Cloud Computing & Big Data

PARALLEL & SCALABLE MACHINE LEARNING & DEEP LEARNING

Rocco Sedona

PhD student at the University of Iceland
High Productivity Data Processing research group
JUELICH SUPERCOMPUTING CENTRE (JSC)

LECTURE 7

@Morris Riedel @MorrisRiedel @MorrisRiedel

Deep Learning Applications in Clouds

October 22, 2020
Online Lecture

EUROPEAN OPEN SCIENCE CLOUD · EOSC NORDIC · EuroHPC Joint Undertaking · ADMIRE · EURO

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum · JÜLICH SUPERCOMPUTING CENTRE · DEEP Projects · HELMHOLTZAI ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Outline of the course

1. Cloud Computing & Big Data Introduction
2. Machine Learning Models in Clouds
3. Apache Spark for Cloud Applications
4. Virtualization & Data Center Design
5. Map-Reduce Computing Paradigm
6. Deep Learning driven by Big Data
7. Deep Learning Applications in Clouds
8. Infrastructure-As-A-Service (IAAS)
9. Platform-As-A-Service (PAAS)
10. Software-As-A-Service (SAAS)
11. Big Data Analytics & Cloud Data Mining
12. Docker & Container Management
13. OpenStack Cloud Operating System
14. Online Social Networking & Graph Databases
15. Big Data Streaming Tools & Applications
16. Epilogue

+ additional practical lectures & Webinars for our hands-on assignments in context

- Practical Topics
- Theoretical / Conceptual Topics

Outline

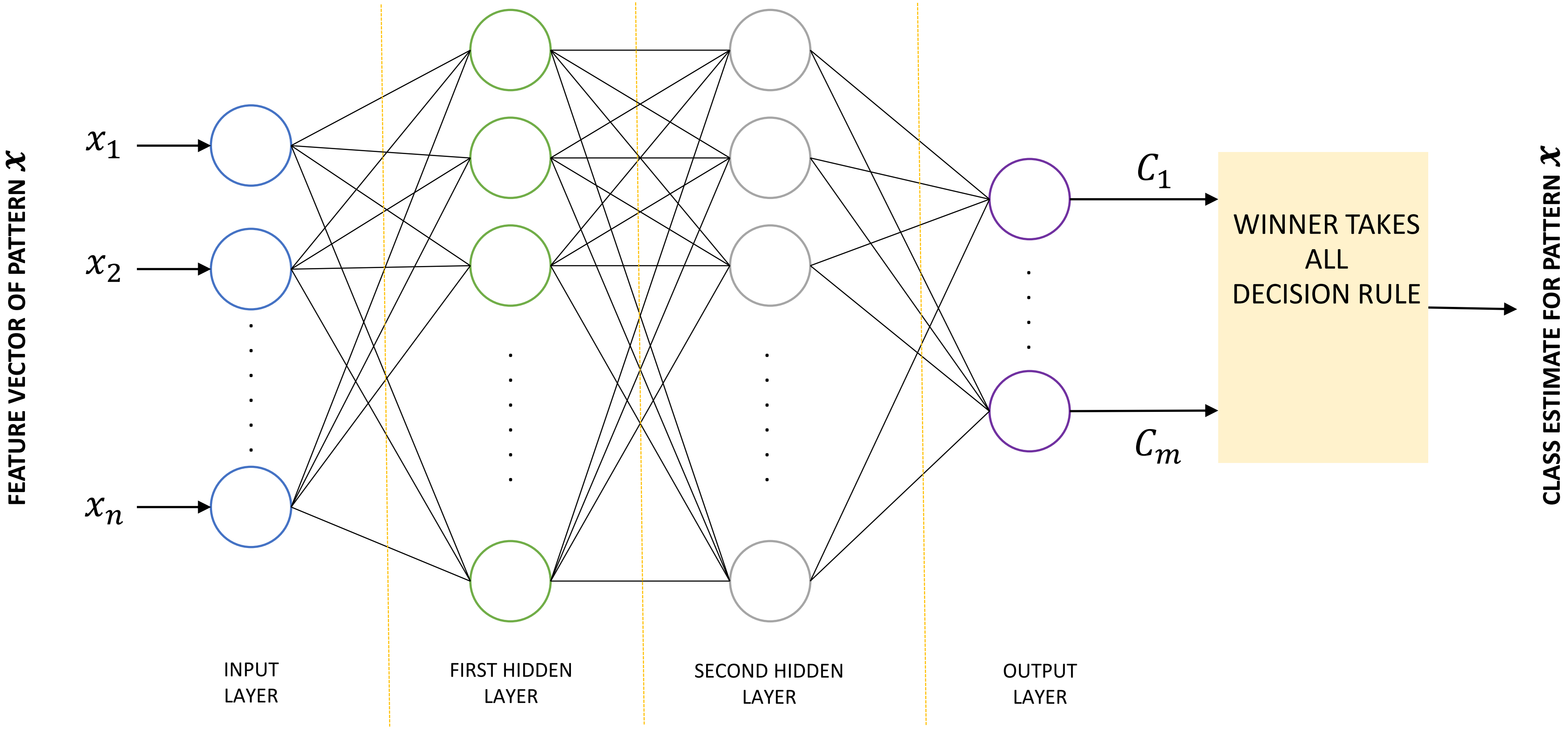
- Introduction
  1. Key Concepts from Lecture 6
  2. Other Important Concepts
- Distributed Training: Theory
- Frameworks
  1. Horovod
  2. DeepSpeed
- A Remote Sensing Application

- **Promises from previous lecture(s):**
- ***Practical Lecture 0.1:* Lecture 6 & 7 will provide more insights into deep learning algorithms and networks including the use of TensorFlow and Keras libraries**
- ***Practical Lecture 0.1:* Lecture 6 & 7 will provide more details on how artificial neural networks (ANNs) and deep learning networks can be used with this data**
- ***Lecture 2:* Lectures 6 & 7 offer more details on feature selection concepts including working with spatial aspects in image recognition tasks**
- ***Lecture 3:* Lecture 6 & 7 offer insights of how to use deep learning with cutting-edge GPUs via Google 'colab' notebooks within the Google Cloud**

Introduction

Multilayer Percepton (MLP) Neural Network (Lecture 6)

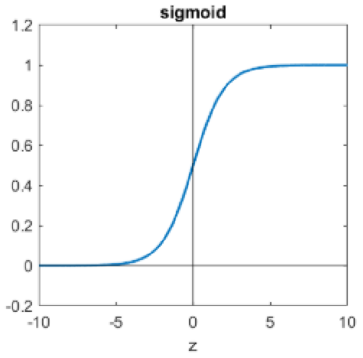
- Forward interconnection of several layers of perceptrons
- MLPs can be used as universal approximators
- In classification problems, they allow modeling nonlinear discriminant functions
- Interconnecting neurons aims at increasing the capability of modeling complex input-output relationships



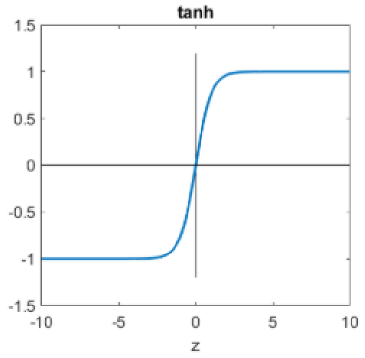
Introduction

Activation functions (Lecture 6)

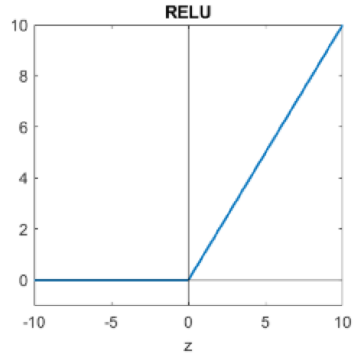
- The choice of the architecture and the activation function plays a key role in the definition of the network
- Each activation function takes a single number and performs a certain fixed mathematical operation on it



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

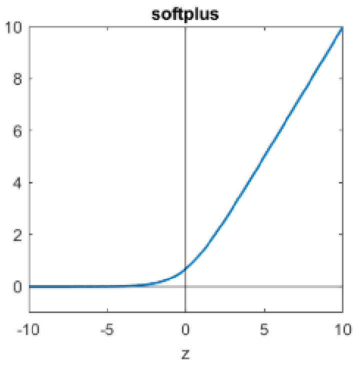


$$h(z) = \tanh z$$

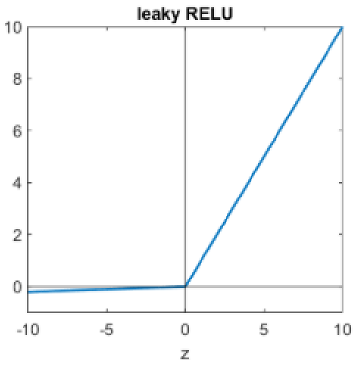


$$h(z) = \max(z, 0)$$

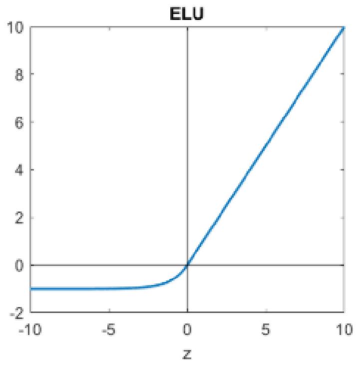
[1] Undestanding the Neural Network



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



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



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

Introduction

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

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

Introduction

Backpropagation (Lecture 6)

- 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

TOTAL NUMBER OF TRAINING SAMPLES

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - d_i)^2$$

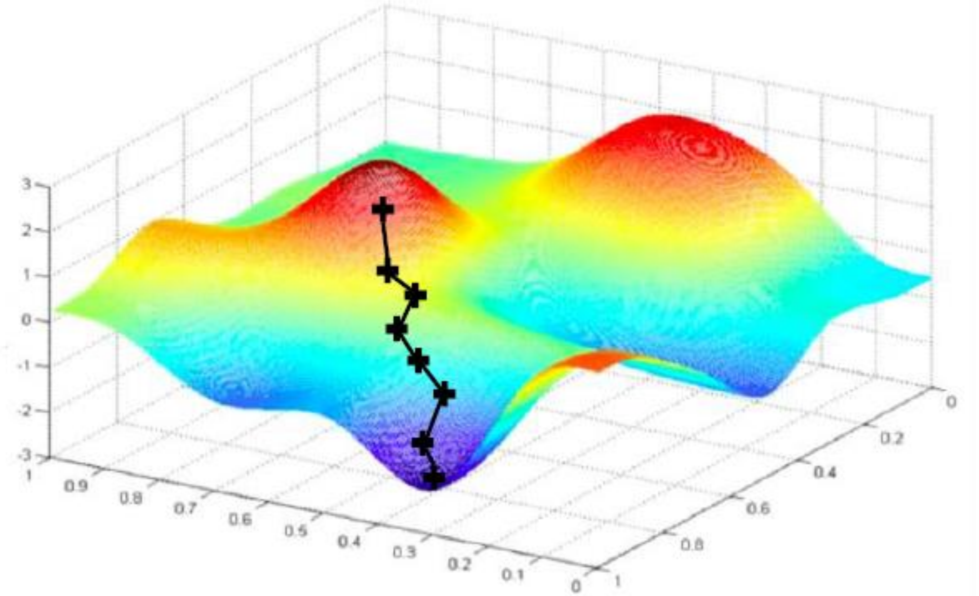
DESIRED OUTPUT (TARGET) VALUE FOR THE i-th SAMPLE

OUTPUT VALUE OBTAINED BY THE MLP FOR THE i-th SAMPLE

Mini-batch Gradient Descent (Lecture 6)

- Gives an estimate of the true gradient by averaging the gradient from each of the B points (mini-batch)
- 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 $\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{\mathbf{1}}{\boldsymbol{B}} \sum_{k=1}^{B} \frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights $\boldsymbol{W} \coloneqq \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

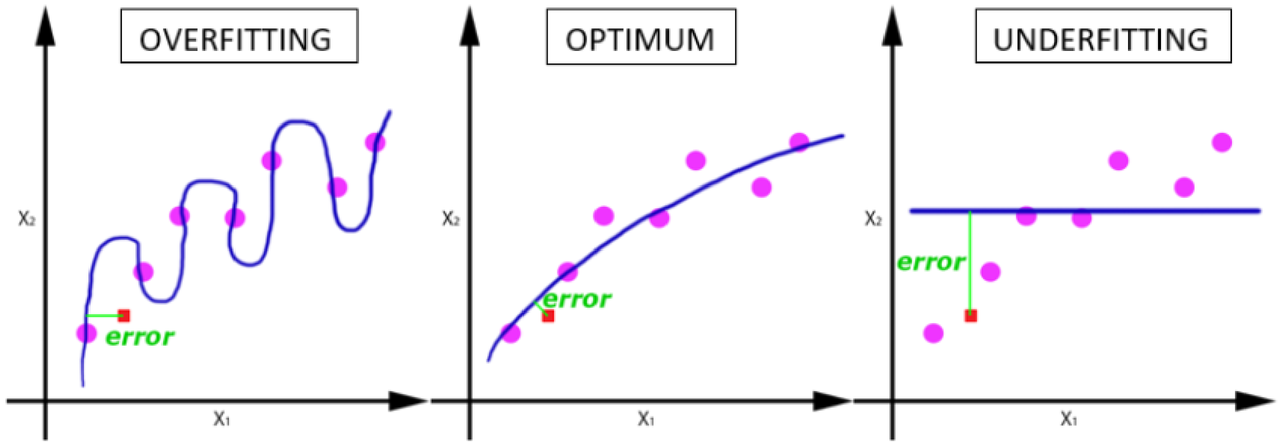


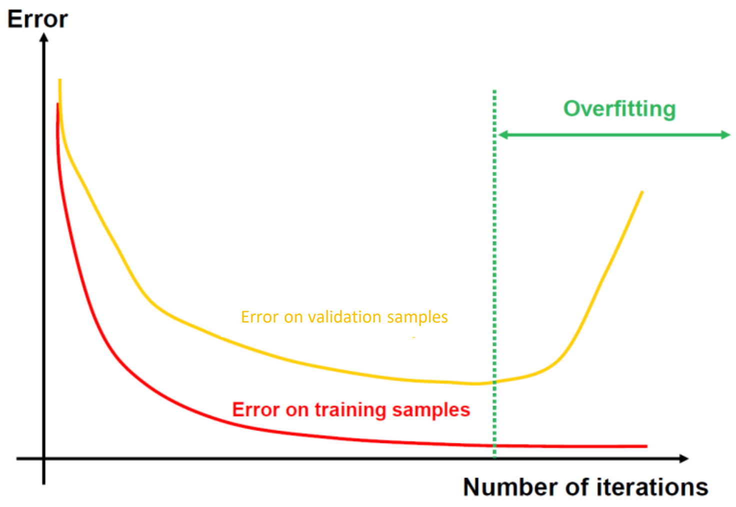
Fast to compute and a good estimate of the true gradient!

Introduction

Epochs and iterations

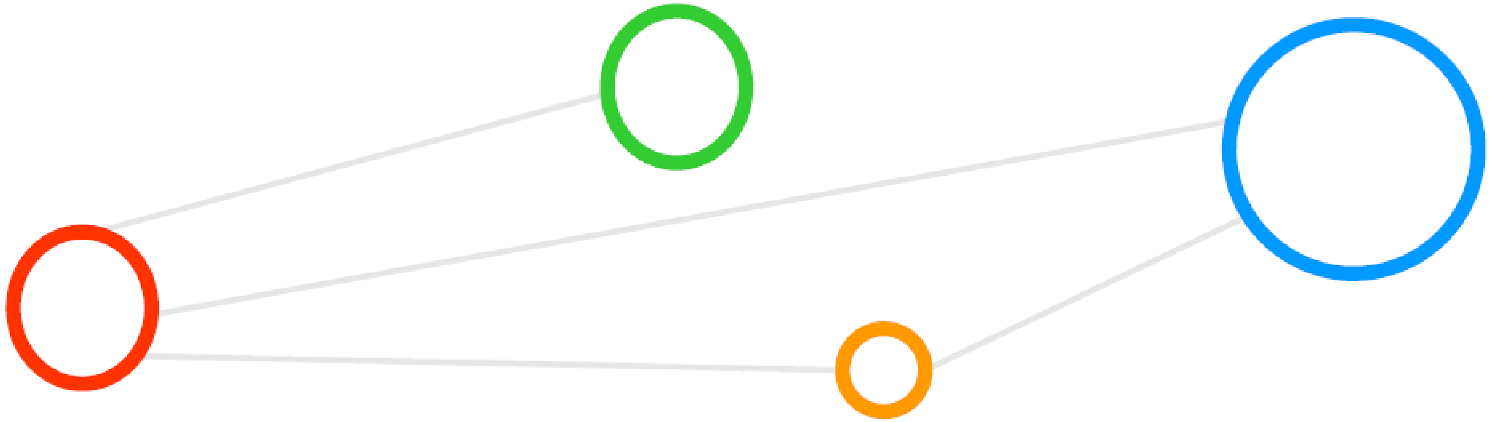
- **1 Epoch**: entire training set passed forward and backward through the network in once
  - The training set is divided in batches since the data can be too large
- **1 iteration**: entire batch passed forward and backward through the network in once
  - If 1000 training samples and batch size set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





What is the right numbers of epochs?

Distributed training: theory

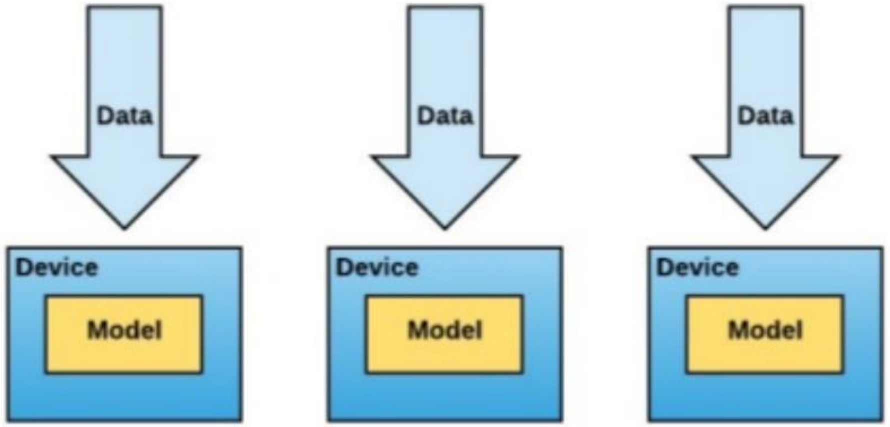


Distributed training

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

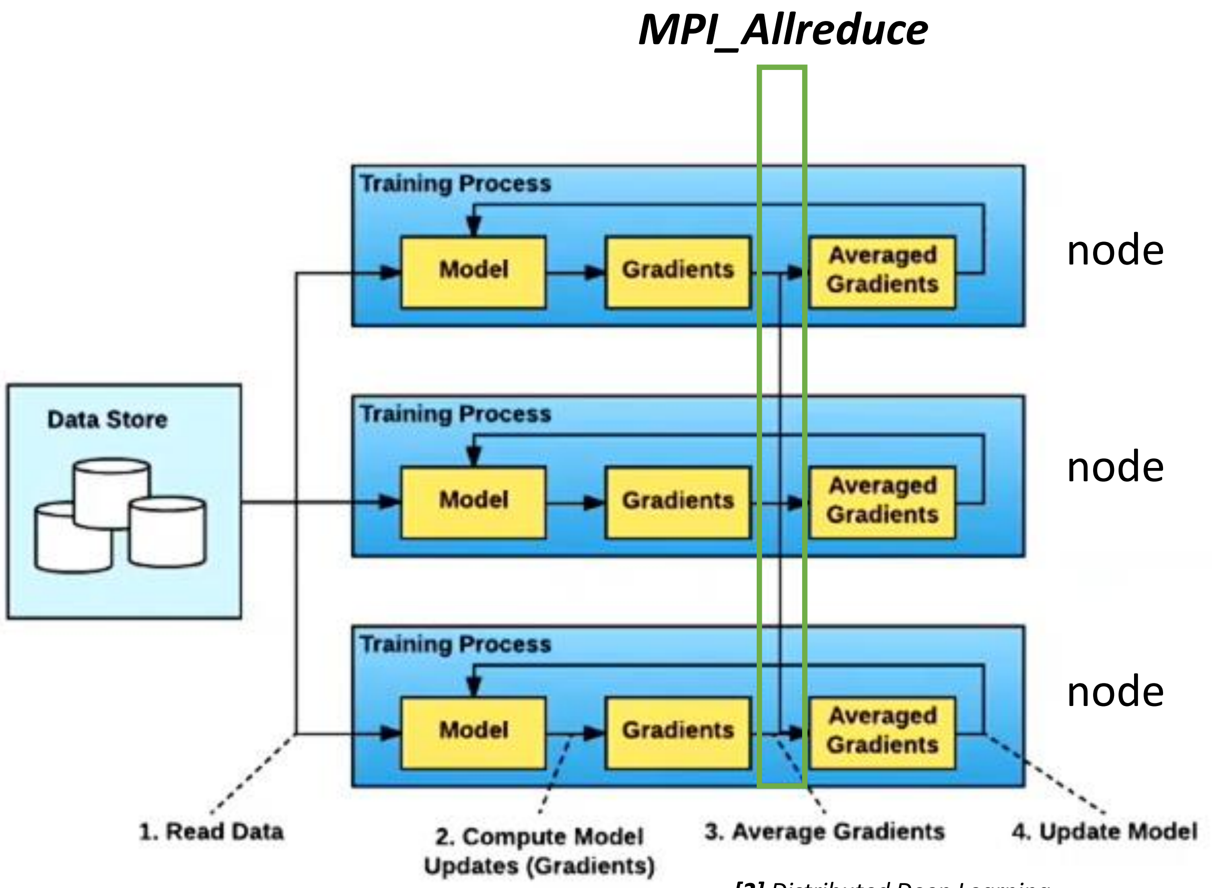


***[2]** Distributed Deep Learning*

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

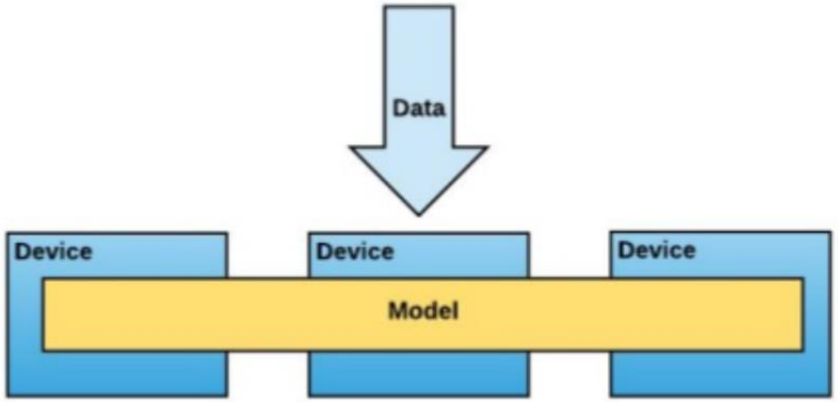


[2] Distributed Deep Learning

Distributed training

With Model Parallelism

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



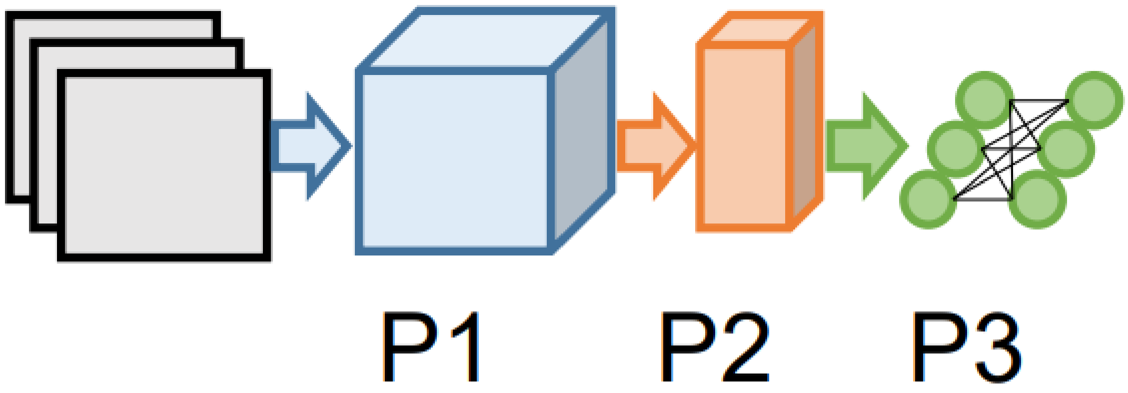
Model Parallelism

***[2]** Distributed Deep Learning*

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

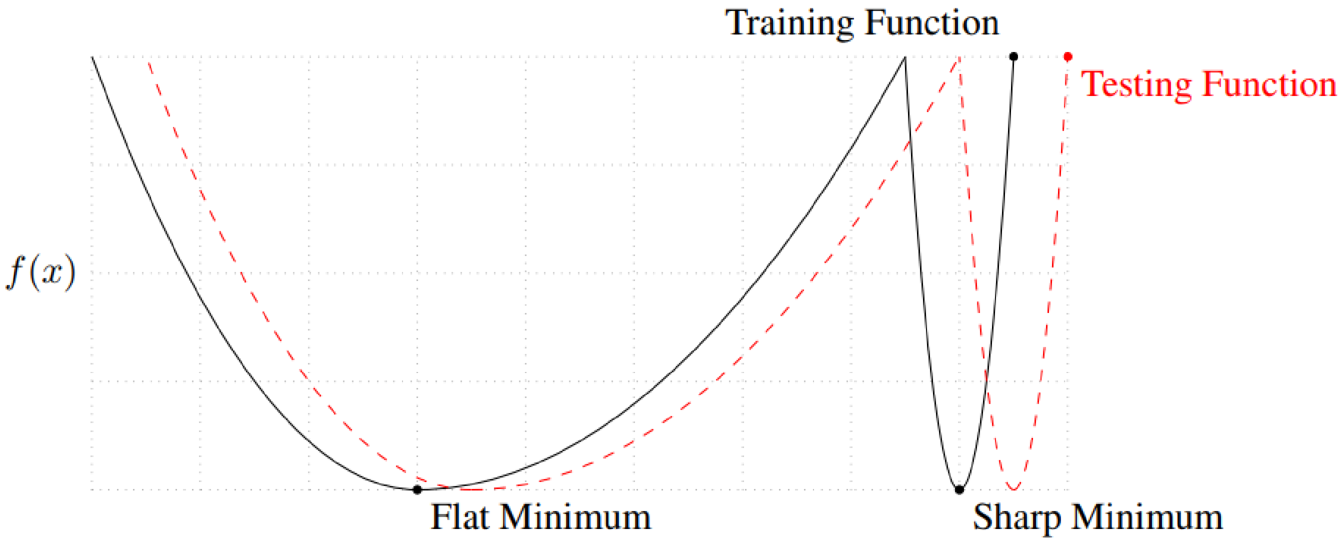


***[3]** Distributed Deep Learning*

Challenges of Distributed Learning

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

- Very large mini-batch size (> 8000 samples) 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]** Large-Batch Training*

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: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

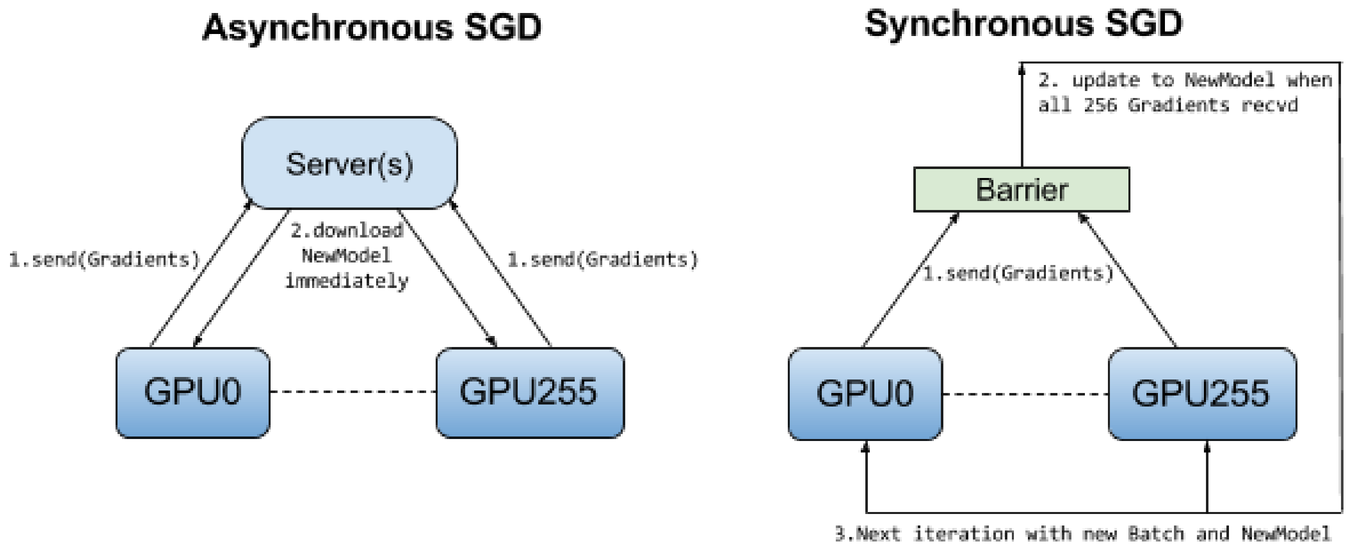
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

***[6]** Batch normalization*

Optimization

- SYNCHRONOUS SGD
  - Stragglers, machines which take a long time to respond [7]
  - 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

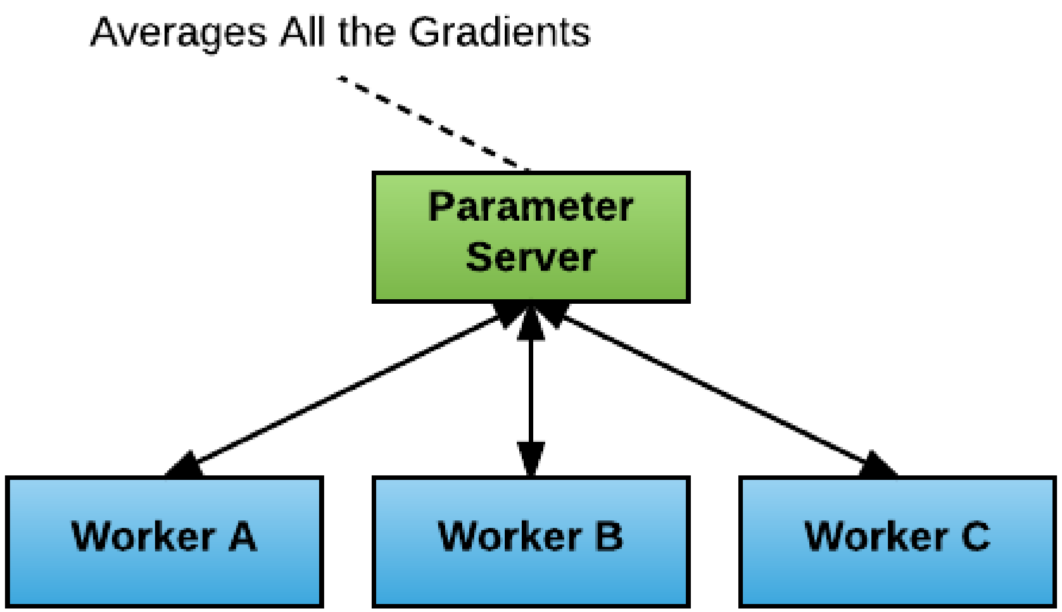


[8] Sync and async SGD

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



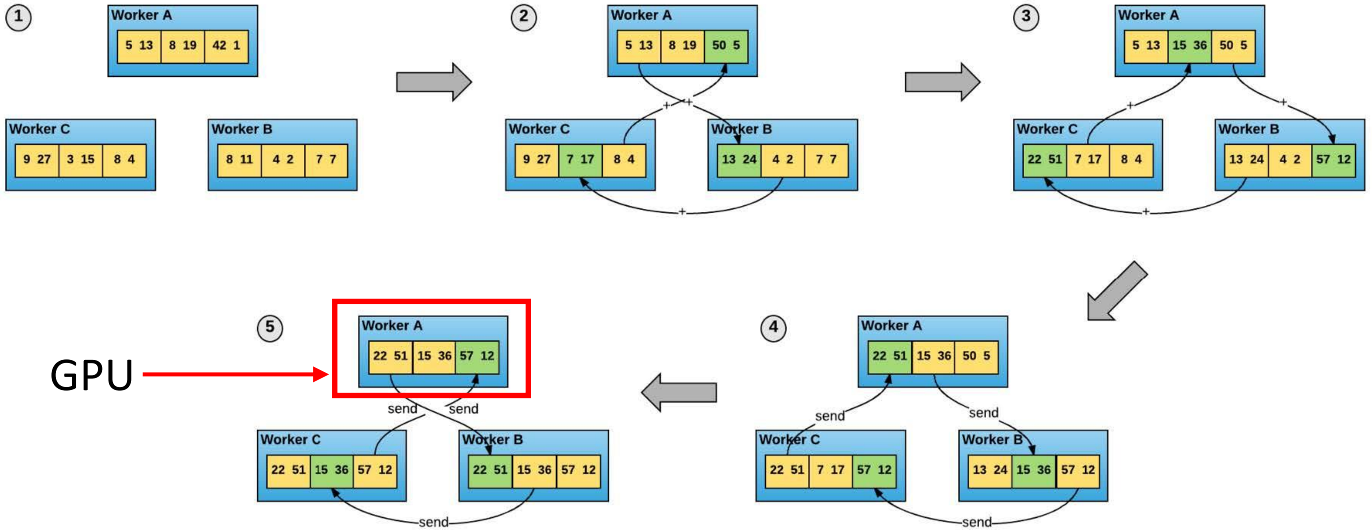
***[9]** Parameter server*

Global gradient update

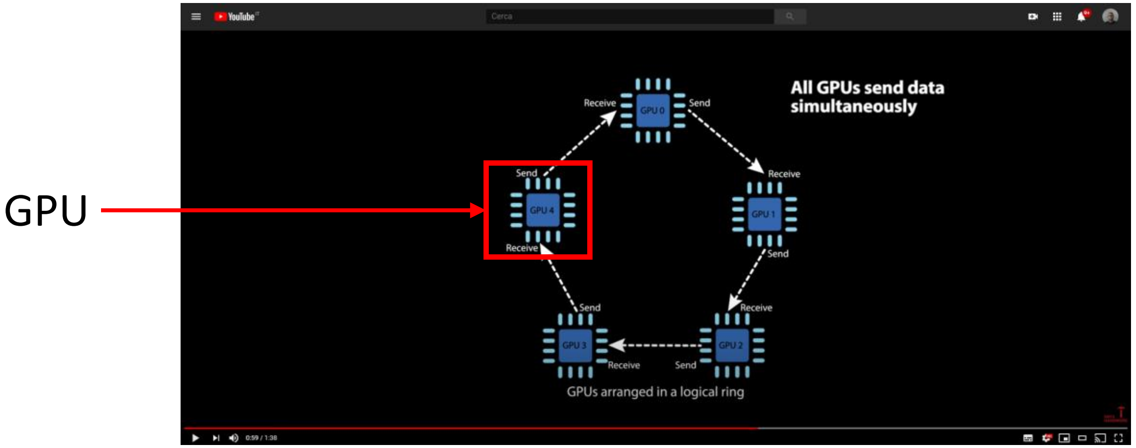
Ring allreduce

- 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] Distributed Deep Learning



[10] Ring AllReduce

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 [11]

- 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:

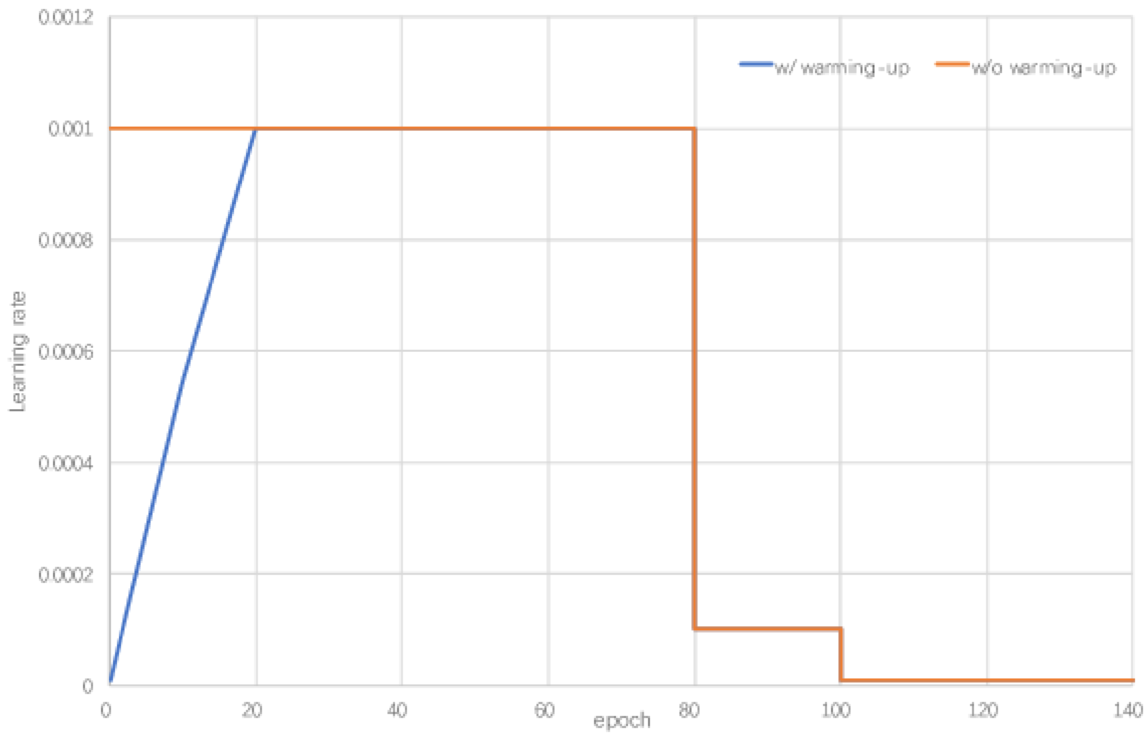
$$\mathrm{cov}\left(\Delta\mathbf{w}, \Delta\mathbf{w}\right) \approx \frac{\eta^2}{M}\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{g}_n\mathbf{g}_n^\top\right)$$

- 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



***[12]** Warm-up*

LARS optimizer

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

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

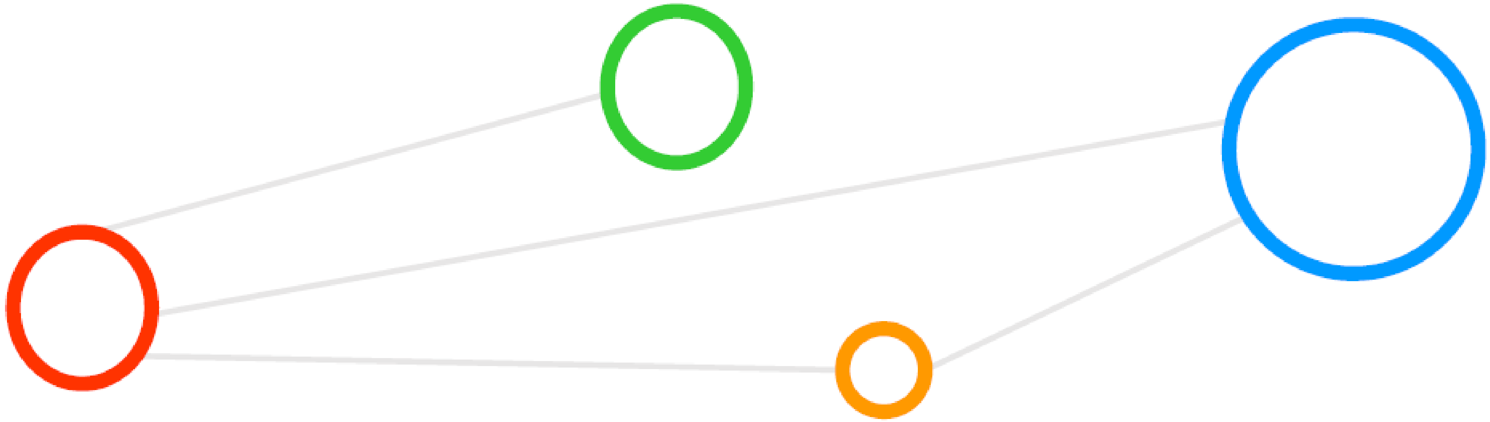
[13]

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

Tensor fusion

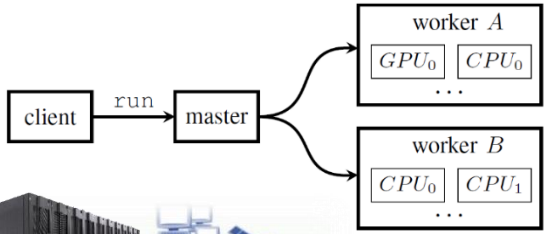
- An efficient communication strategy in a distributed training system should maximize the throughput as well as reduce the latency [14]
- Sizes of gradient tensors to aggregate vary a lot for different types of layers
- Usually, gradient tensor sizes for convolution layers are much smaller than fully-connected 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

Distributed training: frameworks



Popular Deep Learning Frameworks used with Python in Cloud Computing

- TensorFlow
  - One of the most popular deep learning frameworks available today
  - Execution on multi-core CPUs or many-core GPUs



[3] Tensorflow Web page

- **Tensorflow is an open source library for deep learning models using a flow graph approach**
- **Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)**
- **The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)**
- **Tensorflow work with the high-level deep learning tool Keras in order to create models fast**
- **New versions of Tensorflow have Keras shipped with it as well & many further tools**

[4] Keras Web page

- Keras
  - Often used in combination with low-level frameworks like Tensorflow

- **Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano**
- **Created deep learning models with Keras run seamlessly on CPU and GPU via low-level deep learning frameworks**
- **The key idea behind the Keras tool is to enable faster experimentation with deep networks**

Distributed Deep Learning Frameworks

Horovod

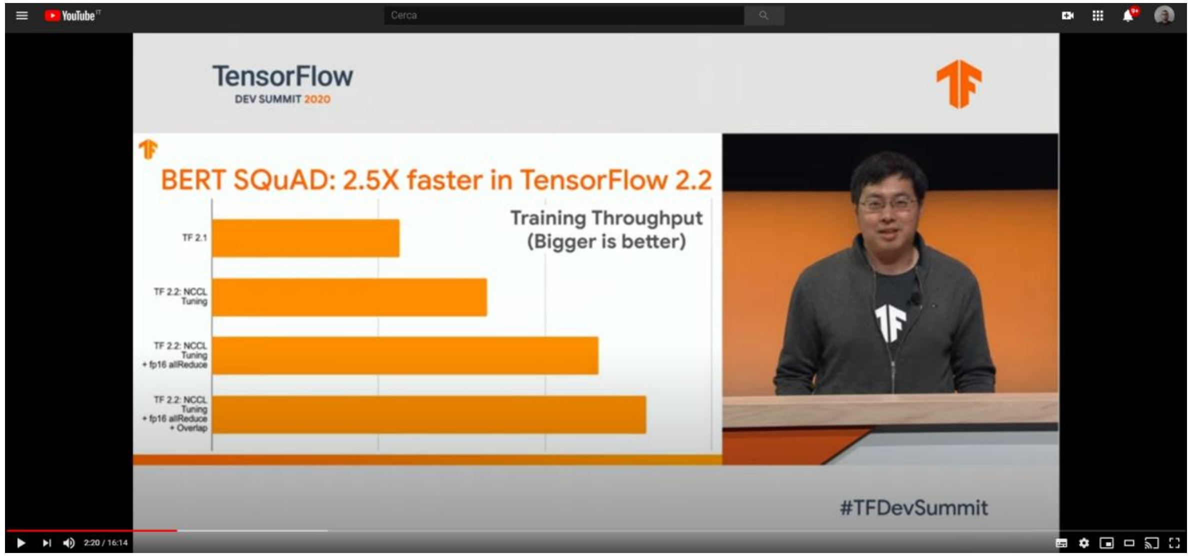
Horovod
- **Data parallel**, each GPU has a copy of the model and a chunk of the data
- Efficient **decentralized framework**,

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

[3] Distributed Deep Learning

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

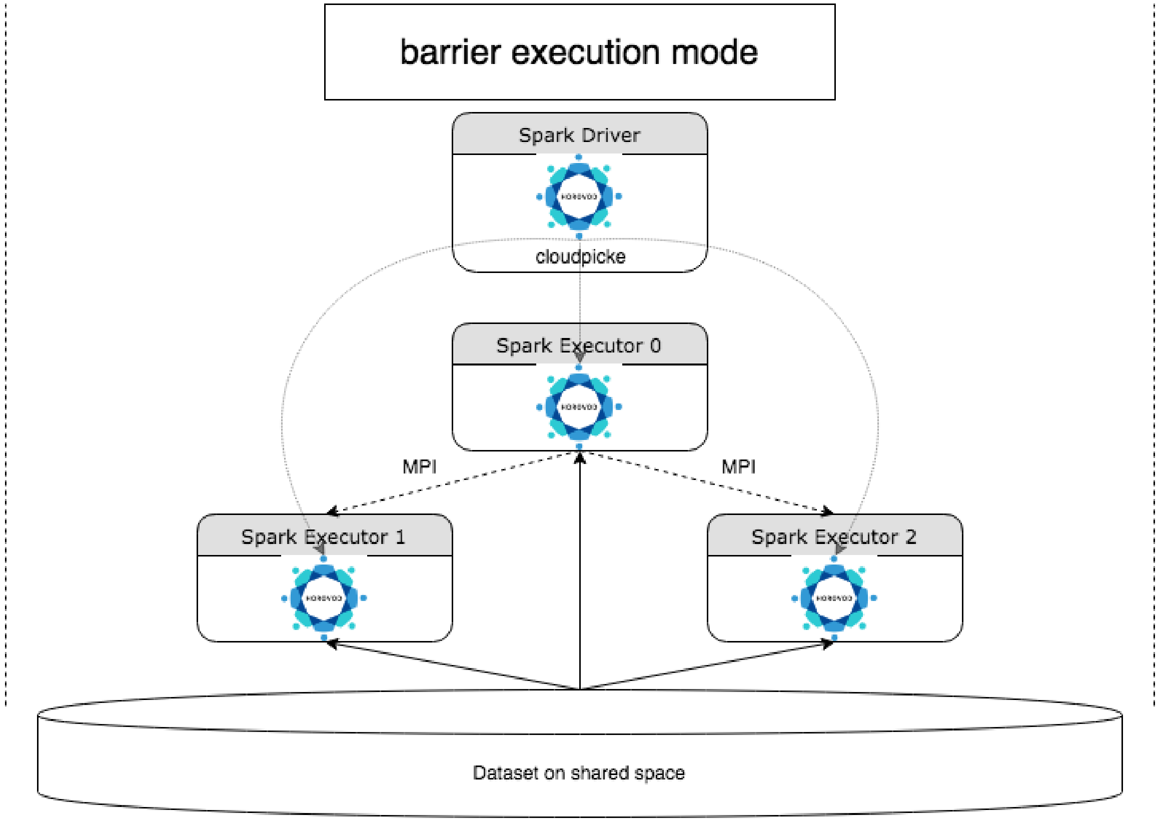


[15] Distributed TF2

Horovod

Horovod in the Cloud

- HorovodRunner is a general API to run distributed deep learning workloads on Azure Databricks using Uber's Horovod framework
- Spark's barrier mode
- Run a Horovod training job invoking main(**kwargs), both the main function and the keyword arguments are serialized using cloudpickle and distributed to cluster workers [16]



***[16]** Horovod on Spark*

Horovod

Development workflow

1. **Prepare single node DL code:** Prepare and test the single node DL code with TensorFlow, Keras, or PyTorch
2. **Migrate to Horovod:** Follow the instructions adding «7 lines of code»
3. **Migrate to HorovodRunner:** HorovodRunner runs the Horovod training job by invoking a Python function, you must wrap the main training procedure into one function
4. **Deployment:** Test HorovodRunner in local mode and distributed mode [16]

```
Python

hr = HorovodRunner(np=2)

def train():
  import tensorflow as tf
  hvd.init()

hr.run(train)
```

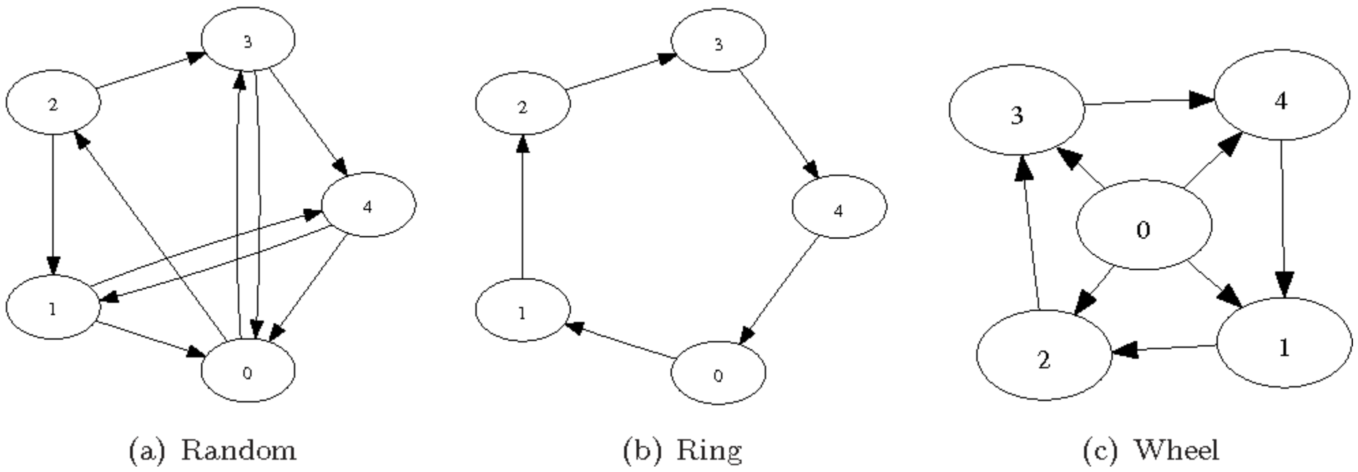
[17] HorovodRunner

```python
from sparkdl import HorovodRunner

hr = HorovodRunner(np=2)
hr.run(train_hvd, learning_rate=0.1)
```

What is MPI?

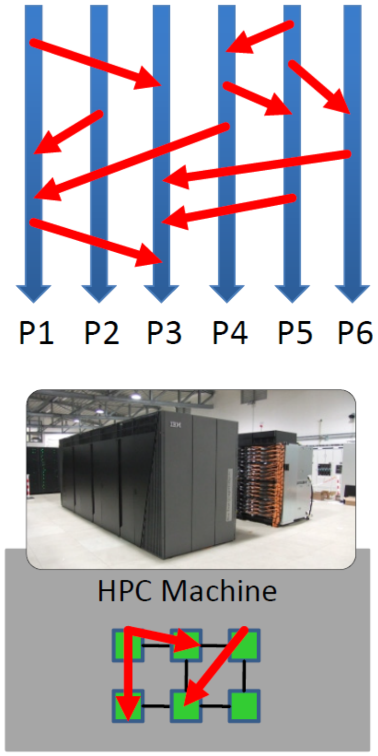
- MPI is a **standard** for **exchanging messages** between multiple computers running a parallel program across **distributed memory**
- Point-to-point and collective communication are supported
- **Different topologies** can be implemented
- Parallel I/O operations
- Blocking and non blocking statements

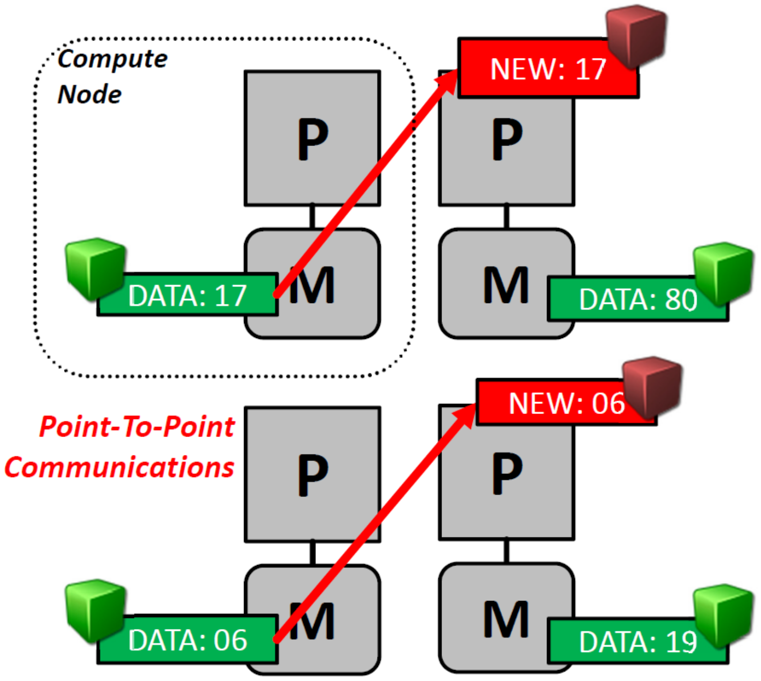


[18] MPI topologies

Message Passing: Exchanging Data

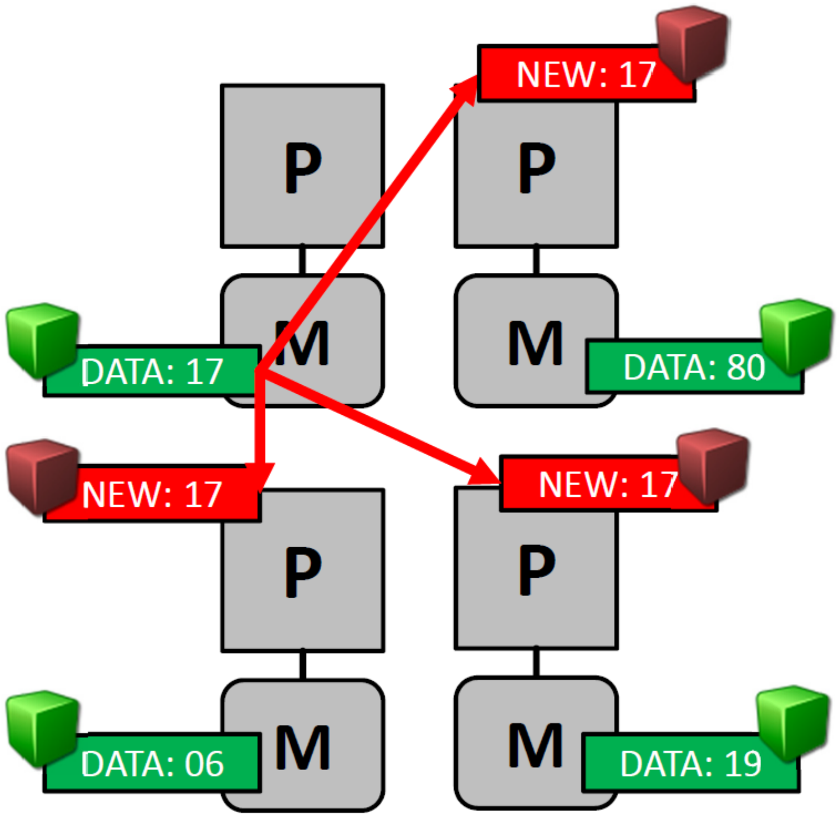
- Each processor has its own data and memory that cannot be accessed by other processors





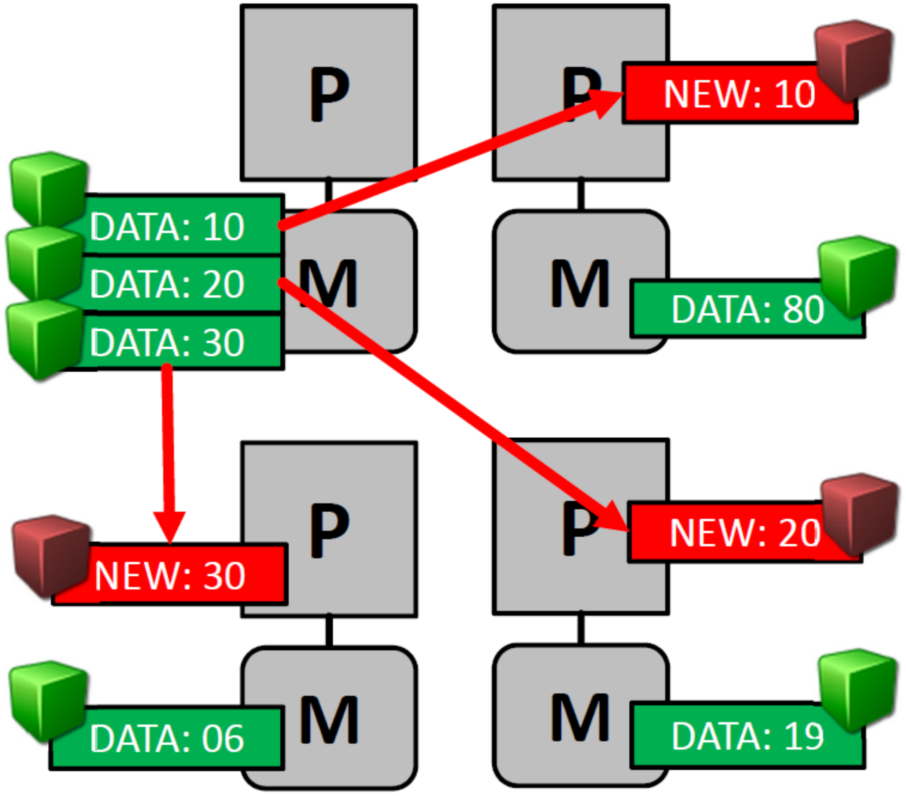
Collective Functions: Broadcast (one-to-many)

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



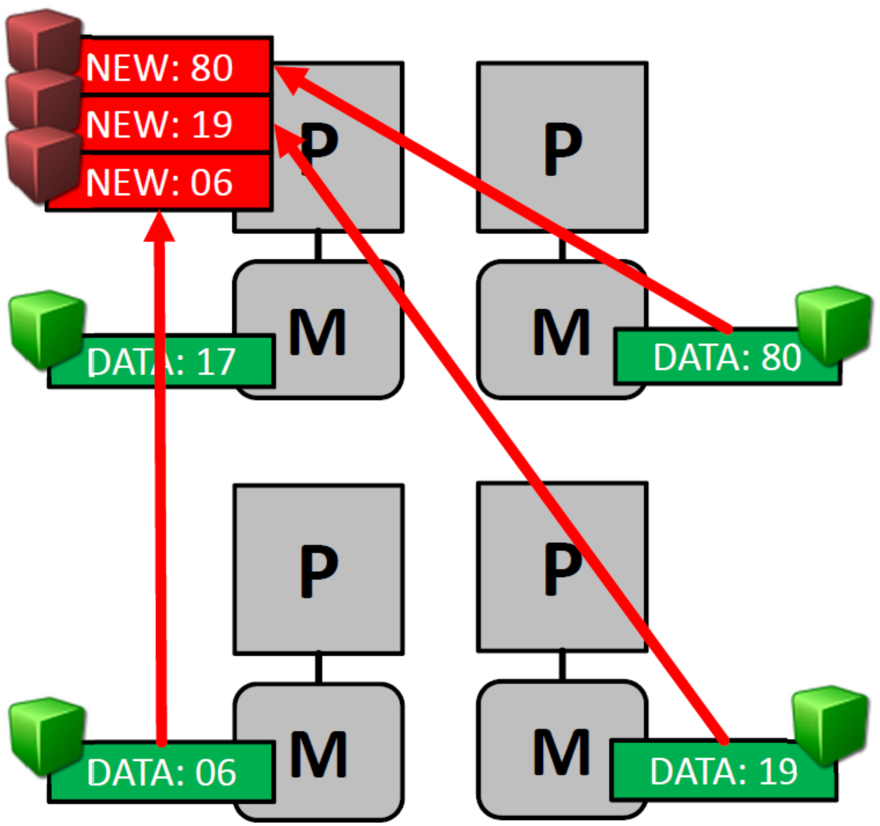
Collective Functions: Scatter (one-to-many)

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



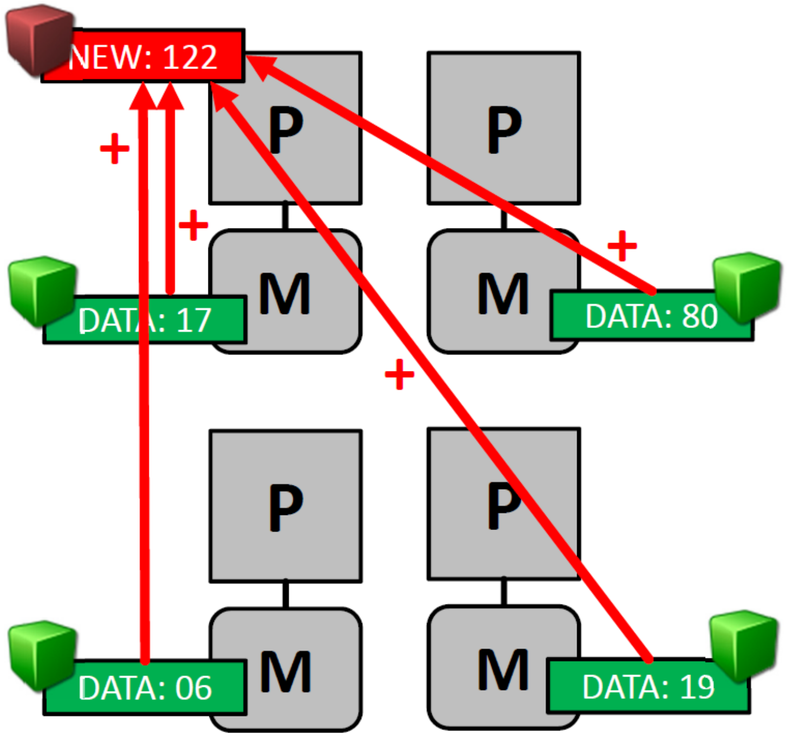
Collective Functions: Gather (many-to-one)

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



Collective Functions: Reduce (many-to-one)

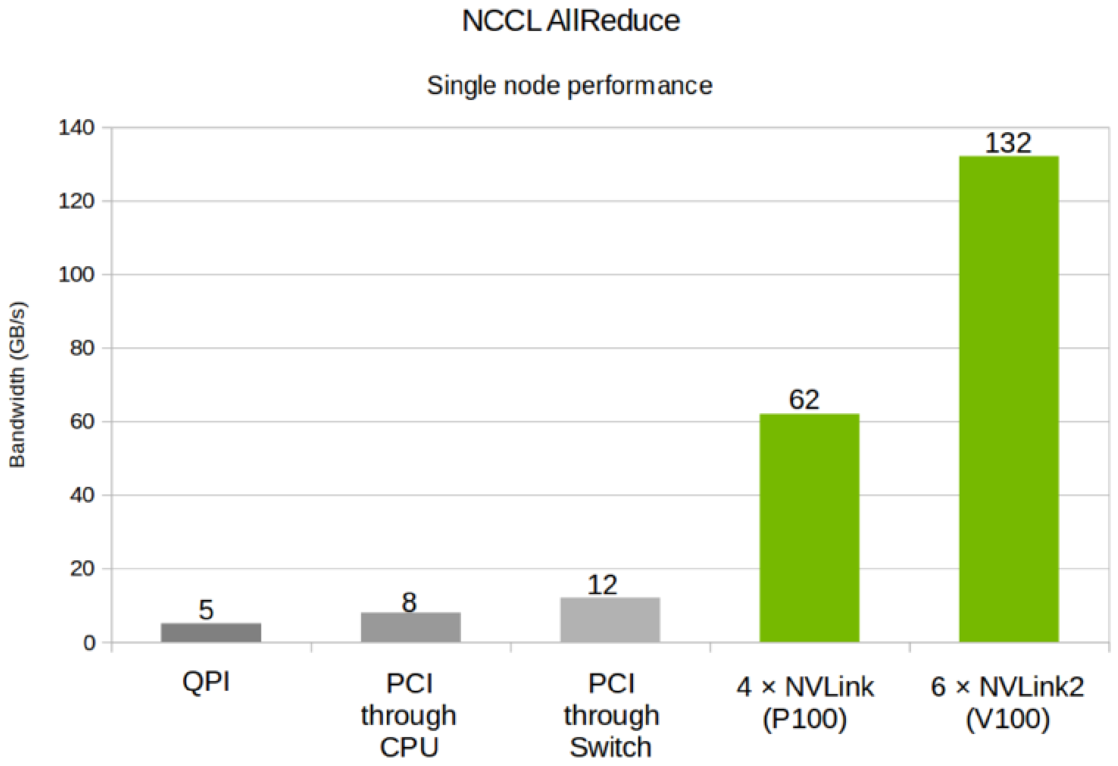
- Each Reduce combines collection with computation based on data from many or even all other processors
- Usage of reduce includes finding a **global minimum or maximum, sum, or product** of the different data located at different processors

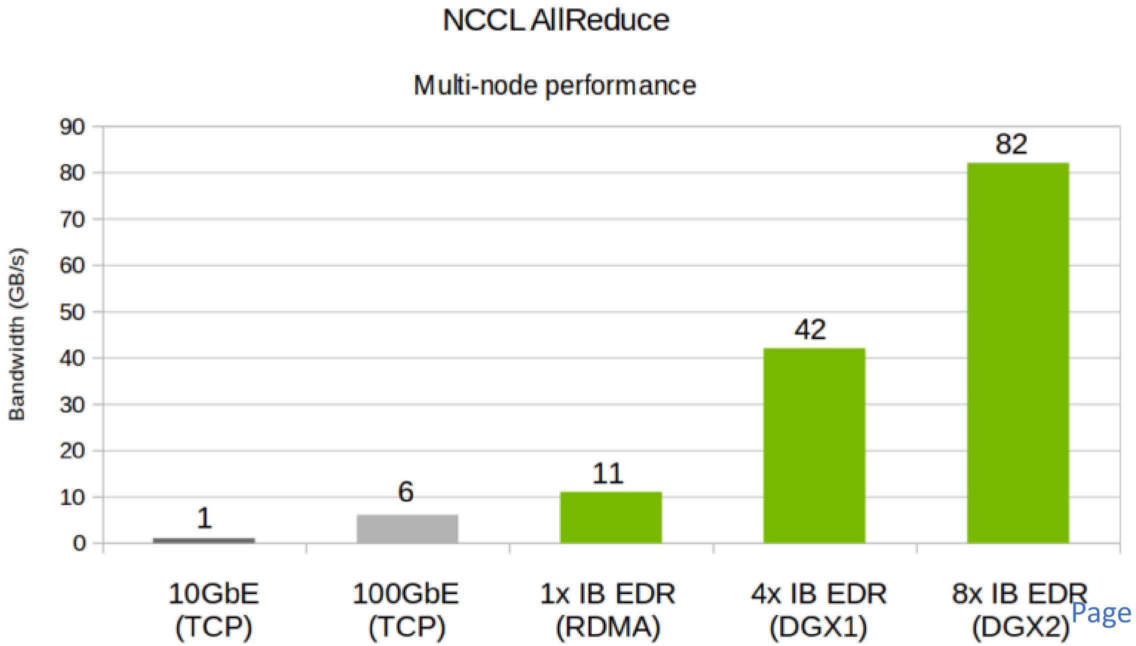


+ global sum as example

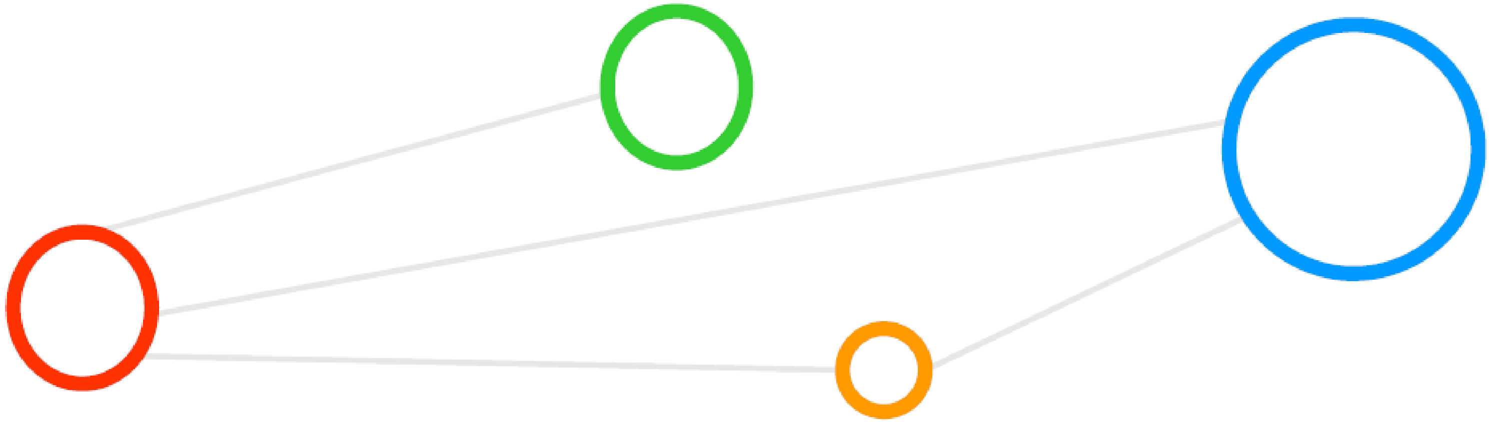
NCCL

- NVIDIA Collective Communications Library (NCCL) [19]
- 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



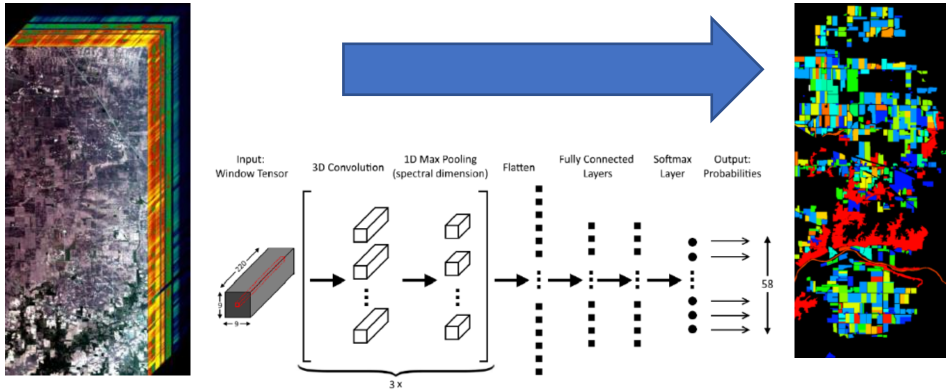


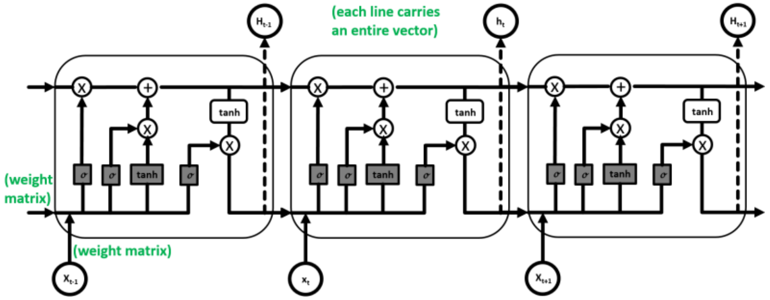
Distributed training: an application



Deep Learning Applications in Clouds

- Learning Image Data Sets
  - Convolutional Neural Networks (CNNs) models in Clouds
  - Several datasets in examples used in cloud environments
  - E.g. hand-written character recognition eamples
  - E.g. remote sensing data & earth observation data sets
- Learning Sequence Data Sets
  - Recurrent Neural Networks (RNNs)
  - Examples of Long Short-Term Memory (LSTM) models in Clouds
  - Several datasets in examples used in cloud environments
  - E.g. Natural Language Processing (NLP) applications & text processing
- Deep Learning requiring massive Computing Power
  - Pretrained Deep Learning Networks & Transfer Learning
  - Neural Architecture Search (NAS)



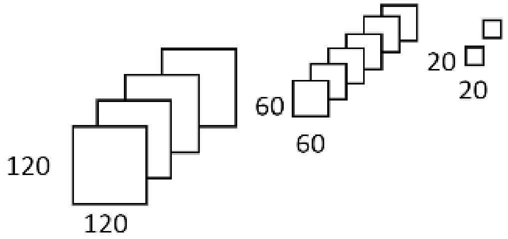


Remote Sensing Application

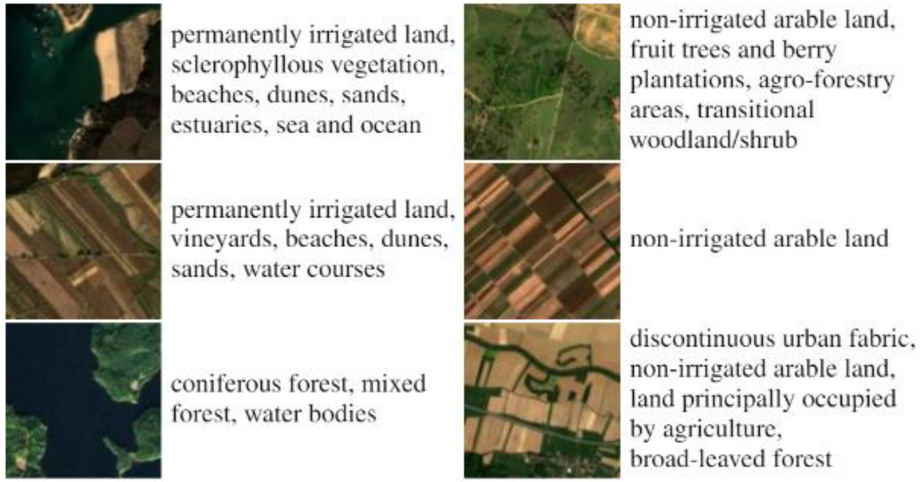
Multi land-cover class patch-based classification

Dataset: Sentinel-2 Data Patches and Annotated with CORINE Land Covers

Datasets	Image type	Image per class	Scene classes	Annotation type	Total images	Spatial resolution (m)	Image sizes	Year	Ref.
BigEarthNet	Satellite MS	328 to 217119	43	Multi label	590,326	10 20 60	120x120 60x60 20x20	2018	[20]



Patch and its dimension



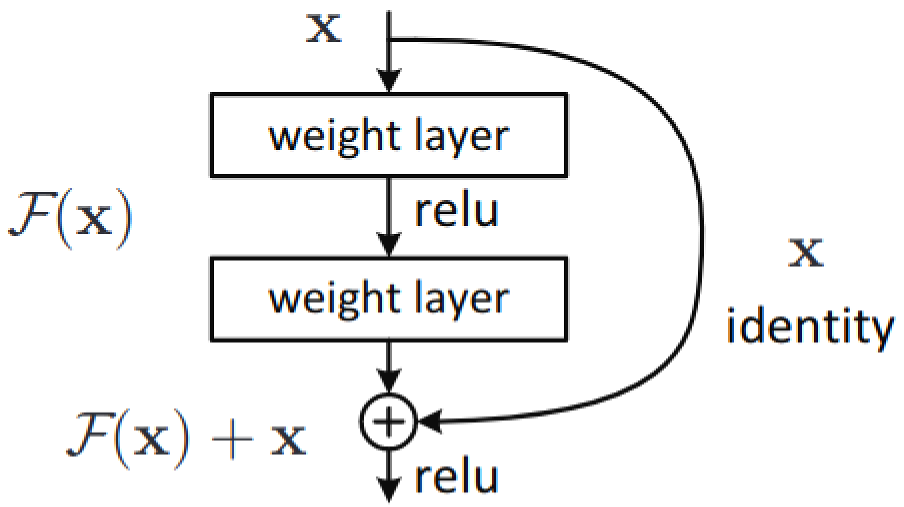
Setup

HPC

- The experiments carried out on Juelich Research on Exascale Cluster Architectures (JURECA) supercomputer,
- Experiments using 32 nodes, i.e., 128 GPUs (NVIDIA K80 GPUs
- with 24GB of memory each).

ResNet50

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



***[21]** ResNet-50*

Challenge

- Keep high accuracy with large batch size is a known issue
- LARS optimizer with Nesterov momentum
- The initial LR is computed using a linear policy as $\eta = (0.1*k*n)/256$, where k is the number of workers (i.e., GPUs) and n is the batch size for each worker (set here to 64 for the 8,000 effective batch size case, to 128 for the 16,000 case and to 256 for the 32,000 case)
- Scheduler with deterministic annealing
- LR computed following a multi-step decay scheme. The original LR was multiplied by 0.1 after 30 epochs, by 0.01 after 60 epochs and by 0.001 after 80 epochs
- To avoid instability problems a warm-up of 5 epochs was used for all the experiments [22]

Results

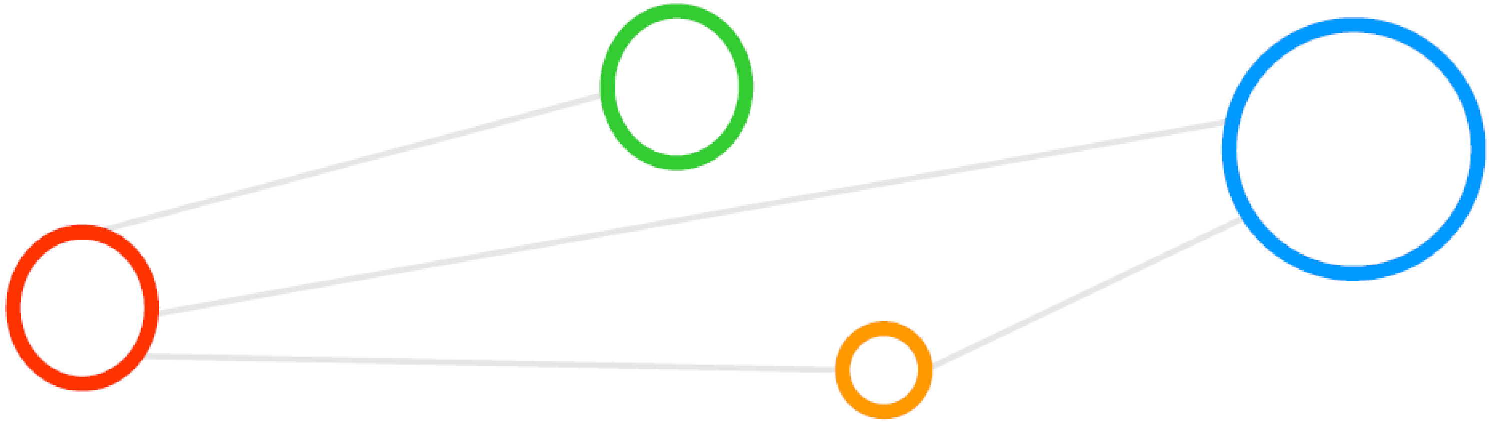
- Accuracy stable up to batch size = 8k
- For batch size > 8k training diverges
- Horovod enabled to significantly cut training time
- Scaling slightly less than linear (possibly due to data loading issues)

batch size	n. GPUs	warm-up	initial LR	F1
512	8	5	0.2	0.78
8,000	128	5	3.2	0.74
16,000	128	5	6.4	0.64 (diverge)
32,000	128	5	12.8	0.43 (diverge)

***[22]** Distributed ResNet-50 for Remote Sensing application*

batch size	n. GPUs	training time [s]
512	8	49,400
8,000	128	3,400
16,000	128	2,800
32,000	128	2,500

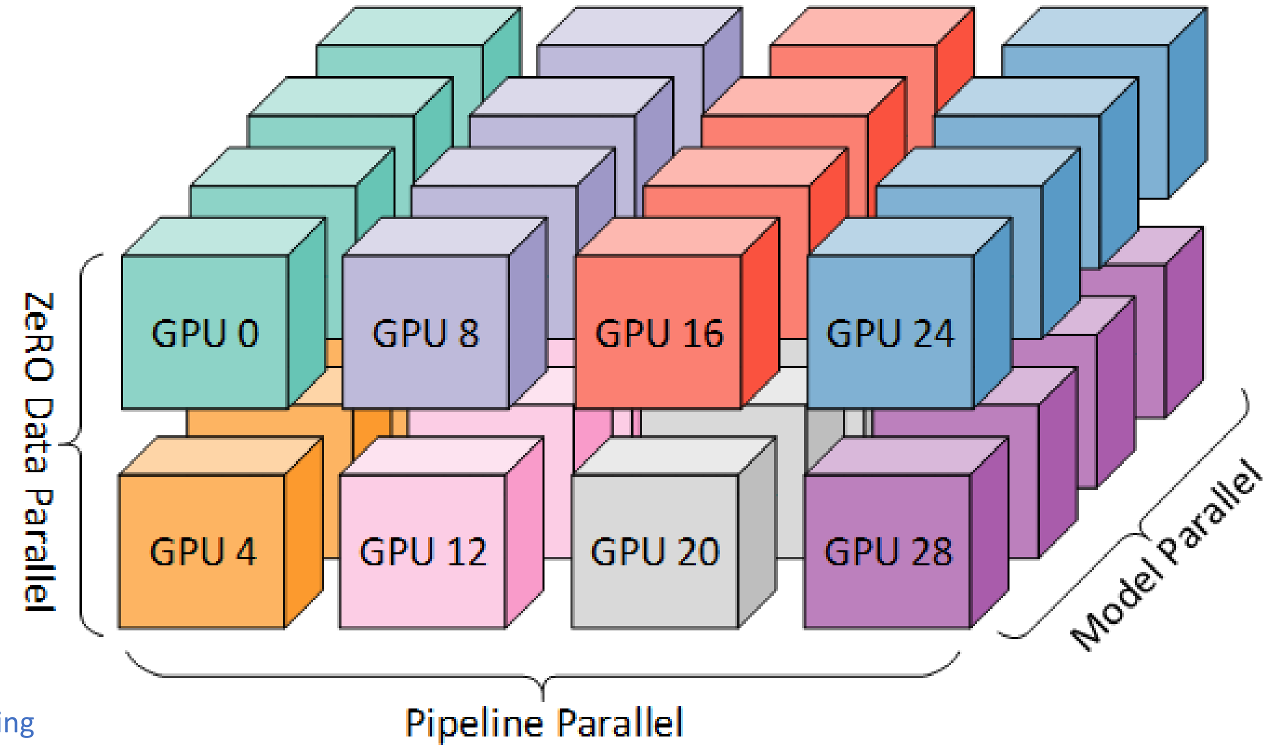
DeepSpeed: a new 3D parallelism framework



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
- DeepSpeed API is a lightweight wrapper on PyTorch



[23] DeepSpeed

DeepSpeed

ZeRO

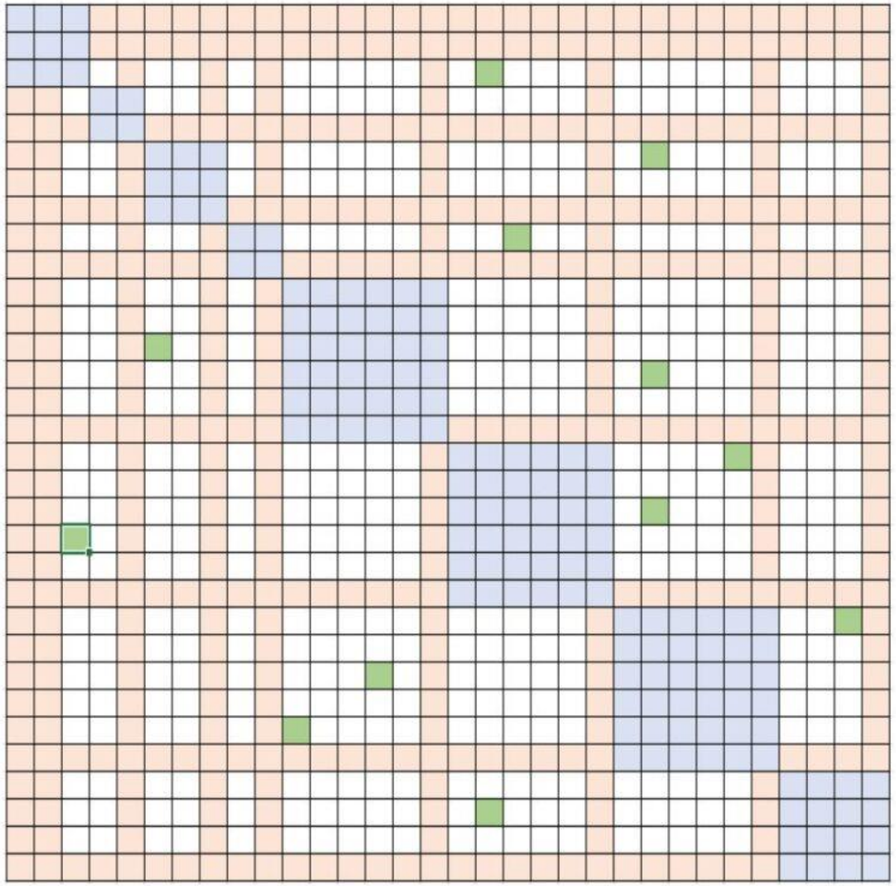
- **10x bigger model training on a single GPU with ZeRO-Offload,** leveraging both CPU and GPU memory for training large models
- Models **of up to 13 billion parameters** on **a single NVIDIA V100 GPU** without running out of memory, 10x bigger than the existing approaches
- ZeRO removes the memory 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

[23]

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 [24]

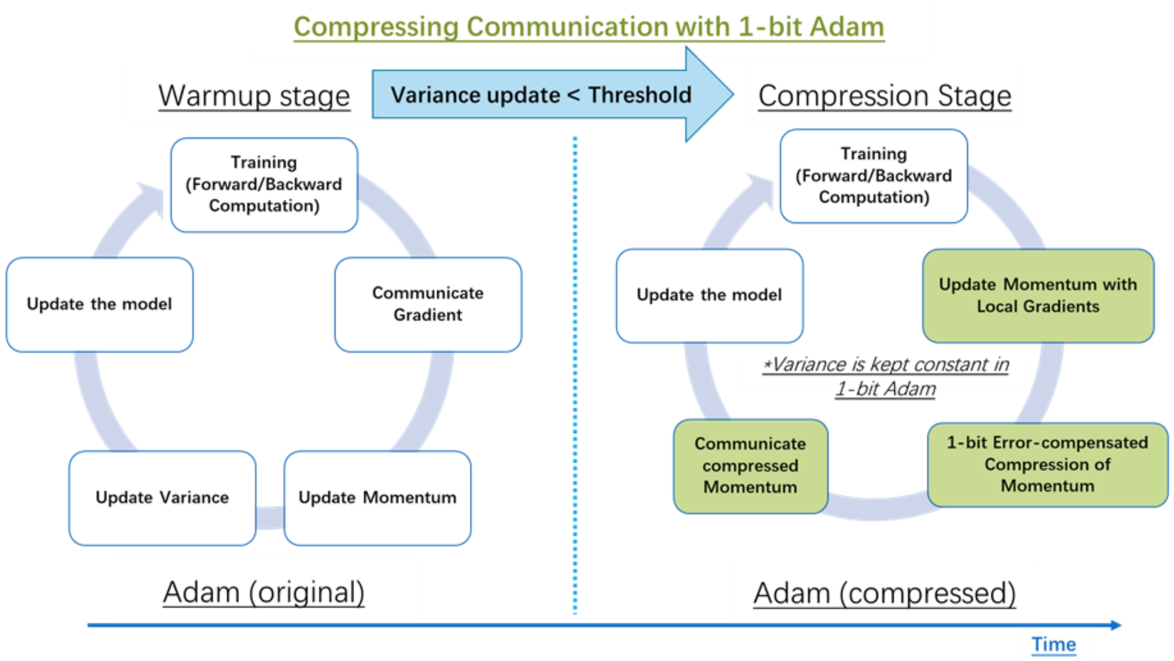


[23] DeepSpeed

DeepSpeed

1-bit Adam

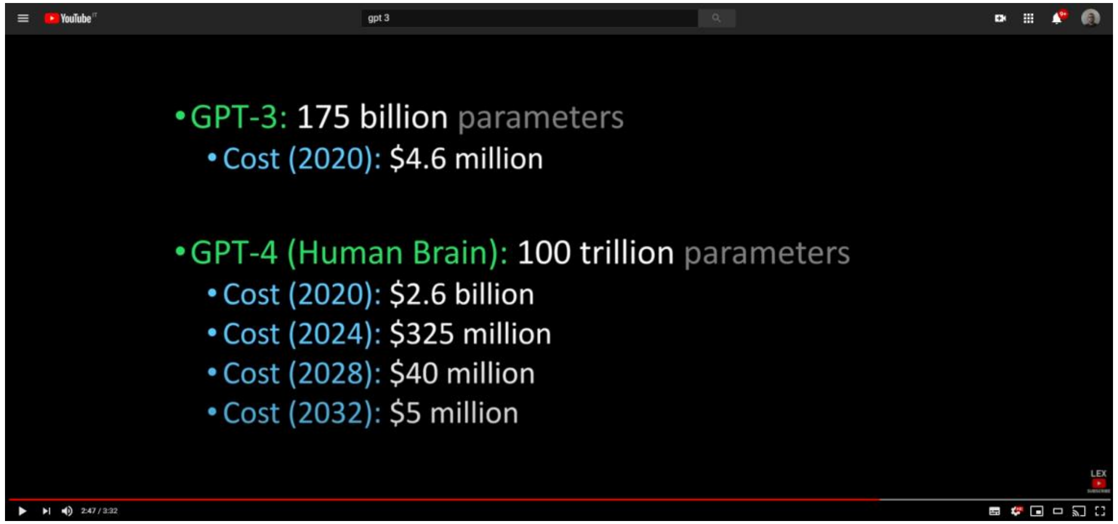
- **1-bit Adam with up to 5x communication volume reduction**
- consists of two parts:
  1. the **warmup** stage, which is the vanilla Adam algorithm
  2. the **compression** stage, which keeps the variance term constant and compresses the remaining linear term, that is the momentum, into 1-bit representation



[23] DeepSpeed

GPT-3 Transformer

- How much would it cost to train in on a Cloud service?
- Let's have a look at **NCsv3-series** [25]
- **355 years** to train GPT-3 on a Tesla V100
- Training cost = 355Y×365D/Y×24H/D×0.9792$/H = **3.045.116$**

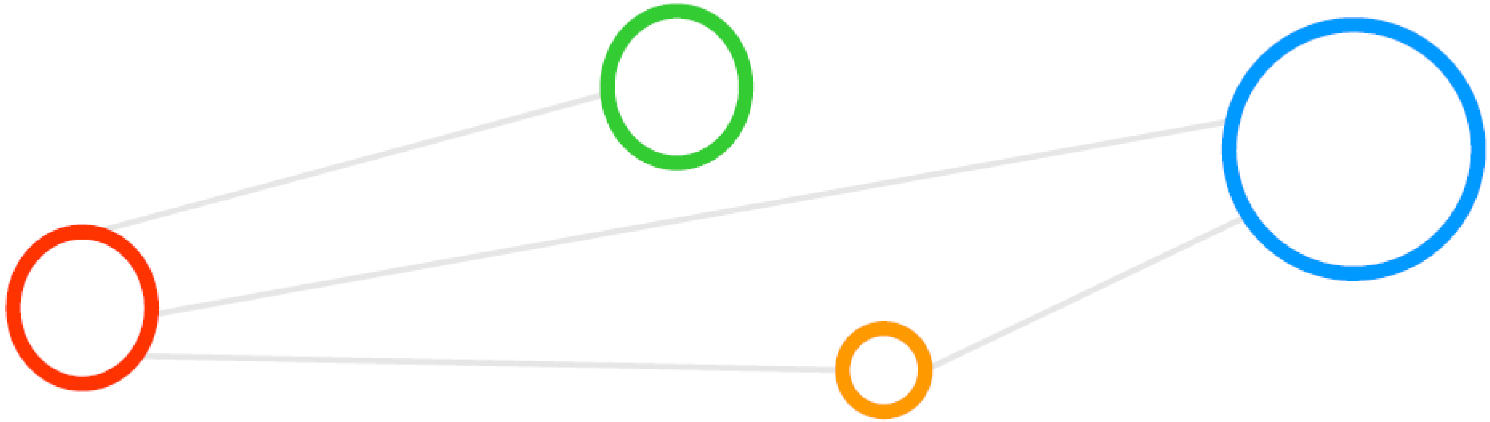


***[27]** Lex Friedman on GPT-3*

Add to estimate	Instance	Core	RAM	Temporary storage	GPU	Pay as you go	1 year reserved (% Savings)	3 year reserved (% Savings)	Spot (% Savings)
⊕	NC6s v3	6	112 GiB	736 GiB	1X V100	$3.06/hour	$1.9492/hour (~36%)	$0.9792/hour (~68%)	$0.306/hour (~90%)
⊕	NC12s v3	12	224 GiB	1,474 GiB	2X V100	$6.12/hour	$3.8984/hour (~36%)	$1.9585/hour (~68%)	$0.612/hour (~90%)
⊕	NC24rs v3	24	448 GiB	2,948 GiB	4X V100	$13.464/hour	$8.5766/hour (~36%)	$5.1002/hour (~62%)	$1.3464/hour (~90%)
⊕	NC24s v3	24	448 GiB	2,948 GiB	4X V100	$12.24/hour	$7.7970/hour (~36%)	$3.9169/hour (~68%)	$1.224/hour (~90%)

***[26]** NCsv3-series pricing*

Lecture Bibliography



Lecture Bibliography (1)

[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/

[10] https://www.youtube.com/watch?v=k8sZXYDRw6A&ab_channel=Tom%27sHardware

Lecture Bibliography (2)

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

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

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

[14] 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

[15] https://www.youtube.com/watch?v=6ovfZW8pepo&ab_channel=TensorFlow

[16] https://docs.microsoft.com/en-us/azure/databricks/applications/machine-learning/train-model/distributed-training/horovod-runner

[17] https://docs.microsoft.com/en-us/azure/databricks/_static/notebooks/deep-learning/mnist-tensorflow-keras.html

[18] 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

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

[20] G. Sumbul, M. Charfuelan, B. Demir, V. Markl, "BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding", IEEE International Geoscience and Remote Sensing Symposium, pp. 5901-5904, Yokohama, Japan, 2019.

Lecture Bibliography (3)

[21] 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

[22] R. Sedona, G. Cavallaro, J. Jitsev, A. Strube, M. Riedel, M. Book, "Scaling up a multispectral ResNet-50 to 128 GPUs", IGARSS 2020

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

[24] 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.

[25] *https://www.reddit.com/r/MachineLearning/comments/h0jwoz/d_gpt3_the_4600000_language_model/*

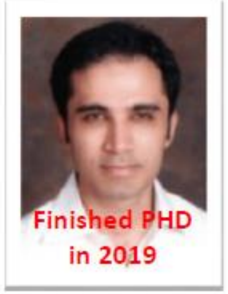
[26] https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/

[27] https://www.youtube.com/watch?v=kpiY_LemaTc&ab_channel=LexFridman

Acknowledgements – High Productivity Data Processing Research Group

PD Dr.
G. Cavallaro

Senior PhD
Student A.S. Memon



Senior PhD
Student M.S. Memon

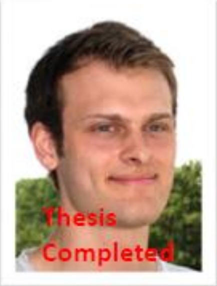
PhD Student
E. Erlingsson

PhD Student
S. Bakarat

PhD Student
R. Sedona

Dr. M. Goetz
(now KIT)

MSc M.
Richerzhagen
(now other division)



MSc
P. Glock
(now INM-1)

MSc
C. Bodenstein
(now
Soccerwatch.tv)

MSc Student
G.S. Guðmundsson
(Landsverkjun)

This research group has received funding
from the European Union's
Horizon 2020 research and
innovation programme under
grant agreement No 763558
(DEEP-EST EU Project)


Data
Science
Big
Research
science
analysis
methods
big
simulations
processing
computational
Computational
technologies
computing
European
TB
Supercomputing
Scientific
research
Climate
modelling
structures
HPC
Juelich
datasets
performance
databases
E.g
Centre
climate
Computing
nodes
centers
community
storage
ENES
EUDAT
increasing
brain
Project
DLCL
DLCLs
System
services
resources
year
Cross-Disciplinary
cross-disciplinary
forms
measurement
Policy-based
within
fusion
new
funding
concepts
device
Analysis
Computer
expertise
disciplines
results
Approaches
construction
Fusion
Enable
Modelling
Environment
Key
scientific
important
different
files
Infrastructure
basis
DKRZ
Use
etc
High
PB
Test
like
EU
model
international
SMAQ
engineering
References
formats
physical
IPCC
Common
Methods
Using
images
Work
GB
create
MapReduce
via
codes
use
JSC
e.g
acquisition
stored
Resources
analyze
Large
Summary
data-intensive
Service
Matters
Concrete
external
levels
often
Level
system
Landscape
hundreds
Nutshell
Services
network
management
compute
access
Structure
General
current
cores
large-scale
distributed
Hardware
AR5
Technologies
Earth
directory
project
Health
Processing
day
Simulation
systems
Energy
Understanding
Germany
NASA
Provide