Training Large Models

Techniques for Memory Saving

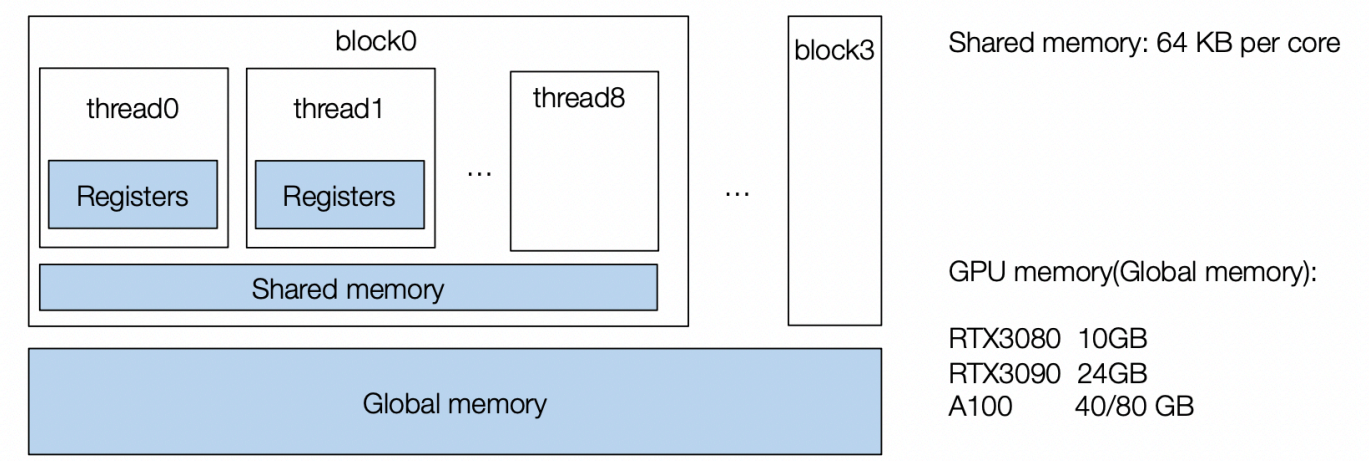
机器学习系统主要包括数据、模型、计算三部分。更大的数据集需要更大的模型，模型参数越多意味着需要的存储空间越大，对计算设备的要求越高。

本节主要讨论以下两个主题：

- 如何降低内存开销，在单个 GPU 上能运行更大的模型
- 如何扩大训练过程的规模，加速训练过程

Sources of Memory Consumption

GPU 的内存容量如下：



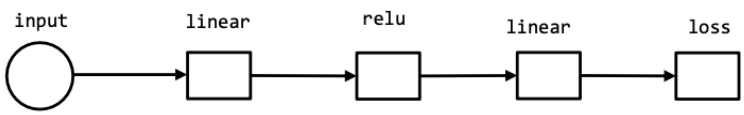
> GPU 内存通常指 global memory.

内存开销来源：

- 模型的权重参数 w_i
- Optimizer 的额外状态 u_i
- 激活函数的中间值
- 输入的数据

Techniques for Memory Saving for Inference

推理过程只关心 forward pass

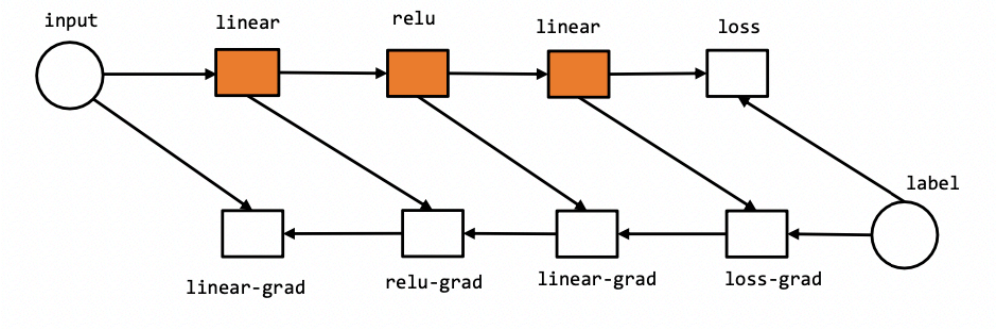


内存开销包括权重参数和激活函数的中间值

对于激活函数的中间值，只需要两个 buffer 来存储数据 (ping-pong buffer)，因为当前计算只依赖于前一层的结果。

Activation Memory Cost for Training

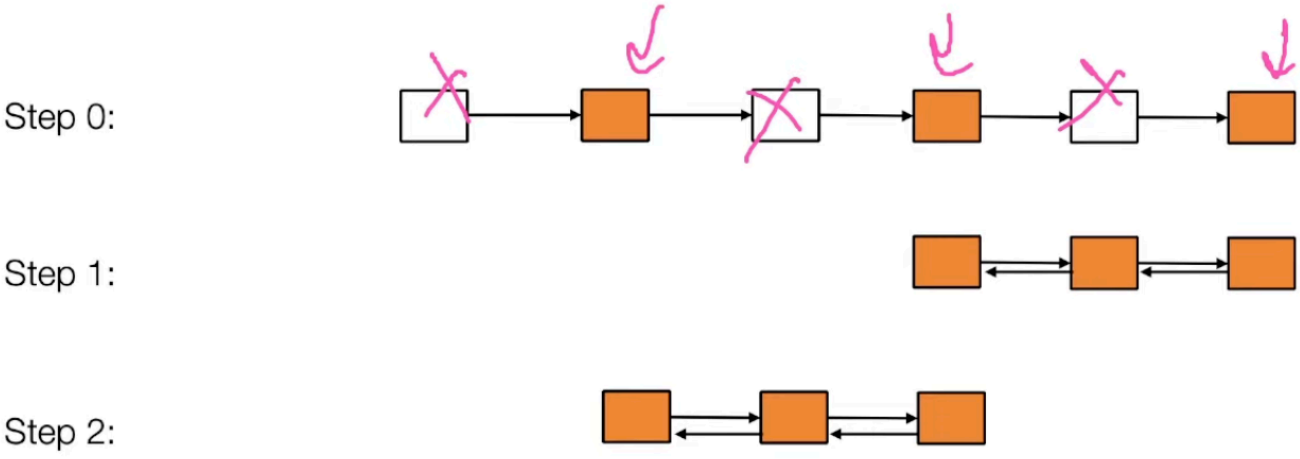
训练过程既有 forward pass 也有 backward pass, backward pass 会用到之前的中间结果，因此 ping-pong buffer 不适用。训练一个 N-layer 神经网络需要 $O(N)$ 的内存，训练比推理所需的内存多很多。



Checkpointing Techniques in AD

为了降低 activation 的内存开销，采用 checkpoint 技术。

核心思想：在 forward pass 中，只存储一部分节点的激活函数值，在 backward pass 中，通过重新计算 forward pass 的一小部分来获取缺失的激活函数值，从而计算 adjoint.



对于 N-layer 神经网络，如果在每 K 层处 checkpoint, 则内存开销为

$$Memory\ cost = O\left(\frac{N}{K}\right) + O(K)$$

Checkpoint cost　　Re-computation cost

取 $K = \sqrt{N}$, 则开销为 $O(\sqrt{N})$

多了大约 25% 的计算开销

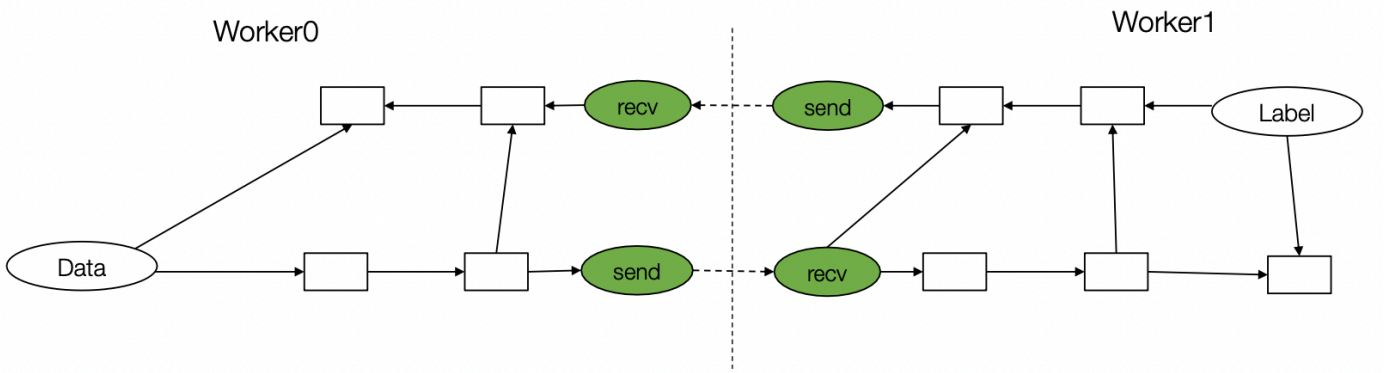
Parallel and Distributed Training

为了加速训练大模型，通常需要分布在多个节点上的多个 GPU 共同训练。

核心思想：将不同部分的计算分配给不同的节点，达到并行效果

Model Parallel Training

核心思想：将计算图划分为多个部分，分配给不同节点，并在边界处加入 send/recv 对进行通信。



对处理多个 micro-batch 的情形，这样能实现 pipeline partition, 达到并行效果。

Data Parallel Training

$$\theta := \theta - \frac{\alpha}{B}\sum_{i=1}^{B}\nabla_\theta l(h_\theta(x^{(i)}), y^{(i)}) \tag{3}$$

在使用 SGD 计算 loss function 时，可以将 minibatch 进一步划分为 B/K 份，分配给节点计算后再相加。

每个节点的计算过程相同，只是数据不同。

Allreduce Abstraction

每个工作节点有一部分数据，计算后调用 `allreduce` 合并。最终每个节点的计算结果相同，为所有节点的局部结果之和。

Interface

```
result = allreduce(float buffer[size])
```

Running Example

Worker 0

```
comm = communicator.create()
a = [1, 2, 3]
b = comm.allreduce(a, op=sum)
```

```
assert b == [2, 2, 4]
```

Worker 1

```
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
```

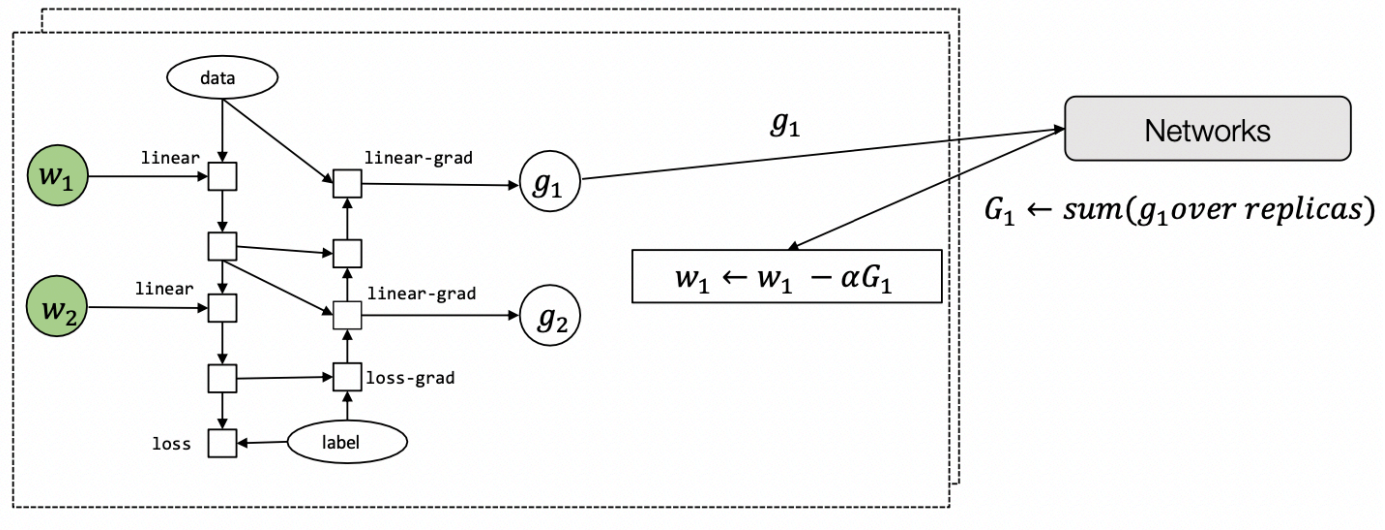
```
assert b == [2, 2, 4]
```

在 NVIDIA 中 NCCL 库实现了 `allreduce`.

Data Parallel Training via Allreduce

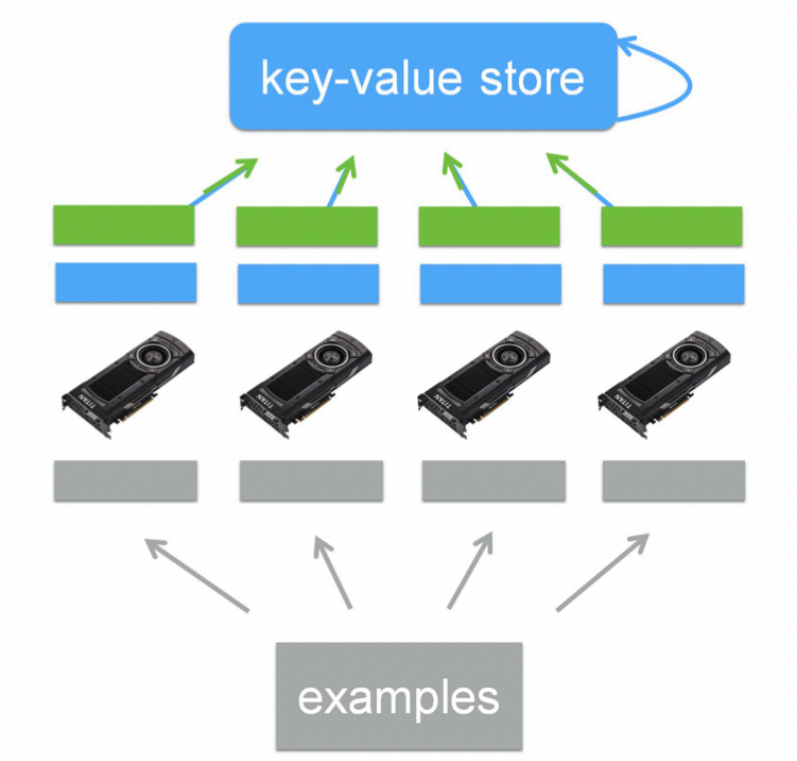
所有 worker 的计算图相同，只是数据不同。

将计算出的梯度发送到网络中同步，接收求和结果，每个节点各自对参数进行更新。



Parameter Server Abstraction

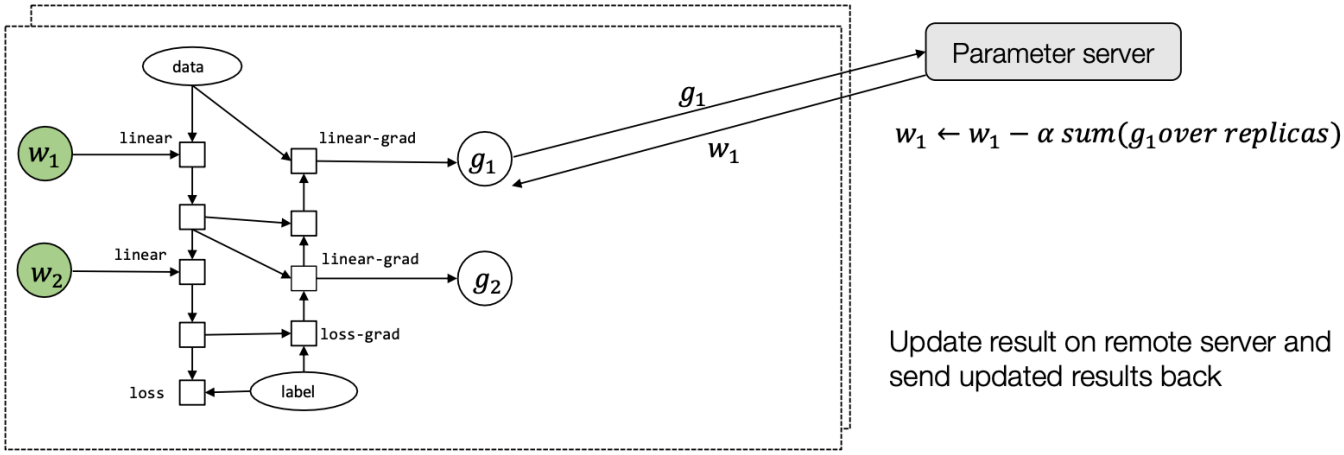
核心思想：allreduce 需要在每个节点上更新参数，parameter server 将参数存放在 sever 端，由 server 端对梯度求和并更新参数，再将参数发送给 worker.



Interface:

- `ps.push(index, gradient)`: 将参数索引和计算出的梯度发送给 parameter server.
- `ps.pull(index)`: 从 parameter server 获取更新后的参数。

Data Parallel Training via Parameter Server

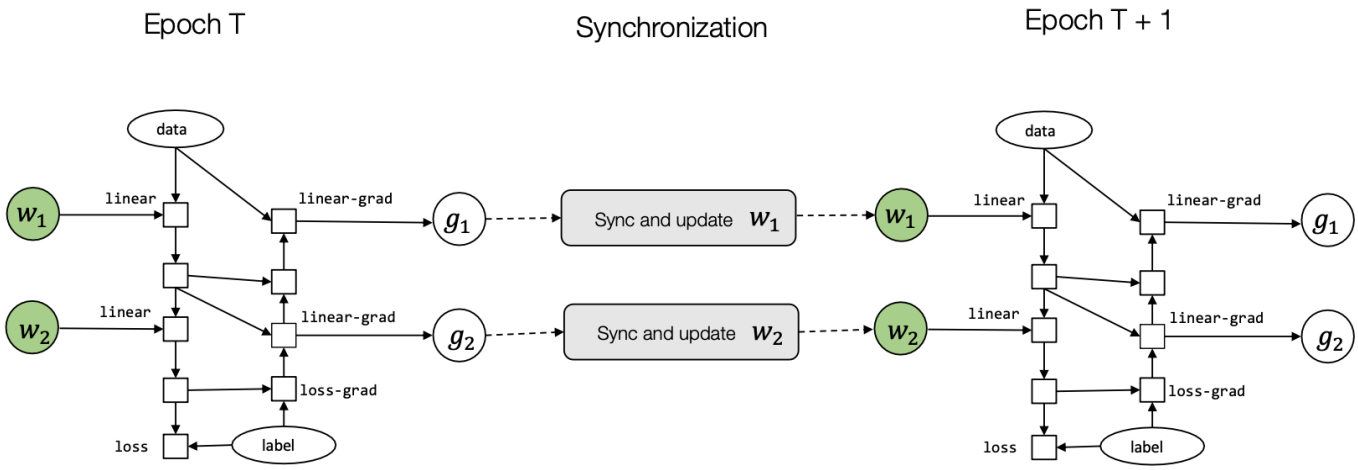


优势：

- Parameter server 不需要等到所有工作节点完成才继续执行，可以规定等到大部分节点完成就更新参数，保证训练过程的 robustness
- 工作节点崩溃时，可以在重启时从 parameter server 获取参数

Communication Computation Overlap

尽可能让计算与传输（通信）过程并行执行，类似于 OS 中磁盘访问与指令执行并行。



在节点将梯度结果发送给 server 或者通过 allreduce 同步时，可以继续计算下一个梯度，不需要等待同步完毕。

同时，更深层的梯度先被计算，但对应的参数在 forward pass 的后期才被用到。因此如果存在多个尚未完成同步的梯度，后计算的梯度的优先级应该更高。

Advanced Parallelization Methods

- ZeRO: Memory Optimziations Toward Training Trillion Parameter Models
- Beyond Data and Model Parallelism for Deep Neural Networks
- GSPMD: General and Scalable Parallelization for ML Computation Graphs