T. Hoefler

High-Performance Communication in Machine Learning

Keynote at the 13th International Conference on Parallel Processing and Applied Mathematics

With contributions from Tal Ben-Nun, Dan Alistarh, Shoshana Jakobovits, Cedric Renggli, and others at SPCL, IST Austria, and Tokyo Tech

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.

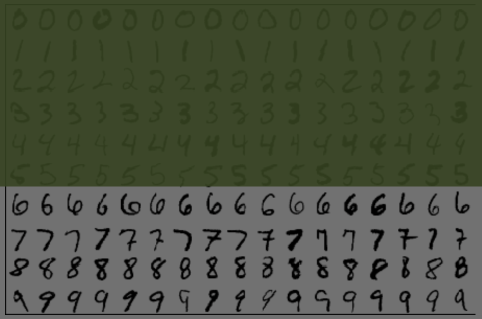
1 INTRODUCTION

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?

Hey Cortana
Hey Siri
Google now
XBOX
amazon alexa
skype Translate
Bai du 百度
...

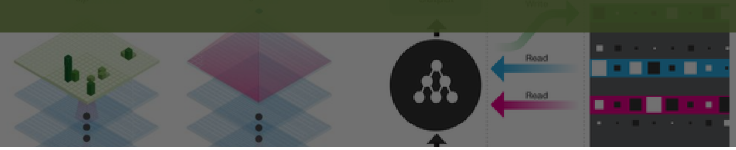
Digit Recognition



Object Classification
Segmentation
Image Captioning

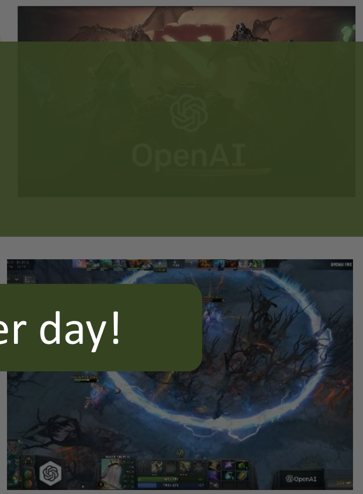
Gameplay AI
Translation

Neural Computers



Gigantic Language Models

Towards Real Physics



A very active area of research!

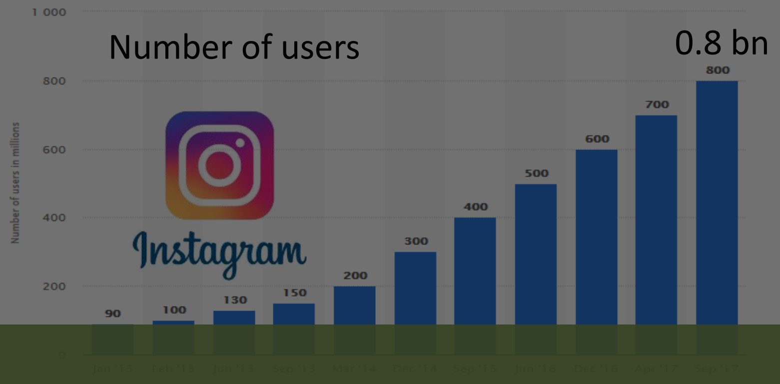
23 papers per day!

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

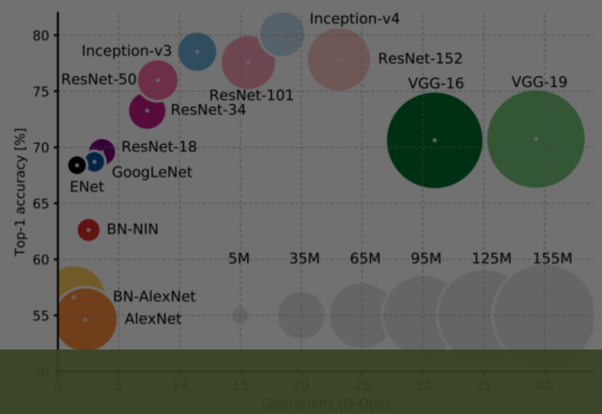
number of papers per year

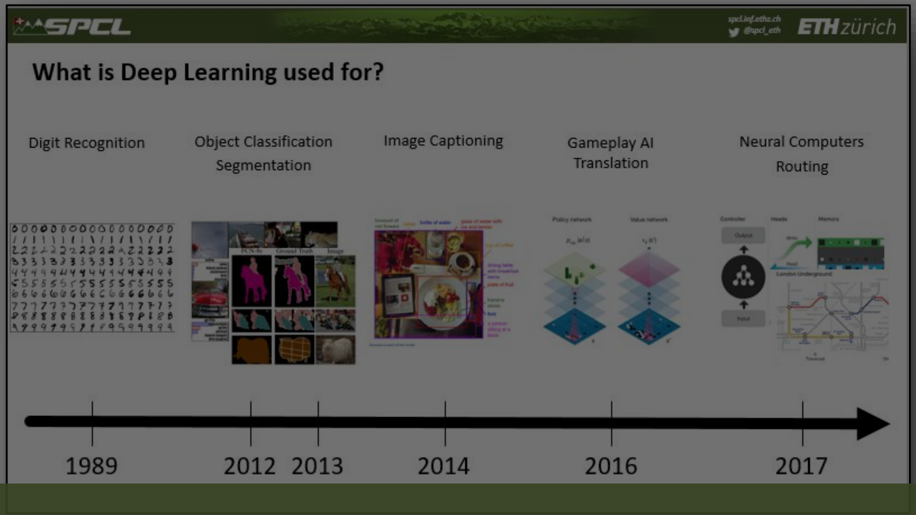
1989 2012 2013 2014 2016 2017 2018 2019

How does Deep Learning work?

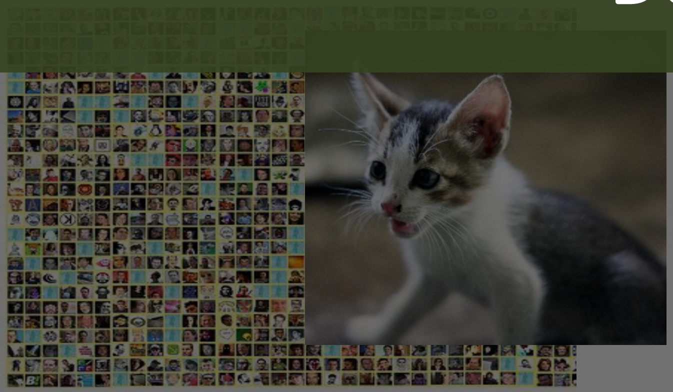


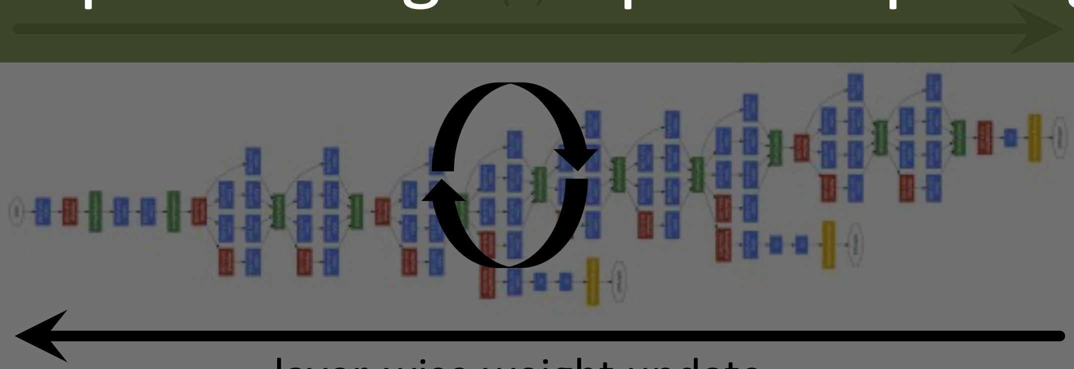
Canziani et al. 2017





Deep Learning is Supercomputing!





layer-wise weight update

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

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Bicycle	0.00
Truck	0.00

- ImageNet (1k): 180 GB
- ImageNet (22k): A few TB
- Industry: Much larger

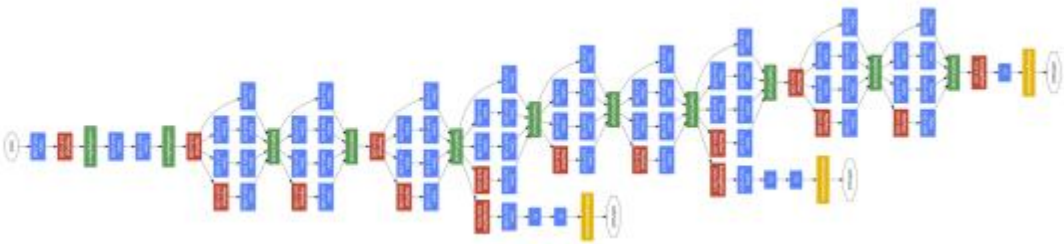
- 100-200 layers deep
- ~100M-2B parameters
- 0.1-8 GiB parameter storage

- 10-22k labels
- growing (e.g., face recognition)
- weeks to train

A brief theory of supervised deep learning

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

$f(x)$



layer-wise weight update

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

$f(x): X \to Y$

network structure (fixed)

weights w (learned)

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

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

$f_1(x)$ | $f_2(f_1(x))$ | … | $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

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

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

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

Stochastic Gradient Descent

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

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

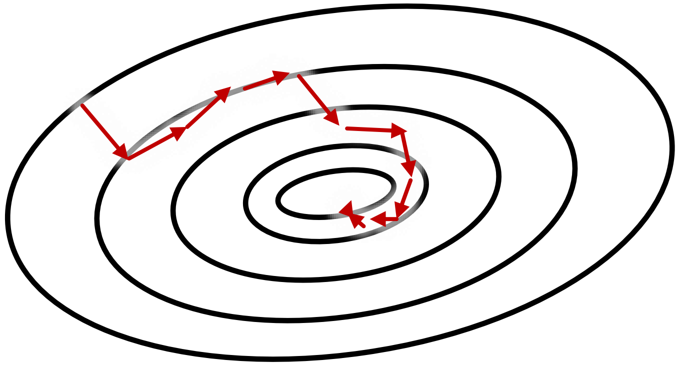
pooling

…

convolution 3

$f(x)$ fully connected

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

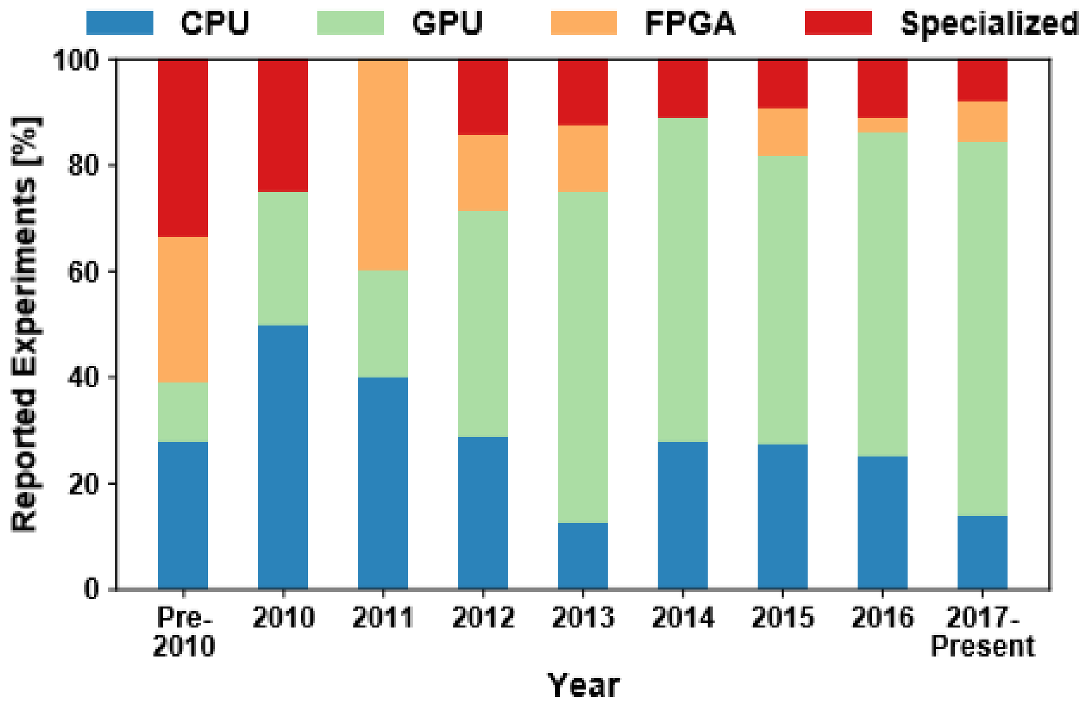


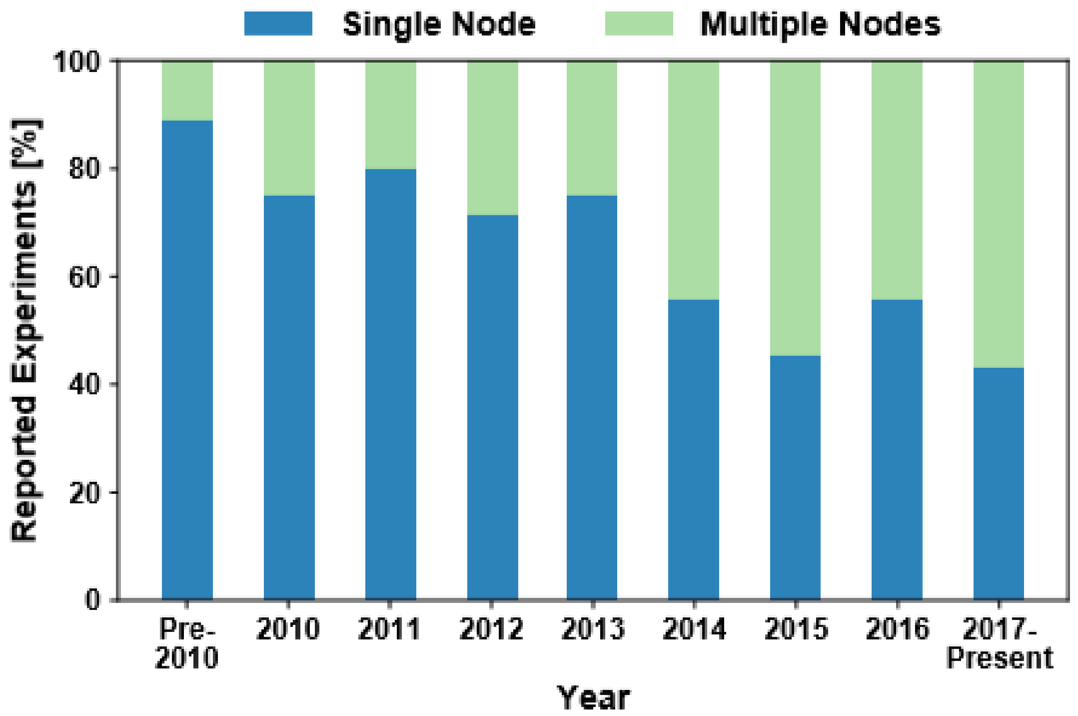
Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla\ell(w^{(t)}, z)$	$= w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$	
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$	
Nesterov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

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018

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



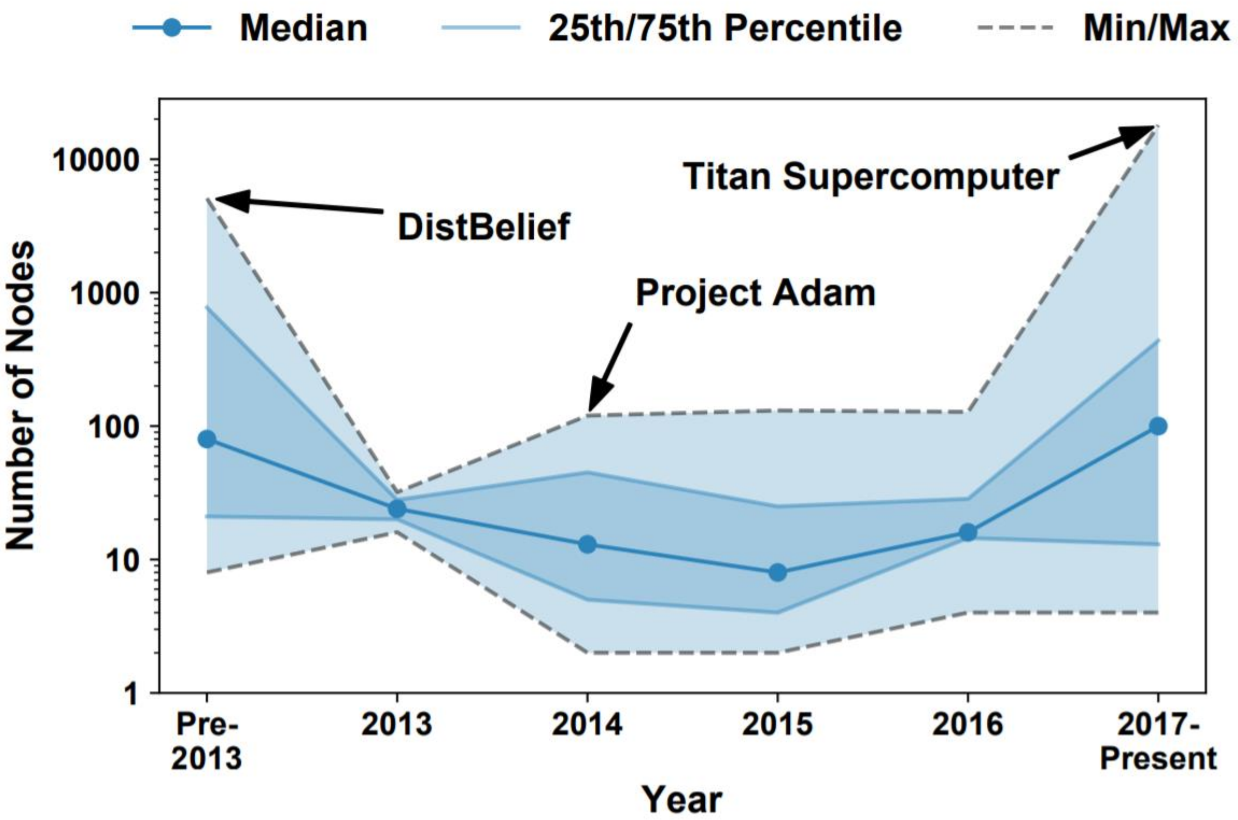


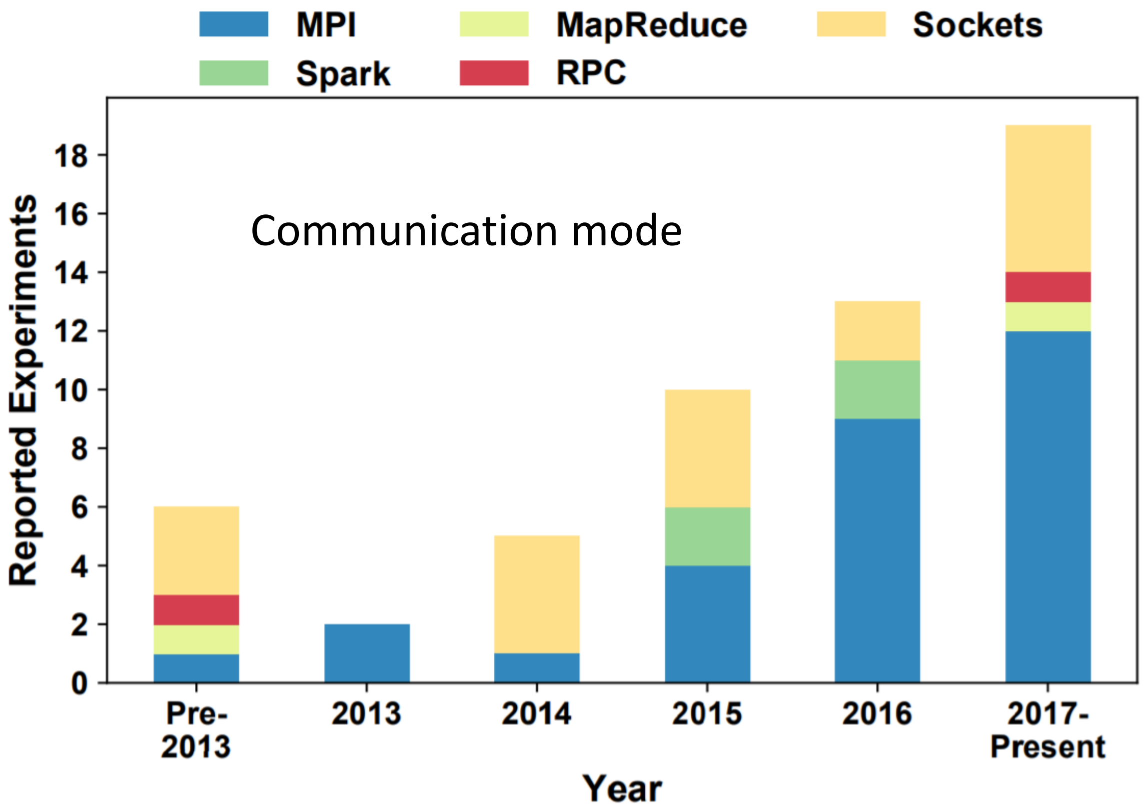
Deep Learning is largely on distributed memory today!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018

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

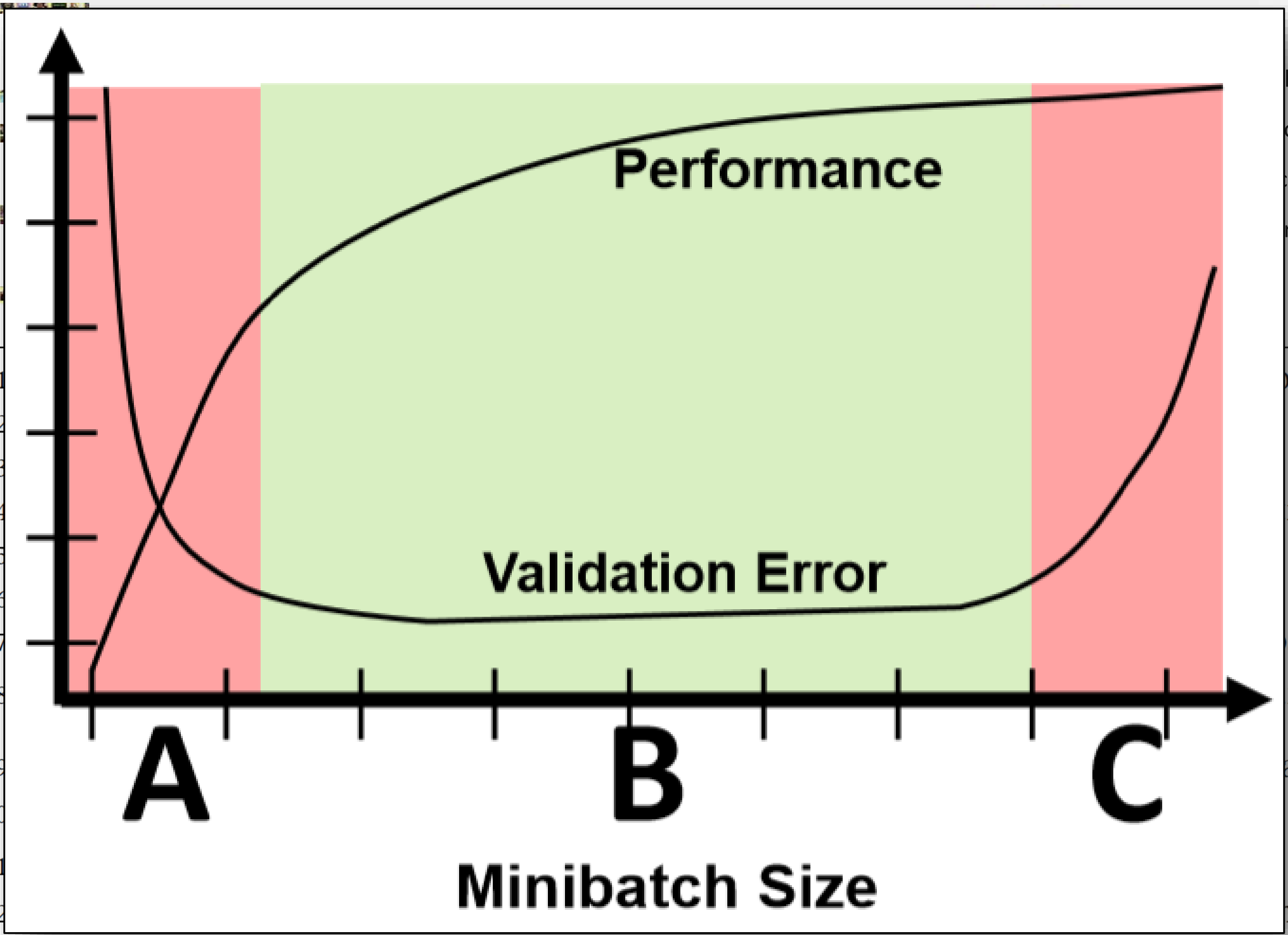




Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018

Minibatch Stochastic Gradient Descent (SGD)



Cat	0.54
Dog	0.28
lane	0.07
orse	0.04
ycle	0.03
ruck	0.02
	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Bicycle	0.00
Truck	0.00
	0.00

ndition
dataset
ion of ℓ
agation
r.t. data
ameters
nction u
13: **end for**

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018

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

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size

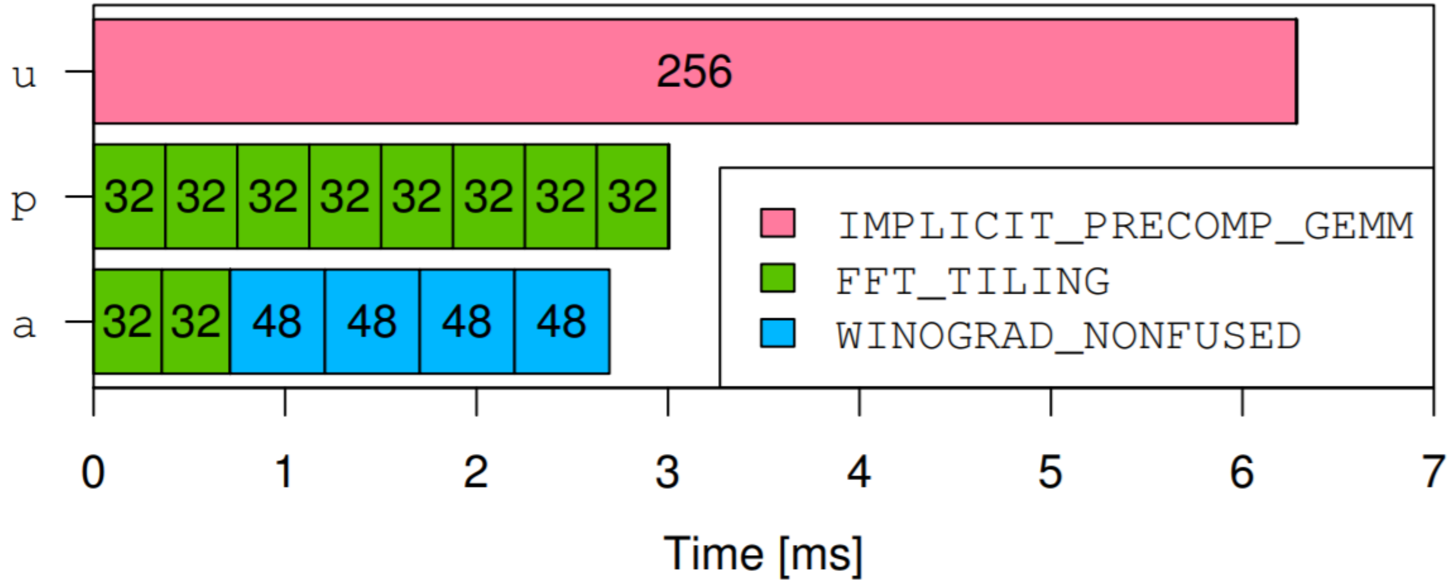
Fast (up to 4.54x faster on DeepBench)

Microbatching Strategy

none (undivided)

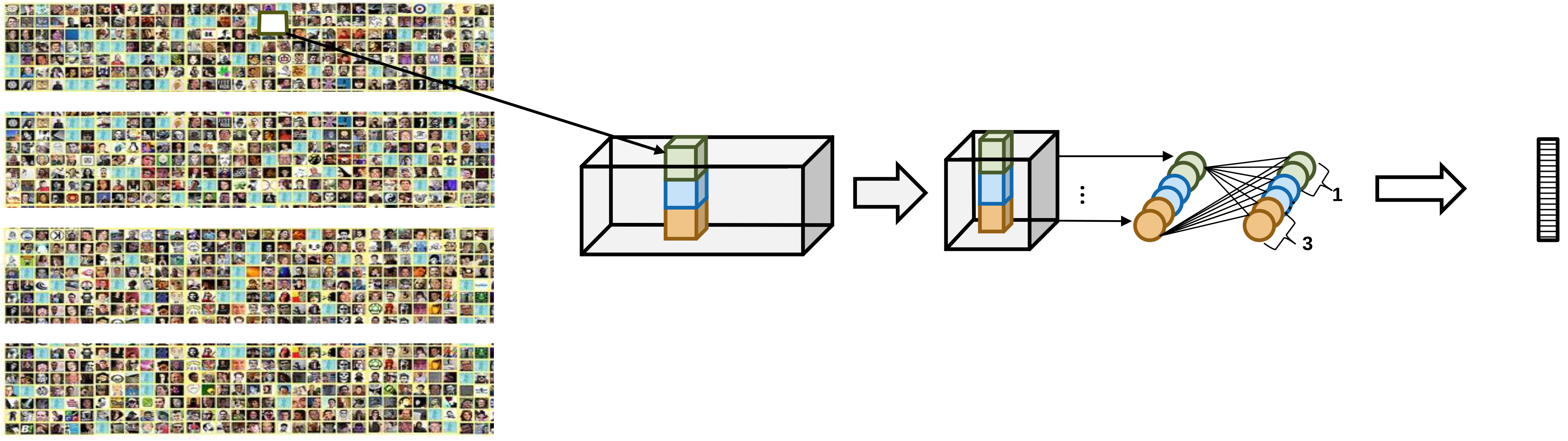
powers-of-two only

any (unrestricted)



Yosuke Oyama, Tal Ben-Nun, TH and Satoshi Matsuoka: μ-cuDNN: Accelerating Deep Learning Frameworks with Micro-Batching, Cluster 2018

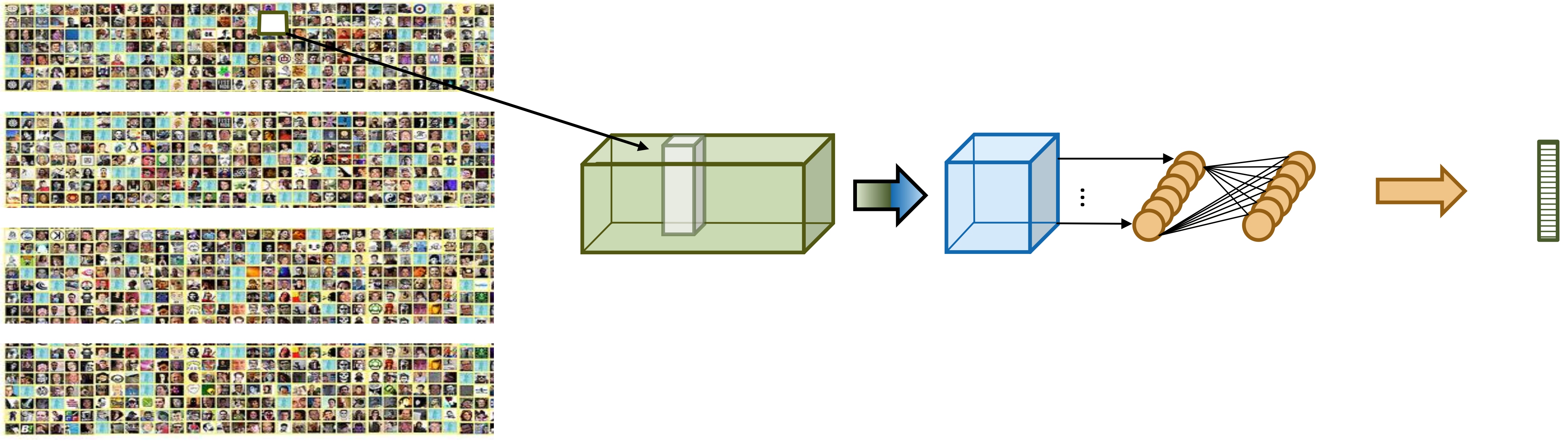
Model parallelism – limited by network size



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

U.A. Muller and A. Gunzinger: Neural Net Simulation on Parallel Computers, IEEE Int'l Conf. on Neural Networks 1994

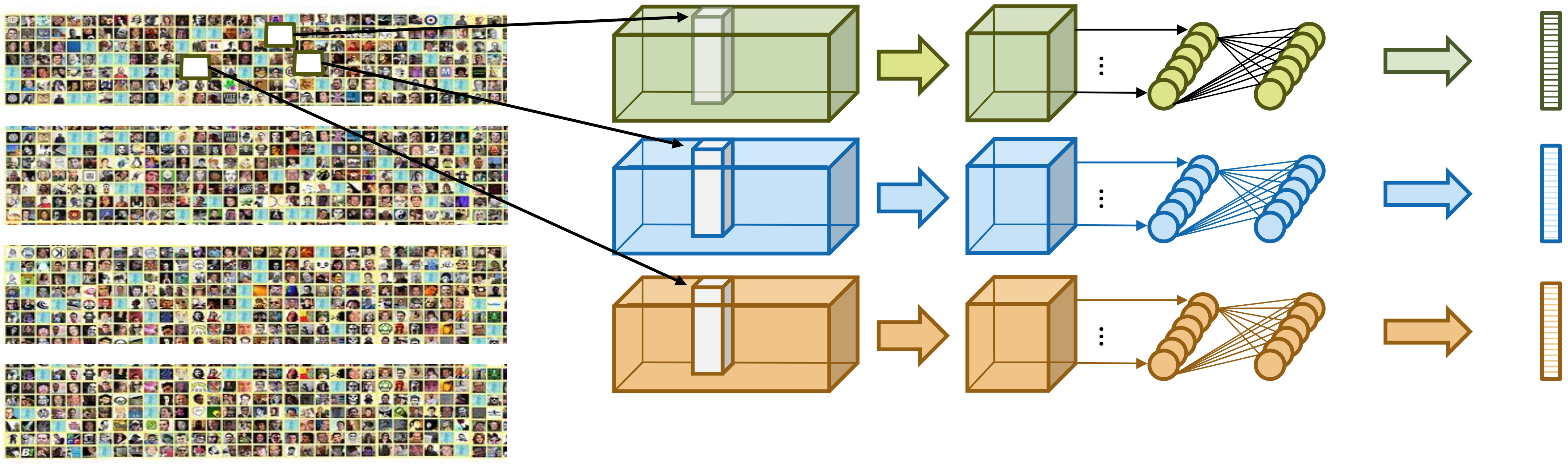
Pipeline parallelism – limited by network size



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

G. Blelloch and C.R. Rosenberg: Network Learning on the Connection Machine, IJCAI'87

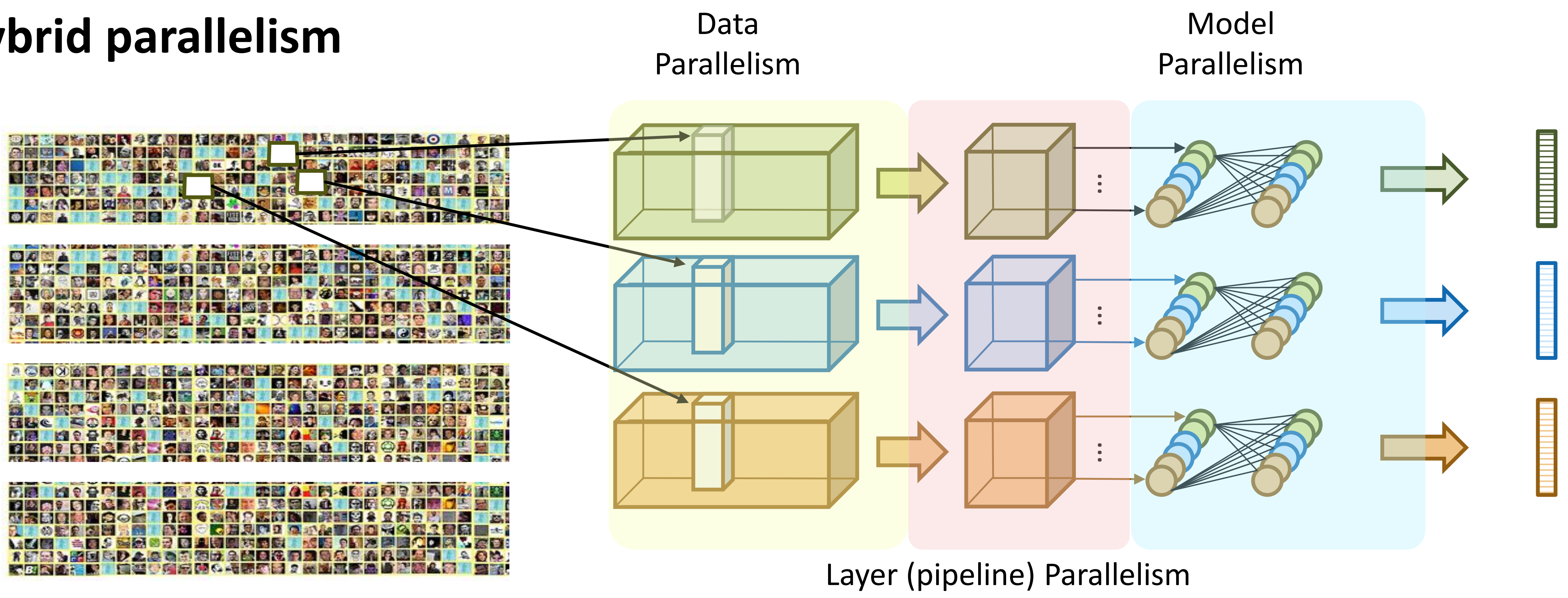
Data parallelism – limited by batch-size



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

X. Zhang et al.: An Efficient Implementation of the Back-propagation Algorithm on the Connection Machine CM-2, NIPS'89

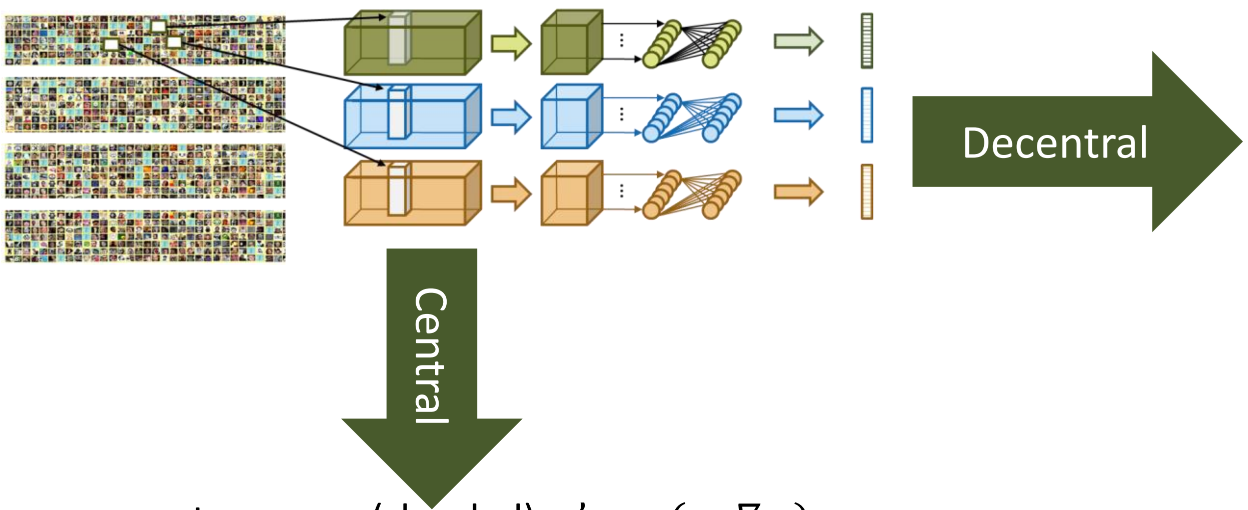
Hybrid parallelism



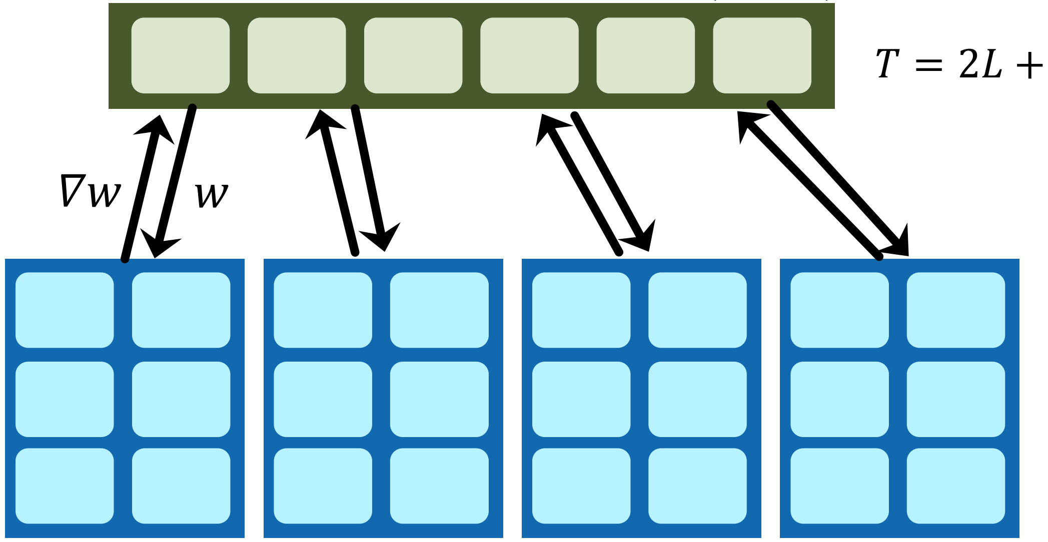
- **Layers/parameters can be distributed across processors**
- **Can distribute minibatch**
- **Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)**
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

A. Krizhevsky: One weird trick for parallelizing convolutional neural networks, arXiv 2014
J. Dean et al.: Large scale distributed deep networks, *NIPS'*12.
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, arXiv Feb 2018

Updating parameters in distributed data parallelism



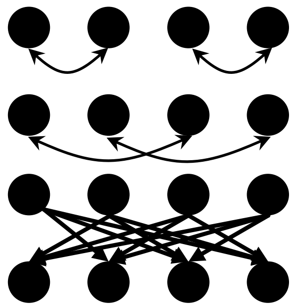
parameter server (sharded) $w' = u(w, \nabla w)$



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

MPI

- Collective operations
- Topologies
- Neighborhood collectives
- RMA?

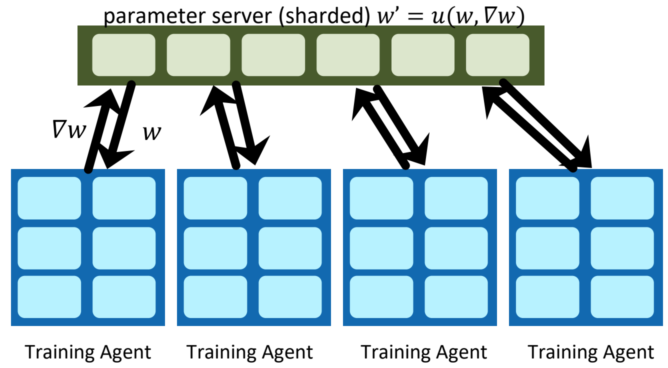


$T = 2L \log_2 P + 2\gamma m G(P-1)/P$

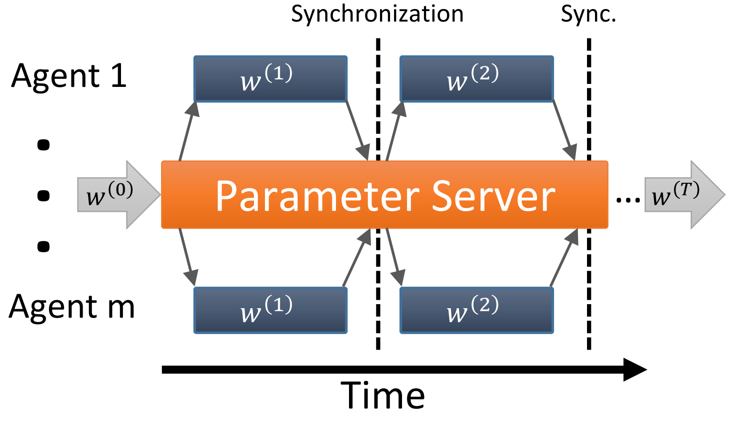
Hierarchical Parameter Server
S. Gupta et al.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study. ICDM'16

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

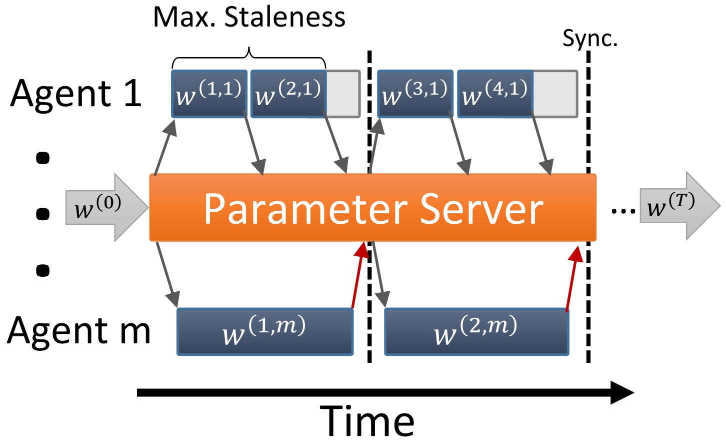
Parameter (and Model) consistency - centralized



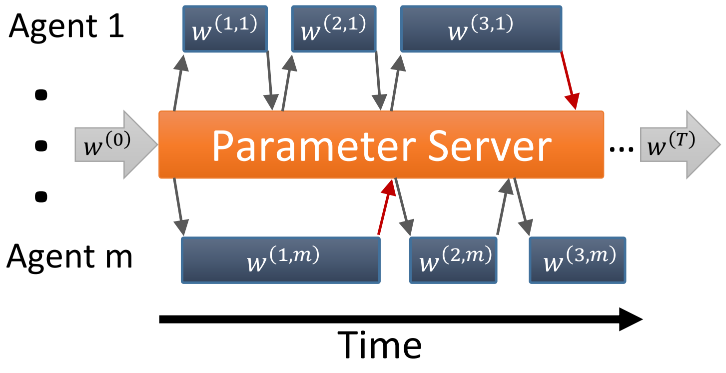
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous



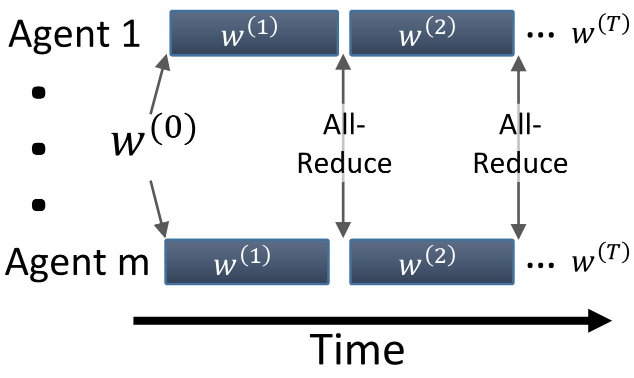
Asynchronous

- **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”**

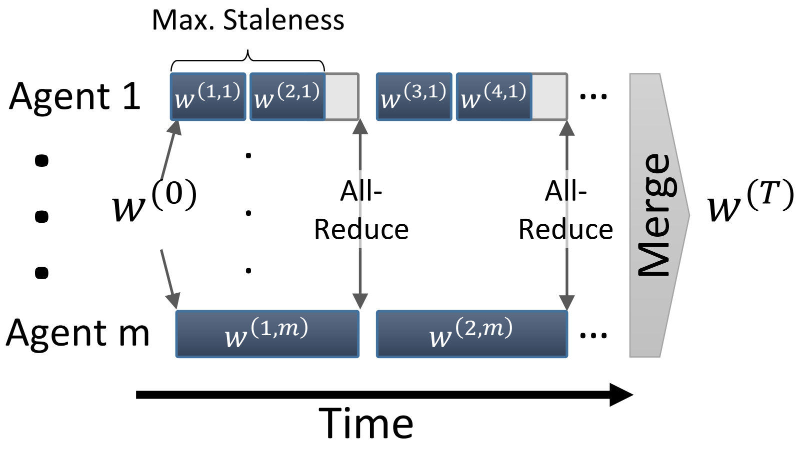
J. Dean et al.: Large scale distributed deep networks, *NIPS*’12.
F. Niu et al.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent, *NIPS*’11.

Parameter (and Model) consistency - decentralized

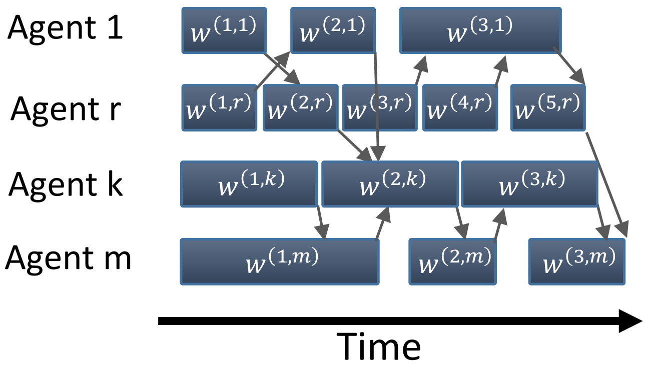
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous

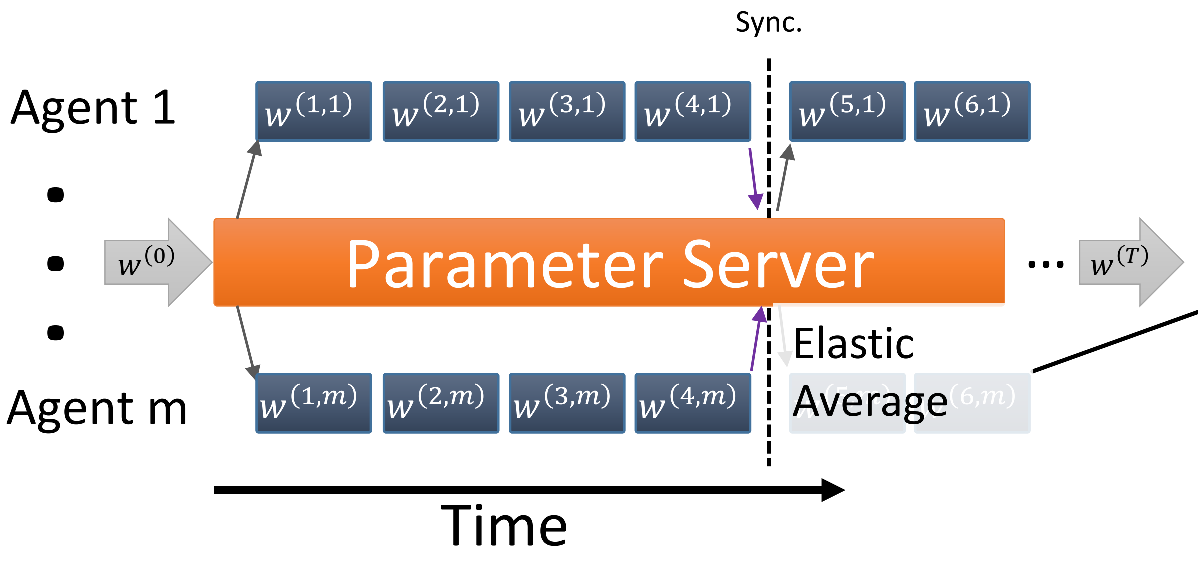


Asynchronous

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

Peter H. Jin et al., "How to scale distributed deep learning?", NIPS MLSystems 2016

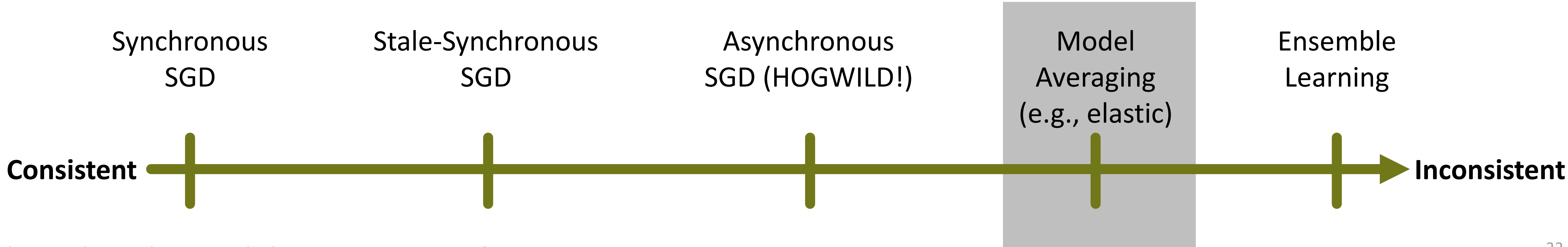
Parameter consistency in deep learning



Using physical forces between different versions of w:

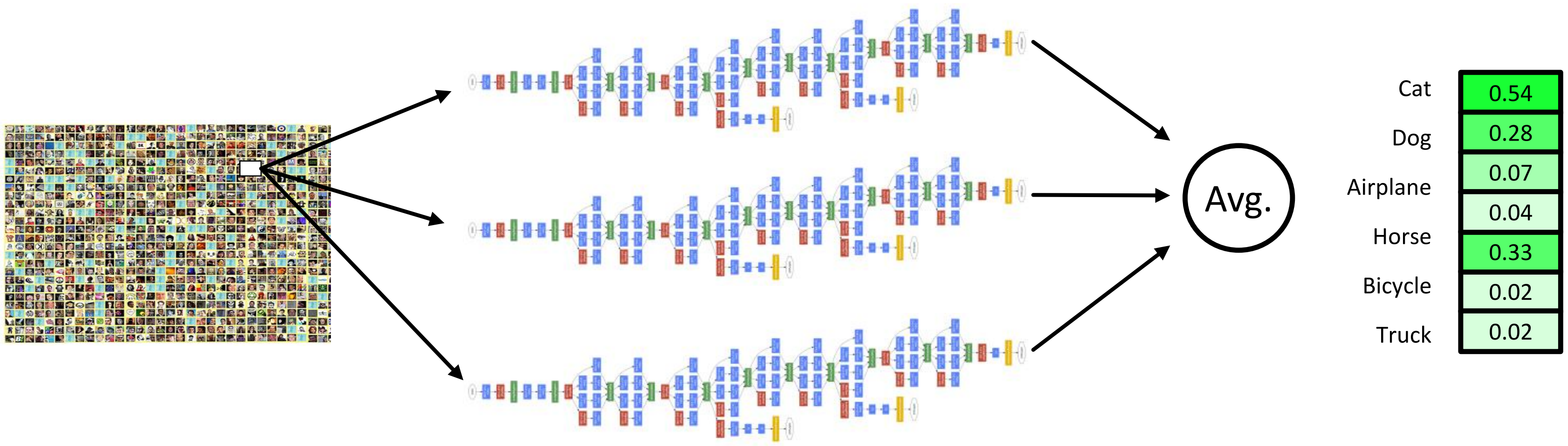
$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \tilde{w}_t\right)$$

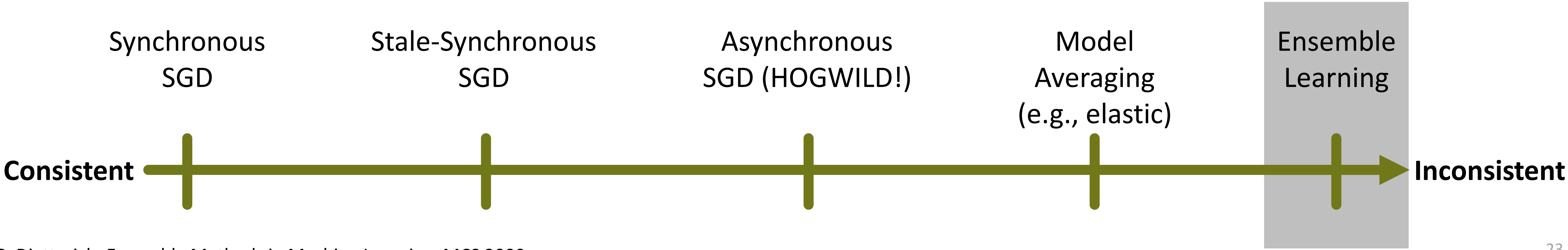
$$\tilde{w}_{t+1} = (1-\beta)\tilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



S. Zhang et al.: Deep learning with Elastic Averaging SGD, NIPS’15

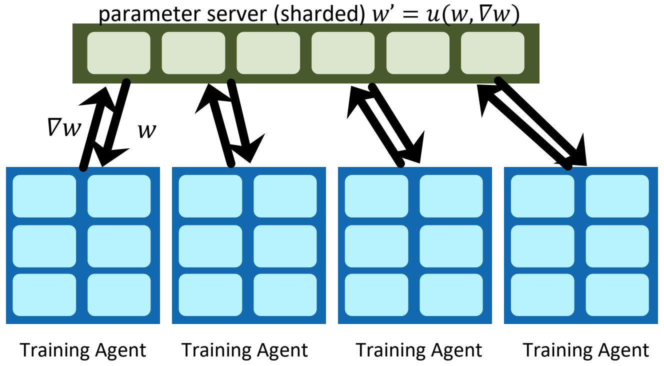
Parameter consistency in deep learning



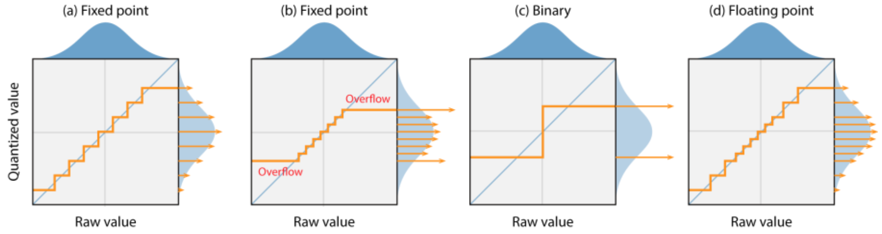


T. G. Dietterich: Ensemble Methods in Machine Learning, MCS 2000

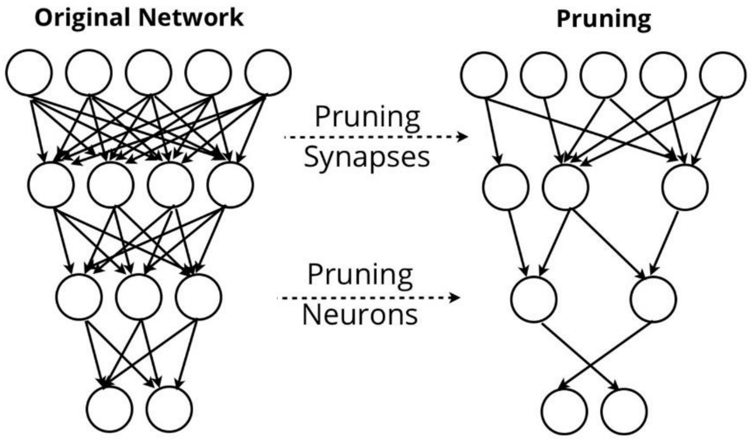
Communication optimizations



- **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



source: ai.intel.com



[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

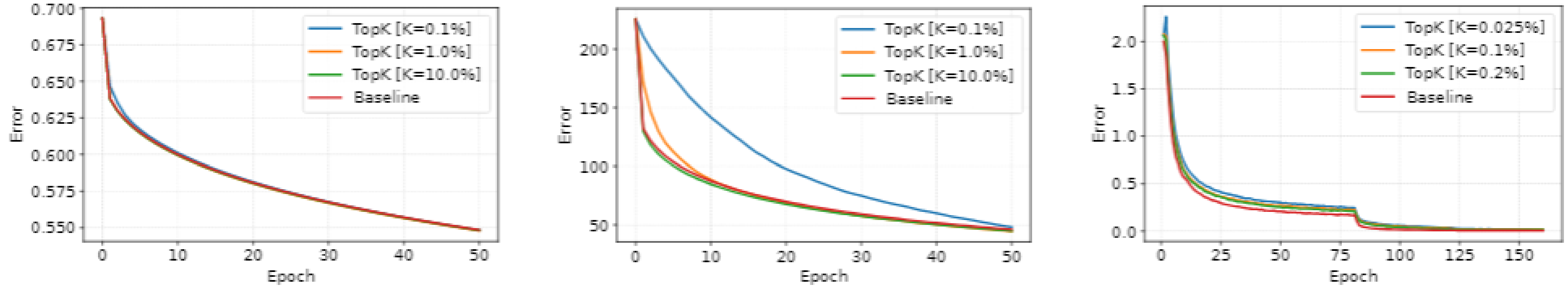
Sparsification – top-k Stochastic Gradient Descent

- **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:*

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

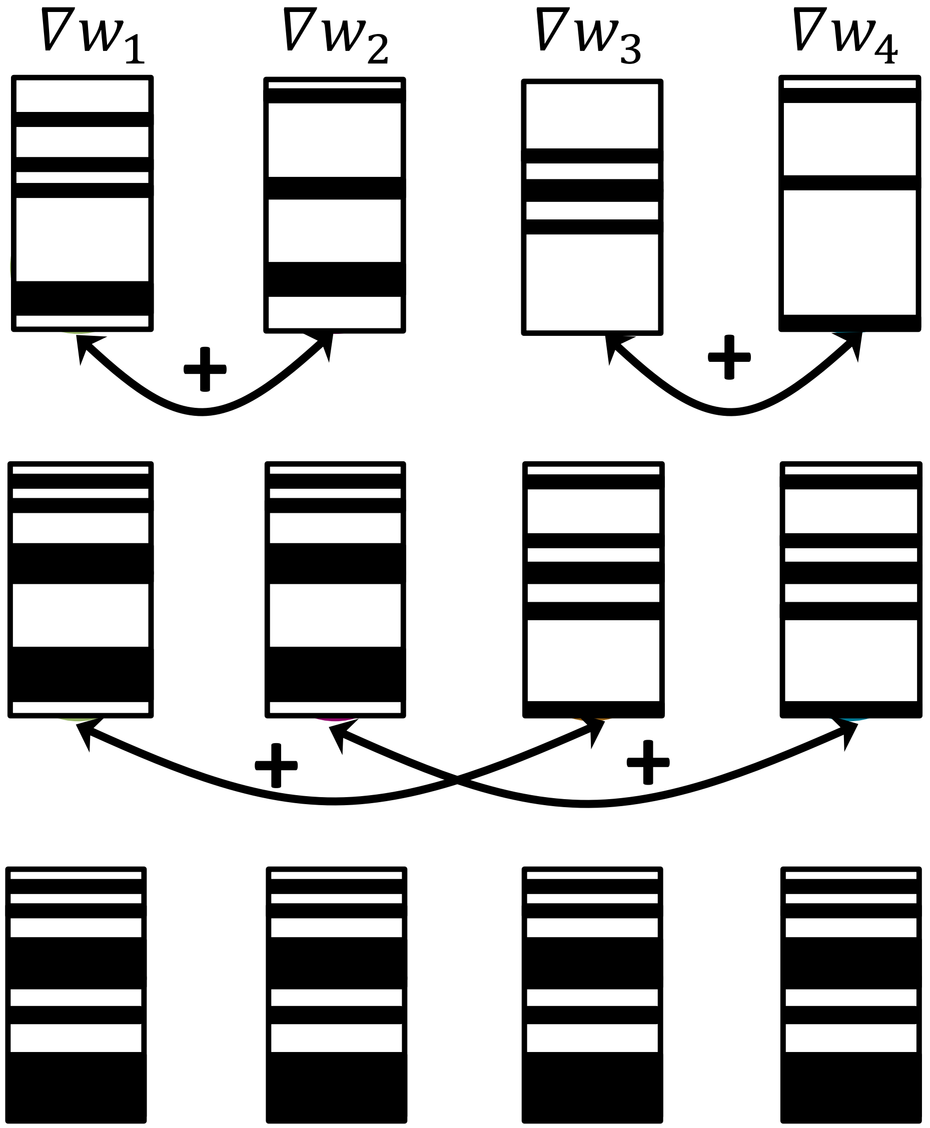
Discussion. We validate Assumption 1 experimentally on a number of different learning tasks in Section 6 (see also Figure 1). In addition, we emphasize the following points:



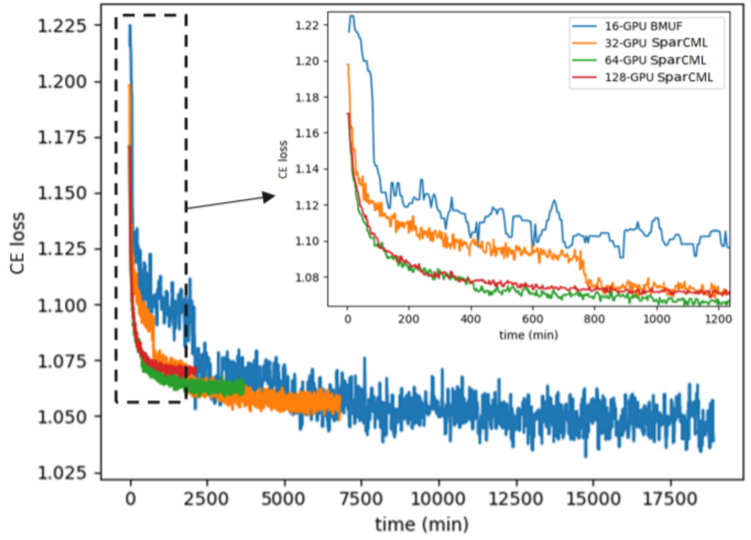
(a) RCV1 convergence. (b) Linear regression. (c) ResNet110 on CIFAR10.

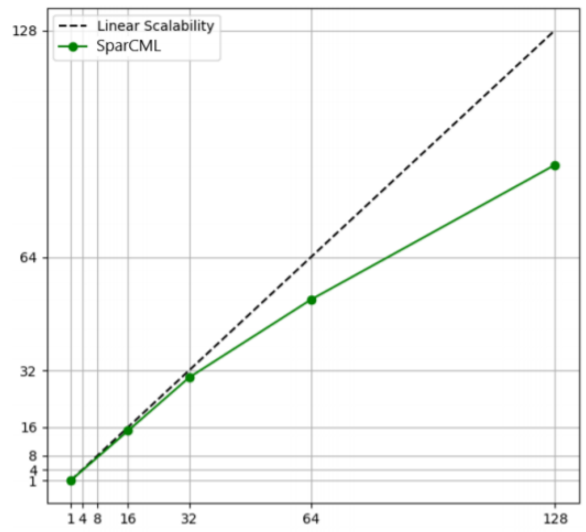
[1] Dan Alistarh, TH, et al.: "The Convergence of Sparsified Gradient Methods", NIPS'18

SparCML – Quantified sparse allreduce for decentral updates



Six epochs, 60 million params





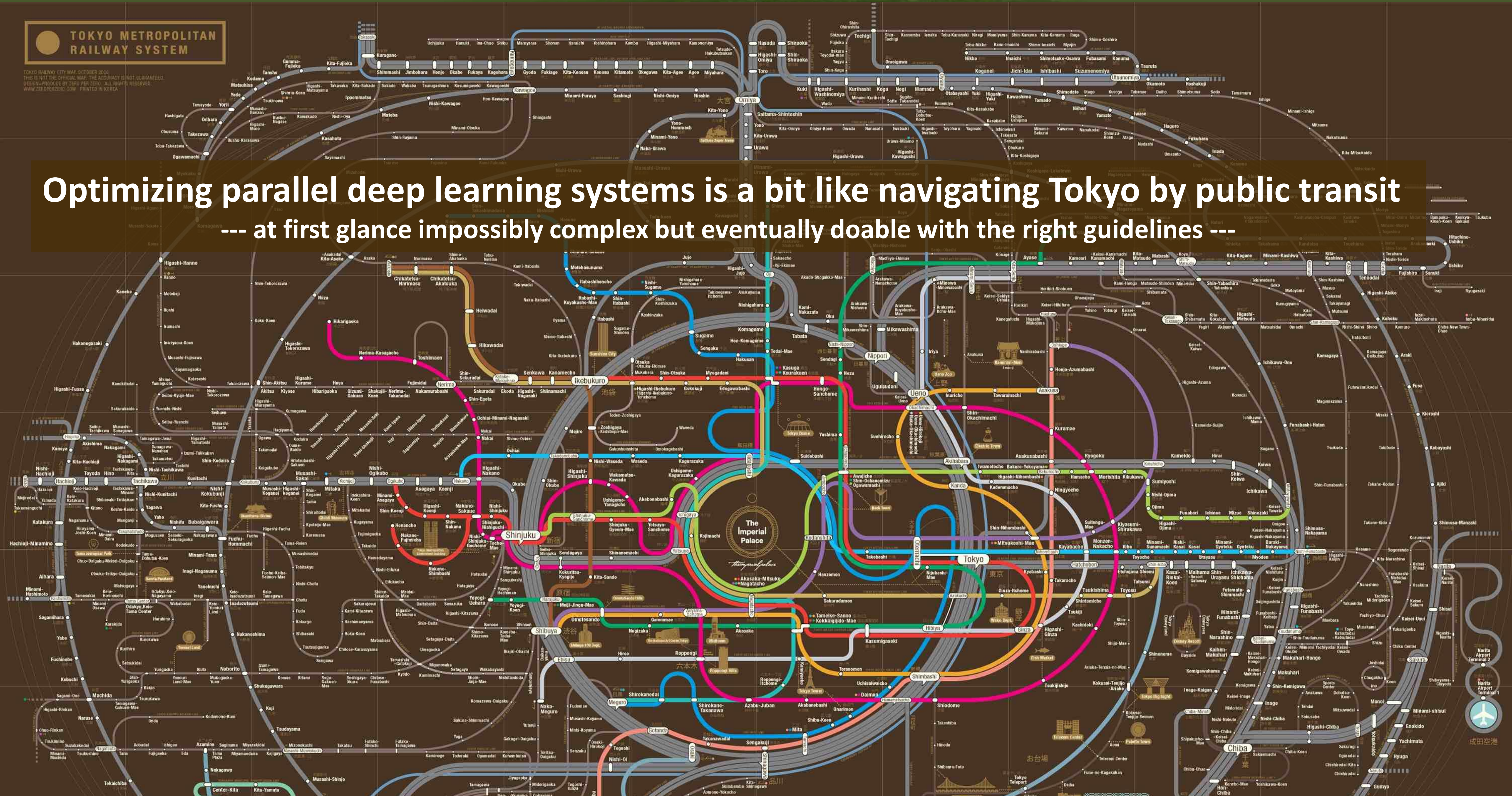
Microsoft Speech Production Workload Results – **2 weeks → 2 days**!

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)

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

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

--- at first glance impossibly complex but eventually doable with the right guidelines ---



Deep500: An HPC Deep Learning Benchmark and Competition

- **Integrates tensorflow, pytorch, caffee2 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!

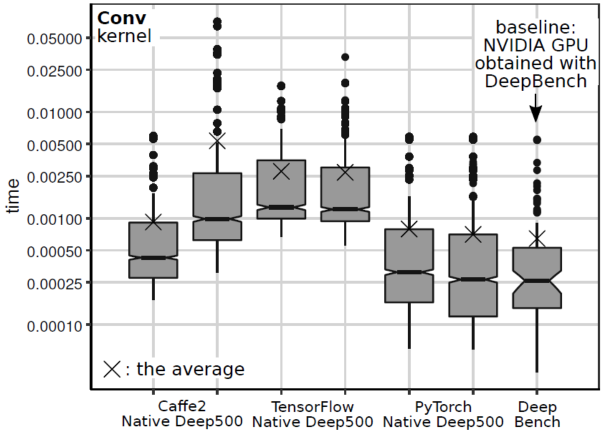
DEEP 500

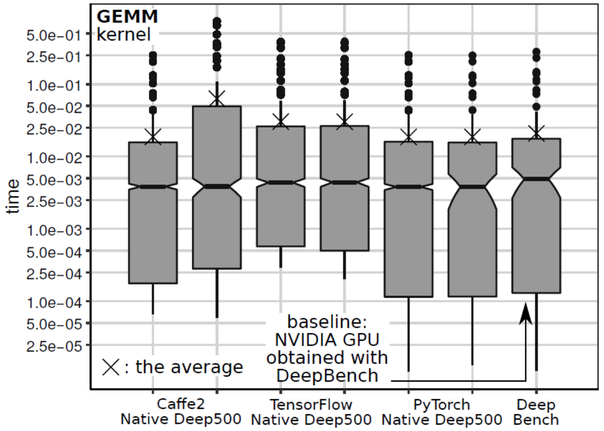
500 ways to train DNNs

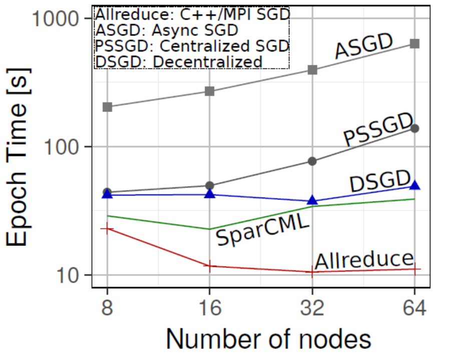
A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning

Tal Ben-Nun, Simon Huber, Maciej Besta, Alexandros Nikolaos Ziogas, Daniel Peter, Torsten Hoefler

Department of Computer Science, ETH Zurich







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

Fig. 11: **Scaling Analysis of Level 3**

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	Parameter Accuracy
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning	• 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 <u>not</u> 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)

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

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/

NOV 8 2018

Twelve ways to fool the masses when reporting performance of deep learning workloads

blog Uncategorized

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

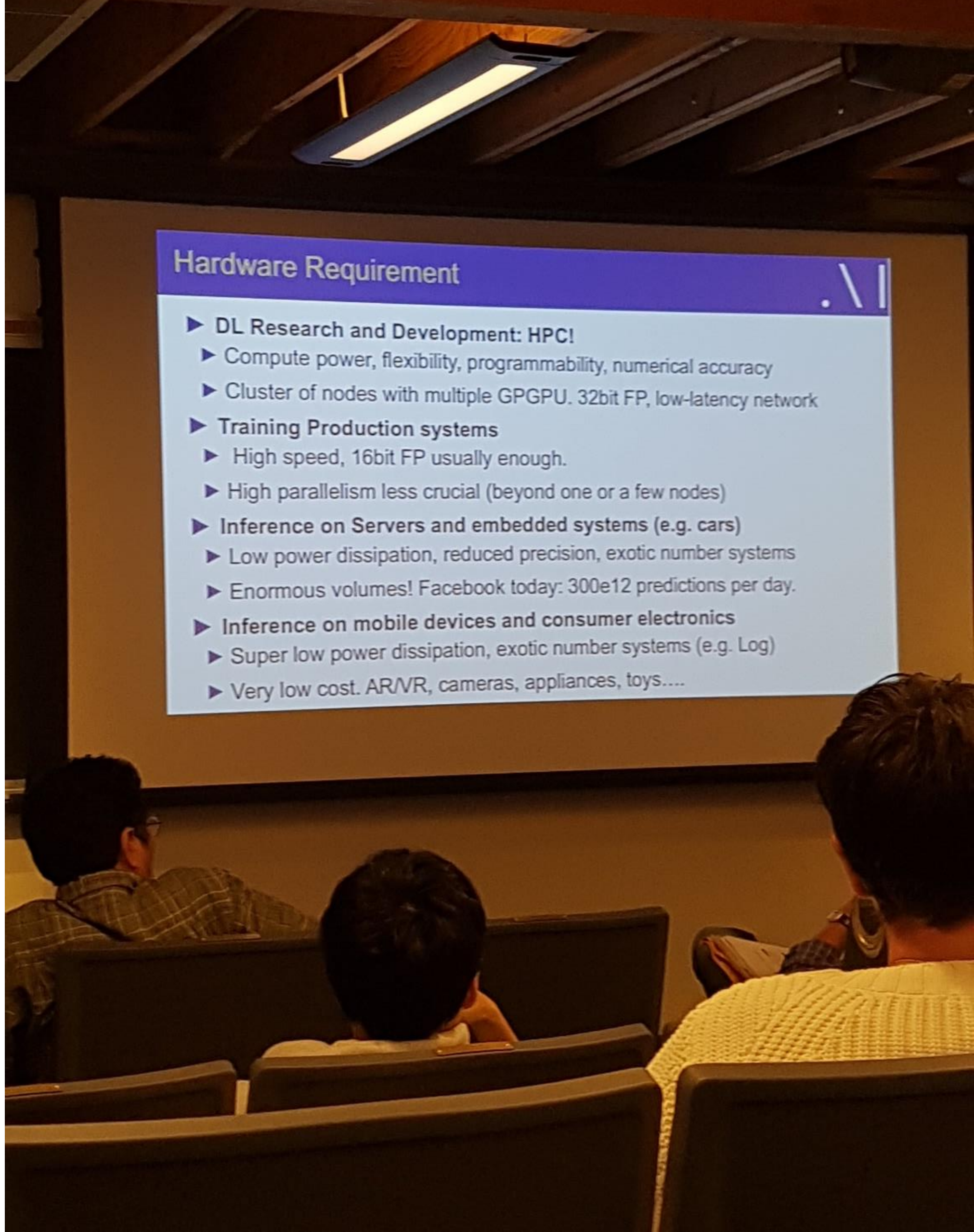
    Besides at SC18, whut!?

- **Risk of unrealism**
  - HPC people know how to do HPC
  - And deep learning is HPC, right?

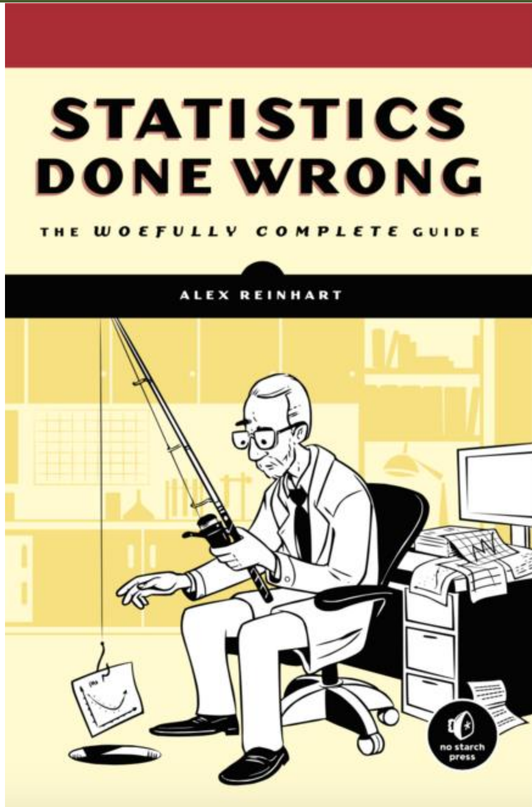
    Not quite … while it's really similar (tensor contractions)

    But it's also quite different!

Yann LeCun's conclusion slide yesterday!



"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!

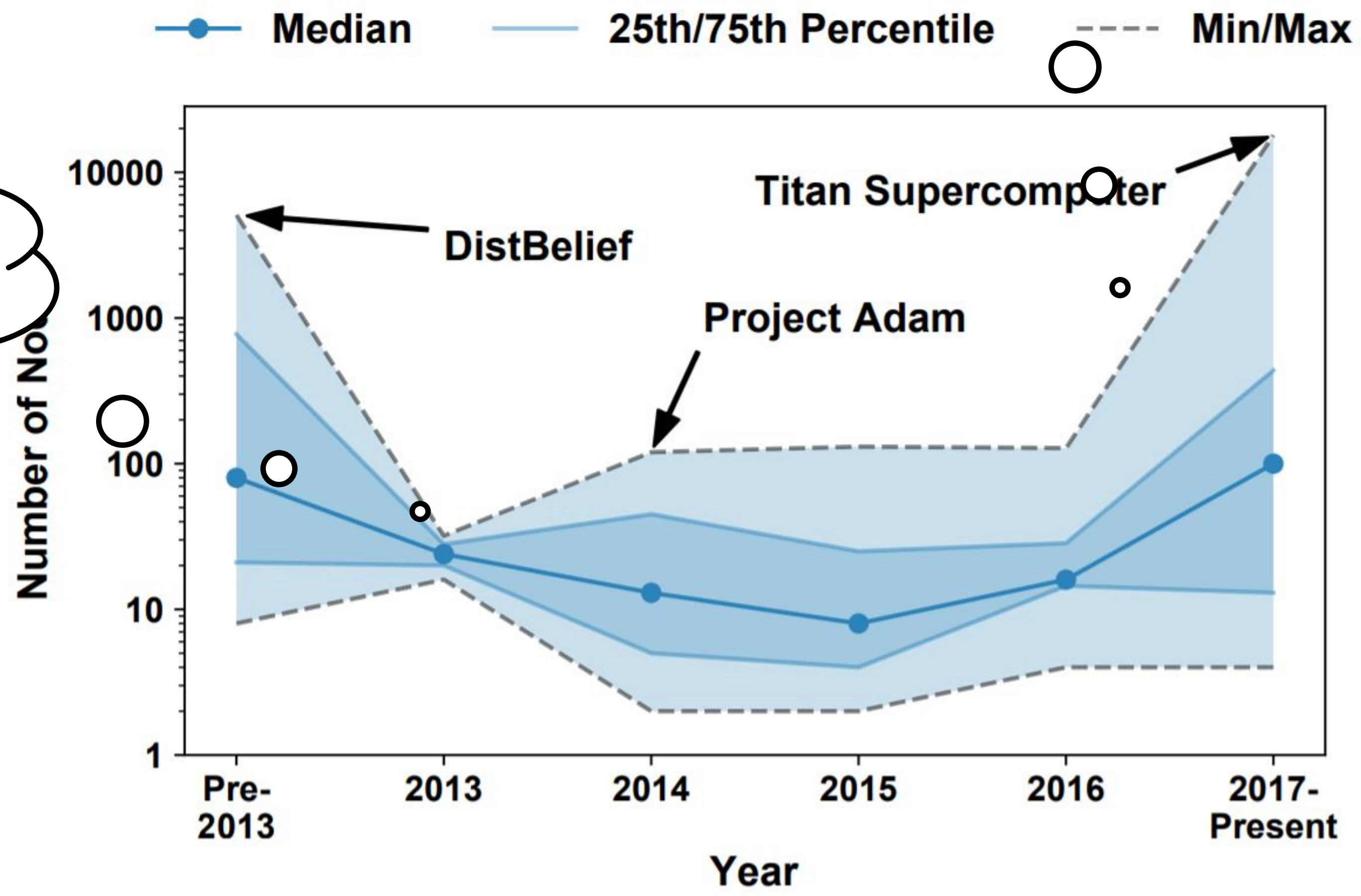
HPC picking up!

- **Too obvious for this audience**
  - Was very popular in 2015!

- **Surprisingly many (still) do this**

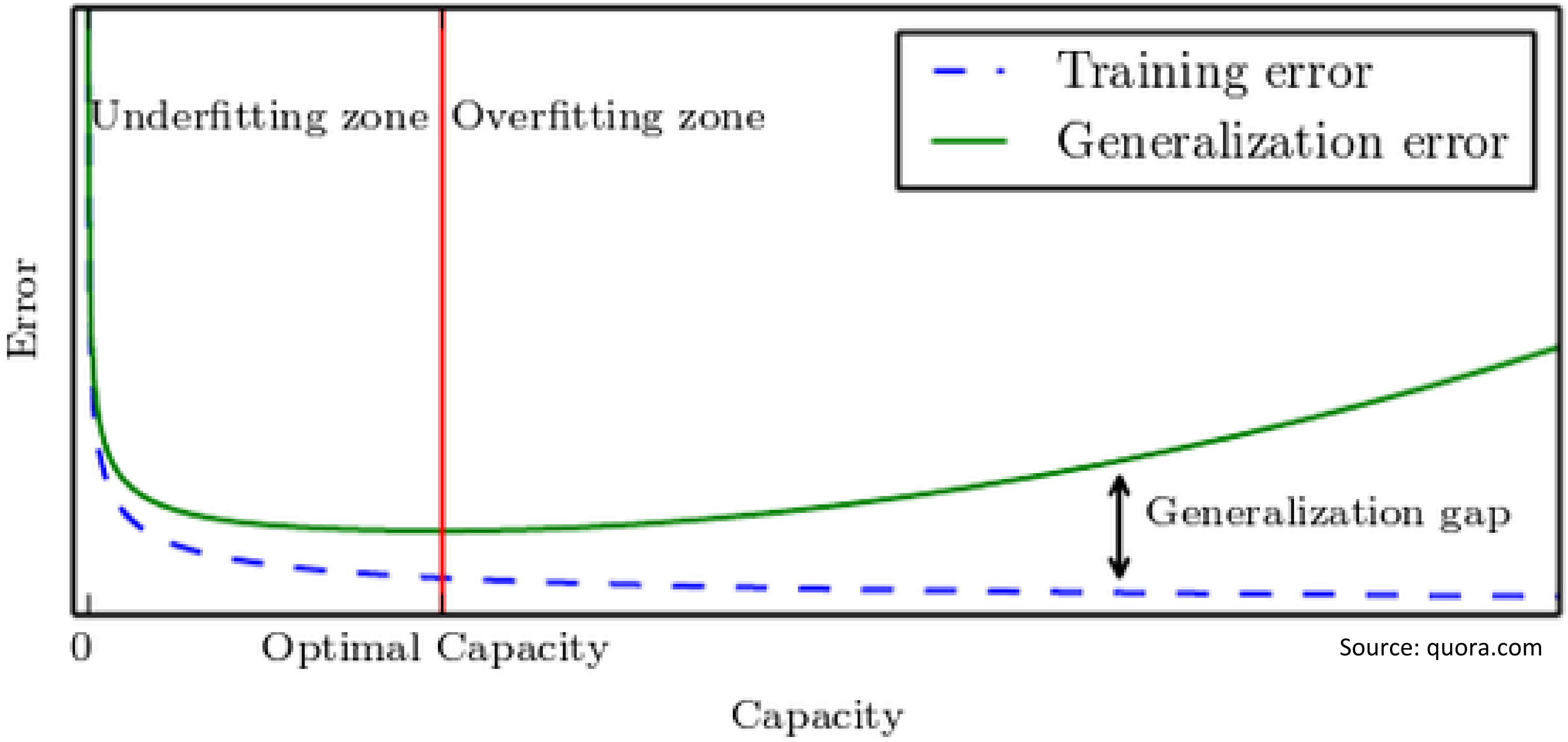
Learning community's self-correction (Y. LeCun)

Scalability without a good baseline? (D. Bailey)



2) Do not report test accuracy!

- **Training accuracy is sufficient isn't it?**

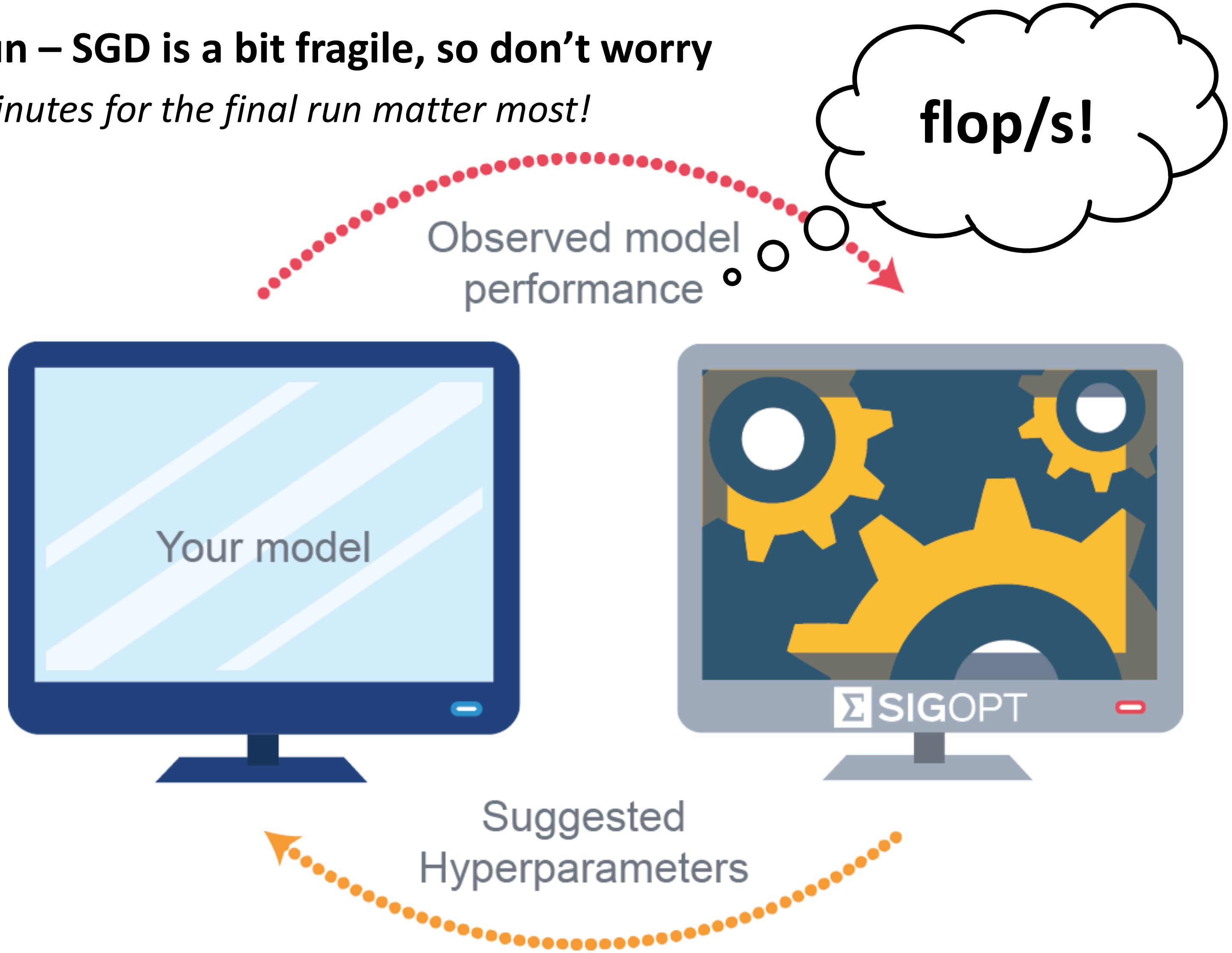


Source: quora.com

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.

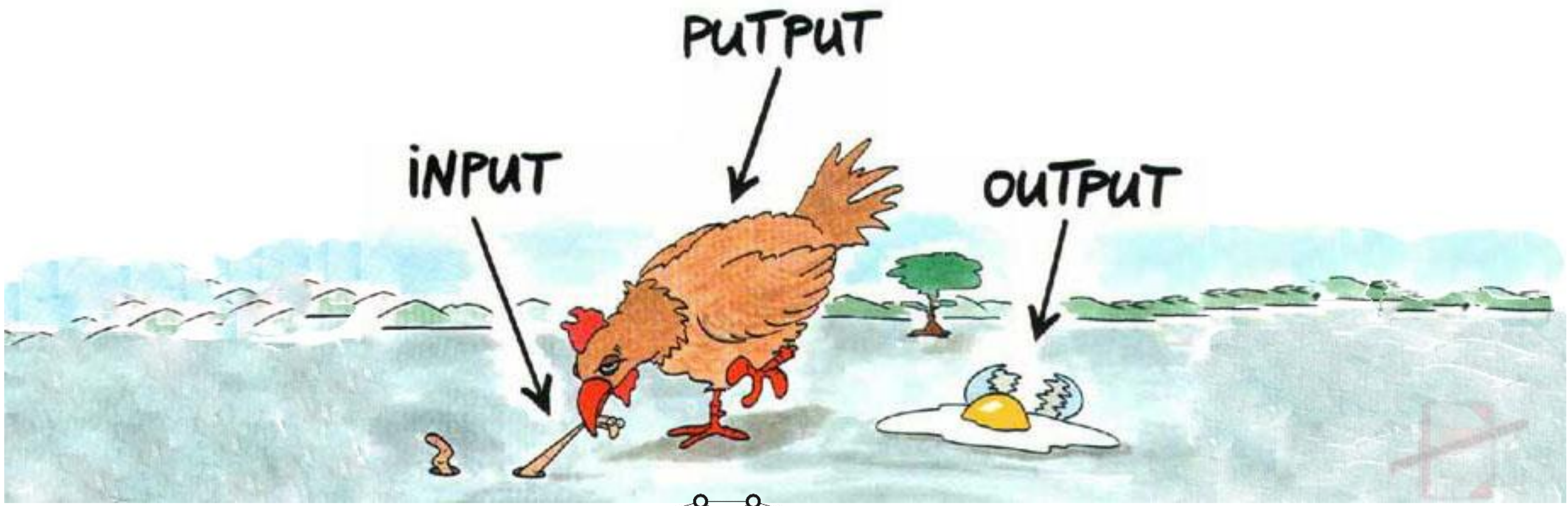
5) Show only kernels/subsets when scaling!

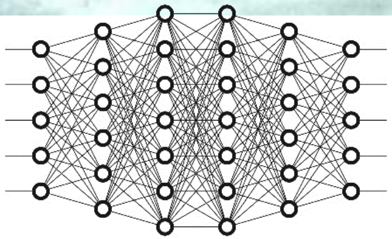
- **Run layers or communication kernels in isolation**
  - Avoids issues with accuracy completely ☺
    Doesn't that look a bit like GoogLeNet?

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!

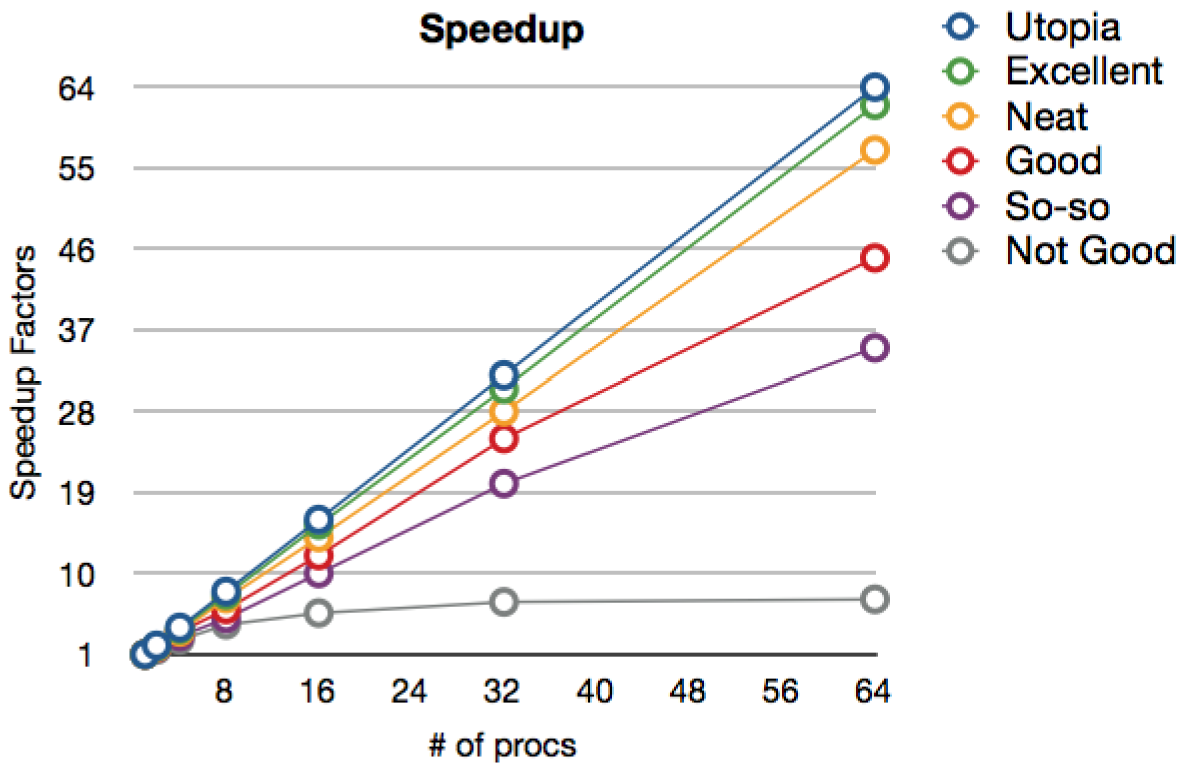
    And never talk about how many more of those ops one needs to find a solution, it's all about the rate, op/s!
- **Actually, my laptop achieves an "exaop":**
  - each of the 3e9 transistors switching a binary digit each at 2.4e9 Hz

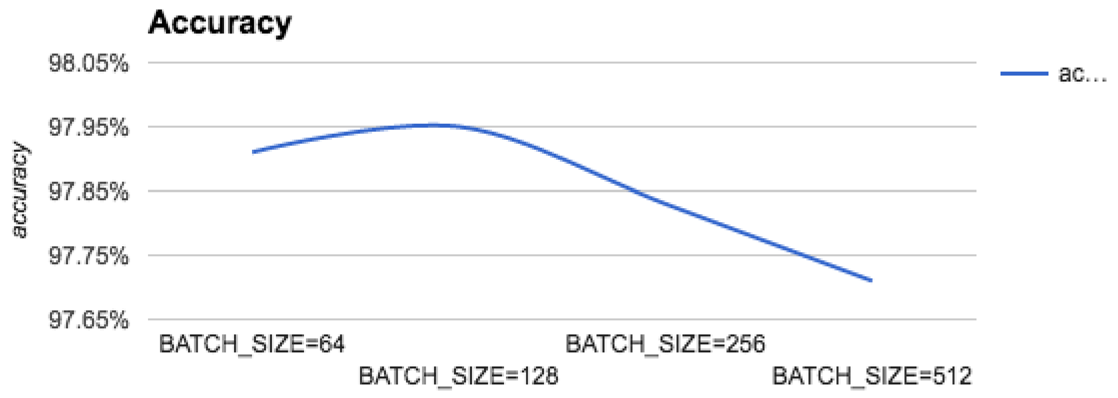
vs.

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!



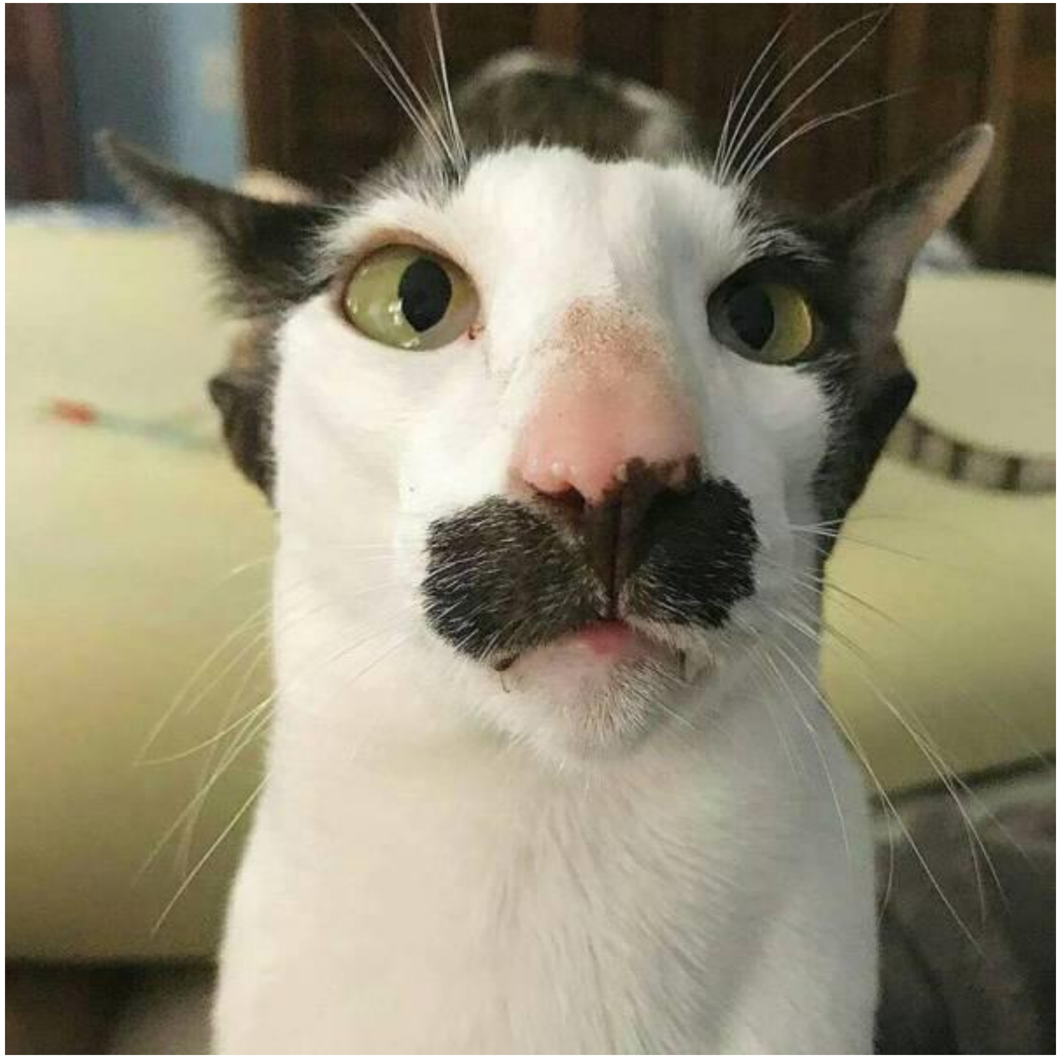


9) Train on (unreasonably) large inputs!

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

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

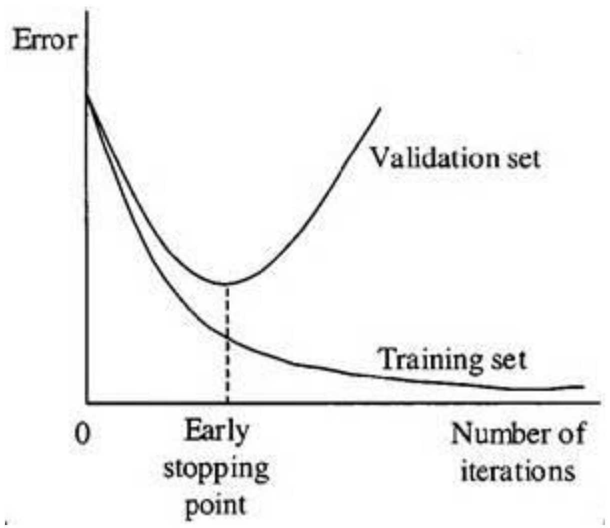
vs.



High-resolution cat (8kx8x – 1 Tflop/example)

10) Run training just for the right time!

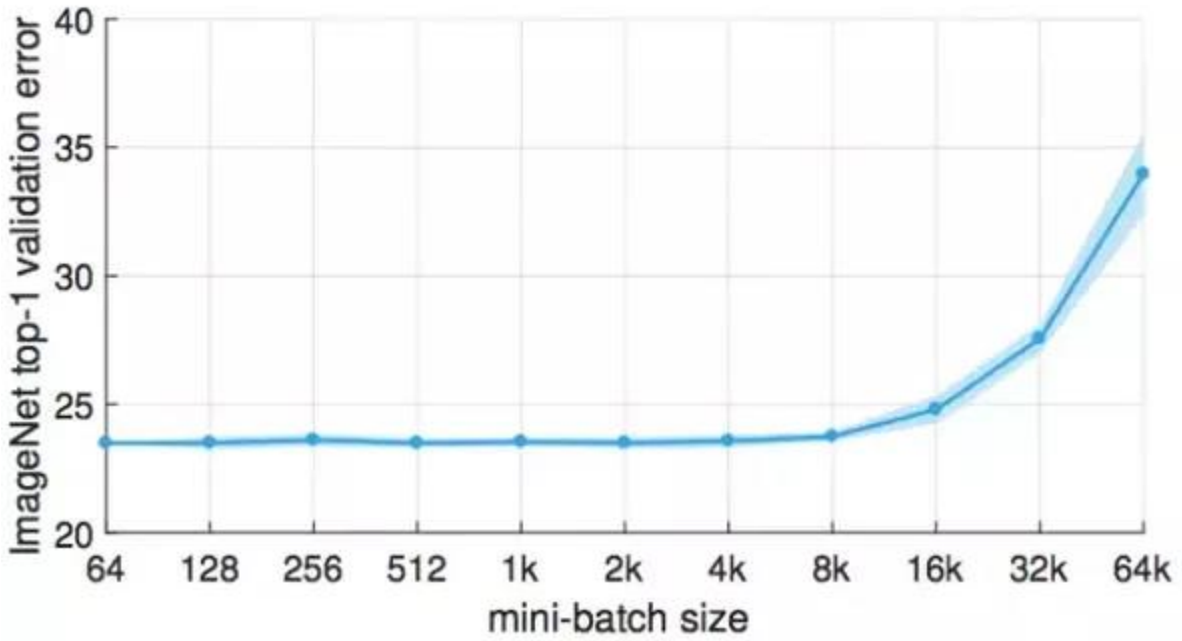
- **Train for fixed wall-time when scaling processors**
  - so when you use twice as many processors you get twice as many flop/s!
    But who cares about application speedup?



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)

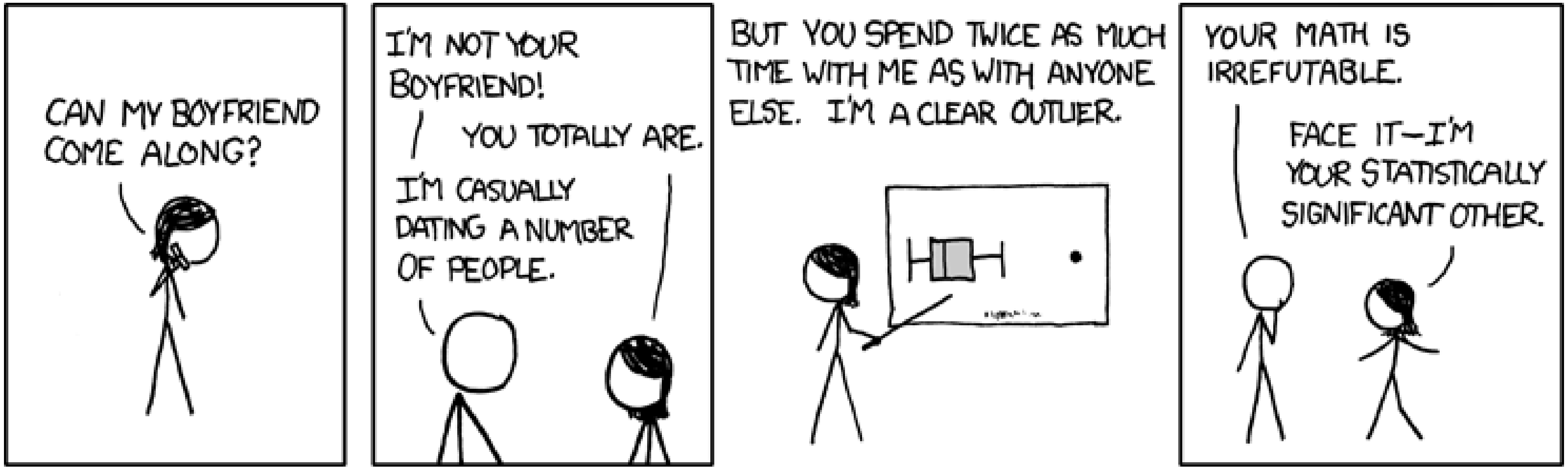
- **Microbatching is not a problem!**



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.



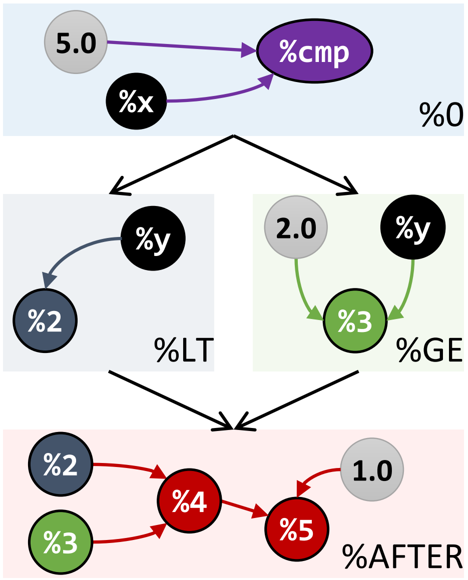
Turning 180-degree – Deep Learning for HPC – Neural Code Comprehension

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

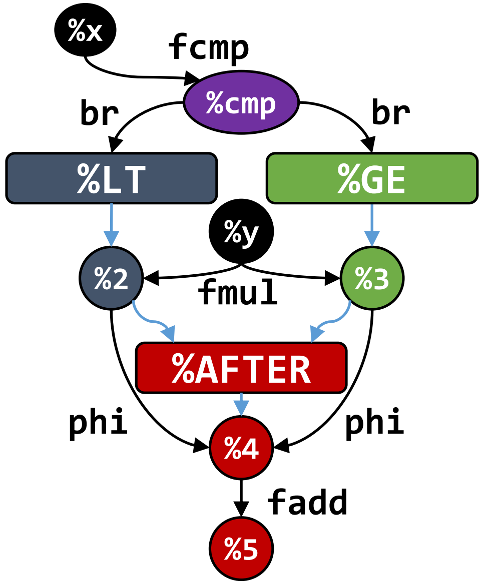
```
double thres = 5.0;

if (x < thres)
    x = y * y;
else
    x = 2.0 * y;


x += 1.0;
```

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

br i1 %cmp, label %LT, label %GE
LT:
  %2 = fmul double %y, %y
GE:
  %3 = fmul double 2.0, %y


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

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



Dataflow (basic blocks)

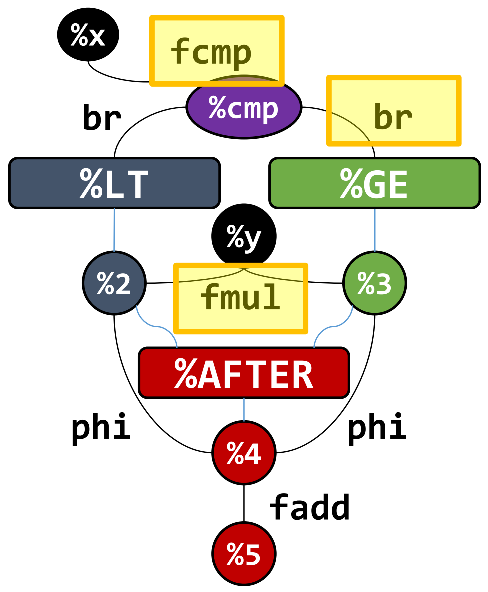


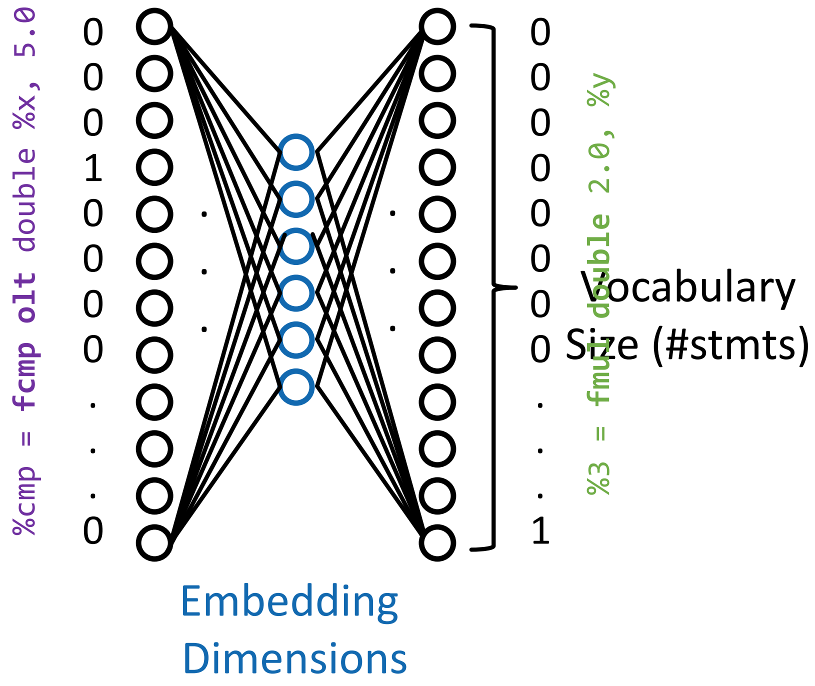
conteXtual Flow Graph

Ben-Nun, Jakobovits, TH: Neural Code Comprehension: A Learnable Representation of Code Semantics, NIPS 2018

Deep Learning for HPC – Neural Code Comprehension

- **Embedding space (using the Skip-gram model)**





Deep Learning for HPC – Neural Code Comprehension

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**

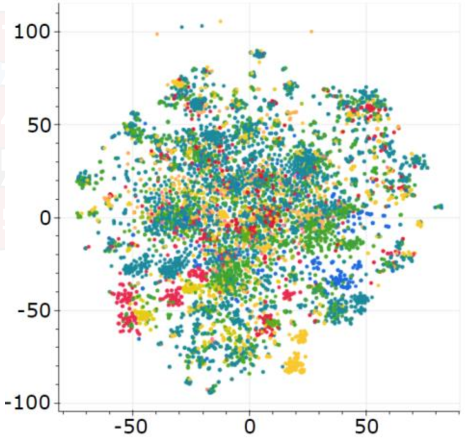


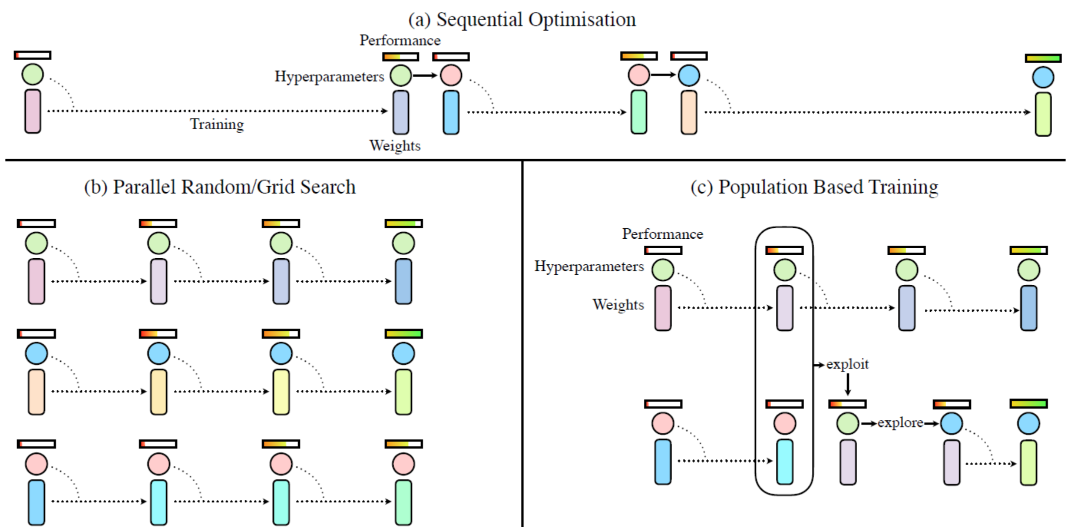
Table 2: Analogy and test scores for inst2vec

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%

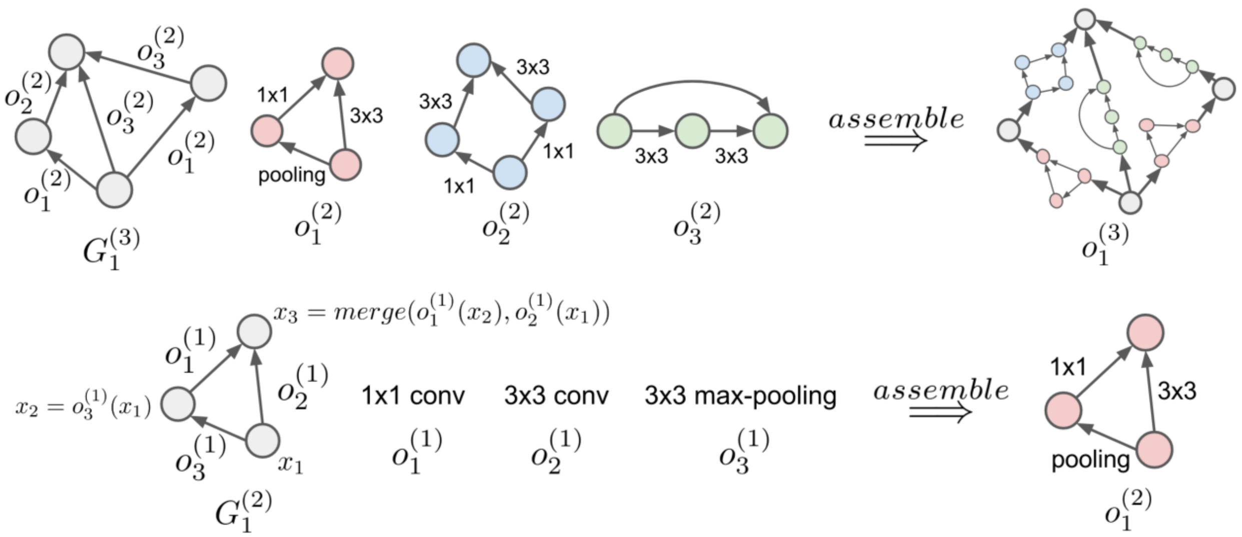
Ben-Nun, Jakobovits, TH: Neural Code Comprehension: A Learnable Representation of Code Semantics, NIPS 2018

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



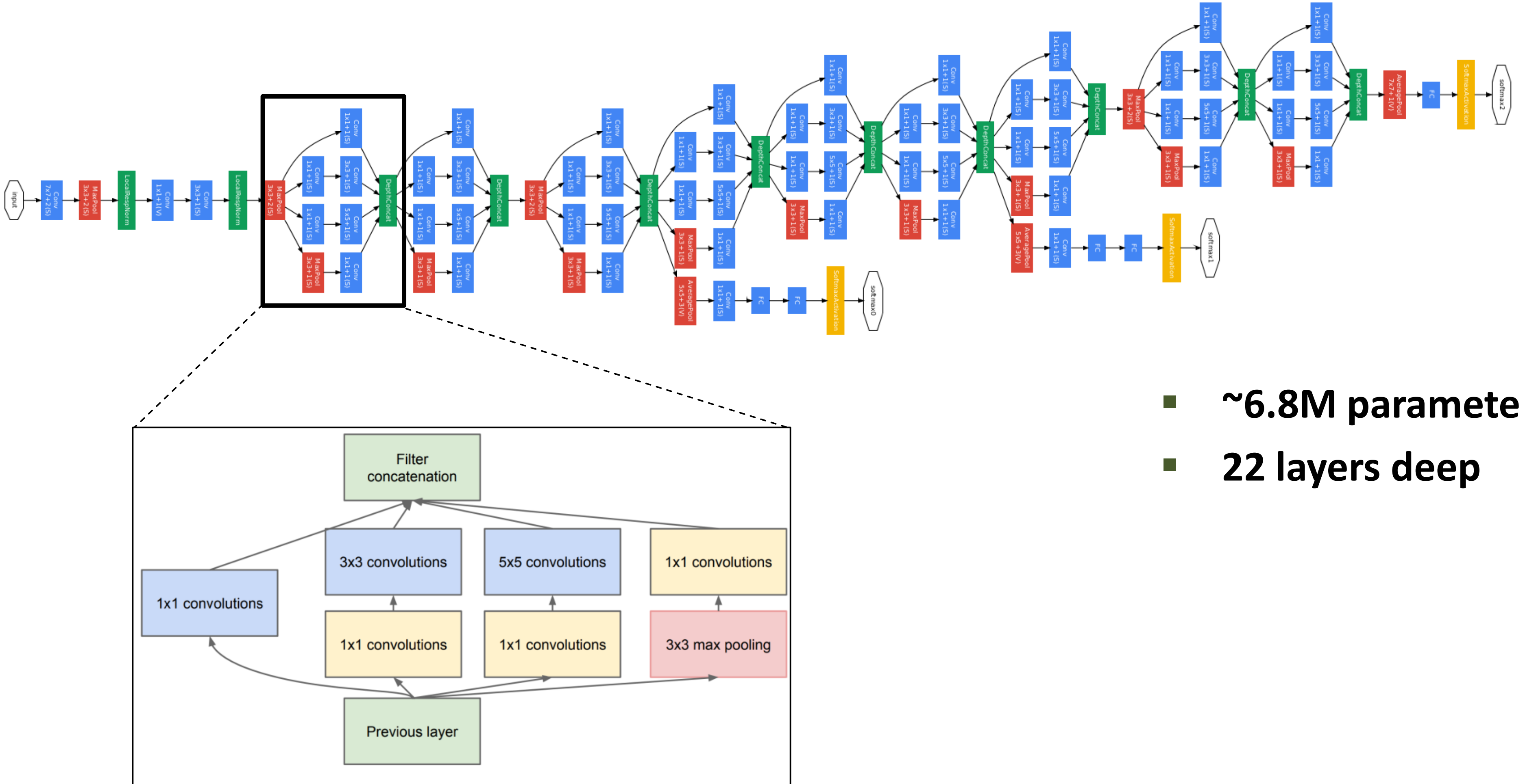
Reinforcement Learning [1]



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



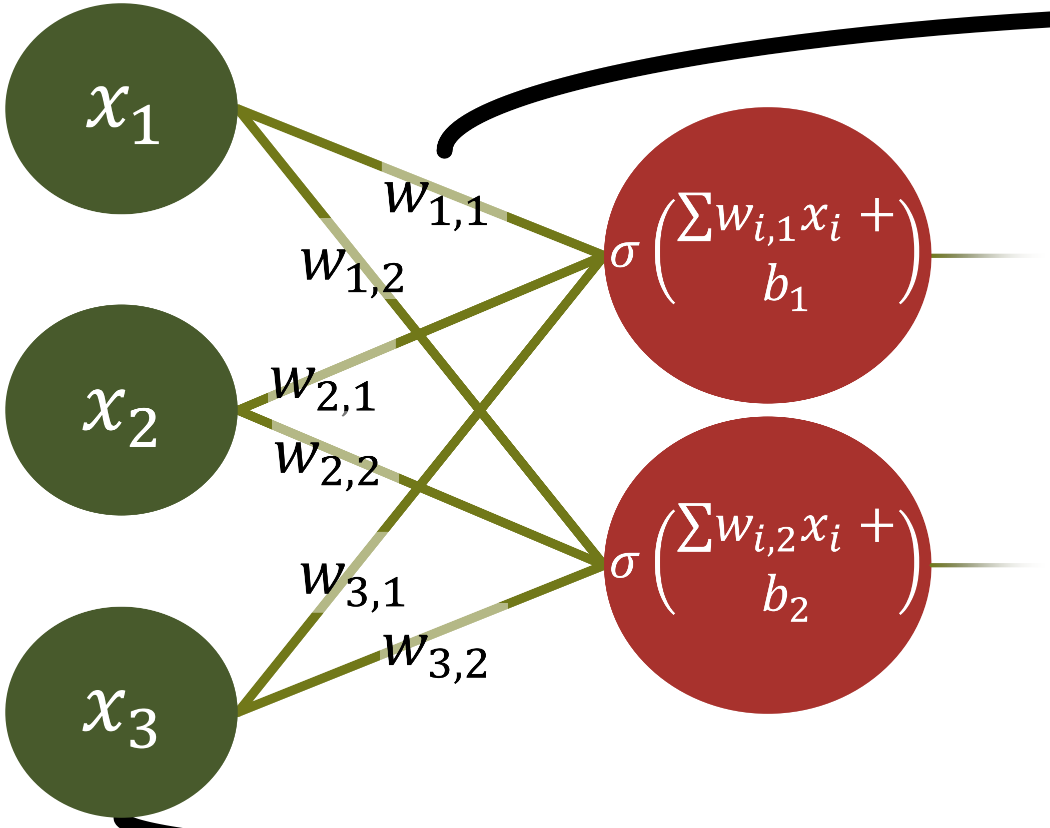
(b) Inception module with dimensionality reduction

- **~6.8M parameters**
- **22 layers deep**

C. Szegedy et al. Going Deeper with Convolutions, CVPR'15

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	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$



$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & 1 \\ x_{2,1} & x_{2,2} & x_{2,3} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & x_{N,3} & 1 \end{pmatrix} \cdot \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \\ b_1 & b_2 \end{pmatrix}$$

Computing convolutional layers

Direct

Indirect

Method	Work (W)	Depth (D)
Direct	$N \cdot C_{out} \cdot H' \cdot W' \cdot C_{in} \cdot K_y \cdot K_x$	$\lceil \log_2 C_{in} \rceil + \lceil \log_2 K_y \rceil + \lceil \log_2 K_x \rceil$
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

K. Chellapilla et al.: High Performance Convolutional Neural Networks for Document Processing, Int'l Workshop on Frontiers in Handwriting Recognition 2016
M. Mathieu et al.: Fast Training of Convolutional Networks through FFTs, ICLR'14
A. Lavin and S. Gray: Fast Algorithms for Convolutional Neural Networks, CVPR'16