Headline

大家好：

这是2018年度第9篇Arxiv Weekly。

本文是 **模型研究** 方向的文章。

Highlight

> **本文利用流水线搭建超大规模网络提升点数，同时引出对up-to-date的数据增强策略和AmoebaNet网络介绍**

Information

Title

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Link

https://arxiv.org/pdf/1811.06965.pdf

Source

- 谷歌（Google）[Quoc V. Le 组]

Introduction

本文针对训练规模巨大的深度神经网络，提出了规模可调的流水线并行训练架构GPipe。

Gpipe通过模型在多卡上的分割和流水线式的前传反传过程，实现了硬件平台上模型并行与数据并行的高效融合。并利用梯度的重复计算来进一步节省储存空间。

实测在8卡平台利用Gpipe架构能够训练25倍大的网络，且能保证梯度计算的连续性和结果与卡数的正交性。最关键的数，Gpipe架构能实现几乎线性的训练加速过程，并且不需要调整任何模型参数。实测利用4倍的卡数训练时能实现3.5倍的提速。

本文中，训练了一个庞大的网络：参数量557 million的最新网络AmoebaNet [配合最新的数据增强方法AutoAugment]。这个网络实现了state-of-the-art的性能，在imageNet上top1 84.3% / top5 97.0%。另外在其他几个分类数据及上，网络经过finetune也表现出惊人的性能，其中在Cifar10上达到了99%的准确率，在Cifar100上达到了91.3%的准确率。

Keys

1. Gpipe特征

Gpipe需要解决的关键问题是以下两个事实的矛盾：

- 随着DNN的发展，模型规模愈加庞大的同时数据质量日益提高。数百层的模型带来百万级的参数；高质量的采集和标注带来4k级的图片数据。
- 硬件平台的发展缓慢，以GPU为例，2014年代表性的Nvidia K40的显存为12GB；而2018年代表性的Nvidia V100的显存为32GB。

之前很多工作已经做了这方面的探索。故而目前有两个有利因素可以利用：

- 数据并行技术高度成熟
- 多卡通信技术高度成熟

> 为了基于目前的环境最大程度地解决大模型训练的问题，本文提出了具有以下特征的Gpipe架构：
>
> - **数据并行和模型并行融合**。利用 *流水线技术* 解决传统模型并行互相等待的尴尬局面；采用“ *micro batch* ”技术进一步释放显存。
> - **取消forward结果的保存**，进一步释放显存给模型和数据使用。[尤其针对深模型和大模型，这点很重要，因为这些模型的forward中间结果也很占地方]
> - 属于高效 **同步训练架构**，且实现模型真正独立的“scalable”属性。也即扩大模型规模本身不影响模型超参数和实

[p.s. 目前Gpipe即将在tensorflow开源，但是在其他主流的训练平台上核心算法也可以复现]

2. Gpipe算法

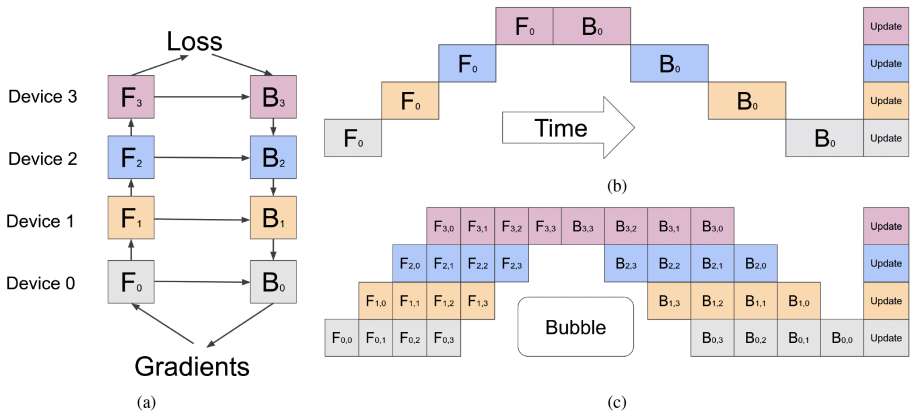


Figure 2: (a) An example neural network with sequential layers is partitioned across four accelerators. F_k is the composite forward computation function of k-th partition. B_k is the corresponding backpropagation function. B_k depends on both B_{k+1} from upper layer and the intermediate activations of F_k. (b) The naive model parallelism strategy leads to severe under-utilization due to the sequential nature of the network. Only one accelerator is active at a time. (c) Pipeline parallelism divides the input mini-batch into smaller micro-batches. It enables different accelerators to work on different micro-batches at the same time. $F_{k,t}$ and $B_{k,t}$ refer to forward and backward computation of the t-th micro-batch on k-th partition. Gradients are applied synchronously at the end of each mini-batch.

可以看到，上图(a)中展示了传统的模型并行方法，而(b)给出了这样并行的代价：将模型并行为K个部分，则运行效率变为原本的1/K倍，也即O((K-1)/K)的bubble time损耗。这也是为什么目前工程上很少使用模型并行。

然而而当模型scale up的时候，存储的局限性使得模型并行变成刚需，此时必须探索一种高效并行的方式。CPU设计中的流水线思想是一个自然的candidate。

> 算法示意图如上图(c)中所示：
>
> 1. 对于特别巨大的模型，我们按照layer等元素将其分成K等分。这里和传统模型并行别无二致。
> 2. 将已有的单卡batch size再分成T等份，使得数据的size从[N, C, W, H]变为[T, N/T, C, W, H]。文章称[N/T, C, W, H]规模的数据为一个“micro batch”。每张卡同时只处理一个micro batch，而非原来的一个batch。
> 3. 在最后一个layer的最后一张卡完成计算后，它负责这个batch中的所有数据，并计算loss，完成前传。
> 4. 反传是一个对偶的过程。
> 5. 对于BN层这样涉及统计特性的层，引入类似SyncBN的技术，每次前传全部算好后进行同步，保证moving值的准确性。

我们可以看到：这样设计的好处是，layer1只需要在layer0完成第一个micro batch的计算后，就能开始计算了。不像原本需要等layer0彻底算完一整个batch后才能开始计算。

利用Gpipe的架构，将模型并行为K个部分，同时将batch分为T个部分，运行效率会变为原本的T/(T+K-1)倍， **bubble time损耗变为了O((K-1)/(T+K-1))。显然有了长足的优化。尤其当T很大的时候，模型并行带来的损失会变得很小。**

Gpipe还有一个重要的实现细节，就是取消前传时中间变量的保存，改为只保存分割后每部分模型最后的输出结果。在反传时，每部分模型通过上一部分模型的最终输出和本部分模型的参数，复现前传过程用来支撑反传。

这样需要储存的中间变量会成为原本的K/L倍，且分散在K张卡上，等于变成了1/L倍。这样一来，对于深度神经网络而言，显存就得到了巨大的释放。

Results

1. 大模型训练点数展示

Model	Image Size	# Parameters	Top-1 Accuracy (%)	top-5 Accuracy (%)
Incep-ResNet V2[47]	299 × 299	55.8M	80.4	95.3
ResNeXt-101 [52]	299 × 299	83.6M	80.9	95.6
PolyNet[54]	331 × 331	92.0M	81.3	95.8
Dual-Path-Net-131[10]	320 × 320	79.5M	81.5	95.8
SENet [24]	320 × 320	146M	82.7	96.2
AmoebaNet-C (6, 228)[13]	331 × 331	155.3M	83.5	96.5
AmoebaNet-B (6, 512)	480 × 480	557M	**84.3**	**97.0**

Table 2: Top-1 / top-5 classification accuracy for AmoebaNet-B (6, 512) compared to other published state-of-the-art models on ImageNet ILSVRC 2012 validation dataset. Data in the this table suggested that better model quality might be obtained by higher model capacity (# of parameters) and more computation (larger input image size).

Dataset	# Training Examples	# Test Examples	# Classes	Our Model Accuracy (%)	Best Published Result (%)
CIFAR-10	50,000	10,000	10	**99.0**	98.5 [13]
CIFAR-100	50,000	10,000	100	**91.3**	89.3 [13]
Oxford-IIIT Pets	3,680	3,369	37	**95.9**	94.3* [28]
Stanford Cars	8,144	8,041	196	**94.6**	94.1* [53]
Food-101	75,750	25,250	101	**93.0**	90.4* [14]
FGVC Aircraft	6,667	3,333	100	92.7	**94.5**** [30]
Birdsnap	47,386	2,443	78	83.6	**83.9**** [30]

Table 3: Transfer learning results using AmoebaNet-B (6, 512) initialized with the best ImageNet model, using an input image size of 480 × 480 and single crop at test time. Our results were averaged across 5 fine-tuning runs. (*) Note that Kornblith *et al.* [28], Yu *et al.* [53], and Cui *et al.* [14] also fine-tuned pre-trained weights on ImageNet. (**) Krause *et al.* [30] leveraged 9.8 millions additional pre-training images collected using Google image search.

可以看到，在GPipe的加成下，成功训练出了较为强大的巨型网络，实现了ImageNet以及多个数据集上，单一网络的最佳测试性能。

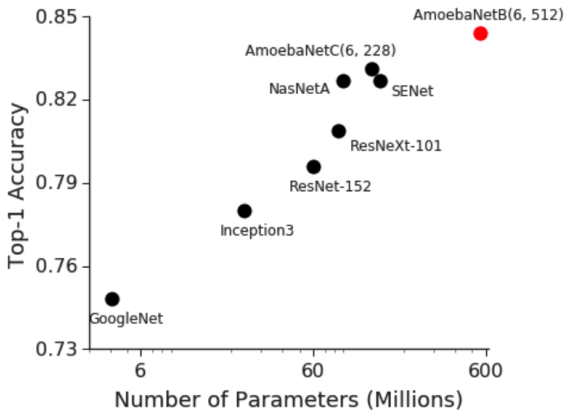


Figure 1: Strong correlation between top-1 accuracy on ImageNet 2012 validation dataset and model size for representative state-of-the-art image classification models in recent years [48, 49, 23, 52, 24, 55, 44]. Red dot shows 84.3% top-1 accuracy for a giant AmoebaNet model trained by GPipe.

本文训练的网络和benchmarks们对比示意图如上。

2. 八卡基础上扩大模型的能力

	Naive-1	Pipeline-1	Pipeline-2	Pipeline-4	Pipeline-8
AmoebaNet-D (L, F)	(6, 208)	(6, 416)	(6, 544)	(12, 544)	(24, 512)
# of Model Parameters	82M	318M	542M	1.05B	1.8B
Total Peak Model Parameter Memory	1.05GB	3.8GB	6.45GB	12.53GB	24.62GB
Total Peak Activation Memory	6.26GB	3.46GB	8.11GB	15.21GB	26.24GB

Table 1: Maximum model size of AmoebaNet supported by GPipe under different scenarios. Naive-1 refers to the sequential version without GPipe. Pipeline-k means k partitions with GPipe using k accelerators. L and F control the number of layers and the number of filters of AmoebaNet, respectively. We recorded maximum model size by increasing L and F until we reached the limits of accelerator memory in each scenario. Input image size was 224×224 and the batch size was 128. GPipe divided the mini-batch into 16 micro-batches. It supported up to 1.8 billion parameters with 8 accelerators. Total peak model parameter memory and activation memory across all accelerators are also shown.

当然，正经考察Gpipe性能还是要看，计算资源不变的条件下，并行训练网络的能力。

从上图可以看到，引入T=16的pipeline后，以AmoebaNet为基础，单卡能容纳的模型参数量增长了4倍，从82M到318M。另外，引入多卡训练pipeline后，模型参数量近似线性增长，8卡能够容纳18亿参数的模型进行训练。（而传统没有模型并行的架构，多卡只能加速训练，不能扩张模型）

3. 多卡加速能力

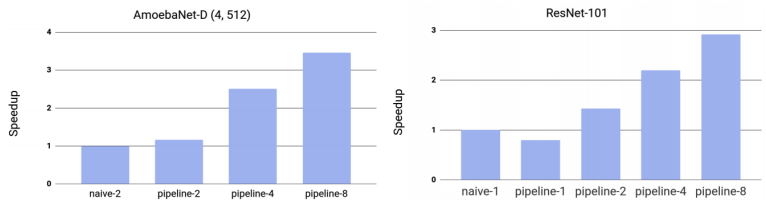
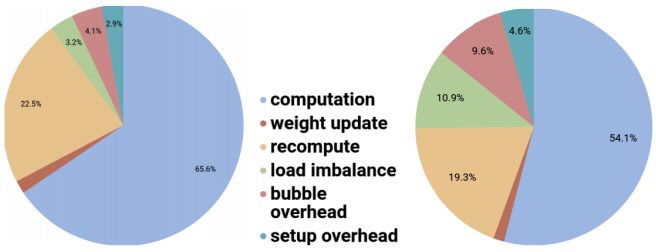


Figure 3: (a): Performance of AmoebaNet-D (4, 512) under different scenarios. This model could not fit into one accelerator. It achieved 3.5 times speedup comparing to the baseline case naive-2: naive model parallelism with 2 partitions. (b): Performance of ResNet-101 under different scenarios. Pipeline-k means k partitions with GPipe using k accelerators. The baseline naive-1 refers to the sequential version without GPipe. The image size for both models was fixed at 224×224. Note that ResNet-101 is a small model that won't be beneficial from any model parallelism. But it allowed us to analyze system performance and identify overheads easily.

可以看到在AmoebaNet上实现了近似线性的训练加速，但是在ResNet上由于模型分割导致的损耗（例如重跑forward等），8卡的加速只能到3.5倍左右。

4. 损耗分析



(a) Two Partitions (b) Four Partitions

Figure 4: Time step breakdowns from ResNet-101 runs with 2 (a) and 4 (b) partitions, respectively. We analyzed the trace files and measured the times spent on different categories. Setup overhead measured the time to divide and reshape the inputs for pipelining. Bubble overhead measured the idle time between forward and backward passes. Load imbalance measured the waiting time for the next mirco-batch due to imbalanced partition. Recompute accounted for the recomputation time during back pass. Weight update measured the time for applying gradients.

显然Gpipe会导致一定程度的损耗，主要表现在如下两个方面:

- 增加了冗余的中间计算，是取消forward暂存释放模型的trade-off。
- 模型的各个子块之间work load的不均衡导致的等待。

为了定量说明Gpipe带来的性能损耗，作者针对ResNet进行了时间-任务饼图分析。

可以看到 **最大的损耗还是在前传的重复计算上**，对于resnet这样本身网络参数量不大的结构尤其如此。而 **随着partitions数量的增加，load imbalance带来的影响也会明显增大**，成为主要损耗项之一。这一点也是很直观的。

Insights

对未来训练高精度图片数据集、训练超大规模网络，都有很强的实用意义。不过需要自己在pytorch、caffe中实现这个架构。