TensorFlow：用于大规模机器学习的系统

简介

TensorFlow 是一种机器学习系统，可在**大规模**和**异构**环境中运行。TensorFlow 使用数据流图来表示计算，共享状态以及改变该状态的操作。它将数据流图的节点映射到集群中的多台机器上，并且单台机器上跨多个计算设备，包括多核 CPU，通用 GPU 和定制设计的 ASIC，称为张量处理单元（TPU）。这种架构为应用程序开发人员提供了灵活性：在以前的“参数服务器”设计中，共享状态的管理内置于系统中，TensorFlow 使开发人员能够尝试新颖的优化和训练算法。 TensorFlow 支持各种应用程序，**重点是深度神经网络**的训练和推理。一些 Google 服务在生产中使用了 TensorFlow，我们已经开始使用它来进行机器学习研究。在本文中，我们描述了 TensorFlow 数据流模型，并展示了 TensorFlow 在几个实际应用中实现的引人注目的性能。

1 介绍

近年来，机器学习推动了许多不同领域的进步。我们将这一成功归功于更复杂的机器学习模型的发明，用于解决这些领域中的问题的大型数据集的可用性，以及软件平台的开发，这些软件平台使得能够容易地使用大量计算资源来训练这些模型。

我们开发了 TensorFlow 系统，用于试验新模型，对大型数据集进行训练，并将其应用到生产。我们的 TensorFlow 基于我们的第一代系统 DistBelief 的多年经验，简化和概括它，使研究人员能够相对轻松地探索更多种类的想法。TensorFlow 支持大规模训练和探索：它有效地使用数百个功能强大（支持 GPU）的服务器进行快速训练，并在各种平台上运行经过训练的模型，用于生产中的探索，范围从数据中心的大型分布式集群到在移动设备上本地运行。同时，它足够灵活，可以支持对新机器学习模型和系统级优化的实验和研究。

TensorFlow 使用统一的数据流图来表示算法中的计算和算法运行的状态。我们从数据流系统的高级编程模型和参数服务器的低效率中汲取灵感。与传统数据流系统不同，其中图顶点表示对不可变数据的功能计算，TensorFlow 允许顶点表示拥有或更新可变状态的计算.Edges 在节点之间携带张量（多维数组），TensorFlow 透明地在分布式之间插入适当的通信。通过在单个编程模型中统一计算和状态管理，TensorFlow 允许程序员尝试不同的并行化方案，例如，将计算移到保持共享状态的服务器上，以减少网络流量。我们还构建了各种协调协议，并通过同步备份实现了令人鼓舞的结果，回应了最近的结果，这些结果与普遍认为可扩展学习需要异步备份的观点相矛盾。

在过去的一年中，Google 的 150 多个团队使用了 TensorFlow，我们将该系统作为开源项目发布。感谢我们庞大的用户社区，我们获得了许多不同机器学习应用程序的经验。在本文中，我们将神经网络训练作为一个具有挑战性的系统问题，并从这中选择两个具有代表性的应用：图像分类和语言建模。这些应用程序分别强调计算吞吐量和聚合模型大小，我们使用它们来演示 TensorFlow 的可扩展性，并评估我们当前实现的效率和可伸缩性。

2 背景和动机

我们首先描述我们以前系统（第 2.1 节）的局限性，并概述我们在 TensorFlow（第 2.2 节）开发中使用的设计原则。

2.1 先前的系统: DistBelief

TensorFlow 是 DistBelief 的继承者，DistBelief 是自 2011 年以来 Google 使用的用于训练神经网络的分布式系统.DistBelief 使用参数服务器架构，在这里我们批评它的局限性，但基于这种架构的其他系统已经解决了这些局限性。我们在 2.3 小节讨论这些系统。

在参数服务器体系结构中，作业包括两个不相交的进程集：在训练模型时执行大量计算的无状态 worker 进程，以及维护模型参数的当前版本的有状态参数服务器进程。 DistBelief 的编程模型类似于 Caffe 的：用户将神经网络定义为层的有向无环图，利用损失函数终止进程。层是数学运算符的组合：例如，完全连接的层将其输入乘以权重矩阵，添加偏置向量，并将非线性函数（例如 sigmoid）应用于结果。损失函数是标量函数，其量化预测值（对于给定输入数据点）与事实值之间的差异。在完全连接的层中，权重矩阵和偏置向量是学习算法将更新的参数，以便最小化损失函数的值。 DistBelief 使用 DAG 结构和层的语义知识，通过反向传播计算每个模型参数的梯度。由于许多算法中的参数更新是可变换的并且具有弱一致性要求，因此工作进程可以独立地计算更新并将“delta”更新写回每个参数服务器，并与其当前状态组合在一起。

虽然 DistBelief 使许多谷歌产品能够使用深度神经网络并形成许多机器学习研究项目的基础，但我们很快就开始感受到它的局限性。其基于 Python 的脚本编写接口，可以用于组合预定义的层，对于具有简单要求的用户来说已经足够了，但是更高级的用户寻求另外三种灵活性：

定义新图层。为了提高效率，我们将 DistBelief 层实现为 C ++类。使用单独的，不太熟悉的编程语言来实现层是机器学习研究人员的障碍，他们试图尝试新的层架构，例如采样的 softmax 分类器和注意模块。

完善训练算法。使用随机梯度下降（SGD）训练许多神经网络，其通过在最大程度地减小损失函数的值的方向上移动它们来迭代地细化网络的参数。对 SGD 的一些改进通过更改更新规则来加速收敛。研究人员经常想要尝试新的优化方法，但在 DistBelief 中执行此操作涉及修改参数服务器实现。此外，参数服务器的 get()和 put()接口并不适用于所有优化方法：在许多情况下，将计算移到到参数服务器上会更有效，从而减少网络流量。

定义新的训练算法。 DistBelief 工作人员遵循固定的执行模式：读取一批输入数据和当前参数值，计算损失函数（通过网络的正向传递），计算每个参数的梯度（向后传递），并写入渐变回到参数服务器。这种模式适用于训练简单的前馈神经网络，但不适用于更高级的模型，例如包含循环的递归神经网络;对抗性网络，其中两个相关网络交替训练;和强化学习模型，其中损失函数由一些 agent 在一个单独的系统中计算，例如视频游戏模拟器。此外，还有许多其他机器学习算法，例如期望最大化，决策森林训练和潜在 Dirichlet 分配，不适合与神经网络相同的训练模型，但也可以从一个通用的，优化良好的分布式运行环境受益。

此外，我们将 DistBelief 设计为单一平台：大型分布式多核服务器集群。我们能够增加对 GPU 加速的支持，当时很明显这种加速对于有效执行卷积内核至关重要，但 DistBelief 仍然是一个重量级系统，适用于在**大型数据集**上训练深度神经网络，并且很难缩小到其他环境。特别是，许多用户希望在本地 GPU 驱动的工作站上完善他们的模型，然后将相同的代码扩展到更大的数据集上进行训练。在集群上训练模型之后，下一步是将模型推向生产，这可能涉及将模型集成到在线服务中，或将其部署到移动设备上以进行离线执行。这些任务中的每一个都有一些共同的计算结构，但我们的同事发现有必要使用或创建满足每个平台不同性能和资源要求的独立系统。TensorFlow 为所有这些环境提供单一编程模型和运行时系统。

2.2 设计原则

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                 # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                  # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

Figure 1:使用 TensorFlow 的 Python API 编写的图像分类器。该程序是 MNIST 数字分类问题（具有 784 像素图像和 10 个输出类）的简单解决方案。

我们设计的 TensorFlow 比 DistBelief 更灵活，同时保留了满足 Google 生产机器学习工作负载需求的能力。TensorFlow 提供了一种简单的基于数据流的编程抽象，允许用户在分布式集群，本地工作站，移动设备和定制设计的加速器上部署应用程序。高级脚本接口（Figure 1）包含数据流图的构造，使用户能够在不修改核心系统的情况下尝试不同的模型架构和优化算法。在本小节中，我们将简要介绍 TensorFlow 的核心设计原则：

原语运算符的数据流图。 TensorFlow 和 DistBelief 都使用数据流表示模型，但最显着的区别是 DistBelief 模型包含相对较少的复杂“层”，而相应的 TensorFlow 模型表示单独的数学运算符（如矩阵乘法，卷积等）作为数据流图中的节点。这种方法使用户可以更轻松地使用高级脚本接口组合新层。许多优化算法要求每个层都有定义的梯度，而使用简单的算子构建层可以很容易地自动区分这些模型（§4.1）。除了函数运算符之外，我们还表示可变状态以及更新它的操作，作为数据流图中的节点，从而实现了不同更新规则的操作。

延迟执行。典型的 TensorFlow 应用程序有两个不同的阶段：第一阶段定义程序（例如，要训练的神经网络和更新规则）作为符号数据流图，包括输入数据的占位符和表示状态的变量；第二阶段在可用设备集上执行程序的优化版本。通过推迟执行，TensorFlow 可以通过使用有

关计算的全局信息来优化执行阶段。虽然这种设计选择使执行更加高效，但我们不得不将更复杂的功能（如动态控制流）推送到数据流图中，以便使用这些功能的模型享受相同的优化。

异构加速器的通用抽象。除了多核 CPU 和 GPU 等通用设备外，用于深度学习的专用加速器可以显着提高性能并节省功耗。在 Google，我们的同事已经建立了专门用于机器学习的 Tensor Processing Unit（TPU）;与替代最先进的技术相比，TPU 每瓦性能提高一个数量级。为了在 TensorFlow 中支持这些加速器，我们为设备定义了一个通用的抽象。设备必须至少实现以下方法：（i）提供内核运行，（ii）为输入和输出分配内存储存，以及（iii）将缓冲区传送到主机存储器和从主机存储器传送缓冲区。每个运算符（例如，矩阵乘法）可以具有针对不同设备的多个专用实现。因此，相同的程序可以根据训练，服务和离线推理的需要轻松定位 GPU，TPU 或移动 CPU。

TensorFlow 使用原始值的张量作为所有设备都能理解的通用交换格式。在最低级别，TensorFlow 中的所有张量都是密集的;稀疏张量可以用密集张量表示（§3.1）。此决策确保系统的最低级别具有用于内存分配和序列化的简单实现，从而减少了框架开销。Tensors 还支持其他内存管理和通信优化，例如 RDMA 和直接 GPU 到 GPU 传输。

这些原则的主要结果是在 TensorFlow 中没有参数服务器这样的东西。在集群中，我们将 TensorFlow 部署为一组任务（可以通过网络通信的命名进程），每个任务都导出相同的图执行 API 并包含一个或多个设备。通常，这些任务的子集承担参数服务器在其他系统中扮演的角色，因此我们称之为 PS 任务;其他则是 worker 任务。但是，由于 PS 任务能够运行任意 TensorFlow 图形，因此它比传统参数服务器更灵活：用户可以使用与定义模型相同的脚本接口对其进行编程。这种灵活性是 TensorFlow 与现代系统之间的关键区别，本文的其余部分将讨论这种灵活性所能实现的一些应用。

2.3 相关工作

单机框架。许多机器学习研究人员在一台通常配备 GPU 的计算机上执行他们的工作，并且一些单机框架支持这种情况。**Caffe** 是一个高性能框架，用于在多核 CPU 和 GPU 上训练声明的神经网络。如上所述，其编程模型类似于 DistBelief（第 2.1 节），因此很容易从现有图层组合模型，但**添加新图层或优化器相对困难**。 Theano 允许程序员将模型表示为原语操作符的数据流图，并生成用于训练该模型的高效编译代码。它的编程模型最接近 TensorFlow，并且它在单个机器中提供了大部分相同的灵活性。

与 Caffe，Theano 和 TensorFlow 不同，Torch 为科学计算和机器学习提供了强大的命令式编程模型。它允许对执行顺序和内存利用率进行精细控制，使高级用户能够优化其程序的性能。虽然这种灵活性对研究很有用，但 Torch 缺乏数据流图作为小规模实验，生产训练和部署可移植表示的优势。

批量数据流系统。从 MapReduce 开始，批量数据流系统已应用于大量机器学习算法，而最近的系统则专注于提高表达性和性能。DryadLINQ 添加了一种高级查询语言，支持比 MapReduce 更复杂的算法。Spark 扩展了 DryadLINQ，能够将先前计算的数据集缓存在内存中，因此当输入数据满足内存时，它更适合迭代机器学习算法（例如 k 均值聚类和逻辑回归）。Dandelion 通过 GPU 和 FPGA 的代码生成扩展了 DryadLINQ。

批处理数据流系统的主要**限制**是它要求**输入数据是不可变**的，并且**所有子计算都是确定性**的，以便系统可以在集群中的机器发生故障时重新执行子计算。该特征对于许多传统工作负载是有益的，使得更新机器学习模型成为昂贵的操作。例如，用于在 Spark 上训练深度神经网络的 SparkNet 系统需要 20 秒来广播权重并从五名 worker 中收集更新。因此，在这些系统中，每个模型更新步骤必须处理更大批量，从而减慢收敛速度。我们在 6.3 小节中表明，TensorFlow 可以在更大的集群上训练更大的模型，步长时间短至 2 秒。

参数服务器。正如我们在 2.1 小节中讨论的那样，参数服务器体系结构使用一组服务器来管理共享状态，这些状态由一组并行 worker 程序更新。这种架构出现在**可扩展主题建模**的工作中，DistBelief 展示了它如何应用于深度神经网络训练。Adam 项目进一步应用该架构进行卷积神经网络的有效训练;和 Li 等人的“参数服务器”在一致性模型，容错和弹性重新缩放方面增加了创新。尽管早先猜测参数服务器与 GPU 加速兼容，但 Cui 等人，最近表明，专门用于 GPU 的参数服务器可以实现小型集群的加速。

MXNet 可能是与 TensorFlow 设计最相似的系统。它使用数据流图表示每个 worker 计算，并使用参数服务器在多台计算机上扩展训练。MXNet 参数服务器导出**键值**存储接口，该接口支持聚合从每个 worker 中的多个设备发送的更新，并使用任意用户提供的函数将传入更新与当前值组合.MXNet 键值存储接口当前不允许使用单个值对稀疏梯度进行更新，这对于大型模型的分布式训练至关重要（§4.2），并且添加此功能需要修改核心系统。

参数服务器体系结构满足了我们的许多要求，并且通过充分的工程工作，可以将本文中描述的大多数功能构建到参数服务器中。对于 TensorFlow，我们寻求一种高级编程模型，允许用户自定义在系统所有部分中运行的代码，因此使用新的优化算法和模型体系结构的实验成本较低。在下一节中，我们将描述更详细地介绍 TensorFlow 程序的构建块。

3 Tensorflow 计算模型

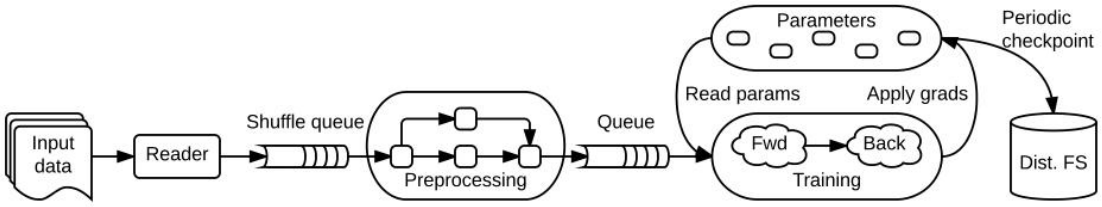

Figure 2:用于训练管道的 TensorFlow 数据流图，包含用于读取输入数据，预处理，训练和检查点状态的子图。

TensorFlow 使用单个数据流图来表示机器学习算法中的所有计算和状态，包括各个数学运算，参数及其更新规则以及输入预处理（图 2）。数据流图明确表示子计算之间的通信，因此可以轻松并行执行独立计算并跨多个设备分区计算。TensorFlow 与批处理数据流系统（第 2.3 节）的不同之处在于两个方面：

- 模型支持在整个图的重叠子图上进行多个并发执行。
- 各个顶点可能具有可变状态，可以在图的不同执行之间共享。

参数服务器体系结构中的**关键**考察点是，在训练非常大的模型时，可变状态是至关重要的，因为可以对非常大的参数进行就地更新，并且尽可能快地将这些更新传播到并行训练步骤。具有可变状态的数据流使 TensorFlow 能够模仿参数服务器的功能，但具有额外的灵活性，因为可以在拥有共享模型参数的机器上执行任意数据流子图。因此，我们的用户已经能够尝试不同的优化算法，一致性方案和并行化策略。

3.1 数据流图元素

在 TensorFlow 图中，每个顶点表示局部计算的单位，每个边表示顶点的输出或输入。我们将**顶点处的计算**称为**操作**，并将**沿着边缘流动的值**称为张量。在本小节中，我们描述了常见的操作类型和张量。

Tensors。在 TensorFlow 中，我们将所有数据建模为张量（n 维数组），其中元素具有少量基本类型，例如 int32，float32 或 string（其中 **string 可以表示任意二进制数据**）。张量自然地代表了许多机器学习算法中常见数学运算的输入和结果：例如，矩阵乘法需要两个 2-D 张量并产生 2-D 张量;并且批量 2-D 卷积需要两个 4-D 张量并产生另一个 4-D 张量。

在最低级别，由于我们在第 2.2 小节中讨论的原因，所有 TensorFlow 张量都很密集。TensorFlow 提供了两种表示稀疏数据的方法：将数据编码为密集张量的可变长度字符串元素，或者使用密集张量的元组（例如，具有 m 个非零元素的 n 维稀疏张量可以用坐标表示 - 列表格式为 m×n 坐标矩阵和长度为 m 的向量值）。张量的形状可以在其一个或多个维度上变化，这使得可以表示具有不同数量的元素的稀疏张量。

Operations。一个操作将 m≥0 个张量作为输入，并产生 n≥0 个张量作为输出。操作具有命名的“类型”（例如 Const，MatMul 或 Assign），并且可能具有零个或多个确定其行为的编译时属性。一个操作在编译时可以是多态的和可变的：它的属性决定了它的输入和输出的预期类型和数量。

例如，最简单的操作 Const 没有输入只有单个输出;它的值是一个编译时属性。例如，AddN 对相同元素类型的多个张量求和，并且它具有定义其类型签名的类型属性 T 和整数属性 N.

Stateful operations: variables。操作可以包含每次执行时读取或写入的可变状态。**variable** 操作拥有可变缓冲区，该缓冲区可用于在训练模型时存储模型的共享参数。变量没有输入，会产生一个引用句柄，它作为用于读取和写入缓冲区的功能。Read 操作将引用句柄 r 作为输入，并将变量（State [r]）的值作为密集张量输出。其他操作修改底层缓冲区：例如，AssignAdd 采用引用句柄 r 和张量值 x，并且在执行时执行更新 State'[r]←State [r] + x。后续的 Read(r) 操作产生值 State'[r]。

Stateful operations: queues。TensorFlow 包含多个队列实现，支持更高级的协调形式。最简单的队列是 FIFOQueue，它拥有一个内部的张量队列，并允许以先进先出顺序进行并发访问。其他类型的队列以随机和优先级顺序对张量进行出队，这确保了输入数据的适当采样。与 **variable** 一样，FIFOQueue 操作生成一个引用句柄，可以由标准队列操作使用，例如 Enqueue 和 Dequeue。这些操作将其输入推送到队列的尾部，并分别弹出 head 元素并输出它。如果给定**队列已满**，则 **Enqueue 将阻塞**，如果给定**队列为空**，则 **Dequeue 将阻塞**。当在预处理输入管道中使用队列时，此阻塞提供**背压**;它还支持同步（§4.4）。队列和动态控制流（第

3.4 节）的组合也可以实现子图之间的流式计算形式。
（背压参考 https://www.cnblogs.com/iceTing/p/6238207.html）

3.2 局部和并发执行

TensorFlow 使用数据流图表示特定应用程序中的所有可能计算。用于执行图的 API 允许客户端以声明方式指定应该执行的子图。客户端选择**零个或多个**边缘以将输入张量馈送到数据流中，并选择**一个或多个**边缘以从数据流中获取输出张量;运行时然后修剪图以包含必要的操作集。每次调用 API 都称为步骤，TensorFlow 支持同一图形上的多个并发步骤。有状态操作允许各执行步骤中共享数据并在必要时进行同步。

Figure 2 显示了一个典型的训练应用程序，其中有多个子图同时执行并通过共享变量和队列进行交互。训练子图的核心依赖于一组模型参数和队列中的输入批次。训练子图的许多并发步骤基于不同的输入批次更新模型，以实现数据并行训练。为了填充输入队列，并发预处理步骤转换各个输入记录（例如，解码图像和应用随机失真），并且单独的 I / O 子图从分布式文件系统读取记录。检查（点）子图定期运行以获得容错（§4.3）。

局部和并发执行是 TensorFlow 灵活性的主要原因。通过队列添加可变状态和协调，使得可以在用户级代码中指定各种模型体系结构，这使高级用户无需修改 TensorFlow 运行环境的内部即可进行实验。默认情况下，TensorFlow 子图的并发执行以异步方式运行，这种异步使得实现具有弱一致性要求的机器学习算法变得简单，其中包括许多神经网络训练算法。正如我们稍后讨论的那样，TensorFlow 还提供了在训练期间同步 worker 所需的原语（§4.4），这导致了一些学习任务不错的结果（§6.3）。

3.3 分布式执行

Dataflow 简化了分布式执行，因为它使子计算间形成明确的通信。它使相同的 TensorFlow 程序能够部署到用于训练的 GPU 集群，用于服务的 TPU 集群和用于移动推测的手机。

每个操作分配在特定设备上，例如特定任务中的 CPU 或 GPU。设备负责为分配给它的每个操作执行内核操作.TensorFlow 允许为单个操作申请多个内核，用来对具有特定设备或数据类型的专用实现（有关详细信息，请参见§5）。对于许多操作，例如元素操作符（Add，Sub 等），我们可以使用不同的编译器为 CPU 和 GPU 编译单个内核实现。

TensorFlow 运行时将操作放置在设备上，受图中的隐式或显式约束。放置算法为每个操作找出一组满足运行的设备，计算必须共置的操作集，并为每个共置组选择满意的设备。它遵循隐式共置的约束，因为每个有状态操作及其状态必须放在同一设备上。另外，用户可以指定部分设备的优先权，诸如“特定任务中的任何设备”或“任何任务中的 GPU”，并且运行时将遵守这些约束。典型的训练应用程序将使用客户端编程构造来添加约束，例如，参数分布在一组”PS”任务中（第 4.2 节）。

因此，TensorFlow 可以灵活地将数据流图中的操作映射到设备。虽然简单的启发式方法可以为新手用户提供足够的性能，但专家用户可以通过手动放置操作来平衡多个任务和这些任务中的多个设备的计算，内存和网络要求来优化性能。一个悬而未决的问题是 TensorFlow 如何自动确定在给定设备集上实现接近最佳性能的放置位置，从而使用户免于这种担忧。即使没有这样的自动化，将放置指令与模型定义的其他方面分开也可能是值得的，因此，例如，

在训练模型之后修改放置指令将影响甚微。

一旦图形中的操作被放置了，并且已经为步骤（§3.2）计算了部分子图，TensorFlow 就将操作划分为每个设备子图。设备 d 的单个设备子图包含分配给 d 的所有操作，以及跨设备边界替换边的附加 Send 和 Recv 操作。只要张量可用，发送就会将其单个输入发送到指定的设备，使用集合键命名该值。Recv 具有单个输出，并在生成该值之前阻塞，直到指定的集合键的值在本地可用。Send 和 Recv 具有多个设备类型对的专用实现;我们在第 5 节中描述了其中一些。

我们优化了 TensorFlow，以低延迟重复执行大型子图。一旦图的步骤被修剪，放置和分区，其子图就会缓存在各自的设备中。客户端 session 维护从步骤定义到缓存子图的映射，以便可以通过向每个参与任务发送一条小消息来启动大图上的分布式步骤。该模型支持静态，可重用的图形，但它可以使用动态控制流支持动态计算，如下一小节所述。

3.4 动态控制流

```
input = ...   # A sequence of tensors
state = 0     # Initial state
w = ...       # Trainable weights

for i in range(len(input)):
  state, out[i] = f(state, w, input[i])
```

Figure 3:RNN 抽象伪代码（§3.4）。函数 f 通常包括可微运算，例如矩阵乘法和卷积.TensorFlow 在其数据流图中实现循环。

TensorFlow 支持包含条件和迭代控制流的高级机器学习算法。例如，诸如 LSTM 的递归神经网络（RNN）可以从顺序数据生成预测。Google 的神经机器翻译系统使用 TensorFlow 训练深度 LSTM，在许多翻译任务中实现最先进的性能。RNN 的核心是递归关系，其中序列元素 i 的输出是在序列中累积的某种状态的函数（Figure 3）。在这种情况下，动态控制流程允许**对具有可变长度的序列进行迭代**，而不将计算展开到最长序列的长度。

正如我们在 2.2 小节中所讨论的那样，TensorFlow 使用通过数据流图的延迟执行来将更大的工作量传送到加速器。因此，为了实现 RNN 和其他高级算法，我们在数据流图本身中添加了条件（if 语句）和迭代（while 循环）编程结构。我们使用这些原语来构建高阶构造，例如 map()，fold()和 scan()。

为此，我们从经典的动态数据流架构中借用了 Switch 和 Merge 原语。Switch 是一个多路分解器：它接收数据输入和控制输入，并使用控制输入选择其两个输出中的哪一个应产生一个值。未接收的 Switch 输出接收一个特殊的 dead 值，该值以递归方式传递到图的其余部分，直到达到 Merge 操作。Merge 是一个多路复用器：它最多将一个非 dead 输入转发到其输出，或者如果它的两个输入都 dead，则产生一个 dead 输出。条件运算符使用 Switch 根据布尔张量的运行时值执行两个分支之一，并使用 Merge 组合分支的输出。while 循环更复杂，并使用 Enter，Exit 和 NextIteration 运算符来确保循环格式正确。

迭代的执行可以重叠，TensorFlow 也可以跨多个设备和进程对条件分支和循环体进行分区。分区步骤添加逻辑以协调每个设备上每次迭代的开始和终止，并确定循环的终止。正如我们将在 4.1 小节中看到的，TensorFlow 还支持控制流构造的自动区分。自动区分将计算梯度的子图添加到数据流图中，TensorFlow 在可能分布的设备上进行分区，以并行计算梯度。

4 可扩展性案例研究

通过为 TensorFlow 中的所有计算选择统一表示，我们使用户能够试验硬编码到 DistBelief 运行时的功能。在本节中，我们将讨论使用数据流原语和“用户级”代码构建的四个扩展。

4.1 差异化和优化

许多学习算法使用 SGD 的一些变体训练一组参数，这需要计算与这些参数相关的损失函数的梯度，然后基于这些梯度更新参数.TensorFlow 包括用户级库，可以区分损失函数的符号表达式并生成梯度的新符号表达式。例如，给定神经网络作为层和损失函数的组合，库将自动派生反向传播代码。

一些差分算法执行广度优先搜索以识别从目标操作（例如，损失函数）到一组参数的所有后向路径，并对每个路径贡献的部分梯度求和。我们的用户经常专门针对某些梯度进行操作，他们已经实现了批量标准化和梯度限幅等优化，以加速训练并使其更加健壮。我们扩展了算法以区分条件和迭代子计算（第 3.4 节），方法是在图中添加节点，在前向传递中记录控制流决策，并在反向传递期间反向重放这些决策。在长序列上区分迭代计算可以导致在存储器中累积大量中间状态，并且我们已经开发了用于在这些计算上管理有限 GPU 存储器的技术。

TensorFlow 用户还可以尝试各种优化算法，这些算法可为每个训练步骤中的参数计算新值。SGD 易于在参数服务器中实现：对于每个参数 W，梯度 ∂ L/ ∂ W 和学习率 α ，更新规则为 W'←W - α × ∂ L/ ∂ W。参数服务器可以通过使用 - =作为写操作来实现 SGD，并且在训练步骤之后将 α × ∂ L/ ∂ W 写入每个 W。

但是，有许多更高级的优化方案难以表达为单个写操作。例如，Momentum 算法基于多个迭代的梯度为每个参数累积“速度”，然后根据该累积计算参数更新;并且已经提出了对该算法的许多改进。在 DistBelief 中实现 Momentum，需要修改参数服务器实现以更改参数数据的表示，并在写操作中执行复杂逻辑;这些修改对许多用户来说都是挑战。优化算法是积极研究的主题，研究人员在 TensorFlow 上实现了几个，包括 Momentum，AdaGrad，AdaDelta，RMSProp，Adam 和 L-BFGS。这些可以使用变量操作和原始数学运算在 TensorFlow 中构建，而无需修改底层系统，因此很容易在新算法出现时进行实验。

4.2 训练非常大的模型

为了训练高维数据的模型，例如文本语料库中的单词，通常使用分布式表示，将训练示例嵌入跨越若干神经元的活动体中，并且可以通过反向传播来学习。例如，在语言模型中，训练示例可以是稀疏矢量，其中非零条目对应于词汇表中的单词的 ID，并且每个单词的分布式表示将是较低维向量。“广泛而深入的学习”通过对分类特征的跨产品转换创建分布式表示，并且 TensorFlow 上的实现用于为 Google Play 应用商店推荐系统提供支持。

推断开始于将一批 b 个稀疏矢量与 n×d 嵌入矩阵相乘，其中 n 是词汇表中的单词数，d 是

期望的维数，以产生小得多的 b×d 密集矩阵表示;对于训练，大多数优化算法仅修改由稀疏乘法读取的嵌入矩阵的行。在处理稀疏数据的 TensorFlow 模型中，n×d 可以达到千兆字节的参数：例如，大型语言模型可以使用超过 10^9 个参数，词汇量为 800,000 个单词，并且我们有文档模型的经验，其中参数占用几兆兆字节这样的模型太大了，无法在每次使用时复制到 worker 中，甚至无法存储在单个主机上的 RAM 中。

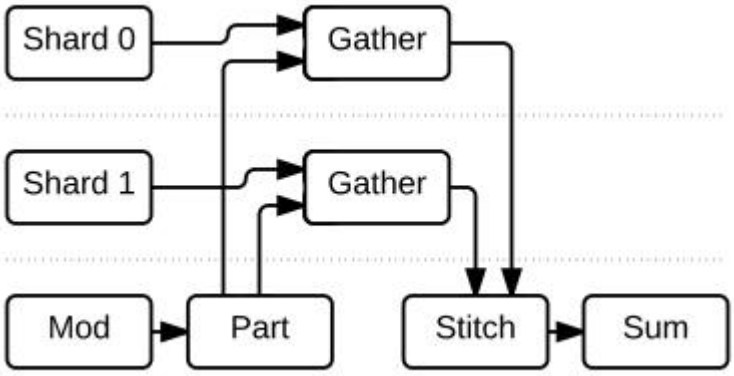

Figure 4:具有双边分片嵌入矩阵的嵌入层（第 4.2 节）的原理图数据流。

我们在 TensorFlow 图中实现稀疏嵌入层作为基本操作的组合。Figure 4 显示了嵌入层的简化图，该嵌入层分为两个参数服务器任务。此子图的核心操作是 Gather，它从张量中提取稀疏的数据，TensorFlow 将此操作的变量与其操作的变量存放一起。动态分区（Part）操作将传入的索引划分为可变大小的张量（包含指定给每个分片的索引），动态拼接（Stitch）操作将每个分片的部分结果重新组合成单个结果张量。这些操作中的每一个都具有相应的梯度，因此它支持自动区分（第 4.1 节），结果是一组稀疏矩阵更新操作，这些操作仅对最初从每个分片收集的值起作用。

编写 TensorFlow 模型的用户通常不会手动构建如图 4 所示的图形。相反，TensorFlow 包含暴露分片参数抽象的库，并根据所需的分布程度构建原始操作的适当计算图。

虽然参数服务器中可以进行稀疏读取和更新，但 TensorFlow 可以灵活地将任意计算移到承载共享参数的设备上。例如，分类模型通常使用 softmax 分类器，该分类器将最终输出乘以具有 c 列的权重矩阵，其中 c 是可能类的数量;对于语言模型，c 是词汇量的大小，可以很大。我们的用户已经尝试了几种加速 softmax 计算的方案。第一个类似于 Project Adam 中的优化，其中权重在几个任务中被分片，并且乘法和梯度计算与分片共同使用。使用采样 softmax 可以进行更有效的训练，softmax 基于以下方式执行稀疏乘法。示例的 true 类和一组随机抽样的 false 类。我们在 § 6.4 中比较了这两种方案的性能。

4.3 容错

即使使用大量机器，训练模型也可能需要几个小时或几天。我们经常需要使用非专用资源来训练模型，例如使用 Borg 集群管理器，这不能保证在训练过程期间相同资源可以一直使用。因此，长期运行的 TensorFlow 作业可能会出现故障或抢占，我们需要某种形式的容错。任务不可能经常失败，不需要对每个操作进行容错，因此像 Spark 的 RDD 这样的机制会带来很大的开销，几乎没有什么好处。不需要对参数状态的每次写入都持久化，因为我们可以从输

入数据开始重新计算任何更新，并且**许多学习算法不需要强一致性**。

我们使用图中的两个操作实现用户级别的容错检查点（Figure 2）：Save 将一个或多个张量写入检查点文件，Restore 从检查点文件读取一个或多个张量。我们的典型配置将任务中的所有 Variable 连接到同一个 Save 操作，每个任务一个 Save，以最大化分布式文件系统的 I / O 带宽。 Restore 操作从文件中读取命名的张量，标准的 Assign 将已还原的值存储在其各自的变量中。在训练期间，典型的客户端定期运行所有 Save 操作以生成新的检查点;当客户端启动时，它会尝试恢复最新的检查点。

TensorFlow 包含一个客户端库，用于构建适当的图形结构以及根据需要调用“保存和还原”。此行为是可自定义的：用户可以将不同的策略应用于模型中 variable 的子集，或自定义检查点保留方案。例如，许多用户在自定义评估指标中保留得分最高的检查点。实现也是可重用的：它可以用于模型微调和无监督预训练，它们是迁移学习的形式，其中在一个任务上训练的模型的参数（例如，识别一般图像）被用作另一项任务的起点（例如，识别狗的品种）。将检查点和参数管理作为图形中的可编程操作，使用户可以灵活地实现这些以及我们未预料到的其他方案。

检查点库不会尝试生成一致性的检查点：如果同时执行训练和检查点，则检查点可以包括训练步骤中的全部或部分更新值。此行为与异步 SGD 的宽保证兼容。一致的检查点需要额外的同步，以确保更新操作不会干扰检查点;如果需要，可以在下一小节中使用该方案在同步更新步骤之后采用检查点。

4.4 副本协调同步

SGD 支持异步操作，并且许多系统使用异步参数更新来训练深度神经网络，这被认为是可扩展的。增加的吞吐量是以在训练步骤中使用旧参数值为代价的。有些人最近重新考虑了同步训练不能扩展的假设。由于 GPU 支持数百台而不是数千台机器的训练，因此同步训练可能比同一平台上的异步训练更快（就质量而言）。

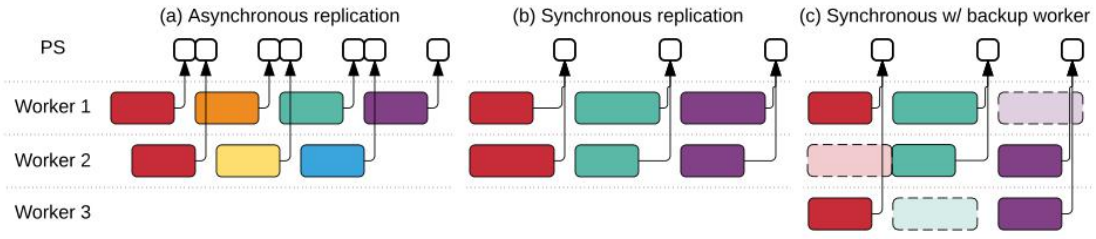

Figure 5:并行 SGD 的三种同步方案。每种颜色代表不同的起始参数值;白色方块是参数更新。在(c)中，虚线矩形表示其结果被丢弃的备份 worker。

虽然我们最初为异步训练设计了 TensorFlow，但我们已经开始尝试使用同步方法。TensorFlow 计算图使用户能够在训练模型时更改参数的读取和写入方式，并且我们实现了三种替代方案。在异步情况下（Figure 5(a)），每个 worker 在每个步骤开始时读取参数的当前值，并在最后将其梯度应用于（可能不同的）当前值：此方法确保高利用率，但每个步骤使用旧的参数值，使每个步骤效果较差。我们使用队列（第 3.1 节）实现同步版本来协调执行：阻塞队列充当障碍以确保所有 worker 读取相同的参数值，并且每个变量队列累积来自**所有**

worker 的梯度更新以便应用它们。简单的同步版本（Figure 5(b)）在应用之前会累积所有 worker 的更新，但是速度慢的 worker 会限制整体吞吐量。

为了缓解速度慢的 worker 的影响，我们实现了备份 worker（Figure 5(c)），这与 MapReduce 备份任务类似。而 MapReduce 反应性地启动备份任务 - 在检测到落后者之后 - 我们的备份 worker 主动运行，并且聚合将产生 n 个更新中的前 m 个。我们利用 SGD 在每一步中随机训练数据的事实，因此每个 worker 处理不同的随机批次，如果忽略特定批次没有问题。在 § 6.3 中，我们展示了备份 worker 如何将吞吐量提高多达 10％。

5 实现

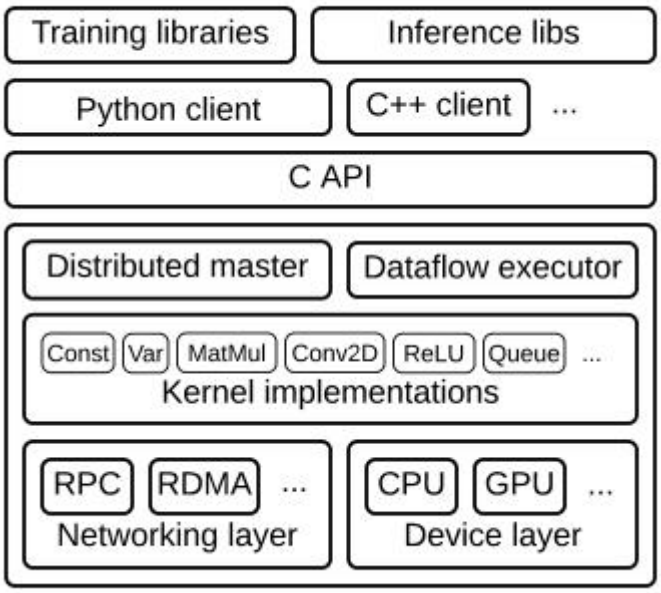

Figure 6:Tensorflow 层次结构

TensorFlow 运行机制是一个跨平台库。Figure 6 说明了它的体系结构：C API 将不同语言的用户级代码与核心运行机制分开。

核心 TensorFlow 库以 C ++实现，具有可移植性和性能：它可在多个操作系统上运行，包括 Linux，Mac OS X，Windows，Android 和 iOS; x86 和各种基于 ARM 的 CPU 架构;和 NVIDIA 的 Kepler，Maxwell 和 Pascal GPU 微体系结构。实现是开源的，我们已经接受了几个外部贡献，使 TensorFlow 能够在其他架构上运行。

分布式 master 将用户请求转换为跨任务执行。给定计算图和步骤定义，它修剪（§ 3.2）和分区（§ 3.3）图以获得每个参与设备的子图，并缓存这些子图，以便它们可以在后续步骤中重复使用。由于 master 看到步骤的整体计算，它应用标准优化，例如公共子表达式消除和常量折叠;修剪是一种消除 dead 代码的形式。然后，它协调一组任务中优化子图的执行。

每个任务中的**数据流执行器**处理来自 master 的请求，并调度包含本地子图的内核操作。我们优化了数据流执行器，以便以较低的开销运行大型图。我们当前的实现可以每秒执行

10,000 个子图（第 6.2 节），这使得大量副本能够进行快速，细粒度的训练。数据流执行器给本地设备分配内核并在可能的情况下并行运行内核，例如通过使用多个 CPU 核心或 GPU 流。

运行机制包含 200 多个标准操作，包括数学，数组操作，控制流和状态管理操作。许多操作内核都是使用 Eigen :: Tensor 实现的，它使用 C ++模板为多核 CPU 和 GPU 生成高效的并行代码;但是，我们自由地使用像 cuDNN 这样的库，可以实现更高效的内核实现。我们还实现了量化，可以在移动设备和高吞吐量数据中心应用等环境中实现更快的训练推断，并使用 gemmlowp 低精度矩阵库来加速量化计算。

我们专门针对每对源设备和目标设备类型进行 Send 和 Recv 操作。本地 CPU 和 GPU 设备之间的传输使用 cudaMemcpyAsync() API 来重叠计算和数据传输;两个本地 GPU 之间的传输使用 DMA 来减轻主机的压力。对于任务之间的传输，TensorFlow 使用多种协议，包括基于 TCP 的 gRPC 和基于融合以太网的 RDMA。我们还在研究使用集体操作的 GPU 到 GPU 通信的优化。

第 4 节描述了我们在用户级代码中完全在 C API 之上实现的功能。通常，用户组成标准操作来构建更高级别的抽象，例如神经网络层，优化算法（§4.1）和分片嵌入计算（§4.2）。TensorFlow 支持多种客户端语言，我们优先考虑 Python 和 C ++，因为我们的内部用户最熟悉这些语言。随着功能变得更加成熟，我们通常将它们移植到 C ++，以便用户可以从所有客户端语言访问优化的实现。

如果将子计算表示为操作组合是困难或低效的，则用户可以注册提供用 C ++编写的有效实现的其他内核。我们发现，为一些性能关键操作手工实现融合内核是有利的，例如 ReLU 和 Sigmoid 激活函数及其相应的梯度。我们目前正在使用基于编译的方法研究自动内核结合。

除了核心运行机制，我们的同事还构建了几个可以帮助 TensorFlow 用户的工具。这些包括服务生产推理的基础设施，使用户能够跟踪训练运行进度的可视化仪表板，帮助用户理解模型中连接的图形可视化工具。我们在扩展白皮书中描述了这些工具。

6 评估

在本节中，我们将评估 TensorFlow 在多个合成和实际工作负载上的性能。
在本文中，我们关注系统性能指标，而不是像精确时间那样学习目标。TensorFlow 是一个允许机器学习从业者和研究员尝试新技术的系统，该评估表明系统（i）的开销很小，（ii）可以采用大量计算来加速实际应用。

7 结论

我们已经描述了 TensorFlow 系统及其编程模型。TensorFlow 的数据流表示包含参数服务器系统上的现有功能，并提供一组统一的抽象，这些抽象允许用户利用大规模异构系统，该系统既可用于生产任务，也可用于尝试新方法。我们已经展示了 TensorFlow 编程的几个示例模型促进了实验（§4）并证明了最终的实现是高效的和可扩展的（§6）。

我们对 TensorFlow 的初步经验令人鼓舞。Google 的大量团队已经在生产中部署了 TensorFlow，TensorFlow 正在帮助我们的研究同事在机器学习方面取得新进展。自从我们发布 TensorFlow 作为开源软件以来，已有超过 14,000 人分享了源代码库，已下载超过一百万次，并且已发布了数十种使用 TensorFlow 的机器学习模型。

TensorFlow 正在进行中。其灵活的数据流表示使高级用户能够实现卓越的性能，但我们尚未确定适用于所有用户的默认策略。对自动优化的进一步研究应该弥补这一差距。在系统层面，我们正在积极开发自动放置，内核结合，内存管理和调度的算法。虽然可变状态和容错的当前实现足以满足弱一致性要求的应用程序，但我们期望一些 TensorFlow 应用程序需要更强的一致性，我们正在研究如何在用户级别构建此类策略。最后，一些用户开始厌倦**静态数据流图的局限性**，特别是对于深度强化学习等算法。因此，即使计算结构动态展开，我们也面临着提供透明有效地使用分布式资源系统的问题。