



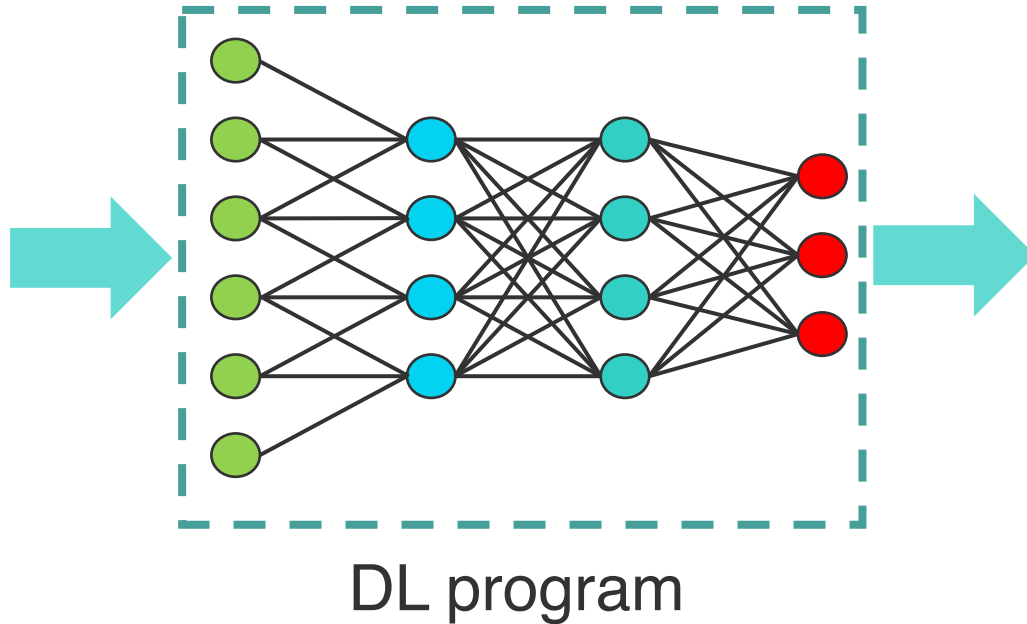
Carnegie
Mellon
University

Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters

Hao Zhang

Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang,
Zhiting Hu, Jianliang Wei, Pengtao Xie, Eric P. Xing
Petuum Inc. and Carnegie Mellon University

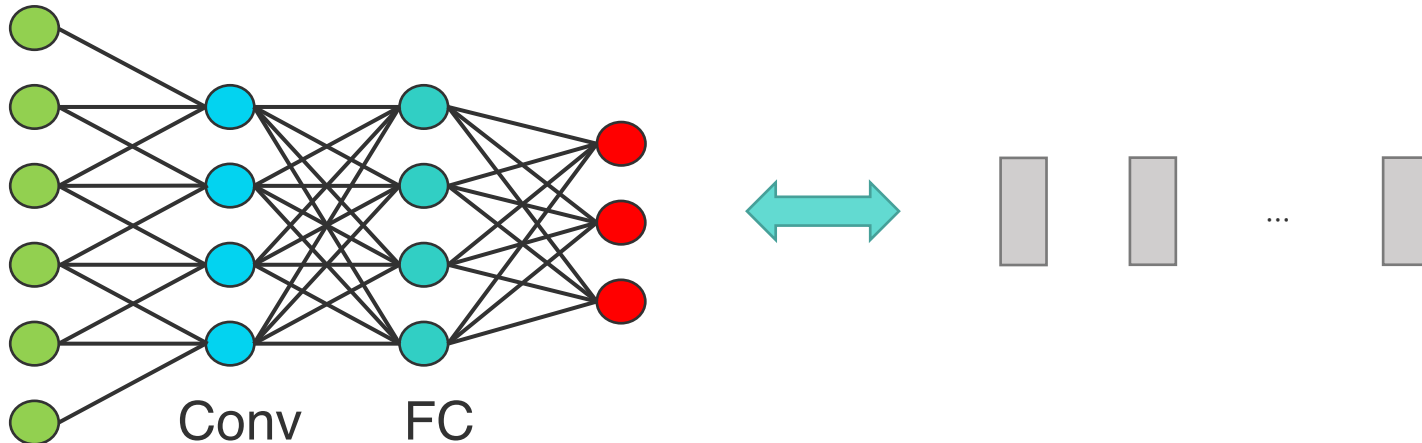
Deep Learning



Training data:
images w/ labels

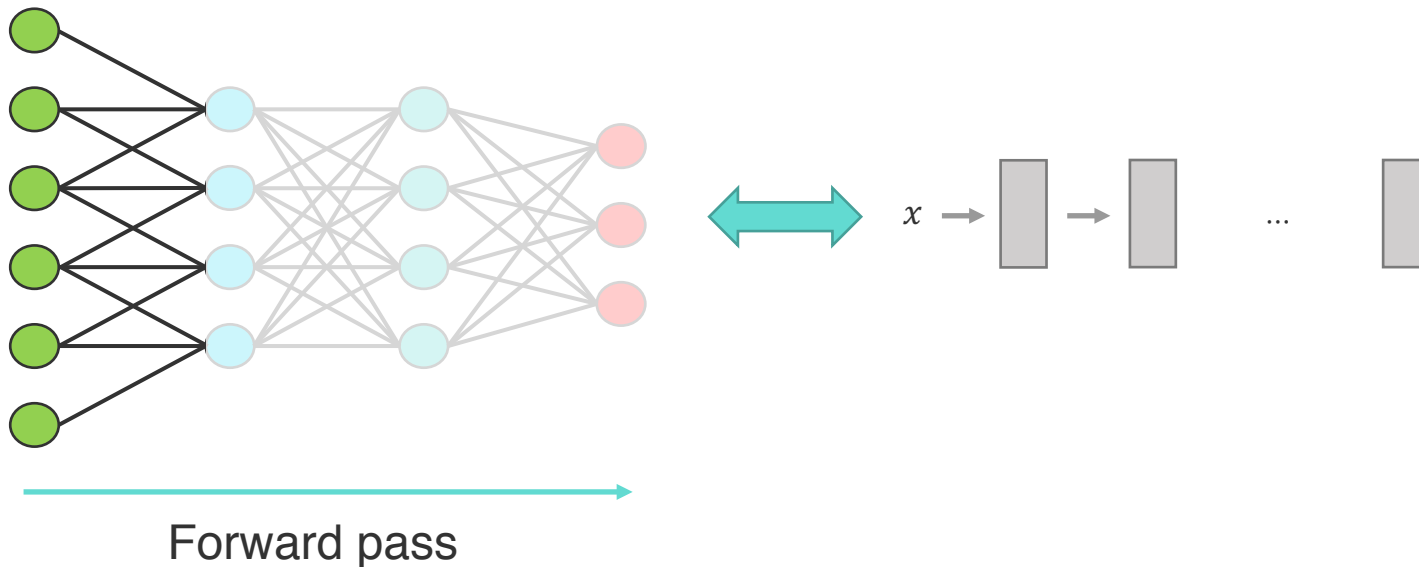
Forward through a Neural Network

- Essentially, A neural network is composed of a few layers
- A layer in a NN is composed of a few (heavy) computational operations



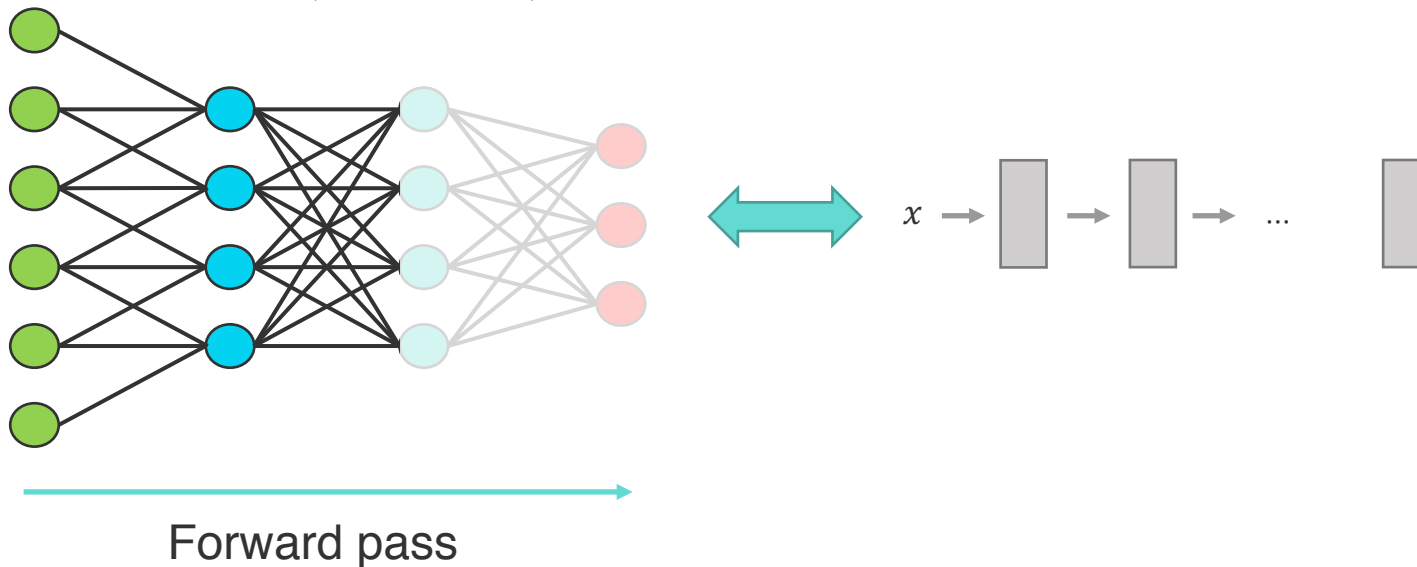
Forward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Forward Pass: input x , Loss \mathcal{L}



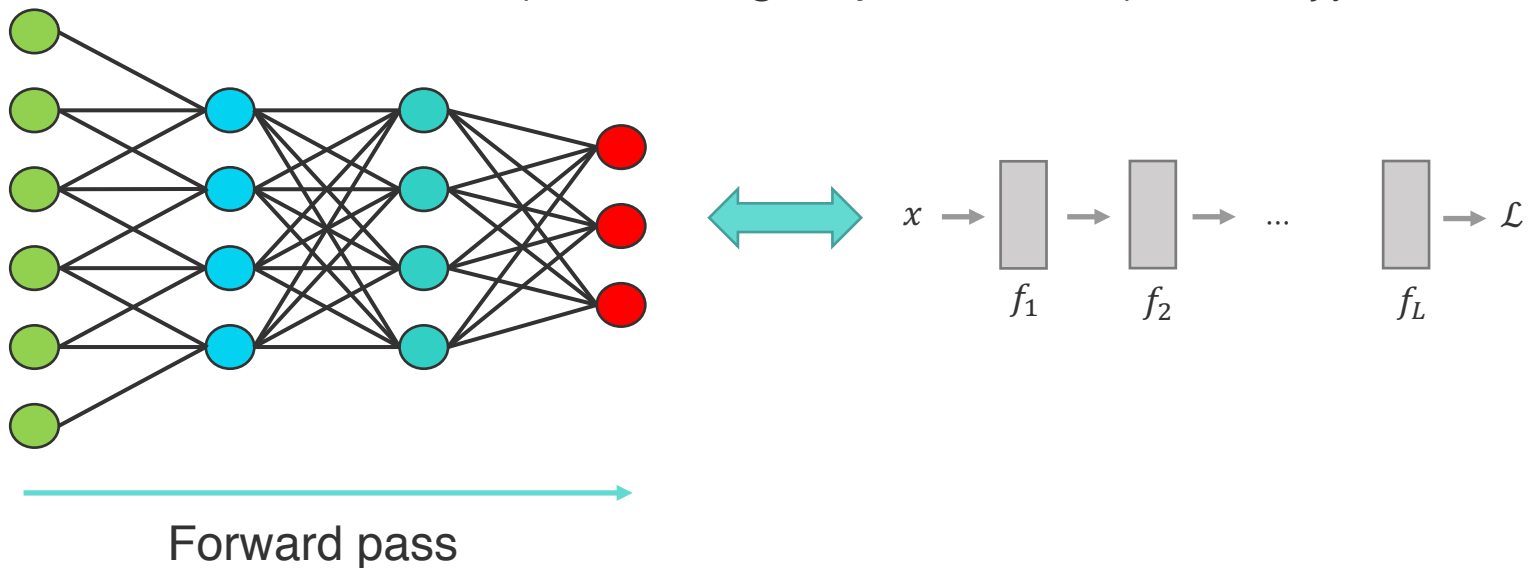
Forward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Forward Pass: input x , Loss \mathcal{L}
 - Forward through the NN, one layer at a time
 - We use l ($l = 1, \dots, L$) to denote the l th layer in a neural network



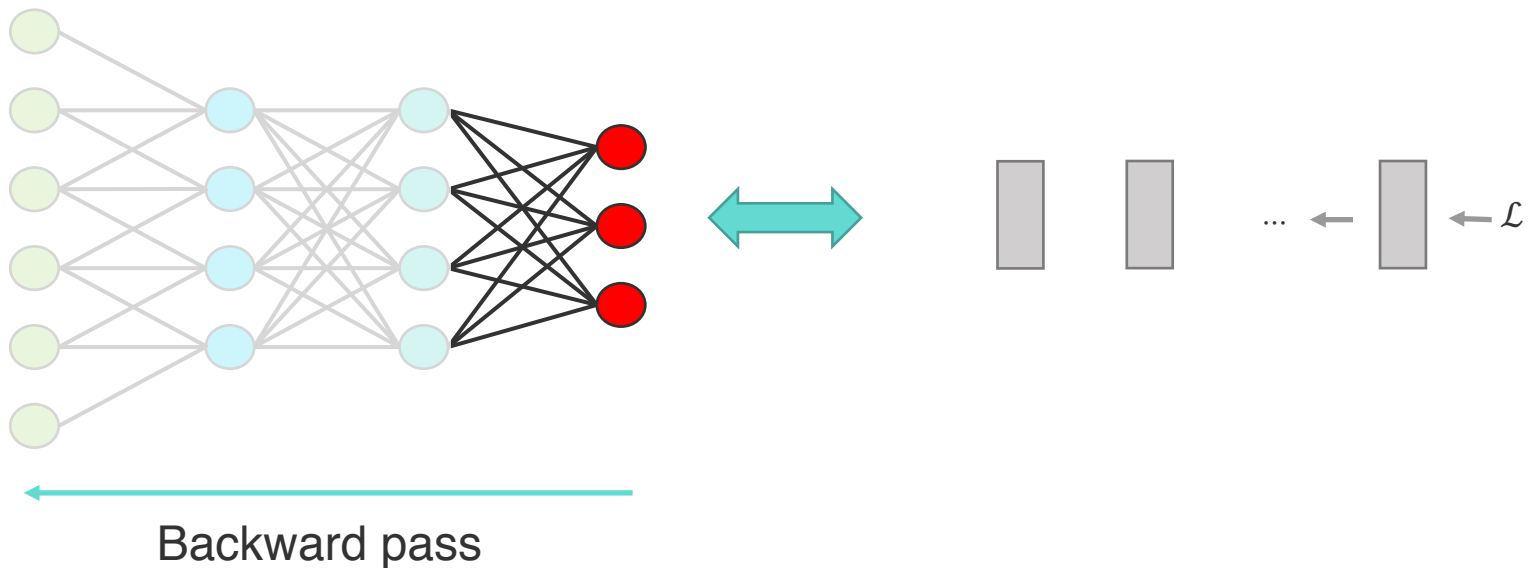
Forward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Forward pass: input x
 - Forward through the NN, one layer at a time, until we get the loss \mathcal{L}
 - Denote the forward pass through layer l as an operation f_l



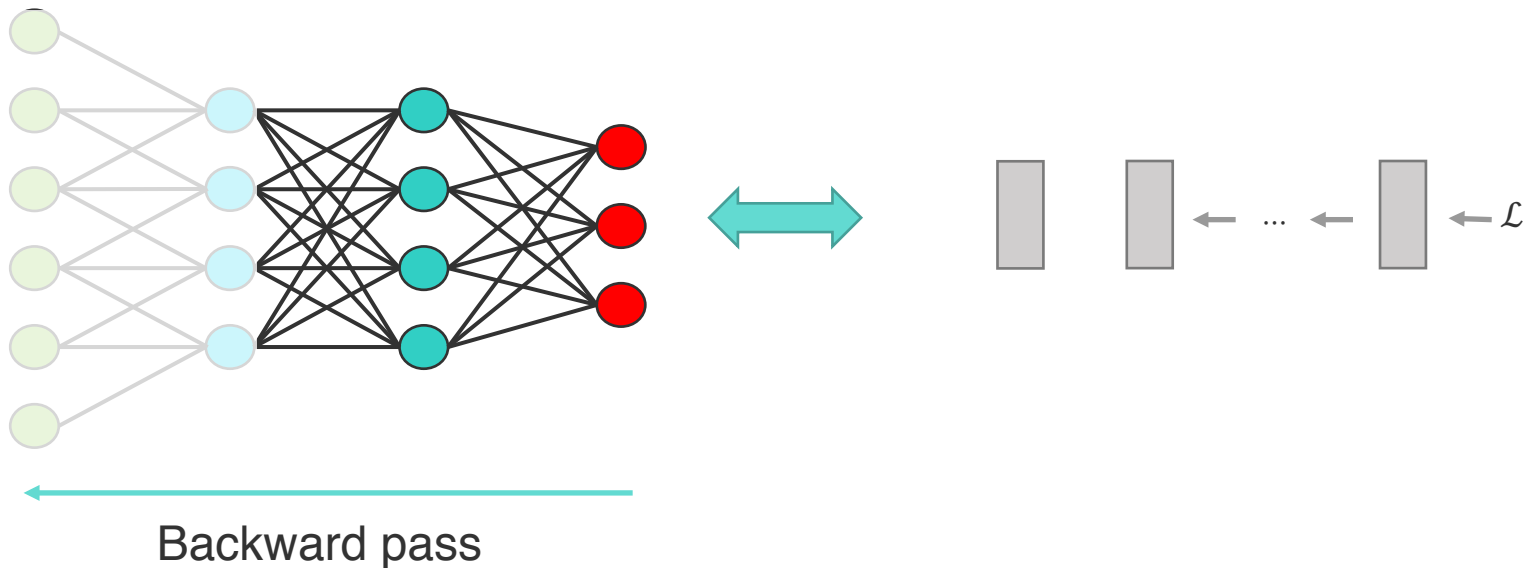
Backward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Backward pass: input is the loss



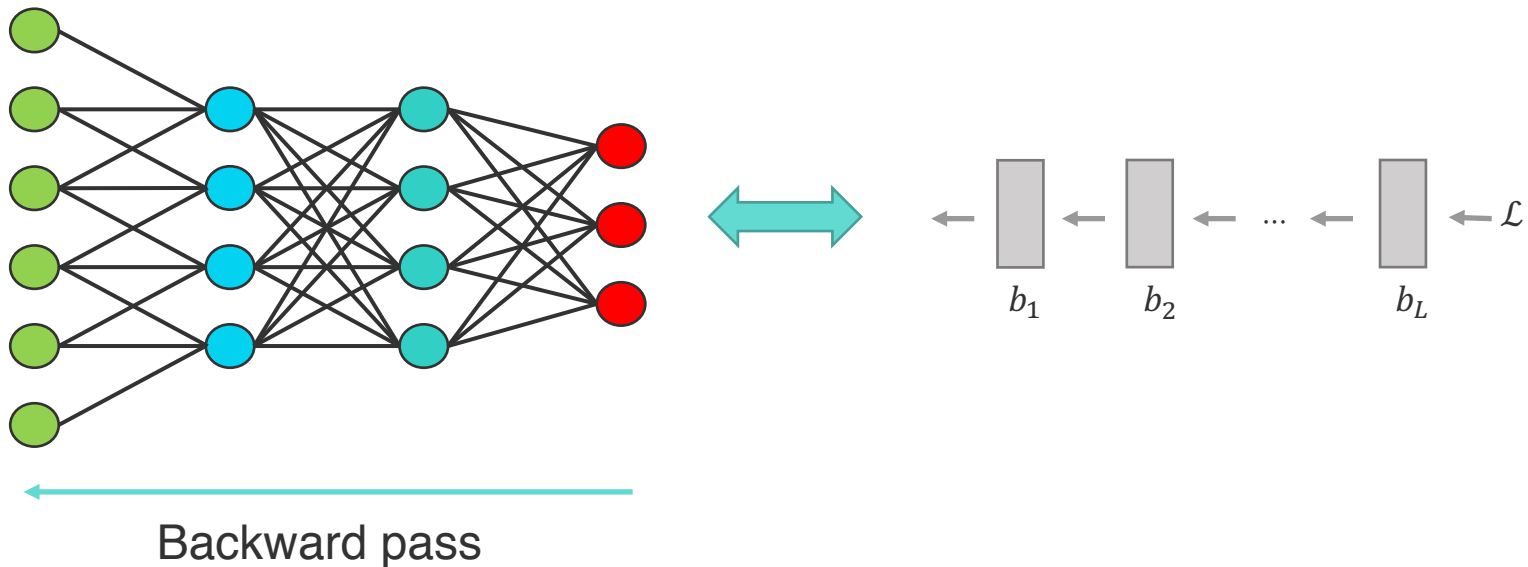
Backward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Backward pass derives the gradients of the parameters of a layer when backward through it



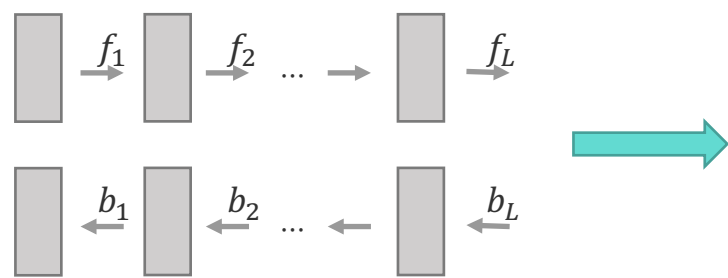
Backward through a Neural Network

- Backpropagation (BP) is a principled algorithm to train NNs
- BP involves two passes through the network
 - Backward derives the gradients of the parameters of a layer when backward through it
 - Denote the backward pass through layer l as an operation b_l



Training a Neural Network

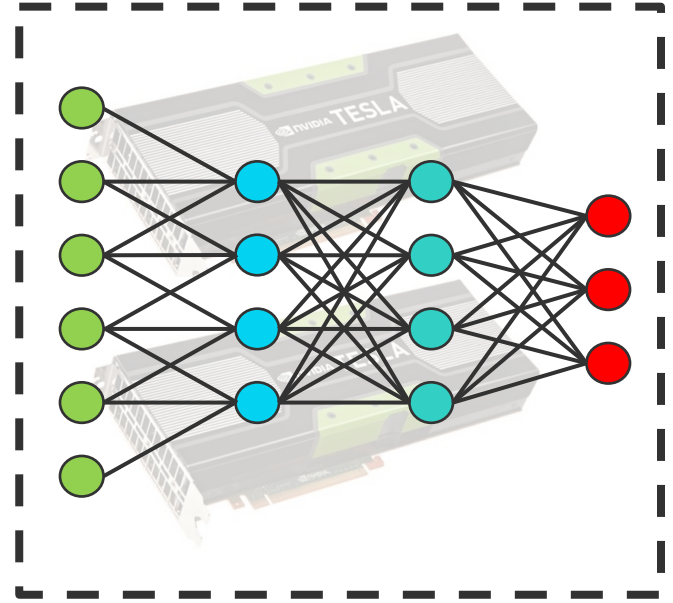
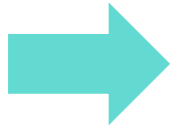
- Stochastic Gradient Descent (SGD) via Backpropagation
 - Forward: sequentially executing f_1, f_2, \dots, f_L
 - Backward: sequentially executing b_L, b_{L-1}, \dots, b_1
 - Update: apply the gradients to update the model parameters
 - Repeat
- Formally, an iterative-convergence formulation


$$\theta^{(t)} = \theta^{(t-1)} + \varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})$$

Model parameters Forward Data

Backward

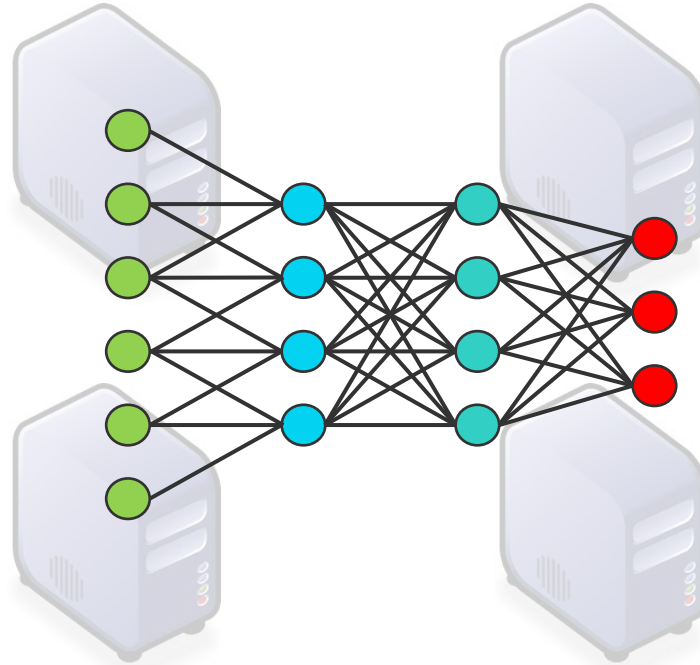
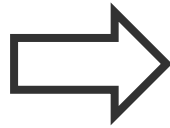
Deep Learning on GPUs



Training data:
images w/ labels

Computational Worker w/ GPUs

Deep Learning on Distributed GPUs

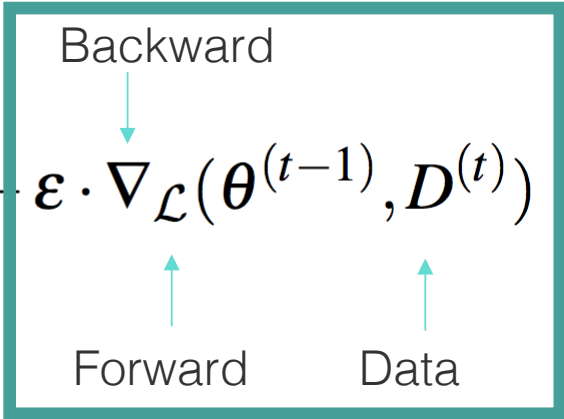


Large scale
Training data

A Cluster of Workers with GPUs

Distributed Deep Learning

- Distributed DL: parallelize DL training using multiple machines.
- i.e. we want to accelerate the heaviest workload (in the box) to multiple machines

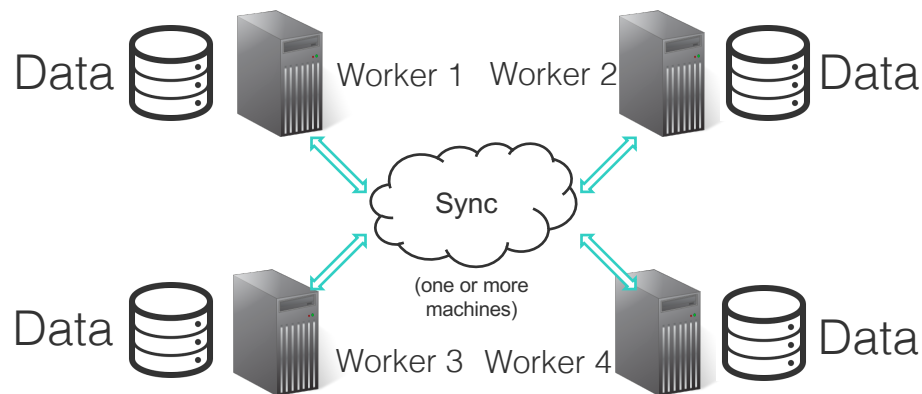
$$\theta^{(t)} = \theta^{(t-1)} + \epsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})$$


The diagram shows the equation $\theta^{(t)} = \theta^{(t-1)} + \epsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})$ with a teal box highlighting the term $\epsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})$. A teal arrow labeled 'Backward' points down to the gradient symbol $\nabla_{\mathcal{L}}$. Two teal arrows labeled 'Forward' and 'Data' point up to $\theta^{(t-1)}$ and $D^{(t)}$ respectively. A teal arrow points from the text below to the bottom-left corner of the box.

Forward and backward are the main computation (99%) workload of deep learning programs.

Data Parallelism with SGD

- We usually seek a parallelization strategy called data parallelism, based on SGD
 - We partition data into different parts
 - Let different machines compute the gradient updates on different data partitions
 - Then aggregate/sync.



Data Parallel SGD

- Data-parallelism requires every worker to have read and write access to the shared model parameters θ , which causes communication among workers;

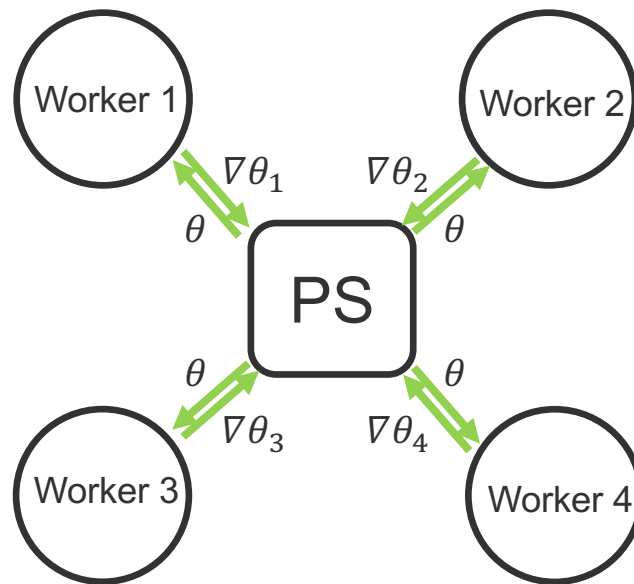
$$\theta^{(t+1)} = \theta^{(t)} + \epsilon \sum_{p=1}^P \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

The diagram illustrates the Data Parallel SGD update equation with four annotations:

- In total P workers**: An arrow points to the summation index P .
- Data partition p**: An arrow points to the data partition $D_p^{(t)}$.
- Happening locally on each worker**: An arrow points to the gradient term $\nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$.
- Collect and aggregate before application, where communication is required**: An arrow points to the summation operation $\sum_{p=1}^P$.

Parameter Server

- Parameter server is a shared key-value storage that provides a shared access for the global model parameters θ for ML
- The server is virtual – physically, it could be distributed (instead of centralized), i.e., a distributed key-value storage



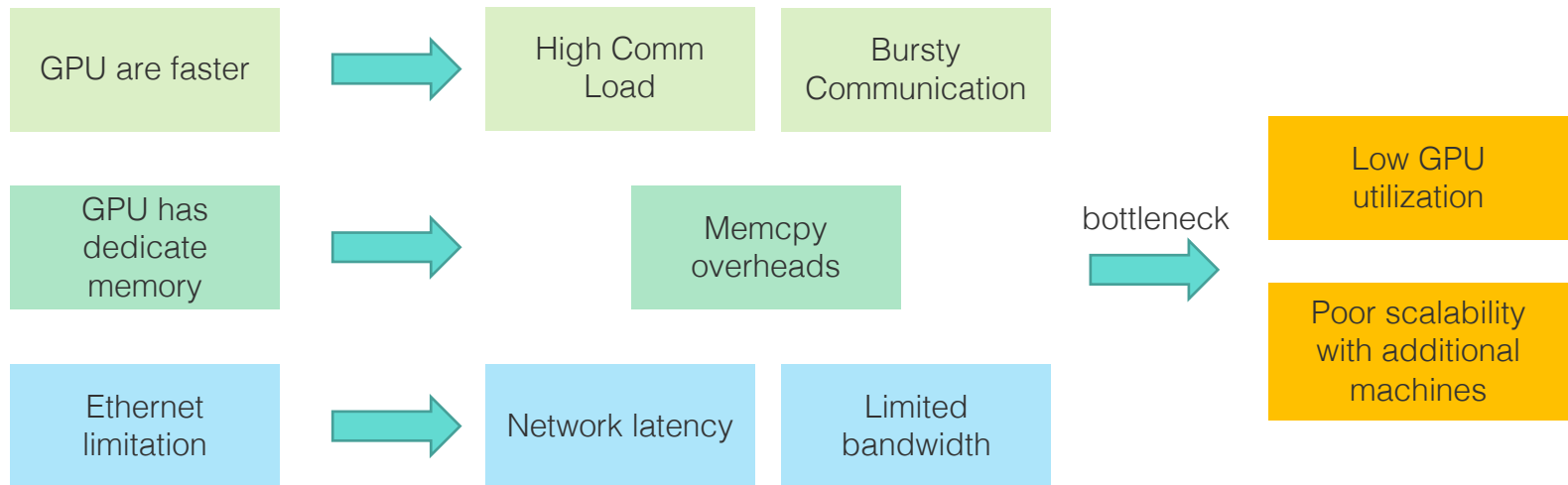
*Deal et al., 2012, Ho et al., 2013
Li et al., 2014
Cui et al., 2014, Cui et al., 2015
Wei et al., 2015
Zhang et al., 2015*

Parameter Server for DL on GPUs

- Deep learning can be trivially data-parallelized over distributed workers using PS by 3 steps:
 - Each worker computes the gradients (∇L) on their own data partition (D_p) and send them to remote servers;
 - servers receive the updates and apply (+) them on globally shared parameters;
 - Each worker pulls back the updated parameters (θ_t)
- However, directly applying PS for GPU-based distributed deep learning will underperform (as will show later).

Parameter Server on GPU Clusters

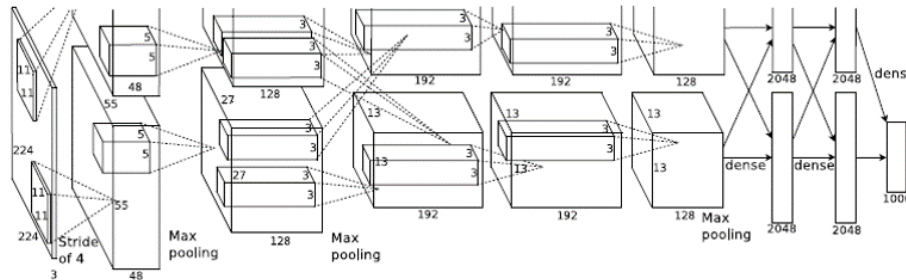
- What prevents the trivial realization of distributed DL on GPUs?
- Communication challenges
 - GPUs are at least one order of magnitude faster than CPUs
 - Ethernet has very limited bandwidth



PS on GPU Clusters: an Example

- Train AlexNet, gradient generation rate 240M floats/(s*GPU)
 - 61.5M float parameters, 0.25s/iter on Geforce Titan X (batchsize = 256)

Figure from
Krizhevsky et al. 2012



- Parallelize it over 8 machines each w/ one GPU using PS.
- To ensure the computation not blocked on GPU (i.e. linear speed-up with additional nodes)
 - Assume: every node holds 1/8 parameters as a PS shard
 - A node needs to transfer $240M * 7/8 * 4 = 840M$ floats/s = 26Gbps

PS on GPU Clusters

- Let's see where we are

This is what the GPU workstation in most labs

One of the most expensive instances AWS could provide you (18\$/h?)

Ethernet standards

Ethernet	Rate(GBit/s)	Rate (Mb/s)	Rate (# floats/s)
1 GbE	1	125	31.25M
10 GbE	10	1250	312.5M
Infiniband	40	5000	1250M

Specialized hardware! Non-commodity anymore, unaffordable

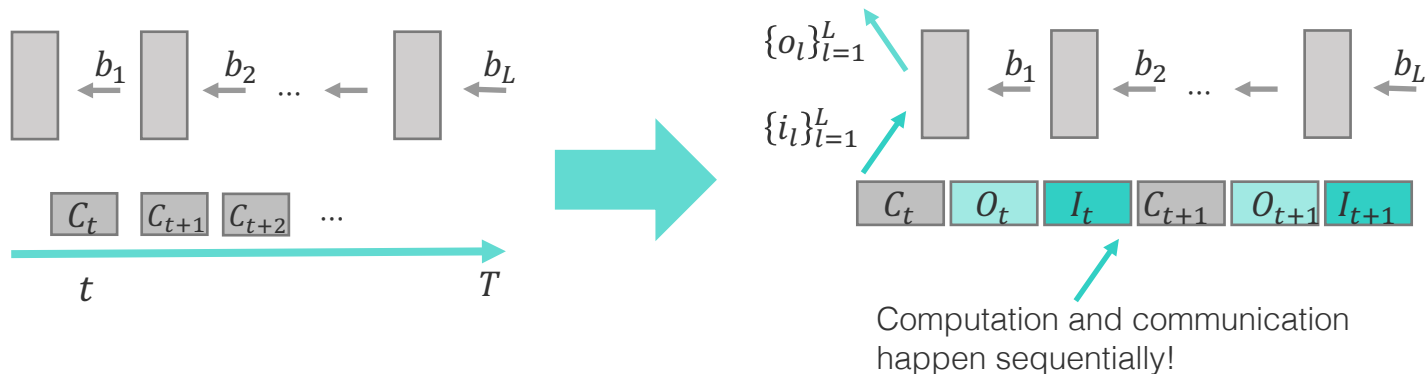
- Unfortunately, the problem will be more severe than described above
 - We only use 8 nodes (which is small). How about 32, 128, or even 256?
 - We haven't considered other issues, e.g.,
 - Memory copy between DRAM and GPU will have a non-trivial cost [Cui et al. 2015]
 - The Ethernet might be shared with other tasks, i.e. available bandwidth is even less.
 - Burst communication happens very often on GPUs (which will explain later).

Address the Communication Bottleneck

- A simple fact: communication time may be reduced, but cannot be eliminated (of course)
- Poseidon's motivation: possible ideas to address the communication bottleneck
 - Wait-free backpropagation (WFBP): hide the communication time by overlapping it with the computation time
 - Hybrid communication (HybComm): (lossless) reduce the size of messages needed to be communications

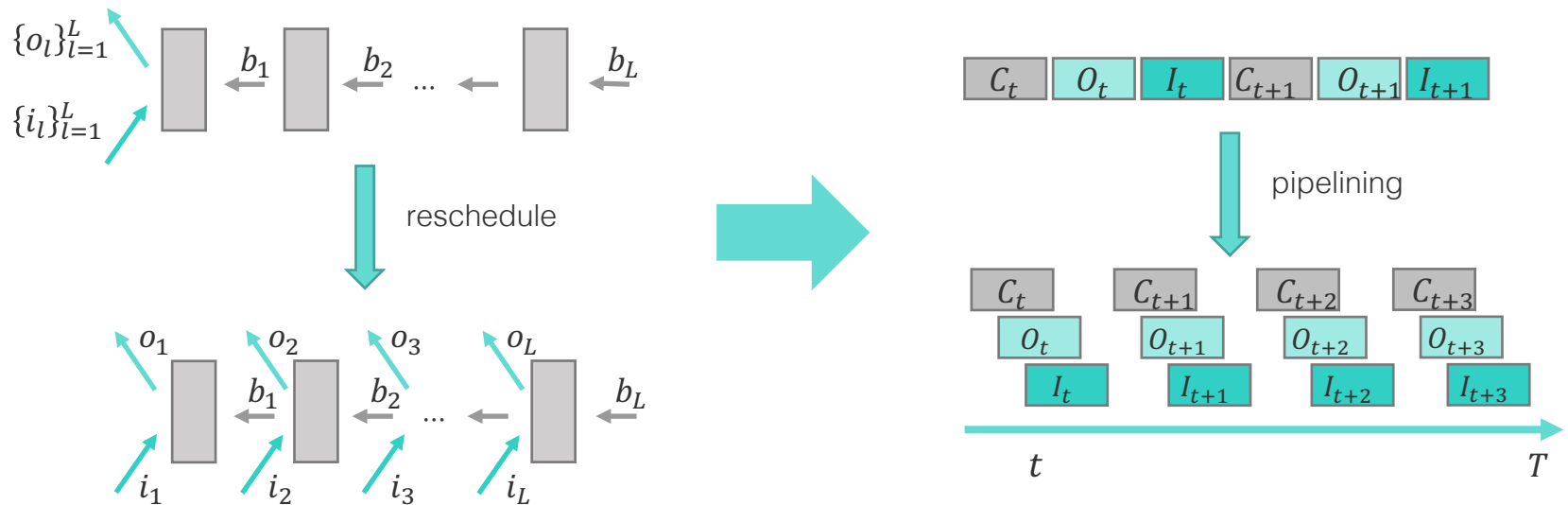
Overlap Computation and Communication

- Recall on a single node the computational flow of BP
 - b_l : backpropagation computation through layer l
 - C_t : forward and backward computation at iteration t
- On multiple nodes, when communication is involved, we introduce two communication operations
 - o_l : send out the gradients in layer l to the remote
 - i_l : pull back the globally shared parameters of layer l from the remote
 - O_t : the set $\{o_l\}_{l=1}^L$ at iteration t
 - I_t : the set $\{i_l\}_{l=1}^L$ at iteration t



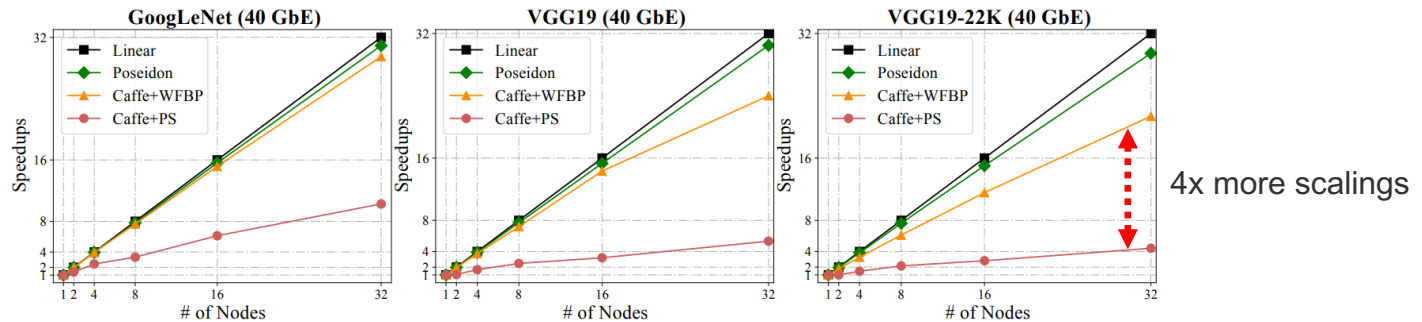
WFBP: Wait-free backpropagation

- Idea: overlap computation and communication by utilizing concurrency
 - Pipelining the updates and computation operations
 - Communication overhead is hidden under computation
 - Results: more computations in unit time

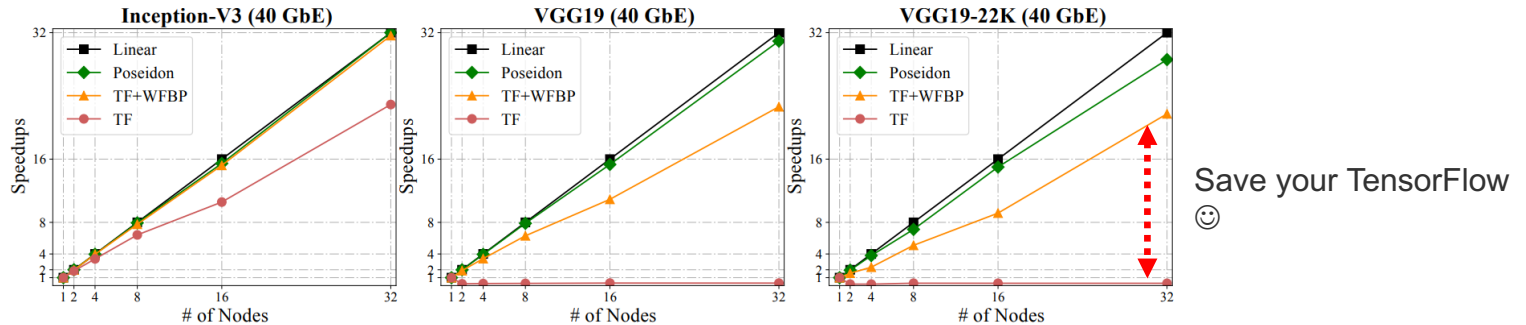


WFBP: Wait-free Backpropagation

- How does WFBP perform?
 - Using Caffe as an engine:

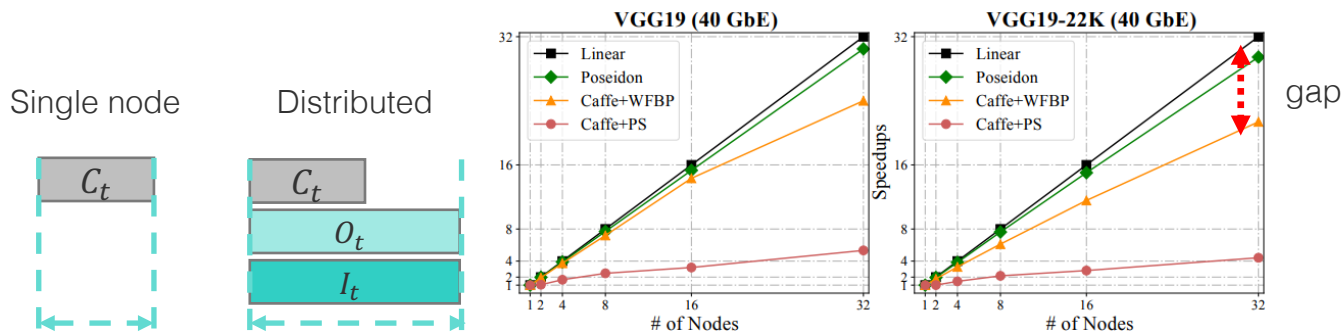


- Using TensorFlow as engine



WFBP: Wait-free Backpropagation

- Does overlapping communication and computation solve all the problems?
 - No, when communication time is longer than computation (see the figure below).
 - Empirically, if communication and computation are perfectly overlapped, how many scalability we can achieve?



Outline

- A simple fact: communication time may be reduced, but cannot be eliminated (of course)
- Poseidon's motivation: possible ideas to address the communication bottleneck
 - Wait-free backpropagation (WFBP): hide the communication time by overlapping it with the computation time
 - **Hybrid communication (HybComm): Reduce the size of messages needed to be communicated**

Sufficient Factor Broadcasting (SFB)

- Matrix-parametrized models (MPMs)

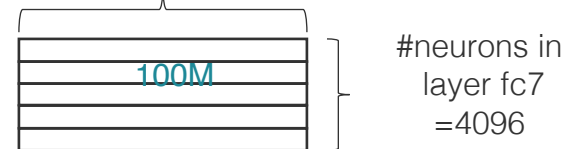
Multiclass Logistic Regression

Feature dim. = 20K



Neural Network (AlexNet)

#neurons in layer
fc6=4096



- Many MPMs have a good mathematical property
 - Full parameter matrix update ΔW can be computed as outer product of two vectors uv^T (called sufficient factors)

$$\min_W \frac{1}{N} \sum_{i=1}^N f_i(Wa_i; b_i) + h(W)$$

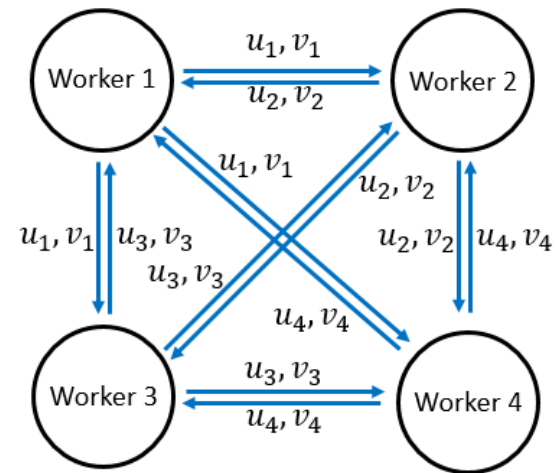
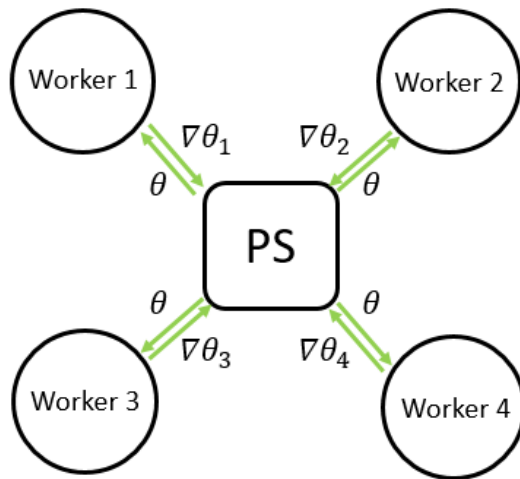
$$\Delta W = uv^T \quad u = \frac{\partial f(Wa_i, b_i)}{\partial (Wa_i)} \quad v = a_i$$

$$\min_Z \frac{1}{N} \sum_{i=1}^N f_i^*(-z_i) + h^*\left(\frac{1}{N} ZA^T\right)$$

$$\Delta W = uv^T \quad u = \Delta z_i \quad v = a_i$$

Sufficient Factor Broadcasting (SFB)

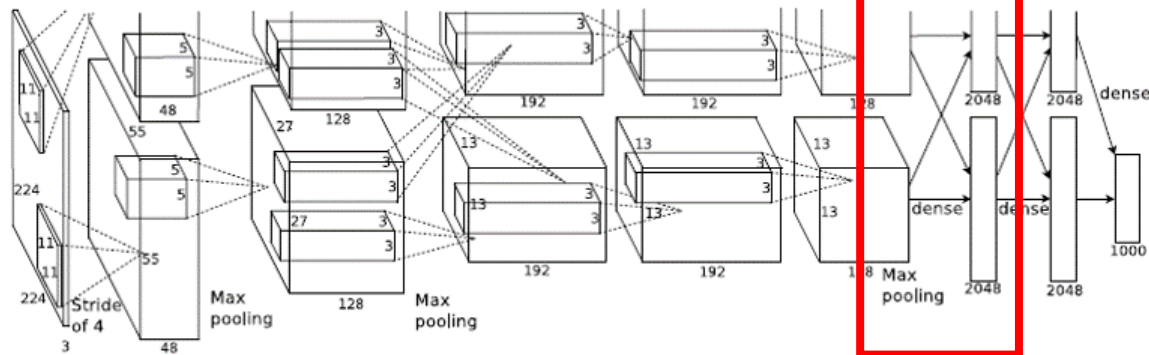
- Idea: Send lightweight SF updates (u, v) , instead of expensive full-matrix ΔW updates!



Hybrid Communication: CNN

- Example: AlexNet CNN model
 - FC6 = 4096 * 30000 matrix (120M parameters)
 - Use SFB to communicate
 - Decouple into two 4096 vectors: u, v
 - Transmit two vectors
 - Reconstruct the gradient matrix

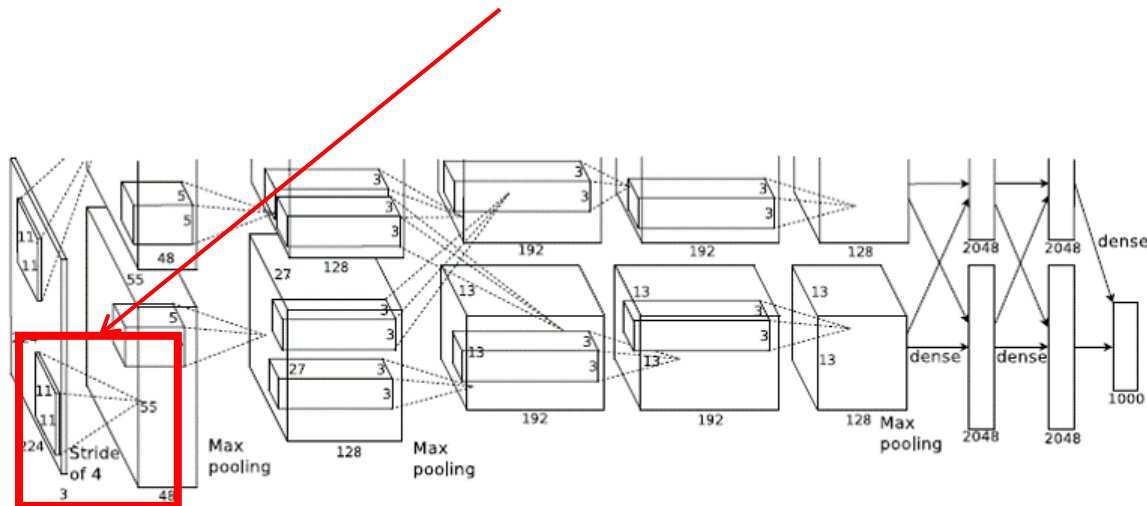
Figure from
Krizhevsky et al. 2012



Hybrid Communication: CNN

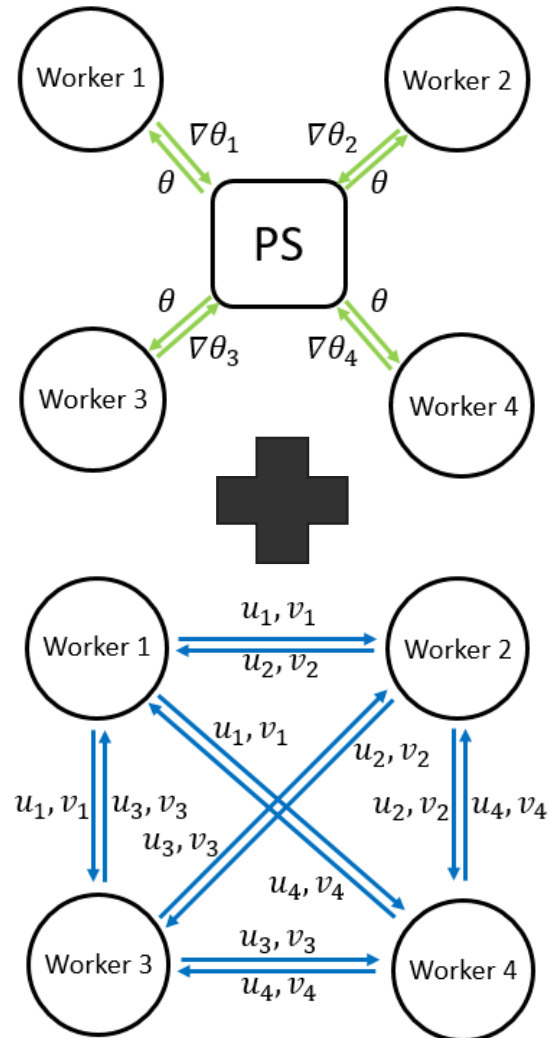
- Example: AlexNet CNN model
 - Convolutional layers = e.g. $11 * 11$ matrix (121 parameters)
 - Use full-matrix updates to communicate
 - SF decomposition does not save much

Figure from
Krizhevsky et al. 2012



Hybrid Communication

- Idea
 - Sync FC layers using SFB
 - Sync Conv layer using PS
- Effectiveness
 - It directly reduces the size of messages in many situations
- Is SFB always optimal?
 - No, its communication load increases quadratically
 - The right strategy: choose PS whenever it results in less communication



Hybrid Communication

- How to choose? Where is the threshold?
- Determine the best strategy depending on
 - Layer type: CONV or FC?
 - Layer size
 - Batch size
 - # of Cluster nodes

Method	Server	Worker	Server & Worker
PS	$2P_1MN/P_2$	$2MN$	$2MN(P_1 + P_2 - 2)/P_2$
SFB	N/A	$2K(P_1 - 1)(M + N)$	N/A
Adam (max)	$P_1MN + P_1K(M + N)$	$K(M + N) + MN$	$(P_1 - 1)(MN + KM + KN)$

Table 1: Estimated communication cost of PS, SFB and Adam for synchronizing the parameters of a $M \times N$ FC layer on a cluster with P_1 workers and P_2 servers, when batchsize is K .

Hybrid Communication

- Hybrid communication algorithm

Determine the best strategy depending on

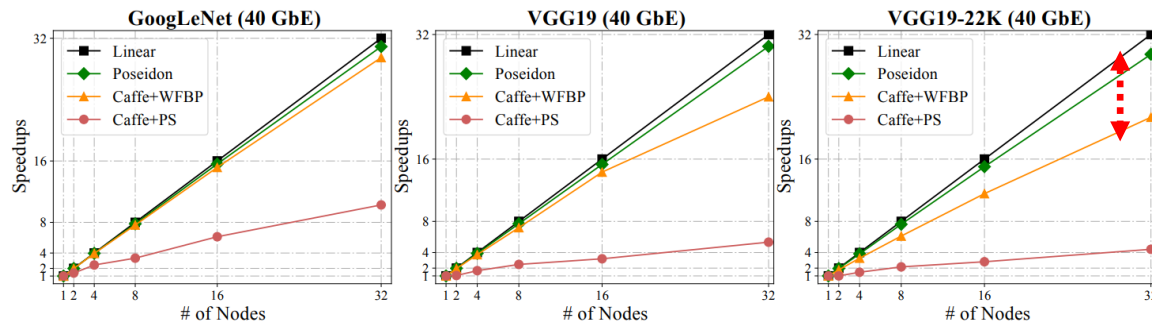
- Layer type: CONV or FC?
- Layer size: M, N
- Batch size: K
- # of Cluster nodes: P_1, P_2

Algorithm 1 Get the best comm method of layer l

```
1: function BESTSCHEME( $l$ )
2:    $layer\_property = \text{Query}(l.name)$ 
3:    $P_1, P_2, K = \text{Query}('n\_worker', 'n\_server', 'batchsize')$ 
4:   if  $layer\_property.type == 'FC'$  then
5:      $M = layer\_property.width$ 
6:      $N = layer\_property.height$ 
7:     if  $2K(P_1 - 1)(M + N) \leq \frac{2MN(P_1 + P_2 - 2)}{P_2}$  then
8:       return 'SFB'
9:     end if
10:  end if
11:  return 'PS'
12: end function
```

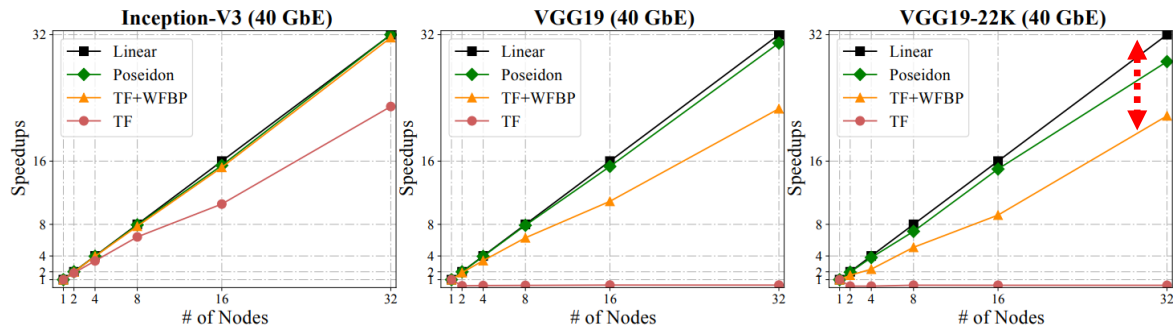
Hybrid Communication

- How does hybrid communication perform?
 - Using Caffe as an engine:



Improve over WFBP

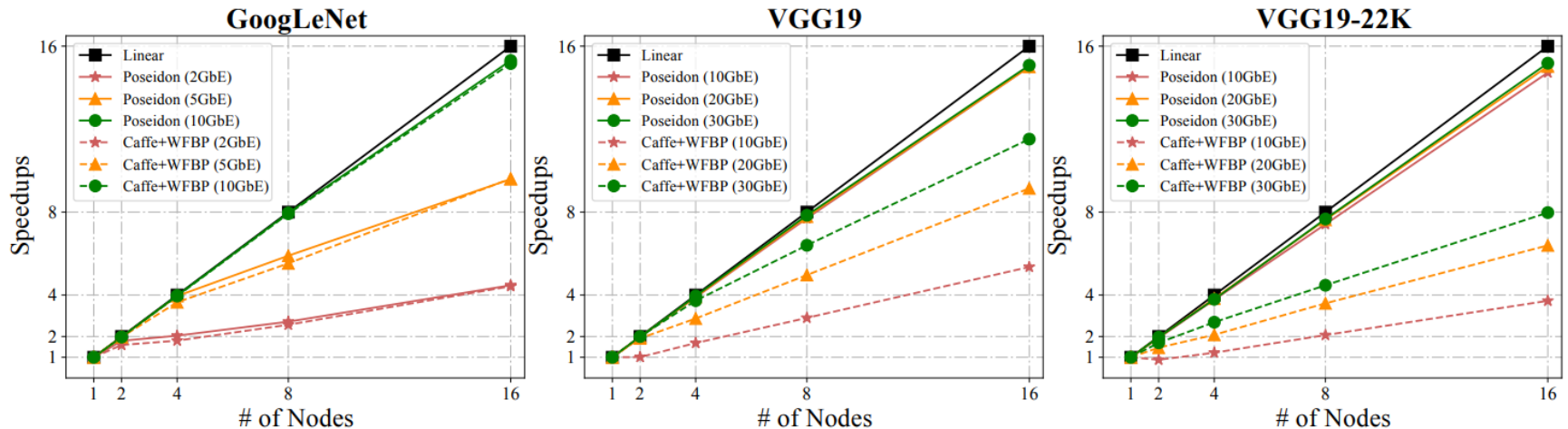
- Using TensorFlow as engine



Improve over WFBP

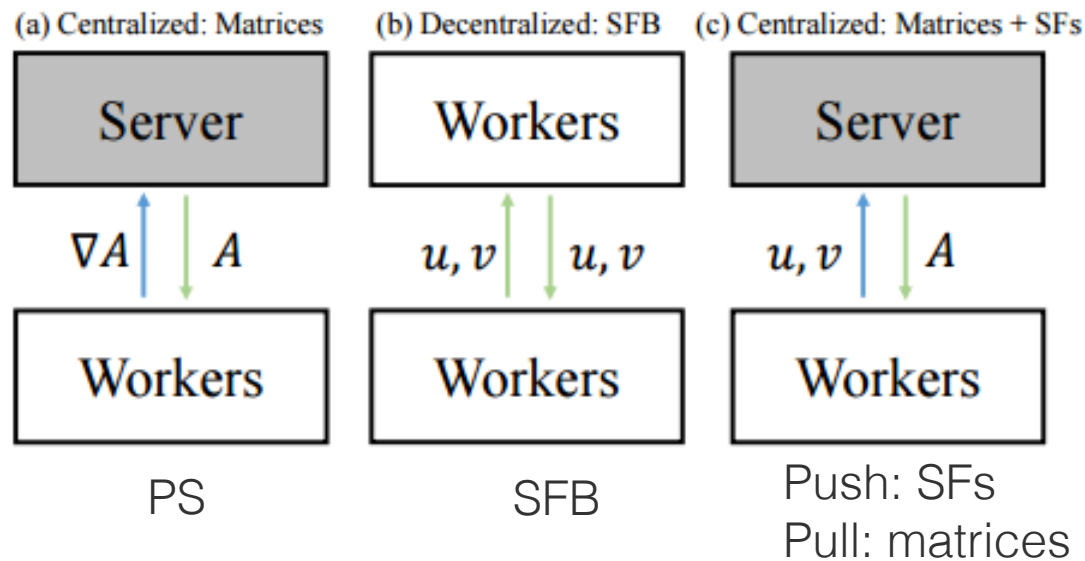
Hybrid Communication

- More importantly, linear scalability on throughput, even with limited bandwidth!
 - Make distributed deep learning affordable



Hybrid Communication

- Discussion: Utilizing SFs is not a new idea
- Microsoft Adam uses the third strategy (c)



Hybrid Communication

- Problem: Adam's strategy leads to communication bottleneck
 - Pushing SFs to server is fine
 - Pulling full matrices back will create a bottleneck on the server node.

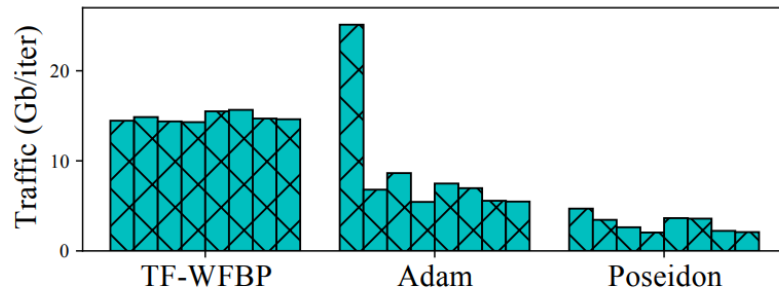
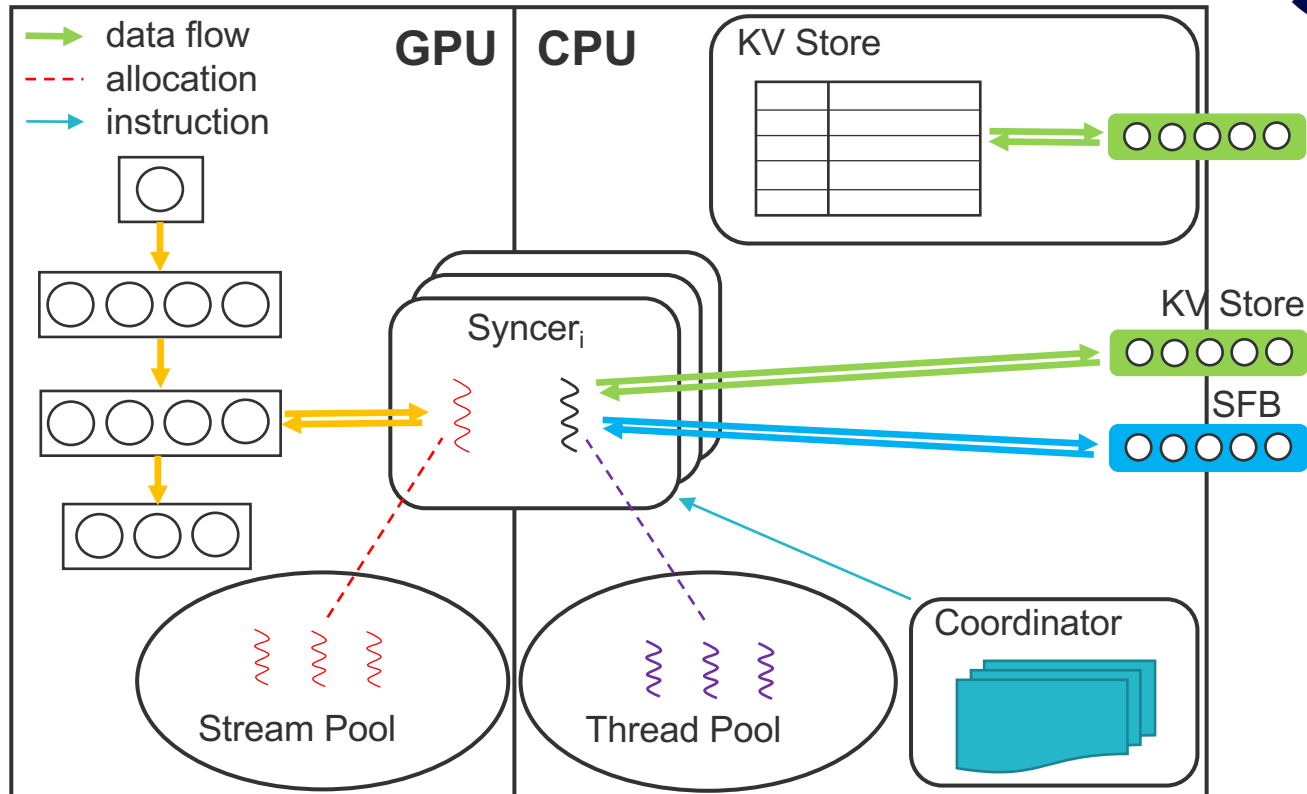


Figure 10: Averaged communication load when training VGG19 using *TF-WFBP*, *Adam* and *Poseidon* with TensorFlow engine. Each bar represents the network traffic on a node.

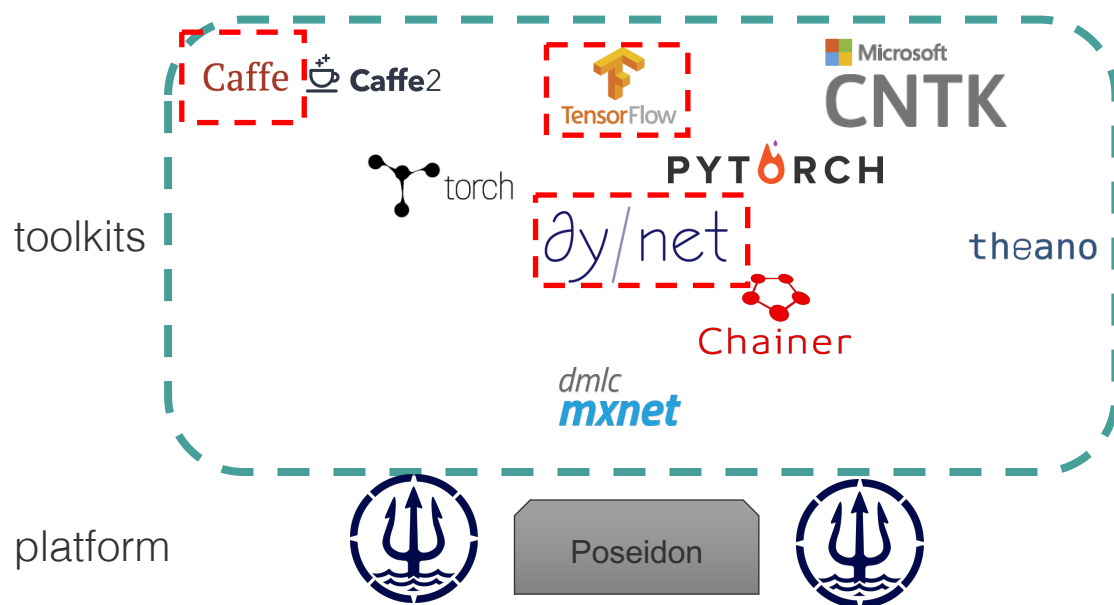
- Hybrid communication yields communication load balancing
 - Which is important to address the problem of burst communication.

Poseidon System Architecture



Poseidon as a Platform

- Poseidon: An efficient communication architecture
 - Efficient distributed platform for amplifying any DL toolkits
 - Preserve the programming interface for any high-level toolkits
 - i.e. distribute the DL program without changing any line of code



Summary: Take-home Messages

- Communication is a bottleneck in distributed DL on GPUs
 - GPUs are too fast
 - Ethernet has limited bandwidth and latency
 - Burst communication
- Poseidon is designed to alleviate this problem
 - WFBP: pipelining the synchronization and computation
 - Hybrid Communication: adaptive protocol to reduce the size of messages
- Results:
 - Linear throughput scalability across different dataset, model sizes, and hardware configuration (Ethernet bandwidth)
 - A Platform to amplify existing DL toolboxes



Carnegie
Mellon
University

Thank You!
Q&A

Backup Slides



WFBP: Distributed Wait-free backpropagation

- Observation: Why DWBP is very effective in DL
 - More statistics of modern CNNs

Params/FLOP distribution of modern CNNs

Parameters	CONV Layers (#/%)	FC Layers (#/%)
AlexNet	2.3M / 3.75	59M / 96.25
VGG-16	7.15M / 5.58	121.1M / 94.42
FLOPs	CONV Layers (#/%)	FC Layers (#/%)
AlexNet	1,352M / 92.0	117M / 8.0
VGG-16	10,937M / 91.3	121.1M / 8.7

- 90% computation happens at bottom layers
- 90% communication happens at top layers
- WFBP overlaps 90% communication with 90% computation

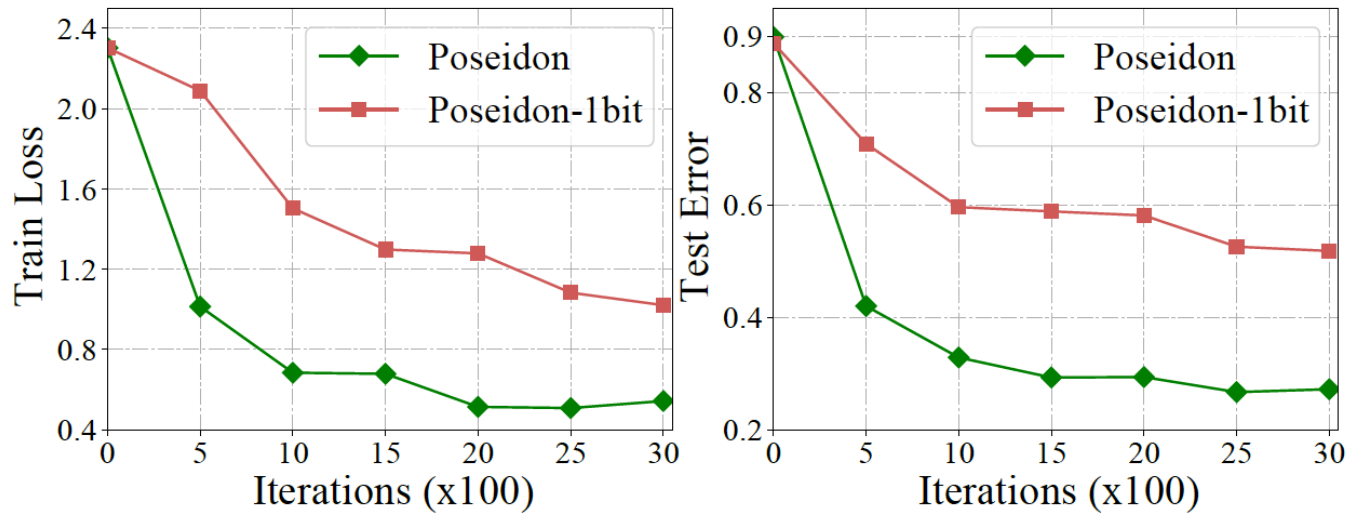
Poseidon API

- KV Store, Syncer and Coordinator
- Standard APIs similar to parameter server
 - *Push/Pull* API for parameter synchronization
 - *BestScheme* method to return the best communication method

Method	Owner	Arguments	Description
BestScheme	Coordinator	A layer name or index	Get the best communication scheme of a layer
Query	Coordinator	A list of property names	Query information from coordinators' information book
Send	Syncer	None	Send out the parameter updates of the corresponding layer
Receive	Syncer	None	Receive parameter updates from either parameter server or peer workers
Move	Syncer	A GPU stream and an indicator of move direction	Move contents between GPU and CPU, do transformations and application of updates if needed
Send	KV store	updated parameters	Send out the updated parameters
Receive	KV store	parameter buffer of KV stores	Receive gradient updates from workers

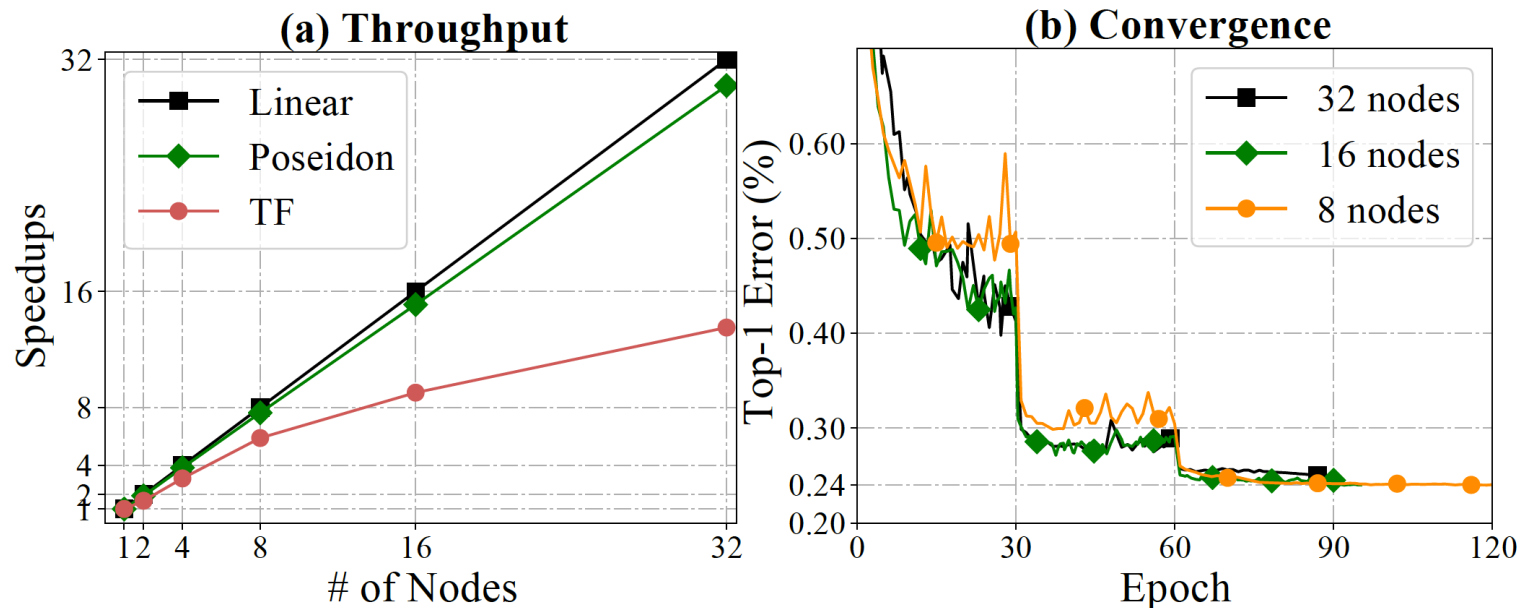
Comparison: 1-bit Quantization

- Microsoft CNTK has 1-bit quantization technique to lossy parameter compression
 - While our experiments reveal that it might not work well in terms of statistical convergence in some domains
- Good news! Some recent works report exciting results on this line



Results: Statistical Convergence

- Poseidon adopts fully synchronous consistency model
 - Distributed training = larger batch size
 - Turning parameters in distributed settings is an open problem
- Linear convergence speedup on ResNet-152 [He et al. 2015]



Discussion: Synchronous vs. Asynchronous

- Empirical results: synchronous updates yield the faster per-iteration convergence in training modern deep neural networks
 - GeePS, Cui et al., Eurosys'16
 - TensorFlow, Abadi et al., OSDI'16
 - Poseidon, Zhang et al., ATC'16, '17
- Stragglers
 - Yes, stragglers exist. However, we'd still prefer synchronous training because the cost is less than having asynchrony
- Does/will asynchronous training work?
 - Yes, but it is domain-specific, e.g. in some speech/NLP application, there are some reported results that asynchronous training yields same-quality results.
 - There are more and more ongoing research towards this direction in both machine learning and systems.