论文分享

论文：TensorOpt: Exploring the Tradeoffs in Distributed DNN Training with AutoParallelism

背景

针对 DNN 分布式训练问题，目前常见的分布式方案有

1. **数据并行**

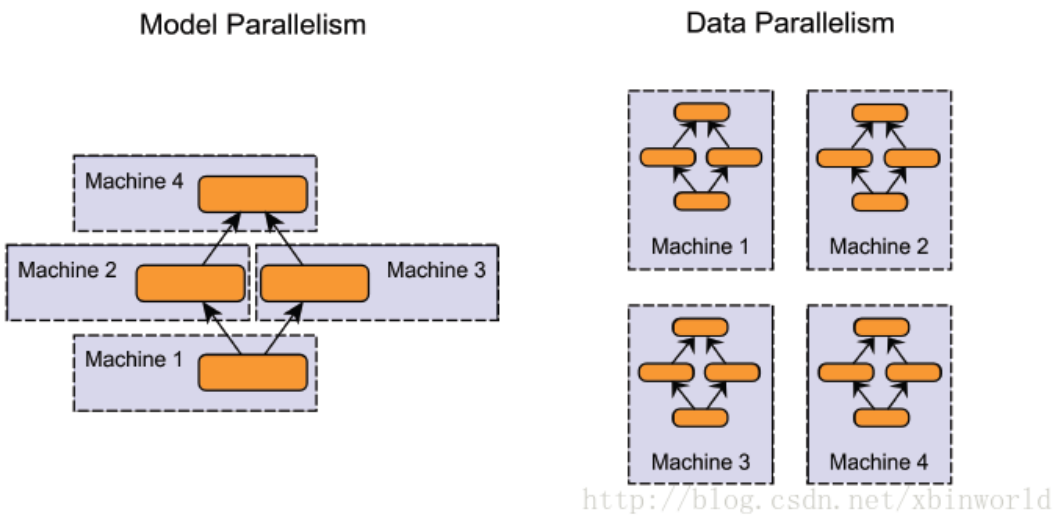
   不同的机器有同一个模型的多个副本，每个机器分配到不同的数据，然后将所有机器的计算结果按照某种方式合并。

   缺点：在大参数层，如全连接层中数据并行效果不好

2. **模型并行**

   分布式系统中的不同机器（GPU/CPU 等）负责网络模型的不同部分。例如，DNN 的不同网络层被分配到不同的机器，或者同一层内部的不同参数被分配到不同机器；

   缺点：受卷积层通信成本的影响



http://blog.csdn.net/xbinworld

3. **自动并行：Auto-Parallel**

   通过设计的算法搜索大量可能的并行化策略。

   OptCNN :使用基于图搜索的动态规划来最小化卷积神经网络 (CNN) 训练的执行时间。

   FlexFlow：用蒙特卡洛马尔科夫链（MCMC）来搜索数据并行和模型并行策略

不足：

现有工作仅优化单一目标（执行时间或内存消耗），这导致适应不同场景的灵活性有限。例如，当使用少量设备训练大型模型时，简单地最小化执行时间的方案可能会导致内存溢出。其次最小化内存消耗的方法不能充分利用额外的内存资源来减少执行时间。

目标

在分布式训练的某些情况下，跟踪不同目标之间的权衡非常重要。
例如，当使用不同数量的资源（例如内存和设备数量）时，获取训练 job 的最短执行时间可以帮助我们在共享 GPU 集群中做出资源分配的决策。
在云上训练 DNN 时，用户需要知道成本（即资源）和效率（即执行时间）之间的权衡，以确定所需的资源量。
因此，**自动并行算法应该足够灵活，能够根据特定场景和用户偏好（在成本-效率权衡上）找到并行化策略，而不是优化单个目标。**

方法

成本边界

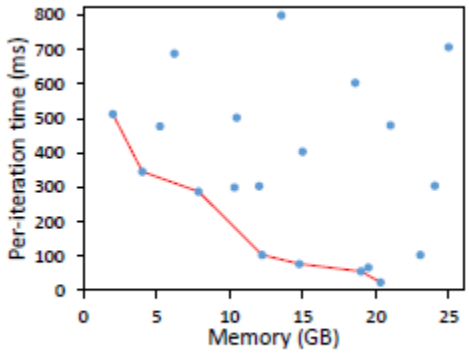


Figure 2: An illustration of cost frontier

假设成本边界是一组最小的并行化策略 F。其中对于给定的任何并行化策略 S，F 中存在一个策略，该策略在每个维度上的成本（例如，执行时间、内存消耗）都小于或等于 S。也就说成本边界之外的并行化策略不需要考虑。
如图所示，红线为成本边界，其中图中的每个点是一个策略元组。成本边界 F 的定义如下

DEFINITION 1. *Let* $\mathcal{C} = \{(S_1, m_1, t_1), (S_2, m_2, t_2), \cdots, (S_K, m_K, t_K)\}$ *be a set of (partial) parallelization strategy tuples, where* S_k *is a (partial) parallelization strategy,* m_k *and* t_k *are the execution time and memory consumption of* S_k*, for* $1 \leq k \leq K$*. The cost frontier of* $\mathcal{C}$ *is the minimum subset* $\mathcal{F}$ *of* $\mathcal{C}$ *such that, for any strategy* $(S_p, m_p, t_p) \in \mathcal{C}$*, there exists a strategy* $(S_k, m_k, t_k) \in \mathcal{F}$ *where* $m_k \leq m_p$ *and* $t_k \leq t_p$*.*

边界追踪-FT 算法

如果通过蛮力搜索寻找成本边界的复杂性是指数的，因此此文设计了 FT 算法，其采用动态规划程序，以实现高效的成本边界跟踪。

FT 算法依靠以下基本操作来控制成本边界

Given two cost frontiers (or two sets of strategy tuples),

$$\mathcal{F} = \{(S_1, m_1, t_1), (S_2, m_2, t_2), ..., (S_K, m_K, t_K)\},$$
$$\mathcal{F}' = \{(S'_1, m'_1, t'_1), (S'_2, m'_2, t'_2), ..., (S'_{K'}, m'_{K'}, t'_{K'})\}.$$

- **Product**, which is the Cartesian product of two frontiers:

$$\mathcal{F} \otimes \mathcal{F}' = \cup_{1 \le k \le K, 1 \le p \le K'} \left\{ \left([S_k, S'_p], m_k + m'_p, t_k + t'_p \right) \right\}.$$

- **Union**, which is the union of two frontiers:

$$\mathcal{F} \cup \mathcal{F}' = \cup_{1 \le k \le K} \{(S_k, m_k, t_k)\} \cup \cup_{1 \le p \le K'} \{(S'_p, m'_p, t'_p)\}.$$

- **Reduce**, which is Algorithm 1, i.e., $\mathcal{F} = \text{reduce}(\mathcal{C})$. As the result of *product* and *union* may no longer be a frontier, we assume that *reduce* is always applied after the two operations.

1. Product 操作通过列举 S 和 S0 的所有可能组合来构造复合并行化策略，并在 product 中总结 S 和 S0 的成本。
2. Union 操作将 F 和 F0 中的元组放在一个集合中。
3. Reduce 会在 Product 和 Union 操作后执行，确保生成成本边界

Algorithm 1 Reduce to frontier

1: **Input:** A set $\mathcal{C}$ containing K strategy tuples
2: **Output:** The cost frontier $\mathcal{F}$ of $\mathcal{C}$
3: Sort tuples in $\mathcal{C}$ in ascending order of memory consumption and denote the result as $\mathcal{C}_m$
4: Initialize $\mathcal{F} = \emptyset$, $v = +\infty$
5: **for** $i = 1$ **to** K **do**
6: **if** $t(\mathcal{C}_m[i]) < v$ **then**
7: $\mathcal{F} = \mathcal{F} \cup \mathcal{C}_m[i]$ and $v = t(\mathcal{C}_m[i])$
8: **end if**
9: **end for**
10: Return $\mathcal{F}$

FT 算法的处理过程

1. Initialization（第三行）：为计算图 G 中的每个 operator(工作节点)和边设置成本边界。
2. Elimination（对应 4-11 行）：将图简化为线性图 G0，并更新 operator 和边的成本边界。
3. LDP(线性动态规划，对应 12 行)：找到简化图 G0 的成本边界
4. Unroll（13-14 行）：重建原始计算图 G 的成本边界中的并行化策略和重建图 G。

Algorithm 2 Frontier Tracking (FT)

1: **Input:** Computation graph $\mathcal{G}$, device graph $\mathcal{D}$
2: **Output:** All parallelization strategies in the cost frontier of execution time and memory consumption
3: Initialize all valid parallelization configurations and their costs for the operators and edges in $\mathcal{G}$
4: **while** true **do**
5: Mark nodes on the linear graph
6: **if** not TryExactEliminate($\mathcal{G}$) **then**
7: **if** not TryHeuristicEliminate($\mathcal{G}$) **then**
8: break
9: **end if**
10: **end if**
11: **end while**
12: Apply LDP in Algorithm 3 on the simplified graph $\mathcal{G}'$ and generate strategies on the cost frontier
13: Unroll the LDP
14: Unroll the elimination

Elimination

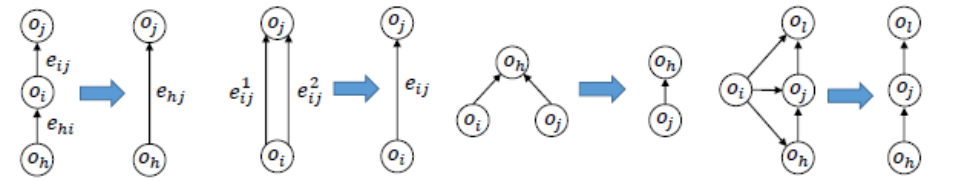


Figure 3: An illustration of different types of elimination in FT

Oi 表示第 i 个 operator，eij 表示 Oi 的输出张量用作 Oj 的输入

LDP

Algorithm 3 Linear Dynamic Programming (LDP)

Input: A linear computation graph $\mathcal{G}'$ and its size n
Output: All parallelization strategies in the cost frontier of execution time and memory consumption
$\mathcal{CF}(o_1, s_1^k) = \mathcal{F}(o_1, s_1^k)$ for $s_1^k \in \mathcal{S}_1$
for $i = 2$ **to** n **do**
  for $s_i^p \in \mathcal{S}_i$ **do**
    $\mathcal{CF}(o_i, s_i^p) = \cup_{s_{i-1}^k \in \mathcal{S}_{i-1}} \{\mathcal{F}(e_{(i-1)i}, s_{i-1}^k, s_i^p) \otimes \mathcal{CF}(o_{i-1}, s_{i-1}^k) \otimes \mathcal{F}(o_i, s_i^p)\}$
  end for
end for
$\mathcal{F}_o = reduce(\cup_{s_n^k \in \mathcal{S}_n} \mathcal{CF}(n, s_n^k))$
Return $\mathcal{F}_o$

实验评估

统计的模型

Table 1: Statistics of the models

Model	Parameter (GB)	Batch Size	Memory (GB)
RNN [21]	108	256	126
WideResNet [30]	7.3	256	83
Transformer [26]	9.7	256	74
VGG16 [24]	0.52	256	30

成本边界评估

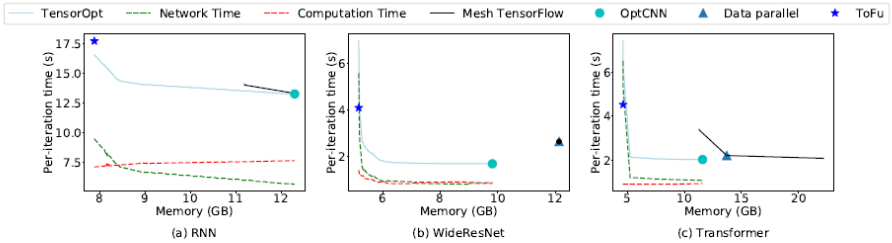


Figure 6: The cost frontier between memory consumption and execution time for some popular models, the solid lines are the cost frontiers while the dotted lines are the network time and computation time of TensorOpt (best viewed in color)

最后，对于所有三种模型，数据并行的性能都很差，内存消耗大，每次迭代时间长。OptCNN 总是在 TensorOpt 的成本边界上找到包层时间最短的点，因为它的设计目的是最小化每次迭代的时间。相比之下，Tofu 总是使用少量的内存，每次迭代的时间很长。与 OptCNN 和 ToFu 相比，TensorOpt 可以在前沿的任何点工作，这为适应资源可用性和成本效益权衡带来了更好的灵活性。

Estimation 评估

Table 2: Estimation error of the FT algorithm

Model	Execution Time	Network Time	Memory
RNN	7.16%	7.16%	4.86%
WideResNet	7.62%	3.05%	4.47%
Transformer	5.02%	7.23%	0.98%

比 OptCNN 和 FlexFlow 要好

运行时间评估

Table 3: Running time of the FT algorithm (in seconds)

Model	WideResNet	RNN	Transformer
FT-LDP	1,292	0.28	201
FT-Elimination	19,666	1.78	3,030
FT-LDP (no multi-thread)	17,432	0.40	1,535

Table 4: Per-iteration time for TensorOpt and Horovod (s)

Model	VGG16	WideResNet	Transformer-S
TensorOpt (mini-time)	0.10	1.99	1.16
TensorOpt (data parallel)	0.16	2.89	1.18
Horovod	0.15	2.80	1.04

总结

缺点：缺乏示例描述如何动态规划选出最佳成本边界。优点：创新点充分