

T. BEN-NUN, T. HOEFLER

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

<https://www.arxiv.org/abs/1802.09941>

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

TAL BEN-NUN* and TORSTEN HOEFLER, ETH Zurich

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs; synchronous and asynchronous stochastic gradient descent; distributed system architectures; communication schemes; and performance modeling. Based on these approaches, we extrapolate potential directions for parallelism in deep learning.

CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Computing methodologies** → **Neural networks**; **Distributed computing methodologies**; **Parallel computing methodologies**; *Machine learning*;

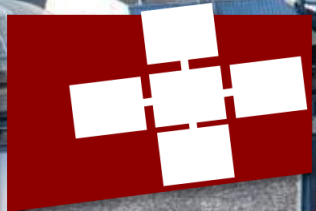
Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms

ACM Reference format:

Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. 60 pages.

1 INTRODUCTION

Machine Learning, and in particular Deep Learning [LeCun et al. 2015], is a field that is rapidly taking over a variety of aspects in our daily lives. In the core of deep learning lies the Deep Neural Network (DNN), a construct inspired by the interconnected nature of the human brain. Trained properly, the expressiveness of DNNs provides accurate solutions for problems previously thought to be unsolvable, simply by observing large amounts of data. Deep learning has been successfully implemented for a plethora of subjects, ranging from image classification [Huang et al. 2017], through speech recognition [Amodei et al. 2016] and medical diagnosis [Cireşan et al. 2013], to autonomous driving [Bojarski et al. 2016] and defeating human players in complex games [Silver et al. 2017] (see Fig. 1 for more examples).



What is Deep Learning good for?

Digit Recognition

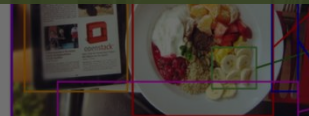
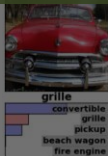
Object Classification
Segmentation

Image Captioning

Gameplay AI
Translation

Neural Computers

A very promising area of research!



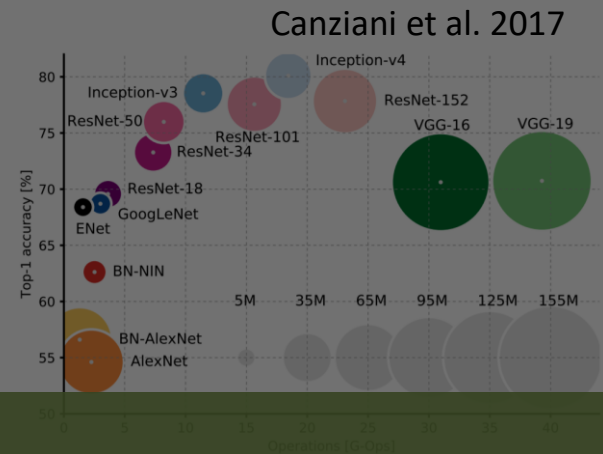
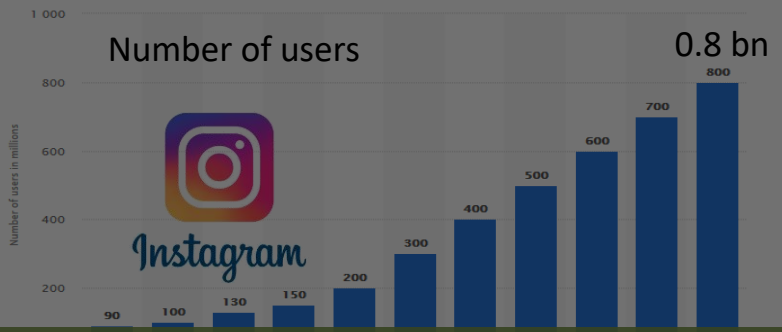
23 papers per day!

Year	2012	2013	2014	2015	2016	2017
cs.AI	1,081	1,765	1,022	1,105	1,929	2,790
cs.CV	577	852	1,349	2,261	3,627	5,693

number of papers per year

1989 2012 2013 2014 2016 2017

How does Deep Learning work?

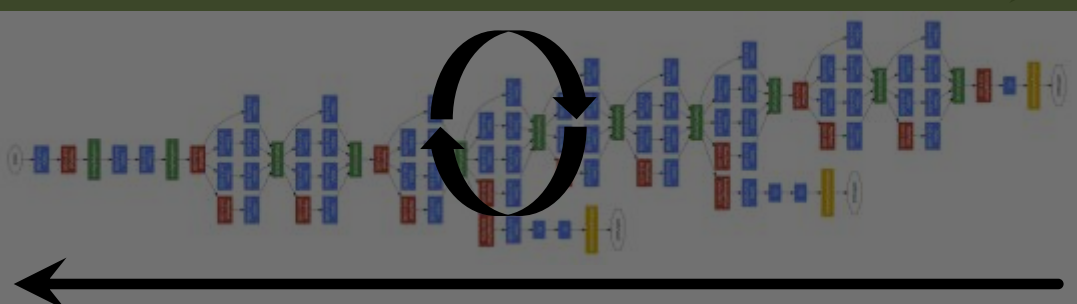
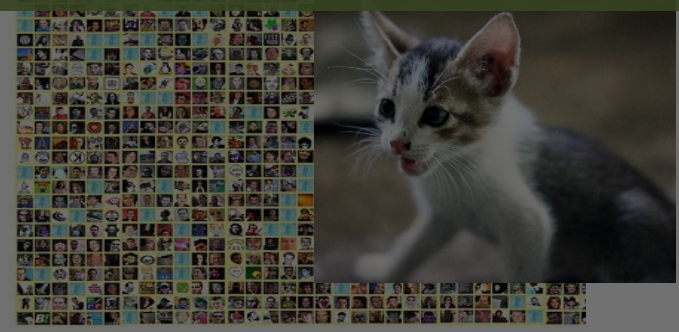


What is Deep Learning used for?

- Digit Recognition
- Object Classification Segmentation
- Image Captioning
- Gameplay AI Translation
- Neural Computers Routing

Timeline: 1989, 2012, 2013, 2014, 2016, 2017

Deep Learning is Supercomputing!

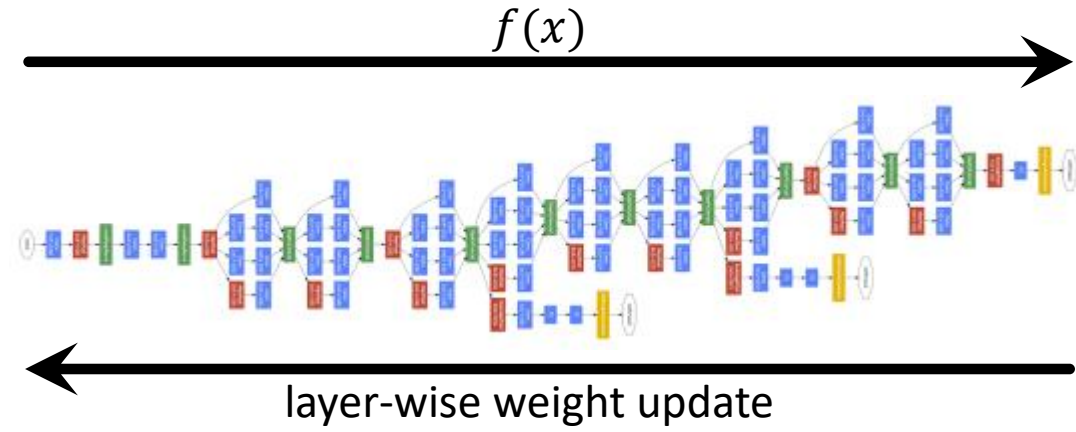


Cat	0.54	Cat	0.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.04	Horse	0.00
Bicycle	0.33	Bicycle	0.00
Truck	0.02	Truck	0.00

layer-wise weight update

- ImageNet (1k): 180 GB
- ImageNet (22k): A few TB
- Industry: Much larger
- 100-200 layers deep
- ~100M-2B parameters
- 0.1-8 GiB parameter storage
- 10-22k labels
- growing (e.g., face recognition)
- weeks to train

A brief theory of supervised deep learning



label domain Y	true label $l(x)$		
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.04	Horse	0.00
Bicycle	0.33	Bicycle	0.00
Truck	0.02	Bicycle	0.00
		Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

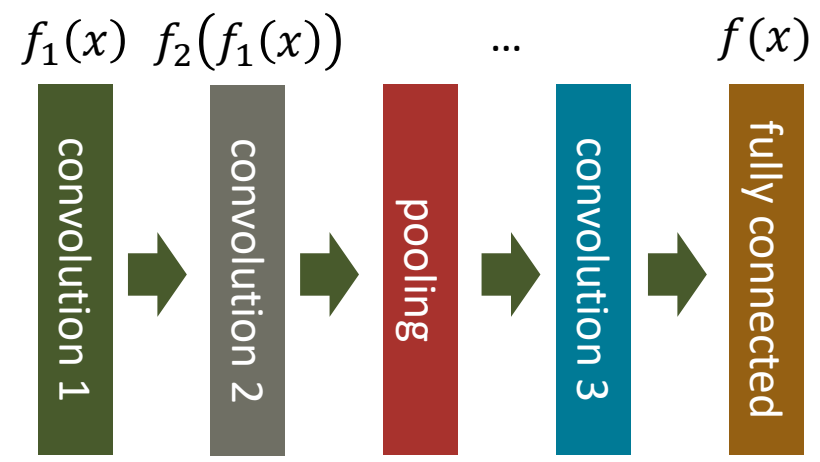
label domain Y true label $l(x)$

$$f(x): X \rightarrow Y$$

$$f(x) = f_n \left(f_{n-1} \left(f_{n-2} \left(\dots f_1(x) \dots \right) \right) \right)$$

network structure (fixed) weights w (learned)

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}} [\ell(w, x)]$$



$$\ell_{sq}(w, x) = (f(x) - l(x))^2$$

$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = - \sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

Stochastic Gradient Descent

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}} [\ell(w, x)]$$

- 1:
- 2:
- 3:
- 4:
- 5:
- 6:
- 7:
- 8:
- 9:
- 10:
- 11:
- 12:

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

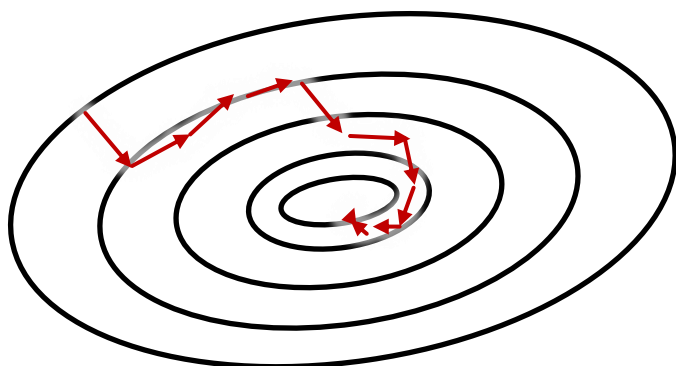
pooling

...

convolution 3

$f(x)$ fully connected

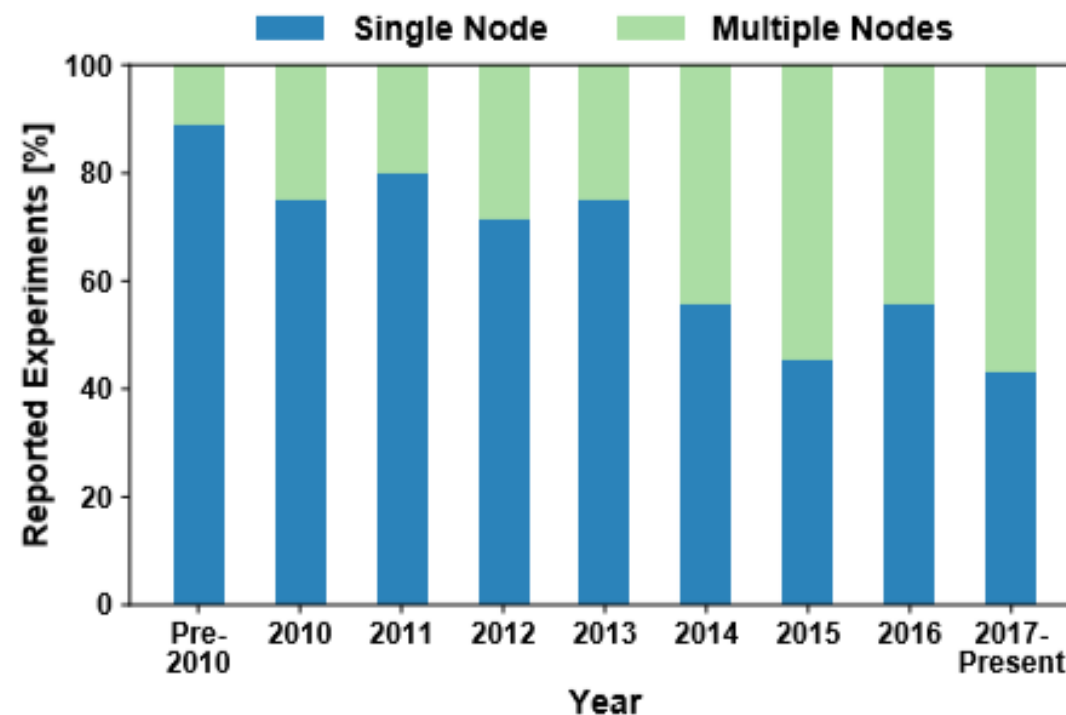
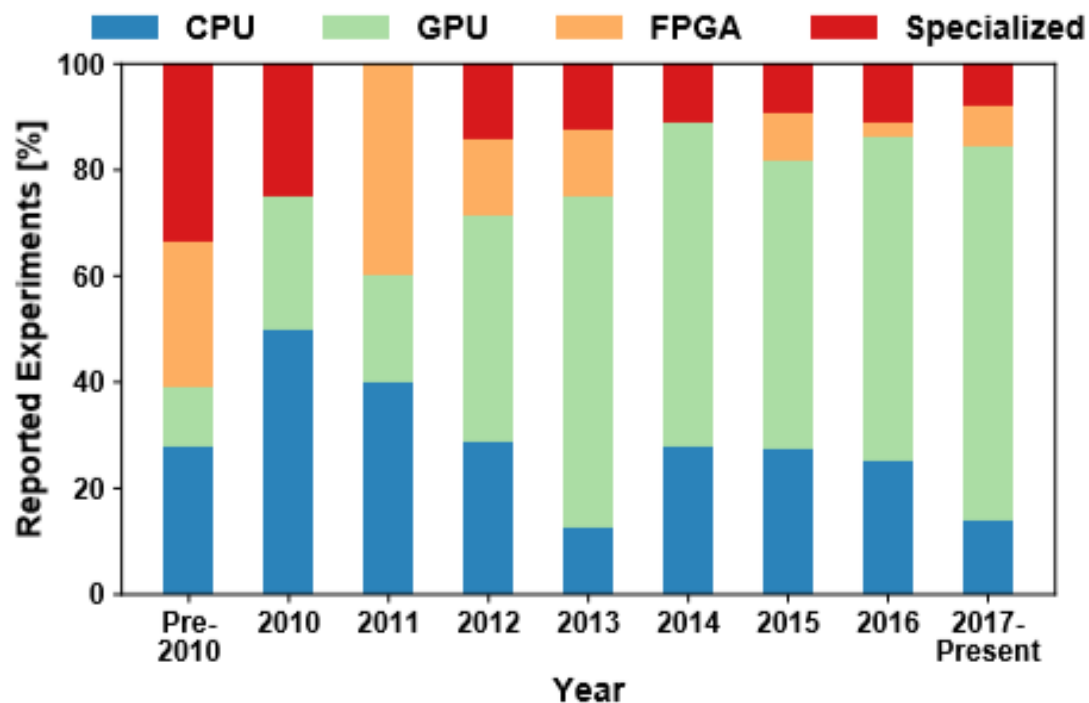
- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla \ell(w^{(t)}, z) = w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$
Nesterov Momentum [Nesterov 1983]	$w^{(t+1)} = w^{(t)} + v_t; \quad v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla \ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [Duchi et al. 2011]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t} + \epsilon}}; \quad A_{i,t} = \sum_{\tau=0}^t (\nabla w_i^{(\tau)})^2$
RMSProp [Hinton 2012]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t} + \epsilon}}; \quad A'_{i,t} = \beta \cdot A'_{i,t-1} + (1 - \beta) (\nabla w_i^{(t)})^2$
Adam [Kingma and Ba 2015]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)} + \epsilon}}; \quad M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1 - \beta_m) (\nabla w_i^{(t)})^m}{1 - \beta_m^t}$

Trends in deep learning: hardware and multi-node

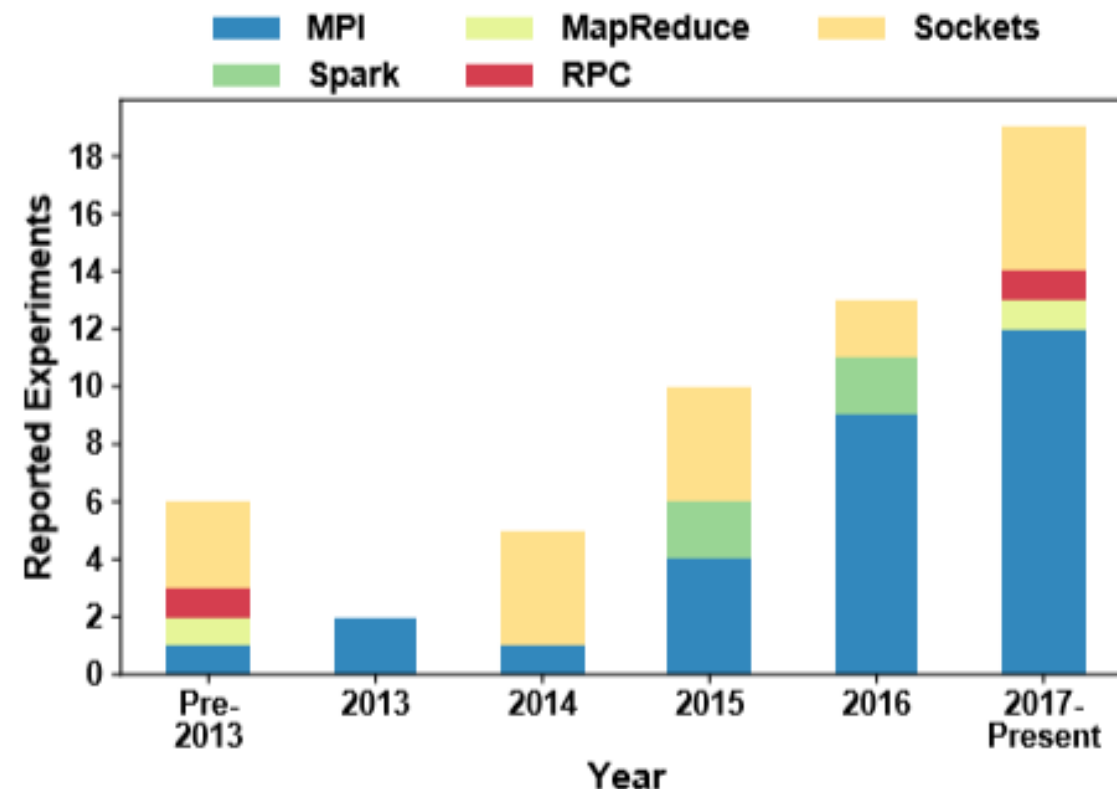
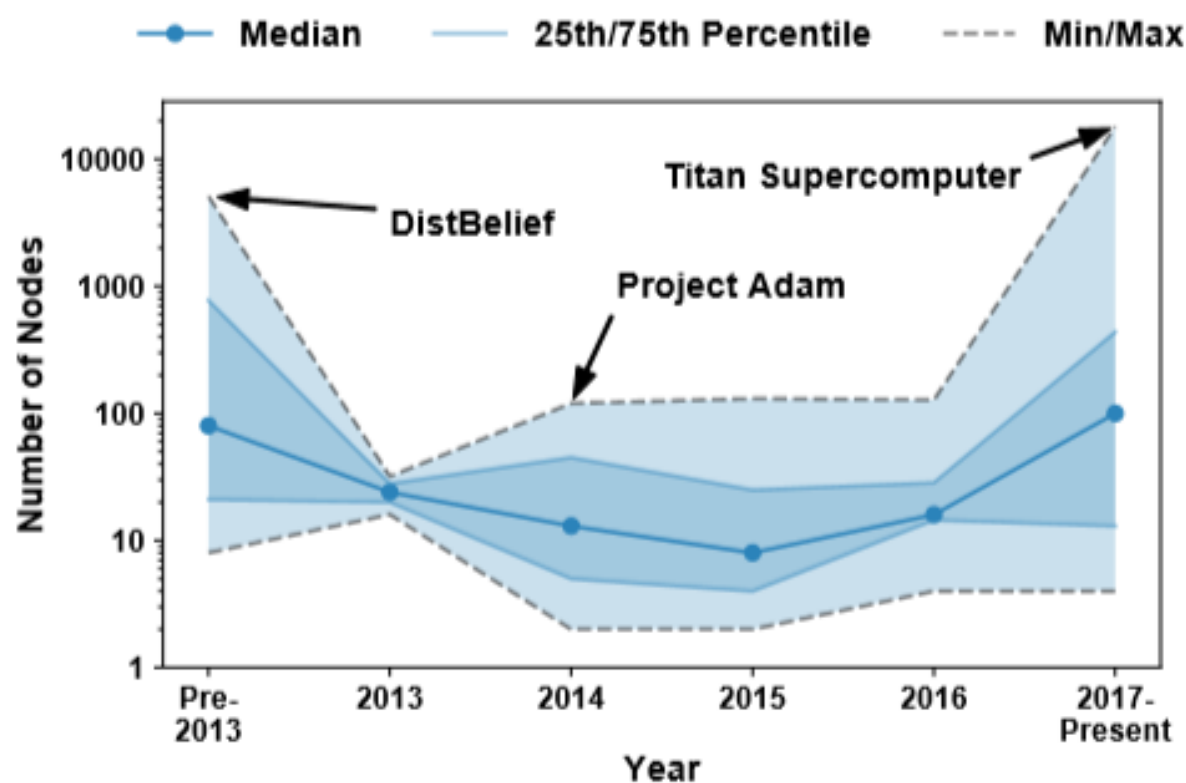
The field is moving fast – trying everything imaginable – survey results from 227 papers in the area of parallel deep learning



Deep Learning is largely on distributed memory today!

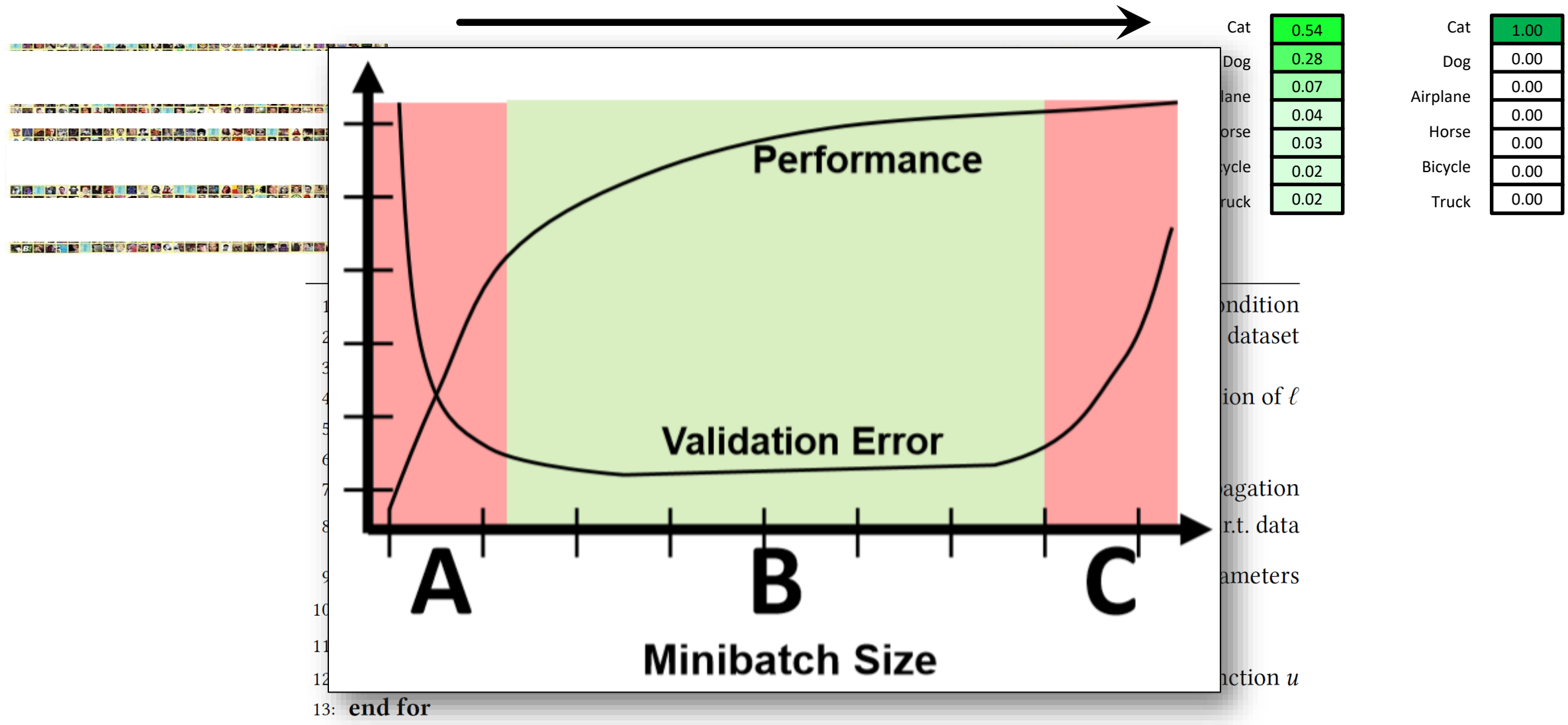
Trends in **distributed** deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 227 papers in the area of parallel deep learning



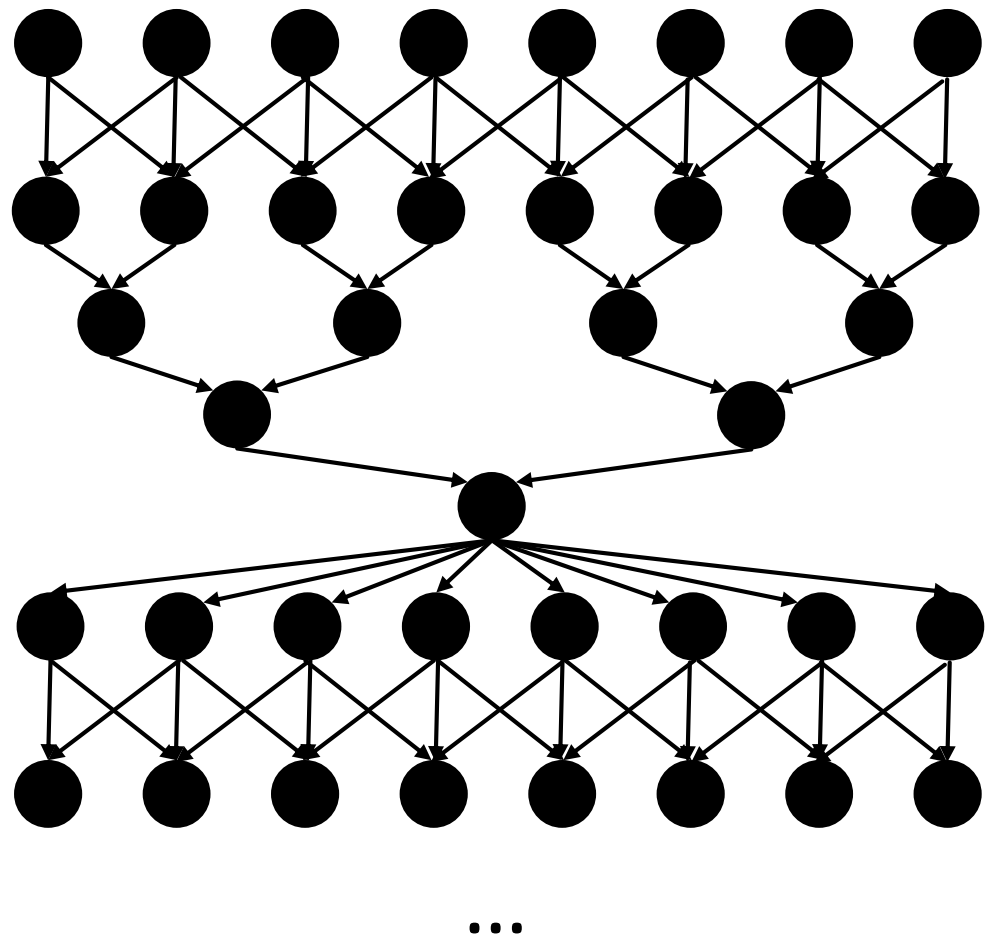
Deep Learning research is converging to MPI!

Minibatch Stochastic Gradient Descent (SGD)



13: end for

A primer of relevant parallelism



Work $W = 39$

Depth $D = 7$

Average parallelism = $\frac{W}{D}$

and communication theory

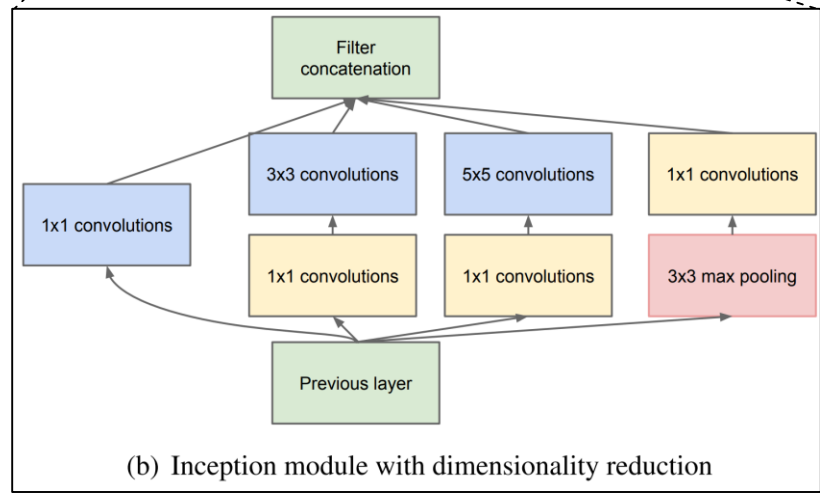
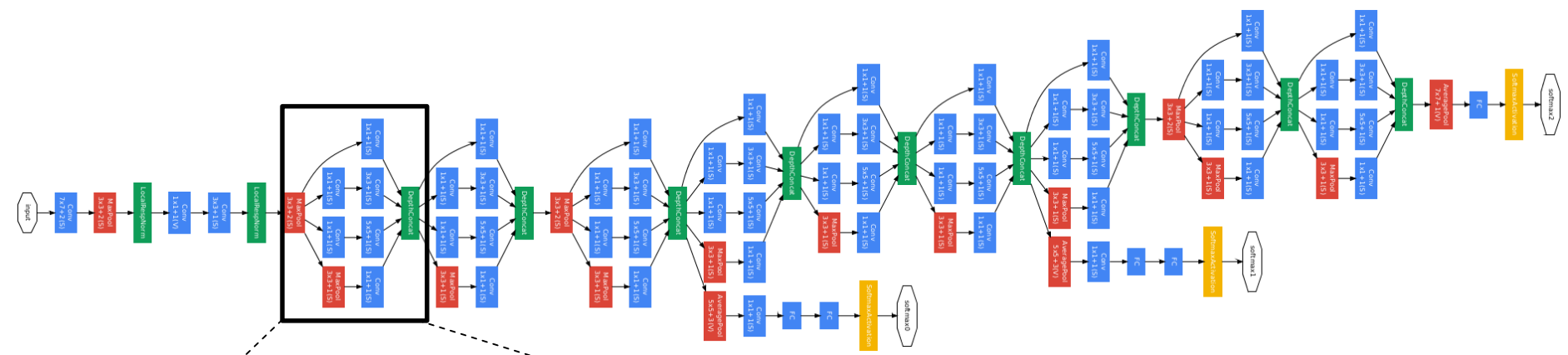
Parallel Reductions for Parameter Updates

$$y = x_1 \oplus x_2 \oplus x_3 \cdots \oplus x_{n-1} \oplus x_n$$

Small vectors		Large vectors	
<p>Tree</p>	<p>Butterfly</p>	<p>Pipeline</p>	<p>RedScat+Gat</p>
$T = 2L \log_2 P + 2\gamma m G \log_2 P$	$T = L \log_2 P + \gamma m G \log_2 P$	$T = 2L(P - 1) + 2\gamma m G(P - 1)/P$	$T = 2L \log_2 P + 2\gamma m G(P - 1)/P$

Lower bound: $T \geq L \log_2 P + 2\gamma m G(P - 1)/P$

GoogLeNet in more detail



- ~6.8M parameters
- 22 layers deep

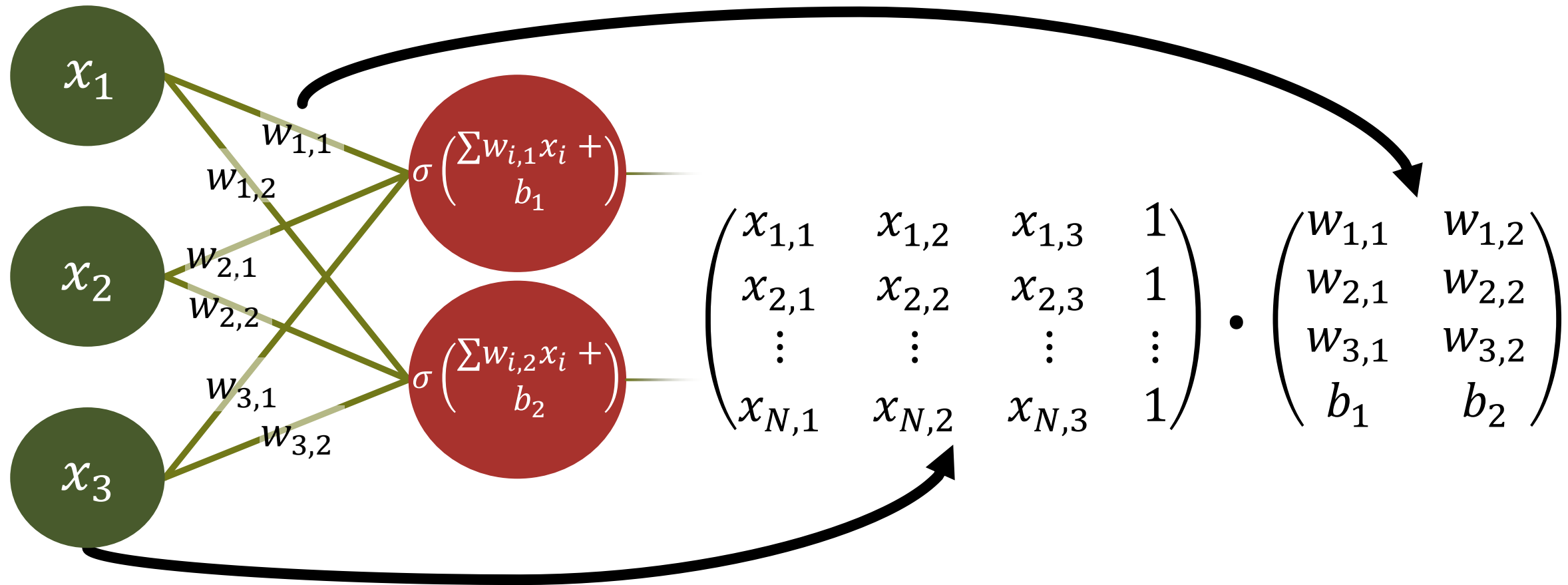
Parallelism in the different layer types

Layer Type	Eval.	Work (W)	Depth (D)
------------	-------	----------	-----------

W is linear and D logarithmic – large average parallelism

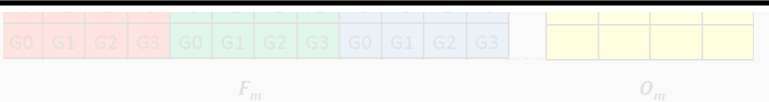
Computing fully connected layers

$f_l(x)$	$O(C_{out} \cdot C_{in} \cdot N)$	$O(\log C_{in})$
∇w	$O(C_{in} \cdot N \cdot C_{out})$	$O(\log N)$
∇o_l	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$



Computing convolutional layers

Direct		Indirect	
Method	Work (W)	FFT	Winograd
Method	Work (W)	Depth (D)	Depth (D)
Direct	$N \cdot C_{out} \cdot H' \cdot W' \cdot C_{in} \cdot K_y \cdot K_x$	$\lceil \log_2 C_{in} \rceil + \lceil \log_2 K_y \rceil + \lceil \log_2 K_x \rceil$	$\lceil \log_2 C_{in} \rceil + \lceil \log_2 K_y \rceil + \lceil \log_2 K_x \rceil$
im2col	$N \cdot C_{out} \cdot H' \cdot W' \cdot C_{in} \cdot K_y \cdot K_x$	$\lceil \log_2 C_{in} \rceil + \lceil \log_2 K_y \rceil + \lceil \log_2 K_x \rceil$	$\lceil \log_2 C_{in} \rceil + \lceil \log_2 K_y \rceil + \lceil \log_2 K_x \rceil$
FFT	$c \cdot HW \log_2(HW) \cdot (C_{out} \cdot C_{in} + N \cdot C_{in} + N \cdot C_{out}) + HWN \cdot C_{in} \cdot C_{out}$	$2 \lceil \log_2 HW \rceil + \lceil \log_2 C_{in} \rceil$	$2 \lceil \log_2 HW \rceil + \lceil \log_2 C_{in} \rceil$
Winograd ($m \times m$ tiles, $r \times r$ kernels)	$\alpha(r^2 + \alpha r + 2\alpha^2 + \alpha m + m^2) + C_{out} \cdot C_{in} \cdot P$ ($\alpha \equiv m - r + 1, \quad P \equiv N \cdot \lceil H/m \rceil \cdot \lceil W/m \rceil$)	$2 \lceil \log_2 r \rceil + 4 \lceil \log_2 \alpha \rceil + \lceil \log_2 C_{in} \rceil$	$2 \lceil \log_2 r \rceil + 4 \lceil \log_2 \alpha \rceil + \lceil \log_2 C_{in} \rceil$

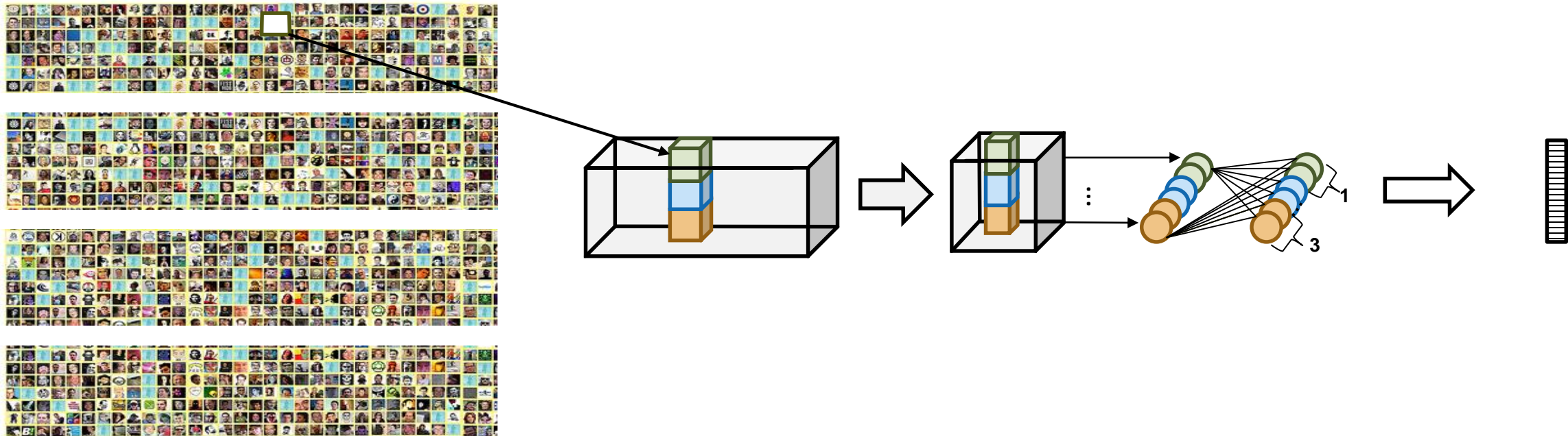


S. Chetlur et al.: cuDNN: Efficient Primitives for Deep Learning, arXiv 2014



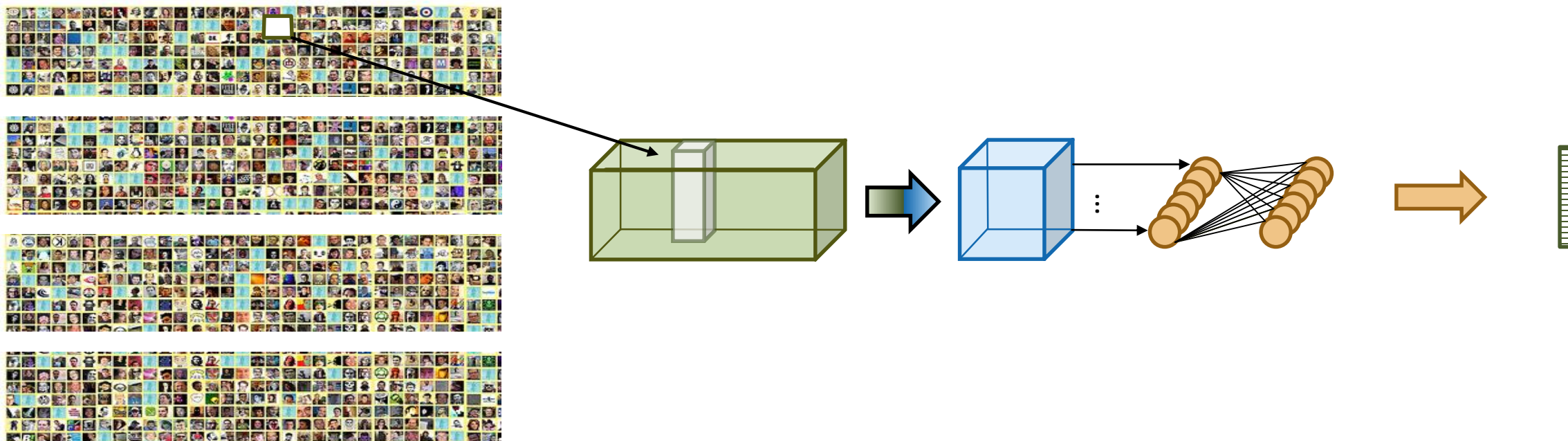
X. Liu et al.: Efficient Sparse-Winograd Convolutional Neural Networks, ICLR'17 Workshop

Model parallelism



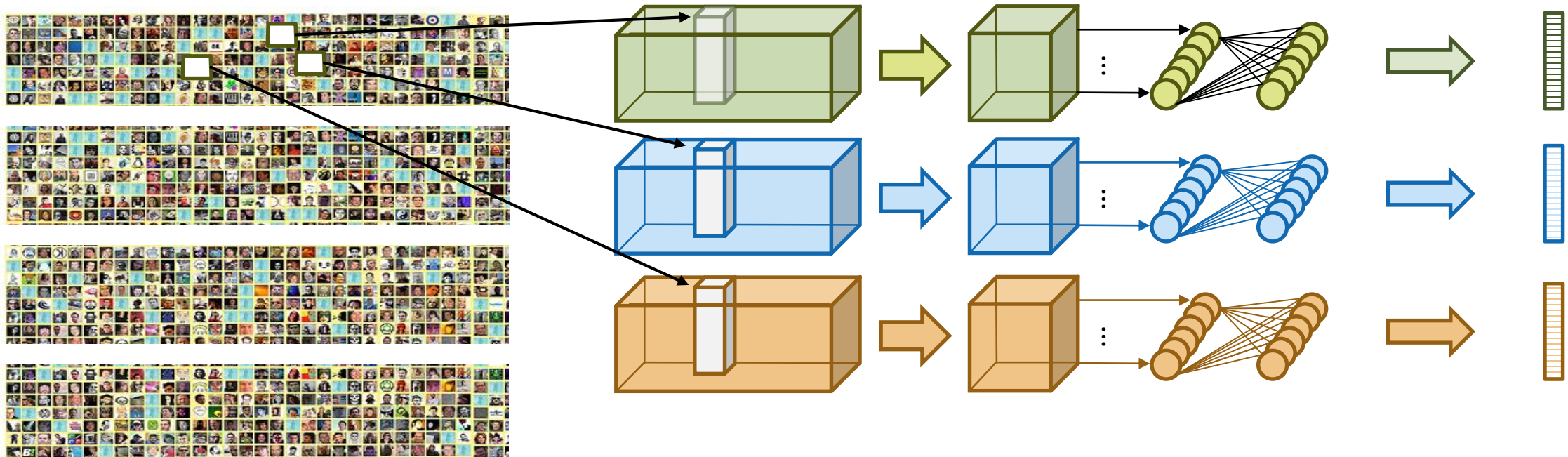
- Parameters can be distributed across processors
- **Mini-batch has to be copied to all processors**
- **Backpropagation requires all-to-all communication every layer**

Pipeline parallelism



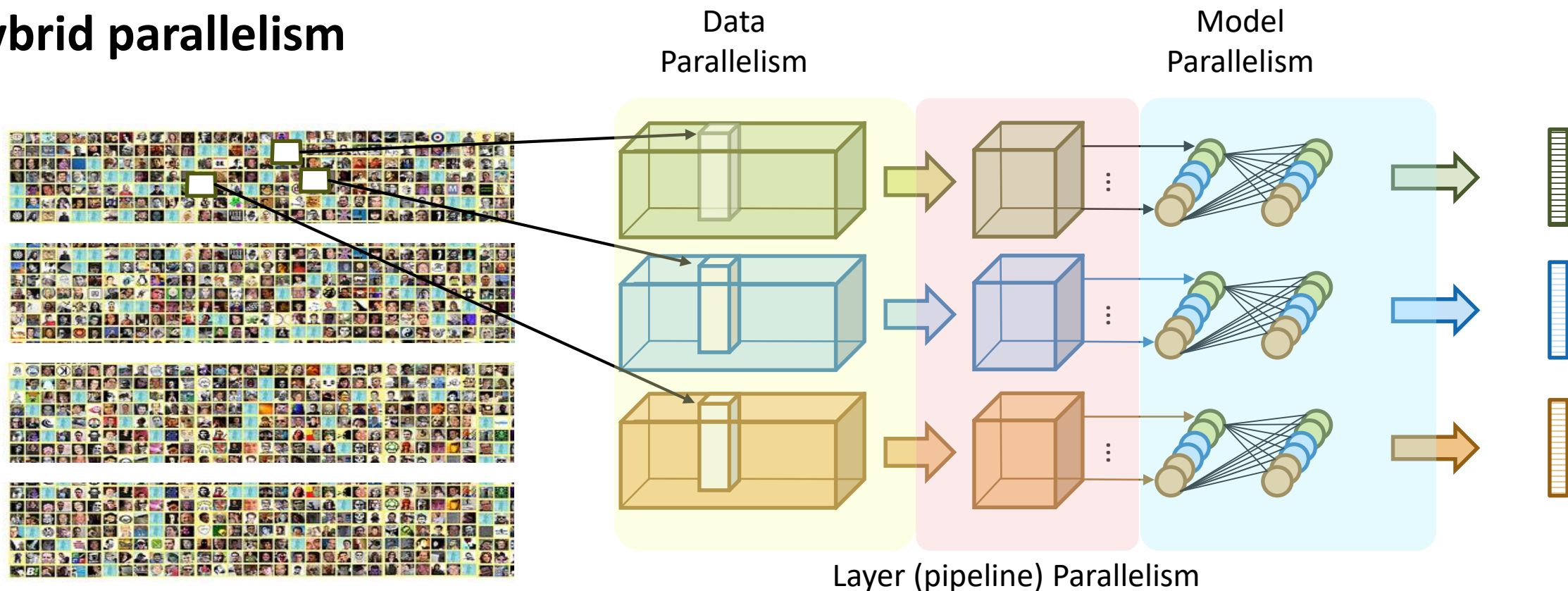
- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- **Mini-batch has to be copied through all processors**

Data parallelism



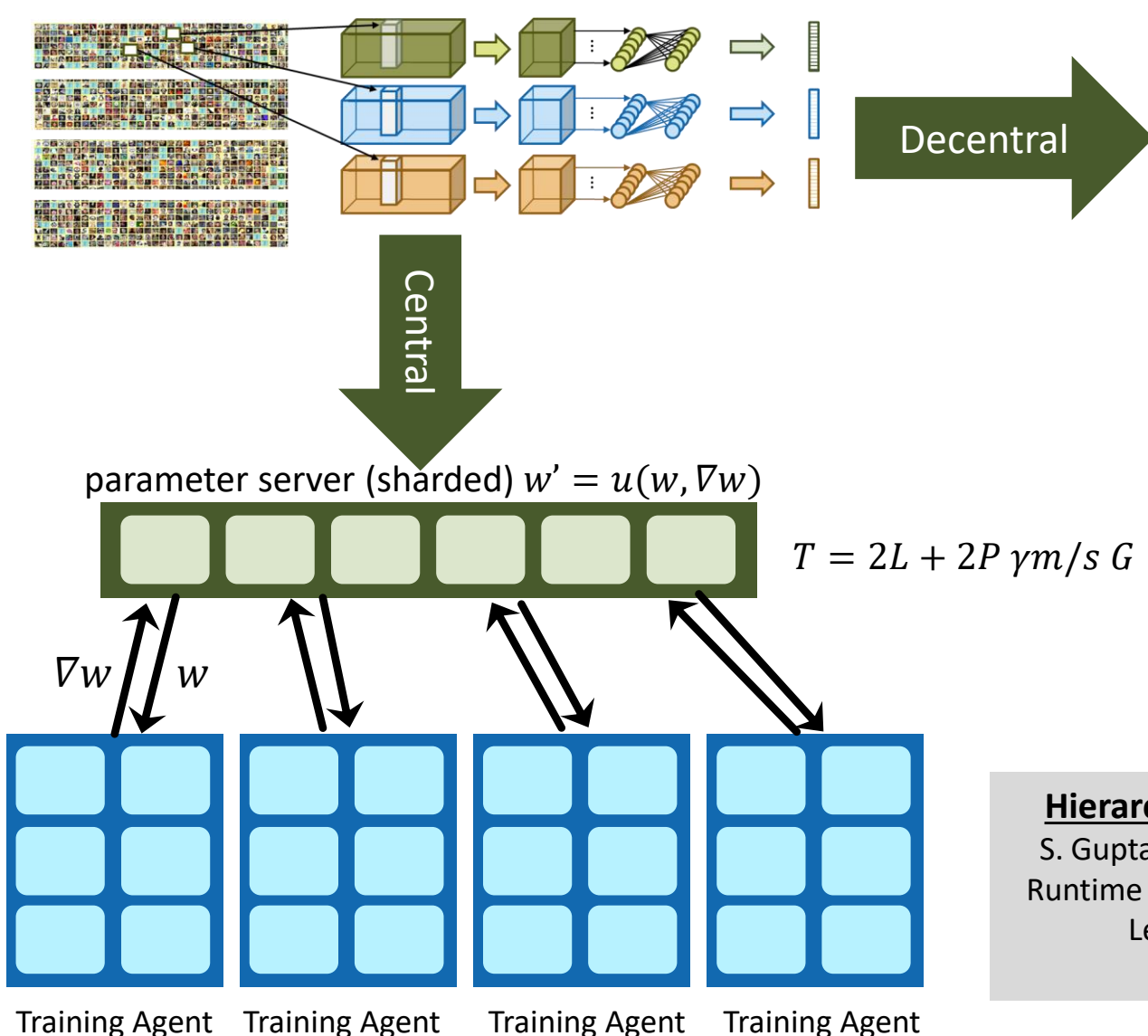
- Simple and efficient solution, easy to implement
- **Duplicate parameters at all processors**

Hybrid parallelism

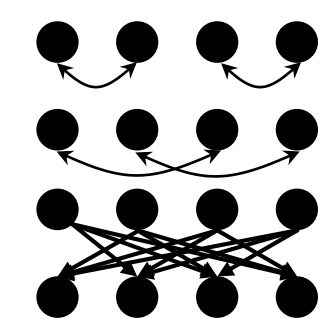


- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
 - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

Updating parameters in **distributed** data parallelism



- Collective operations
- Topologies
- Neighborhood collectives
- RMA?



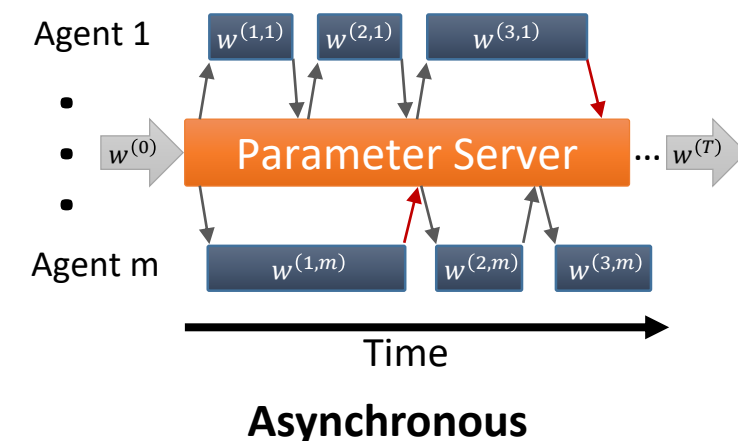
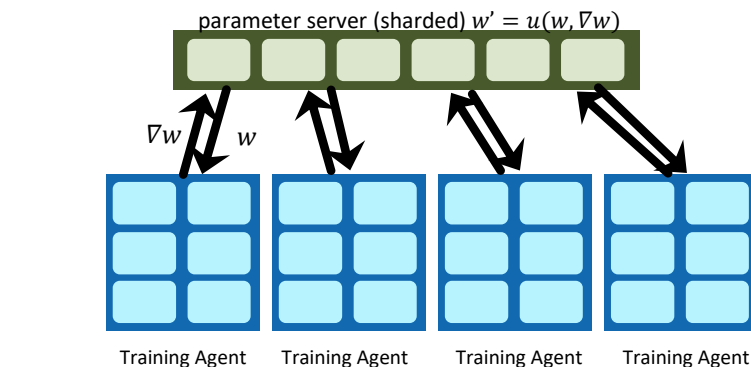
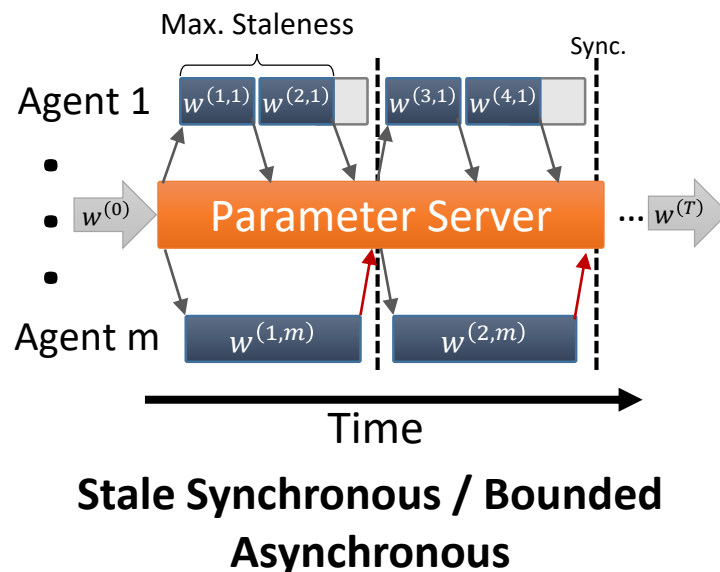
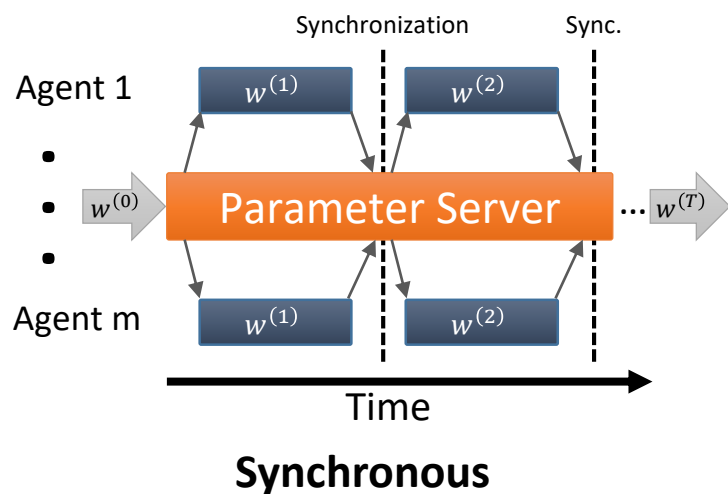
$$T = 2L \log_2 P + 2\gamma m G (P - 1) / P$$

Hierarchical Parameter Server
S. Gupta et al.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. ICDM'16

Adaptive Minibatch Size
S. L. Smith et al.: Don't Decay the Learning Rate, Increase the Batch Size, arXiv 2017

Parameter (and Model) consistency - centralized

- Parameter exchange frequency can be controlled, while still attaining convergence:



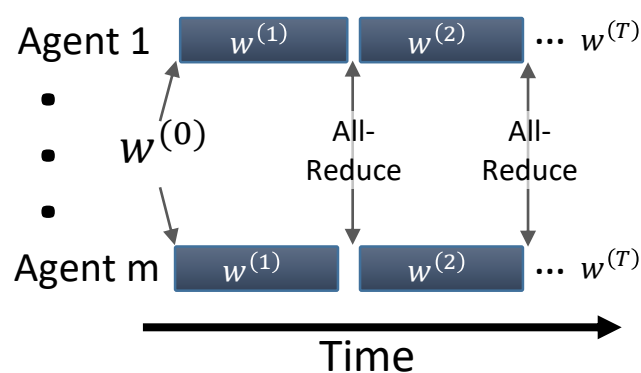
- Started with Hogwild! [Niu et al. 2011] – shared memory, by chance
- DistBelief [Dean et al. 2012] moved the idea to distributed
- Trades off “statistical performance” for “hardware performance”

Parameter (and Model) consistency - decentralized

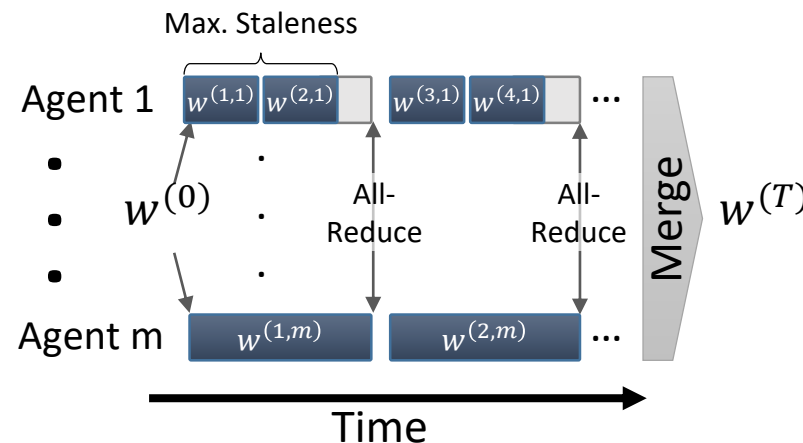
- Parameter exchange frequency can be controlled, while still attaining convergence:



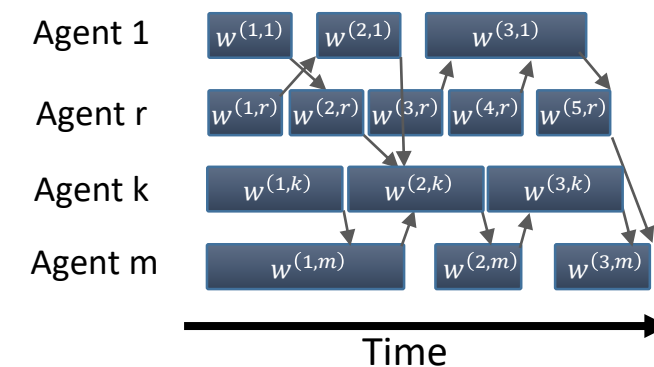
Training Agent Training Agent Training Agent Training Agent



Synchronous



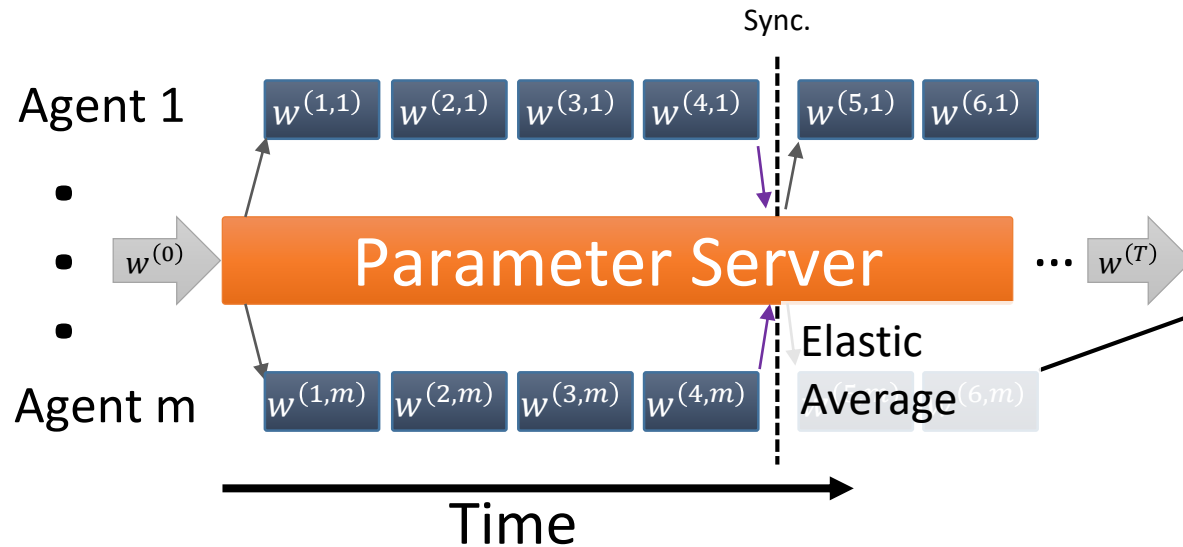
Stale Synchronous / Bounded Asynchronous



Asynchronous

- May also consider limited/slower distribution – gossip [Jin et al. 2016]

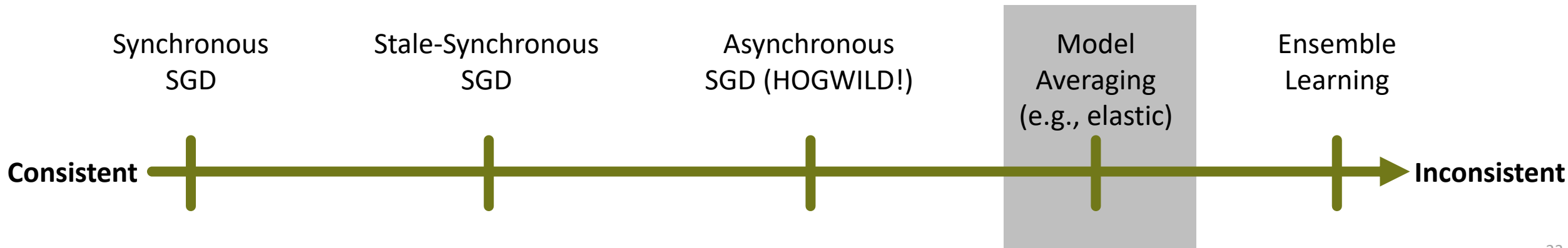
Parameter consistency in deep learning



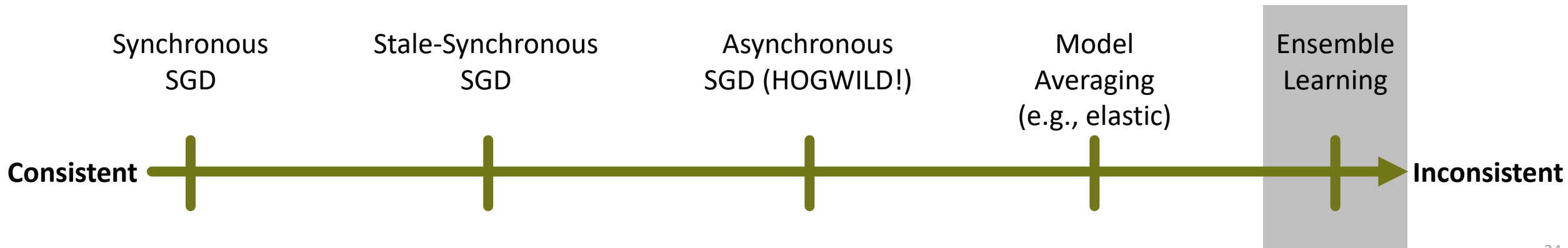
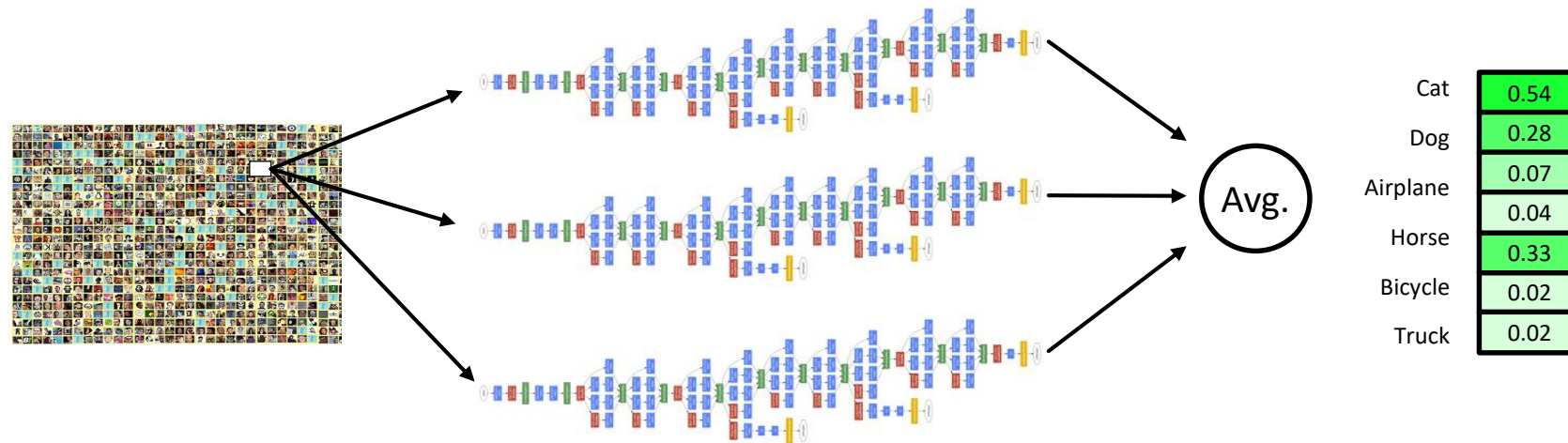
Using physical forces between different versions of w :

$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha (w^{(t,i)} - \tilde{w}_t)$$

$$\tilde{w}_{t+1} = (1 - \beta) \tilde{w}_t + \frac{\beta}{m} \sum_{i=1}^m w^{(t,i)}$$

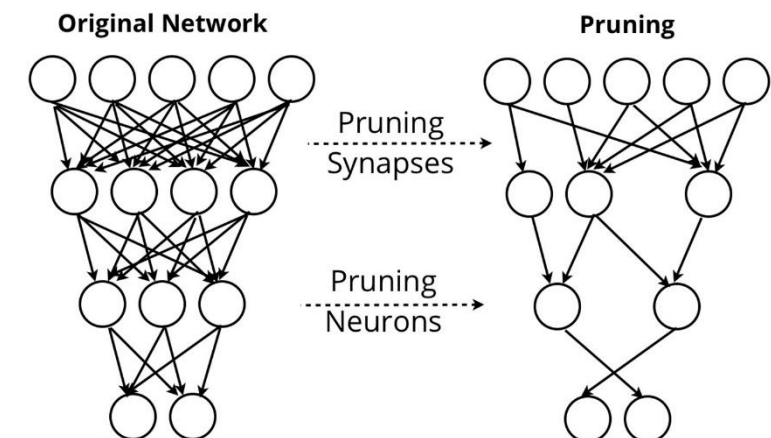
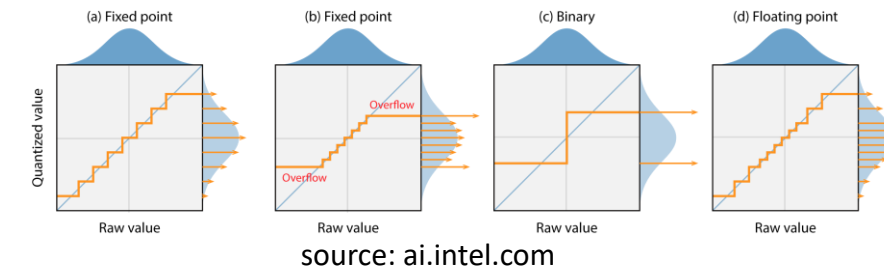
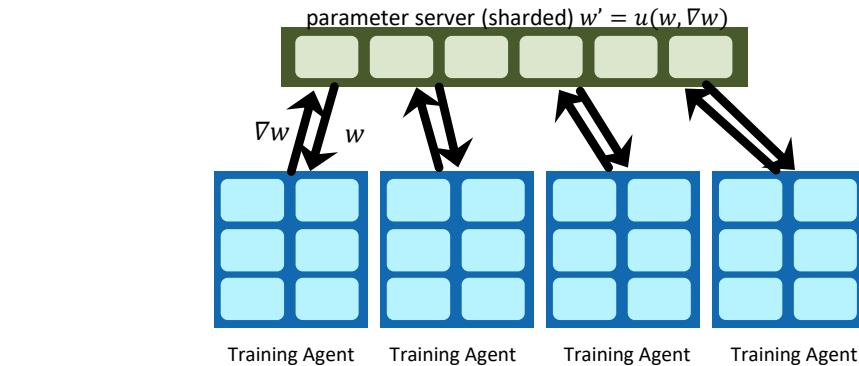


Parameter consistency in deep learning



Communication optimizations

- **Different options how to optimize updates**
 - Send ∇w , receive w
 - Send FC factors (o_{l-1}, o_l) , compute ∇w on parameter server
Broadcast factors to not receive full w
 - Use lossy compression when sending, accumulate error locally!
- **Quantization**
 - Quantize weight updates and potentially weights
 - Main trick is stochastic rounding [1] – expectation is more accurate
Enables low precision (half, quarter) to become standard
 - TernGrad - ternary weights [2], 1-bit SGD [3], ...
- **Sparsification**
 - Do not send small weight updates **or** only send top-k [4]
Accumulate them locally



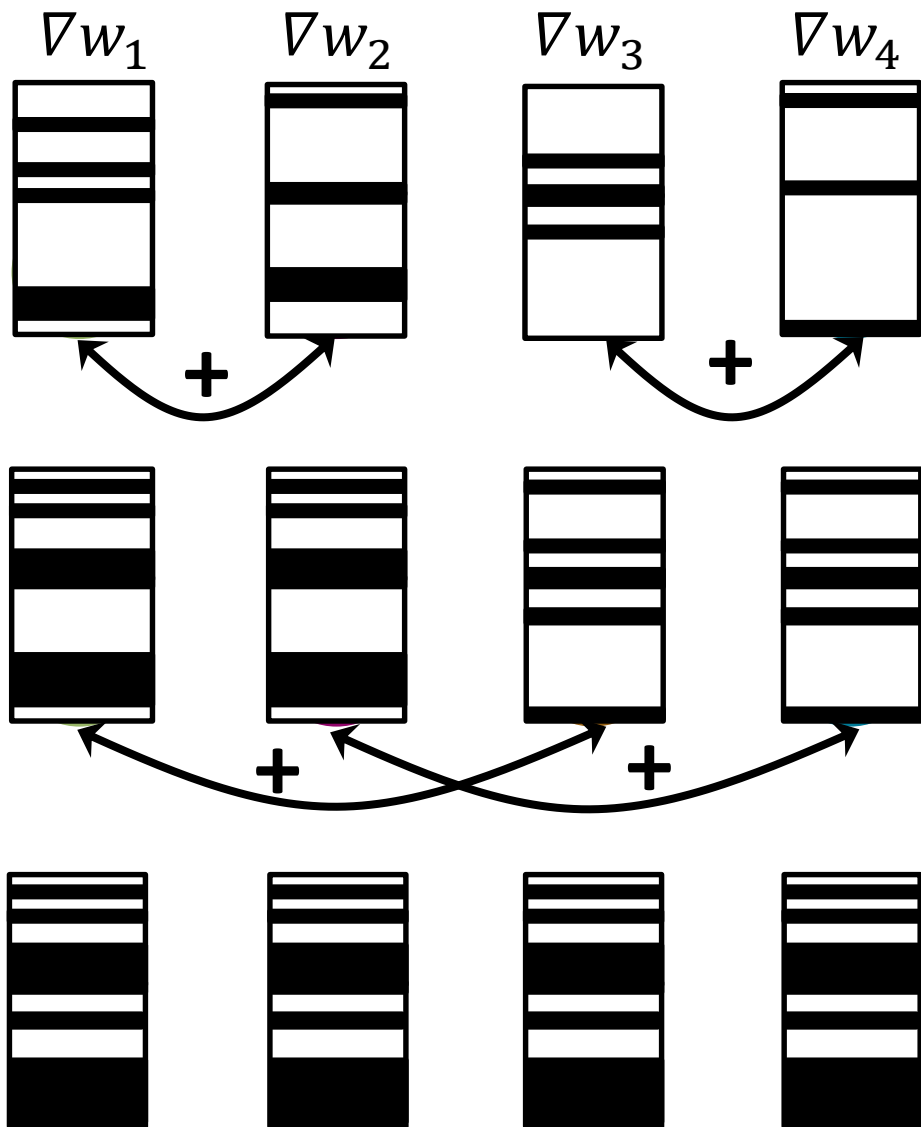
[1] S. Gupta et al. Deep Learning with Limited Numerical Precision, ICML'15

[2] F. Li and B. Liu. Ternary Weight Networks, arXiv 2016

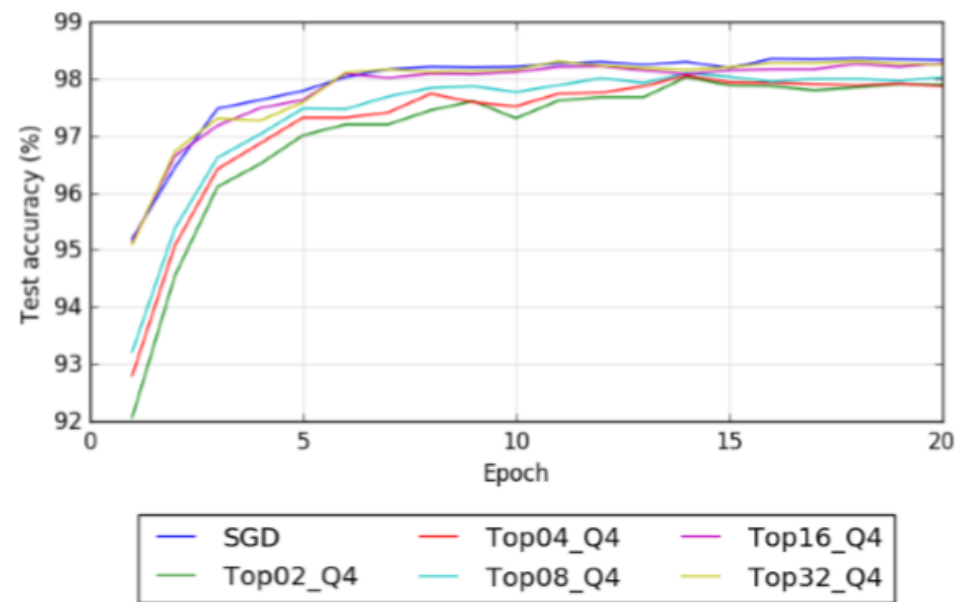
[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014

[4] C. Renggli et al. SparCML: High-Performance Sparse Communication for Machine Learning, arXiv 2018

SparCML – Quantified sparse allreduce for decentral updates



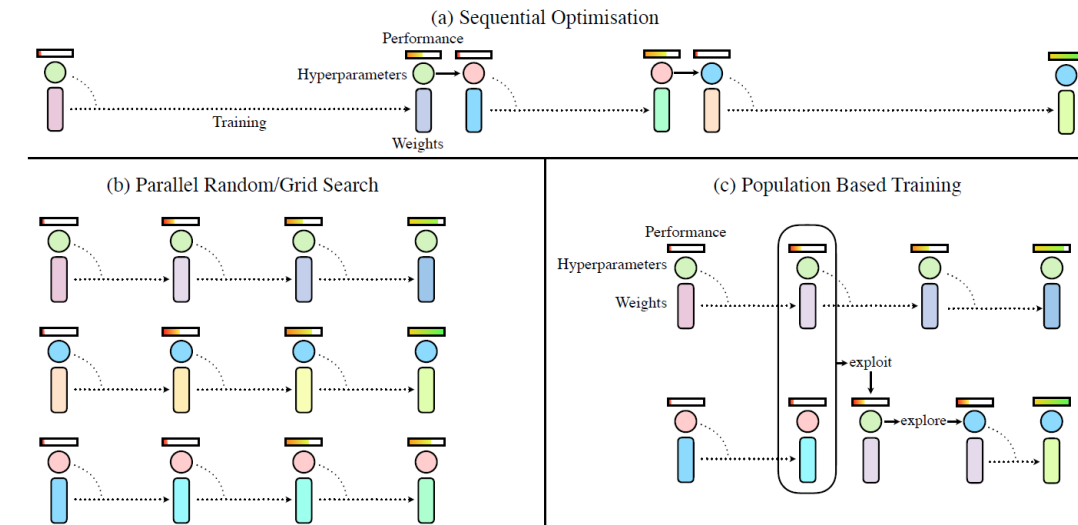
System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	1.55 (3.31)
Piz Daint	ImageNet	AlexNet	16	Q4	1.30 (1.36)
Piz Daint EC2	MNIST	MLP	8	Top16_Q4 Top16_Q4	3.65 (4.53) 19.12 (22.97)



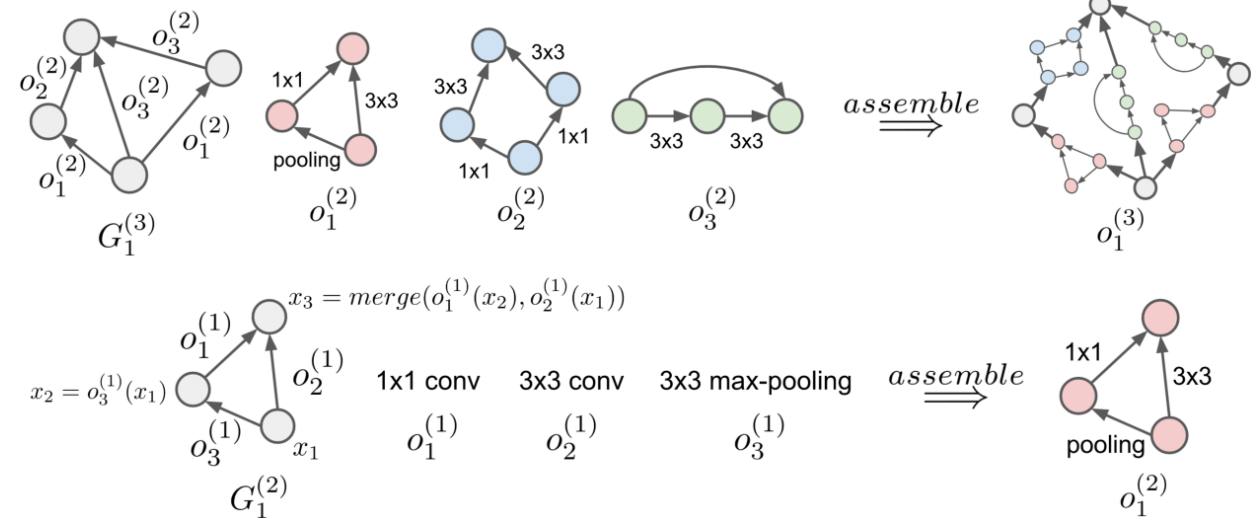
MNIST test accuracy

Hyperparameter and Architecture search

- Meta-optimization of hyper-parameters (momentum) and DNN architecture**
 - Using Reinforcement Learning [1] (explore/exploit different configurations)
 - Genetic Algorithms with modified (specialized) mutations [2]
 - Particle Swarm Optimization [3] and other meta-heuristics



Reinforcement Learning [1]



Evolutionary Algorithms [4]

[1] M. Jaderberg et al.: Population Based Training of Neural Networks, arXiv 2017
 [2] E. Real et al.: Regularized Evolution for Image Classifier Architecture Search, arXiv 2018
 [3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO'17
 [4] H. Liu et al.: Hierarchical Representations for Efficient Architecture Search, ICLR'18

Outlook

- **Full details in the survey (60 pages)**
 - Detailed analysis
- **Additional content:**
 - Unsupervised (GAN/autoencoders)
 - Recurrent (RNN/LSTM)
- **Call to action to the HPC and ML/DL communities to join forces!**
 - It's already happening on the tool basis
 - Need more joint events!



<https://www.arxiv.org/abs/1802.09941>

Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

TAL BEN-NUN* and TORSTEN HOEFLER, ETH Zurich

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and techniques range from distributed algorithms to low-level circuit design. In this survey, we describe the problem from a theoretical perspective, followed by approaches for its parallelization. Specifically, we present trends in DNN architectures and the resulting implications on parallelization strategies. We discuss the different types of concurrency in DNNs; synchronous and asynchronous stochastic gradient descent; distributed system architectures; communication schemes; and performance modeling. Based on these approaches, we extrapolate potential directions for parallelism in deep learning.

CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Computing methodologies** → *Neural networks*; *Distributed computing methodologies*; *Parallel computing methodologies*; *Machine learning*;

Additional Key Words and Phrases: Deep Learning, Distributed Computing, Parallel Algorithms

ACM Reference format:

Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. 60 pages.

1 INTRODUCTION

Machine Learning, and in particular Deep Learning [LeCun et al. 2015], is a field that is rapidly taking over a variety of aspects in our daily lives. In the core of deep learning lies the Deep Neural Network (DNN), a construct inspired by the interconnected nature of the human brain. Trained properly, the expressiveness of DNNs provides accurate solutions for problems previously thought to be unsolvable, simply by observing large amounts of data. Deep learning has been successfully implemented for a plethora of subjects, ranging from image classification [Huang et al. 2017], through speech recognition [Amodei et al. 2016] and medical diagnosis [Cireşan et al. 2013], to autonomous driving [Bojarski et al. 2016] and defeating human players in complex games [Silver et al. 2017] (see Fig. 1 for more examples).