPUs) 的定制设计 ASICs 中，TF 映射数据流图的节点。TF 的实现分为单机实现和分布式实现：在单机实现中，构建好图后，使用拓扑算法来决定执行哪一个节点，即对每个节点使用一个计数，值表示所依赖的未完成的节点数目；当一个节点的运算完成时，将依赖该节点的所有节点的计数减一；如果节点的计数为 0，将其放入准备队列待执行。在分布式实现中，需要实现的是对 client，master，worker process 不在同一台机器上时的支持。此时，关于这些进程的调度，使用的是原始论文中参考文献 51 的调度方式。关于分布式和单机的不同，如下图所示。



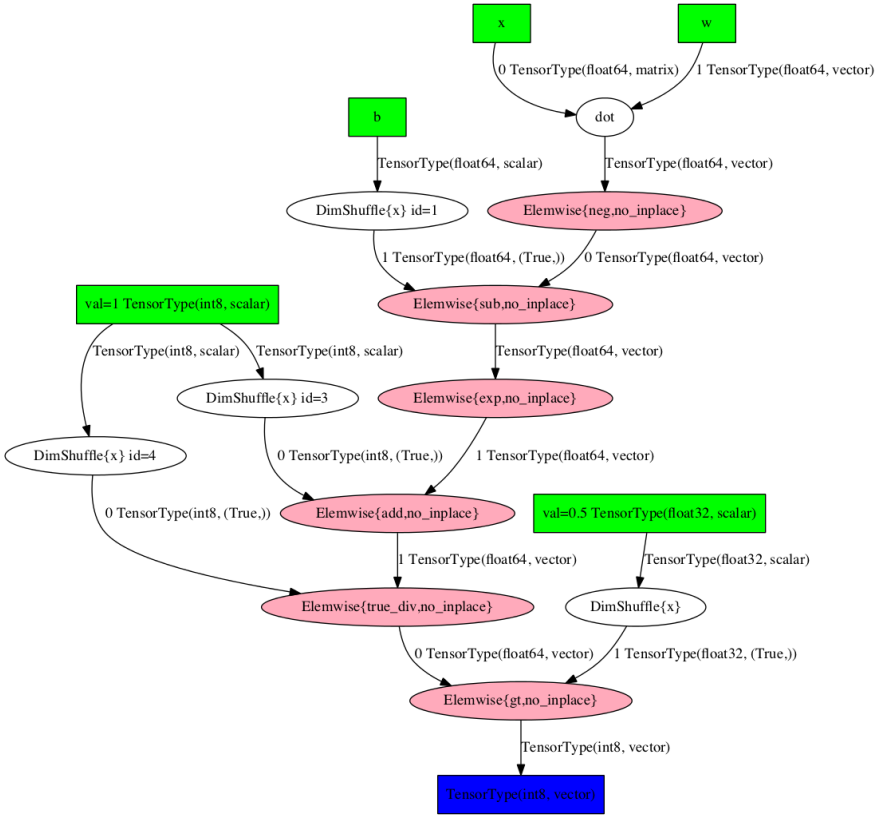
TensorFlow 系统架构图



TensorFlow 单机实现 (左图) 和分布式实现 (右图)

Theano (3.5 pts)

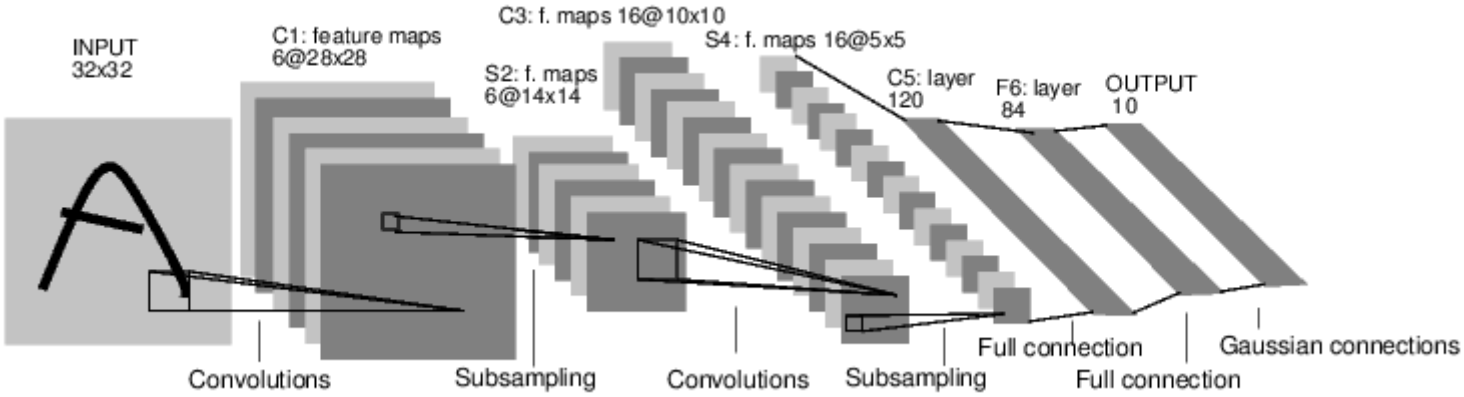
Theano 的架构比较旁门左道，它的整个代码库都是 Python 的，就连 C/CUDA 代码也要被打包为 Python 字符串，这使得它难以导航、调试、重构和维护。它允许使用者有效地定义、优化和评估涉及多维数组的数学表达式，同时支持 GPU 和高效符号分化操作，很容易用 Lasagne/Keras 实现新网络或者编辑现存网络。Theano 可与其他学习库配合使用，非常适合数据探索和研究活动。



Theano 计算框架图

Torch (5 pts)

Torch 的核心是流行的神经网络和优化库，它们易于使用，同时在实现复杂的神经网络拓扑结构时具有最大的灵活性；你可以建立任意的神经网络图，并在 CPUs 和 GPUs 上有效地并行化。最新 Torch7 和 NN 类库拥有清晰的设计和模块化接口。在 Torch 中，模块 (Module) 是构建神经网络的基石。模块本身也是神经网络，它可以和其他网络借助容器 (container) 构建更复杂的神经网络。下图是用于数字图像分类的 Torch 网络结构，一个简单的前馈神经网络，依次包含如下层：输入层、卷积层、池化层、卷积层、池化层、全连接层、全连接层 (线性层)、高斯连接层，构成一个有序的网络容器。



Torch 网络容器架构图

3.调度方式

Caffe (4 pts)

Caffe 模型的定义，是使用 Buffer 协议语言写成配置文件。Caffe 支持任意有向无环图形式的网络体系结构。在实例化时，Caffe 完全根据需要为网络储备尽可能多的内存，从底层主机或 GPU 中抽象。在单片 CPU 和 GPU 之间实现切换正是函数调用。

CNTK (4 pts)

CNTK 网络图有一些特殊的节点。它们是描述输入数据和训练标签的 FeatureNodes 和 LabelNodes，用来评估训练结果的 CriterionNodes 和 EvalNodes，和表示输出的 OutputNodes。CNTK 网络需要用到两个脚本：一个控制训练和测试参数的配置文件，是使用“Simple Network Builder”，只需设置几个参数就能生成一个简单的标准神经网络；另一个用于构建网络的网络定义语言 (Network Definition Language，NDL) 文件。

GraphLab (4.5 pts)

GraphLab 更新调度描述顺序，更新函数被应用到顶点，由一个叫调度器的并行数据结构表示。调度器抽象地表示任务动态列表 (顶点函数对)，这是由 GraphLab 引擎来执行。对于某个顶点，其被部署到多台机器，一台机器作为 master 顶点，其余机器上作为 mirror。Master 作为所有 mirror 的管理者，负责给 mirror 安排具体计算任务；mirror 作为该顶点在各台机器上的代理执行者，与 master 数据的保持同步。对于某条边，GraphLab 将其唯一部署在某一台机器上，而对边关联的顶点进行多份存储，解决了边数据量大的问题。同一台机器上的所有 edge 和 vertex 构成 local graph，在每台机器上，存在本地 id 到全局 id 的映射表。vertex 是一个进程上所有线程共享的，在并行计算过程中，各个线程分摊进程中所有顶点的 gather->apply->scatter 操作。

GraphX (4 pts)

GraphX 公开了一个类似 Pregel 的操作，它是广泛使用的 Pregel 和 GraphLab 抽象的一个融合。在 GraphX 中，更高级的 Pregel 操作是一个约束到图拓扑的批量同步并行 (bulk-synchronous) 消息抽象。Pregel 操作者执行一系列的超级步骤 (super steps)，在这些步骤中，顶点从之前的超级步骤中接收进入 (inbound) 消息的总和，为顶点属性计算一个新的值，然后在以后的超级步骤中发送消息到邻居顶点。GraphX 不像 Pregel 而更像 GraphLab，消息作为一个边三元组的函数被并行计算，消息计算既访问了源顶点特征也访问了目的顶点特征。在超级步骤中，没有收到消息的顶点被跳过；当没有消息遗留时，Pregel 操作停止迭代并返回最终的图。与更标准的 Pregel 实现不同的是，GraphX 中的顶点仅仅能发送信息给邻居顶点，并利用用户自定义的消息函数构造消息；这些限制允许在 GraphX 进行额外的优化。

MXNet (5 pts)

MXNet 其核心是动态依赖调度程序，该程序可以动态自动进行并行化符号和命令的操作。其中部署的图形优化层使得符号操作更快，内存利用率更高。该库轻量且便携带，并且可扩展到多个 GPU 和多台主机上。

Petuum (5 pts)

Petuum 的系统设计建立于机器学习的特征之上，目前包含两个主要模块：key-value store 和 scheduler，主要处理两类并行化方法：(1) 数据并行；(2) 模型并行。数据并行，简单而言，就是把数据分布到不同机器上，每台机器计算一个模型的更新，然后对这些 update 进行汇总并用之更新模型。模型并行，把模型参数进行切分并放置到不同机器上，每台机器对自己那部分进行更新。Key-value store 模块负责数据并行，采用的架构是 parameter server。另外一个模块 scheduler 用于模型并行，scheduler 提供的编程接口主要包含三个操作：(1) schedule，调度节点根据模型参数的相互依赖性和收敛的不均匀性，自动选择一个待更新的参数子集；(2) push，调度节点令计算节点并行地为选好的参数计算 update；(3) pull，调度节点从计算节点收集 update，并更新参数。

TensorFlow (4.5 pts)

TensorFlow 自动分化表达式。用户可以定义一个神经网络作为层的组成和损失函数，以及获得反向传播的库。分化算法执行广度优先搜索，从目标操作 (例如，损失函数) 到参数集，确定所有向后路径，并对每个路径的贡献进行局部梯度求和。TensorFlow 用户还可以尝试广泛的优化算法，计算出每个训练步骤中的参数新值。

Theano (4.5 pts)

Theano 可以自动计算复杂表达式的符号分化，忽略不需要的变量计算最终输出，重用部分结果避免重复计算，应用数学简单化。在可能最小化内存使用的情况下，计算操作，并应用数值稳定性优化来克服或减少由于硬件近似带来的错误。要做到这一点，用户定义的数学表达式要存储为变量和操作的图，在编译时进行修剪和优化。为了克服其 Python 接口在数值计算时内存和速度的局限性，Theano 利用 Python 语言的压实性和延性与一种快速、优化的计算引擎相结合。

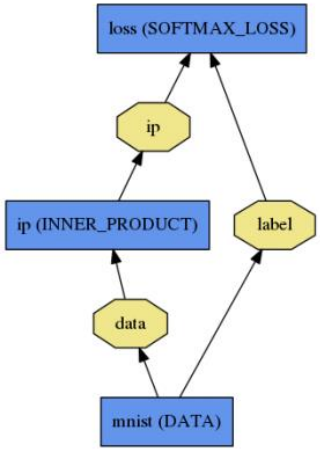
Torch (4 pts)

Torch 的优点在于快速，灵活，支持 CPU 模式和 GPU 模式，在多 CPU 模式下提速效果最为明显。同样是由于 Lua 语言书写，缺点在于没有 Python 接口，无法整合 Python 上的资源。如同 Caffe 一样，Torch 对新型网络连接和架构的支持不如 Theano 和 TensorFlow，深度神经网络中层的种类受到限制。

4.计算方式

Caffe (4 pts)

Caffe 是典型的功能 (过程) 计算方式，它首先按照每一个大功能 (可视化、损失函数、非线性激励、数据层) 将功能分类并针对部分功能实现相应的父类，再将具体的功能实现成子类，或者直接继承 Layer 类，从而形成了 XXXLayer 的形式。然后将不同的 layer 组合起来就成了 net。



Caffe 计算网络结构

CNKT (3.5 pts)

CNTK 和 TensorFlow 都是通过符号化分析流程图来计算梯度下降训练算法中所用到的梯度值。现阶段 CNTK 只支持一种学习方法：Mini-batch 随机梯度下降法。

GraphLab (4.5 pts)

Graphlab 中的计算用 vertex-program 表示。通过一个顶点为中心的模型，其中计算被定义为每个顶点的内核运行，它能够在每个顶点上并行执行。在另一方面，GraphLab 是一个顺序共享内存的抽象，每个顶点都可以在相邻的顶点和边上读取和写入数据。GraphLab 运行时间负责确保一致并行执行。因此，GraphLab 简化了图并行算法的设计和实施，从而释放用户关注顺序计算而非并行移动数据 (如，及时消息)。

GraphX (5 pts)

GraphX 是一个新的 Spark API，它用于图和并行图 (graph-parallel) 的计算。GraphX 通过引入 Resilient Distributed Property Graph：带有顶点和边属性的有向多重图，来扩展 Spark RDD。为了支持图计算，GraphX 公开一组基本的功能操作以及 Pregel API 的一个优化。另外，GraphX 包含了一个日益增长的图算法和图 builders 的集合，用以简化图分析任务。

MXNet (5 pts)

MXNet 提供 cpu/gpu 的矩阵和矢量计算，能够自动并行。MXNet 的 NDArray 类似 numpy.ndarray，也支持把数据分配在 GPU 或者 CPU 上进行运算。但是与 numpy 和 caffe 不同的是，当在操作 NDArray，它能自动的将需要执行的数据分配到多台 GPU 和 CPU 上进行计算，从而完成高速并行。在调用者的眼中代码可能只是一个单线程的，数据只是分配到了一块内存中，但是背后执行的过程实际上是并行的。将指令 (加减等) 放入中间引擎，然后引擎来评估哪些数据有依赖关系，哪些能并行处理。定义好数据之后将它绑定到网络中就能处理它了。

MXNet 是符号计算和过程计算混合，它设计了 symbol 大类，提供了很多符号运算的接口，每个 symbol 定义了对数据进行怎样的处理，symbol 只是定义处理的方式，这步还并未真正的执行运算。其中一个需要注意的是 symbol 里面有 Variable，它作为承载数据的符号，定义了需要传递什么样的数据给某个

Variable，并在后续的操作中将数据绑定到 Variable 上。下面的代码是一个使用示例，它实现了将激励函数连接到前面定义好的 net 后面，并给出了这一个 symbol 的名字和激励函数类型，从而构造出 net。下图左边部分是定义 symbol 的合集，中间将数据绑定到 Variable 上之后变成了右边真正的执行流程图。

net = mx.symbol.Activation(data=net, name='relu1', act_type="relu")

MXNet 计算网络结构

Petuum (4.5 pts)

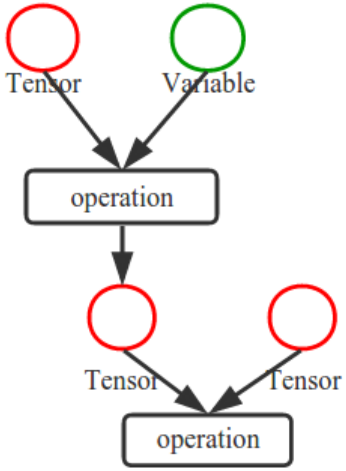
Petuum 全局共享参数表示为 (可能是稀疏) 向量和矩阵，更方便机器学习应用程序的数据结构，而不是广泛使用 (key，value) 存储或表。高性能、多线程的线性代数操作方便，如参数和本地训练数据之间的矢量-矩阵乘法，提供方便的应用程序开发。Petuum 平台由 Bosen 和 Strads 两个子系统构成。Bosen 是一个参数服务器的实现，参数服务器是近年来新出现的专门用于机器学习算法的数据并行抽象，它通过采用一个分布式的 Key-Value 存储模型存放参数，这样就提供了有效的机制用于在分布式系统不同的 Worker 节点之间同步模型参数，而每个 Worker 只需要保存它计算时所依赖的小部分参数即可为避免以 Key-Value 为单元进行频繁的参数数据交互会导致过高的通信开销。参数服务器通常采用数学封装来进行参数同步，比如向量，张量，矩阵的行列等。

TensorFlow (5 pts)

TensorFlow 相当于 N 维的 array 或者 list，维数可变，数据类型一旦定义不能改变。TensorFlow 的 tensor，它相当于 N 维的 array 或者 list，与 MXNet 类

似，都是采用了以 Python 调用的形式展现出来。某个定义好的 tensor 的数据类型是不变的，但是维数可以动态改变。用 Tensor rank 和 TensorShape 来表示它的维数 (例如 rank 为 2 可以看成矩阵，rank 为 1 可以看成向量)。tensor 是个比较中规中矩的类型。唯一特别的地方在于在 TensorFlow 构成的网络中，tensor 是唯一能够传递的类型，而类似于 array、list 这种不能当成输入。

TensorFlow 选择的是符号计算方式，它的程序分为计算构造阶段和执行阶段，构造阶段是构造出 computation graph，computation graph 就是包含一系列符号操作 Operation 和 Tensor 数据对象的流程图，跟 MXNet 的 symbol 类似，它定义好了如何进行计算（加减乘除等）、数据通过不同计算的顺序 (也就是 flow，数据在符号操作之间流动的感觉)。但是暂时并不读取输入来计算获得输出，而是由后面的执行阶段启动 session 的 run 来执行已经定义好的 graph。这样的方式跟 MXNet 很相似，应该都是借鉴了 Theano 的想法。其中 TensorFlow 还引入了 Variable 类型，它不像 MXNet 的 Variable 属于 symbol (TF 的 operation 类似 MXNet 的 symbol)，而是一个单独的类型，主要作用是存储网络权重参数，从而能够在运行过程中动态改变。TF 将每一个操作抽象成了一个符号 Operation，它能够读取 0 个或者多个 Tensor 对象作为输入 (输出)，操作内容包括基本的数学运算，支持 reduce、segment (对 tensor 中部分进行运算。例如 tensor 长度为 10，可以同时计算前 5 个，中间 2 个，后面三个的和)，对 image 的 resize、pad、crop、filpping、transposing 等。



TensorFlow 计算图

Theano (4.5 pts)

Theano 可以自动计算复杂表达式的符号分化，忽略不需要的变量计算输出，重用部分结果，避免重复计算，应用数学简单化。Theano 在可能最小化内存使用的情况下计算操作，并应用数值稳定性优化来克服或减少由于硬件近似带来的错误。要做到这一点，用户定义的数学表达式要存储为变量和操作的图，在编译时进行修剪和优化。

Torch (4.5 pts)

Torch 实现并优化了基本的计算单元，使用者可以很简单地在此基础上实现自己的算法核心计算单元使用 C 或者 CUDA 做了很好的优化，使用 Lua 构建了常见模型。支持全面的卷积网络操作，支持时间卷积：输入长度可变，对 NLP 非常有用，而 TF 和 Theano 都不支持；支持 3D 卷积：对视频识别很有用，而 TF 不支持。

5.一致性协议

Caffe (3.5 pts)

Caffe 实现了单机多 GPU 数据并行，通过 I/O 模块给每个 GPU 预缓冲 batch 数据，然后使用同步随机梯度下降算法进行训练。数据并行训练中，每个 GPU 持有一份模型的完整拷贝，各自训练后计算得到梯度值，然后进行参数交换。Caffe 采用树形拓扑结构进行梯度的归约和模型参数的分发。为了保证模型参数的数据一致性，所有 GPU 同时训练一个批次的训练数据，完成后经过同步等待，再同时交换参数。Caffe 的 I/O 模块从文件中读取并分发下一 batch 数据，以达到用计算时间掩盖 I/O 时间的目标。Caffe 中多 GPU 在同一个数据集训练时，数据输入层 DataLayer 无论是否允许共享，都只能有一个线程读取数据库。相较而言，每个模型拥有一个独立的数据输入层的并行度更高，负载均衡也较好。

由于 Caffe 不支持多机多卡分布式并行，难以应对实际生产环境中 PB 级的训练数据量。从设计实现方面来讲，Caffe 采用树形拓扑结构同步地交换参数这种实现方案是较为原始和低效的。Caffe 采用同步更新参数的方式维护全局参数的数据一致性，每一轮 Batch 训练时要等待所有的 GPU 计算结束才归约梯度，并行速度受最慢的 GPU 限制，同步等待时间较长。其次，树形拓扑导致每个归并周期后，总有一半 GPU 不再参与之后的归并过程，闲置了其计算能

力和所在节点上的通信带宽。树形拓扑的可扩展性也不够好，当 GPU 数目为奇数时构建树形结构 Caffe 会报错。此外，由于 C 语言源生多线程在 Python 里是无效的，因此 Caffe 的 Python 接口不能使用 GPU 并行训练。更好的方案是移除 Caffe 内部的并行方案，将多线程写在外面。

CNTK (5 pts)

CNTK 只提供数据并行，它采用参数服务器模型实现了一种称为 1-Bit Quantized SGD 的算法，其目的就是用于节约带宽，其主要思想是压缩梯度的表示到只用 1bit，把残差带到下一次的 minibatch 中。相比用浮点数 (32 位) 表示梯度值，1-Bit SGD 相当于节约了 30 多倍的传输带宽。

GraphLab (4.5 pts)

GraphLab 采用的是 Asynchronously Dynamic Update，这种动态计算的主要思想是根据 vertex 的 priority 更新，每台机器上都有一个优先队列，每次迭代中如果当前 vertex 变化量不大的话就不再将该点的 scope (一步可达的点) 入队了，ghost 顶点不需要入队。

GraphX (4.5 pts)

GraphX 通过引入 Resilient Distributed Property Graph (一种点和边都带属性的有向多图) 扩展了 Spark RDD 这种抽象数据结构，这种 Property Graph 拥有两种 Table 和 Graph 两种视图 (及视图对应的一套 API)，而只有一份物理存储。两种视图都有自己独有的操作符，从而获得了灵活操作和执行效率。图的分布式或者并行处理其实是把图拆分成很多的子图，然后分别对这些子图进行计算，计算的时候可以分别迭代进行分阶段的计算，即对图进行并行计算。

MXNet (5 pts)

MXNet 依赖于 PS-Lite 提供分布式模型训练，因此我们可以直接看 PS-Lite。在设计上包含一个 Server Group 和若干个 Worker Group，Server Group 用来做参数服务器，每个 Server Node 存放一个参数分片，由 Server Manager 管理整个 Server Group，维持整个 Server Group 的元数据的一致性视图，以及参数分片情况。提供支持其分布式训练特性的正是 parameter server，所以一致性协议是 SSP。

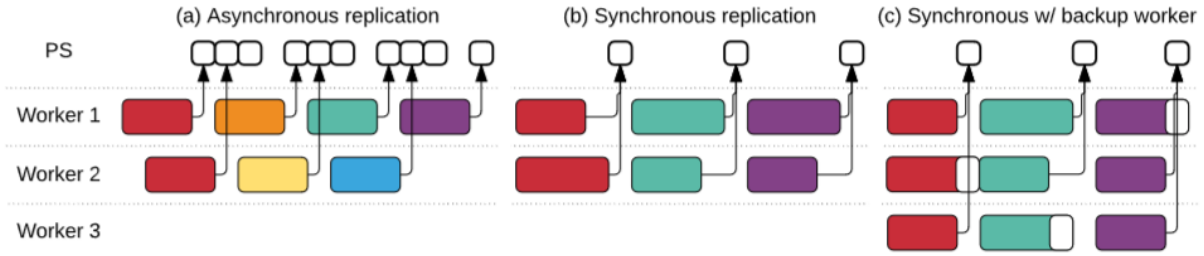
Petuum (5 pts)

Petuum 一致性协议是 Staleness Synchronous Parallel (SSP)。SSP 的基本思想是允许各机器以不同步调对模型进行更新，但是加一个限制，使得最快的机器的进度和最慢机器的进度之差不要太大。这样做的好处是：既减轻慢的机器拖整个系统的后腿，又能保证模型的最终收敛。通过调节 SSP 的 staleness 参数，SSP 可以转化成数据流系统常用的 BSP（Bulk Synchronous Parallel) 协议或者早期机器学习系统 (如 Yahoo LDA) 使用的 ASP(Asynchronous Parallel)。

Parameter server 提供了一个易于读写 Global 模型参数的接口，而 SSP 协议允许 distributed workers 读写本地缓存中 stale 版本的参数 (而不是每次都花大量时间时间等待 central storage 传回最新参数)。更进一步，通过限制参数的 stale 程度，SSP 模型提供了机器学习算法的正确性保证。SSP 协议的好处在于，faster worker 会遇到参数版本过于 stale 的问题，导致每一步迭代都需要网络通信，从而达到了平衡计算和网络通信时间开销的效果。

TensorFlow (4 pts)

TensorFlow 虽然设计了异步训练，也已经开始试验同步方法。TensorFlow 图使用户在训练模型时能够更改参数读取和写入。如图，TensorFlow 实现三种替代。TensorFlow 采用数据并行，利用 DownpourSGD (既 DistBelief) 结构的参数服务器可以很容易实现分布式深度学习，这种方式很适合于大量数据和较小模型的训练方式。



TensorFlow 三种协议模式

Theano (3 pts)

Theano 对分布式运行模式支持没有其他框架好。

Torch (4.5 pts)

Torch 框架上提供了 AutoGrad，同时提供 AllReduceSGD 和 AllReduceEA 接口，其中后者是异步实现，提供了一个区别于 DownpourSGD 的异步 SGD 实现 EASGD (Elastic Average SGD)，跟 DownpourSGD 相比，EASGD 把 Worker 节点上的参数跟参数服务器的中心变量联系在一起，这样使得 Worker 本地的变量会围绕中心变量进行变化，从理论上可以证明比 DownpourSGD 有更快的收敛和更小的数据传输，也就是说，通过更快收敛来达到节省带宽传输的目的。

6.部署模式

Caffe (5 pts)

Caffe 是基于 C++的，因此可以在多种设备上编译，具有跨平台性，在部署方面是最佳选择。

CNTK (4.5 pts)

CNTK 与 Caffe 一样也是基于 C++并且跨平台的，大部分情况下部署非常简单。但是它不支持 ARM 架构，这限制了它在移动设备上的能力。

GraphLab (5 pts)

GraphLab 优化 C++执行引擎，在大量多线程操作和同步 I/O 操作之间能很好地平衡，支持 C++、Java、Python 等语言，跨平台，可以运行在多处理机的单机系统、集群或是亚马逊的 EC2 等多种环境下。

GraphX (4.5 pts)

GraphX 基于 Spark 平台提供对图计算和图挖掘简洁易用且丰富的接口，极大方便分布式图处理。支持 Java、C#、Python、Scala 多种语言，跨平台。

MXNet (5 pts)

MXNet 允许使用者将符号编程和命令式编程相结合，以追求效率和生产力的最大化；支持接口语言很多，包括：C++、Python、Julia、Matlab、JavaScript、R、Scala，并且跨多种平台，包括移动设备。

Petuum (4.5 pts)

Petuum 可高效地运行于各种硬件环境下，包括实验室集群服务器和云计算平台，如 Amazon EC2、Google GCE；主要支持 Linux 和 Ubuntu 平台。

TensorFlow (4.5 pts)

TensorFlow 支持 C++接口，同时由于它使用了 Eigen 而不是 BLAS 类库，所以能够基于 ARM 架构编译和优化。TensorFlow 的用户能够将训练好的模型部署到多种设备 (或者移动设备) 上，不需要实现单独的模型解码器或者加载 Python/LuaJIT 解释器。但是 TensorFlow 并不支持 Windows，因此其模型无法部署到 Windows 设备上。

Theano (3 pts)

Theano 缺少底层的接口，并且其 Python 解释器也很低效，对工业用户而言缺少吸引力。虽然对大的模型其 Python 开销并不大，但有限制。亮点是天然跨平台，模型能够部署到 Windows 环境上。

Torch (3 pts)

Torch 模型运行需要 LuaJIT 的支持，虽然这样做对性能的影响并不大，但却对集成造成了很大的障碍，使得它的吸引力不如 Caffe/CNTK/TensorFlow 等直接支持 C++的框架。

7.接口

Caffe (3.5 pts)

Caffe 支持 pycaffe 接口，但这仅仅是用来辅助命令行接口的，而即便是使用 pycaffe 也必须使用 protobuf 定义模型，支持 Python、Matlab。

CNTK (3 pts)

CNTK 的使用方式与 Caffe 相似，也是通过指定配置文件并运行命令行，但 CNTK 不如 Caffe 的地方是没有 Matlab 语言的接口。

GraphLab (4 pts)

GraphLab 支持 C++、Java、Python 等语言。

GraphX (4.5 pts)

GraphX 可以认为是 GraphLab (C++) 和 Pregel (C++) 在 Spark (Scala) 上的重写及优化，支持 Java、C#、Python、Scala 等接口语言。

MXNet (5 pts)

MXNet 支持众多接口语言，包括 C++、Python、Julia、Matlab、JavaScript、R、Scala。

Petuum (3 pts)

Petuum 目前支持的接口语言不多，只有 C++、Java。

TensorFlow (4 pts)

TensorFlow 支持 Python 和 C++两种类型的接口。用户可以在一个相对丰富的高层环境中做实验并在需要本地代码或低延迟的环境中部署模型。

Theano (3 pts)

Theano 支持 Python 接口。

Torch (4 pts)

Torch 运行在 LuaJIT 上，与 C++、C#以及 Java 等工业语言相比速度非常快，用户能够编写任意类型的计算，不需要担心性能，唯一的问题就是 Lua 并不是主流的语言。

8.并行性

Caffe (3 pts)

Caffe 应用于分布式情况比较少，并行计算效果不好。

CNTK (4.5 pts)

CNTK 适合并行计算，支持大规模 GPU，实现了跨多 GPU 和服务器自动分化和并行化的随机梯度下降 (误差反向传播) 学习。

GraphLab (5 pts)

GraphLab 是图并行抽象，对于多核处理器和分布式集群环境，一次编写程序即可高效地运行。内置支持 HDFS，能直接读取 HDFS 中数据或者将数据直接写入 HDFS。可以高效并行执行具有稀疏的计算依赖性强的迭代性算法，适用于大规模机器学习任务，也适用于许多数据挖掘方面的计算任务。

GraphX (4.5 pts)

GrapxX 融合了图并行以及数据并行的优势，是 Spark 中用于图 (如社交网络) 和图并行计算 (如 PageRank、协同过滤) 的 API，可以方便且高效地完成图计算的一整套流水作业。GraphX 天然就是一个分布式的图处理系统，基于 Spark 平台提供对图计算和图挖掘简洁易用且丰富的接口，极大方便分布式图处理。

MXNet (4.5 pts)

MXNet 是分布式机器学习通用工具包 DMLC 的重要组成部分，注重灵活性和效率，同时强调提高内存使用的效率。对“云计算”友好，直接兼容 S3、HDFS 和 Azure。

Petuum (5 pts)

Petuum 可使用户从复杂繁琐的分布式系统编程和调试中解脱出来，将更多精力集中在优化模型和算法上，可高效地运行于各种硬件环境下，包括实验室集群服务器和云计算平台，如 Amazon EC2、Google GCE，可支持几百台机器。

TensorFlow (5 pts)

TensorFlow 灵活的架构可以让使用者可以多样化地将计算部署在台式机、服务器或者移动设备的一个或多个 CPU 上，而且无需重写代码。为并行计算而设计，支持 CUDA，大规模 GPU 支持才是真正的设计点。

Theano (3 pts)

Theano 很少运用在分布式并行情景中。

Torch (4 pts)

Torch 有一个在机器学习领域大型生态社区驱动库包，包括计算机视觉软件包，信号处理，并行处理等。Torch 的核心是流行的神经网络，它使用简单的优化库，同时具有最大的灵活性，实现复杂的神经网络的拓扑结构，通过 CPU 和 GPU 等有效方式，可以建立神经网络和并行任意图。

9.性能

Caffe (4.5 pts)

Caffe 简单快速，对于科研来说，接近工业化的速度处理大规模数据，拥有当前最牛掰的算法。

CNTK (5 pts)

CNTK 简单快速，并行计算优势大。在多 GPU 方面，CNTK 相较于其他的深度学习工具包表现更好，它实现了 1-bit SGD (Stochastic Gradient Descent) 和自适应的 minibatching。

GraphLab (4.5 pts)

GraphLab 能够智能地选择存储和计算的节点，算法设计优良，图并行抽象，适用于大规模机器学习任务，也适用于许多数据挖掘方面的计算任务。

GraphX (4 pts)

GraphX 在 Spark 之上提供一栈式数据解决方案，完成图计算的一整套流水作业方便且高效；从整个图处理流水线视角 (图构建、图合并、最终结果查询) 看，性能非常具有竞争性。

MXNet (4.5 pts)

强调提高内存使用的效率，甚至能在智能手机上运行诸如图像识别等任务；整合了各种编程方法的优势，最大限度地提高灵活性和效率。

Petuum (5 pts)

在不损失模型性能的前提下，可在更少的硬件上更快地运行更大模型，高性能，最大化系统性能。

TensorFlow (4.5 pts)

TensorFlow 初代比较慢，2016 年 5 月以后，在 ConvNet 速度上已经赶上其他框架，但即使如此它的性能依然要比同样使用 cuDNN v2 的 Torch 要慢，延迟毫秒如下，在单片 Titan X GPU 下测试。

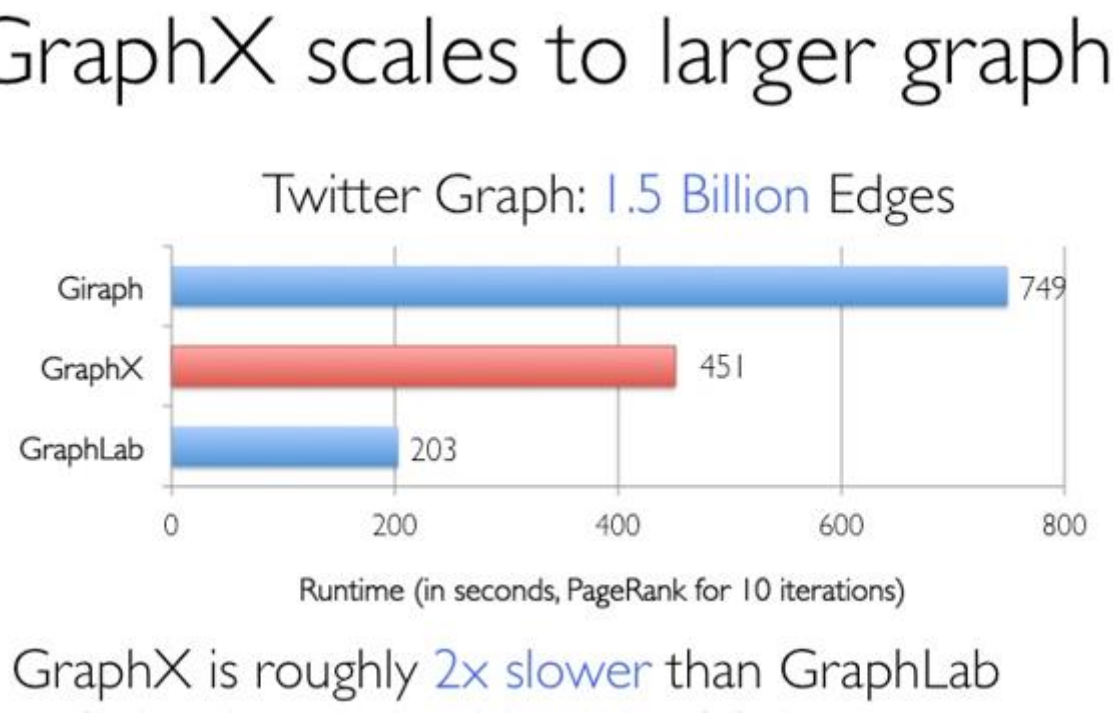
Network	TF 0.6	TF 0.8	Torch FP32
AlexNet	292	97	81
Inception v1	1237	518	470

Theano (3.5 pts)

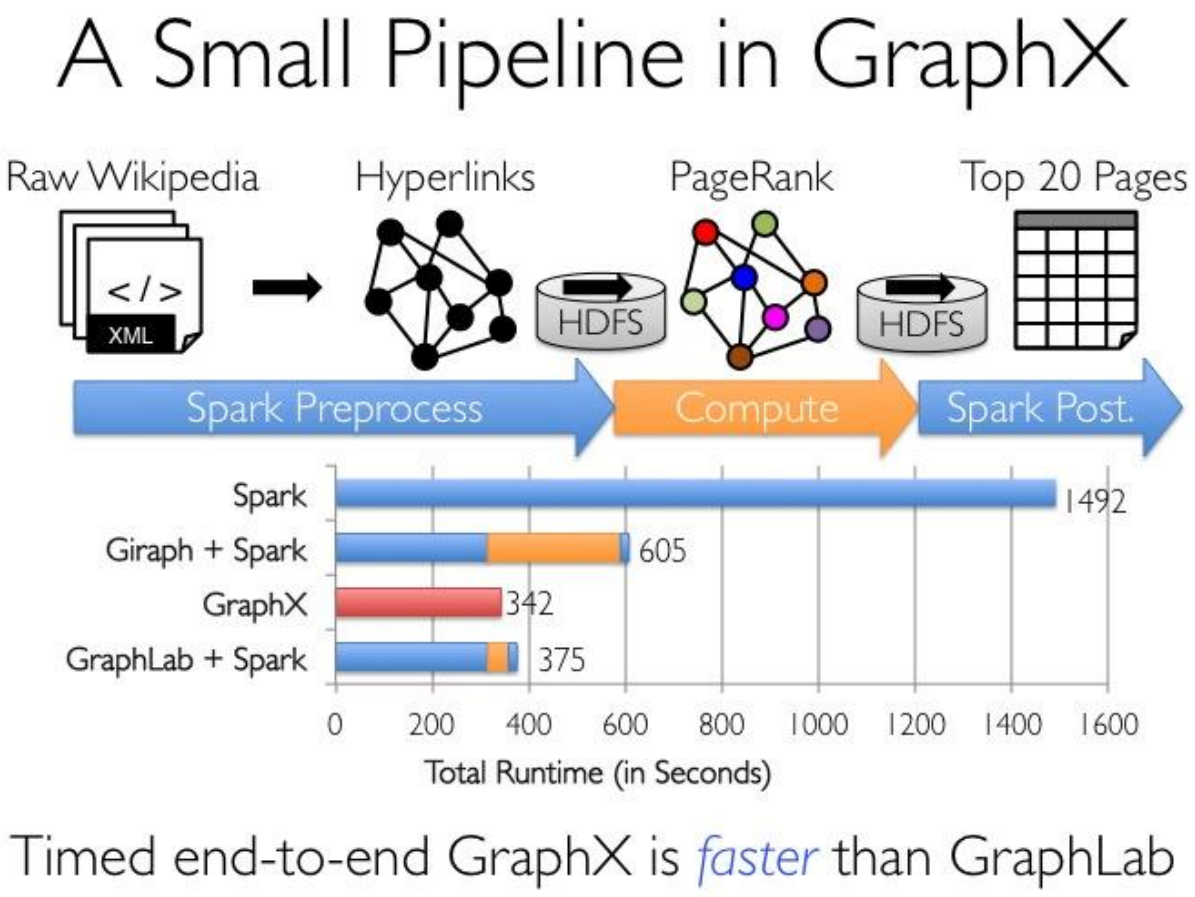
Theano 在大型网络上的性能与 Torch7 不相上下。但它的主要问题是启动时间特别长，因为它需要将 C/CUDA 代码编译成二进制，实际上，深度学习研究者花更多的时间调试而不是训练大量模型。TensorFlow 并没有这个问题。此外，Theano 的导入也会消耗时间，并且在导入之后无法摆脱预配置的设备(例如 GPU0)。

Torch (4.5 pts)

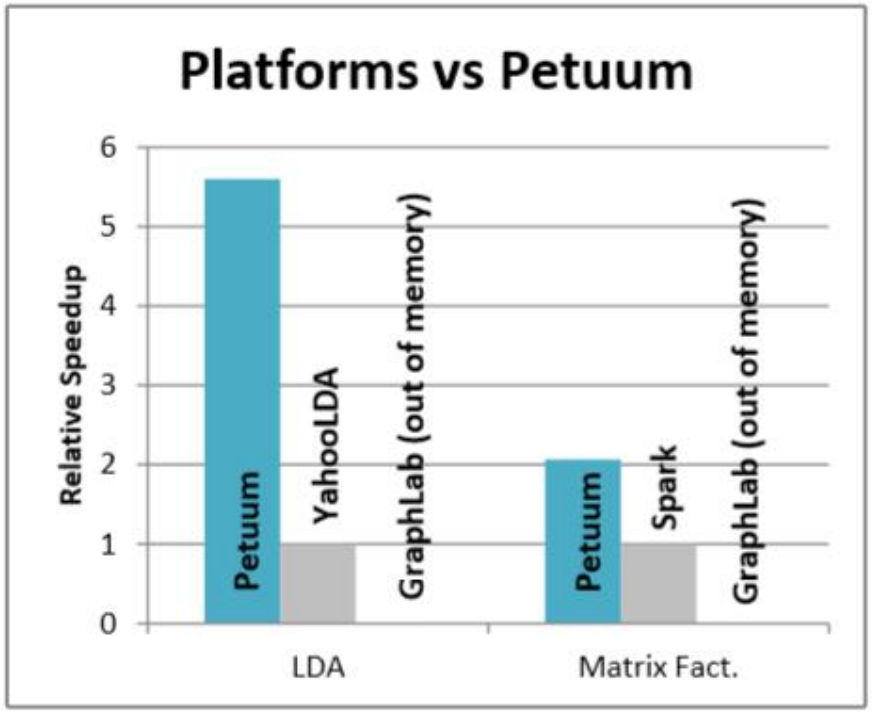
Torch 非常好，没有 TensorFlow 和 Theano 的问题，有较好的灵活性和速度。

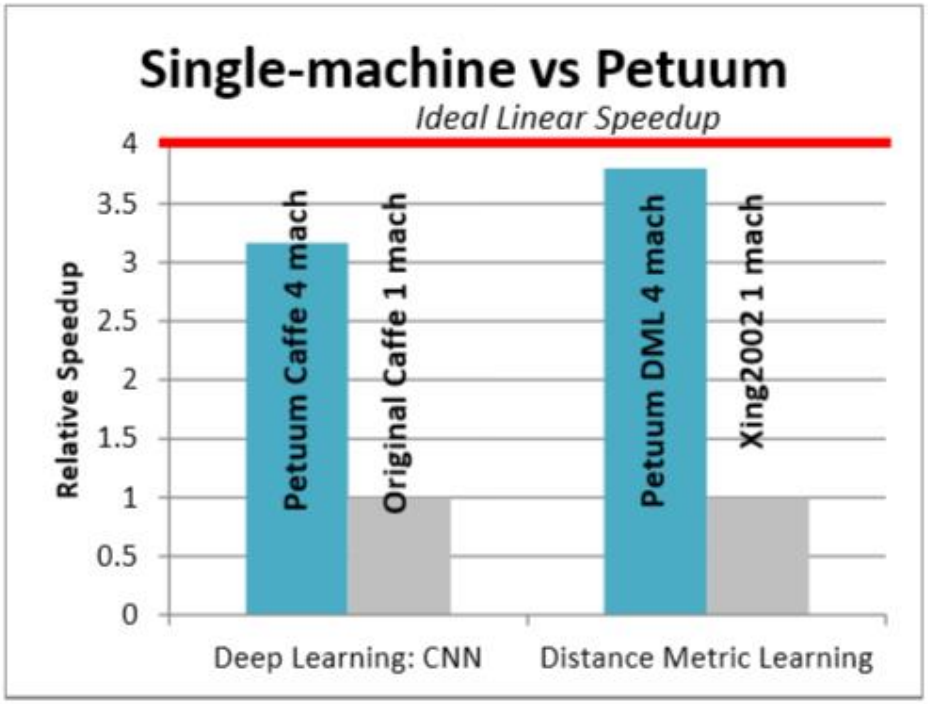


Graph、GraphX 和 GraphLab 关于 Twitter 图计算运行时间的比较

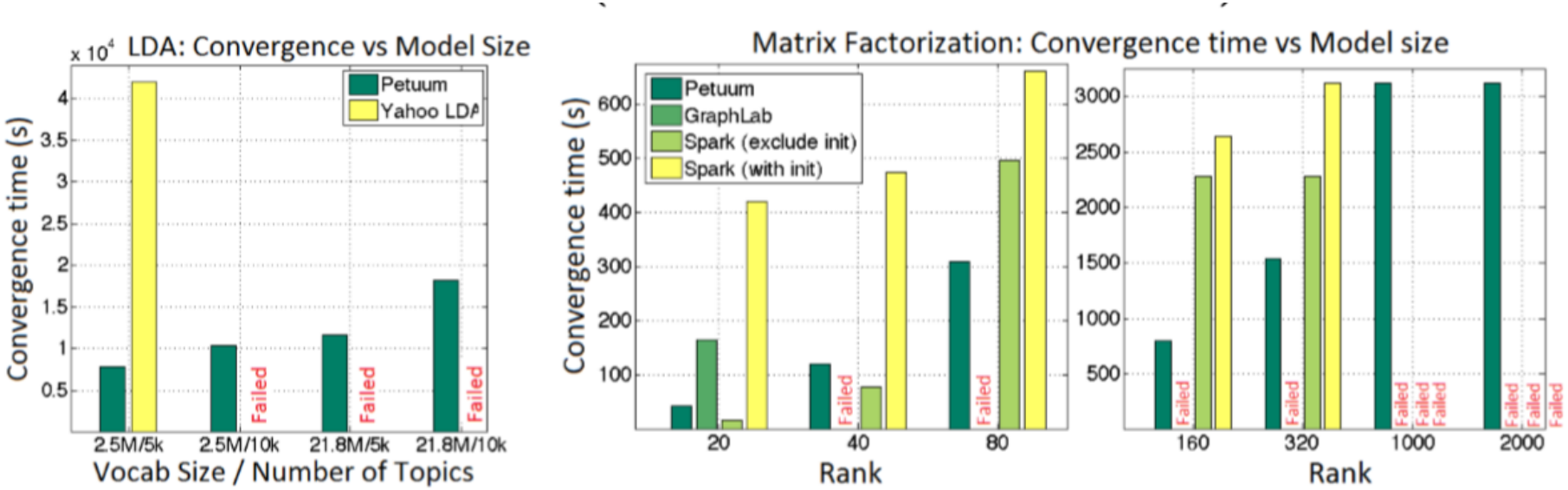


从整个图计算 Pipeline 来说，GraphX 的总体 Runtime 少于 GraphLab+Spark

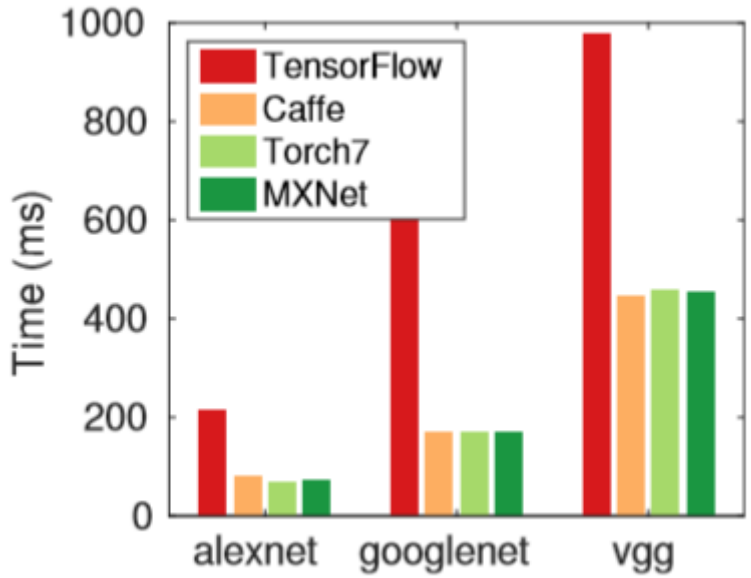




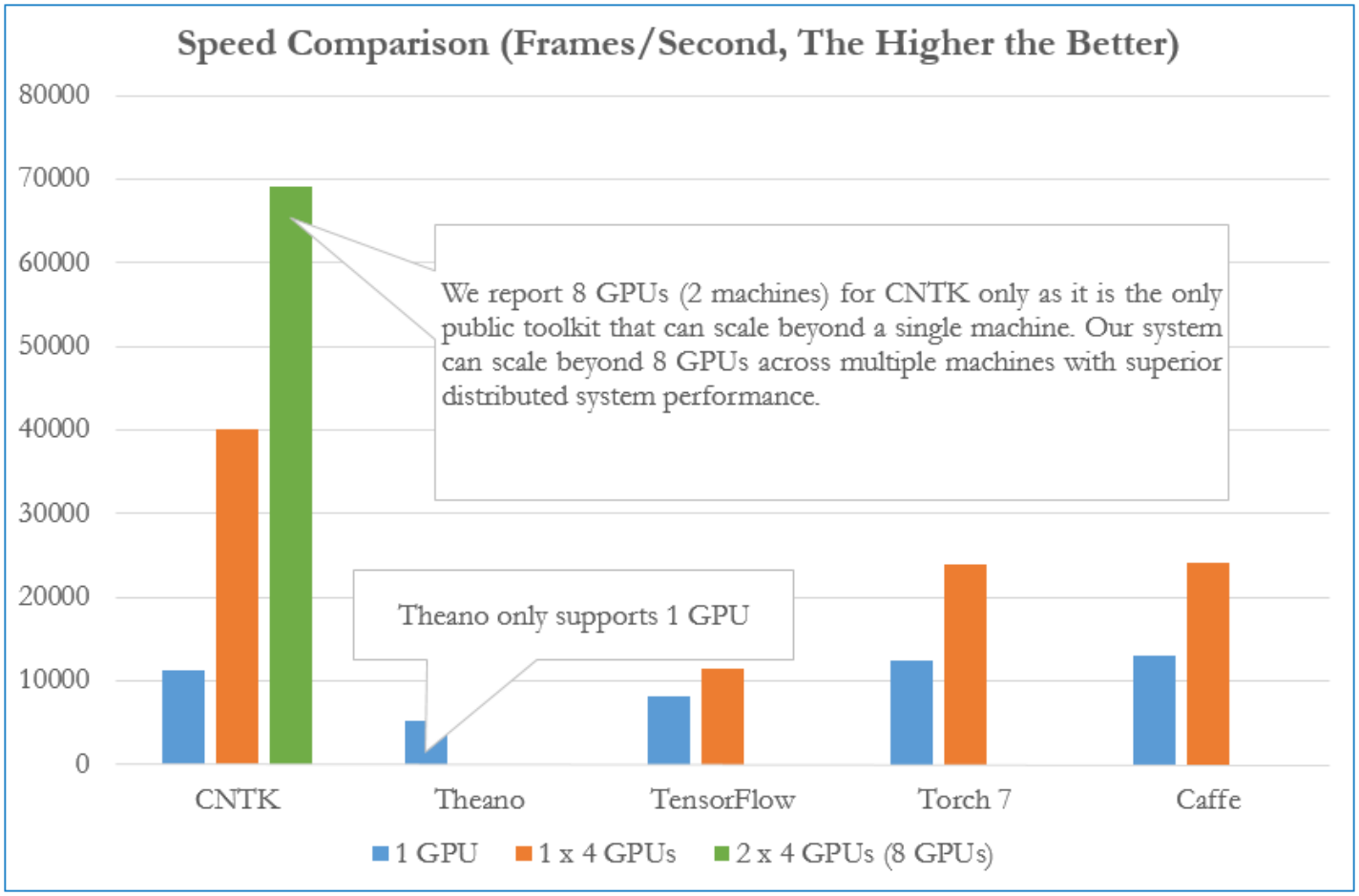
左图是 Petuum 的性能表现，Petuum 比其他 ML 框架的相对加速快 2-10 倍，GraphLab 内存溢出；右图是单机算法集群版的加速表现，增加机器 (Caffe CNN 和 DML) 数量的 Petuum 性能表现几乎成线性增长



左图是关于 LDA (线性判别分析) 收敛时间，Petuum 数据+模型并行 LDA 收敛比 YahooLDA (仅数据并行) 快得多，随着参数不断增多，YahooLDA 收检失败；右边两图是关于矩阵分解收敛时间，Petuum 最快，最节约内存，而且 Petuum 是可以在给定的硬件预算中，处理矩阵的秩超过 1000 的大型矩阵分解模型的唯一框架。Petuum 的模型可拓展性主要归功于两个要素：(1) 模型并行化，跨机器划分模型；(2) 轻量级的参数服务器系统，拥有最小的存储开销 (从应用简单数组到哈希图)



TensorFlow、Caffe、Torch7 和 MXNet 在单个前向-后向反馈神经网络上的性能表现



各种工具集处理速度 (帧处理/秒) 的比较。配置：使用一个完全连接的 4 层神经网络和一个迷你高效的批处理大小 (8192)，在相同的硬件上于 2015 年 12 月 3 日获得所有结果

10.容错

Caffe (4 pts)

Caffe 记录任何有向无环图的层，确保向前和向后传递的正确性。Caffe 模型是端到端的机器学习系统，一个典型网络开始于磁盘加载的数据层，结束于为分类或重建等任务计算目标的损失层。Caffe 循环中产生一些错误，但模型最终收敛。

CNTK (4.5 pts)

CNTK 支持服务器自动分化和并行化的随机梯度下降 (误差反向传播) 学习，具有高容错。

GraphLab (4 pts)

GraphLab 在容错方面做的还不是很好，主要是 Chandy-Lamport 的 asynchronous snapshotting algorithm。

GraphX (5 pts)

GraphX 建立在 Spark 上，提供基于 Spark 系列的容错，开销可以忽略不计，并且有可选的数据集复制。

MXNet (4.5 pts)

MXNet 即使在每个循环中产生一些错误，模型最终仍能收敛。参数收敛是非均匀性的，有些参数几轮迭代就会收敛，而有的参数却需要上百轮迭代。

Petuum (4.5 pts)

Petuum 节点失败是不可避免的，特别是在大规模使用商品服务器时。例如，1000 个节点 3 年的故障平均时间共计为，每天失败一次。在工业部署中，调度器先发制人可以显著提高比率。Petuum 使用一种经过优化的数据复制体系结构，高效地在多个服务器节点上存储数据，从而快速 (少于 1 秒) 从节点故障中恢复。此外，客户端节点相互独立后，新客户端可以在一个服务器发生故障时自动启动，以同样的方式 MapReduce 能够重新调度新的映射。

TensorFlow (4 pts)

TensorFlow job 在训练过程中可能会经历失败，所以需要某种形式的容错。然而，失败是不太可能如此普遍，只有个别操作需要容错，所以像 Spark RDDs 的机制开销显著，收益很小。因此，就不需要每个写入参数的持久状态，因为我们可以从输入数据重新计算任何更新，许多学习算法不需要强一致性。尽管 TensorFlow 不使用强一致性的训练状态，但依靠一个像 Chubby 或 ZooKeeper 的系统映射任务 id 到 IP 地址。TensorFlow 中的容错是使用图中的原始操作，执行用户级的检查点。TensorFlow 分布式执行中的故障可在各种地方检测出来。检测的主要错误是：发送和接收节点对之间通信的错误，从主进程到每个工作进程的定期健康检查。分布式的容错采取了 consistent checkpointrestart 策略，似乎没有更加 fancy 容错，但是又非常实用。

Theano (4 pts)

Theano 忽略不需要的变量计算最终输出，重用部分结果避免重复计算，应用数学简单化。在可能最小化内存使用的情况下，计算操作，并应用数值稳定性优化来克服或减少由于硬件近似带来的错误。

Torch (4.5 pts)

Torch 有非常灵活的容错及通讯管理机制，最后的结果就是系统功效的巨大提升。

11.代码复用性

Caffe (4 pts)

训练深度网络非常耗时，所以 Caffe 发布了一些预训练模型/权重 (model zoo)，能作为初始权重被用于特殊领域或自定义图像的迁移学习或微调深度网络。

CNTK (5 pts)

CNTK 不需要使用低层次的语言就能创建新的、复杂的层类型，有一套极度优化的运行系统来训练和测试神经网络，它是以抽象的计算图形式构建；CNTK 里面也集成了多种深度学习的模型。

GraphLab (5 pts)

GraphLab 在自身提供的 API 接口之上实现了大量开箱即用的工具集；对于多核处理器和分布式集群环境，一次编写程序即可高效地运行。

GraphX (4.5 pts)

GraphX 基于 Spark 平台提供对图计算和图挖掘简洁易用且丰富的接口，极大方便分布式图处理；代码很简洁，在 Spark 之上提供一栈式数据解决方案。

MXNet (4.5 pts)

MXNet 有一个 Caffe 转换工具，能够转换基于 Caffe 的预训练模型权重，使其可以适应 MXNet。另外，MXNet 有预训练模型，表现、记忆效果好；但编辑现存训练好的网络有点困难，因此使用特定域的自定义数据也有点难。

Petuum (5 pts)

Petuum 可编程性强，提供了简单易用的编程接口，用户可实现自己的机器学习算法，有丰富的机器学习库。

TensorFlow (3.5 pts)

TensorFlow 不支持预训练模型，已预定型的模型不多。

Theano (5 pts)

TheanoLasagne 是构建在顶尖 Theano 上的高级构架，在 Lasagne 中使用 Caffe 预训练模型权重非常容易。

Torch (4 pts)

Torch 大量模块化组件，容易组合；通常需要用户自己编写定型代码 (即插即用相对少)，需要一点时间学习。

12.短板

Caffe (2.5 pts)

Caffe 虽有很多扩展，但由于遗留的架构问题，不够灵活且对递归网络和语言建模支持很差。基于层的网络结构，扩展性差，对于新增加的层，需要自行实现前后向传播和梯度更新；运用于大型网络 (GoogLeNet、ResNet) 时过于繁琐。Caffe 支持 pycaffe 接口，仅用来辅助命令行接口；使用 pycaffe 时必须用 protobuf 定义模型。Caffe 不适用于文本、声音或时间序列数据等其他类型的深度学习应用。

CNTK (4 pts)

CNTK 不支持 ARM 架构，这限制了它在移动设备上的能力，现阶段只支持一种学习方法：Mini-batch 随机梯度下降法。

GraphLab (4 pts)

GraphLab 抽象成图不直接，深度学习多层结构中，已有算法移植到基于图抽象上需要不少工作，基于图抽象有时还会导致程序不正确，或陷入次优化。

GraphX (3 pts)

GraphX 与 GraphLab 相比，运行速度有一定差距，在单机上的计算性能不如 GraphLab 等计算框架。

MXNet (4 pts)

MXNet 迁移学习和微调网络可能实现，但和 Lasagne、Keras 相比不太容易，编辑现存训练好的网络更困难，因此使用特定域的自定义数据也有点难。

Petuum (4 pts)

Petuum 目前主要适用于几十台机器的集群，在更大集群上处理有风险，调度器设计有点复杂。

TensorFlow (4 pts)

TensorFlow 速度比其他框架慢，运行时间是最新深度学习工具的四倍，内存占用较大 (如相比 Torch)。支持的层没有 Torch 和 Theano 丰富，特别是没有时间序列的卷积。卷积也不支持动态输入尺寸，这些功能在 NLP 中非常有用；不支持双向 RNN 和 3D 卷积，同时公共版本的图定义也不支持循环和条件控制；RNN 实现不理想，因为必须要使用 Python 循环且无法进行图编译优化。TensorFlow 目前不支持所谓的内联矩阵运算，必须要复制矩阵才能对其进行运算，复制非常大的矩阵会导致成本全面偏高。TensorFlow 计算图纯粹基于 Python，所以速度较慢，也不适合 Java 和 Scala 用户群。TensorFlow 比 Torch 笨重许多，也更难理解，已预定型的模型不多。

Theano (3 pts)

Theano 速度较慢，编译过程慢，但同样采用符号张量图的 TensorFlow 无此问题；启动时间长，缺乏底层接口，Python 解释器很低效，对工业用户缺少吸引力。开发者难改进，因为底层代码是 Python，C/CUDA 代码被打包在 Python 字符串中。原始的 Theano 级别偏低，对已预定型模型的支持不够完善，错误信息可能没有帮助。Theano 大型模型的编译时间可能较长，比 Torch 笨重许多，更难理解。

Torch (3 pts)

Torch 接口为 Lua 语言，通常需要自己编写定型代码 (即插即用相对少)，需要一点时间学习，Torch 目前没有 Python 接口，与 Caffe 一样，基于层的网络结构，扩展性不好，对于新增加的层，需要自己实现。RNN 没有官方支持，不太适合递归神经网络。以图层的方式定义网络，这种粗粒度的方式使得其对新图层类型的扩展缺乏足够支持。Torch 模型运行需要 LuaJIT 的支持，虽对性能影响不大，但对集成造成很大障碍。Torch 吸引力不如 Caffe、CNTK、TensorFlow 等直接支持 C++的框架。

13.概述图

库名称	底层语言	支持接口	硬件	分布式	多核GPU	速度	灵活性	支持网络	平台	网络结构	安装难度	研究与应用领域
Caffe	C++	C++/Python/matlab	CPU/GPU	N	Y	速度快	一般	CNN/DNN	Windows/Linux/Mac	分层方法	容易	视觉计算
CNTK	C++	Python/C/C++	CPU/GPU	Y	Y	简单快速	一般	CNN/RNN/DNN	Windows/Linux	符号张量图	很简单	通用
GraphLab	C++	C++/Java/Python	CPU/GPU	Y	Y	高性能，速度快	比较灵活	CNN/RNN	Windows/Linux/Mac	符号张量图	比较容易	通用
GraphX	Java	Java/C#/Python/Scala	CPU/GPU	Y	Y	有竞争优势	操作灵活	CNN	Windows/Linux/Mac	符号张量图	简单	通用
MXNet	C++	C++/Python/R/Julia/Go	CPU/GPU/Mobile	Y	Y	快	好	CNN/RNN	所有电脑手机系统	符号张量图	中等	图像识别
Petuum	C++	C++/Java	CPU/GPU	Y	Y	高性能，很快	一般	CNN/RNN/DNN	Linux/Ubuntu	符号张量图	中等	通用

库名称	底层语言	支持接口	硬件	分布式	多核GPU	速度	灵活性	支持网络	平台	网络结构	安装难度	研究与应用领域
TensorFlow	C++/Python	C++/C/Python	CPU/GPU/Mobile	Y	Y	中等，慢于 Theano 和 Torch	好	CNN/RNN	Linux/Mac OS X	符号张量图	容易	通用
Theano	Python	Python/C++	CPU/GPU	N	N	快，与 Torch 相当	很灵活	CNN/RNN/DNN	Windows/Linux/Mac	符号张量图	一般	通用
Torch	C++/Lua	Lua	CPU/GPU/FPGA	Y	Y	快，与 Theano 相当	好	CNN/RNN/DNN	Windows/Linux/Mac OS X	分层方法	CenOS难	通用

14.总结

Caffe (3.75 pts)

Caffe 是目前最流行的深度学习框架之一，在计算机视觉领域依然是最流行的工具包。Caffe 有辉煌的历史，但是随着深度学习模型发展的越来越复杂，对灵活性要求越来越高，已经很难适应需求了。

CNTK (4.33 pts)

CNTK 目前热度不高，但是微软在持续投入，最近刚加入了 Python 前端，未来怎么样还不好说。

GraphLab (4.42 pts)

GraphLab 可以运行在多处理机的单机系统、集群或是亚马逊的 EC2 等多种环境下。GraphLab 自成立以来就是一个发展很迅速的开源项目，其用户涉及的范围也相当广泛。

GraphX (4.25 pts)

GraphX 可以认为是 GraphLab (C++) 和 Pregel (C++) 在 Spark (Scala) 上的重写及优化。GraphX 最大的贡献是，在 Spark 之上提供一栈式数据解决方案，可以方便且高效地完成图计算的一整套流水作业。

MXNet (4.67 pts)

MXNet 给人的感觉是非常用心，更加注重高效，文档也非常的详细，不仅上手很容易，运用也非常的灵活。XNet 的代码量小，容易深度定制，同时支持 imperative (类似 Torch) 和 declarative (类似 TF)两种 API，所以更灵活。另外 MXNet 的单机和分布式性能都是很好的。缺点是，相对于 Torch 学习难度会大一些，另外以前没有资金支持，文档和宣传做的比较差。

Petuum (4.46 pts)

Petuum 可使用户从复杂繁琐的分布式系统编程和调试中解脱出来，将更多精力集中在优化模型和算法上。目前主要适用于几十台机器的集群，在更大集群上处理有风险，调度器设计有点复杂。

TensorFLow (4.38 pts)

TensorFlow 则是功能很齐全，能够搭建的网络更丰富而不是像 Caffe 仅仅局限在 CNN，但是单机性能还不够好。TensorFlow 希望做一个大而全的机器学习框架，而不只满足于深度学习。但是大而全的负面因素就是代码量大、抽象层数多、代码冗余，对性能和可定制性都会带来负面影响。可能并不适合需要深度定制、对性能敏感的企业和做前沿研究的 research。

Theano (3.71 pts)

Theano 是深度学习模型的极佳选择，它允许使用者有效地定义、优化和评估涉及多维数组的数学表达式，同时支持 GPU 和高效符号分化操作。Theano 很容易用 Lasagne/Keras 实现新网络或者编辑现存网络，非常适合数据探索和研究活动。

Torch (4.17 pts)

Torch 比较像一个免费且原生支持 GPU 的 Matlab，在学术圈习惯 Matlab 或者不用 Python 的人群中很受欢迎。Torch 的封装少，简单直接，前期学习和开发时的思维难度都比较低。但是由于封装少和 Lua 本身的限制，工程性不好、容易乱写、代码可复用性差，导致 Torch 不适合做大项目的开发。最新版 Torch7 虽然功能强大，但其设计并不适合在两个群体中大范围普及，即 Python 学术界和 Java 工业界。

(注：总结部分的评分=之前所有项评分的均分)