

T. HOEFLER

HPC for ML and ML for HPC - Scalability, Communication, and Programming

Keynote at the MLHPC workshop at ACM/IEEE Supercomputing 2019

WITH CONTRIBUTIONS FROM TAL BEN-NUN, DAN ALISTARH, SHOSHANA JAKOBOVITS, CEDRIC RENGGLI, AND OTHERS AT SPCL, IST AUSTRIA, AND TOKYO TECH



ACM Computing Surveys (CSUR)

publishes comprehensive, readable tutorials and survey papers that give guided tours through the literature and explain topics to those who seek to learn the basics of areas outside their specialties.

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



[Home](#) [Authors](#) [Editors](#) [Reviewers](#) [Archive](#) [About](#) [Announcements](#) [Contact](#) [Subscribe](#)

Latest Articles

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

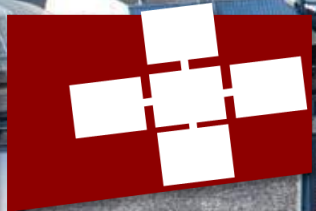
Tal Ben-Nun, Torsten Hoefler

Deep Neural Networks (DNNs) are becoming an important tool in modern computing applications. Accelerating their training is a major challenge and... (more)

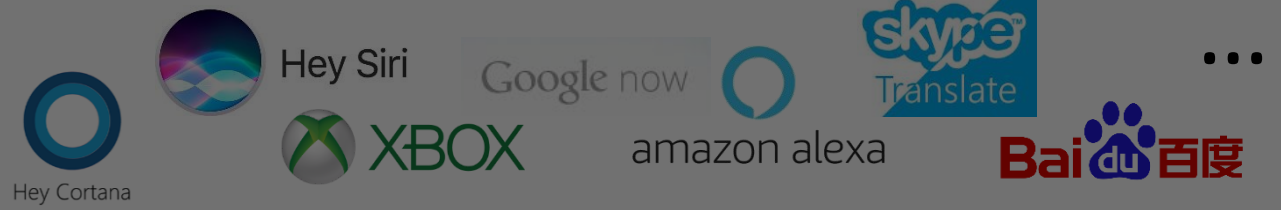
Search CSUR

enter search term and/or author name below:

Current Issue: Volume 52 Issue 4, August 2019 (Issue-in-Progress)



What is Deep Learning good for?



Digit Recognition

Object Classification
Segmentation
Image Captioning

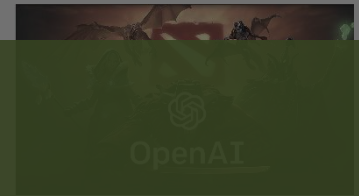
Gameplay AI
Translation

Neural Computers

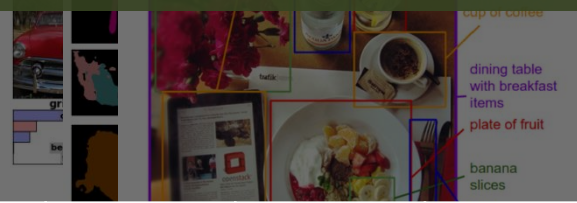
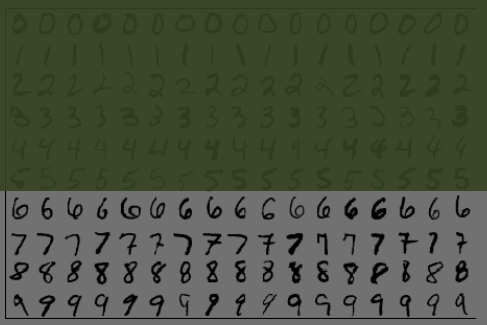
Gigantic Language
Models

Towards
Real Physics

GPT-2 BERT



A very active area of research!



Her gown with the dots that are pink, white and

44 papers per day!

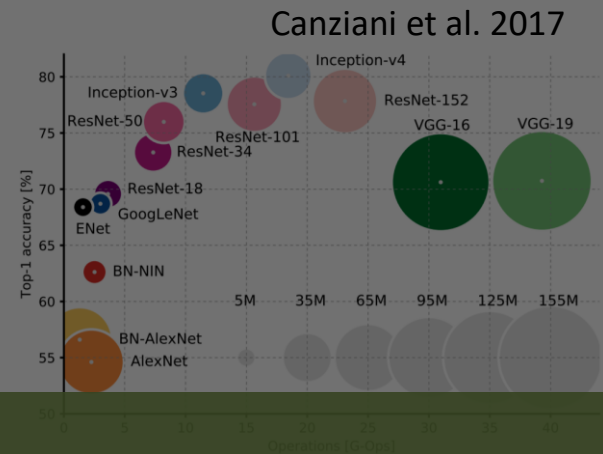
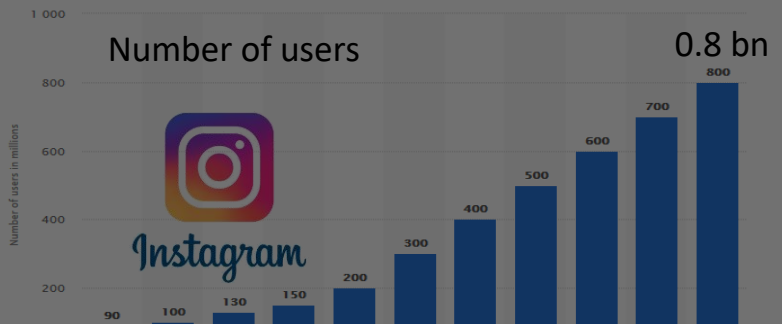


Year	2012	2013	2014	2015	2016	2017	2018	2019
cs.AI	1,081	1,765	1,022	1,105	1,929	2,790	4,262	4,176
cs.CV	577	852	1,349	2,261	3,627	5,693	8,599	9,825

number of papers per year

1989 2012 2013 2014 2016 2017 2018 2019

How does Deep Learning work?

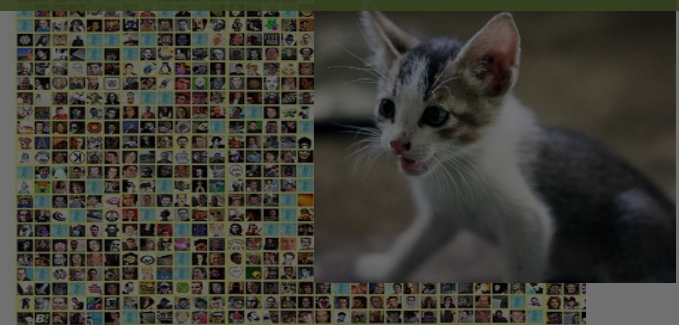


What is Deep Learning used for?

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

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

Deep Learning is Supercomputing!



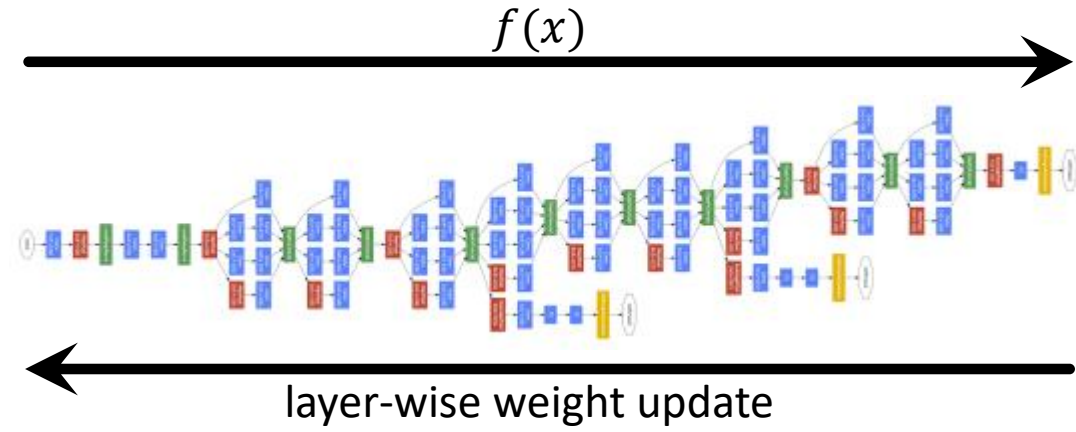
Cat	0.54	Cat	0.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.04	Horse	0.00
Bicycle	0.02	Bicycle	0.00
Truck	0.02	Truck	0.00

- 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

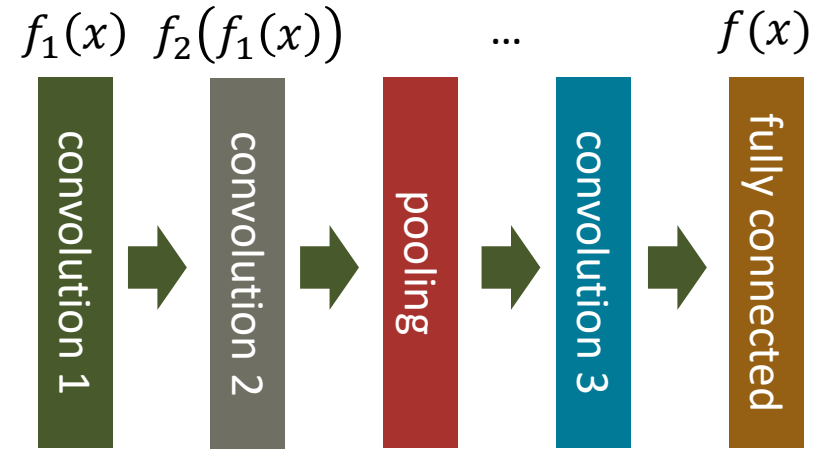
labeled samples $x \in X \subset \mathcal{D}$

label domain Y true label $l(x)$

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

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

network structure (fixed) weights w (learned)



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

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

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

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

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

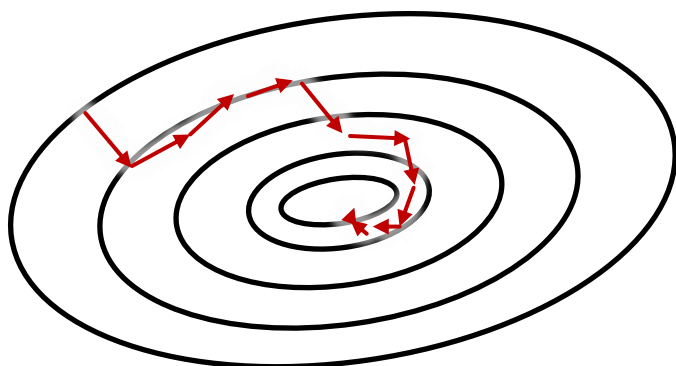
pooling

...

convolution 3

$f(x)$ fully connected

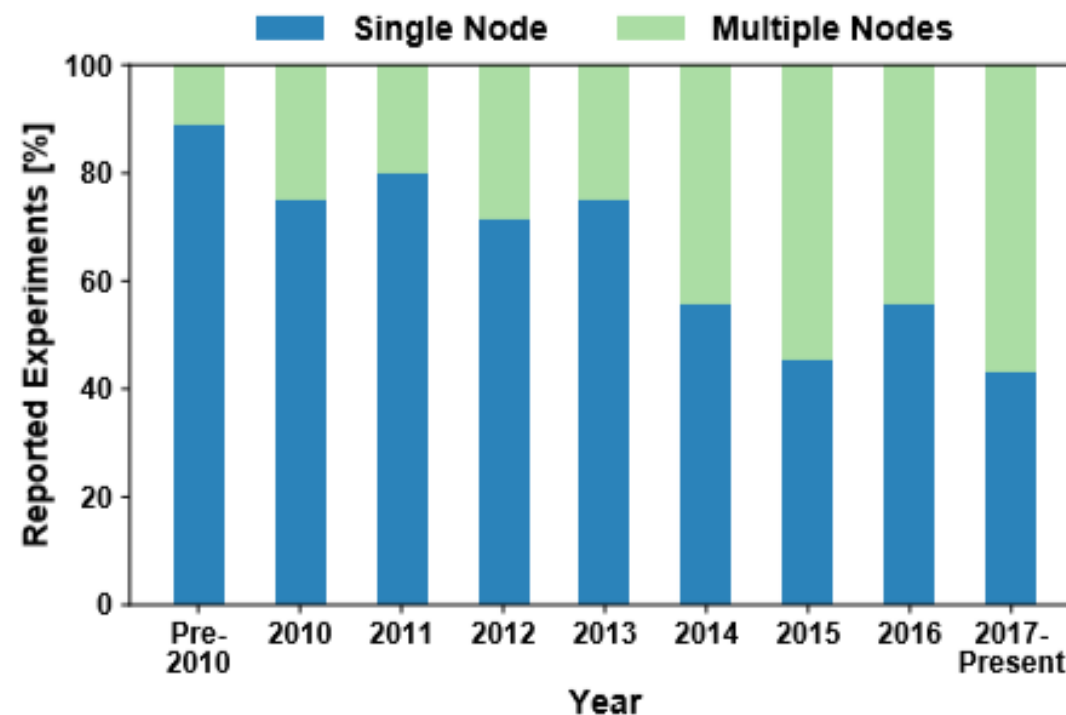
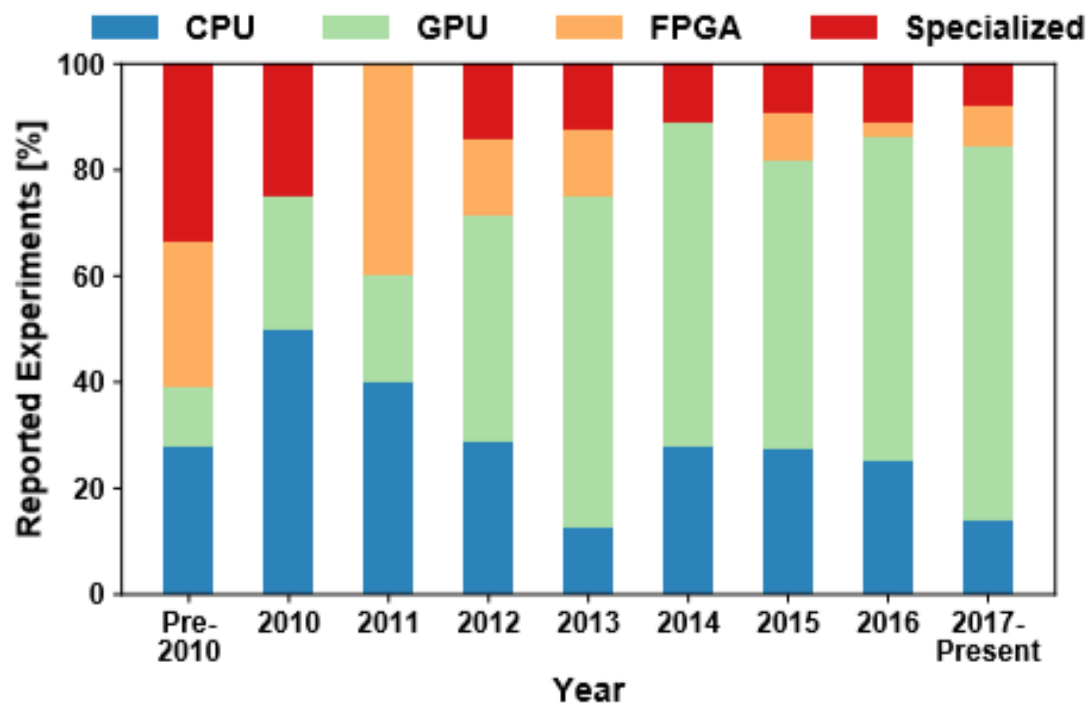
- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



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

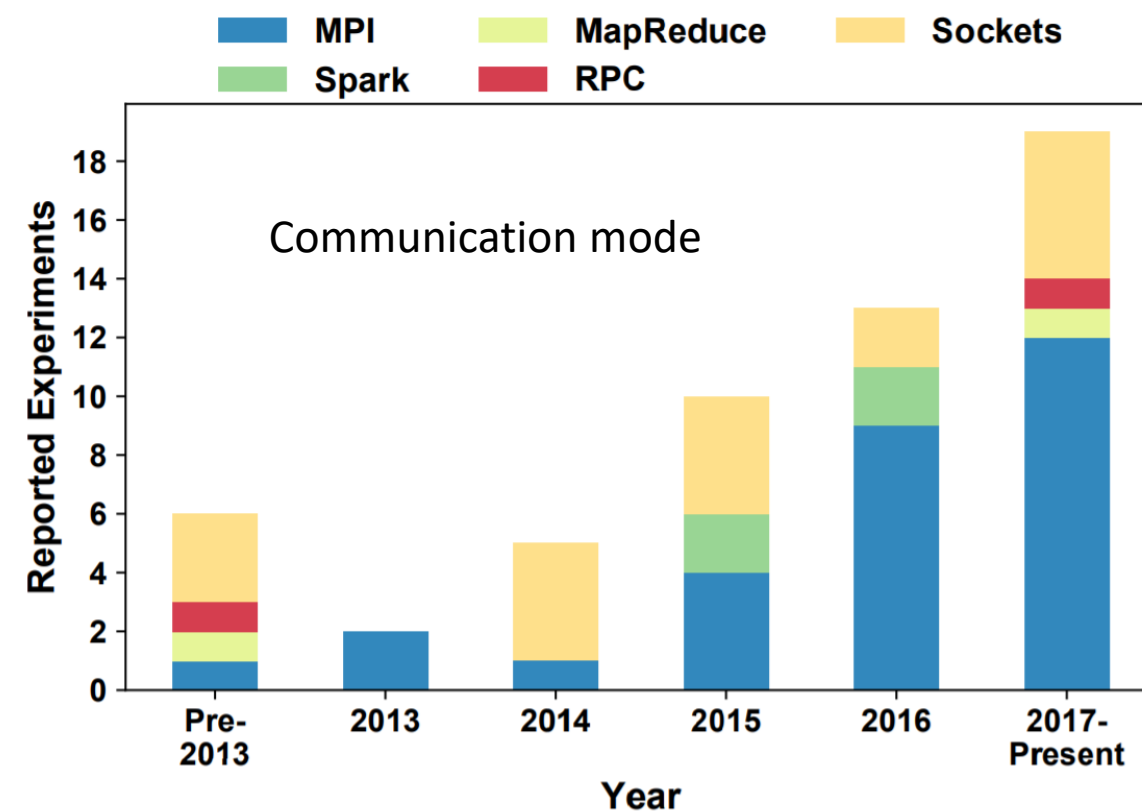
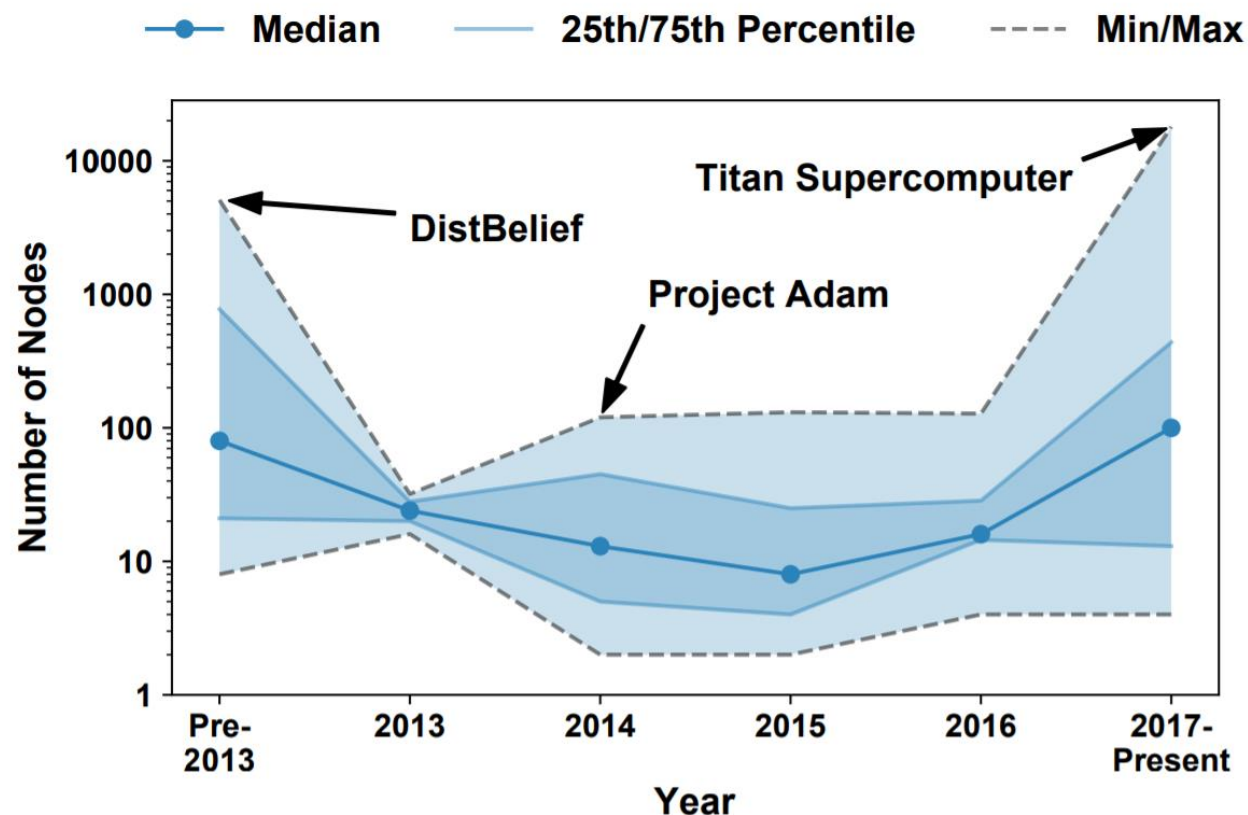
The field is moving fast – trying everything imaginable – survey results from 227 papers in the area of parallel deep learning



Deep Learning is largely on distributed memory today!

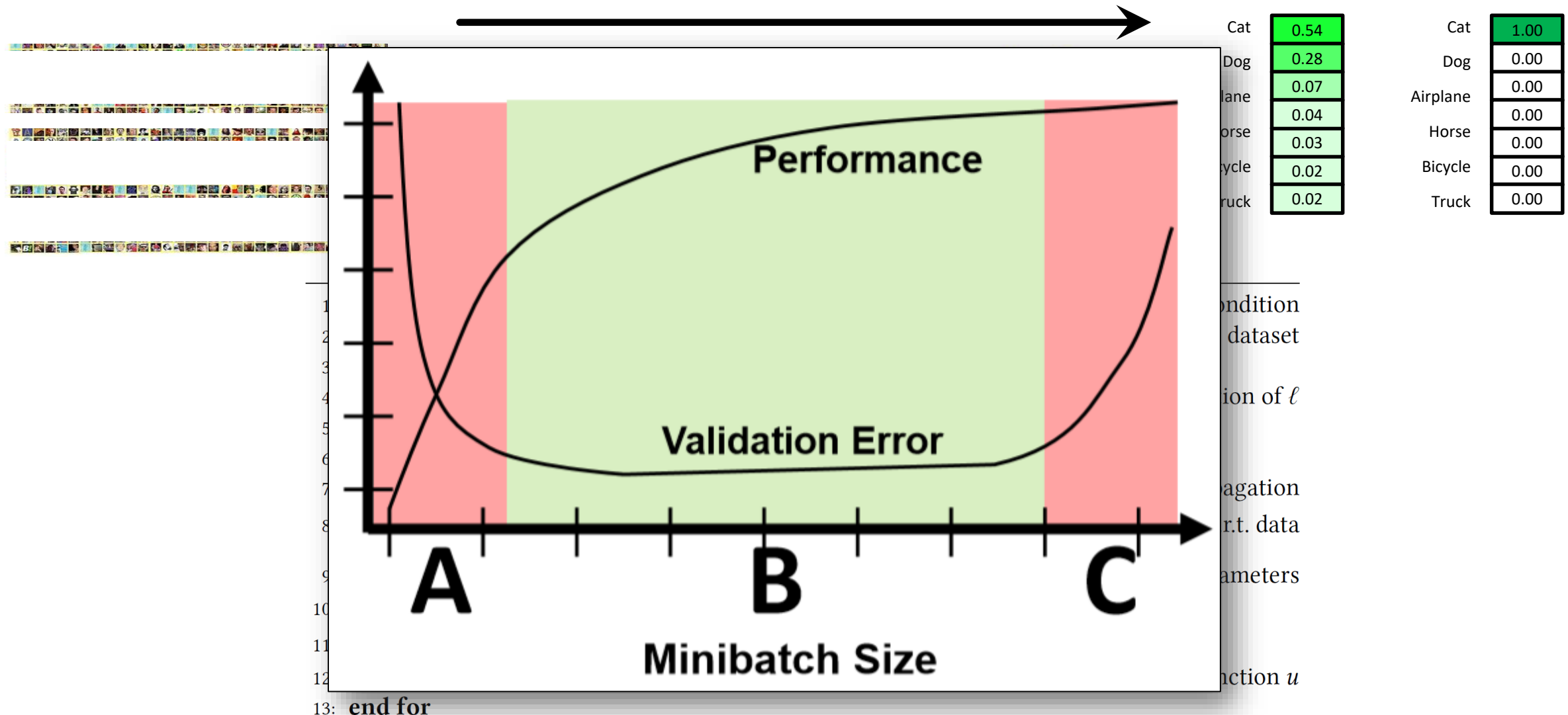
Trends in **distributed** deep learning: node count and communication

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

Microbatching (μ -cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace for different algorithms

Fast (up to 4.54x faster on DeepBench)

Microbatching Strategy

- How to choose microbatching strategy

none (undivided)

$$T(b) = \min_{\mu} \left\{ \begin{array}{l} T_{\mu}(b), \\ \min_{\nu=1,2,\dots} T_{\nu}(b) \end{array} \right.$$

powers-of-two only

Dynamic Program

any (unrestricted)

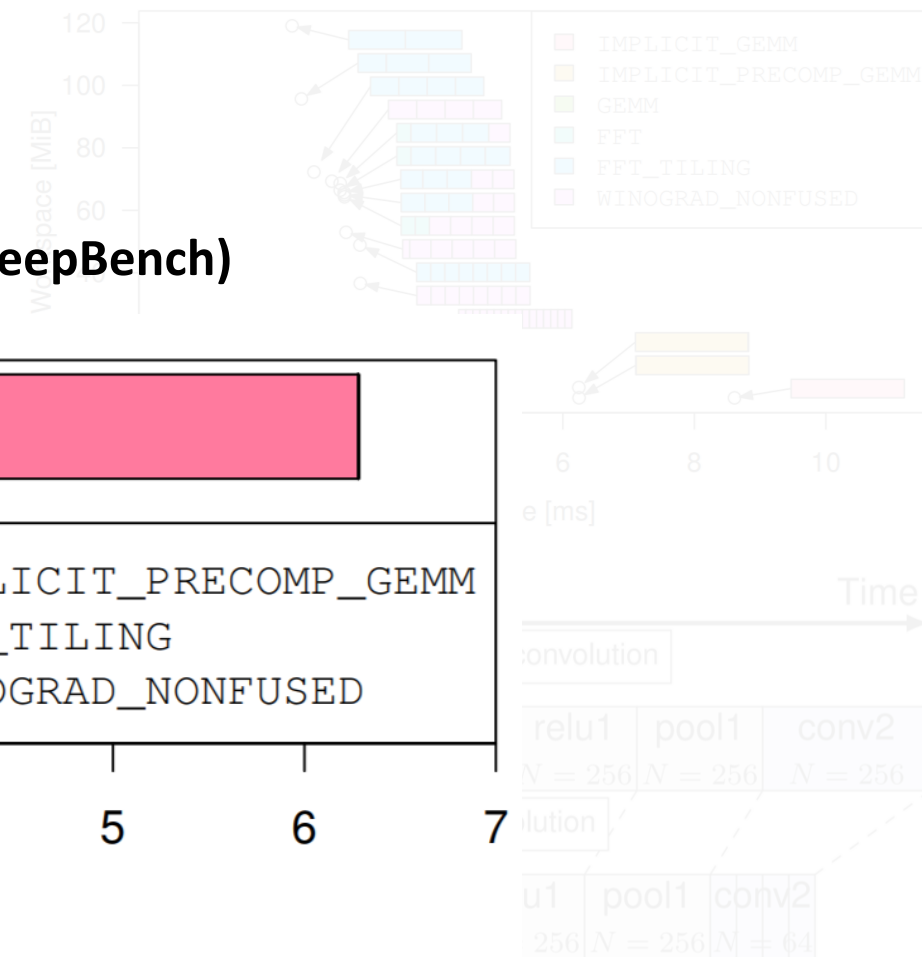
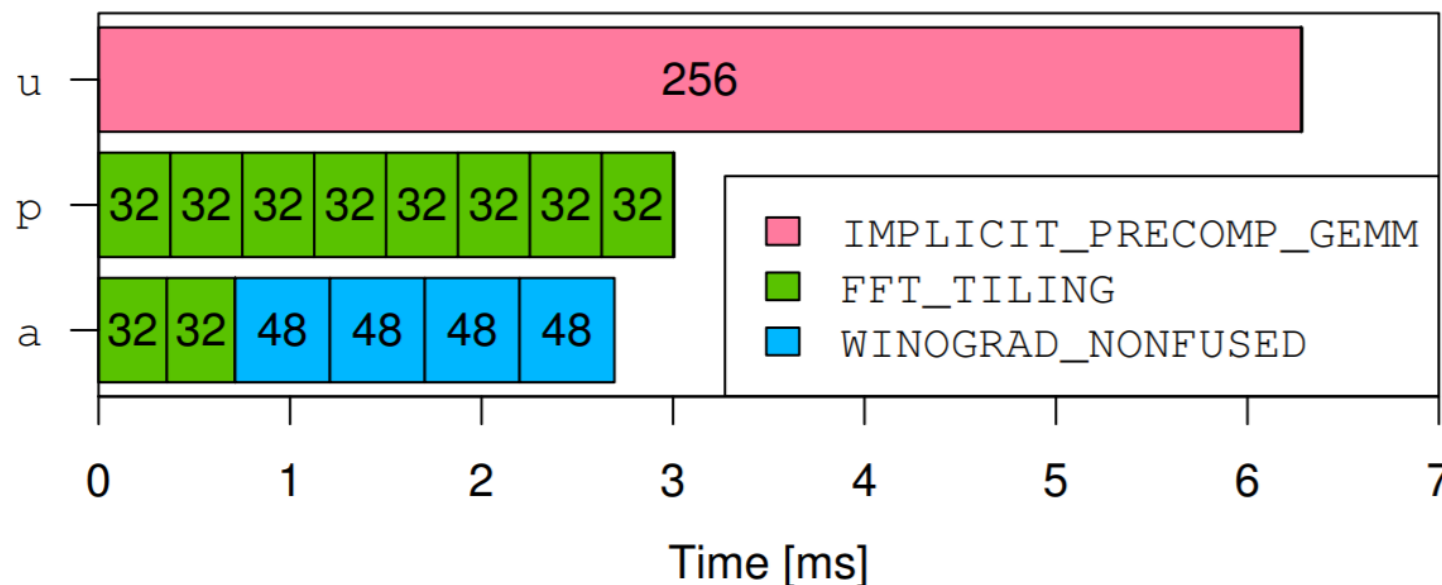
$$\min_{x_{k,c}} T = \sum_{k \in \mathcal{K}} T_k$$

subject to

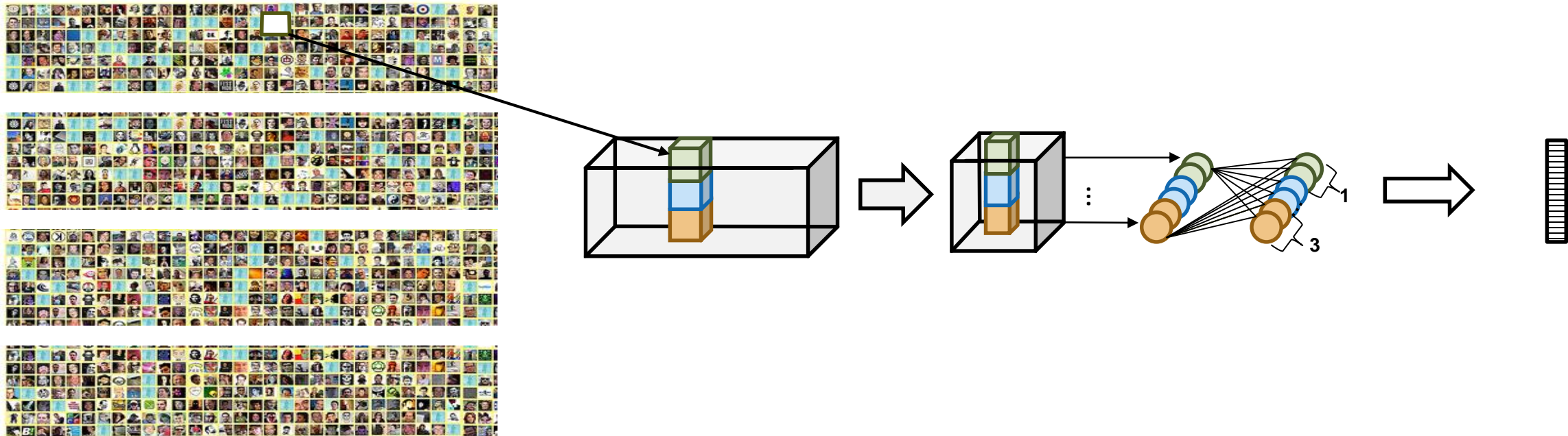
$$\sum_{k \in \mathcal{K}} \sum_{c \in C_k} x_{k,c} = 1$$

$$x_{k,c} \in \{0, 1\} \quad (\forall k \in \mathcal{K}, \forall c \in C_k)$$

Integer Linear Programming (Space Sharing)

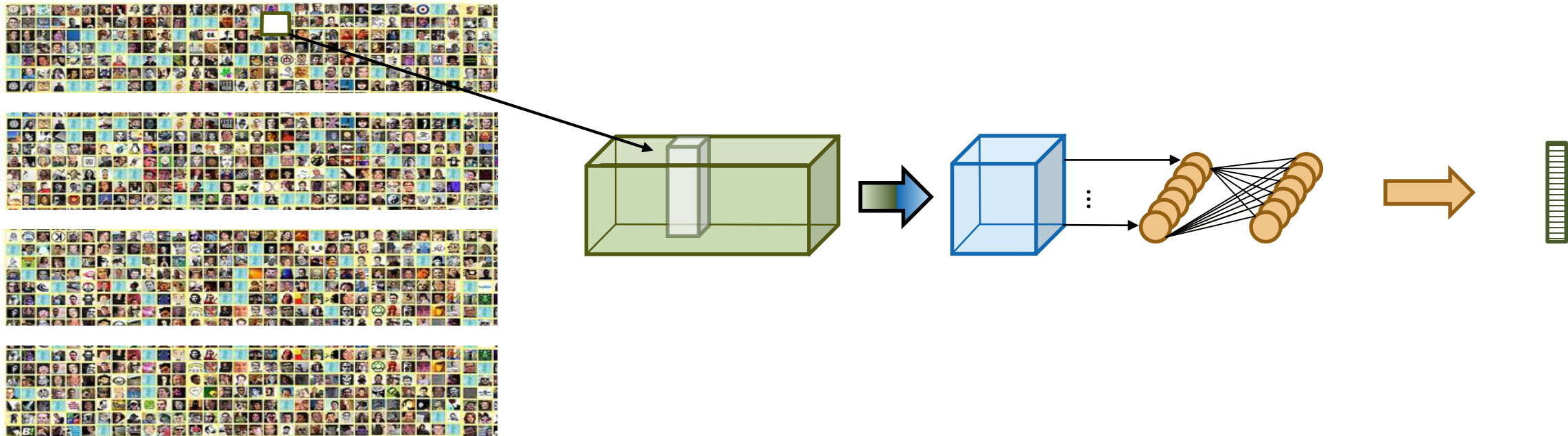


Layer parallelism – limited by network size



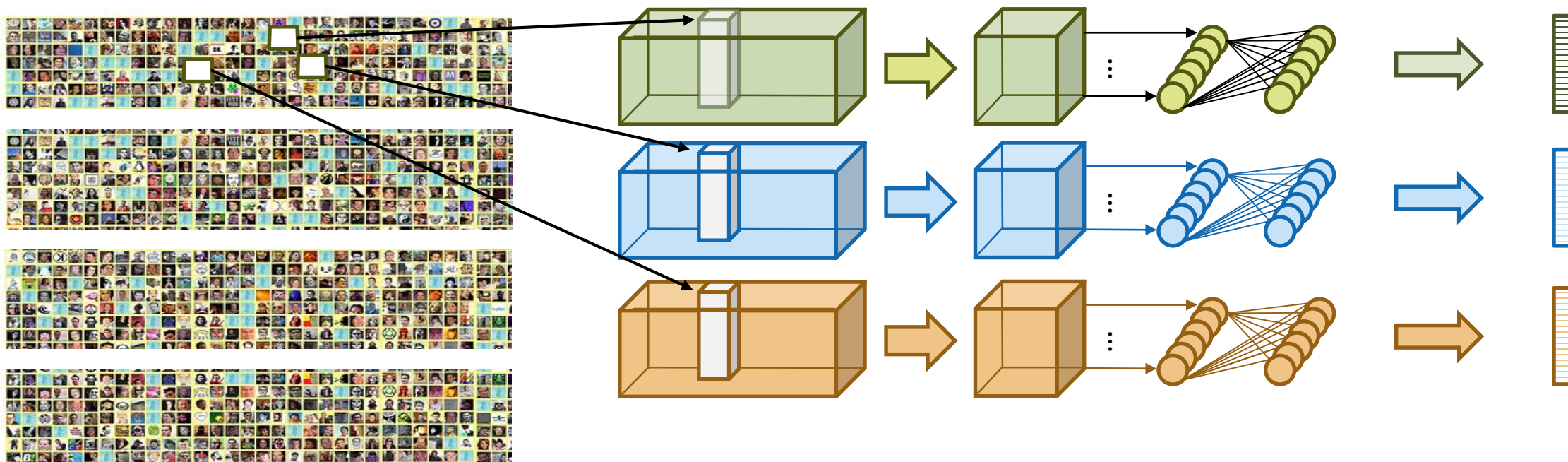
- Parameters can be distributed across processors
- **Mini-batch has to be copied to all processors**
- **Backpropagation requires all-to-all communication every layer**

Pipeline parallelism – limited by network size



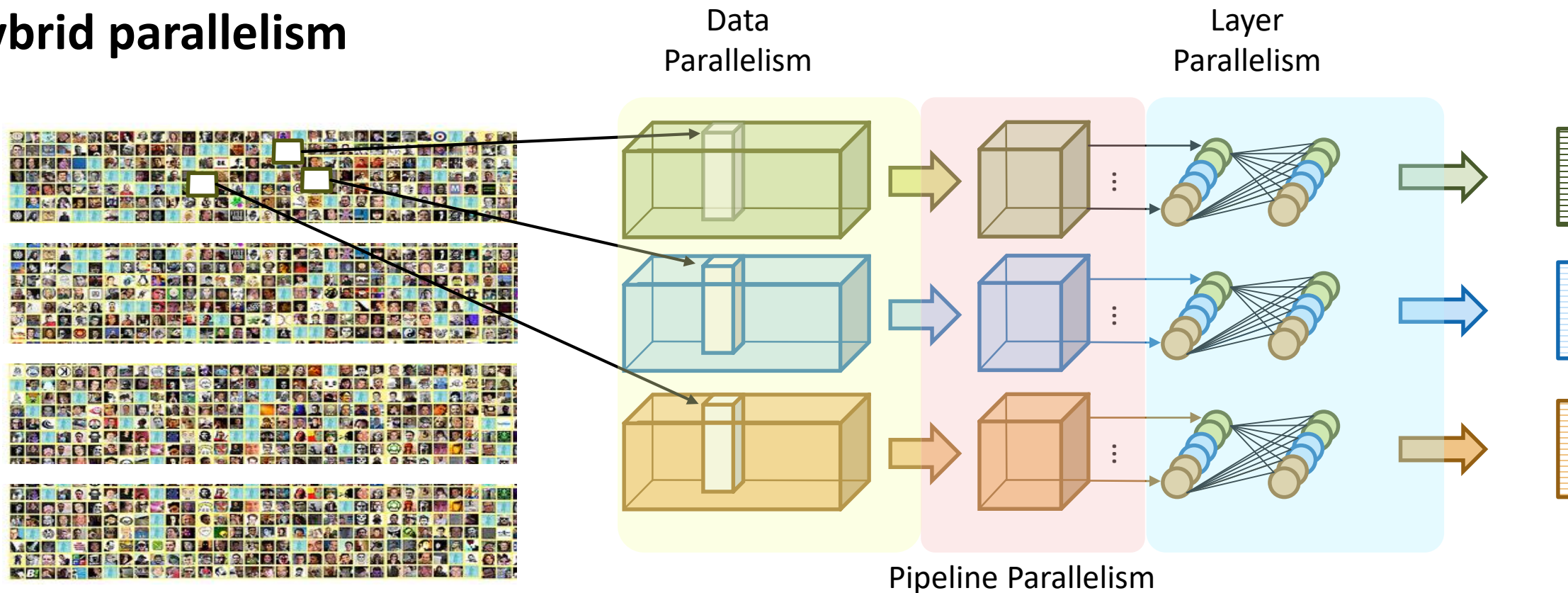
- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- **Mini-batch has to be copied through all processors**

Data parallelism – limited by batch-size



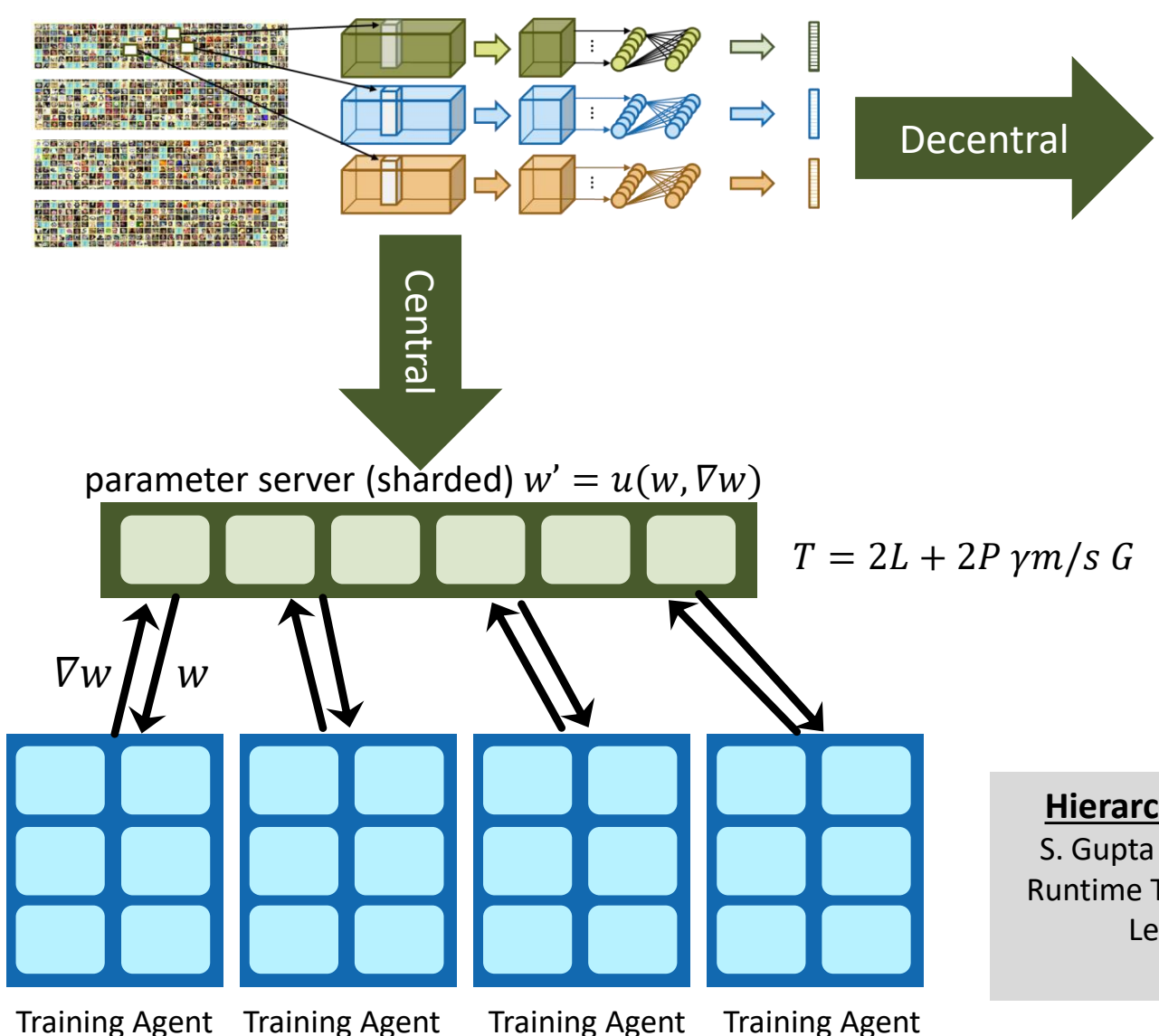
- Simple and efficient solution, easy to implement
- **Duplicate parameters at all processors**

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!

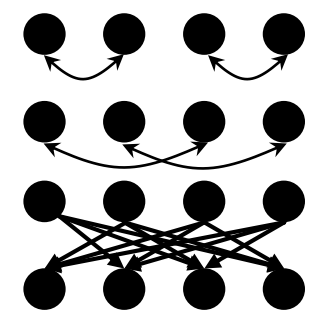
Updating parameters in **distributed** data parallelism



$$T = 2L + 2P \gamma m / s G$$



- Collective operations
- Topologies
- Neighborhood collectives
- RMA?



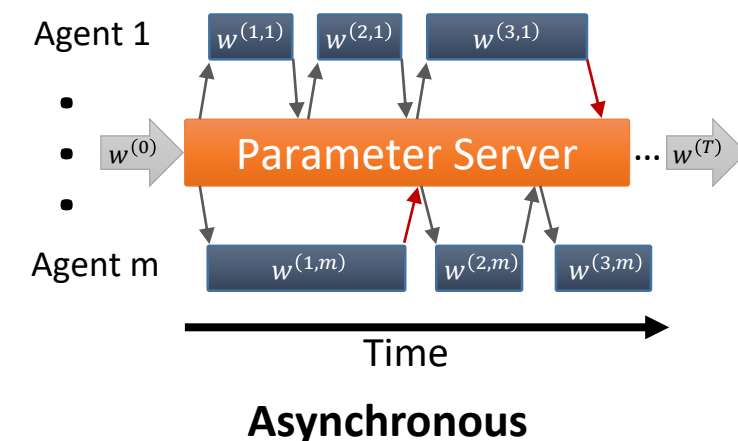
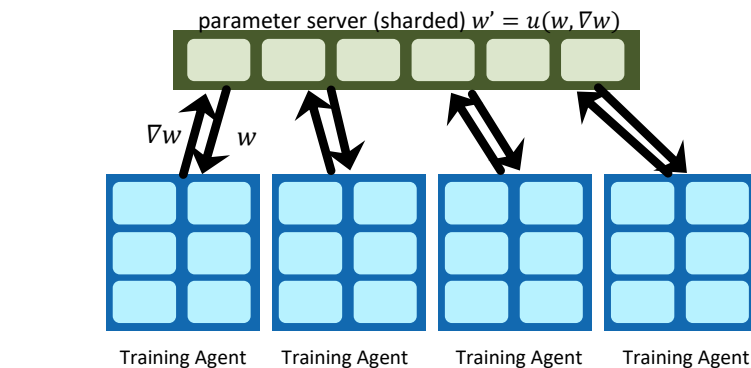
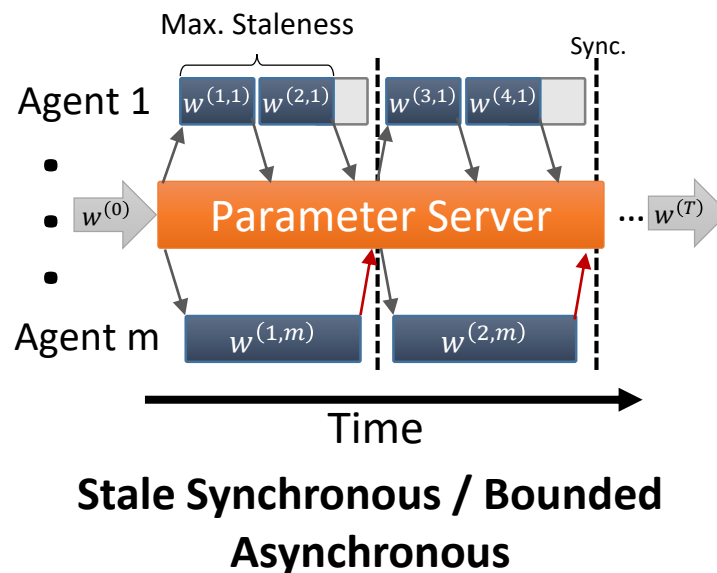
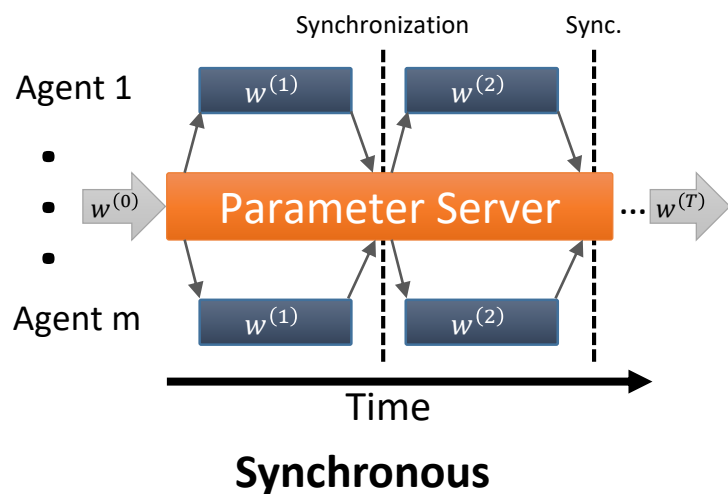
$$T = 2L \log_2 P + 2\gamma m G (P - 1) / P$$

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:



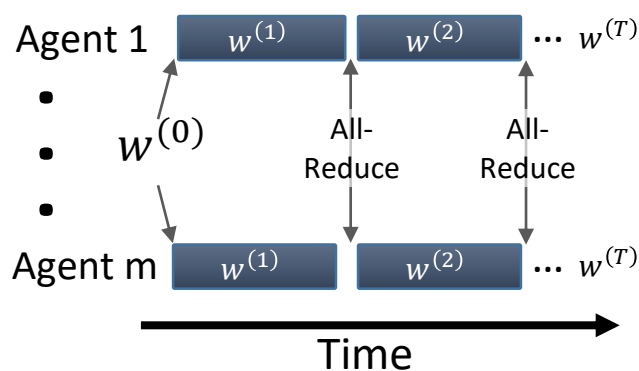
- Started with Hogwild! [Niu et al. 2011] – shared memory, by chance
- DistBelief [Dean et al. 2012] moved the idea to distributed
- Trades off “statistical performance” for “hardware performance”

Parameter (and Model) consistency - decentralized

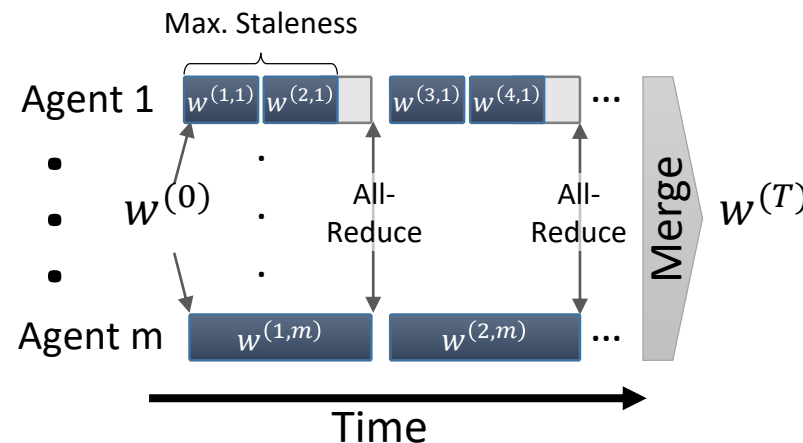
- Parameter exchange frequency can be controlled, while still attaining convergence:



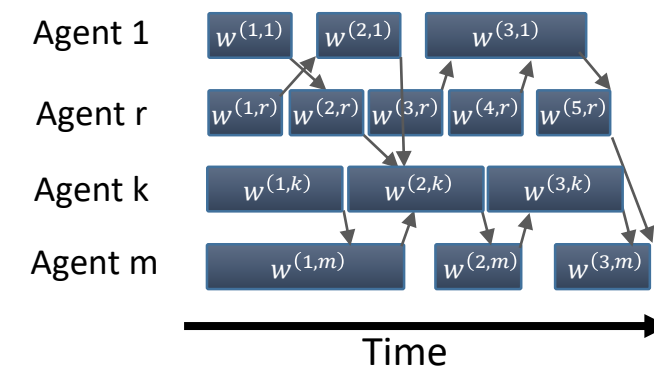
Training Agent Training Agent Training Agent Training Agent



Synchronous



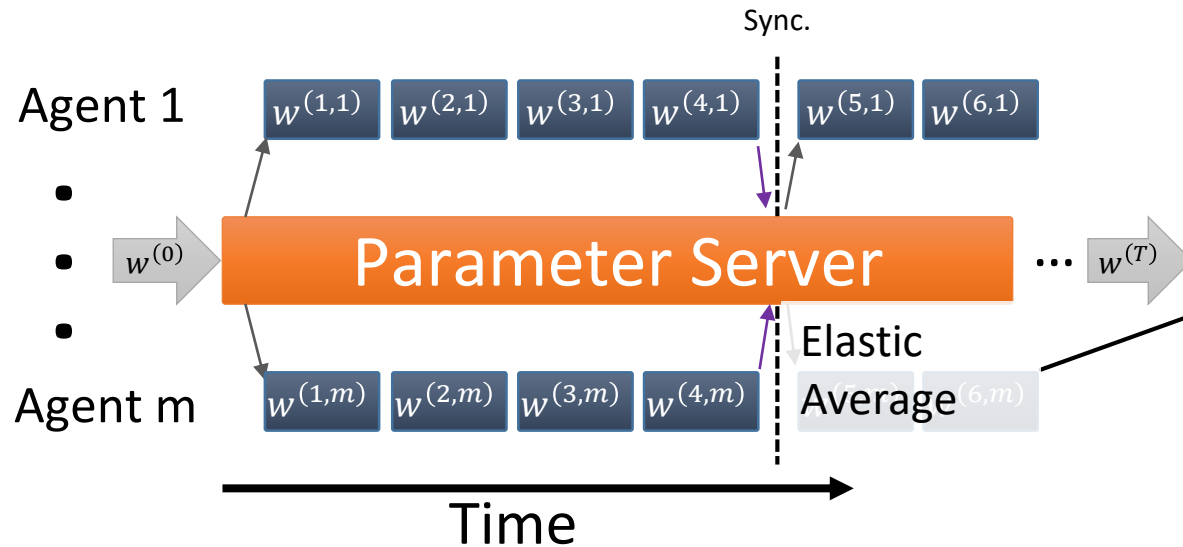
Stale Synchronous / Bounded Asynchronous



Asynchronous

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

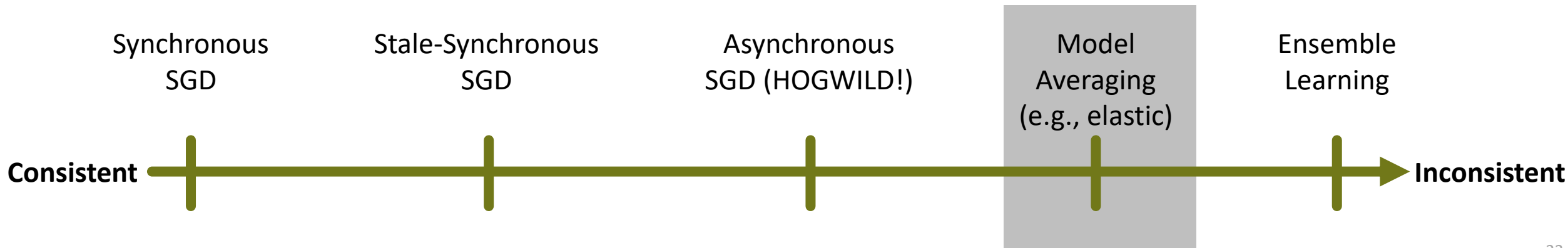
Parameter consistency in deep learning



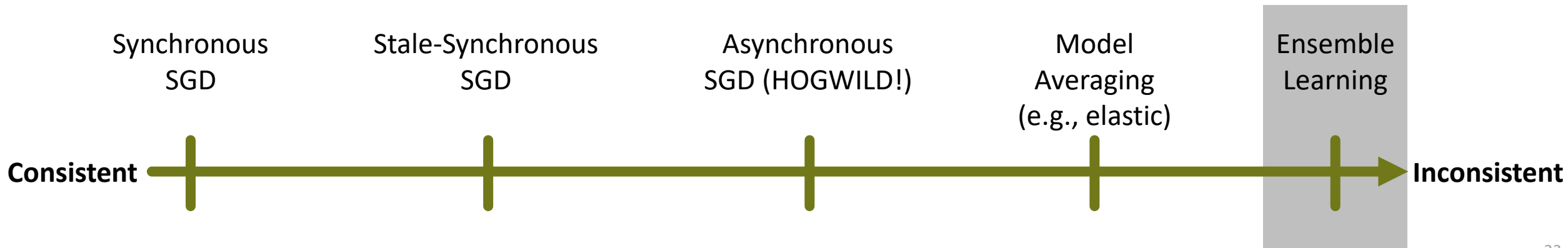
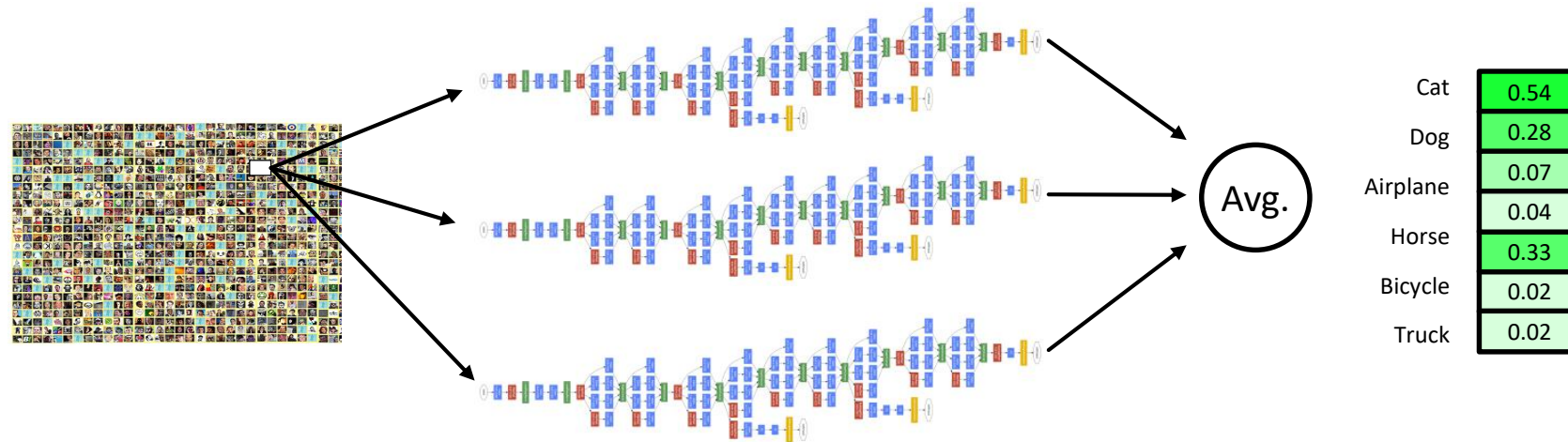
Using physical forces between different versions of w :

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

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

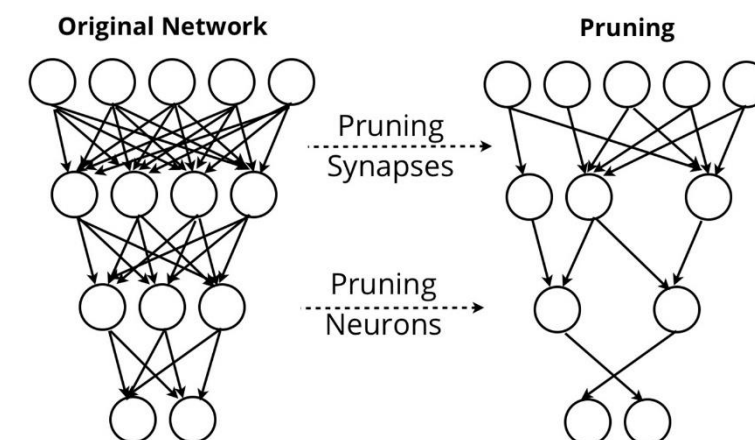
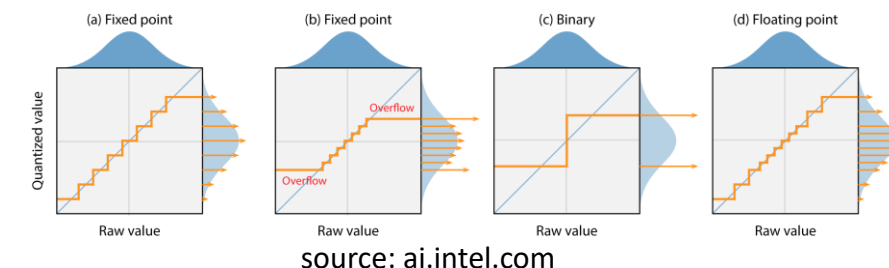
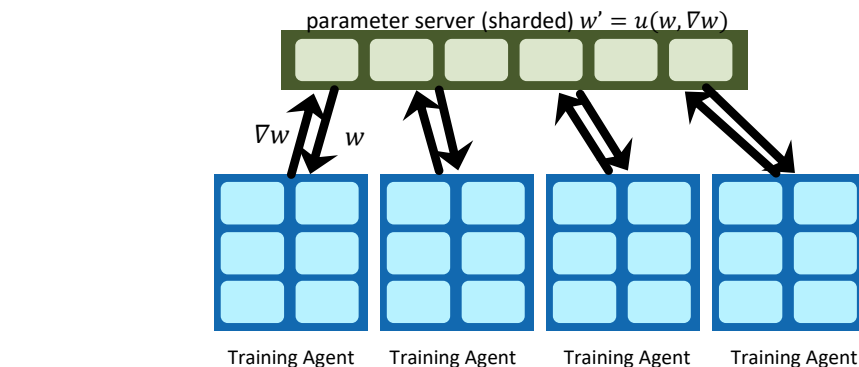


Parameter consistency in deep learning



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 omitted gradients 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

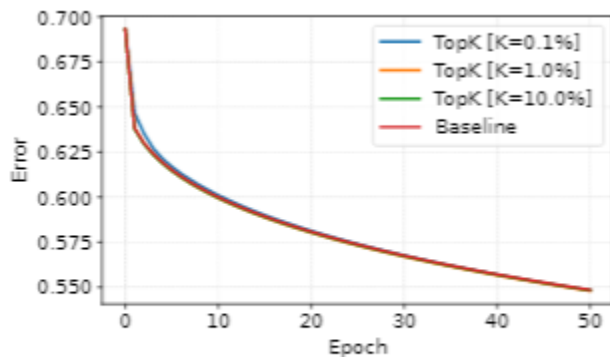
Sparsification – top-k Stochastic Gradient Descent

- Pick the k-largest elements of the vector at each node!**
 - Accumulate the remainder locally (convergence proof, similar to async. SGD with implicit staleness bounds [1])

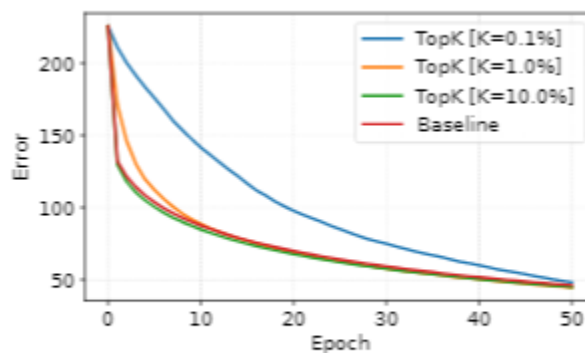
Assumption 1. *There exists a (small) constant ξ such that, for every iteration $t \geq 0$, we have:*

$$\left\| \text{TopK} \left(\frac{1}{P} \sum_{p=1}^P \left(\alpha \tilde{G}_t^p(v_t) + \epsilon_t^p \right) \right) - \sum_{p=1}^P \frac{1}{P} \text{TopK} \left(\alpha \tilde{G}_t^p(v_t) + \epsilon_t^p \right) \right\| \leq \xi \|\alpha \tilde{G}_t(v_t)\|.$$

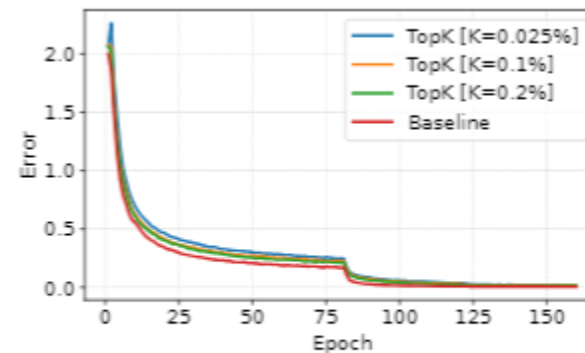
Discussion. We validate Assumption 1 experimentally on a number of different learning tasks in Section 6 (see also Figure 1). In addition, we emphasize the following points:



(a) RCV1 convergence.

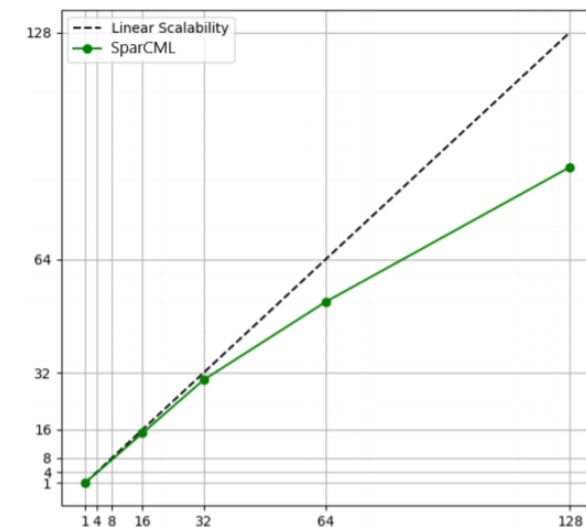
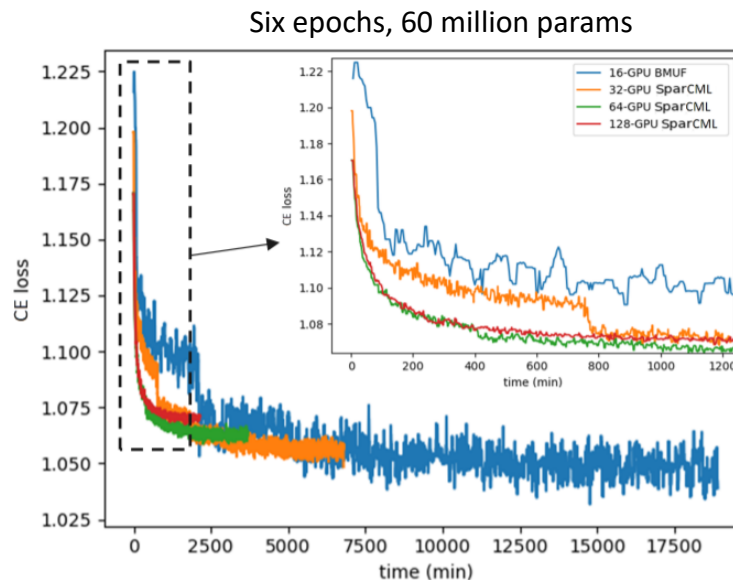
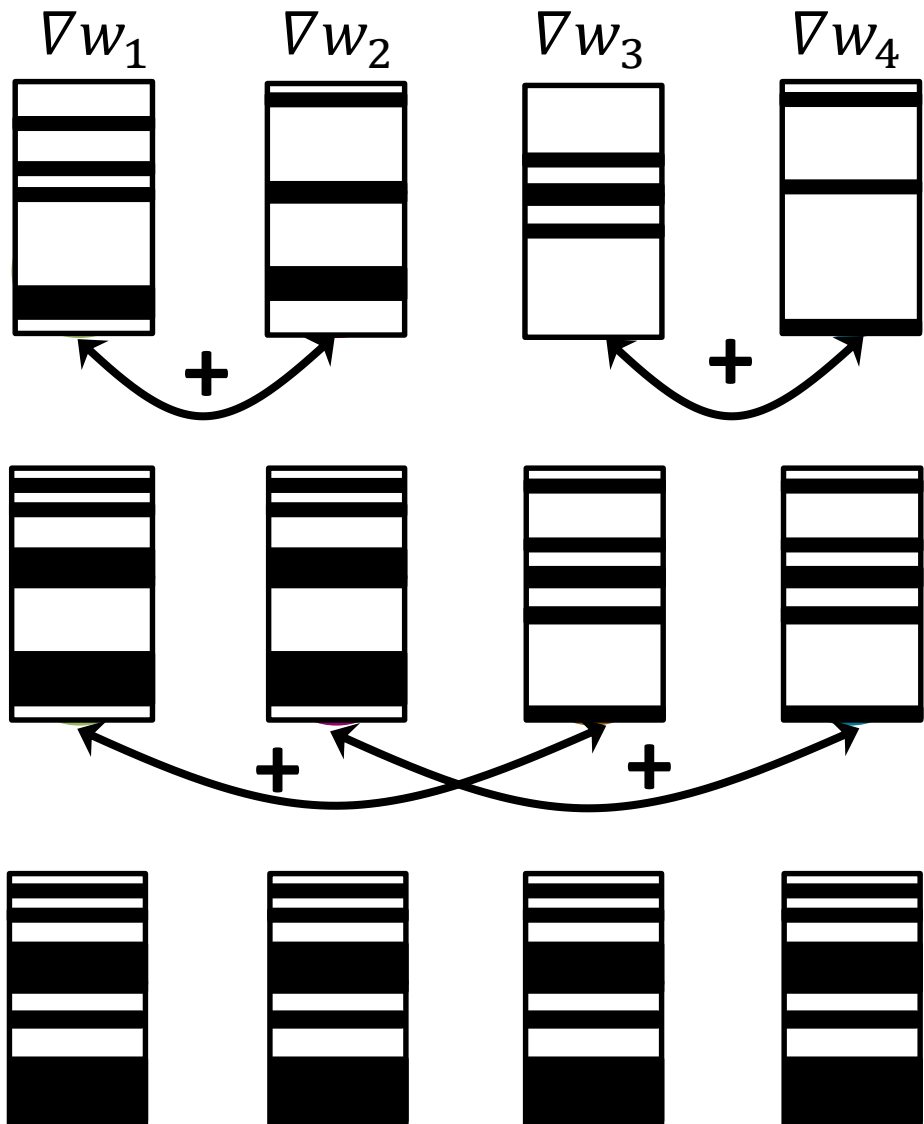


(b) Linear regression.



(c) ResNet110 on CIFAR10.

SparCML – Quantified sparse allreduce for decentral updates



Microsoft Speech Production Workload Results – **2 weeks** → **2 days!**

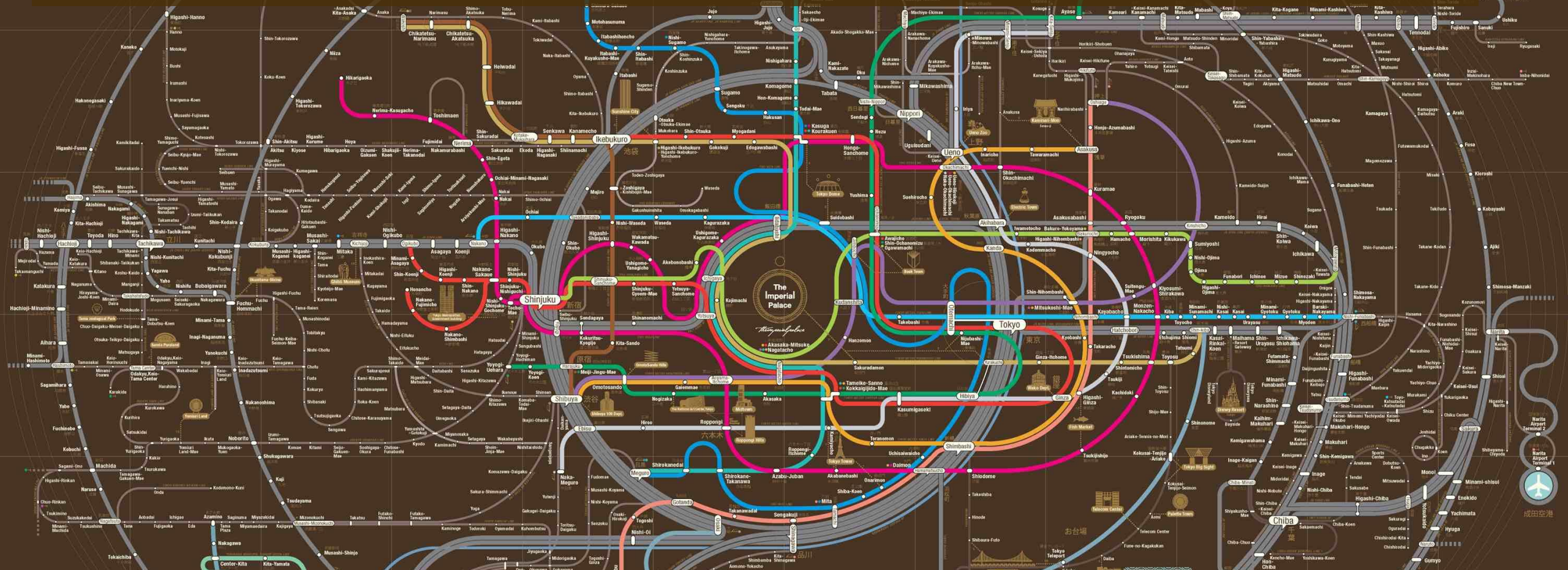
System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	1.55 (3.31)
Piz Daint	ImageNet	AlexNet	16	Q4	1.30 (1.36)
Piz Daint EC2	MNIST	MLP	8	Top16_Q4 Top16_Q4	3.65 (4.53) 19.12 (22.97)

TOKYO METROPOLITAN RAILWAY SYSTEM

TOKYO RAILWAY CITY MAP, OCTOBER 2006
THIS IS NOT THE OFFICIAL MAP. THE DESIGNER IS NOT GUARANTEED.
DESIGN/PRODUCT BY SPCL P&S 2010. ALL RIGHTS RESERVED.
WWW.SPCLP&S.COM PRINTED IN KOREA

Optimizing parallel deep learning systems is a bit like navigating Tokyo by public transit

--- at first glance impossible but eventually doable with the right guidelines ---



Deep500: An HPC Deep Learning Benchmark and Competition

- Integrates tensorflow, pytorch, caffe2 into a single benchmarking framework

- Separate definition of benchmark metrics, shared across all levels

- Lean reference implementations – simple to understand and change

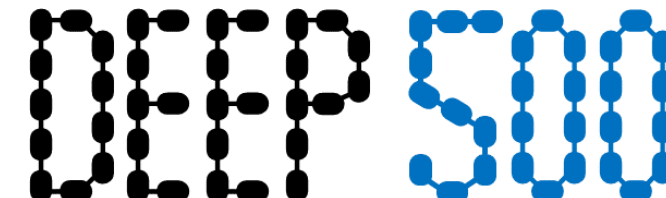
- Operators (layer computations)
- Optimizers (SGD etc.)
- Distribution schemes (cf. Horovod)

Similar to reference LINPACK benchmark

- Supports optimization of components

- E.g., no need to reimplement an optimizer to replace gradient compression!

Easily compare to all frameworks!



500 ways to train DNNs

A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning

Tal Ben-Nun, Simon Huber, Maciej Besta, Alexandros Nikolaos Ziogas, Daniel Peter, Torsten Hoeftler
Department of Computer Science, ETH Zurich

Abstract—We introduce Deep500: the first customizable benchmarking infrastructure that enables fair comparison of the plethora of deep learning frameworks, algorithms, libraries, and techniques. The key idea behind Deep500 is its *modular* design, where deep learning is factorized into four distinct levels: operators, network processing, training, and distributed training. Our evaluation illustrates that Deep500 is *customizable* (enables combining and benchmarking different deep learning codes) and *fair* (uses carefully selected metrics). Moreover, Deep500 is *fast* (incurs negligible overheads), *verifiable* (offers infrastructure to analyze correctness), and *reproducible*. Finally, as the first distributed and reproducible benchmarking system for deep learning, Deep500 provides software infrastructure to utilize the most powerful supercomputers for extreme-scale workloads.

Index Terms—Distributed Deep Learning, High-Performance Deep Learning, Parallel Deep Learning, Benchmarking

Deep500 code for reproducibility: <https://github.com/deep500/deep500>

I. INTRODUCTION

Deep Learning (DL) has transformed the world and is now ubiquitous in areas such as speech recognition, image classification, or autonomous driving [3]. Its central concept is a Deep Neural Network (DNN), a structure modeled after the human brain. Thanks to rigorous training, DNNs are able to solve various problems, previously deemed unsolvable.

Recent years saw an unprecedented growth in the number of approaches, schemes, algorithms, applications, platforms, and frameworks for DL. First, DL computations can aim at inference or training. Second, hardware platforms can vary significantly, including CPUs, GPUs, or FPGAs. Third, operators can be computed using different methods, e.g., im2col [5] or Winograd [26] in convolutions. Next, DL functionalities have been deployed in a variety of frameworks, such as TensorFlow [14] or Caffe [20]. These functionalities may incorporate many parallel and distributed optimizations, such as data, model, and pipeline parallelism. Finally, DL workloads are executed in widely varying environments, such as mobile phones, multi-GPU clusters, or large-scale supercomputers.

This richness of the DL domain raises a question we have not seen addressed so far: How can one ensure a leveled, fair ground for comparison, competition, and benchmarking in Deep Learning? The key issue here is that the recent benchmarking approaches such as DAWN-Bench [9] or MLPerf [30] are merely lists of results that do not directly consider the rich nature of today's DL efforts.

To answer this question, we propose Deep500: a benchmarking system that enables fair analysis and comparison of diverse DL efforts. Deep500 is based on the following five pillars:

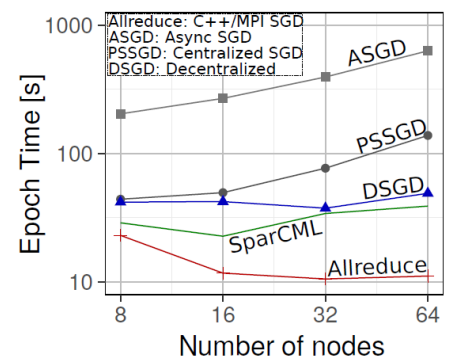
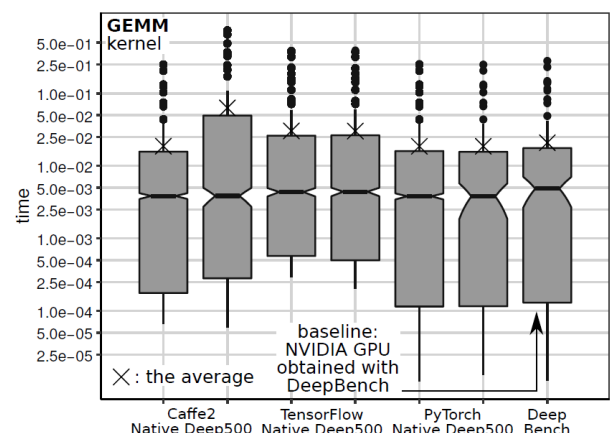
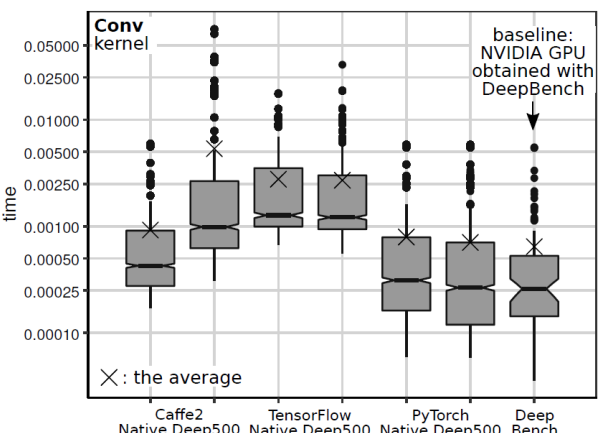
- Customizability,
 - Metrics,
 - Performance,
 - Validation, and
 - Reproducibility.
 - Customizability

Deep500 enables benchmarking of arbitrary combinations of DL elements, such as various frameworks running on different platforms, and executing custom algorithms. To achieve this, we design Deep500 to be a *meta-framework* that can be straightforwardly extended to benchmark any DL code. Table I illustrates how various DL frameworks, libraries, and frontends can be integrated in Deep500 to enable easier and faster DL programming. "Metrics" indicates that Deep500 embraces a complex nature of DL that, unlike benchmarks such as Top500 [15], makes a single number such as FLOPS an insufficient measure. To this end, we propose metrics that consider the accuracy-related aspects of DL (e.g., time required to ensure a specific test-set accuracy) and performance-related issues (e.g., communication volume). "Performance" means that Deep500 is the first DL benchmarking infrastructure that can be integrated with parallel and distributed DL codes. "Validation" indicates that Deep500 provides infrastructure to ensure correctness of aspects such as convergence. Finally, Deep500 embraces "Reproducibility" as specified in recent HPC initiatives [18] to help developing reproducible DL codes.

Table II compares Deep500 to other benchmarking infrastructures with respect to the offered functionalities. Deep500 is the only system that focuses on performance, accuracy, and convergence, while simultaneously offering a wide spectrum of metrics and criteria for benchmarking, enabling customizability of design, and considering a diversity of workloads.

System	Operators		Networks				Training				Dist. Training		
	Site	Cus	Def	Eag	Com	Trn	Def	Opt	Cus	PS	Dec	Any	Cus
(L) cuDNN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(L) MKL-DNN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) TensorFlow [1]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) Caffe_Caffe2 [20]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) MXNet [10, 34]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) CNTK [46]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) Theano [6]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) Chainer (0.8) [43]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) Chainer [27]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) DL4j [42]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) DSSSE [1]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) PaddlePaddle [9]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(F) HMMT [1]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(E) Xeon [8]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(E) Horovod [41]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(E) TransLayer [14]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(E) Lasagne	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(E) TFLearn [11]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Integration within Deep500 [This work]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE I: An overview of DL frameworks, related systems that can be integrated within Deep500, and the advantages of each integration. Each column is a specific feature/functionality, they are explained in more detail in background (Fig. 8). Site: Standard Operator, Cus: Customizable (without full recompilation), Def: Default Execution Mode, Eag: Eager Execution Mode (also called "eager-by-default"), Com: Network Compilation, Trn: Trainable, Def: Default Network Integration, Opt: Standard Optimizers, PS: Parameter Server, Dec: Decentralized, Any: Asynchronous SGD, UR: Update Rule Optimizers, C: A given system does offer a given feature, A: A given system offers a given feature in a limited way, A: A given system does not offer a given feature, (L): a library, (F): a framework, (E): a hardware. Native system support for a category of features: *none*, *partial*, *full*.



(a) Strong scaling (Wide ResNet 28x10).

Fig. 11: Scaling Analysis of Level 3

Turning 180-degree – Deep Learning for HPC – Neural Code Comprehension

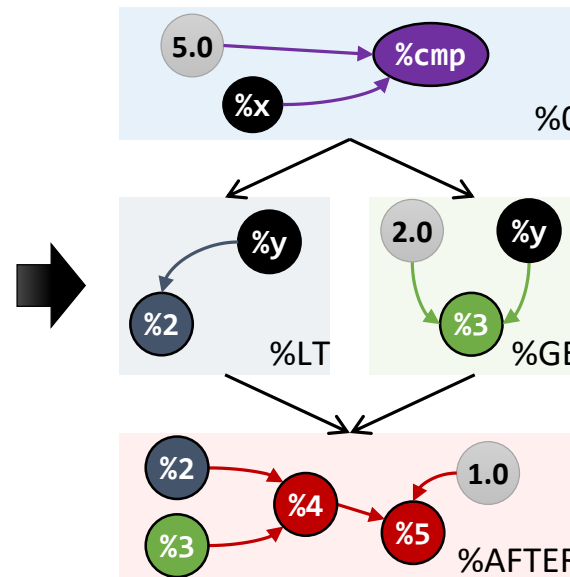
- In 2017, GitHub reports 1 billion git commits in 337 languages!
- Can DNNs *understand* code?
- Previous approaches read the code directly → suboptimal (loops, functions)

```
double thres = 5.0;           %cmp = fcmp olt double %x, 5.0

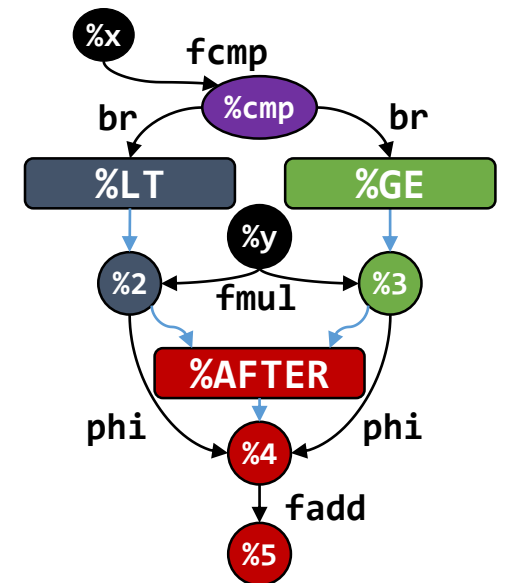
if (x < thres)                br i1 %cmp, label %LT, label %GE
    x = y * y;                 LT:
else                            %2 = fmul double %y, %y
    x = 2.0 * y;               GE:
                                %3 = fmul double 2.0, %y

x += 1.0;                       AFTER:
                                %4 = phi double [%2,%LT], [%3,%GE]
                                %5 = fadd double %4, 1.0
```

C/C++ FORTRAN
 Python Java
 CUDA OpenCL
 ...



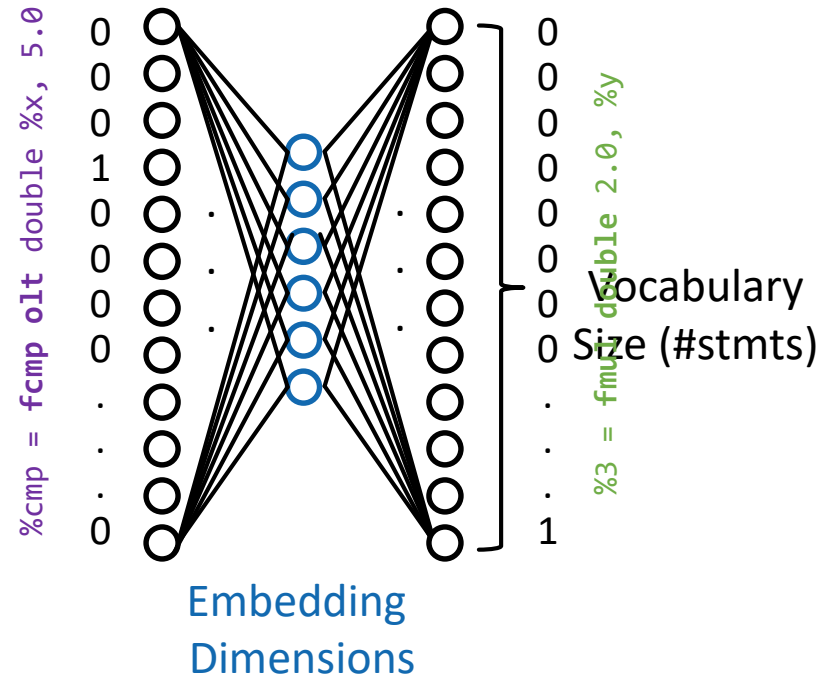
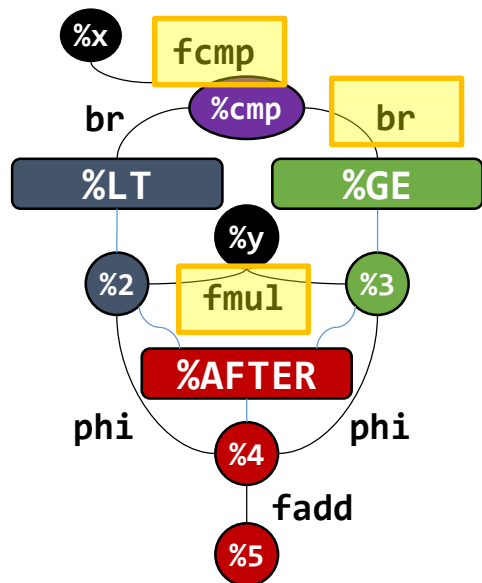
Dataflow (basic blocks)



Contextual Flow Graph

Deep Learning for HPC – Neural Code Comprehension

- Embedding space (using the Skip-gram model)



Deep Learning for HPC – Neural Code Comprehension

Embedding space (u)

Table 3: Algorithm classification test accuracy

Metric	Surface Features [46] (RBF SVM + Bag-of-Trees)	RNN [46]	TBCNN [46]	inst2vec
Test Accuracy [%]	88.2	84.8	94.0	94.83

Predicts which device is faster (CPU or GPU)

Table 4: Heterogeneous device mapping results

Architecture	Prediction Accuracy [%]			Speedup		
	Grewe et al. [27]	DeepTune [17]	inst2vec	Grewe et al.	DeepTune	inst2vec
AMD Tahiti 7970	73.38	83.68	82.79	2.91	3.34	3.42
NVIDIA GTX 970	72.94	80.29	81.76	1.26	1.41	1.39

Optimal tiling

Table 5: Speedups achieved by coarsening threads

Computing Platform	Magni et al. [43]	DeepTune [17]	DeepTune-TL [17]	inst2vec
AMD Radeon HD 5900	1.21	1.10	1.17	1.25
AMD Tahiti 7970	1.01	1.05	1.23	1.07
NVIDIA GTX 480	0.86	1.10	1.14	1.02
NVIDIA Tesla K20c	0.94	0.99	0.93	1.03

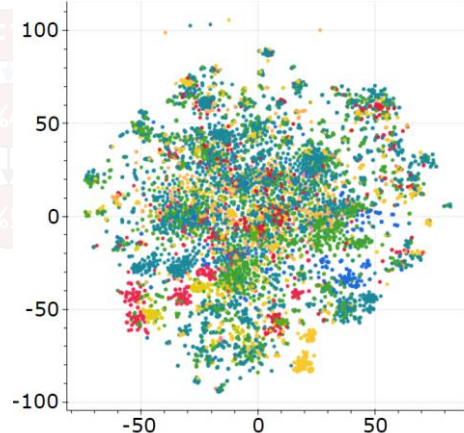


Table 2: Analogy and test scores for inst2vec

Context Size	Syntactic Analogies		Semantic Analogies		Semantic Distance Test
	Types	Options	Conversions	Data Structures	
1	101 (18.04%)	13 (24.53%)	100 (6.63%)	3 (37.50%)	60.98%
2	226 (40.36%)	45 (84.91%)	134 (8.89%)	7 (87.50%)	79.12%
3	125 (22.32%)	24 (45.28%)	48 (3.18%)	7 (87.50%)	62.56%

HPC for Deep Learning – Summary

- **Deep learning is HPC – very similar computational structure, in fact very friendly**
 - Amenable to specialization, static scheduling, all established tricks - microbatching
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency

- Bounded synchronous SGD
- Central vs. distributed parameter server
- EASGD to ensemble learning

Parameter Accuracy

- Lossless compression of gradient updates
- Quantization of gradient updates
- Sparsification of gradient updates

- **Very different environment from traditional HPC**
 - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**

T. Hoefler: “Twelve ways to fool the masses when reporting performance of deep learning workloads”

(my humorous guide to floptimization in deep learning will be published this week during IPAM)

How to not do this

“Twelve ways to fool the masses when reporting performance of deep learning workloads”
(my humorous guide to floptimize deep learning, blog post Nov. 2018)

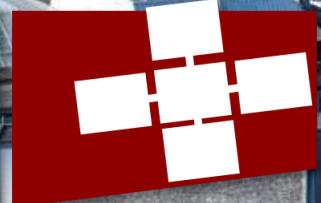


T. HOEFLER

Twelve ways to fool the masses when reporting performance of deep learning workloads! (not to be taken too seriously)

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

<http://htor.inf.ethz.ch/blog/index.php/2018/11/08/twelve-ways-to-fool-the-masses-when-reporting-performance-of-deep-learning-workloads/>



Deep learning and HPC

- **Deep learning is HPC**
 - In fact, it's probably (soon?) bigger than traditional HPC
Definitely more money ...
- **Interest in the HPC community is tremendous**
 - Number of learning papers at HPC conferences seems to be growing exponentially
Besides at SC18, whut!?
- **Risk of unrealism**
 - HPC people know how to do HPC
 - And deep learning is HPC, right?
Not quite ... while it's really similar (tensor contractions)
But it's also quite different!

Yann LeCun's conclusion slide yesterday!

Hardware Requirement

- ▶ DL Research and Development: HPC!
 - ▶ Compute power, flexibility, programmability, numerical accuracy
 - ▶ Cluster of nodes with multiple GPGPU. 32bit FP, low-latency network
- ▶ Training Production systems
 - ▶ High speed, 16bit FP usually enough.
 - ▶ High parallelism less crucial (beyond one or a few nodes)
- ▶ Inference on Servers and embedded systems (e.g. cars)
 - ▶ Low power dissipation, reduced precision, exotic number systems
 - ▶ Enormous volumes! Facebook today: 300e12 predictions per day.
- ▶ Inference on mobile devices and consumer electronics
 - ▶ Super low power dissipation, exotic number systems (e.g. Log)
 - ▶ Very low cost. AR/VR, cameras, appliances, toys....

“Statistical performance” vs. “hardware performance”

■ Tradeoffs between those two

- Very weird for HPC people – we always operated in double precision
Mostly out of fear of rounding issues

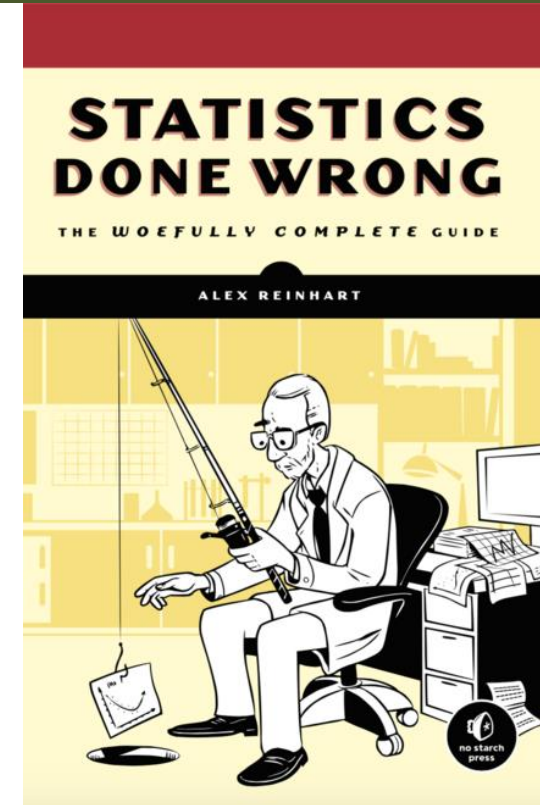
■ Deep learning shows how little accuracy one can get away with

- Well, examples are drawn randomly from some distribution we don't know ...
- Usually, noise is quite high ...
- So the computation doesn't need to be higher precision than that noise

Pretty obvious! In fact, it's similar in scientific computing but in tighter bounds and not as well known

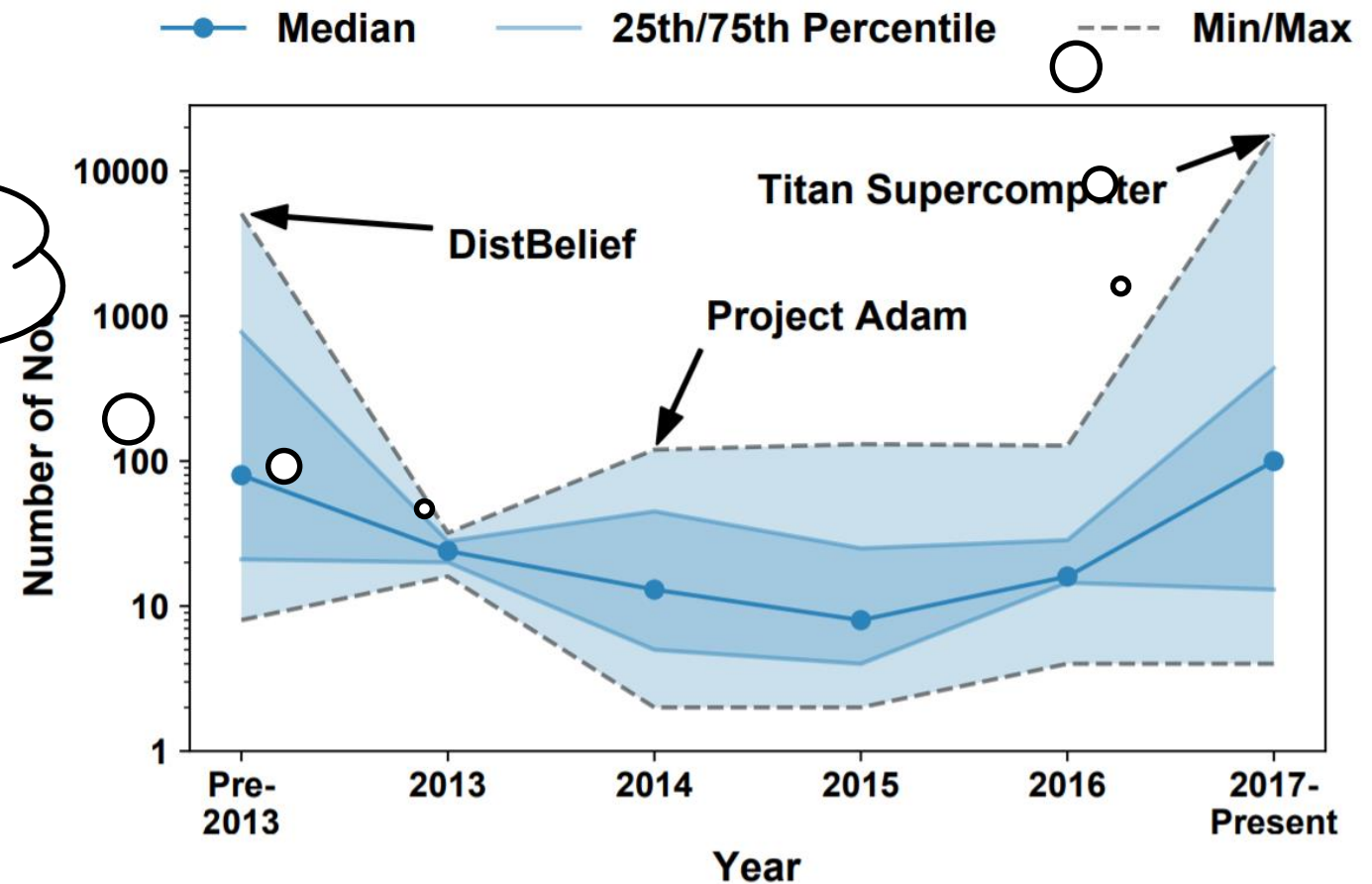
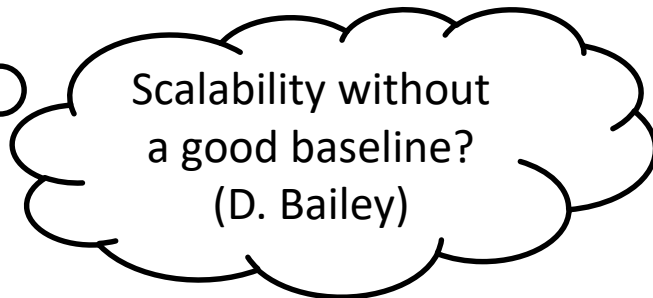
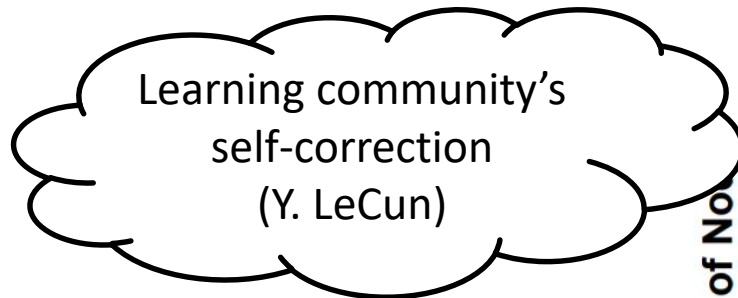
■ But we HPC folks like flop/s! Or maybe now just ops or even aiops? Whatever, fast compute!

- A humorous guide to **floptimization**
- Twelve rules to help present your (not so great?) results in a much better light



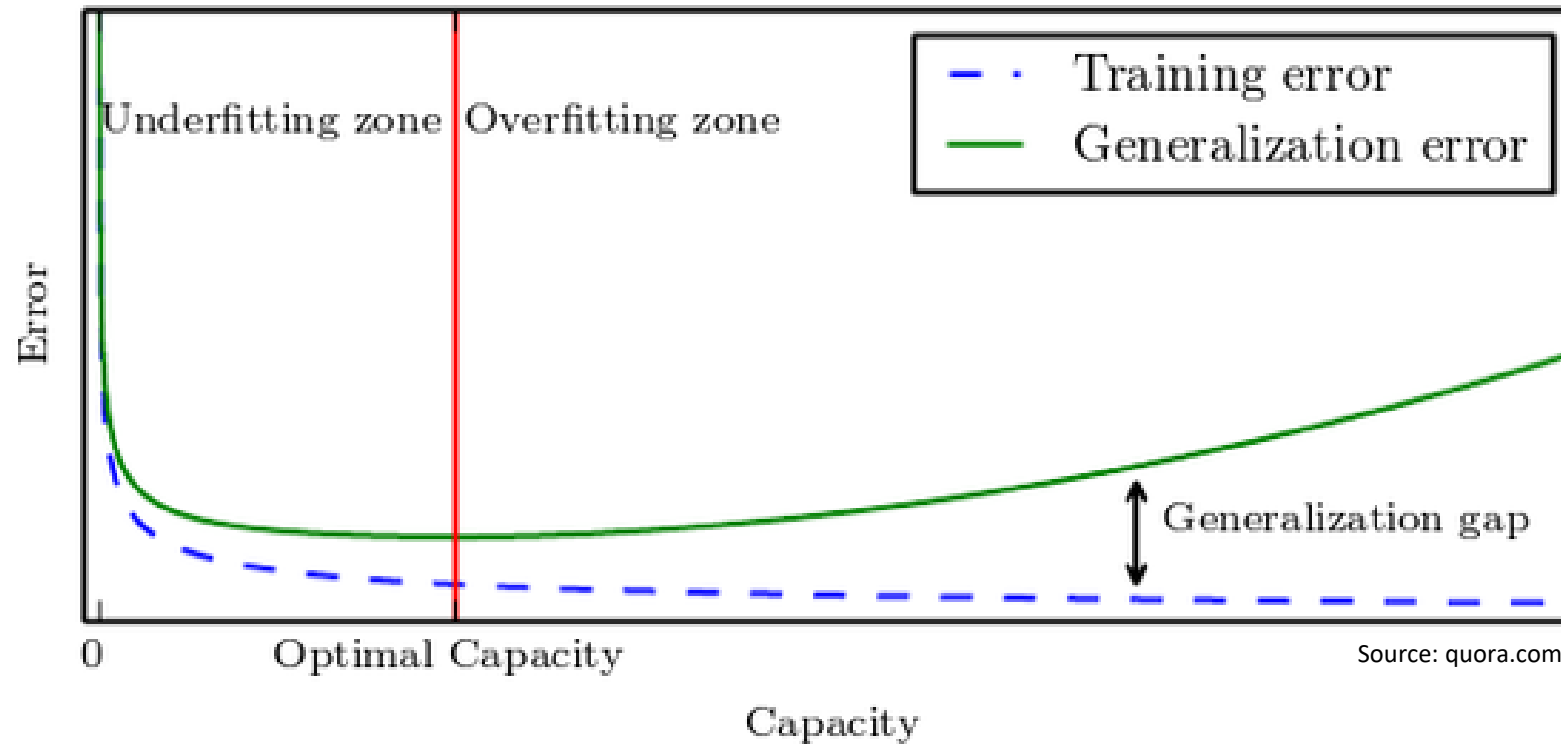
1) Ignore accuracy when scaling up!

- **Too obvious for this audience**
 - Was very popular in 2015!
- **Surprisingly many (still) do this**



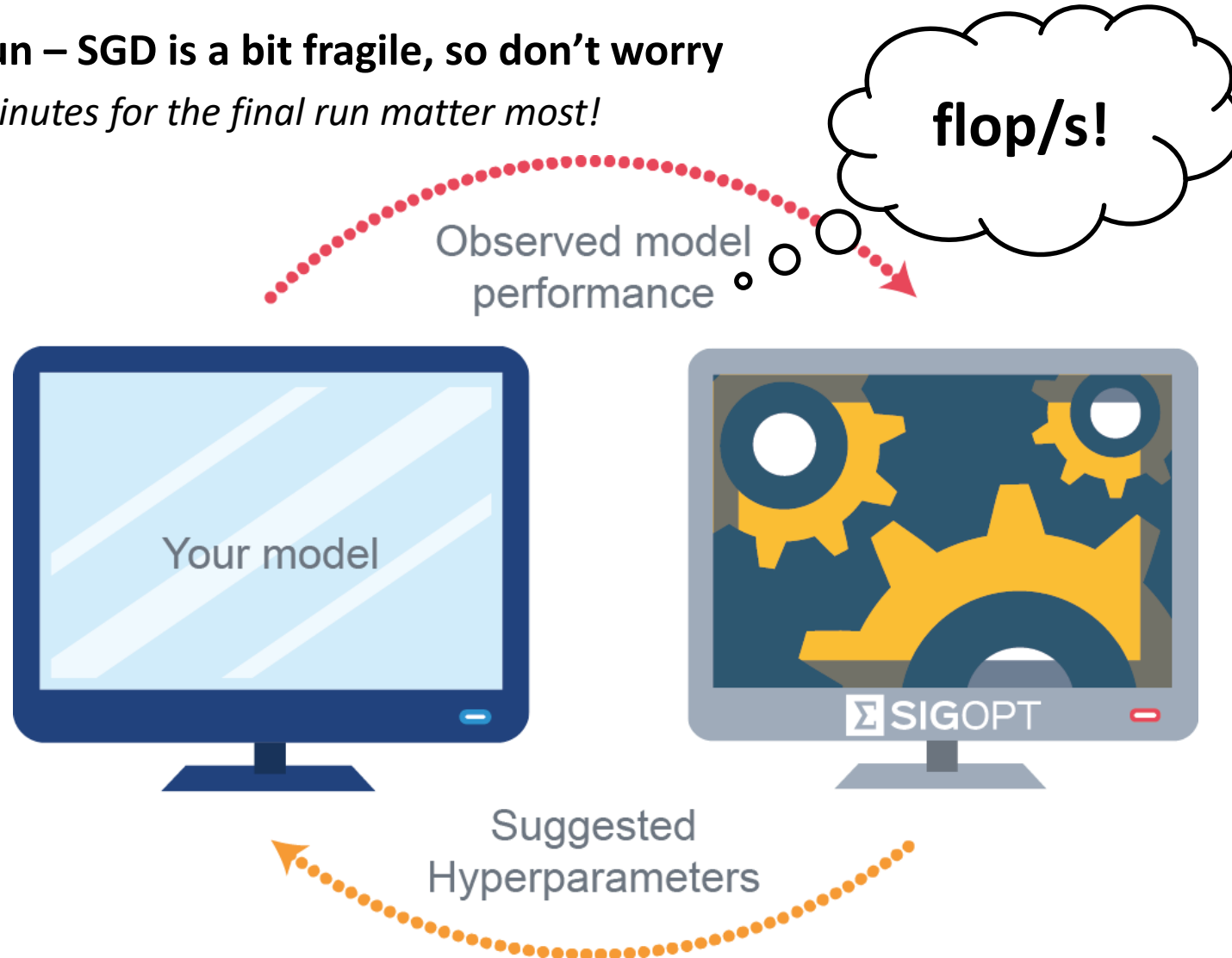
2) Do not report test accuracy!

- Training accuracy is sufficient isn't it?



3) Do not report all training runs needed to tune hyperparameters!

- Report the best run – SGD is a bit fragile, so don't worry
At the end, the minutes for the final run matter most!



4) Compare outdated hardware with special-purpose hardware!

- Tesla K20 in 2018!?

Even though the older machines would win the beauty contest!



VS.



5) Show only kernels/subsets when scaling!

- Run layers or communication kernels in isolation
 - Avoids issues with accuracy completely 😊
Doesn't that look a bit like GoogLeNet?

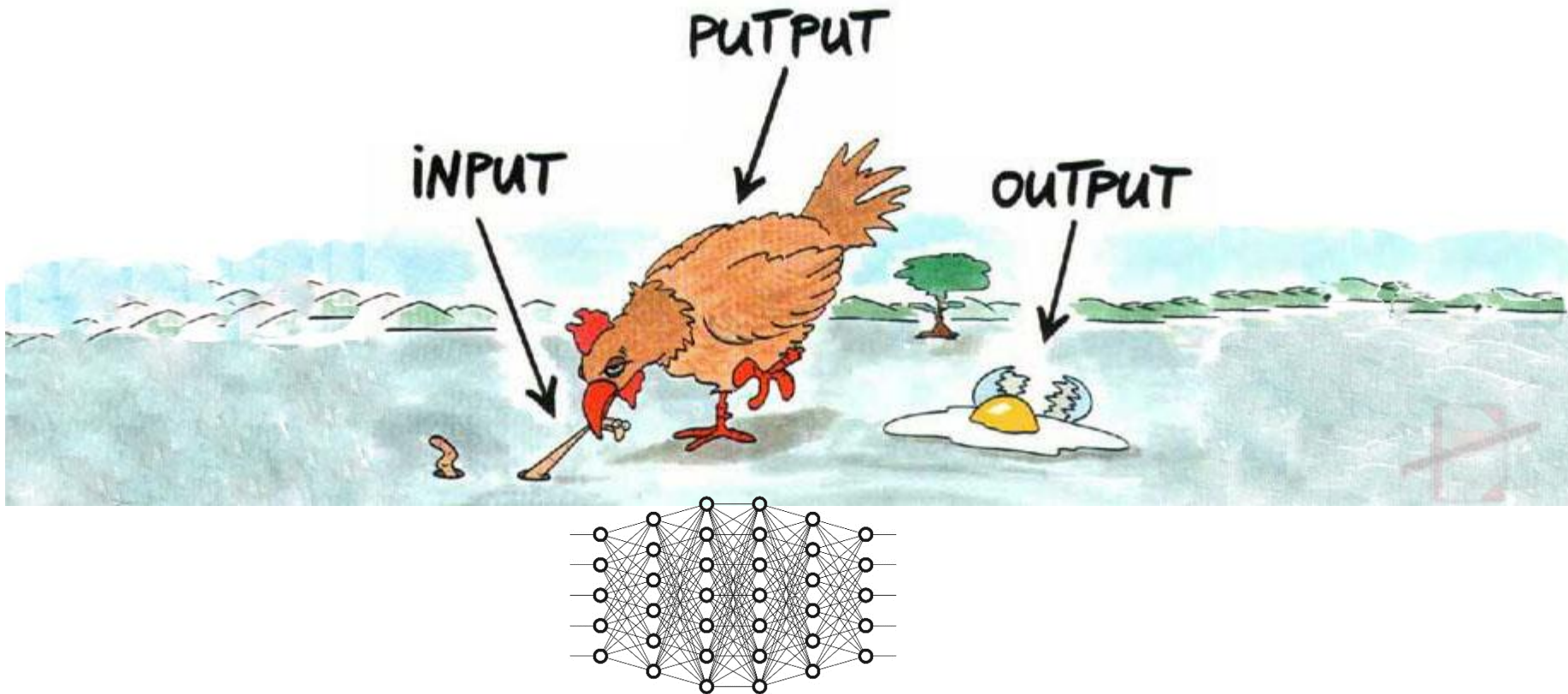


VS.



6) Do not consider I/O!

- Reading the data? Nah, make sure it's staged in memory when the benchmark starts!



7) Report highest ops numbers (whatever that means)!

- Yes, we're talking ops today, 64-bit flops was so yesterday!
 - If we don't achieve a target fast enough, let's redefine it!
And never talk about how many more of those ops one needs to find a solution, it's all about the rate, op/s!
- **Actually, my laptop achieves an "exaop":**
 - each of the $3e9$ transistors switching a binary digit each at $2.4e9$ Hz



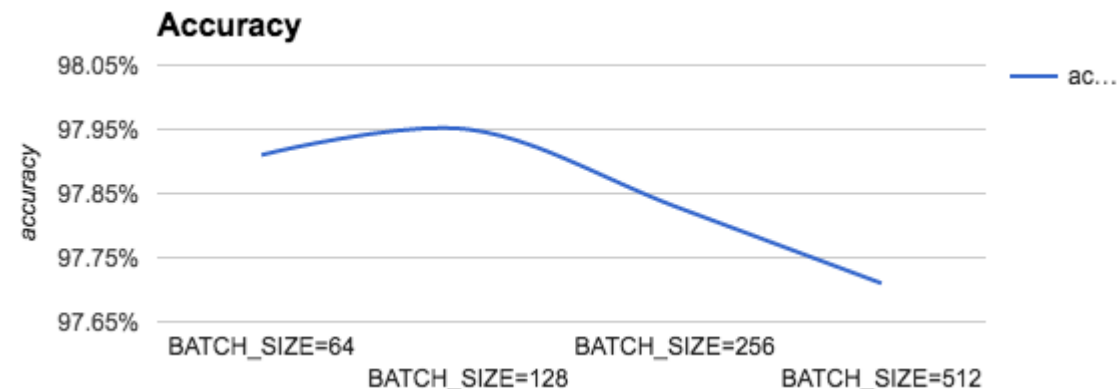
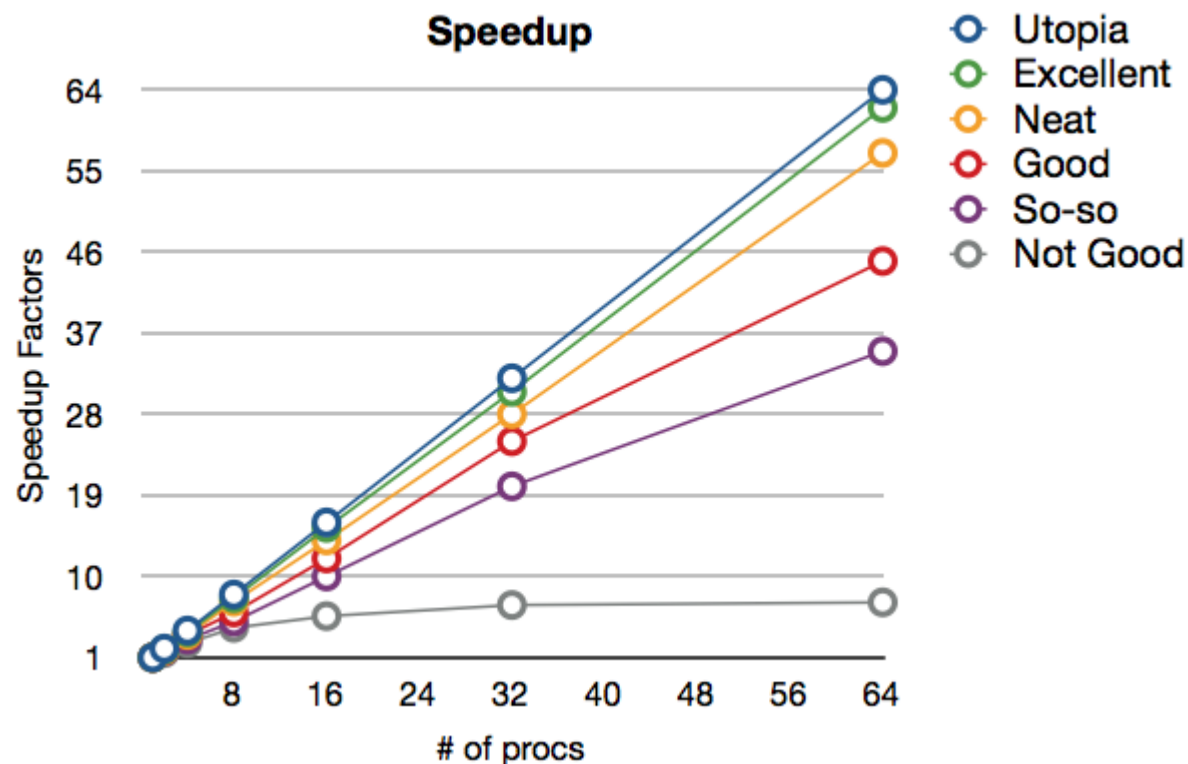
VS.



8) Show performance when enabling option set A and show accuracy when enabling option set B!

- Pretty cool idea isn't it? Hyperparameters sometimes conflict

So always tune the to show the best result, whatever the result shall be!



9) Train on (unreasonably) large inputs!

- The pinnacle of floptimization! Very hard to catch!
But Dr. Catlock Holmes below can catch it.



Low-resolution cat (244x244 – 1 Gflop/example)

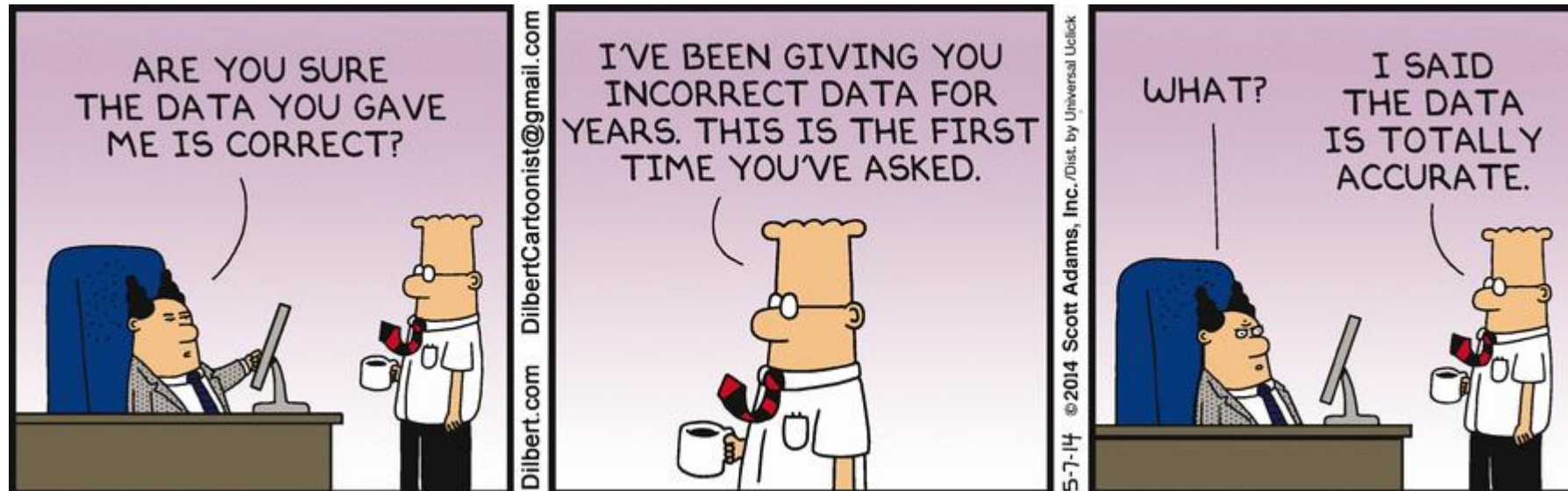
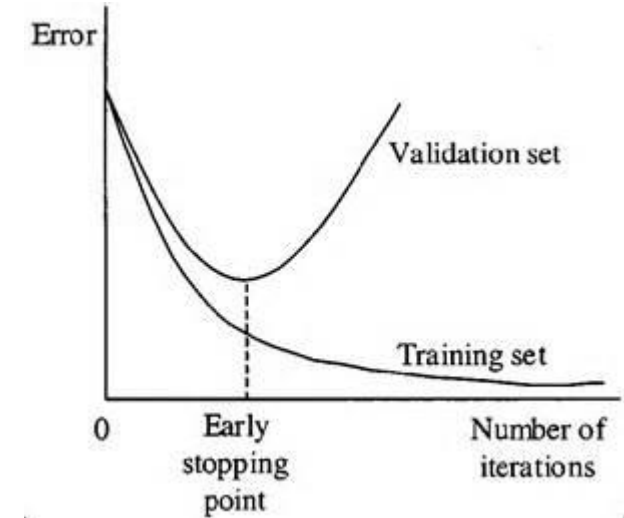
VS.



High-resolution cat (8kx8k – 1 Tflop/example)

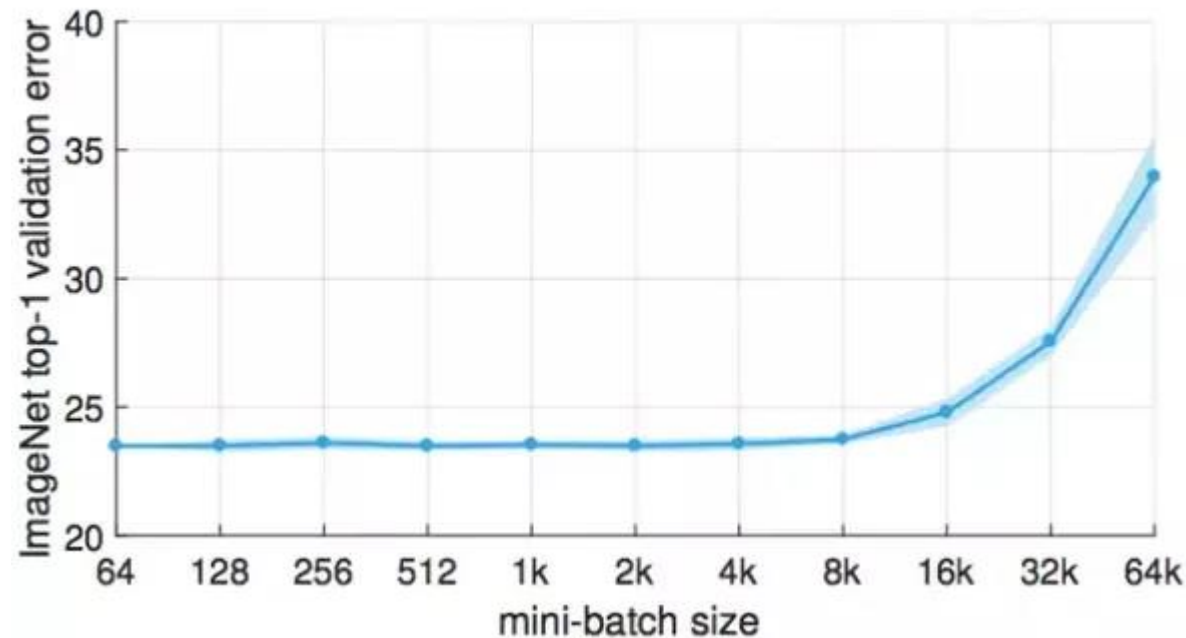
10) Run training just for the right time!

- Train for fixed wall-time when scaling processors
 - so when you use twice as many processors you get twice as many flop/s!
But who cares about application speedup?



11) Minibatch sizing for fun and profit – weak vs. strong scaling.

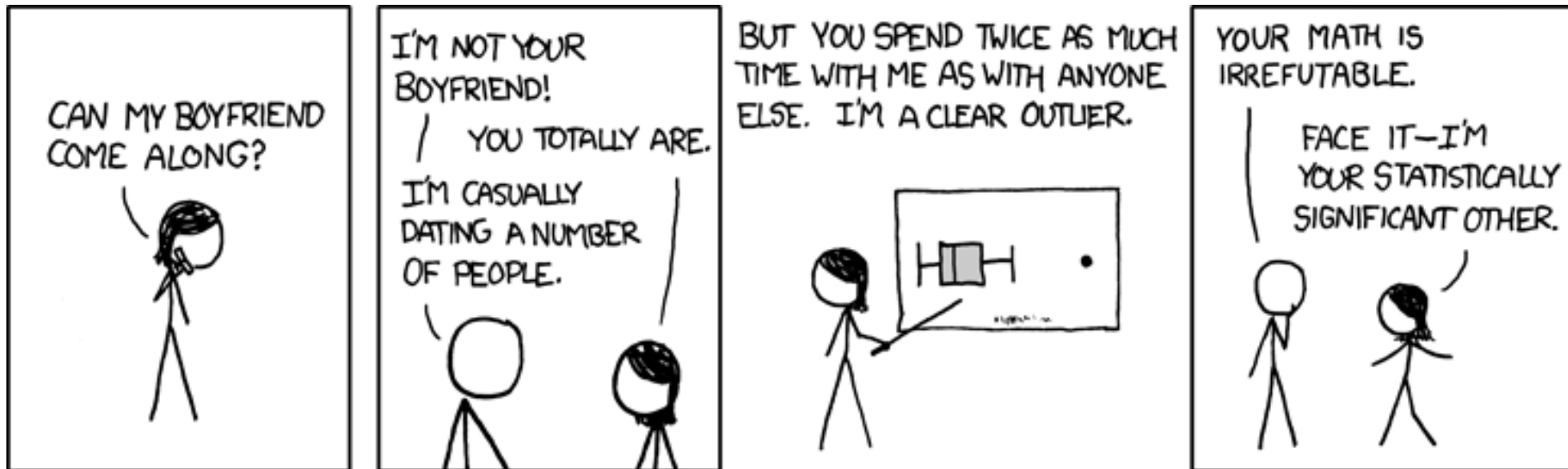
- All DL is strong scaling – limited model and limited data
- So just redefine the terms relative to minibatches:
 - Weak scaling keeps MB size per process constant – overall grows (less iterations per epoch, duh!)
 - Strong scaling keeps overall MB size constant (better but harder)
- Microbatching is not a problem!



12) Select carefully how to compare to the state of the art!

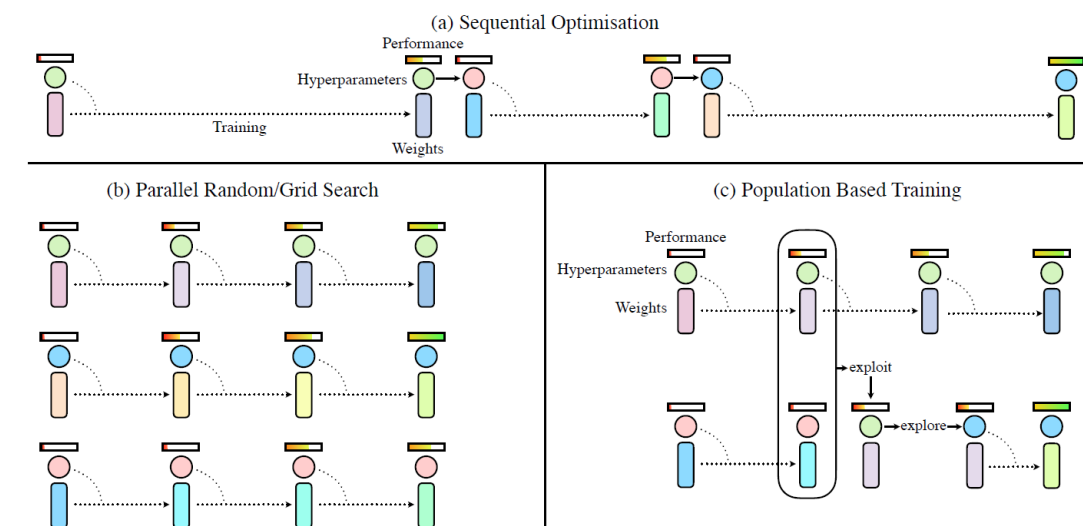
- Compare either time to solution or accuracy if both together don't look strong!

There used to be conventions but let's redefine them.

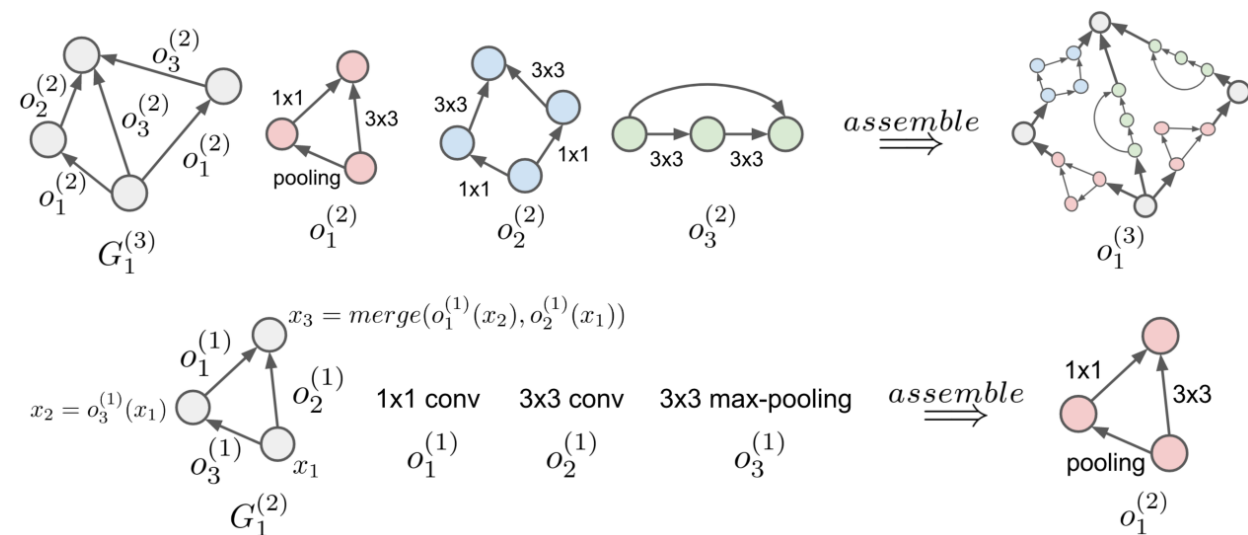


Hyperparameter and Architecture search

- 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]

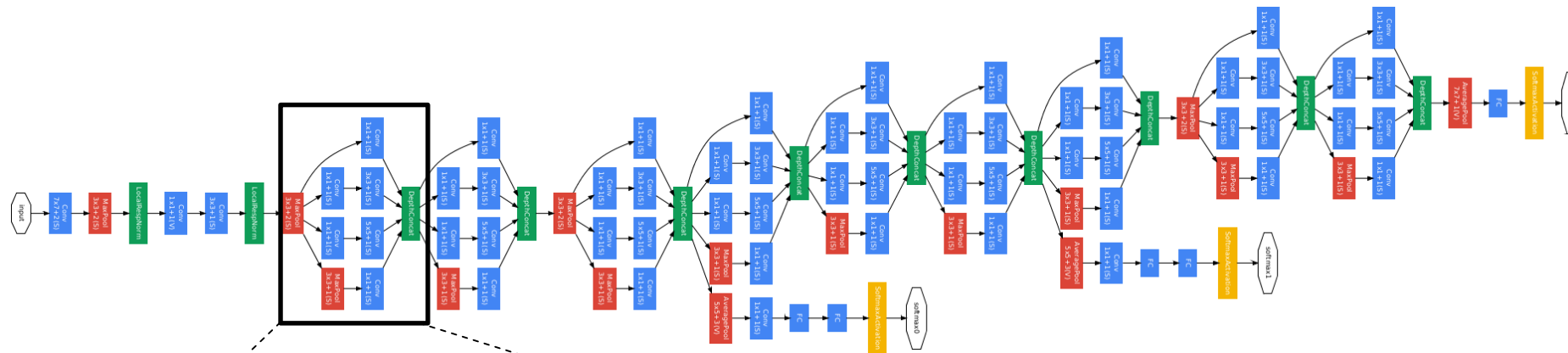
[1] M. Jaderberg et al.: Population Based Training of Neural Networks, arXiv 2017

[2] E. Real et al.: Regularized Evolution for Image Classifier Architecture Search, arXiv 2018

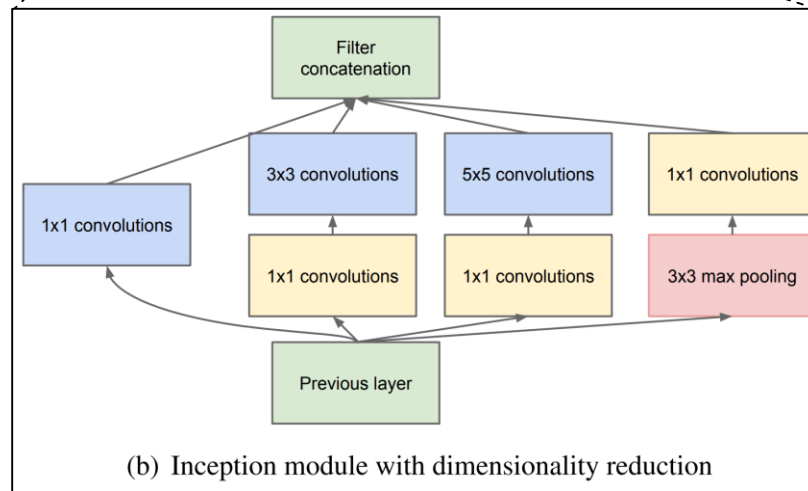
[3] P. R. Lorenzo et al.: Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization, GECCO'17

[4] H. Liu et al.: Hierarchical Representations for Efficient Architecture Search, ICLR'18

GoogLeNet in more detail



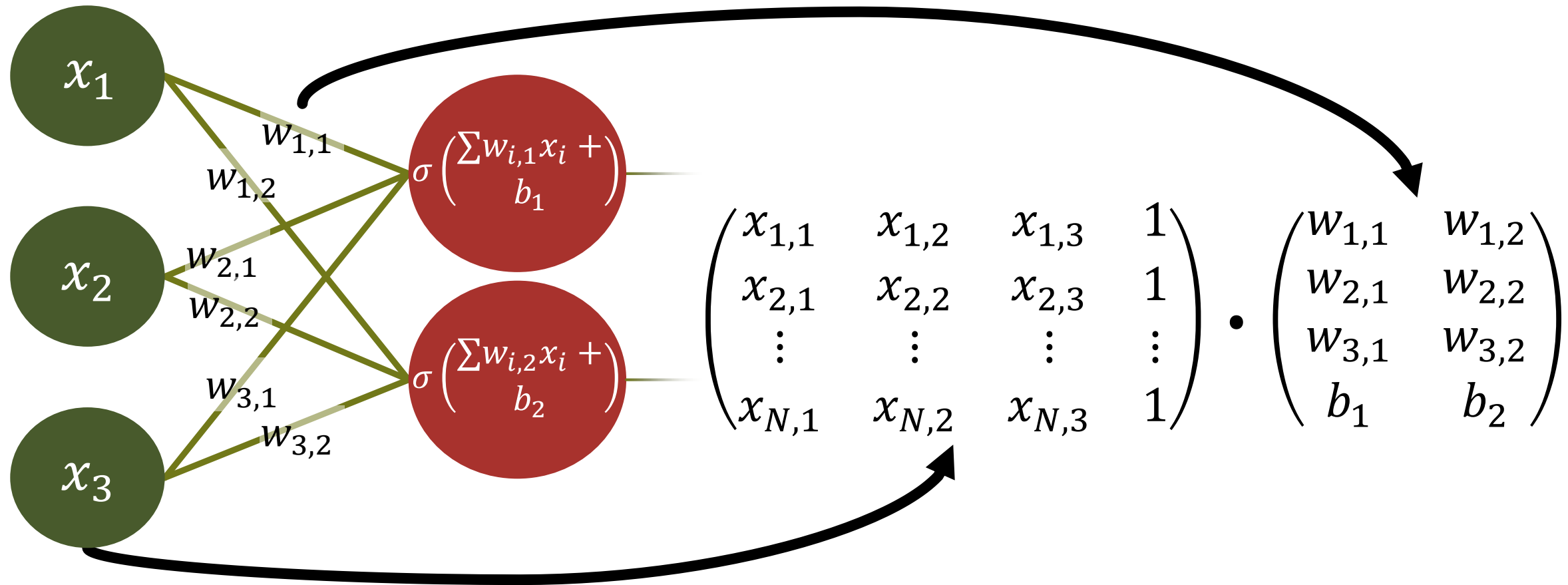
- ~6.8M parameters
- 22 layers deep



(b) Inception module with dimensionality reduction

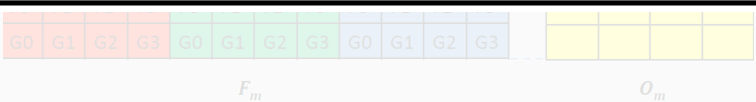
Computing fully connected layers

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



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