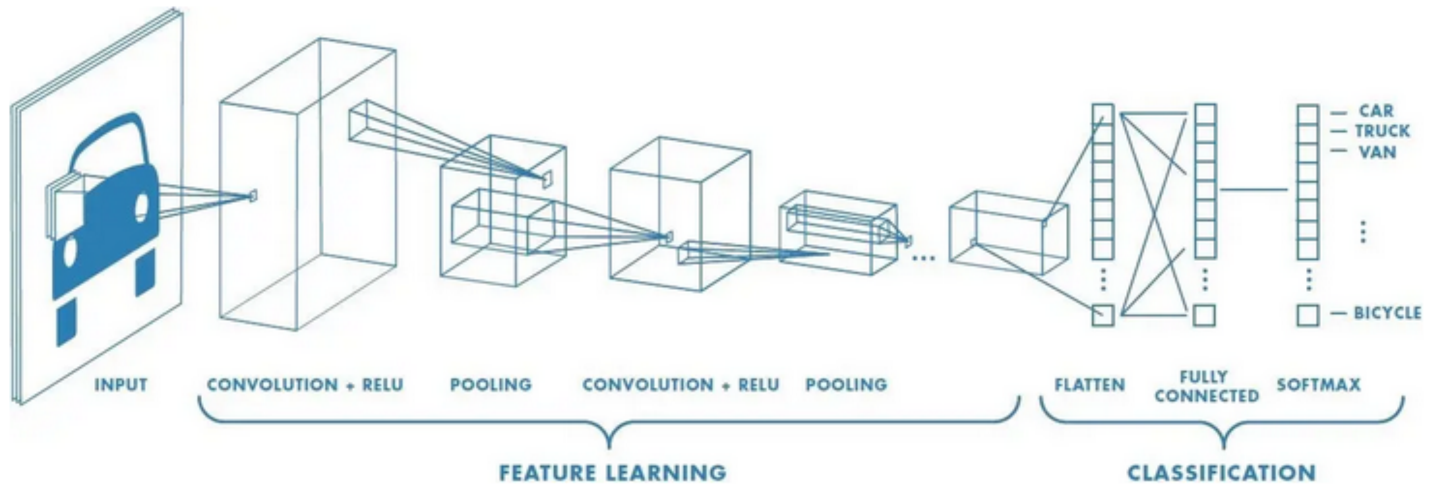
Introduction to Distributed Deep Learning

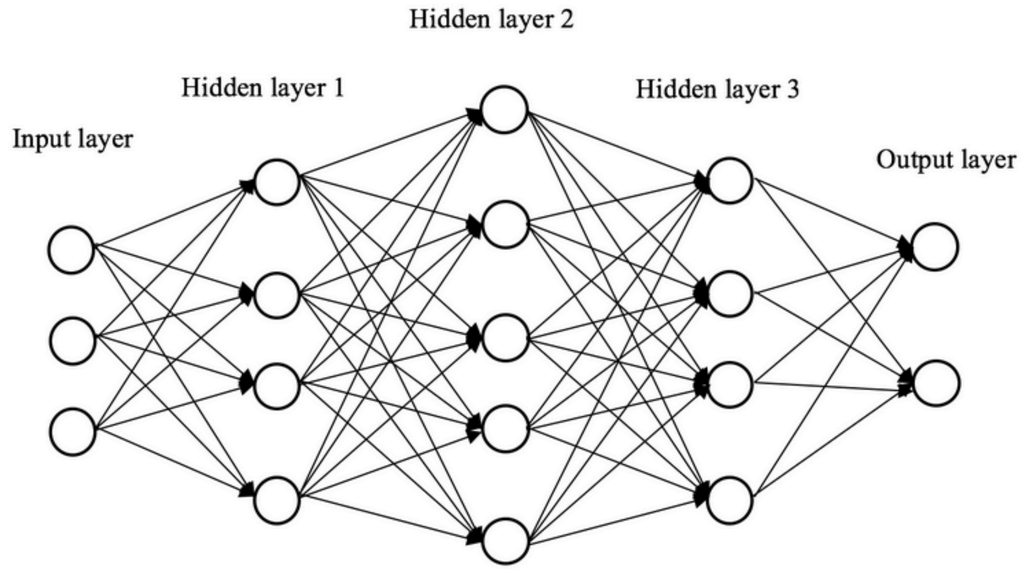
E-317/517 HIGH PERFORMANCE COMPUTING, Spring 2024

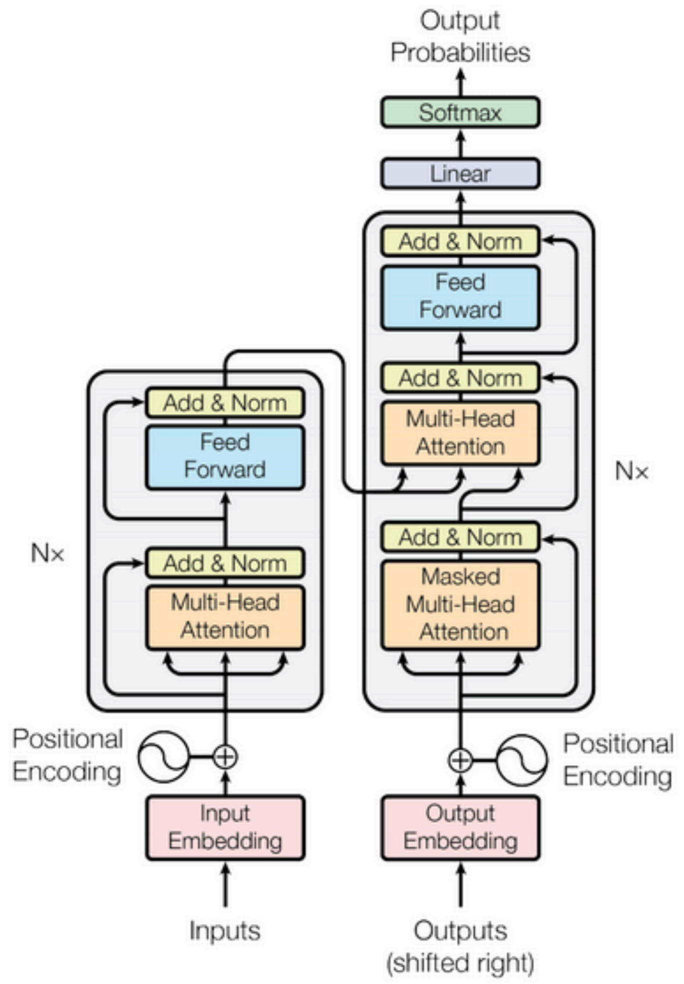
INDIANA UNIVERSITY BLOOMINGTON

Deep Learning

- Machine Learning is a branch of AI that leverages data and algorithms for insights
- Classified into: supervised, unsupervised, semi-supervised, reinforcement learning
- Deep Learning is a subset of ML based on deep or artificial neural networks with representation learning
- DNN/ANNs are inspired by the human brain







Exponential Growth in Model Size

- **2018**: GPT-1 (100M parameters), BERT (340M)
- **2019**: Transformer-XL (257M), GPT-2 (1B)
- **2020**: BART (140M), DialogGPT (1.5B), Turing-NLG (17B)
- **2021**: ViT (630M), GPT-Neo (20B), DALL-E (12B)
- **2022**: Stable Diffusion (890M), Megatron-Turing-NLG (530B), PaLM (540B), GLaM (1.2T)
- **2023**: GPT-3.5 (1.3B, 6B and 175B), Chat-GPT (175B), Bard (137B), LLaMa (7-65B), Gemini (?)

According to OpenAI, the compute requirements to train SOTA DNNs doubles every 3.4 months!

The '*Learning*' in Deep Learning

- ***Gradient descent*** is the workhorse of DL that enables iterative learning

$$w_{t+1} = w_t - \eta \nabla f \quad \text{where} \quad \nabla f = \partial \mathscr{L} / \partial w_t$$

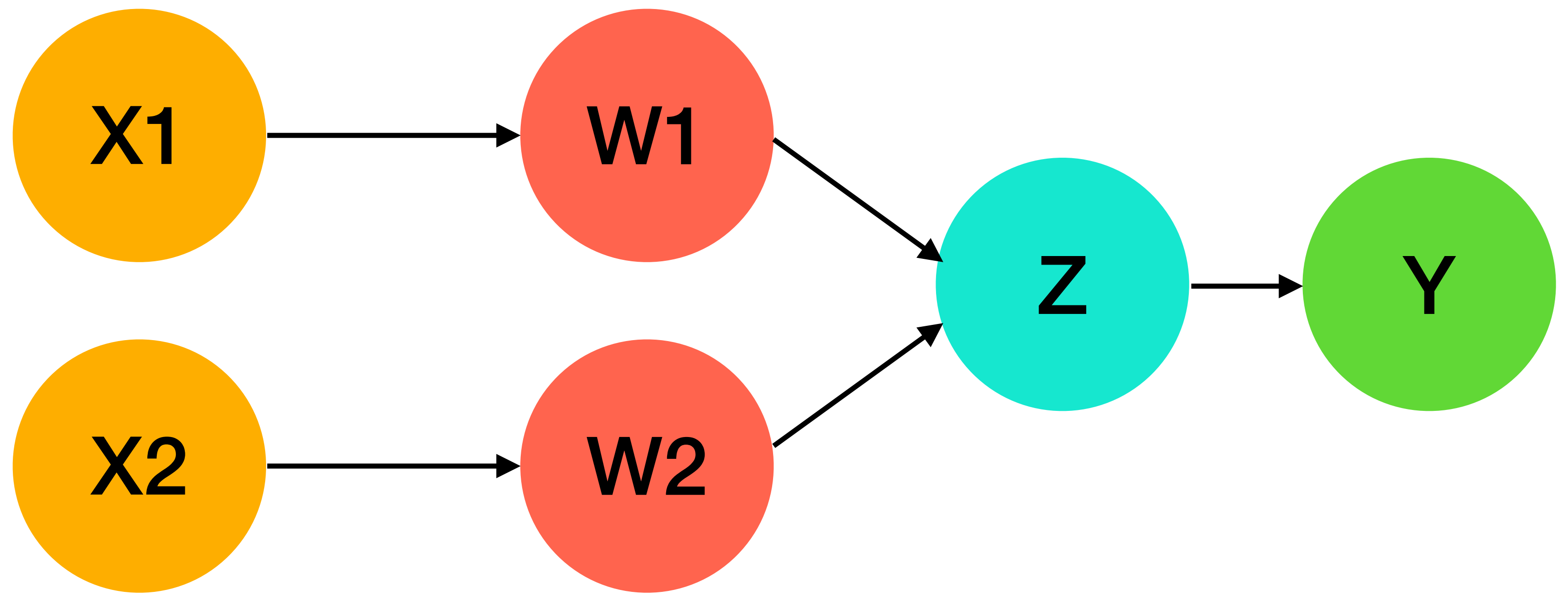
- Gradient descent is inherently sequential as it relies on the chain rule of calculus: $\frac{\partial}{\partial x} f(g(x)) = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$

- DNN training is an iterative and repetitive process with three key phases: ***forward pass***, ***backward pass*** and ***parameter update***

- Model quality influenced by a number of training-specific variables or ***hyperparameters***; e.g., choice of activation function, step-size or learning-rate, dropout, regularization, order of optimization, mini-batch, number of iterations or epochs, etc.

$$Z = WX$$

$$Y = \sigma(Z)$$



$$Z_{pred} = W_1 X_1 + W_2 X2$$

$$Y_{pred} = \sigma(Z_{pred})$$

$$\mathscr{L} = || Y_{truth} - Y_{pred} ||^2$$

$$\frac{\partial \mathscr{L}}{\partial Z} = \frac{\partial \mathscr{L}}{\partial Y} \cdot \frac{\partial Y}{\partial Z}$$

$$\frac{\partial \mathscr{L}}{\partial W} = \frac{\partial \mathscr{L}}{\partial Z} \cdot \frac{\partial Z}{\partial W}$$

Training Data for Gradient Descent

- The more the data, the better is the model quality
- Given the training data, Gradient descent can be computed over :
  - The entire training data (*Full GD*)
  - A random sample (*Stochastic GD*)
  - A batch of data (*Mini-batch GD*)

Each of these variants impacts the training throughput and statistical performance of a model

Hardware/Software for Deep Learning

- Training DNNs is compute intensive and requires numerous FLOPs or multiply-accumulate (MACs) operations on massive tensors at *every* iteration
- Ideas of MLP, CNNs and LSTMs existed long before the last decade!
- GPUs GTX580 first used to train AlexNet in 2012 with model-parallelism
- Led to development of ASICs with dedicated MMUs for MIMD execution
- Tensor processing unit (TPC) v1.0 was 15x faster and 30x more efficient than NVIDIA K80 at the time
- Various DL frameworks developed over the years: *Caffe, Keras, DistBelief, MXNet, TensorFlow, CNTK, Petuum, PyTorch*

Parallelizing Deep Learning

- Training data for SOTA models grows exponentially, and so do the DNNs themselves!
- Given a system, DL parallelization is done either to *accelerate* or *accommodate* training
- Broadly classified as:
  - **Data Parallelism**
  - **Model Parallelism**
  - **Pipeline Parallelism**

For this talk and assignment, we will mainly cover data-parallelism

Data Parallelism

- Multiple processes collaboratively train a model such that each worker contains a local model replica or copy and trains independently on a different subset of data

- The exact communication pattern may vary with different distributed data-parallel algorithms

- Can be broadly classified as:

    - *Bulk-synchronous parallel* or *BSP training*

    - *Asynchronous parallel* or *ASP training*

    - *Semi-synchronous parallel* or *SSP training*

Bulk-Synchronous Parallel Training

- Multiple independent processes train independently on i.i.d. sampled data and aggregate their updates collectively at the end of ***each*** iteration

$$w_{i+1} = w_i - \eta \frac{1}{N} \sum_{n=1}^{N} \frac{1}{|b|} \sum_{x_{(n,i)} \in \mathcal{B}_n} \frac{\partial}{\partial w_i} \mathcal{L}(x_{(n,t)}, w_t)$$

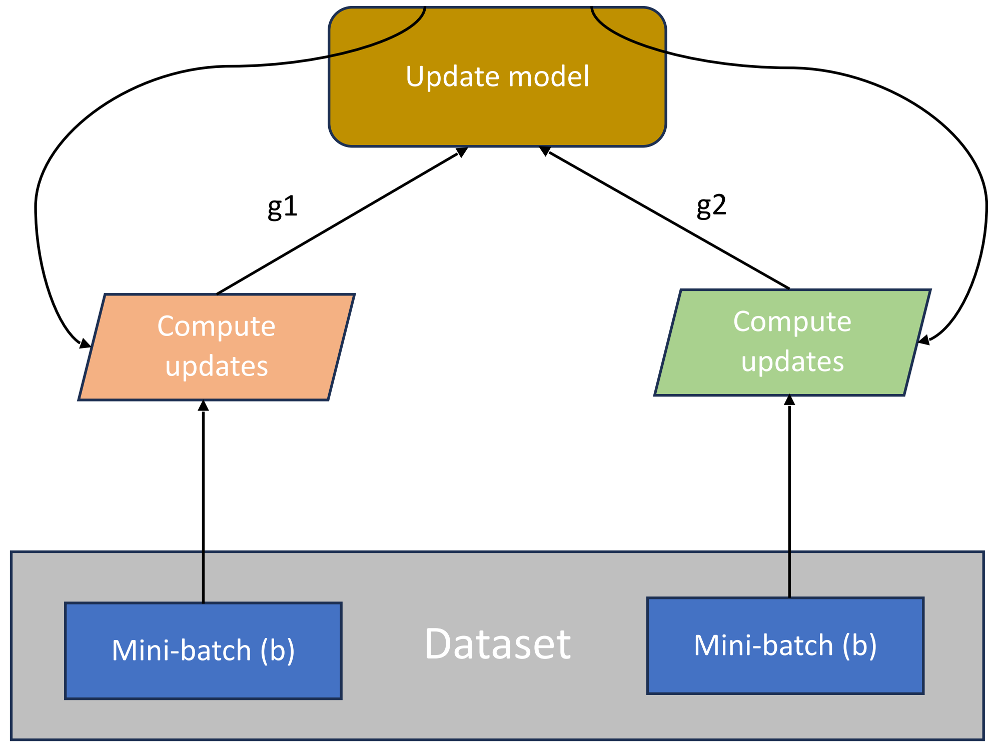
$$t_{iteration} = t_{compute} + t_{sync} + t_{IO}$$

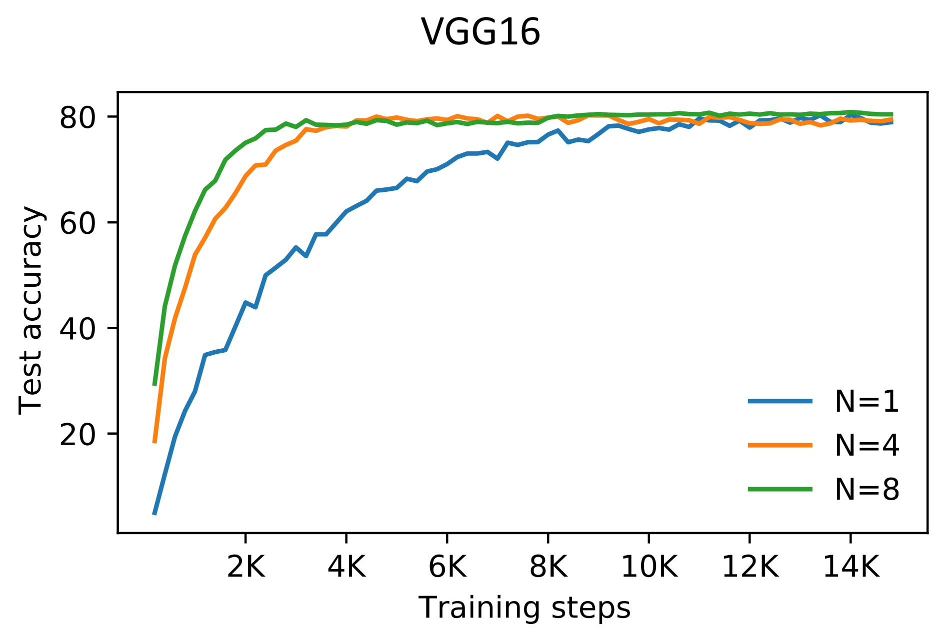
- Has a convergence-rate of $\mathcal{O}(\frac{1}{\sqrt{NI}})$ as we perform more work per-iteration

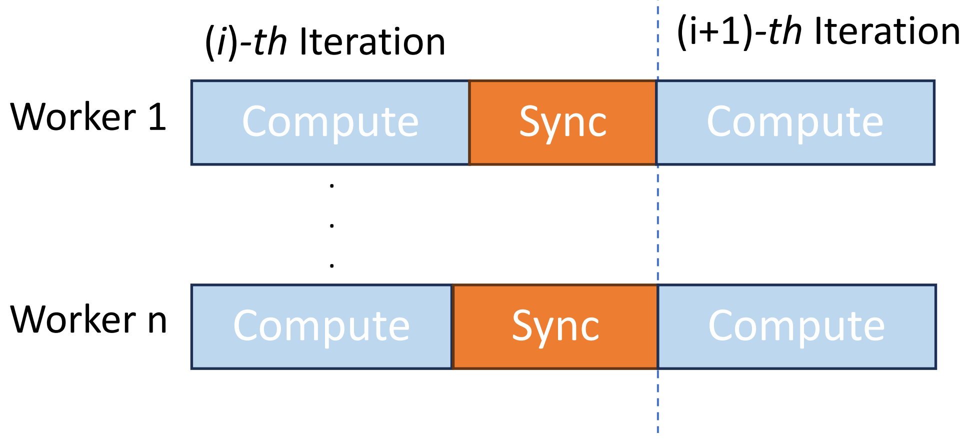
ring any bells w.r.t scaling?

- Has a ***synchronization barrier*** at the end of each iteration, so communication bound especially in case of stragglers

- Maybe centralized or decentralized that affects overall training throughput





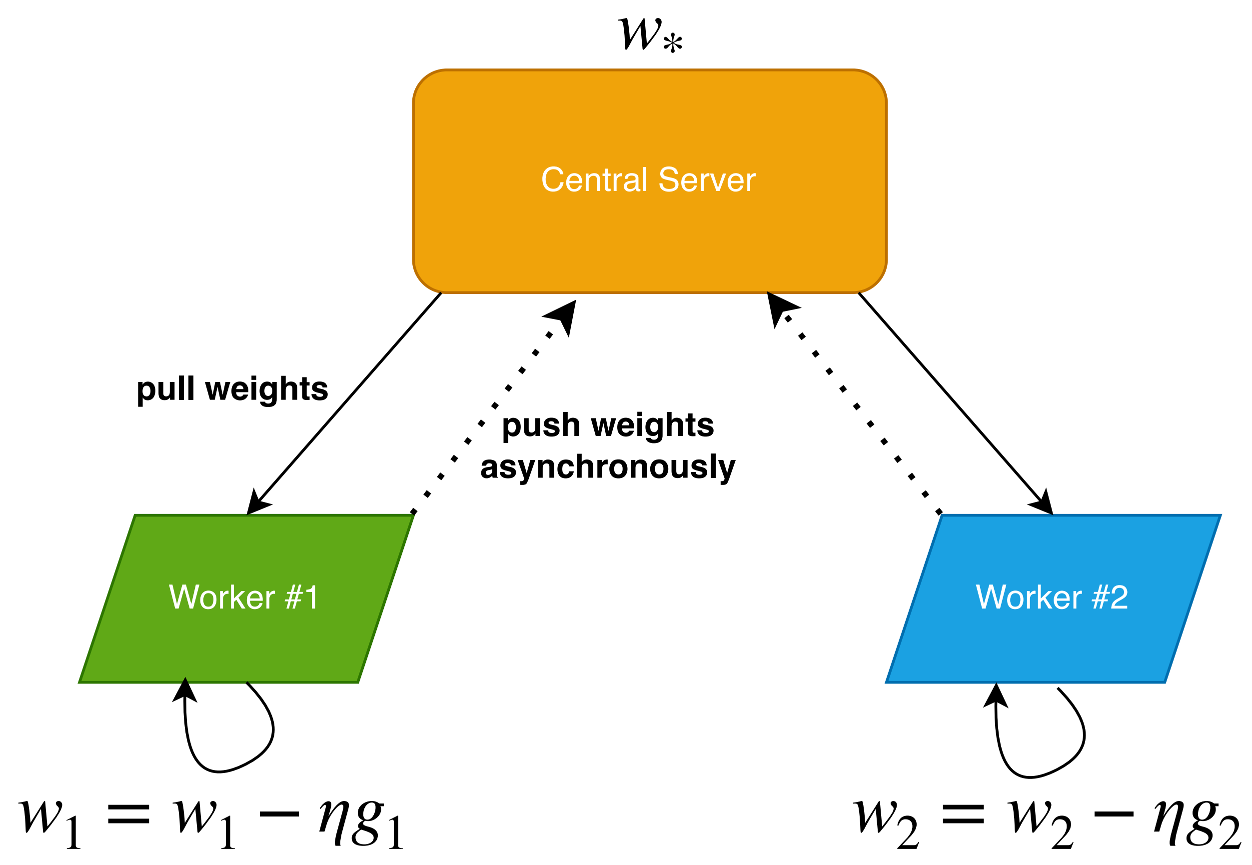


Asynchronous Parallel Training

- To mitigate high communication overhead at every iteration, each worker trains an independent model state in a centralized system setting

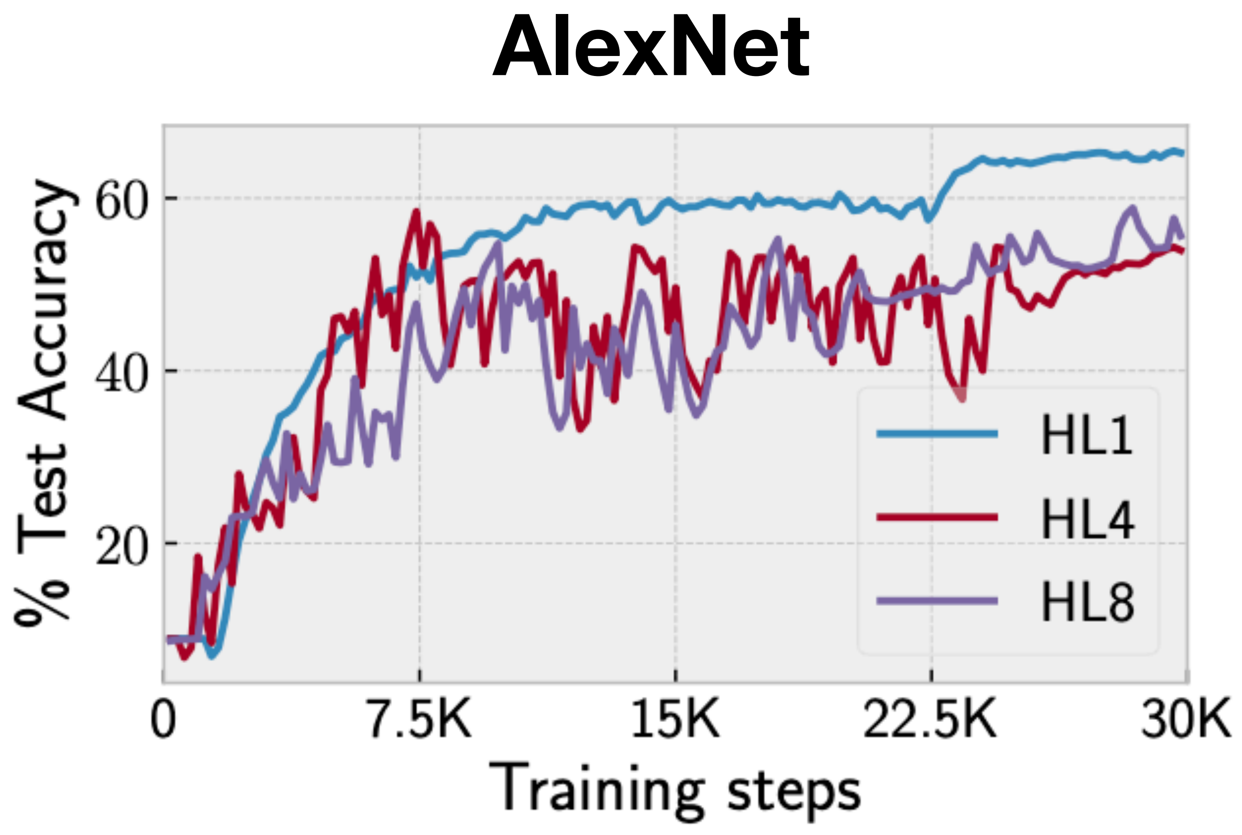
$$w_{t+1} = w_t - \eta \frac{1}{|b|} \sum_{x_{(n,t)} \in \mathscr{B}_n} \frac{\partial}{\partial w_t} \mathscr{L}(x_{(n,t)}, w_{n,(t-\tau_{n,t})}) \;\; \forall \;\; n \in [\; 1,2,3,\ldots N \;]$$

$$t_{iteration} = t_{compute} + t_{pull-from-server} + t_{IO}$$



- Compared to BSP, lesser work done per-iteration; converges in $\mathcal{O}(\frac{1}{\sqrt{I}})$

- May suffer from staleness in model updates; ***as heterogeneity rises in a cluster, staleness gets worse and degrades model quality***



Semi-Synchronous Parallel Training

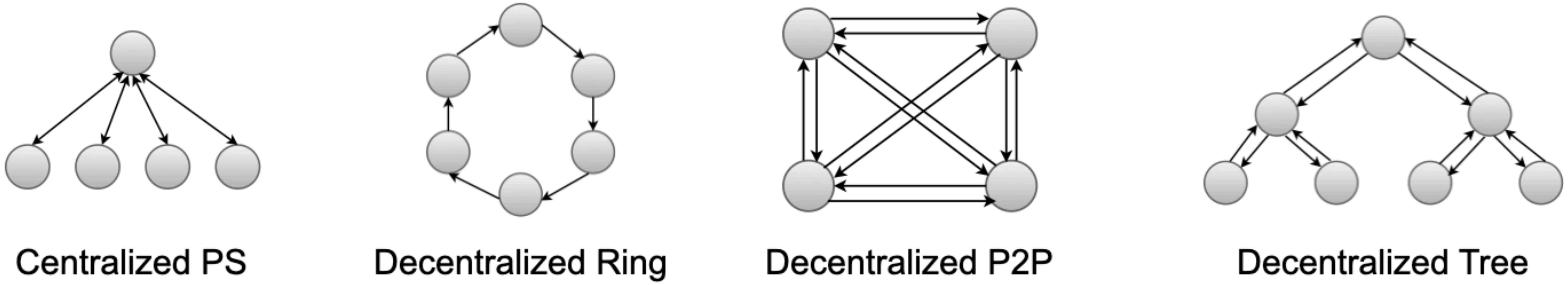
- Middle ground between synchronous and asynchronous training

- In Stale-synchronous parallel, training processes are allowed to run asynchronously, but only up to a certain ***staleness threshold***

$$w_{n,t+1} = w_0 - \eta \sum_{i=1}^{t-s} \sum_{j=1}^{N} \frac{1}{|b|} \sum_{x_{(j,i)} \in \mathcal{B}_n} \frac{\partial}{\partial w_{j,i}} \mathcal{L}(x_{(j,i)}, w_{j,i}) - \eta \sum_{i=t-s}^{t} \frac{1}{|b|} \sum_{x_{(n,i)} \in \mathcal{B}_n} \frac{\partial}{\partial w_{n,i}} \mathcal{L}(\mathrm{x}_{(n,i)}, w_{n,i}) - \eta \sum_{(j,i) \in \mathcal{S}_{n,t+1}} \frac{1}{|b|} \sum_{x_{(j,i)} \in \mathcal{B}_n} \frac{\partial}{\partial w_{j,i}} \mathcal{L}(x_{(n,i)}, w_{j,i})$$

- In fact, stale-synchronous parallel generalizes to BSP or ASP training based on the set value of staleness threshold

Cluster Topology



- Can be centralized or decentralized
- Physical arrangement of nodes or ***topology*** determines degree of distribution & latency between workers
- Based on the communication pattern of a distributed algorithm and its cost, overall throughput may vary!

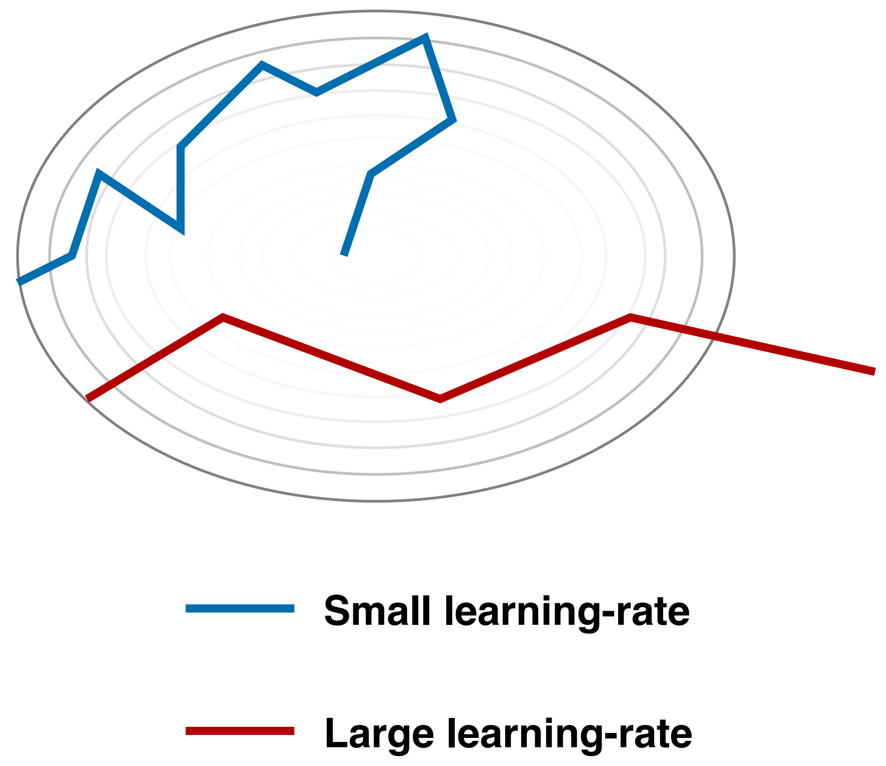
Communication Cost Analysis

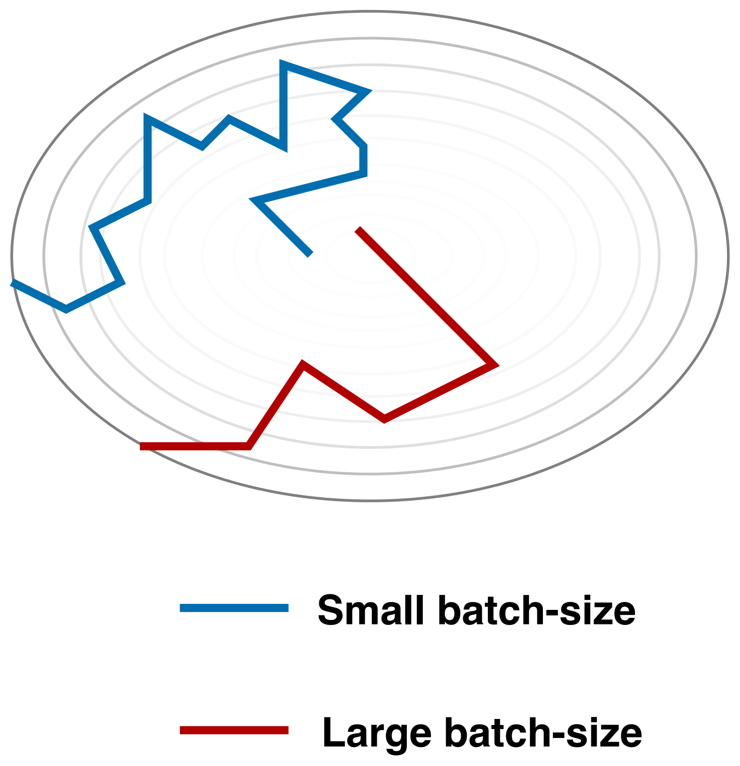
- Different collectives have different communication costs associated with them, based on cluster-size, model-size, latency, bandwidth and the specific collective implementation
- For e.g., Ring-AllReduce in decentralized systems (ringARexample)
- Based on the $(\alpha - \beta)$ communication cost model, where alpha is the ***latency*** and 1/beta is the communication ***bandwidth***

Operation	Latency Complexity	BW Complexity	Communication cost
PS	$\mathcal{O}(1)$	$\mathcal{O}(MN)$	$2\alpha + 2(N-1)M\beta$
Ring-Allreduce	$\mathcal{O}(N)$	$\mathcal{O}(M)$	$2(N-1)\alpha + 2\frac{(N-1)}{N}M\beta$
Tree-Allreduce	$\mathcal{O}(\log(N))$	$\mathcal{O}(M\log(N))$	$2\alpha\log(N) + 2\log(N)M\beta$
Broadcast	$\mathcal{O}(\log(N))$	$\mathcal{O}(M\log(N))$	$\alpha\log(N) + \log(N)M\beta$
Allgather	$\mathcal{O}(\log(N))$	$\mathcal{O}(MN)$	$\alpha\log(N) + (N-1)M\beta$

Statistical Efficiency in DNN Training

- Training throughput or ***parallel-efficiency*** can be improved by reducing computation, IO or communication overhead
- But distributed DNN training has a ***statistical efficiency*** aspect associated as well
- Depends on architecture specific parameters, length of training, type of optimization and scaling of DL training (lr, batch and cluster-size)
- A small learning-rate takes small steps towards minima, while a very high value may overshoot and diverge the model
- Mini-batch size influences the quality of gradients; larger batches take fewer steps to reach minima compared to smaller batches



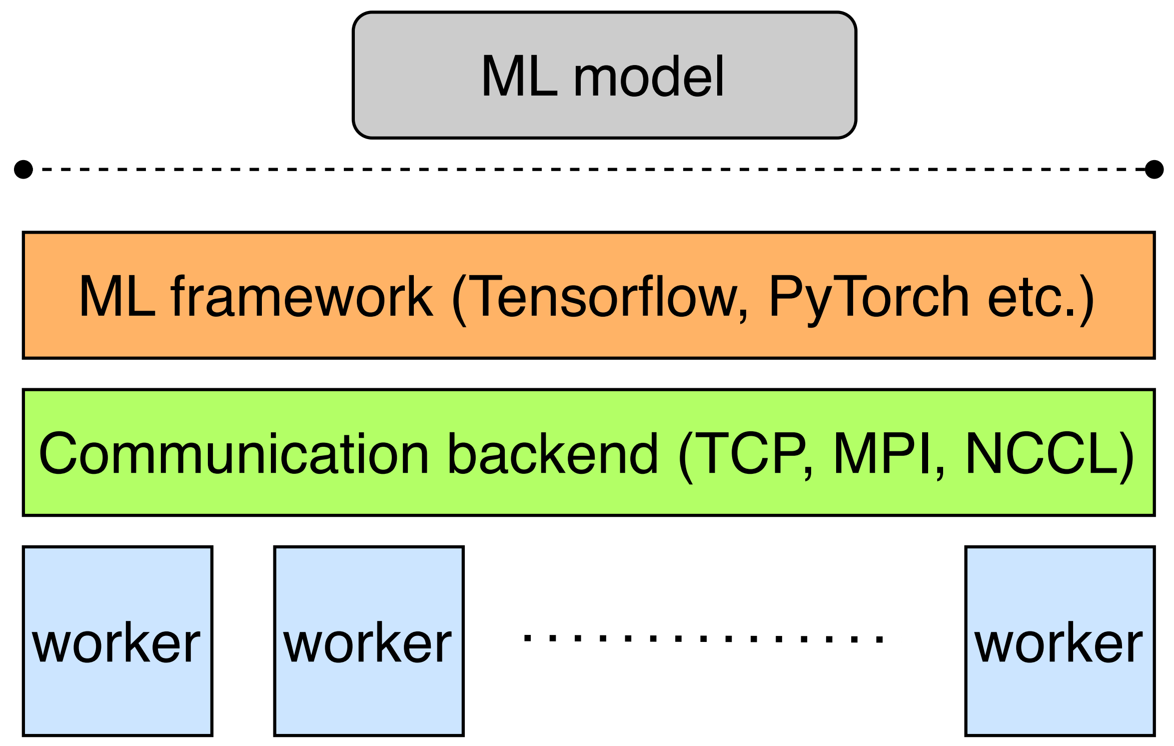


Training with PyTorch

- Training a CNN image classifier over CIFAR10 involves:
  - Loading and normalizing training data using ***torchvision*** module
  - Define a DNN
  - Define a loss and optimizer function
  - Train model over training data
  - Test model over test data
- Example: ***basicnn_train.py***

Distributed Training with PyTorch

- PyTorch optimizes performance with native support for asynchronous execution from Python
- ***DataParallel (DP)*** and ***DistributedDataParallel (DDP)*** modules in PyTorch are SIMD training paradigm that single/multiple machine multi-GPU settings
- ***FullyShardedDataParallel*** on single machine multi-GPU when model does not fit on one
- ***RPC*** framework allows for other distributed training abstractions
- Collective Communication is supported via ***MPI***, ***NCCL*** and ***Gloo***; compatible collectives ([here]() and [here]())
- In multi-node settings you can specify the network interface to use for communication



Thank you!