

Scalable Machine Learning with High Performance and Cloud Computing

Lecture 3: Distributed Deep Learning

ROCCO SEDONA

HIGH PRODUCTIVITY DATA PROCESSING RESEARCH GROUP

JUELICH SUPERCOMPUTING CENTRE (JSC)

IGARSS 2020

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

UNIVERSITY OF ICELAND

OUTLINE

1. Introduction

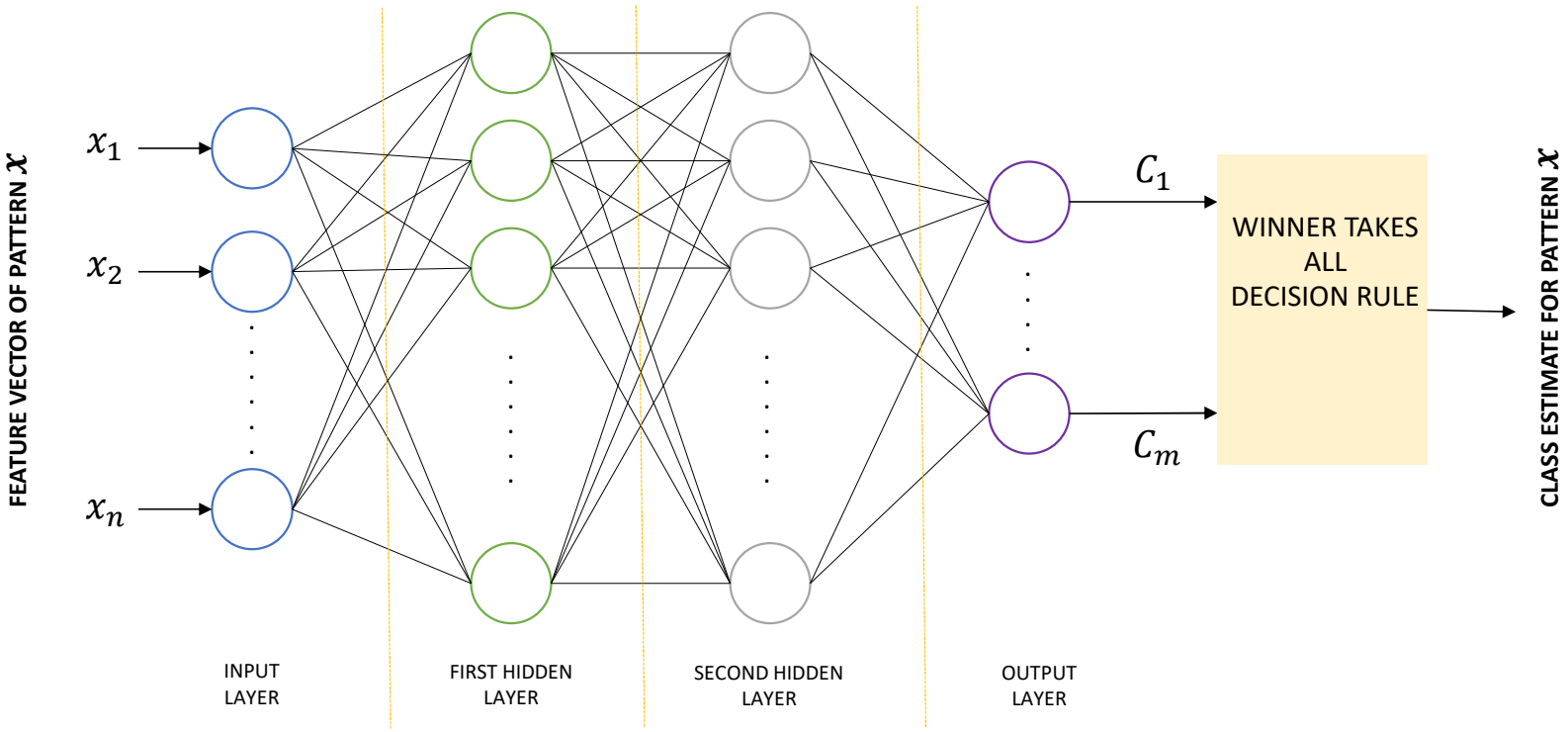
2. Distributed training

3. Horovod

4. DeepSpeed

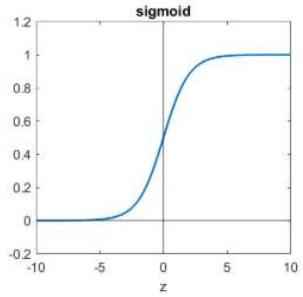
Multilayer Percepton (MLP) Neural Network

- **Forward interconnection** of several layers of perceptrons
- MLPs can be used as **universal approximators**
- In classification problems, they allow modeling **nonlinear discriminant functions**
- Interconnecting neurons aims at increasing the capability of **modeling complex** input-output relationships

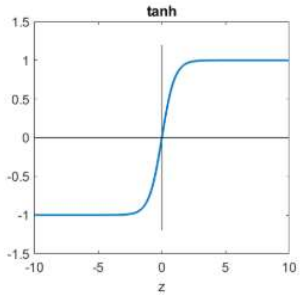


Activation function

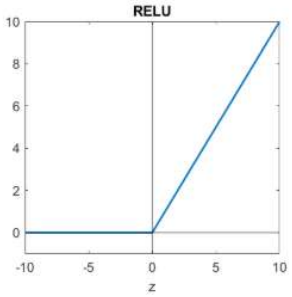
- The choice of the architecture and the **activation function** plays a key role in the definition of the network
  - The step function is not used in practice



$$h(z) = \frac{1}{1 + e^{-z}}$$

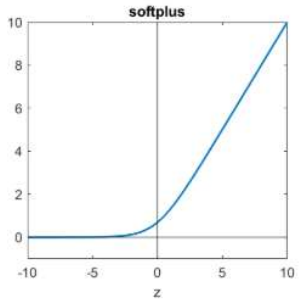


$$h(z) = \tanh z$$

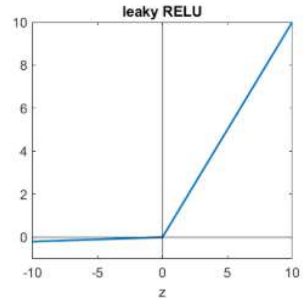


$$h(z) = \max(z, 0)$$

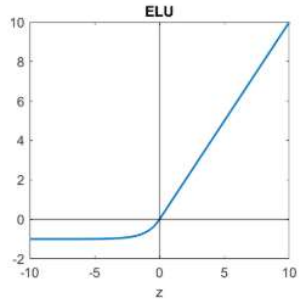
[1]



$$h(z) = \log(1 + e^{z})$$



$$h(z) = \max(z, z\alpha)$$
$$0 < \alpha < 1$$



$$h(z) = \begin{cases} z, z > 0 \\ \alpha(e^{z} - 1) z \leq 0 \end{cases}$$

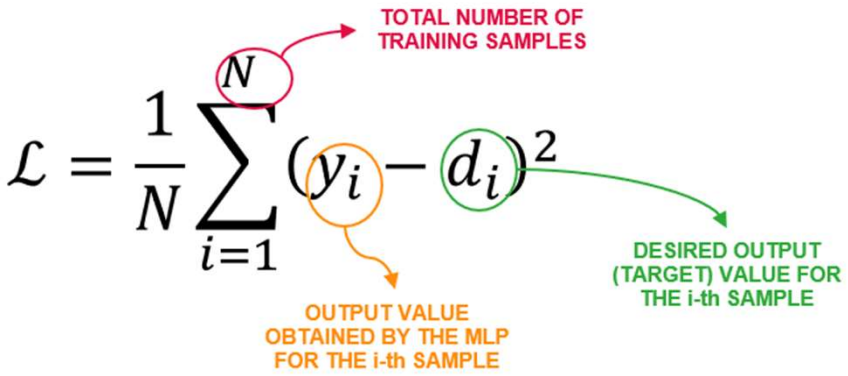
Training

- As for all supervised classifiers, one of the most important issue with ANNs is **how to train them**
- Training means finding an opportune **architecture** and related **weight and bias** values
- The **highly nonlinear** nature of ANNs makes it not trivial to find an analytical solution to the problem
  - Therefore, one has to resort to **numerical optimizers**
- Another problem is the **number of weights and biases** to optimize

$$MLP\ with \begin{cases} 10\ input\ neurons \\ 20\ hidden\ neurons \Rightarrow 325\ weights\ (+biases) \\ 5\ output\ neurons \end{cases}$$

Backpropagation

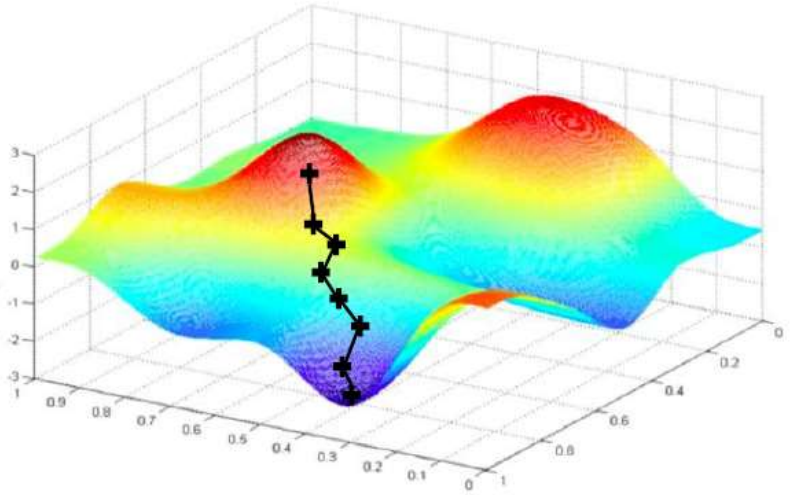
- What weights should be modified (and how much) to obtain correct classification?
  - I.e., Understand what connections are increasing or reducing to the error in the output
- Looking for an algorithm which modifies the different weights to **minimize the error rate**
- **Backpropagation:** iterative algorithm which has hugely contributed to neural network fame
- It is a **gradient-based search** method which allows finding a minimum of the **sum of squared error criterion**



Mini-batch Gradient Descent

- Gives an estimate of the true gradient by averaging the gradient from each of the B points
- Minibatch sampling: implemented by shuffling the dataset S, and processing that permutation by obtaining contiguous segments of size B from it

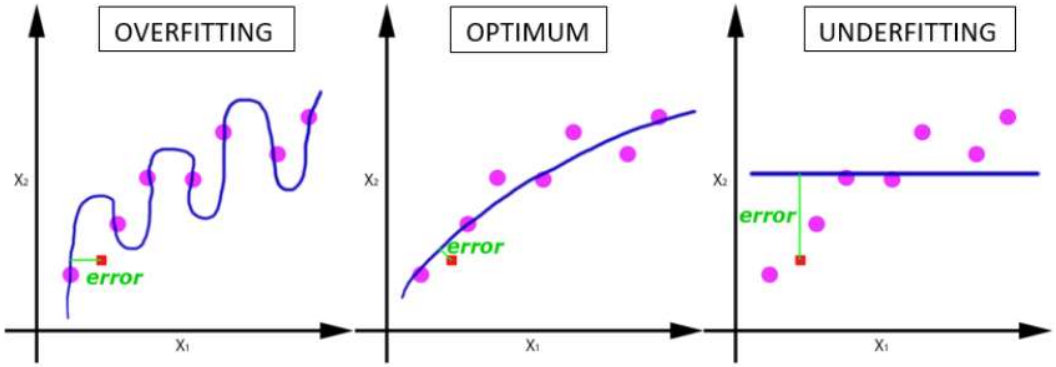
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3. **Pick batch of B data points**
4. Compute gradient $\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights $\boldsymbol{W} \coloneqq \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

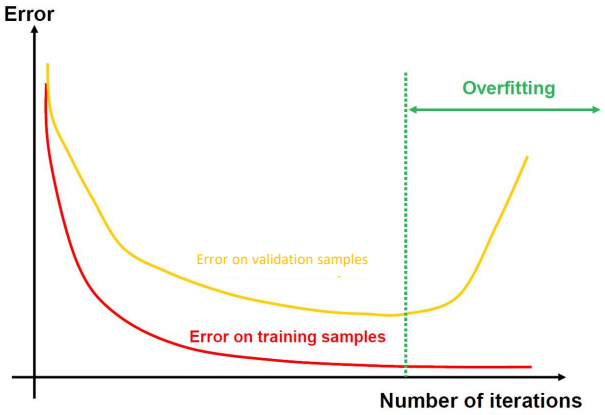


Fast to compute and a much better estimate of the true gradient!

Epochs and iterations

- **1 Epoch**: **entire training** set passed forward and backward through the network in once
  - The training set is divided in batches since the data can be too large
- **1 iteration**: **entire batch** passed forward and backward through the network in once
  - If 1000 training samples and batch size set to 500, it means 2 iterations to complete 1 Epoch





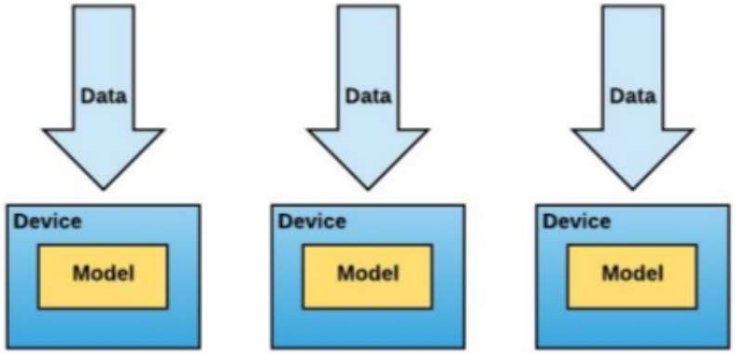
What is the right numbers of epochs?

Distributed training

With Data Parallelism

- **Mini-Batch Gradient Descent:**
  - More accurate estimation of gradient and smoother convergence
  - Allows for larger learning rates (i.e., trust more the gradient , training faster)
  - Can **parallelize computation** and achieve significant speed increases

Send batches across the GPUs, compute their gradient simultaneously and aggregate them back

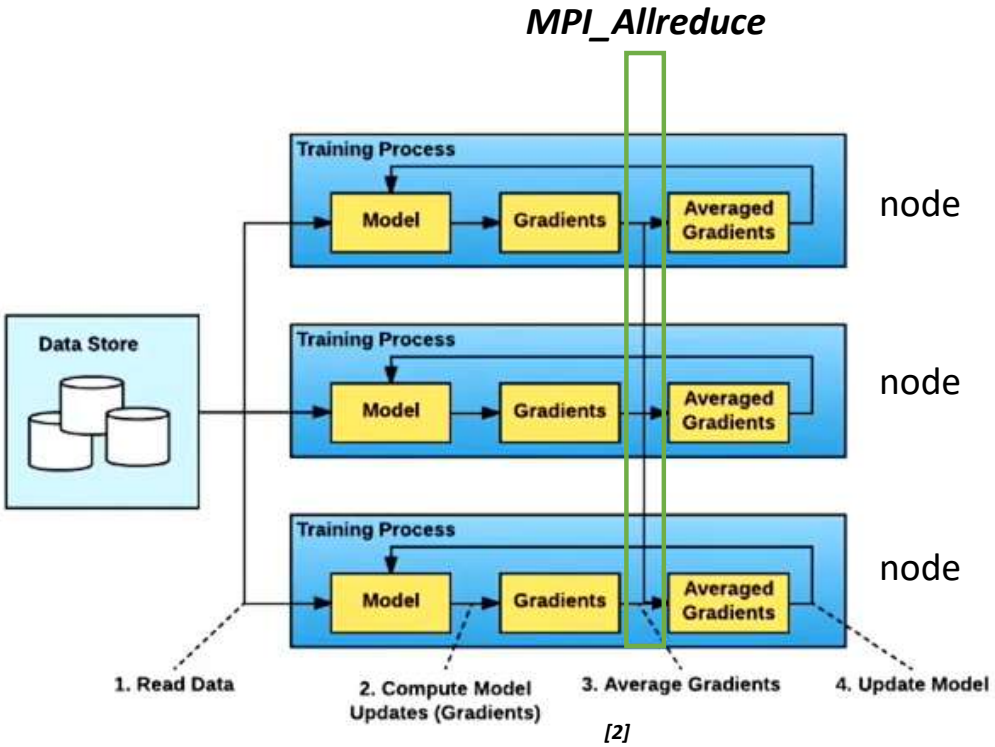


[2]

Distributed training

With Data Parallelism

- The gradients for different batches of data are calculated separately on each node
- But averaged across nodes to apply consistent updates to the model copy in each node

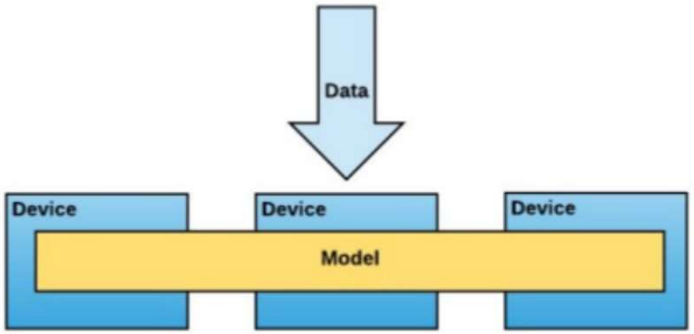


[2]

Distributed training

With Model Parallelism

- minibatch copied to all processors
- different parts of the DNN computed on different processors
- the DNN architecture creates layer interdependencies
- F.e. fully connected layers incur all-to-all communication [3]



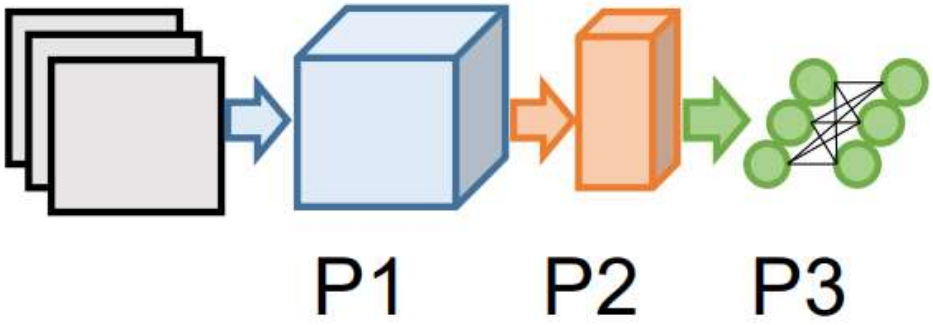
Model Parallelism

[2]

Distributed training

Pipelining

- can either refer to
  1. overlapping computations, i.e., between one layer and the next (as data becomes ready)
  2. or to partitioning the DNN according to depth, assigning layers to specific processors

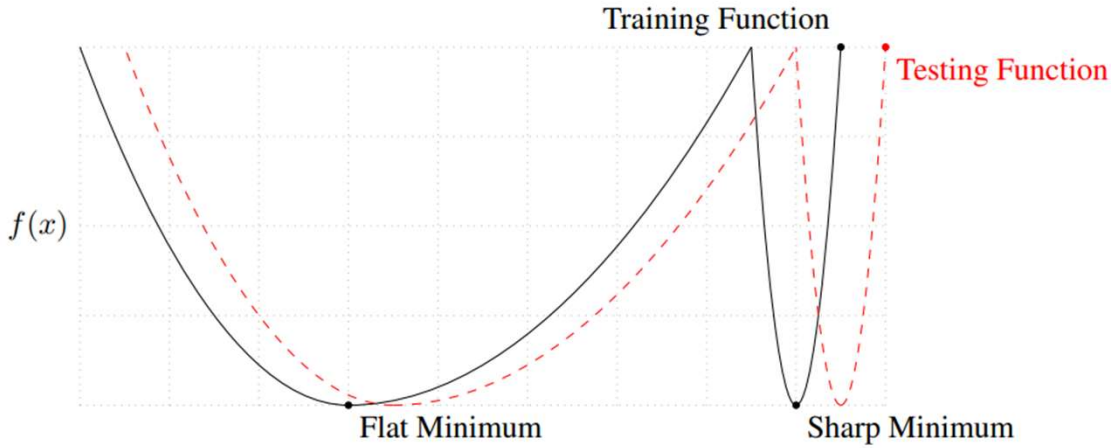


[3]

Challenges of Distributed Learning

However, there are two challenges when using large batch across large clusters

- Larger mini-batch size often leads to lower test accuracy
  - Generalization gap caused by sharp minima [4]
  - Optimization difficulties [5]
- When using large clusters, it is harder to achieve near-linear scalability as the number of machines increases, especially for models with the high communication-to-computation ratio



[4]

Batch normalization

- Problem: internal covariate shift, the change in the distribution of network activations due to the change in network parameters during training
- Objective: reduce the internal covariate shift and improve training
- Batch normalization fixes the means and variances of layer inputs
- Batch Normalization makes training more resilient to the parameter scale
- It helps addressing vanishing and exploding gradients
- It is a form of regularization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

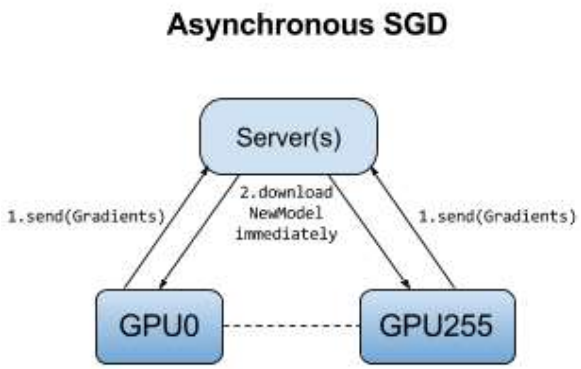
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

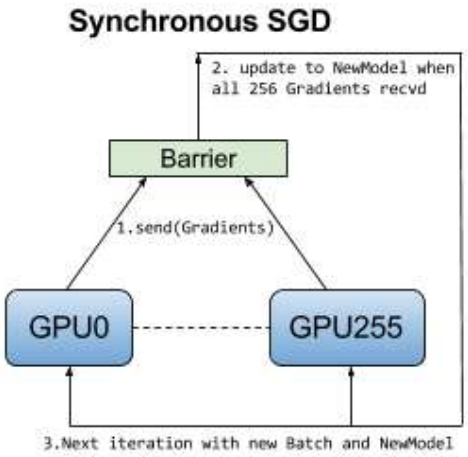
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[6]

Optimization

- SYNCHRONOUS SGD [7]
  - Stragglers, machines which take a long time to respond
  - Presence of synchronization Barrier
  - Convervenge guaranteed
- ASYNCHRONOUS SGD
  - Stale Gradients , some workers could be computing gradients using model weights that may be several gradient steps behind current of global weights
  - convergence not guaranteed



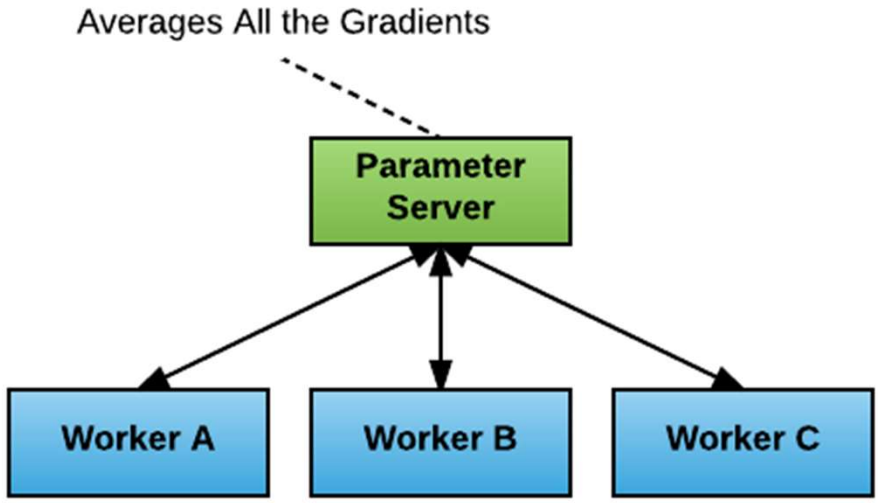


[8]

Global gradient update

Parameter server

- Key-value store dedicated storing variables and does not conduct any computation task
- Adapts one-to-all, and all-to-one collective communication topology for exchanging the gradients and model between servers and workers



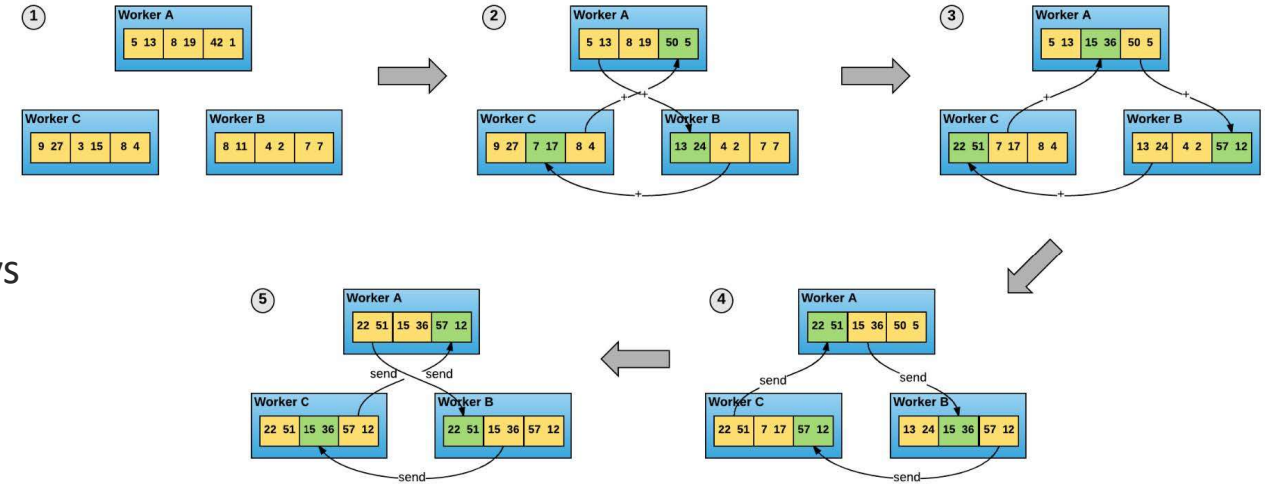
[9]

Global gradient update

Ring allreduce

- Two step process
  1. share-reduce step
  2. share-only step

- 2 ((N/P) ×(P-1)) operations vs 2(N ×(P-1)) in standard allreduce, P processes, N length of data array



[3]

Learning rate policy

Linear policy [5]

- When the minibatch size is multiplied by k, multiply the learning rate by k
- Why? Recall SGD

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

- after k iterations of SGD with learning rate η and a minibatch size of n

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j})$$

- taking a single step with the large minibatch Bj of size kn and learning rate η yields

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t)$$

- Strong assumption $\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$ and setting $\hat{\eta} = k\eta$ would lead to $\hat{w}_{t+1} \approx w_{t+k}$

Learning rate policy

Squared root policy [10]

- Less aggressive than linear policy
- In SGD the weight updates are proportional to the estimated gradient $\Delta \mathbf{w} \propto \eta \hat{\mathbf{g}}$
- the covariance matrix of the parameters update step Δw is:

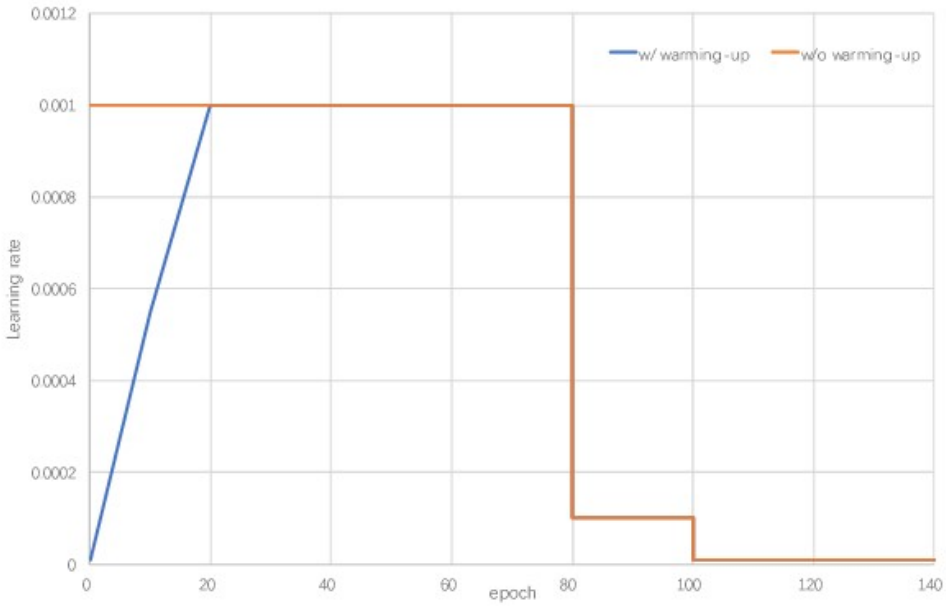
$$\operatorname{cov}(\Delta \mathbf{w}, \Delta \mathbf{w}) \approx \frac{\eta^2}{M} \left(\frac{1}{N} \sum_{n=1}^{N} \mathbf{g}_n \mathbf{g}_n^\top \right)$$

- simple way to make the covariance matrix the same for all mini-batch sizes is to increase the learning rate by the square root of the mini-batch size

$$\eta \propto \sqrt{M}$$

Warm-up

- Early stages of the training: the linear scaling rule breaks down when the network is changing rapidly
- Strategy: using **less aggressive learning rates at the start of training**
- Two types:
  - Constant warmup, constant (lower) learning rate for a few epochs
  - Gradual warmup, ramps up the learning rate from a small to a large value [5]
- After warmup, back to original learning rate schedule



[11]

LARS optimizer

- Layer-wise Adaptive Rate Scaling
- Adaptation of **SGD**
- **Local** LR λ^l is defined for each layer through trust coefficient η

$$\lambda^l = \eta \times \frac{||w^l||}{||\nabla L(w^l)||} \qquad [12]$$

- Magnitude of the update for each layer doesn't depend on the magnitude of the gradient
- Addresses vanishing and exploding gradient problems

Tensor fusion

- An efficient communication strategy in a distributed training system should **maximize the throughput as well as reduce the latency** [13]
- Sizes of gradient tensors to aggregate vary a lot for different types of layers
- Usually, gradient tensor sizes for convolution layers are much smaller than fully-connected layers.
- Sending too many small tensors in the network will not only cause the bandwidth to be under-utilized but also increase the latency
- The core idea of tensor fusion is to **pack multiple small size tensors together** before all-reduce to better utilize the bandwidth of the network
- Set parameter θ. In the backward phase, tensors are fused into a buffer pool if the total size is less than θ
- Send the fused tensor out for all-reduce when the total size is larger than θ

Horovod

Horovod

- **data parallel**, each GPU has a copy of the model and a chunk of the data
- Efficient **decentralized framework**,

based on MPI and NCCL libraries, where actors exchange parameters without the need of a parameter server

- Works on top of Keras, TensorFlow, PyTorch and Apache MXNet

Tensorflow

- Parameter server for asynchronous training
- Mirrored strategy for synchronous training

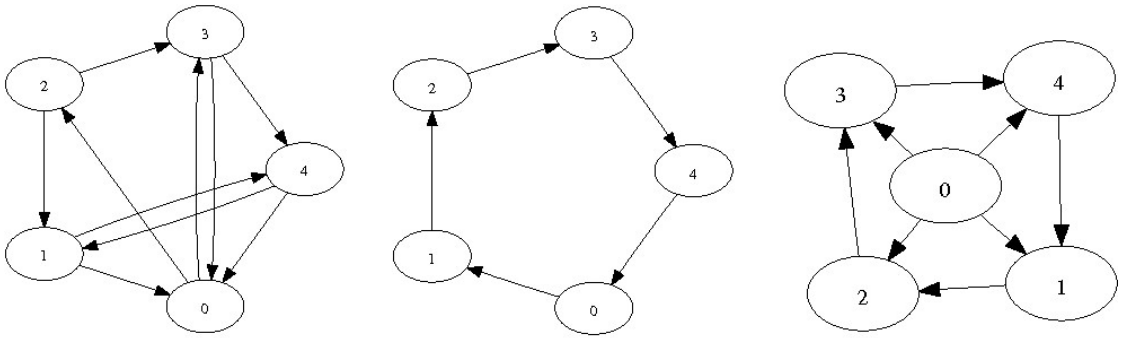
Pytorch

- Distributed Data-Parallel Training (DDP)
- RPC-Based Distributed Training supports general training structures

[3]

What is MPI?

- MPI is a **standard** for **exchanging messages** between multiple computers running a parallel program across **distributed memory**
- point-to-point and collective communication are supported
- **Different topologies** can be implemented
- Parallel I/O operations
- Blocking and non blocking statements

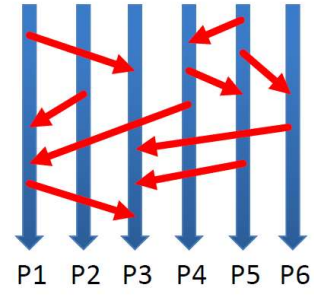


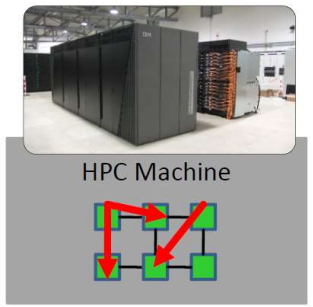
(a) Random (b) Ring (c) Wheel

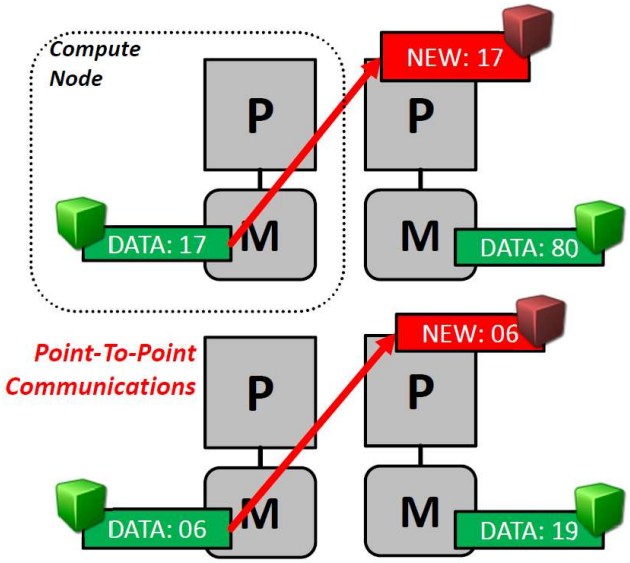
[14]

Message Passing: Exchanging Data

- Each processor has its own data and memory that cannot be accessed by other processors

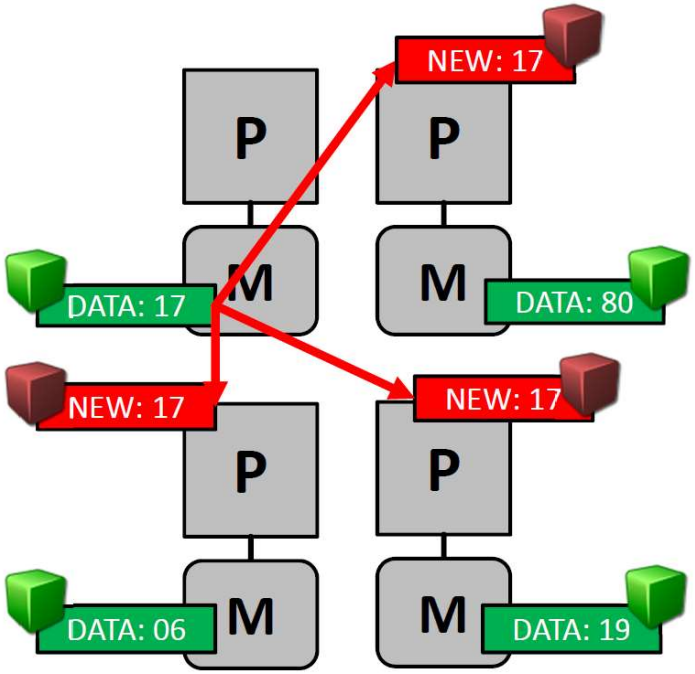






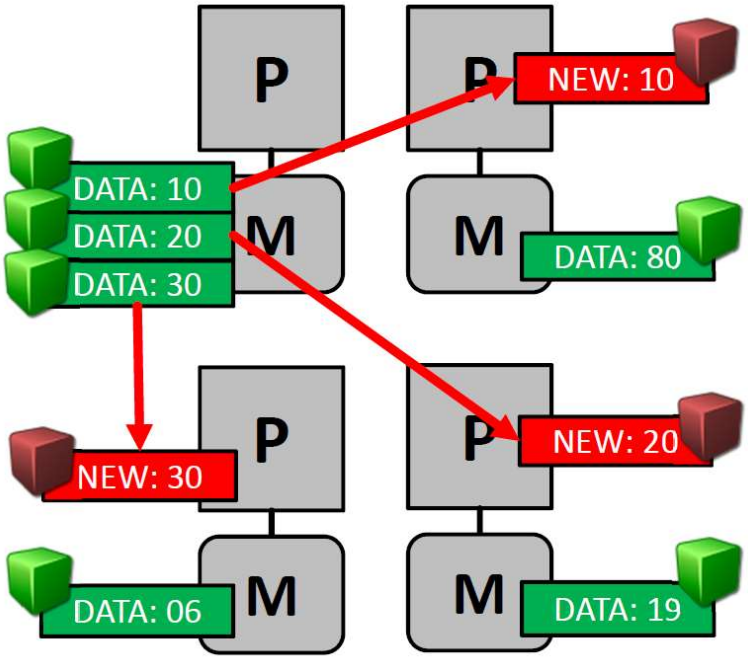
Collective Functions: Broadcast (one-to-many)

- Broadcast distributes the **same** data to many or even all other processors



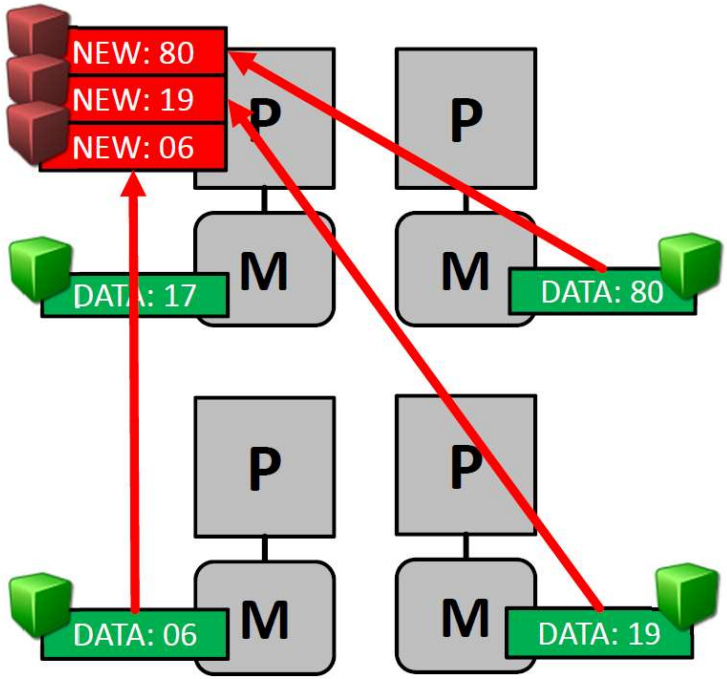
Collective Functions: Scatter (one-to-many)

- Scatter distributes different data to many or even all other processors



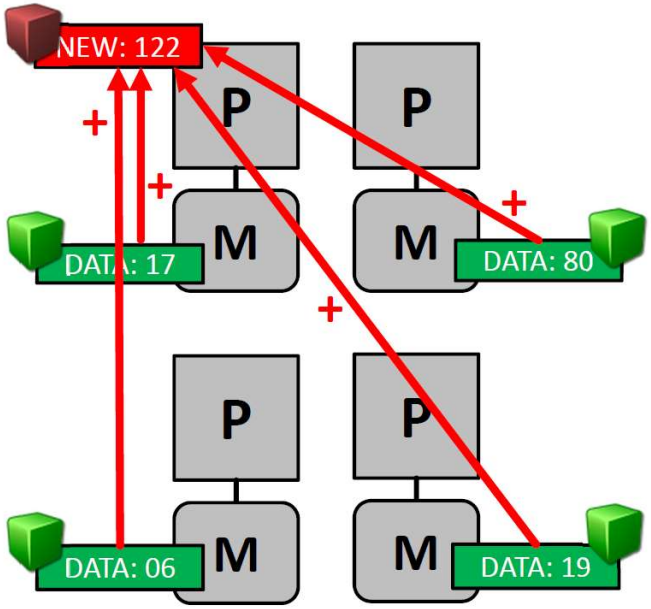
Collective Functions: Gather (many-to-one)

- Gather **collects** data from many or even all other processors to one specific processor



Collective Functions: Reduce (many-to-one)

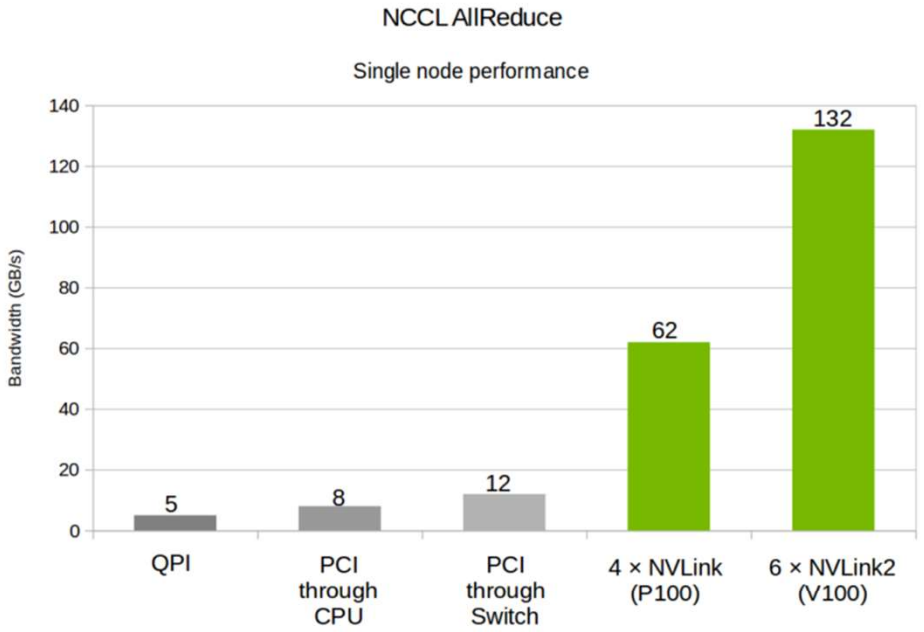
- Each Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a **global minimum or maximum, sum, or product** of the different data located at different processors

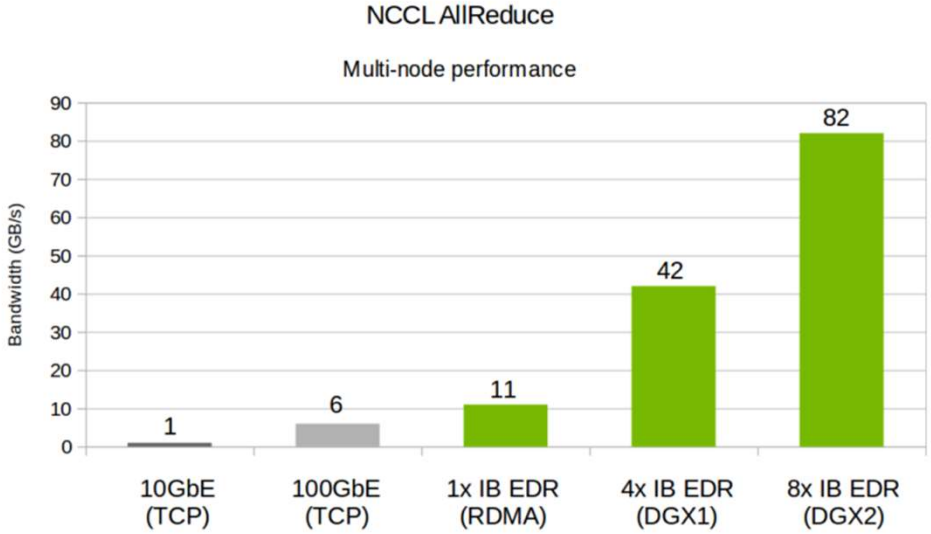


+ global sum as example

NCCL

- NVIDIA Collective Communications Library (NCCL) [15]
- Provides **optimized implementation of inter-GPU communication** operations, such as allreduce and variants
- Optimized for **high bandwidth and low latency** over PCI and NVLink high speed interconnect for intra-node communication
- Sockets and InfiniBand for inter-node communication





Horovod flags

- Hierarchical Algorithm: first allreduce locally, then allreduce across nodes in parallel
- Tensor Fusion: attempting to combine all the tensors that are ready to be reduced at given moment of time into one reduction operation
- Gradient Compression: compress all floating-point gradients to 16-bit
- Auto Tuning: uses Bayesian Optimization for parameter optimization

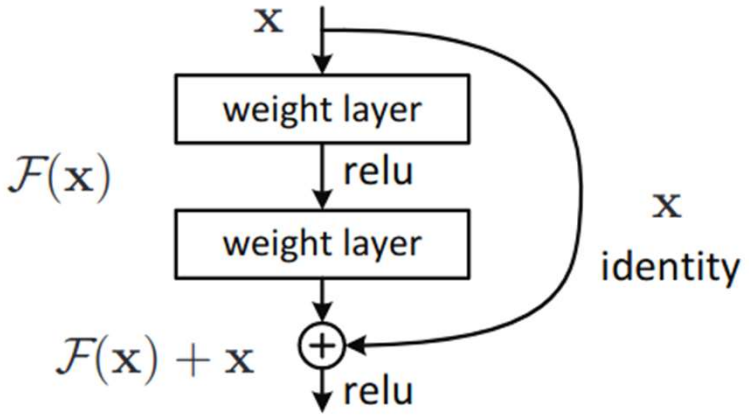
Setup

HPC

- The experiments carried out on JUWELS and JUSUF supercomputer,
- Experiments using 32 nodes, i.e., 128 GPUs

ResNet50

- ResNet-50 is CNN
- Overcomes the difficulties of training with a large number of layers (vanishing gradient problem) by using skip connections



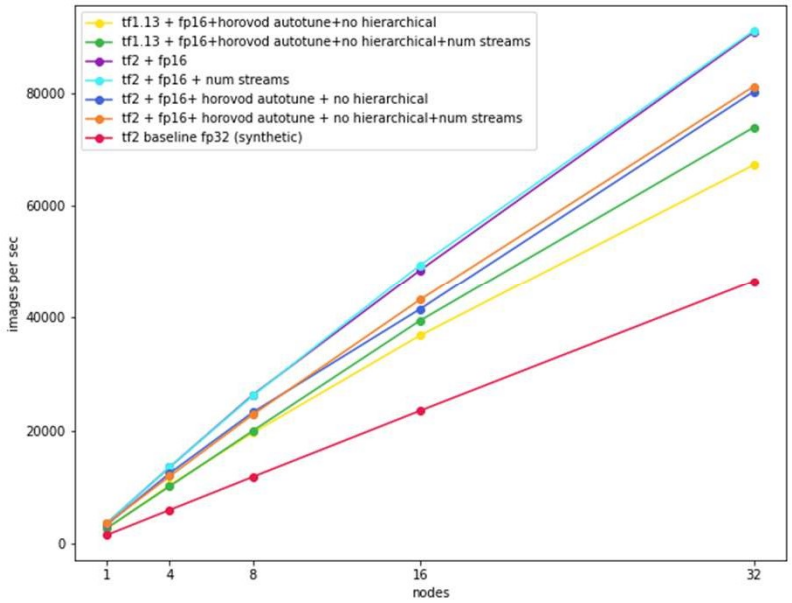
[16]

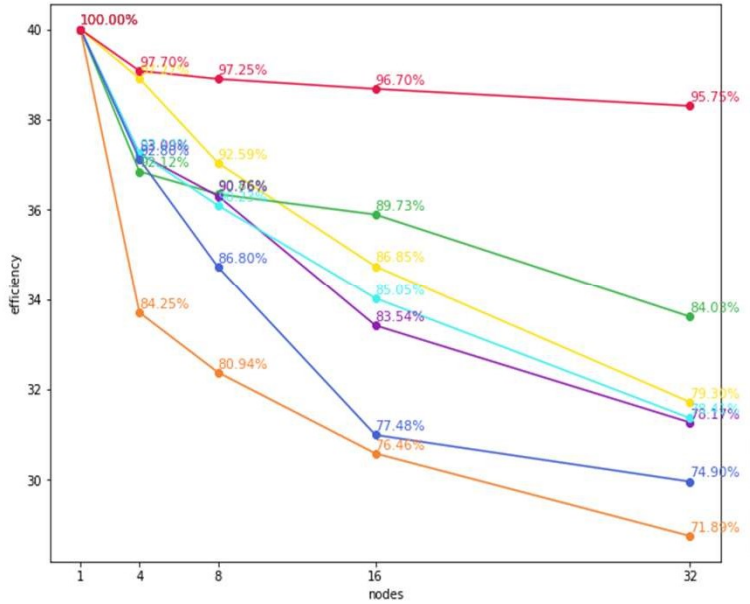
General considerations on Horovod 1

Technical analysis by *M. Cherti, A. Strube, J. Jitsev* (Helmholtz AI Local at JSC)

Experiments on JUWELS, batch size=64

- Horovod autotune introduces overhead and might cause oscillations of efficiency
- Hierarchical did not improve efficiency





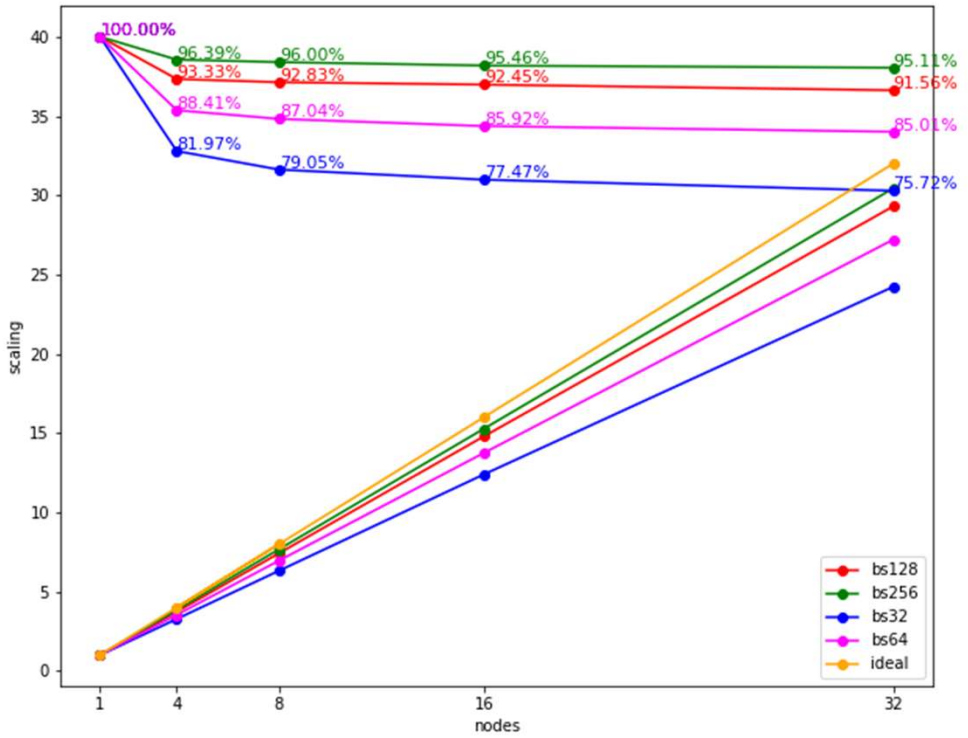
[17]

General considerations on Horovod 2

Experiments on JUSUF

- FP16: more images per sec than FP32 but worse scaling efficiency due to idle GPUs (NVProf, NVVP, Horovod timeline)
- FP16 scales worse because GPU usage is not optimal and NCCL reduce operations for communication become dominant

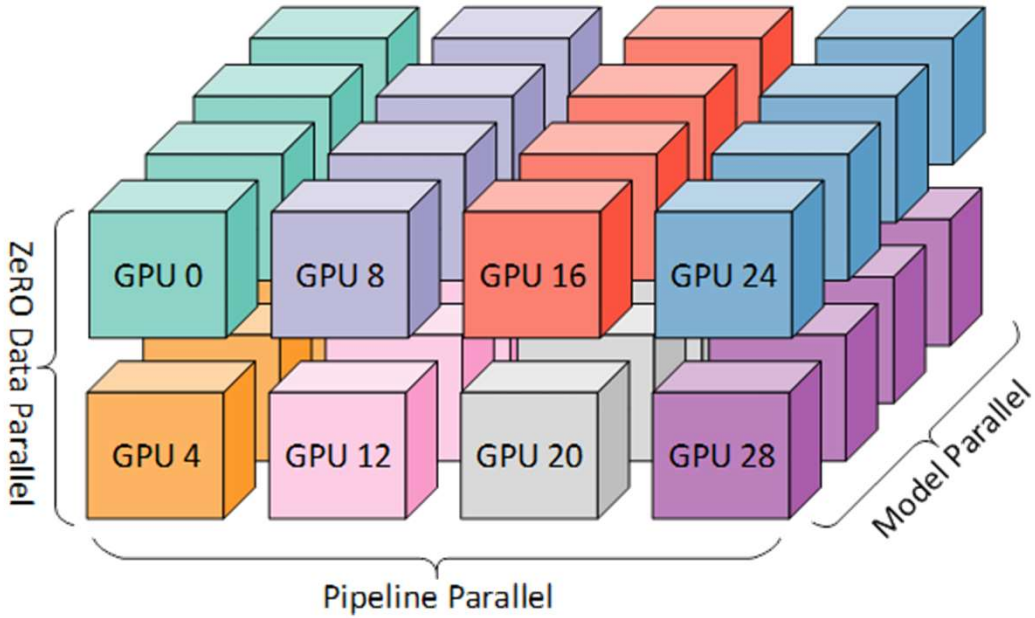
- Larger batch size leads to better scaling



[17]

DeepSpeed: 3D parallelism

- Released in May 2020 by Microsoft
- DeepSpeed is optimized for low latency, high throughput training
- 3D Parallelism to train models with up to 1 trillion parameters



[18]

DeepSpeed: ZeRO

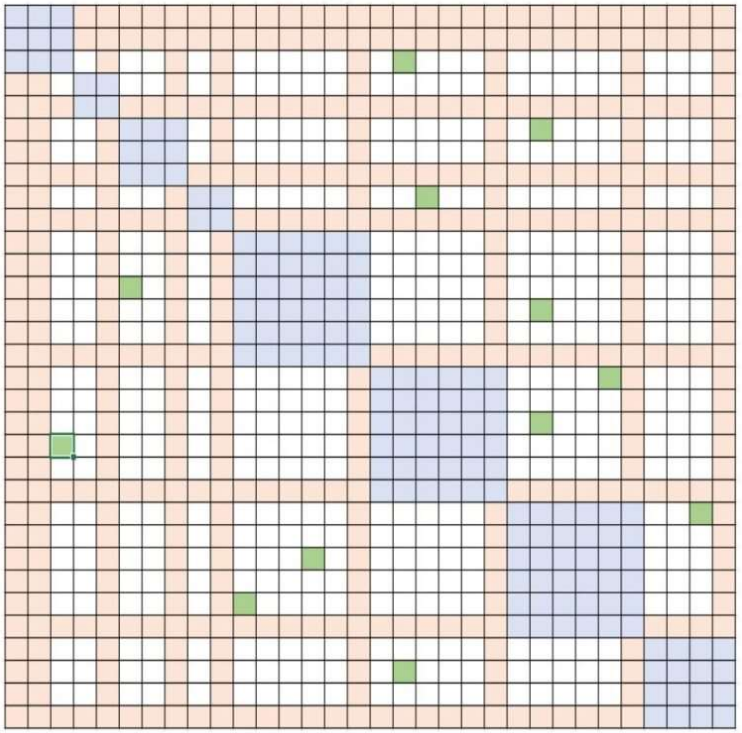
- **10x bigger model training on a single GPU with ZeRO-Offload,** leveraging both CPU and GPU memory for training large models
- Models **of up to 13 billion parameters** on **a single NVIDIA V100 GPU** without running out of memory, 10x bigger than the existing approaches
- ZeRO removes the memory redundancies across data-parallel processes by partitioning the three model states (optimizer states, gradients, and parameters) across data-parallel processes instead of replicating them
- ZeRO-Offload democratizes large model training by making it possible even on a single GPU. It is based on ZeRO 2

[18]

DeepSpeed: Sparse Attention

- **Powering 10x longer sequences and 6x faster execution through DeepSpeed Sparse Attention**
- Attention-based deep learning models, such as Transformers, are highly effective in capturing relationships between tokens in an input sequence, even across long distances
- SA can also allow random attention or any combination of local, global, and random attention

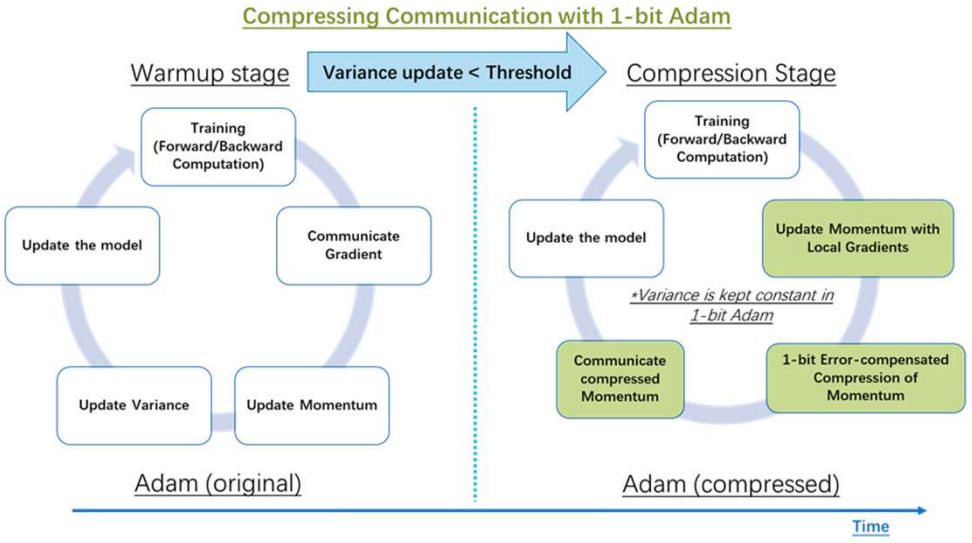
[19]



[18]

DeepSpeed: 1-bit Adam

- **1-bit Adam with up to 5x communication volume reduction**
- consists of two parts:
  1. the **warmup** stage, which is the vanilla Adam algorithm
  2. the **compression** stage, which keeps the variance term constant and compresses the remaining linear term, that is the momentum, into 1-bit representation



[18]

Thanks for the attention!

References:

[1] http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2019/www/hwnotes/HW1p1.html

[2] A. Sergeev and M. D. Balso, “Horovod: Fast and Easy Distributed Deep Learning in TensorFlow”, arXiv:1802.05799, 2018.

[3] Ben-Nun, T., & Hoefler, T. (2019). Demystifying Parallel and Distributed Deep Learning. ACM Computing Surveys, 52(4), 1–43. https://doi.org/10.1145/3320060

[4] N. S. Keskar and D. Mudigere and J. Nocedal and M. Smelyanskiy and P.T.P. Tang, On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, 2016

[5] P. Goyal and P. Dollár and R. Girshick and P. Noordhuis and L. Wesolowski and A. Kyrola and A. Tulloch and Y. Jia and K. He, ccurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018

[6] S. Ioffe and C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, 2015

[7] K. Chahal and M. Singh Grover and K. Dey, A Hitchhiker's Guide On Distributed Training of Deep Neural Networks, 2018

[8] https://www.oreilly.com/library/view/hands-on-convolutional-neural/9781789130331/ab605c5f-d9a4-4271-8e97-f162832a290d.xhtml

[9] https://eng.uber.com/horovod/

Thanks for the attention!

References:

[10] E. Hoffer, I. Hubara, D. Soudry, Train longer, generalize better: closing the generalization gap in large batch training of neural networks, 2017

[11] https://www.researchgate.net/figure/Our-learning-rate-schedules-with-or-without-warming-up-Blue-one-is-with-warming-up-and_fig5_326144703

[12] Y. You, I. Gitman, B. Ginsburg, Large Batch Training of Convolutional Networks, 2017

[13] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, X. Chu, Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes, 2018

[14] Hoefler, T., Rabenseifner, R., Ritzdorf, H., de Supinski, B. R., Thakur, R., & Träff, J. L. (2010). The scalable process topology interface of MPI 2.2. Concurrency and Computation: Practice and Experience, 23(4), 293–310. https://doi.org/10.1002/cpe.1643

[15] https://developer.nvidia.com/blog/scaling-deep-learning-training-nccl/]

[16] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90

[17] Technical analysis by M. Cherti, A. Strube, J. Jitsev (Helmholtz AI Local at JSC)

[18] https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

[19] *Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.*