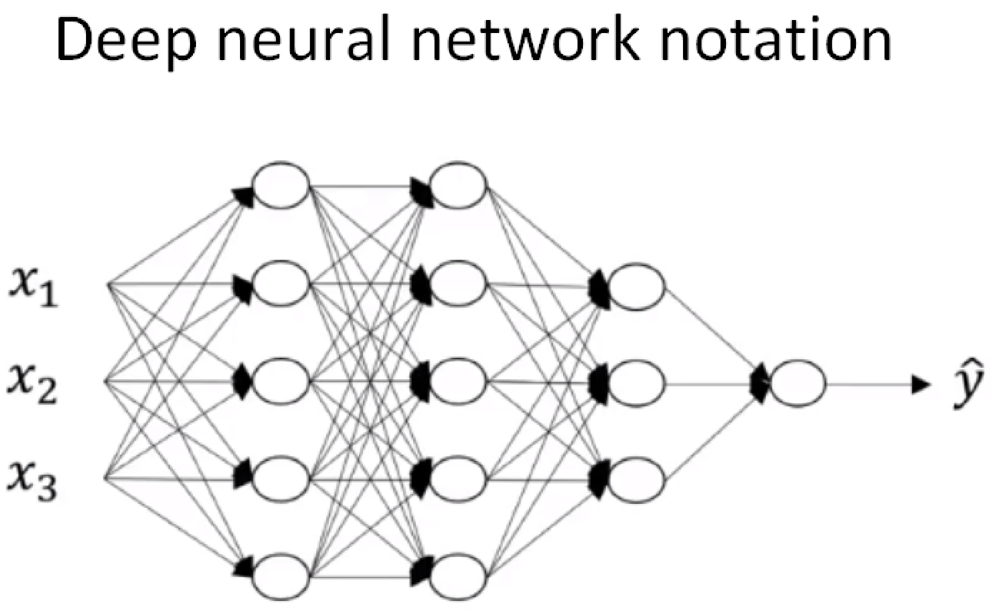
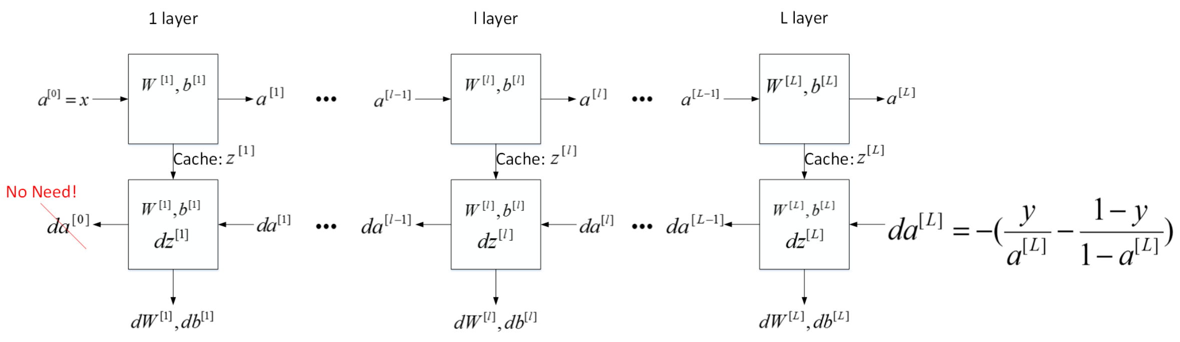
PipeMare: Asynchronous Pipeline Parallel DNN Training

deep neural network(DNN)



Forward and Backward Propagation



forward

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

backward

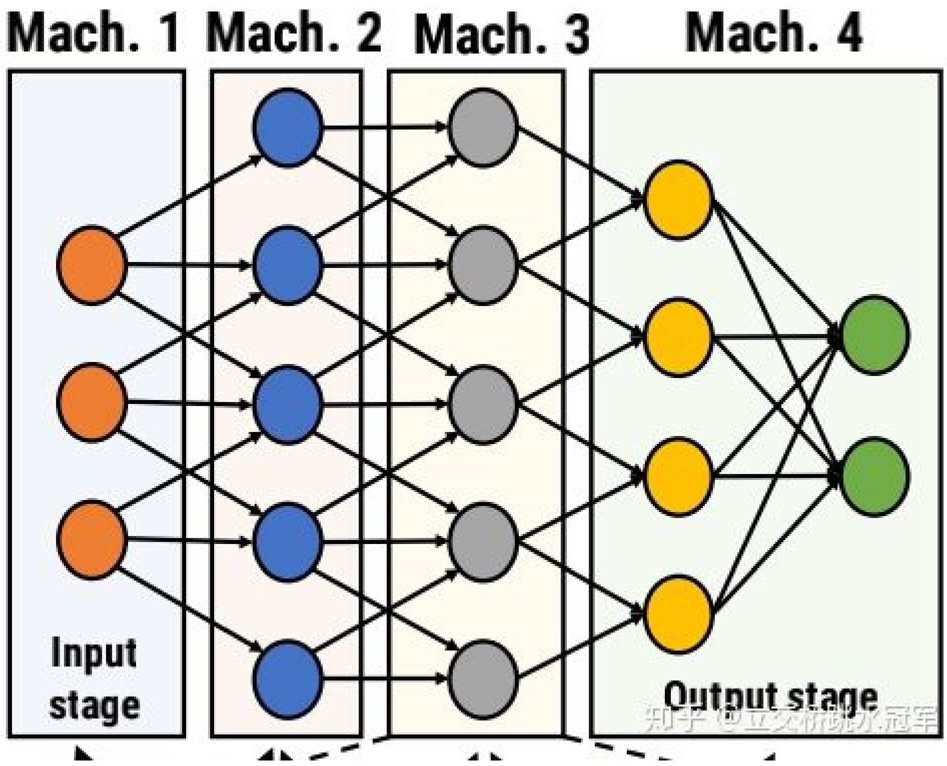
$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis = 1)$$
$$dA^{[l-1]} = W^{[l]} dZ^{[l]}$$

key point

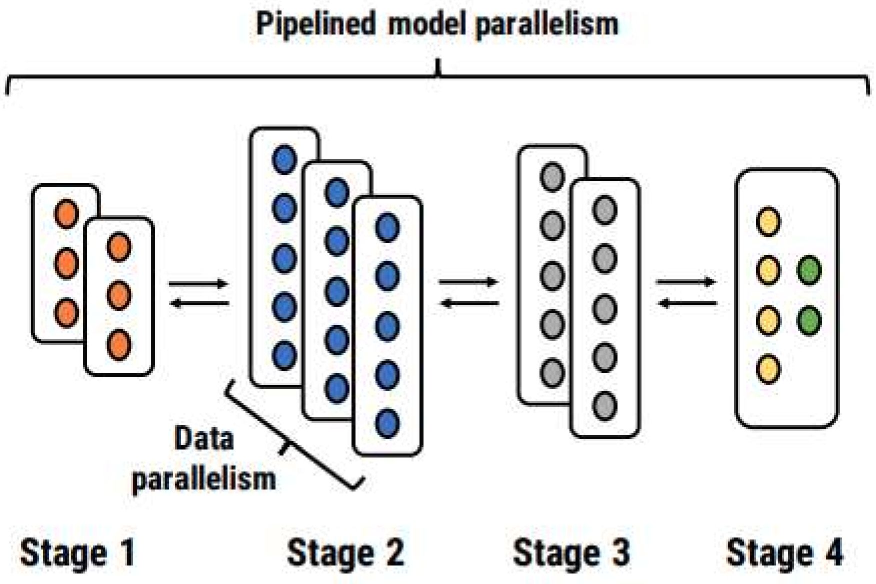
- layer之间存在数据相关,前向传播的时候下一层(l层)依赖上一层(l-1层)的激活，反向传播的时候下一层(l层)依赖上一层(l+1)层的梯度。如果直接按层划分，采用model parallelism是无法并行计算的
- 一次迭代中需要使用俩次模型参数W，前向传播和反向传播中各一次，且都基于同一个版本的模型参数进行计算。
- 反向传播计算过程中，除了需要使用上一层的梯度之外，还需要用到本层前向传播的中间结果Z,因此为了方便，前向传播结束后，往往会将Z进行cache。

pipeline parallelism

pipeline parallelism介绍



1. 神经网络的L层划分为P个stage，每个stage包含一层或多个连续的层。每个stage由一台worker计算，也可以和data parallelism结合，即每个stage由多个worker计算，如下图



2. 将一个mini batch(size=N)划分为T个micro batch，那么每个micro batch size=N/T
3. 流水化运行，如第一台机器计算第一个micro batch的前向传播完后，第一个micro batch流入第二台机器，由第二台机器进行第二个stage的前向传播，此时第一台机器不会空闲，而是立刻进行第

二个macro batch的stage-1的计算

pipeline parallelism 的优点

- 相比data parallelism，通信量降低
- 提高machine的利用率

pipeline parallelism 需要考虑的点

- 如何划分层(负载均衡)
- 如何处理神经网络层之间的数据相关？或者说怎么更新模型(速度和准确度的trade off)
- 是否容易扩展以处理更大的DNN模型(分析计算、存储、通信的瓶颈)

PipeDream vs Gpipe vs PipeMare

PipeDream: Fast and Efficient Pipeline Parallel DNN Training

主要解决：数据并行的通信瓶颈

> Its pipeline parallel computing model avoids the slowdowns faced by data-parallel training when large models and/or limited network bandwidth induce high communication-to computation ratio

Automatic layer division

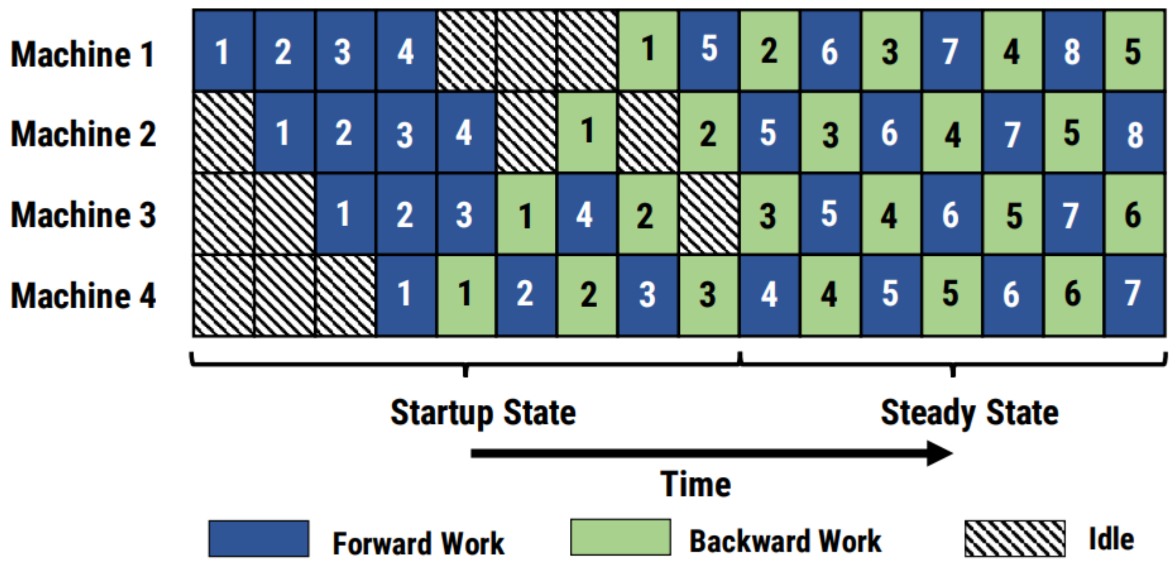
1. 先在单机跑一个profile程序，记录每一层的计算时间(包括前向传播和反向传播)
2. 根据profile程序的输出，及参数大小，等信息执行一个划分算法

最终partition algorithm输出的是如何划分层以及micro batch的数量

划分算法目标:保证层间计算负载均衡的同时最小化通信开销

Work Scheduling

每台machine



> In the startup phase, the input stage admits NOAM minibatches to keep the pipeline full in steady state. Once in steady state, each stage alternates between performing the forward and backward pass for a minibatch. We call this mechanism one-forward-one-backward (1F1B).

- 进入steady state之后，流水线完全运行，没有机器空闲

- 异步的更新模型参数，同一次迭代中反向传播和前向传播使用的模型参数不同

> 以machine 2为例，当计算第五个batch的前向传播时，使用的模型参数是更新俩次后的，但当计算反向传播时，模型参数已经被更新四次
>
> Our experimental results show that naive pipelining does not achieve the same accuracy as data-parallel training.

Weight Stashing

每台machine保存多个版本的weight,前向传播使用的是哪个版本的参数，反向传播就使用哪个版本的参数进行计算.

> While the output stage has to maintain intermediate state for only one active minibatch, the input stage needs to do so for NOAM minibatches

以上图为例，machine 1 (input stage)需要保存4个版本的参数，而machine 4 (output stage)仅需保存1个版本的参数

虽然保证了模型的准确度，但是引入了内存的overhead，scalability变差

experiment results

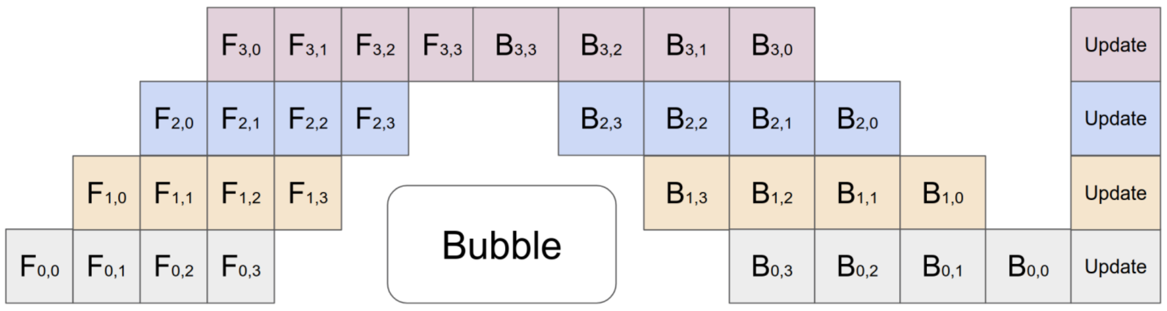
- combining pipelining, model parallelism, and data parallelism performs significantly better than using model parallelism or data parallelism alone
- PipeDream greatly reduces the overhead of communication compared to data parallel training
- PipeDream's improvements are higher for configurations that have a lower computation to-communication ratio

pros and cons

- 支持自动分层，不需要用户当做超参数来调
- 同时结合了data parallelism，model parallelism，pipeline parallelism，解决了data parallelism的通信瓶颈
- 流水线利用率高
- 引入内存开销，不适用扩展到更大的模型

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Gpipe:解决的是内存限制的问题



$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ Update
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ Update
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ Update
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ Update
Bubble

(c)

> Gradients are consistently accumulated across micro-batches, so that the number of partitions does not affect the model quality

- 反向传播时重新计算activation, 从而降低了需要需要缓存的容量(仍需缓存每一个stage最后的activation)
- 同步更新模型参数，等价于同步随机梯度下降(SGD)

pros and cons

- 内存开销小，扩展性强
- 实现了近线性加速比
- 同步更新->精度保证
- 流水线存在bubble，利用率低
- recomputation 增加了计算开销

PipeMare: Asynchronous Pipeline Parallel DNN Training

相关概念vs指标含义

Synchronous execution vs Asynchronous execution (在DNN的流水线并行模型这个语境下)

Synchronous execution 就是指前向传播计算和反向传播计算用的参数是一致的，反之，不一致就是Asynchronous execution

statistical efficiency vs hardware efficiency

statistical efficiency:模型收敛/达到目标精度所需要的epoch数

hardware efficiency:完成一个epoch所需要的时间

总的训练时间 = statistical efficiency * hardware efficiency

fine-grained PP vs coarse-grained PP

粗粒度(coarse-grained)的流水线并行: P(stages) << L(layers)

细粒度(fine-grained)的流水线并行: P(stages) ≈ L(layers)

流水线利用率(吞吐):任意给定时刻active的stage数占总stage的百分比

> existing PP techniques sacrifice hardware efficiency by decreasing pipeline utilization or incurring extra memory costs.

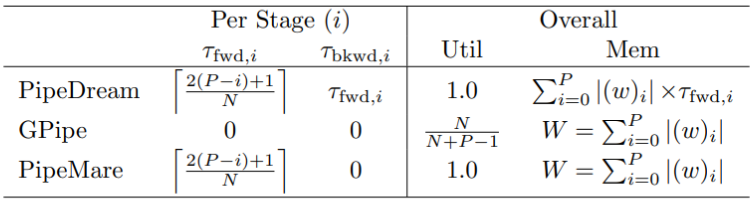
流水线delay:在读取模型参数以计算梯度和使用该梯度更新模型之间经过的模型被更新的次数

> the number of optimizer steps that pass between when the weights are read to compute a gradient and when that gradient is used to update the weights.

PipeMare希望解决的是PipeDream 引入的 memory开销和Gpipe牺牲的流水线利用率

> In PipeMare we let the computation proceed asynchronously: we just compute gradients with whatever weights are in memory at the time we need to use them. This avoids any need to store extra copies of our model weights (Mem = W) or introduce bubbles into our pipeline (Util = 1.0),

流水线利用率和delay的分析对比



	Per Stage (i)		Overall	
	$\tau_{\text{fwd},i}$	$\tau_{\text{bkwd},i}$	Util	Mem
PipeDream	$\left\lceil \frac{2(P-i)+1}{N} \right\rceil$	$\tau_{\text{fwd},i}$	1.0	$\sum_{i=0}^{P} \lvert(w)_i\rvert \times \tau_{\text{fwd},i}$
GPipe	0	0	$\frac{N}{N+P-1}$	$W = \sum_{i=0}^{P} \lvert(w)_i\rvert$
PipeMare	$\left\lceil \frac{2(P-i)+1}{N} \right\rceil$	0	1.0	$W = \sum_{i=0}^{P} \lvert(w)_i\rvert$

Table 1: Characterization of pipeline parallel training methods. τ_{fwd} and τ_{bkwd} are the pipeline delays for model weights in the forwards and backwards pass. W is one copy of the weights. P is the number of pipeline stages. N is the number of microbatches in a minibatch. i indexes the pipeline stage. $\lvert(w)_i\rvert$ denotes the number of weights in the ith layer.

fundamental question: is preserving synchronous execution necessary during neural network training?

Learning rate rescheduling

> Unfortunately, when we try running naively with a standard step size scheme, asynchronous PP SGD can significantly underperform the synchronous baseline. This happens because a large value of τ can cause SGD to diverge even when using a step size α for which the baseline synchronous algorithm converges

论文理论上证明了学习率应该和延迟程度成反比

Discrepancy correction

Discrepancy:指前向传播和反向传播使用的模型参数不一致 (τfwd = τbkwd)

> Essentially, this technique adjusts the value of the weights used in the backward pass by extrapolating what the weights were during the forward pass based on the recent average trajectory of the weights

通过根据权重的最近平均轨迹推断前向传递期间的权重来调整反向传递中使用的权重值

pros and cons

- 异步更新，在保证流水线利用率高、内存消耗低的情况下，可以达到模型准确度
- 不支持自动分层，如何划分stage和microbatch size的大小是俩个超参数，需要用户调节