COMP4007: 并行处理和体系结构

第七章：并行编程高级主题II

授课老师：王强、施少怀
助　　教：林稳翔、刘虎成

哈尔滨工业大学（深圳）

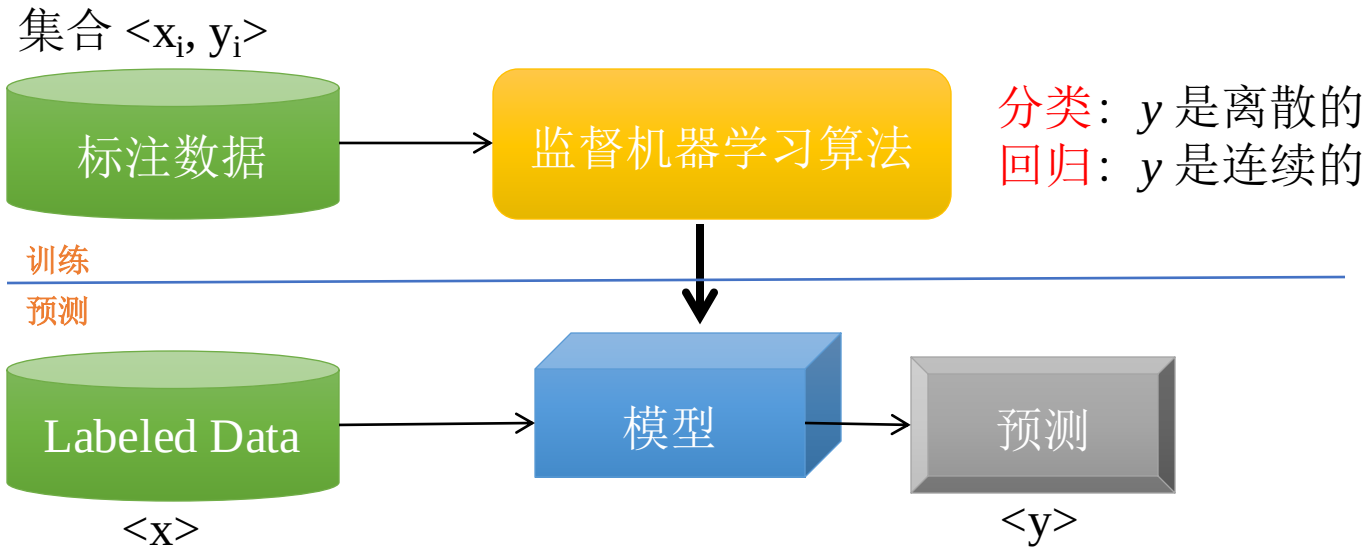
要点

- 机器学习基础
  - 有监督机器学习
  - 微积分回顾
  - 梯度下降
- （深度）神经网络
- 模型训练：SGD与反向传播
- GPU集群如何加速深度神经网络训练

机器学习基础

- 机器学习(ML) 是计算机科学的一部分，它允许计算机在**不显式编程**情况下进行学习
- 我们这里只介绍**有监督机器学习**

有监督机器学习要解决的问题：给定一个新的数据样本 ***x***，它对应的标签 *y* 是什么？



它是一种可以从数据中学习，并基于数据进行预测的算法

有监督ML如何运作？

- 机器学习模型是有个有很多未知参数的复杂函数。给定一个输入，它会给出一个输出响应
  - 注意：这个输出可能与真值接近，也可能相反
  - 一个好的模型有更高的概率会给出一个好的预测，也就是接近真值，即使是从未见过的数据（也就是该数据不在现有的数据集里）。

- 寻找一个好的ML模型的一般方法框架：
  - 1. 将数据集划分为3个子集：训练集、验证集和测试集
  - 2. 设计模型结构，也就是函数的形式
  - 3. 在训练集上训练：用以确定那些未知的参数。该步将花费很多时间，它实际是在求解一个优化问题
    - 如何设置那些未知参数，使得模型的预测输出与真值的总体误差最小？
  - 4. 在验证集上评价模型的性能。如果不够好，返回第2步，即微调模型或更换方法
  - 5. 一旦模型被最终确定，在测试集上测试它的性能

微分

- 考虑一个单变量函数 y = f(x)， x 是一个实数输入，y 是一个实数输出
- y = f(x) 的导数表征函数的输出对输入 x 的变化的敏感程度
- 函数 f(x) 在点 x 位置的导数为：

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- 莱布尼茨符号：函数 y = f(x) 的一阶导数写为：

$$\frac{dy}{dx} \text{ or } \frac{df}{dx} \text{ or } \frac{d}{dx} f$$

- **微分**就是寻找函数导数的过程

实践：逻辑斯蒂函数

- 在人工神经网络(ANNs)中，sigmoid 函数常被用来作为激活函数，下面的逻辑斯蒂函数是其中一种 sigmoid 函数：

$$f(x) = \frac{1}{1 + e^{-x}}$$

请证明：$f'(x) = f(x) \cdot (1 - f(x))$

这是一个非常棒的特性，因为：

- $f(x)$ 是在前馈步骤中计算好的
- $f'(x)$ 在反向传播时可以快速被计算出来

偏微分规则

Product Rule: $$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}}g(\boldsymbol{x}) + f(\boldsymbol{x})\frac{\partial g}{\partial \boldsymbol{x}}$$

Sum Rule: $$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x}) + g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}} + \frac{\partial g}{\partial \boldsymbol{x}}$$

Chain Rule: $$\frac{\partial}{\partial \boldsymbol{x}}(g \circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}}$$

梯度

- n 变量函数有 n 个偏导数

- 函数 $f(x_1, \ldots, x_n)$ 的梯度是所有偏导数组成的**向量**

$$\nabla f = \frac{df}{dx} = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

训练ML模型

- 一个ML模型可以被看作是一个数学函数

$$y = f(x, w),$$

其中 x 是一个给定的输入样本，w 是模型参数，y 是模型输出。通常情况下，x, y, w 都是向量

- 训练就是寻找一组模型参数 w，对于给定的数据 $D=\{(x_i, y_i)|1\leq i\leq n\}$，平均损失 $L(D, w)$ 最小。使用均方误差(MSE)作为例子

$$L(D, w) = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \tilde{y}_i\|^2 = \frac{1}{n}\sum_{i=1}^{n}\|y_i - f(x_i, w)\|^2$$

最小化非负损失是一个优化问题

如果模型可以完美预测所有样本，则 $L(D, w) = 0$;
否则， $L(D, w) > 0$。 **更小的损失意味着更好的模型**

梯度下降

- 梯度下降是一个流行的优化策略，通过迭代的方法调整 w，降低损失
  - 为了方便讨论，我们假设 w 是一个有两个实数 w_1 和 w_2 组成的向量
- 假设数据集 D 是固定的，$L(D, w)$可以被写作 $L(w_1, w_2)$
- 现在我们假设 w 上的一个小的改变：$\Delta w = (\Delta w_1, \Delta w_2)$
- 损失loss上的改变为：$\Delta L \approx \frac{\partial L}{\partial w_1} \Delta w_1 + \frac{\partial L}{\partial w_2} \Delta w_2$
- 我们试图使ΔL 为负数，也就是通过调整w_1 和 w_2 减小损失
- 假如我们设定 $\boldsymbol{\Delta w = -r \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}\right)}$，那么

$$\Delta L \approx -r \left(\left(\frac{\partial L}{\partial w_1}\right)^2 + \left(\frac{\partial L}{\partial w_2}\right)^2\right) \leq 0 \text{ if } r > 0$$

- 梯度 $\left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}\right)$ 表示为 ∇L，那么 $\Delta L \approx -r \|\nabla L\|^2$

梯度下降

- 我们可将前面的讨论推广到k-维 w
- 更新规则简单：

$$w \leftarrow w - r\,\nabla L,$$

其中

- r 是学习率（正数），∇L 是梯度向量
- 分解向量：

$$w = (\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_k),\ \nabla L = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_k}\right),$$

更新规则为：

$$w_i \leftarrow w_i - r\frac{\partial L}{\partial w_i},\ 1 \le i \le k.$$

- 剩余挑战为如何为每一个 $\mathrm{w_i}$ 计算 $\dfrac{\partial L}{\partial w_i}$

梯度下降的伪代码

- 问题：寻找参数使得模型在数据集 $D = \{(x_i, y_i) | 1 \le i \le n\}$ 上的损失 $L(D, w)$ 最小化

/* **Gradient decent algorithm** */
Initialize w
while not converge {
    calculate $L(D, w)$
    calculate $\nabla L = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_k})$
    $w \leftarrow w - r \nabla L$
}

Drawback of gradient decent:

Given a large dataset *D*,
it may take a very long time to
calculate L(D, w):

for(i = 1; i ≤ n; i++) {
    calculate $f(x_i, w)$
    …
}

我们在遍历整个数据集后对参数进行一次更新

小批量随机梯度下降(SGD)

- 一个更加实用的方案：将数据集D随机划分为若干个小批次(mini-batch)，每个包含有 B 个样本
  - 现在共有 $S = D/B$ 个 mini-batches，表示为 $D_1, D_2, \ldots, D_S$，基于统计我们希望 $L(Di, w)$ 可以逼近 $L(D, w)$

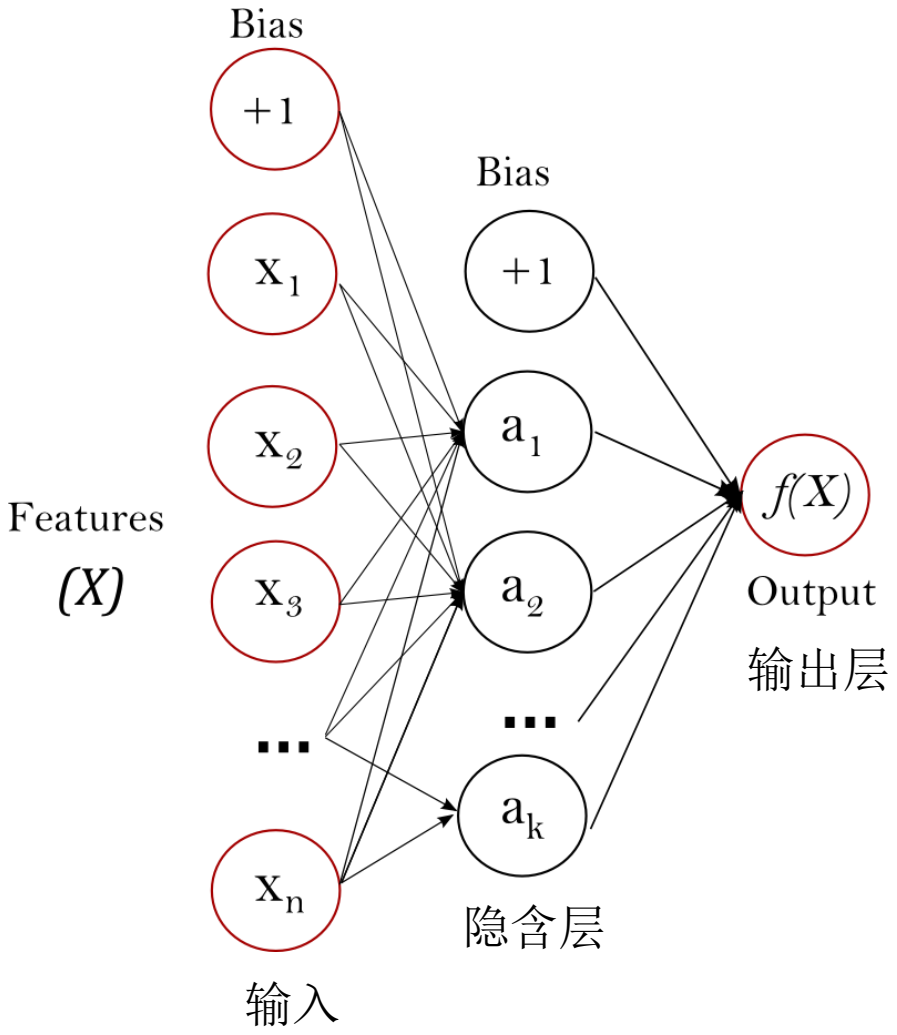
```
/* Mini-batch SGD algorithm */
Initialize w
while not converge {
    Randomly shuffle D and divide it into mini-batches D_1, …, D_S
    for(i = 1; i ≤ S; i++)
        calculate L(D_i, w)
        calculate ∇L = (∂L/∂w_1, ∂L/∂w_2, …, ∂L/∂w_k)
        w ← w – r ∇L
    } // 这称之为一个 epoch
}
```

Details later

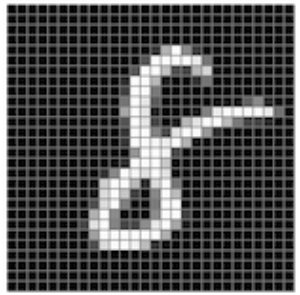
我们在一个epoch中调整参数 S 次

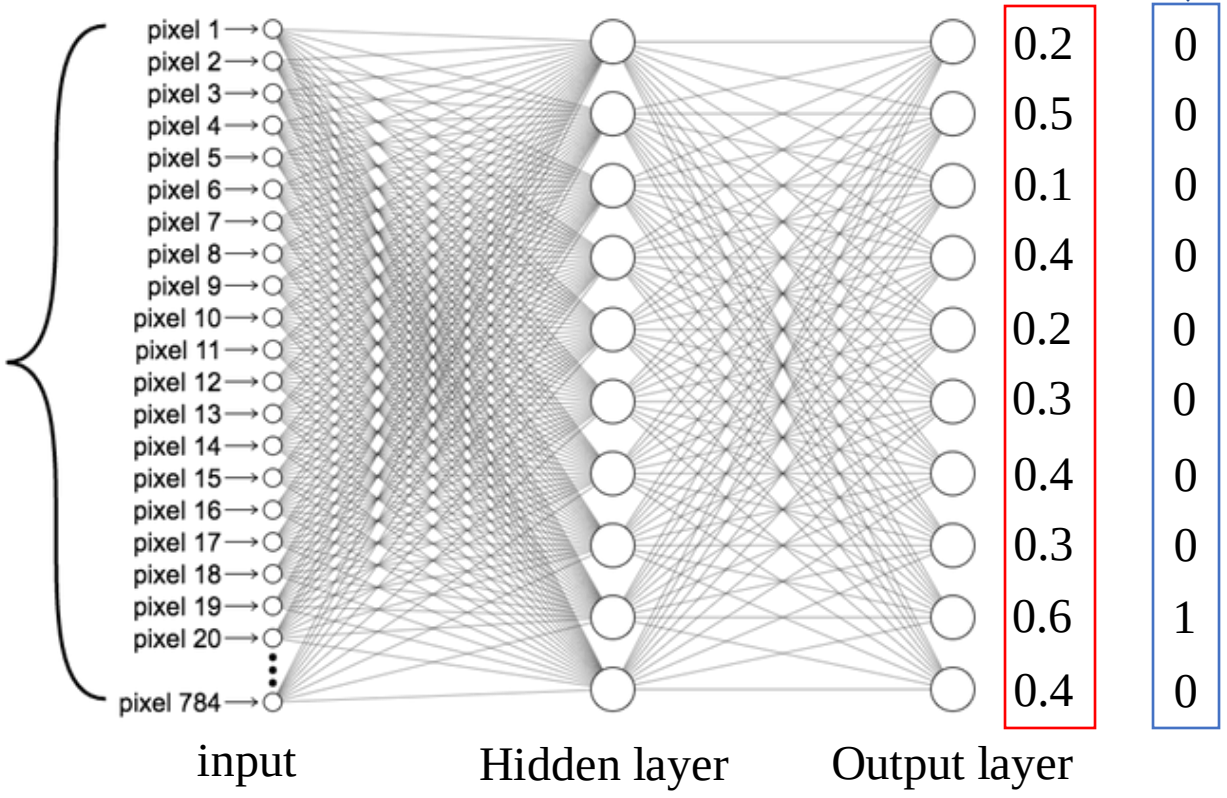
基于多层感知机的神经网络

- 感知机是一个单层神经网络，他在逼近函数时的性能是有限的（线性可分）
- 但多层感知机(MLP)可以逼近非常复杂的函数
- 在输入和输出之间至少有一层隐含层
- 每一层当中的每一个神经元都与前一层中所有神经元相连(FC)。每一条连接上都有一个权值



如何训练一个ANN

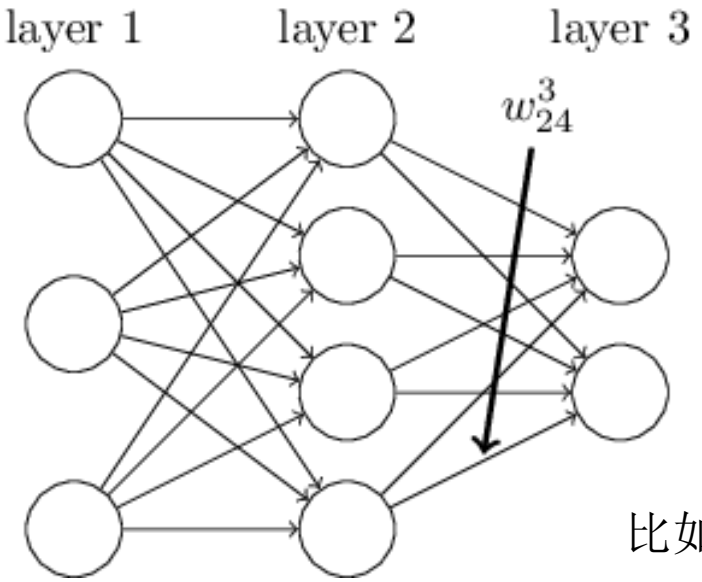




$$f(x) = \frac{1}{1+e^{-x}} \qquad f(x) = \frac{1}{1+e^{-x}}$$

Mini-batch SGD + 反向传播

- Mini-batch SGD 提供了一个训练ANN的架构
- 但我们需要找到一个方法来计算梯度 $\nabla L = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_k})$，这对于ANN来说不是一个简单的事
- 误差反向传播是一种计算ANN梯度的方法



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

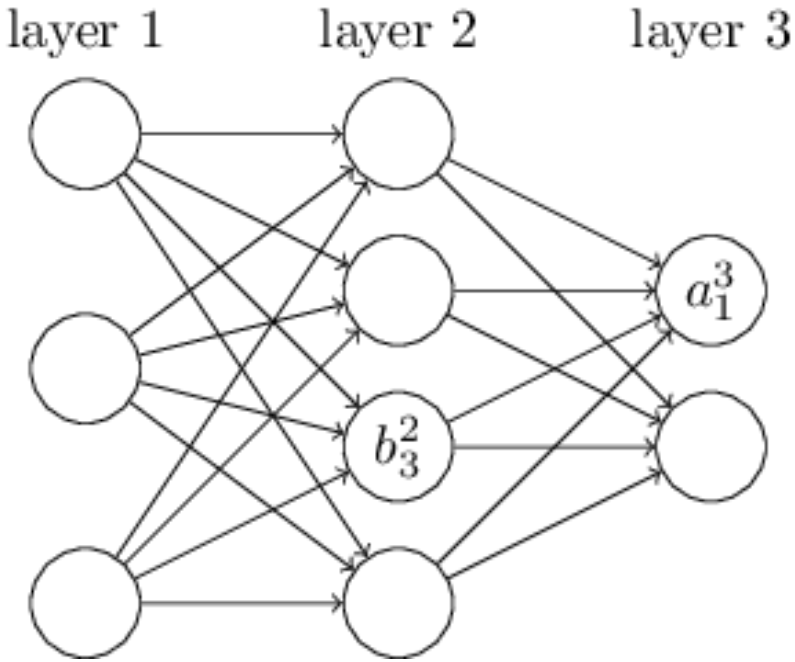
比如，layer 3 的权重是：

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix}$$

Ack: Some diagrams are from http://neuralnetworksanddeeplearning.com/chap2.html

前馈

- 以 b^l_j 表示第 l 层第 j 个神经元的偏置
- 以 a^l_j 表示第 l 层第 j 个神经元的激活值



- Elementwise:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

- Vectorized form:

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

其中 a^l 是激活向量s，w^l 是权重矩阵，
b^l 是偏置向量

计算最后一层的误差

- 误差由损失函数定义。通常平均损失与单个输入样本 x 的单一损失相关，这里我们先聚焦一个单一损失，考虑下面的损失函数：

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

- 通过计算，我们有 $\partial C/\partial a_j^L = (a_j^L - y_j)$

- 用 $\nabla_a C$ 表征偏微分向量 $\partial C/\partial a_j^L$

- 最后一层(layer L)的误差这样计算：

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

误差反向传播

- 第 l 层的误差由第 l+1 层的误差和第 l+1 层的权重矩阵计算而来

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- 而后利用误差计算梯度：

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

反向传播算法

——SGD算法细节在13页

1. **Input a set of training examples**

2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:

   - **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.

   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.

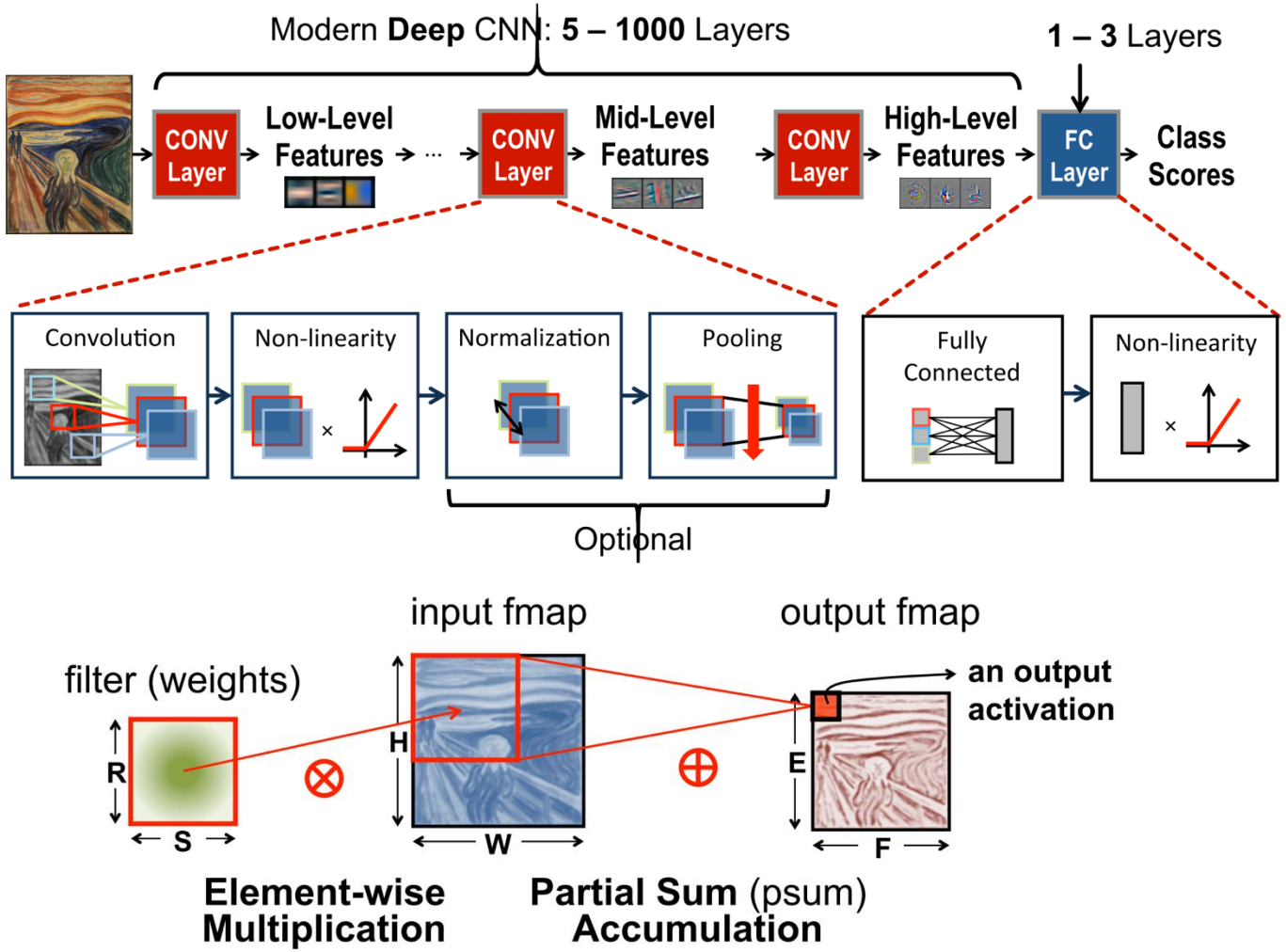
   - **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

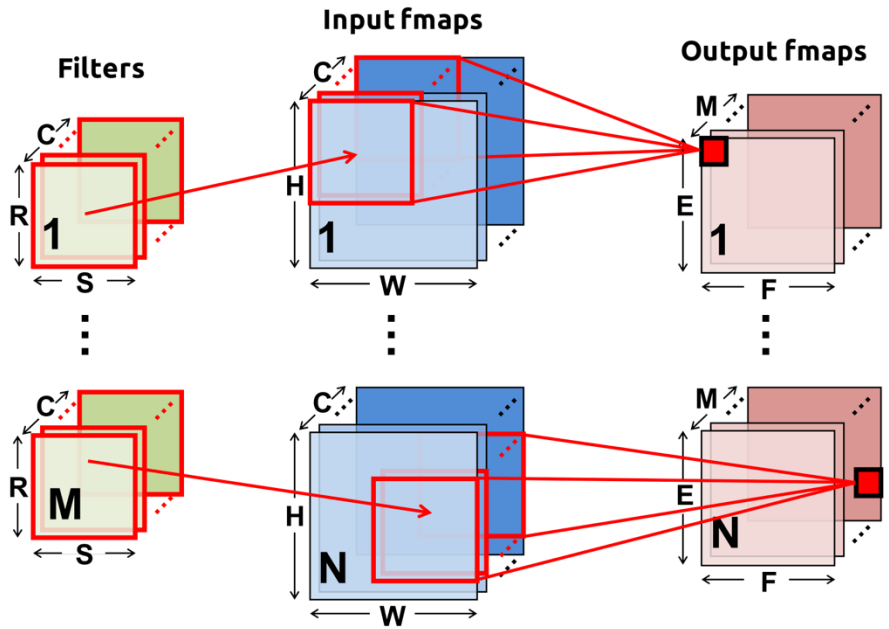
3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

注意该外积得到的是一个矩阵

DNN训练是计算密集型任务

- 计算机视觉(CV)中的卷积神经网络 (CNNs)





Source: Sze, Vivienne, et al. "Efficient processing of deep neural networks: A tutorial and survey." Proceedings of the IEEE 105.12 (2017): 2295-2329.

CNN 工作负载（一次迭代）

Layer Type	Eval.	Work (W)
Activation	y	$O(NCHW)$
	∇w	$O(NCHW)$
	∇x	$O(NCHW)$
Fully Connected	y	$O(C_{out} \cdot C_{in} \cdot N)$
	∇w	$O(C_{in} \cdot N \cdot C_{out})$
	∇x	$O(C_{in} \cdot C_{out} \cdot N)$
Convolution (Direct)	y	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$
	∇w	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$
	∇x	$O(N \cdot C_{out} \cdot C_{in} \cdot H \cdot W \cdot K_x \cdot K_y)$
Pooling	y	$O(NCHW)$
	∇w	–
	∇x	$O(NCHW)$
Batch Normalization	y	$O(NCHW)$
	∇w	$O(NCHW)$
	∇x	$O(NCHW)$

每个不同类型层的工作负载

N is the mini-batch size

Metrics	LeNet 5	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Top-5 error†	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	n/a	19.8	17.0	8.8	10.7	7.0
Input Size	28×28	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	2	5	5	13	57	53
Depth in # of CONV Layers	2	5	5	13	21	49
Filter Sizes	5	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	1, 20	3-256	3-1024	3-512	3-832	3-2048
# of Filters	20, 50	96-384	96-1024	64-512	16-384	64-2048
Stride	1	1,4	1,4	1	1,2	1,2
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	2	3	3	3	1	1
Filter Sizes	1,4	1,6	1,6,12	1,7	1	1
# of Channels	50, 500	256-4096	1024-4096	512-4096	1024	2048
# of Filters	10, 500	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58k	58.6M	130M	124M	1M	2M
MACs	58k	58.6M	124M	130M	1M	2M
Total Weights	60k	61M	146M	138M	7M	25.5M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G

流行 CNN 汇总（一个样本前馈中的的乘加操作总数）

*反向传播差不多是前向传播计算量的两倍

Source:

Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4 (2019): 1-43.

Sze, Vivienne, et al. "Efficient processing of deep neural networks: A tutorial and survey." Proceedings of the IEEE 105.12 (2017): 2295-2329.

CNN 工作负载（训练过程）

- SGD 是一种迭代算法

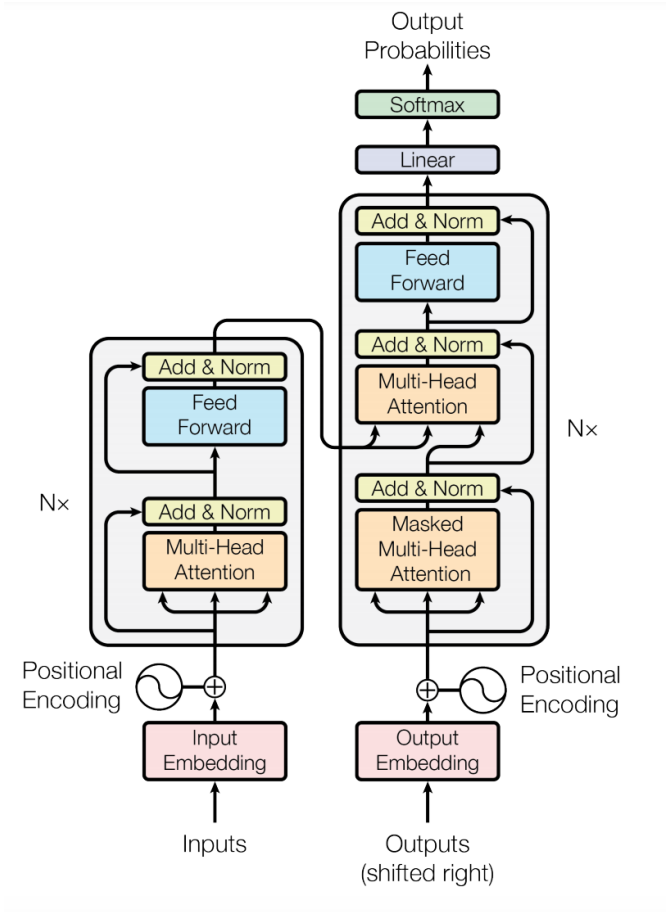
$$W_{i+1} = W_i - \eta \cdot \frac{1}{N} \sum_{j=1}^{N} \nabla L(W_i, D_i^j)$$

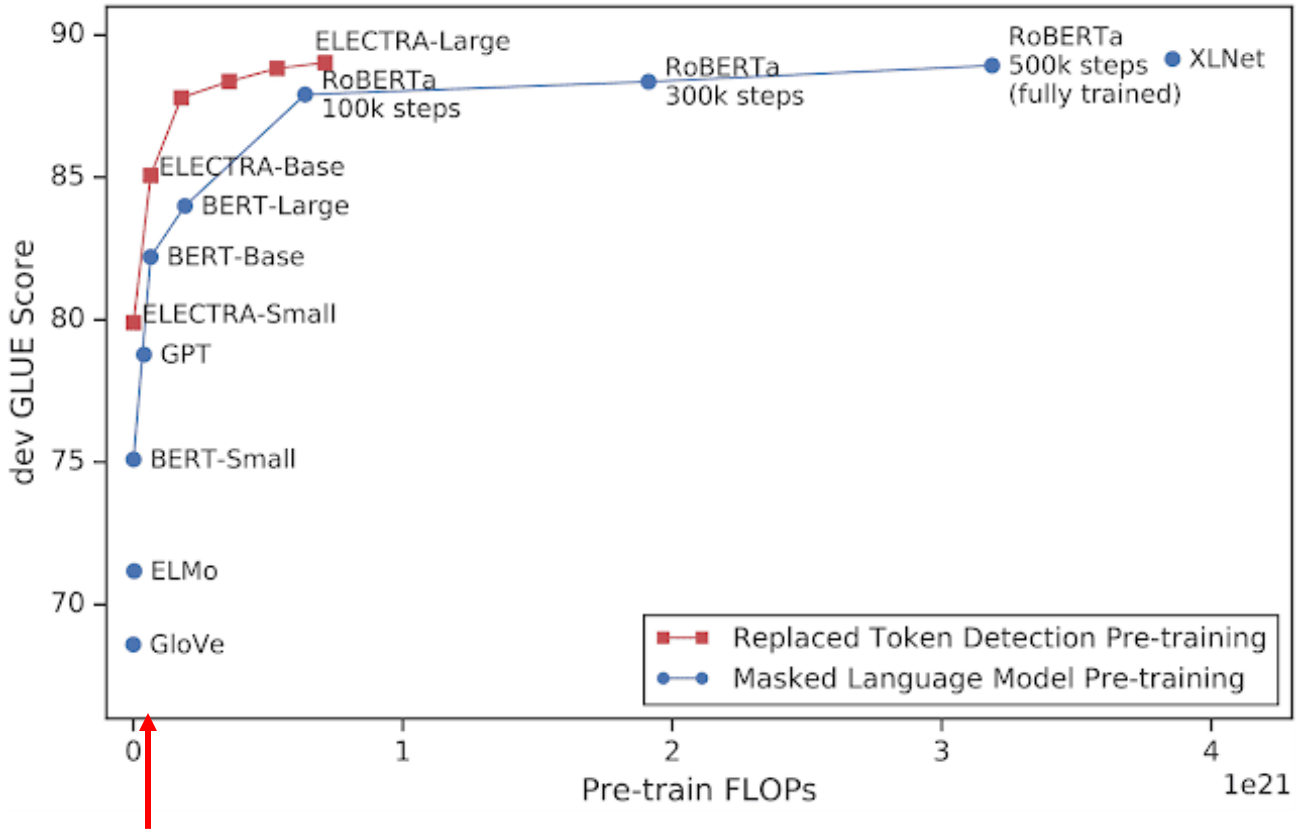
- 通常需要遍历整个数据集几百次
  - **Epoch**：遍历数据集一次
  - **Iteration**：上面的公式更新一次称为一次迭代
  - 收敛通常需要100 – 300 epochs
  - 一个例子：
    - 数据集：ImageNet-2012包含大约128万张训练图像
    - CNN：ResNet-50 包含有约 11.7G MACs（向前+向后传播）以及约 2500万 (25M) 参数
      - 一个 epoch：2*11.7G*1.28*10^6=~30000TFLOPs
      - 一般需要 **90** epochs
    - Nvidia Tesla V100 GPU（32GB显存）：算力约 112 TFLOPS （包含张量核心）
    - 理论结果
      - 最小时间：30000*90/112 = ~24000 seconds = ~**6.7 hours**
    - 实际最快
      - MXNet 在batch-size=256时，处理速度为约 1,457 图/s
      - 90 个 epoch 花费约 90 * 1.28*106 / 1457 = ~79000 seconds = ~**22 hours**

#. https://developer.nvidia.com/deep-learning-performance-training-inference

Transformer 工作负载（训练过程）

- 自然语言处理 (NLP) 中的 Transformer
  - 有许多矩阵乘法运算





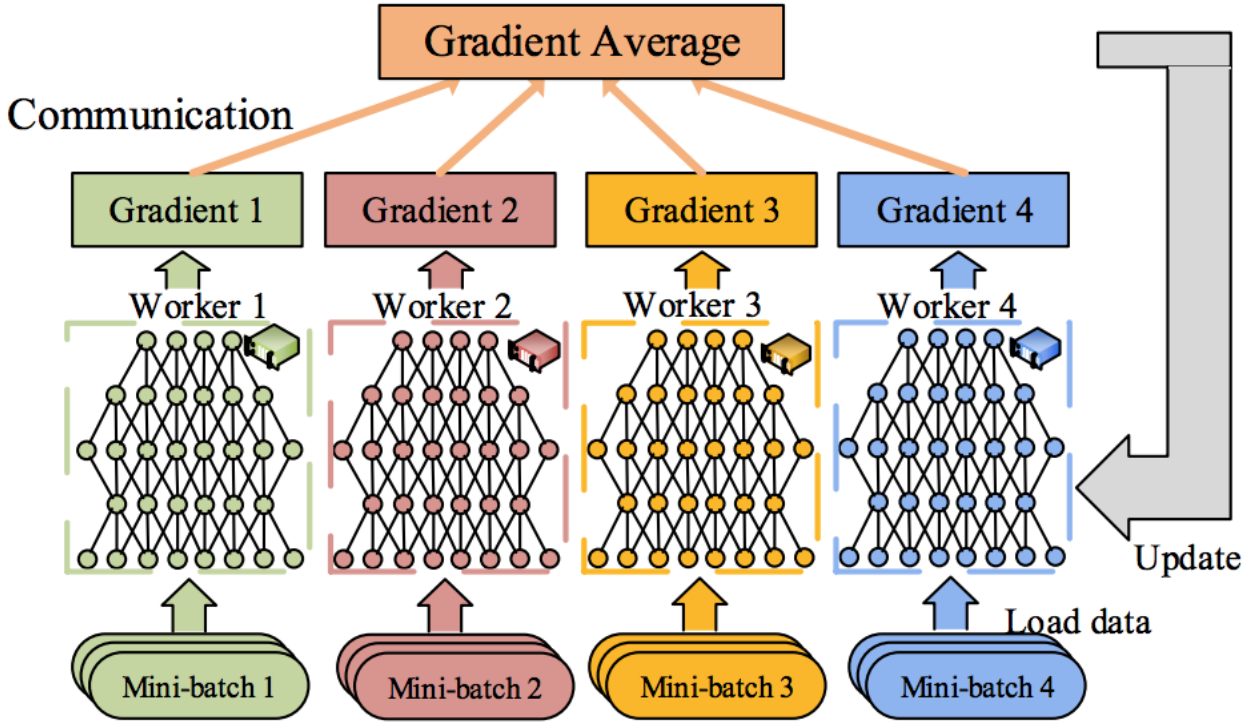
No.1 超级计算机：2.146176e18 FLOPS（半精度 for AI）！

Source:
Vaswani, Ashish, et al. "Attention is All you Need." NIPS. 2017.
https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html

GPU集群如何加速DNN训练

- 数据并行

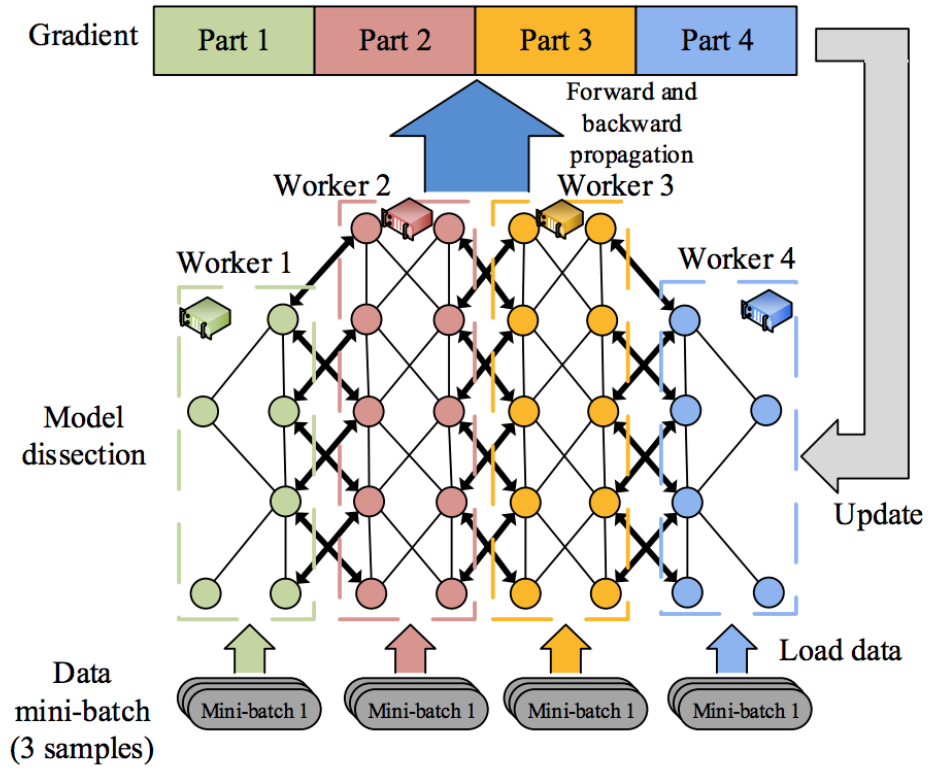
$$W_{i+1} = W_i - \eta \cdot \frac{1}{N} \sum_{j=1}^{N} \nabla L(W_i, D_i^j)$$

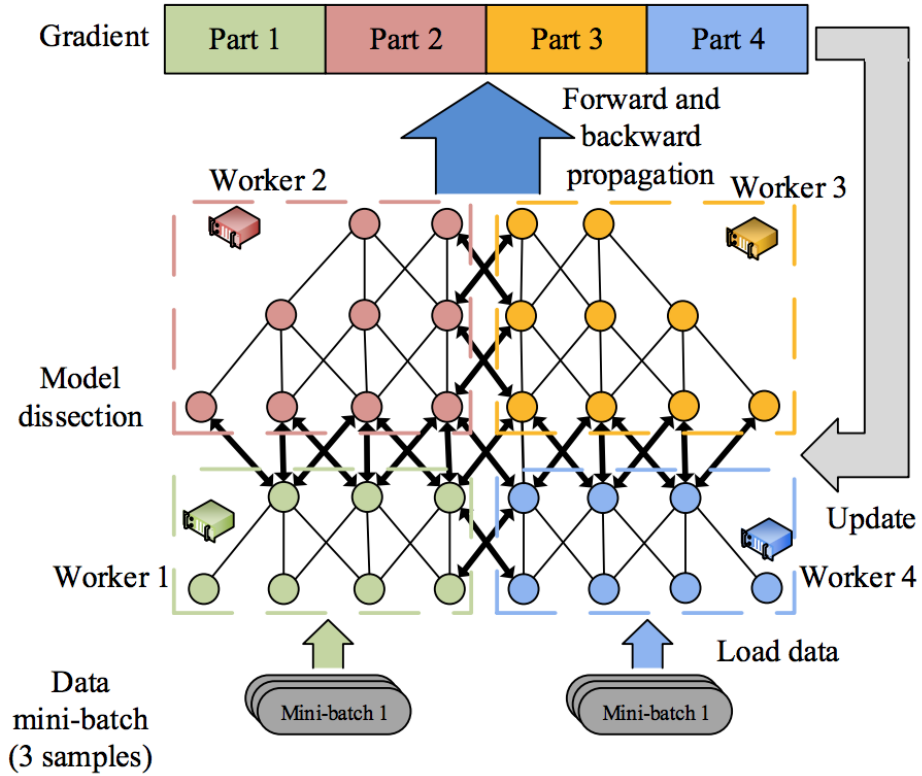


Source: Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020

GPU集群如何加速DNN训练

- 模型并行
  - 两种分割方式

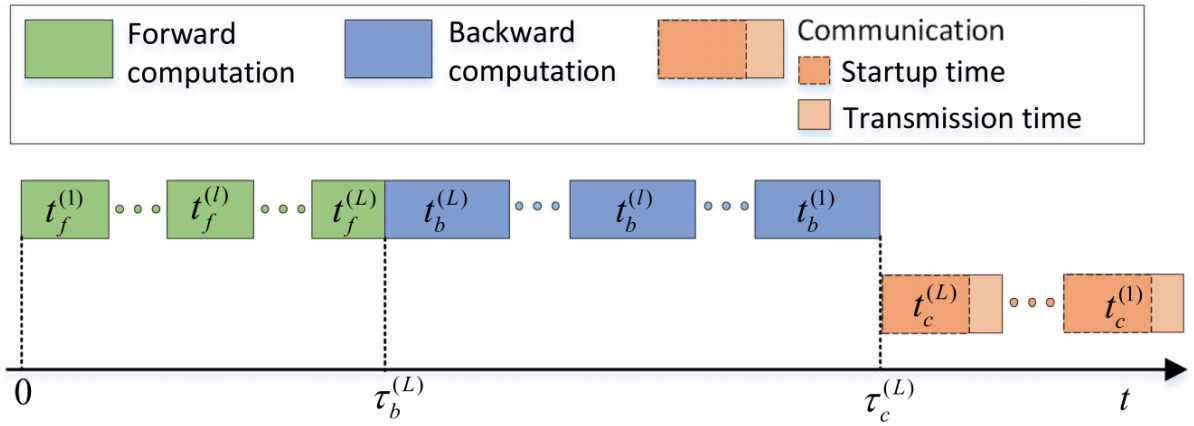




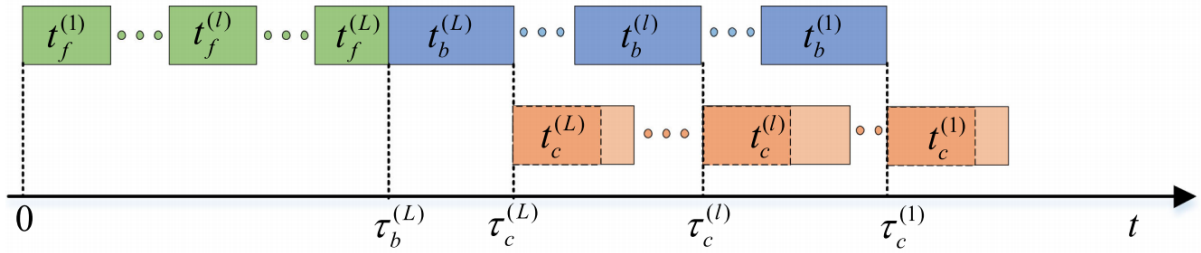
Source: Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020

GPU集群如何加速DNN训练

- 数据并行流水线
  - 梯度通信可与计算重叠进行



(a) Naive S-SGD.

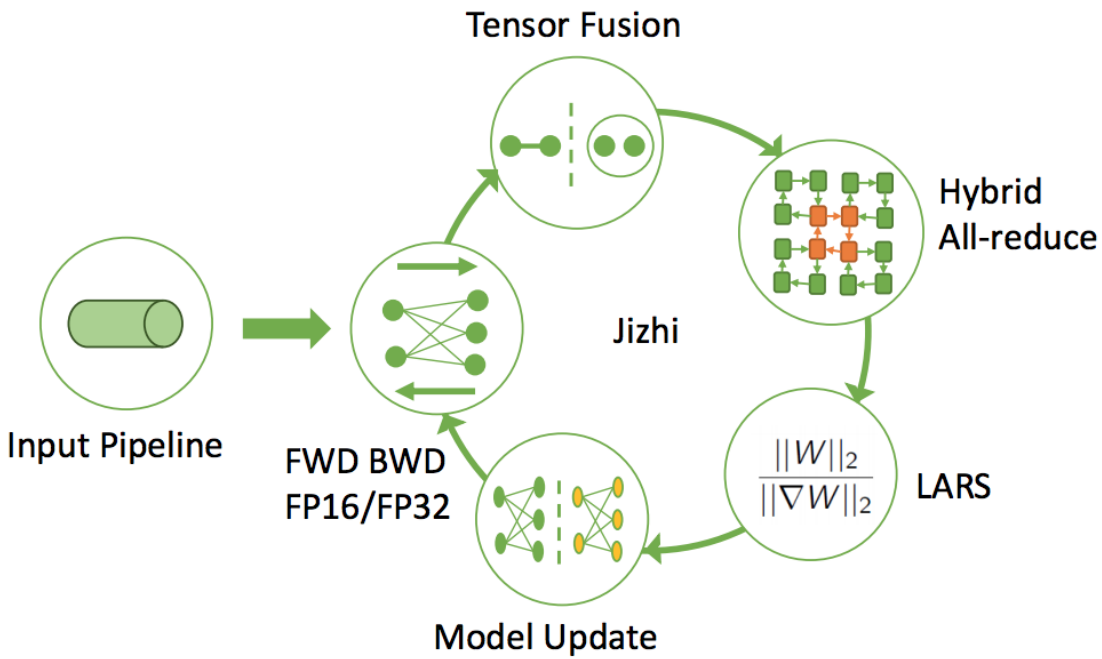


(b) WFBP S-SGD.

Source: Shi et al., MG-WFBP: Merging Gradients Wisely for Efficient Communication in Distributed Deep Learning, TPDS 2021

一个例子：在1024个GPU上训练ResNet-50

- 输入流水线
  - 数据的读取与计算并行：减少 I/O 耗时
  - 输入数据预处理
    - 多进程/多线程/**OpenMP**
- 向前/向后传播
  - 使用 FP16 进行前后向计算：提高GPU吞吐量（**GPU** 计算）
- 张量融合
  - 融合小张量为大张量：提升通信吞吐量
- 混合归约
  - 节点内通信 vs. 节点间通信：更好地利用带宽资源来减少通信时间（**MPI**编程）



- LARS
  - 大 batch 训练的收敛性保证

Source: Jia et al., Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes, NeurIPS workshop 2018

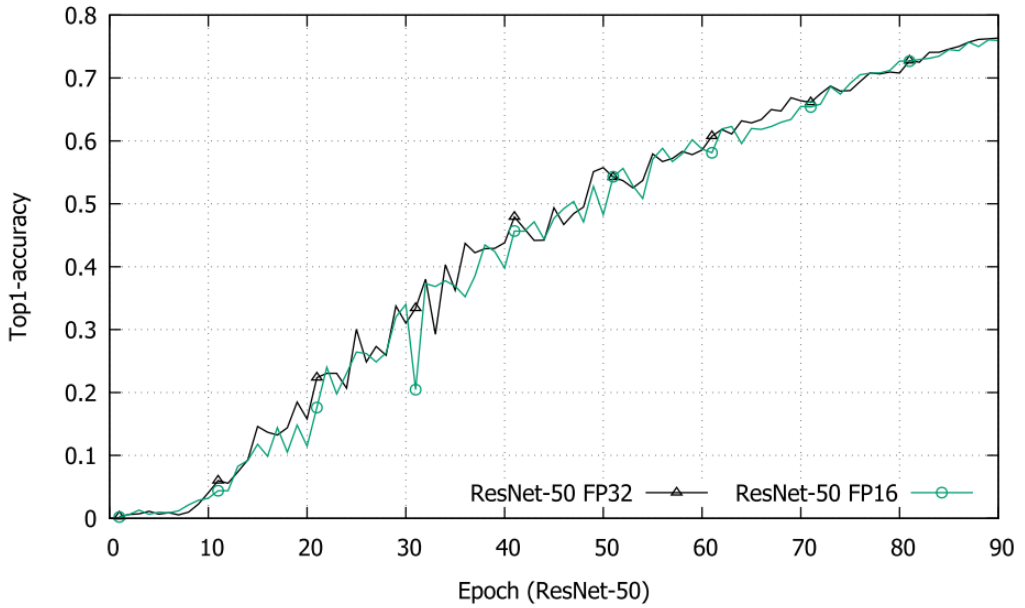
测试平台

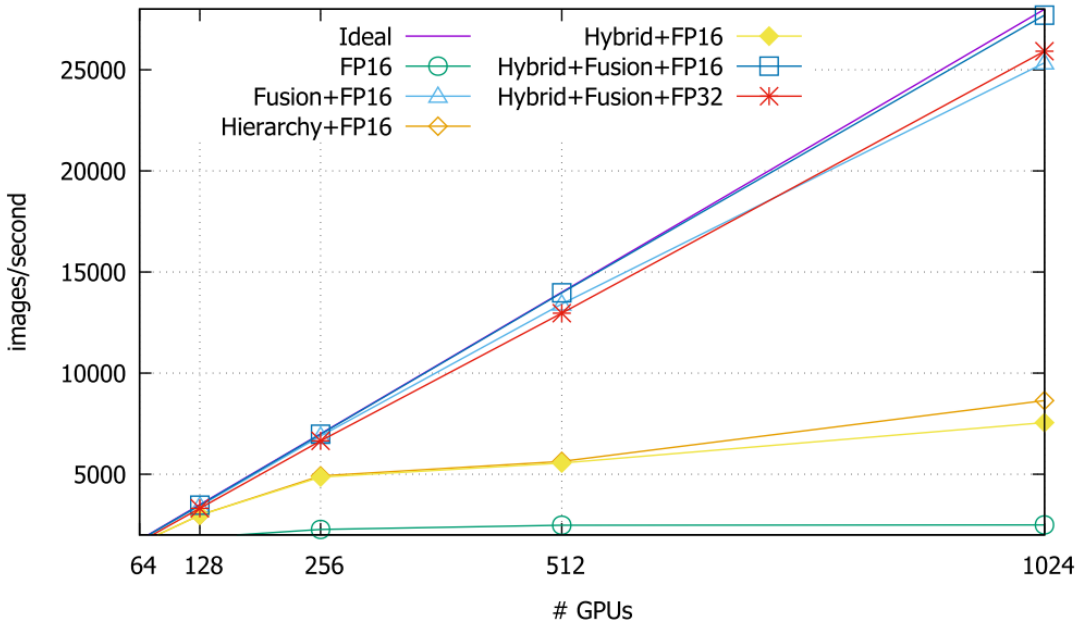
- 硬件
  - 256 个节点 100Gbps 以太网互联 (i.e., Mellanox ConnectX-4 that supports RoCEv2)
    - 将实验 128-节点和 256-节点
  - 每个节点:
    - 8 个 Nvidia Tesla P40 GPUs (PCIe)
    - 2T NVMe SSDs
- 软件
  - TensorFlow：深度学习框架
    - 底层编码：C/C++
    - 高级接口：Python
  - cuDNN：用于GPU上实现DNN计算的高性能 CUDA 库
  - Horovod与OpenMPI：数据并行使用的分布式训练框架
  - NCCL：GPU集群的高性能通信库

Source: Jia et al., Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes, NeurIPS workshop 2018

结果

- 收敛性
  - 训练了 90 个 epoch
  - Top-1 验证精度：76.3%
- 弱缩放性能
  - 单节点(8-GPU)性能：218图/秒， 每个GPU 的 batch_size为 64
  - 128-节点(1024-GPU)性能：约 27500 图/秒
  - 缩放效率：25700/218/128 = ~99.2%
  - 90个epoch， 耗时 **7.8** 分钟
- 256-节点(2048 GPUs)
  - 90个epoch， 耗时 **6.6** 分钟
  - 使用双倍数量的GPU，仅提升了 **18%**
    - 还可以进一步优化！





Source: Jia et al., Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes, NeurIPS workshop 2018

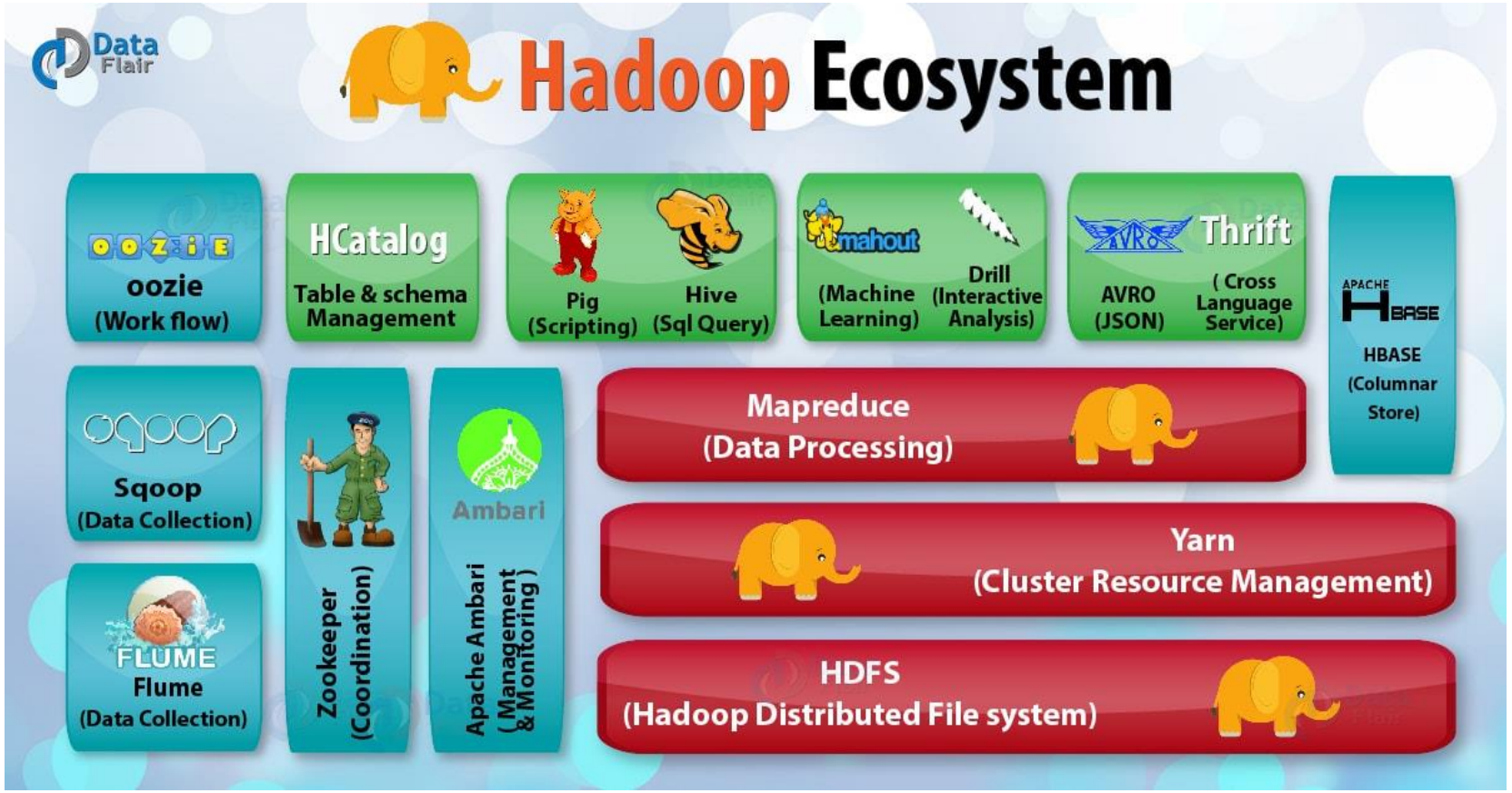
产品级分布式训练框架

- PyTorch-DDP
  - https://pytorch.org/docs/stable/notes/ddp.html
  - 主要用于数据并行
- Horovod
  - https://github.com/horovod/horovod
  - 主要用于数据并行
- DeepSpeed
  - https://github.com/microsoft/DeepSpeed
  - 主要用于模型并行和ZeRO
- Megatron-LM
  - https://github.com/NVIDIA/Megatron-LM
  - 主要用于模型并行
- Ray
  - https://www.ray.io/
  - 通用分布式计算

Narayanan et al., Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, SC '21

Hadoop Ecosystem

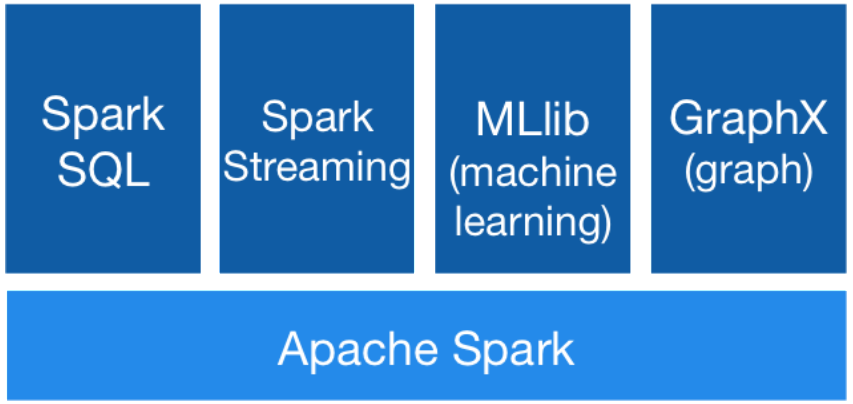
- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
  - https://hadoop.apache.org/



Source: https://data-flair.training/blogs/hadoop-ecosystem-components/

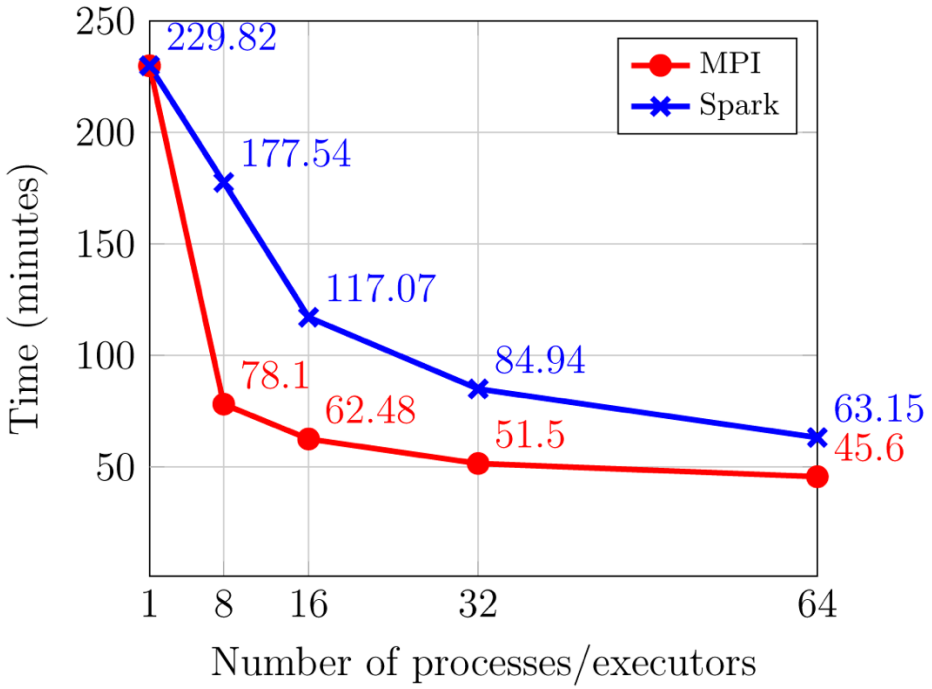
Apache Spark

- Apache Spark is a unified analytics engine for large-scale data processing.
  - Runs on many frameworks
  - High speed: ~100x faster
  - Ease-of-use: supports Java, Scala, Python, R, and SQL.
  - A stack of libraries: SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming

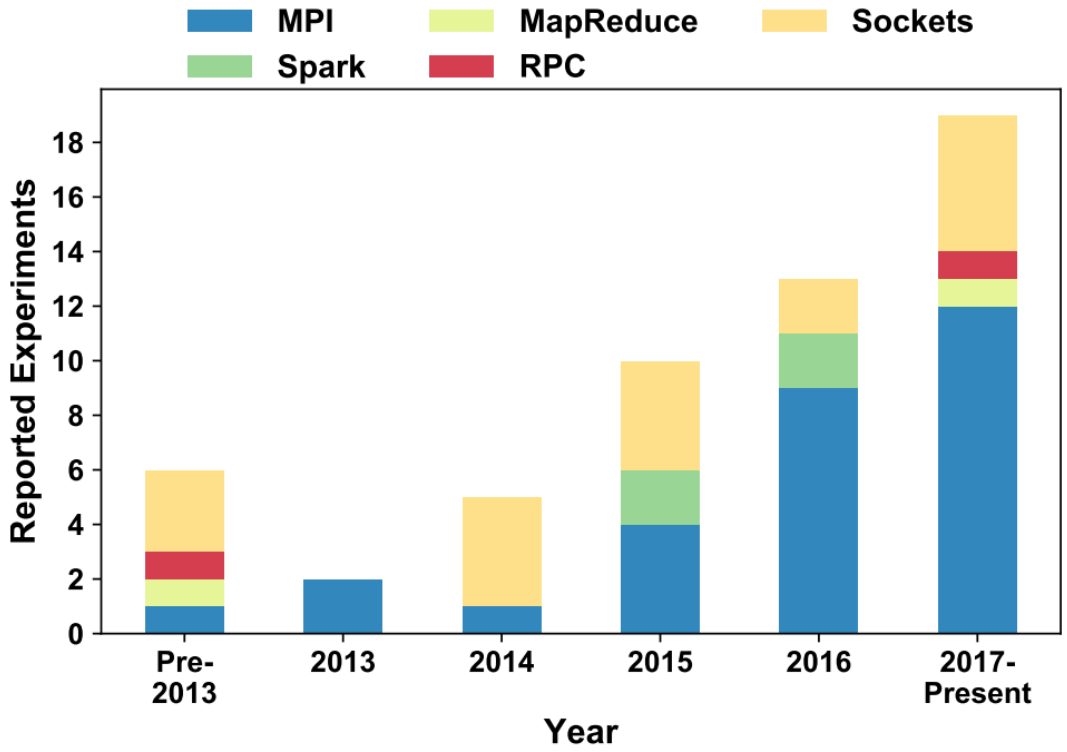


Source: https://spark.apache.org/

Spark vs. MPI



Performance comparison [1]



Deep Learning research [2]

[1] Abuín, J. M., Lopes, N., Ferreira, L., Pena, T. F., & Schmidt, B. (2020). Big Data in metagenomics: Apache Spark vs MPI. *Plos one*, *15*(10), e0239741.
[2] Ben-Nun, T., & Hoefler, T. (2019). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. ACM Computing Surveys (CSUR), 52(4), 1-43.