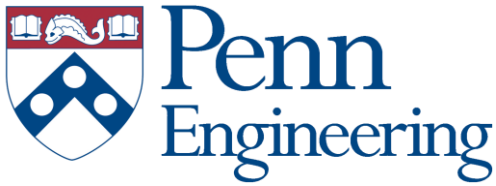
Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

Tal Ben-Nun and Torsten Hoefler

Presenter: Leshang Chen @ Stat 991 DL Seminar, March 28th, 19



DL is Hot!

- Over 20 DL papers a day!
- Universal usage.

Table 1. Yearly arXiv Papers in Computer Science (AI and Computer Vision)

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

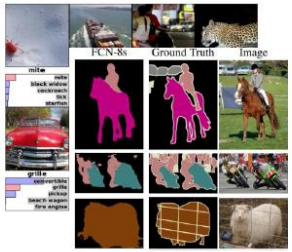
Digit Recognition

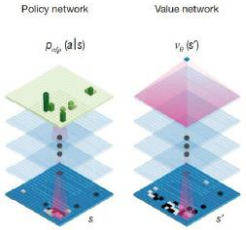
Object Classification
Segmentation

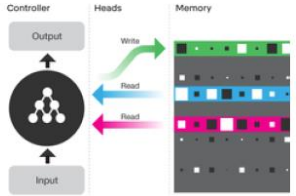
Image Captioning

Gameplay AI
Translation

Neural Computers







1989 2012 2013 2014 2016 2017

- Scale is a problem.

Table 5. Popular Neural Network Characteristics

Property	LeNet [145]	AlexNet [136]	GoogLeNet [226]	ResNet [93]	DenseNet [108]
$\|w\|$	60K	61M	6.8M	1.7M–60.2M	~15.3M–30M
Layers ($\propto$ D)	7	13	27	50–152	40–250
Operations ($\propto$ W, ImageNet-1k)	N/A	725M	1566M	~1000M–2300M	~600M–1130M
Top-5 Error (ImageNet-1k)	N/A	15.3%	9.15%	5.71%	5.29%
Top-1 Error (CIFAR-10)	N/A	N/A	N/A	6.41%	3.62%

Penn Engineering

Intro to the paper

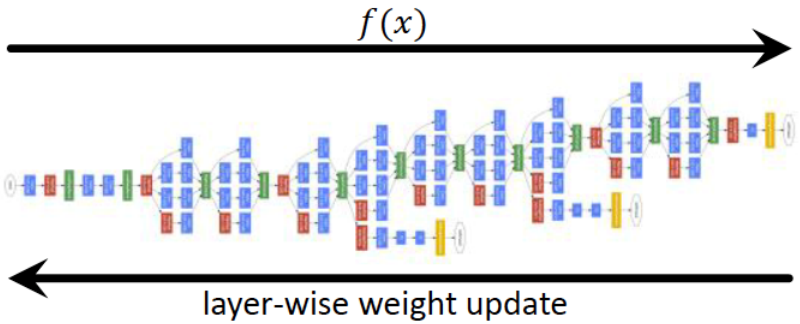
- What's this survey generally about?
  - How to design/implement supervised deep learning in parallel.
  - 47 Pages, 269 Papers referenced.
  - We won't cover all of them! (e.g. RNN)

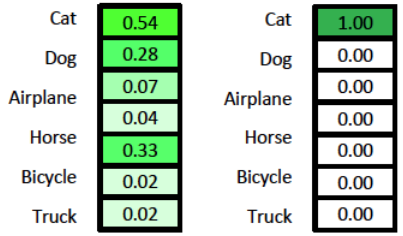
- What we will cover?
  - Basics (Ch. 2-4)
  - Operator Level Parallelism (Ch. 5)
  - Network Level Parallelism (Ch. 6)
  - Distributed Training (Ch 7)

- Any additional public resource? - Presentation in HPC 2018.
  - Video by author: https://www.youtube.com/watch?v=xtxxLWZznBI
  - Slides by author: https://htor.inf.ethz.ch/publications/img/ddl-slides.pdf

Supervised DL

- Task: Classification or Regression.
- Process: Feed-forward + Back-propagation.





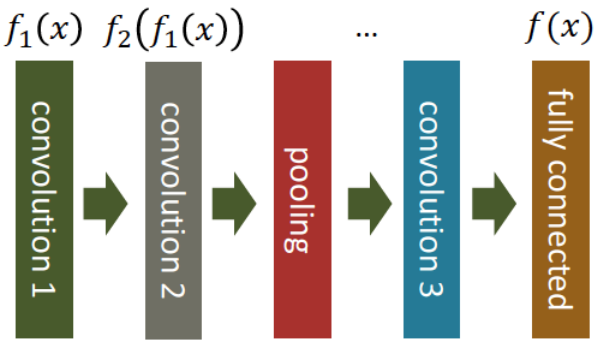
labeled samples $x \in X \subset \mathcal{D}$

label domain Y true label $l(x)$

$f(x): X \to Y$

network structure (fixed) weights w (learned)

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



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

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

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

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

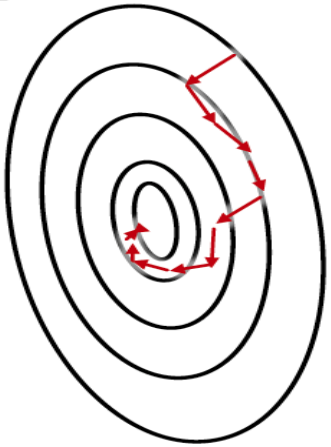
Penn Engineering

Stochastic Gradient Descent

- SGD to solve supervised DL. $w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$

Algorithm 1 Stochastic Gradient Descent (SGD)

1: **for** $t = 0$ **to** T **do** ▷ SGD iterations
2: $z \leftarrow$ Random element from S ▷ Sample dataset S
3: $g \leftarrow \nabla \ell(w^{(t)}, z)$ ▷ Compute gradient of ℓ
4: $w^{(t+1)} \leftarrow w^{(t)} + u(g, w^{(0,\ldots,t)}, t)$ ▷ Update weights with function u
5: **end for**



- Many adaptive weight-update rules!

Table 3. Popular Weight Update Rules

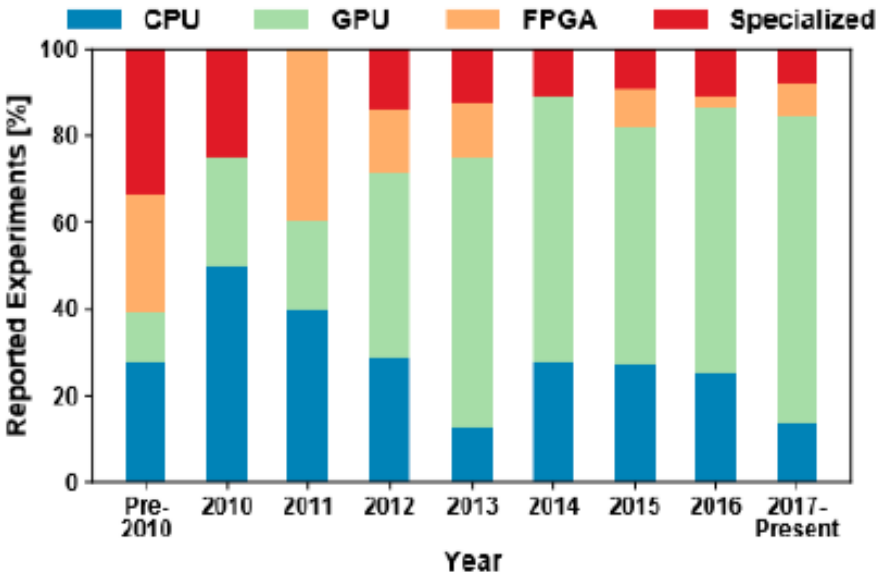
Method	Formula	Definitions
Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla w^{(t)}$	$\nabla w^{(t)} \equiv \nabla \ell(w^{(t)}, z)$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$	
Momentum [197]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$	
Nesterov Momentum [178]	$w^{(t+1)} = w^{(t)} + v_t$	$v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla \ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [68]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t} + \varepsilon}}$	$A_{i,t} = \sum_{\tau=0}^{t} \left(\nabla w_i^{(t)}\right)^2$
RMSProp [95]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t} + \varepsilon}}$	$A'_{i,t} = \beta \cdot A'_{t-1} + (1-\beta)\left(\nabla w_i^{(t)}\right)^2$
Adam [130]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)}} + \varepsilon}$	$M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1-\beta_m)\left(\nabla w_i^{(t)}\right)^m}{1-\beta_m^t}$

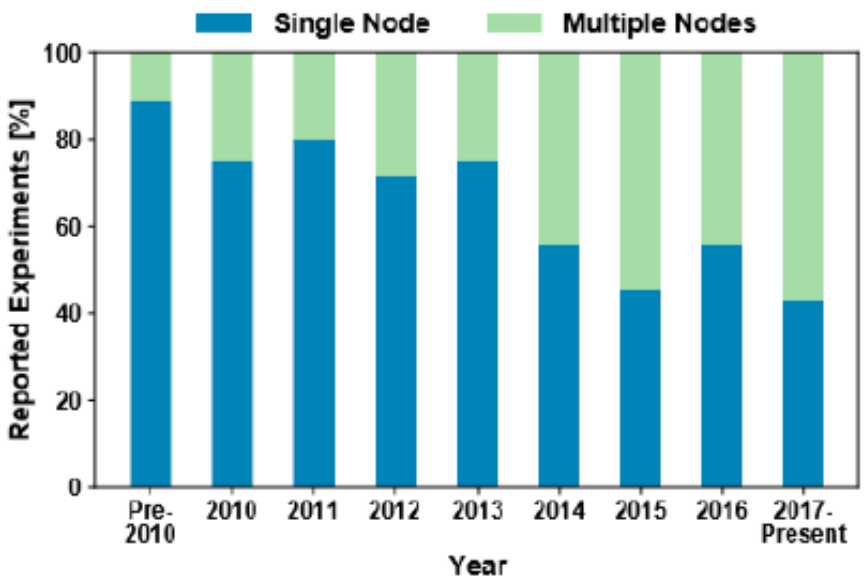
Penn Engineering

Trends in Deep Learning

- Survey results from 227 papers in the area of parallel deep learning
  - The field is moving fast – trying everything imaginable
  - Hardware and multi-node!

- **Single/multi**-machine Parallelism**.**

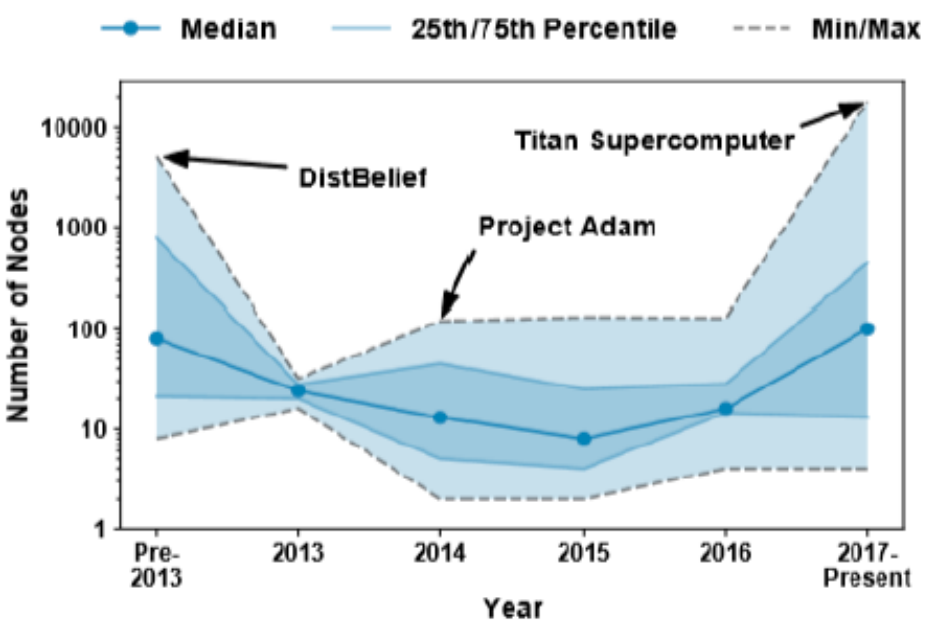


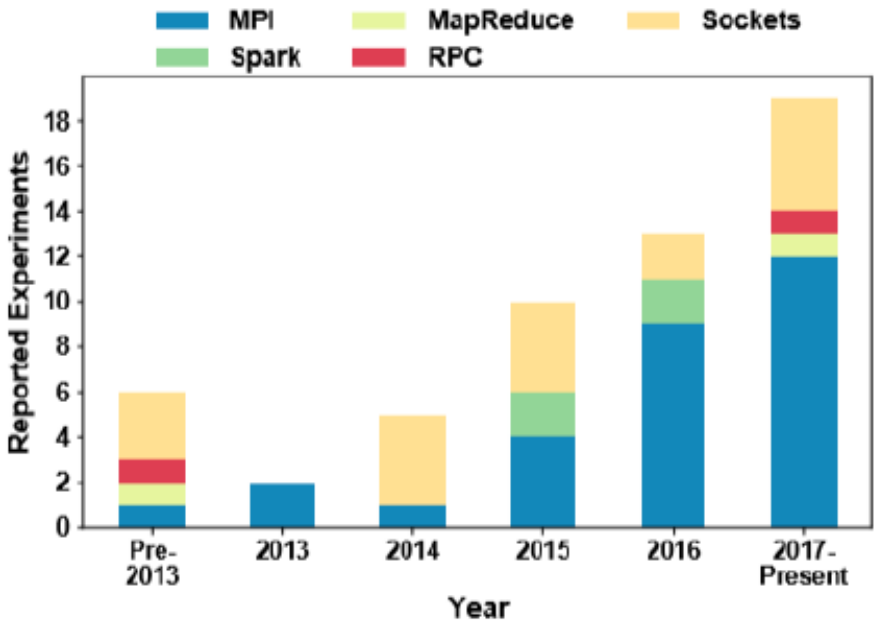


Penn Engineering

Trends in Distributed DL

- Deep Learning is largely on distributed memory today!
- **Distributed** deep learning:
  - Node count and communication
  - MPI the low level portable communication become de-facto.
  - Nobody use TF/Pytorch in DL research? (Hard to modify? Not low level?)





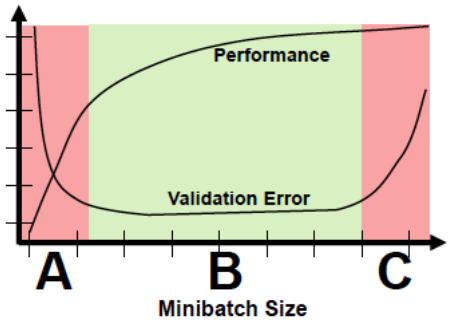
Penn Engineering

Mini-Batch SGD

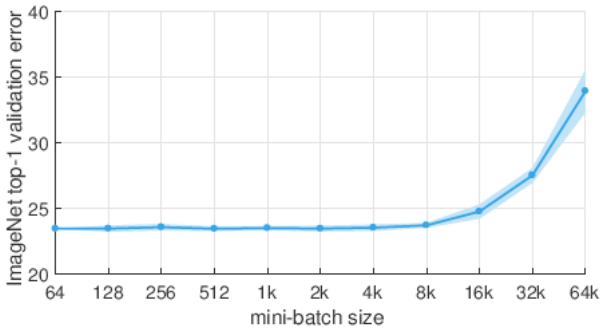
- Sample a subset for gradient. Compromise of SGD and batch GD.
  - Key improvement towards parallelism from SGD.

Algorithm 2 Minibatch Stochastic Gradient Descent with Backpropagation

1: **for** $t = 0$ **to** $\frac{|S|}{B} \cdot epochs$ **do**
2: $\quad \vec{z} \leftarrow$ Sample B elements from S ▷ Obtain samples from dataset
3: $\quad w_{mb} \leftarrow w^{(t)}$ ▷ Load parameters
4: $\quad f \leftarrow \ell(w_{mb}, \vec{z}, h(\vec{z}))$ ▷ Compute forward evaluation
5: $\quad g_{mb} \leftarrow \nabla\ell(w_{mb}, f)$ ▷ Compute gradient using backpropagation
6: $\quad \Delta w \leftarrow u(g_{mb}, w^{(0,\ldots,t)}, t)$ ▷ Weight update rule
7: $\quad w^{(t+1)} \leftarrow w_{mb} + \Delta w$ ▷ Store parameters
8: **end for**



(a) Performance and accuracy of minibatch SGD after a fixed number of epochs (Illustration).



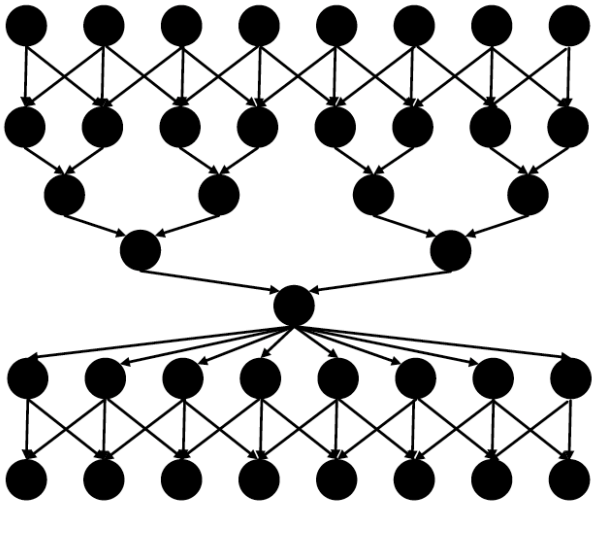
(b) Empirical accuracy (ResNet-50, figure adapted from [83], lower is better).

Fig. 6. Minibatch Size Effect on Accuracy and Performance

Parallelism Primer (W/D)

- DL model: Directed Acyclic Graph (DAG)
  - Work **W**: number of vertices.
  - Depth **D**: longest path in DAG.
- Different best scheme in different settings! (L, G, P def. in paper.)

A primer of relevant parallelism



...

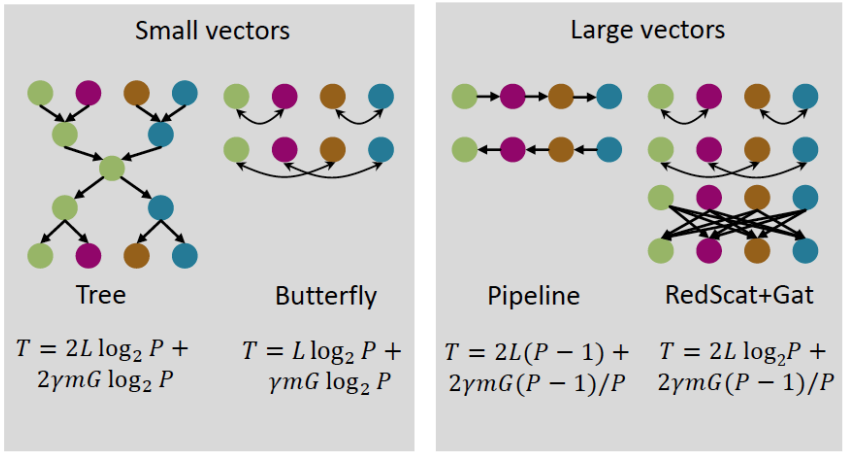
Work W = 39 Depth D = 7

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

and communication theory

Parallel Reductions for Parameter Updates

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



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

E. Chan et al.: Collective communication: theory, practice, and experience. CCPE'07
TH, D. Moor: Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations, JSFI'14

Penn Engineering

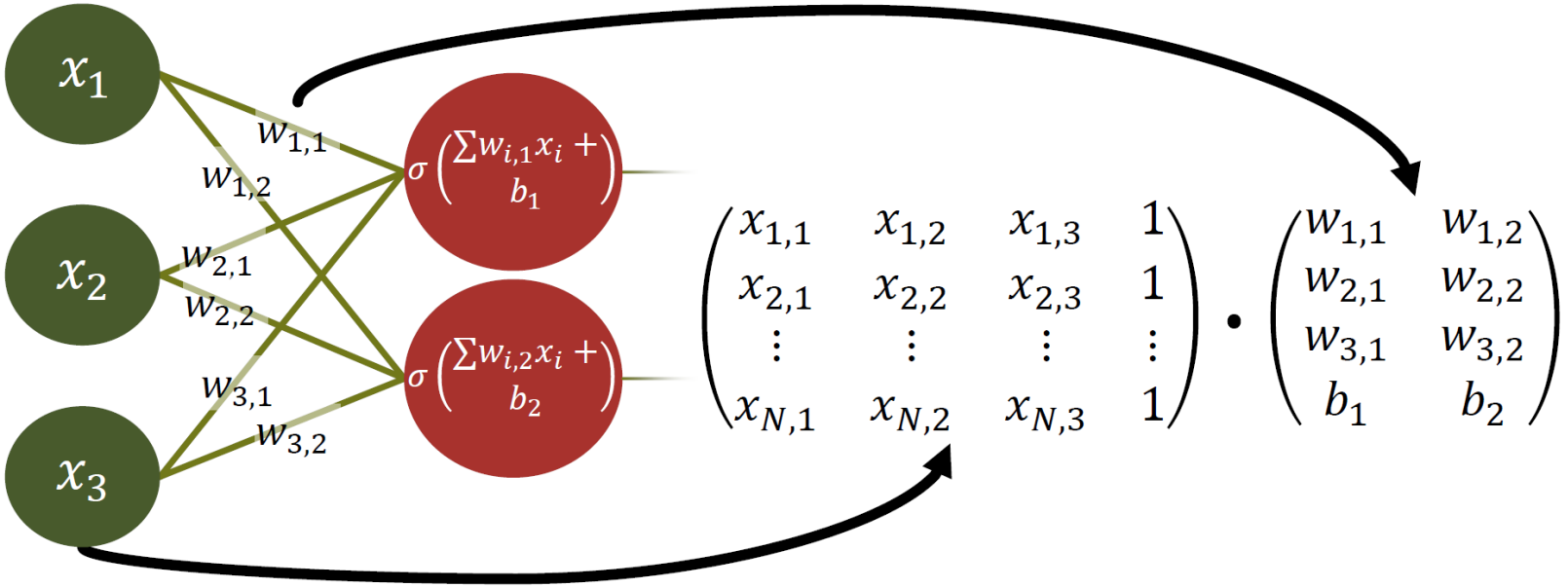
Levels of Parallelism

- Concurrency in Operators
  - Design operator computations to adapt parallelism.
  - Fully connected (Most Common)
  - Convolutional (Takes Images)
  - Recurrent (LSTM, See it in paper.)
- Concurrency in Networks.
  - Evaluate the whole network concurrently w.r.t. dimensions.
  - Data Parallelism
  - Model/Network Parallelism
  - Pipelining
  - Hybrid parallelism.
- Concurrency in Training process.
  - Maybe multiple agents / models / copies of weights w. different values.
  - Issues regarding: consistency, (de-) centralization, param compression, model consolidation, hyper-param/archtecture search.

Cost in Fully-Connected Layers

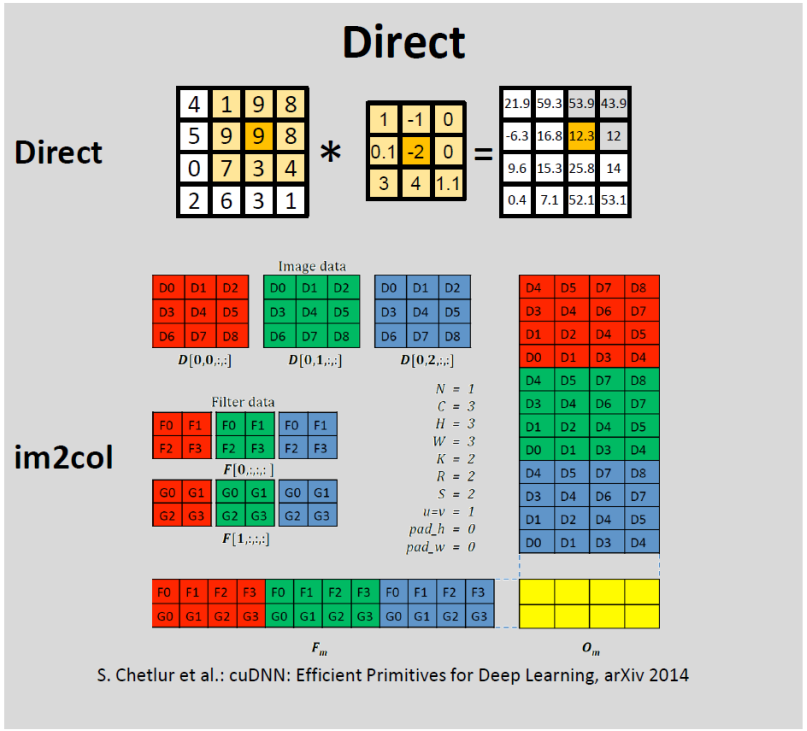
Computing fully connected layers

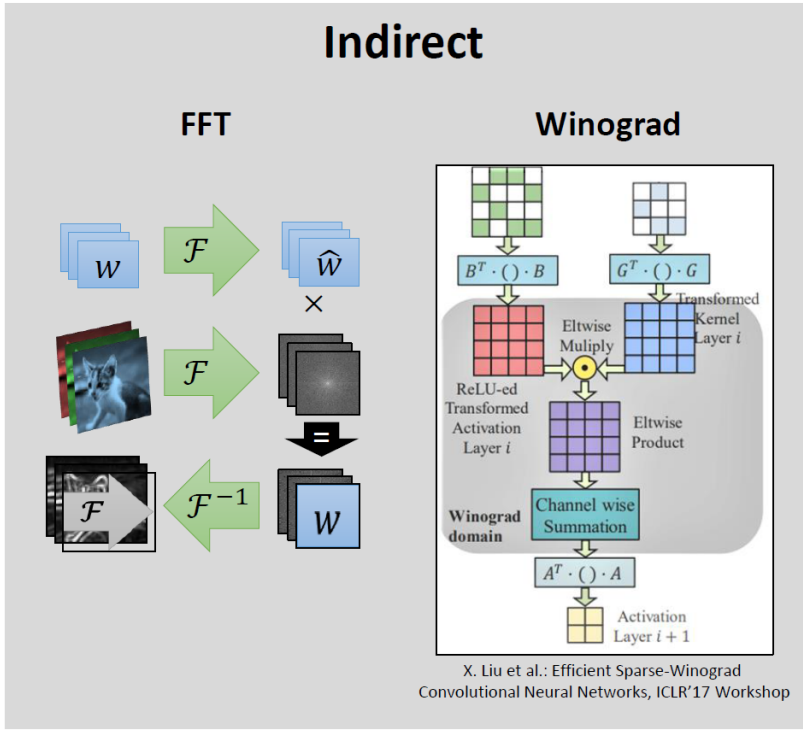
$f_l(x)$	$O(C_{out} \cdot C_{in} \cdot N)$	$O(\log C_{in})$
∇w	$O(C_{in} \cdot N \cdot C_{out})$	$O(\log N)$
∇o_l	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$



- N: # samples; C_{in}: (Channel) input dim; C_{out}: output dim;
- Data Sample Size: Height (H), Width (W).

Cost in Convolutional Layers





K. Chellapilla et al.: High Performance Convolutional Neural Ne
M. Mathieu et al.: Fast Training of Convolutional Networks thr
A. Lavin and S. Gray: Fast Algorithms for Convolutional Neural

Table 6. Work-Depth Analysis of Convolution Implementations

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)	$\alpha(r^2 + \alpha r + 2\alpha^2 + \alpha m + m^2) + C_{out} \cdot C_{in} \cdot P$ ($\alpha \equiv m - r + 1$, $P \equiv N \cdot \lceil H/m \rceil \cdot \lceil W/m \rceil$)	$2\lceil \log_2 r \rceil + 4\lceil \log_2 \alpha \rceil + \lceil \log_2 C_{in} \rceil$

Penn Engineering

Work-Depth Characteristics

Table 4. Asymptotic Work-Depth Characteristics of DNN Operators

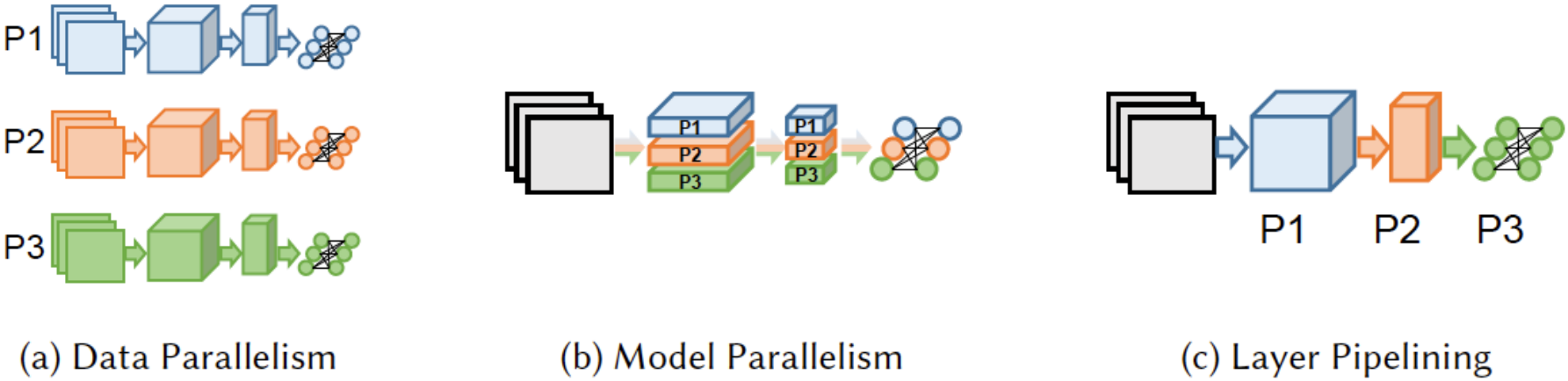
Operator Type	Eval.	Work (W)	Depth (D)
Activation	y	$O(NCHW)$	$O(1)$
	∇w	$O(NCHW)$	$O(1)$
	∇x	$O(NCHW)$	$O(1)$
Fully Connected	y	$O(C_{out} \cdot C_{in} \cdot N)$	$O(\log C_{in})$
	∇w	$O(C_{in} \cdot N \cdot C_{out})$	$O(\log N)$
	∇x	$O(C_{in} \cdot C_{out} \cdot N)$	$O(\log C_{out})$
Convolution (Direct)	y	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇w	$O(N \cdot C_{out} \cdot C_{in} \cdot H' \cdot W' \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
	∇x	$O(N \cdot C_{out} \cdot C_{in} \cdot H \cdot W \cdot K_x \cdot K_y)$	$O(\log K_x + \log K_y + \log C_{in})$
Pooling	y	$O(NCHW)$	$O(\log K_x + \log K_y)$
	∇w	–	–
	∇x	$O(NCHW)$	$O(1)$
Batch Normalization	y	$O(NCHW)$	$O(\log N)$
	∇w	$O(NCHW)$	$O(\log N)$
	∇x	$O(NCHW)$	$O(\log N)$

- Work is linear and Depth logarithmic –large average parallelism
  - In table W is different! (Width)
  - Theory does not correspond to computation time in practice.

Concurrency in Networks

- Evaluate the whole network concurrently w.r.t. different dimensions.
  - e.g. Partition forward evaluation and backpropagation among processors.

- One model, large dataset, large network, multiple machines.
  - Types: Data, Model, Pipelining, Hybrid
  - Communication design is essential.



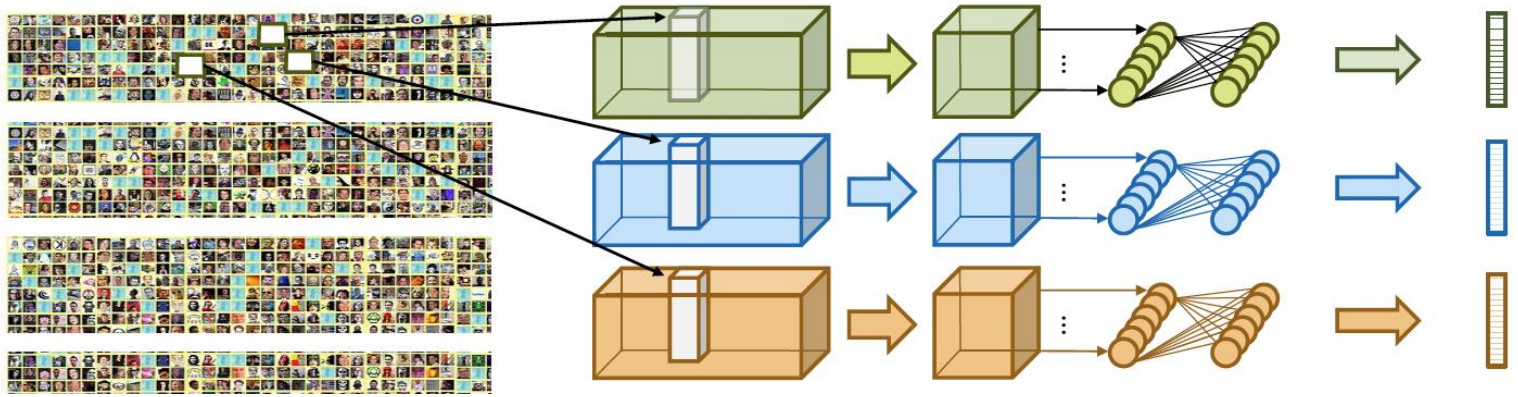
(a) Data Parallelism (b) Model Parallelism (c) Layer Pipelining

Fig. 14. Neural Network Parallelism Schemes

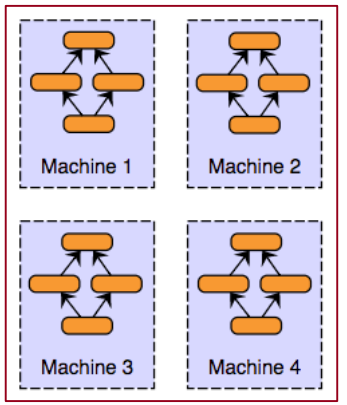
- External Reference: https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/

Data Parallelism

- Partition the work of mini-batch samples among multiple resources.
  - Scaling is naturally defined by mini-batch size (tradeoff!)
  - Model and parameters are duplicated to all machines.
  - Intermediate results have to be gathered & averaged in update phase.

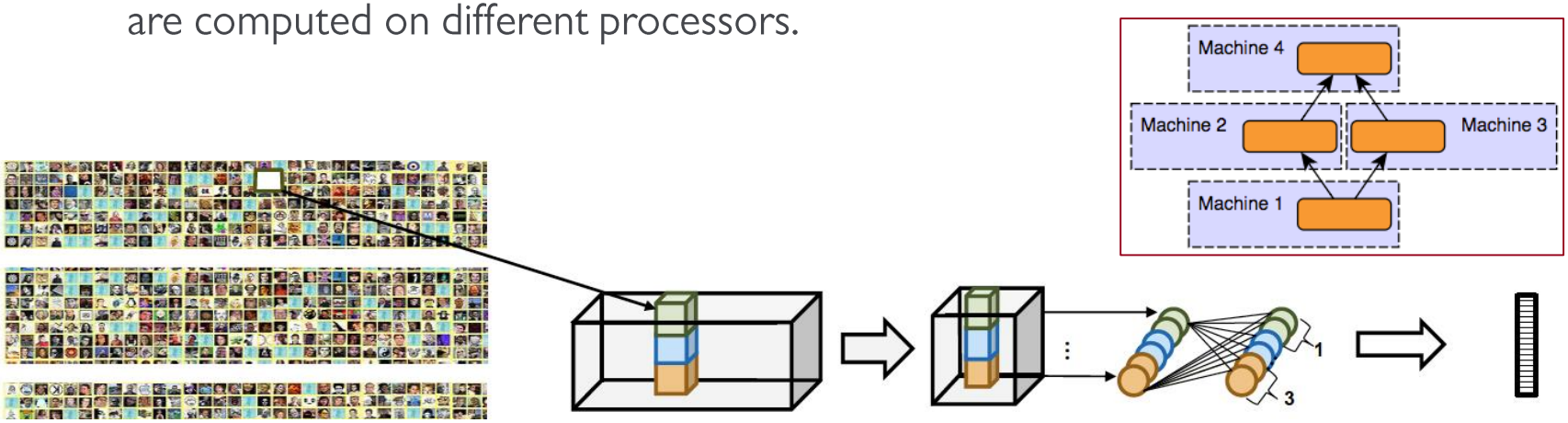


- Pros: Simple & efficient, Easy to implement.
- Cons: Synchronization is a problem.
- Instances and solutions:
  - Small independent batch (BN). Redesign weights. (WN).
  - Coarse/Fine Grain: ParallelSGD. MapReduce/Spark.



Model Parallelism

- Divides the work according to the neurons in each layer.
  - Sample mini-batch is copied to all processors, and different parts of the DNN are computed on different processors.

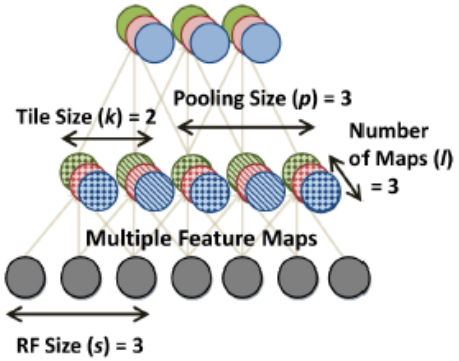


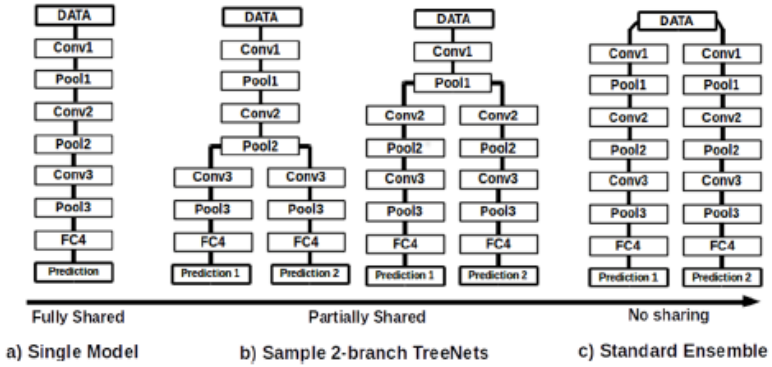
- Pros: Neurons/Parameters can be distributed. No batch size tradeoff.
- Cons: Layer dependencies generates heavy communication costs.
  - Mini-batch data copied to each processor
  - Requires all-to-all communication between layers.
  - CNN is hard to divide efficiently

Solution to Model Parallelism

- For DNN fully-connected layer: Redundant computation
  - Neuron overlap

- For Convolutional Layers: Refine Structure
  - Introduce local filter: LCN
  - Incorporate partial Ensembles: TreeNets





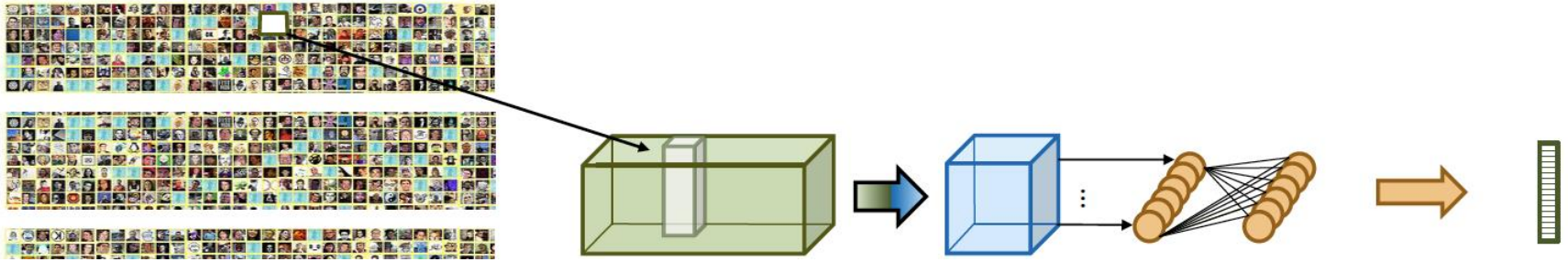
(a) Locally Connected Networks [180]

(b) TreeNets [148]

Fig. 16. Model Parallelism Schemes

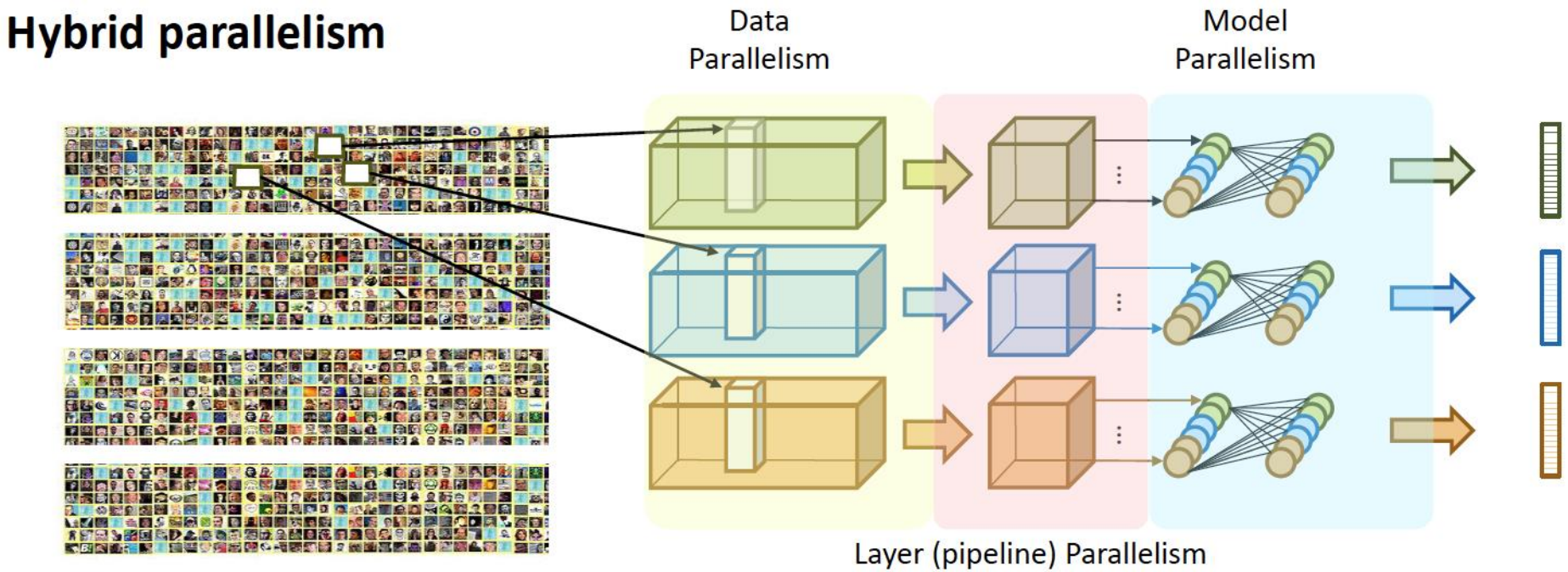
Pipeline Parallelism

- Special Case of Data Parallelism. Either refer to :
  - Overlapping computations, i.e., between one layer and the next (as data becomes ready); OR to
  - Partitioning the DNN according to depth, assigning layers to processors.



- Pros:
  - Layers/parameters can be distributed across processors.
  - Sparse communication pattern allows further optimization.
- Cons: Mini-batches have to be copied.
- Instance: Deep Stacking Network.

Hybrid Parallelism

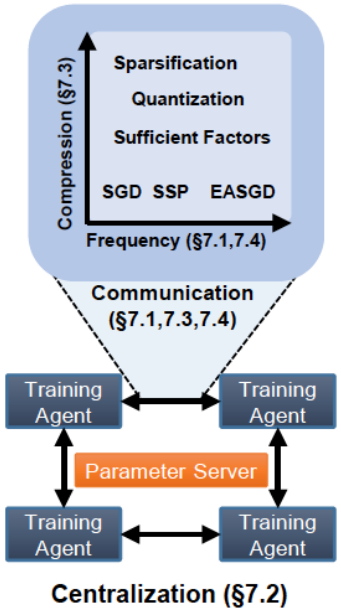


- Combines the advantages.
  - Distribute neurons along with data mini-batches.
- Often specific to layer-types
  - (e.g., distribute fc layers but handle conv layers data-parallel)
- Enables arbitrary combinations of data, model, and pipeline parallelism

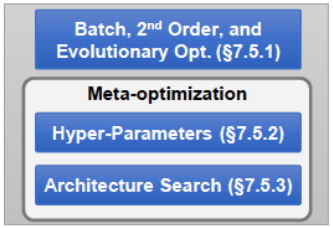
Concurrency in Training

- Maybe multiple instances of SGD agents running independently.
  - Multiple different copies of weights. (updates not instantly visible)
  - System design issues. OR -- Even multiple models.

Category	Method
Model Consistency	
Synchronization	Synchronous [45,112,173,210,224,246]
	Stale-Synchronous [88,98,120,156,261]
	Asynchronous [55,56,127,182,190,205,260]
	Nondeterministic Comm. [54,121,201]
Parameter Distribution and Communication	
Centralization	Parameter Server (PS) [52,112,128,152]
	Sharded PS [39,56,120,137,142,244,254,255]
	Hierarchical PS [88,107,253]
	Decentralized [9,54,121,155]
Compression	Quantization [5,35,50,51,55,63,87,90,109,132,151,202,210,237,265]
	Sparsification [3,32,67,158,206,214,224]
	Other Methods [41,106,111,129,150,243,254]
Training Distribution	
Model Consolidation	Ensemble Learning [114,148,217]
	Knowledge Distillation [10,96]
	Model Averaging: Direct [20,33,168,266], Elastic [121,155,248,258], Natural Gradient [11,195]
Optimization Algorithms	First-Order [43,123,126,141,145,163,227]
	Second-Order [11,28,56,94,133,165,172]
	Evolutionary [169,192,234,242]
	Hyper-Parameter Search [13,89,91,118,131,163,169,219,256]
	Architecture Search: Reinforcement [12,193,264,268,269], Evolutionary [160,203,204,242,251], SMBO [27,71,159,161,176]



Centralization (§7.2)



Distributed Optimization (§7.5)

Penn Engineering

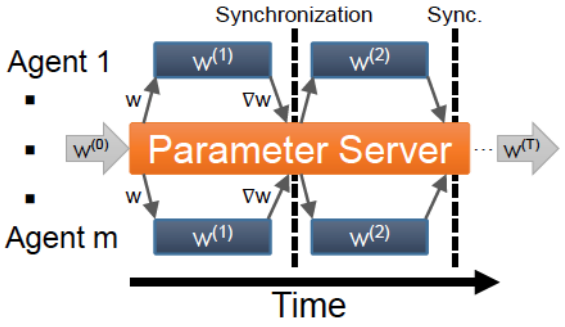
Model Consistency

- Synchronization is always a problem in distributed systems. Tradeoffs!
  - Enforce strong synchronization hinders scaling.
  - Loose synchronization degrades accuracy and prevent convergence.
- Recent works creates inconsistent model that relax restriction.

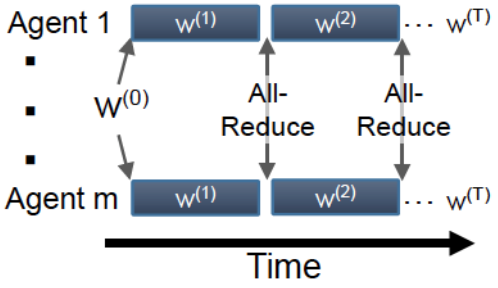
- Staleness:

Training agent i
At time t contains
Weights w (τ,i)

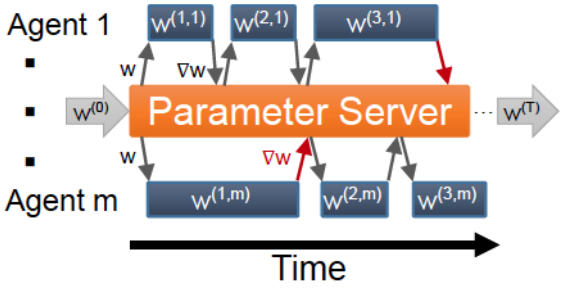
Stale: t - τ



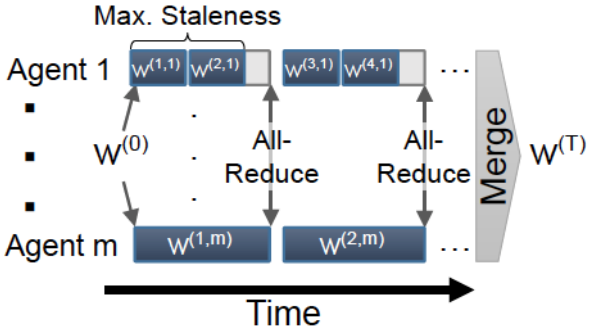
(a) Synchronous, Parameter Server



(b) Synchronous, Decentralized



(c) Asynchronous, Parameter Server



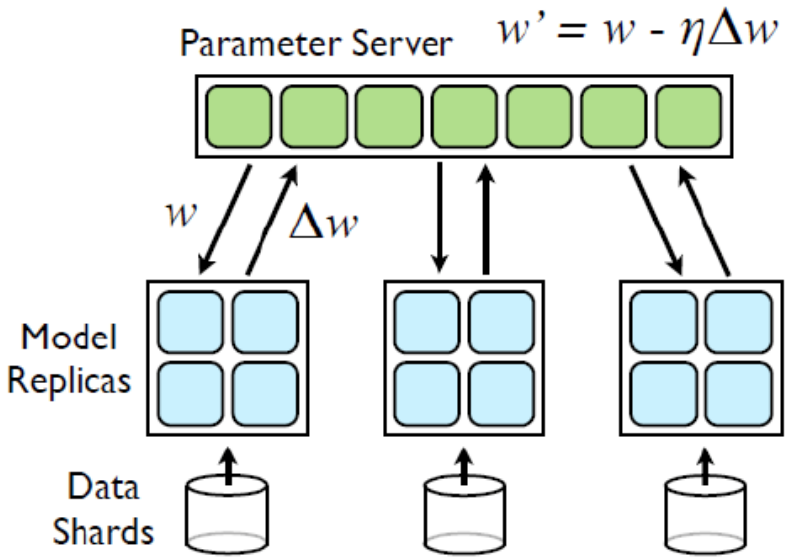
(d) Stale-Synchronous, Decentralized

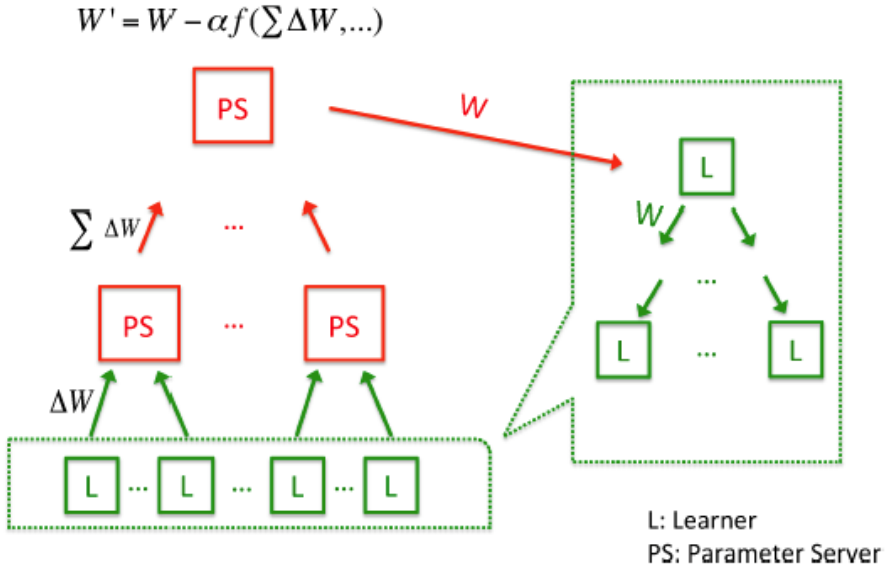
Penn Engineering

Fig. 20. Training Distribution in Deep Learning (Model Consistency, Centralization)

Centralization

- Utilize a Parameter Server to update weights. Reduce-and-Broadcast.
- Easy consistency. Easy to modify nodes. Handles heterogeneity.
- Server communication bottleneck. Single point of failure. Slow update.
- Some Improvements:
  - Replicate parameter servers (w. hierarchy).
  - Use Paxos gossiping model between candidate PS.

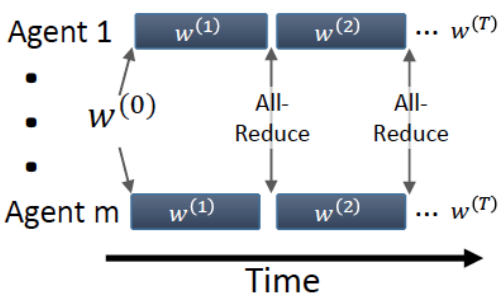




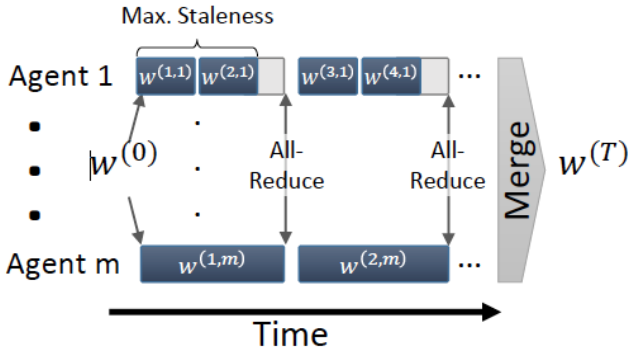
Penn Engineering

Decentralization

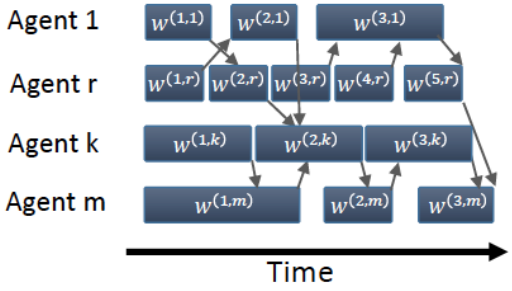
- Parameter exchange frequency can be controlled, still attaining convergence.



Synchronous



Stale Synchronous / Bounded Asynchronous



Asynchronous

- Needs careful design and tuning in structure, protocol and rates.
  - Robustness is also an issue.

- May also consider limited/slower distribution
  - Gossip Algorithm limits a group of nodes to communicate

Model Consolidation

- Model Averaging: Running multiple SGD on different machines, allow different gradient exploring schemes.
- Ensemble Learning: Average the output of different networks.

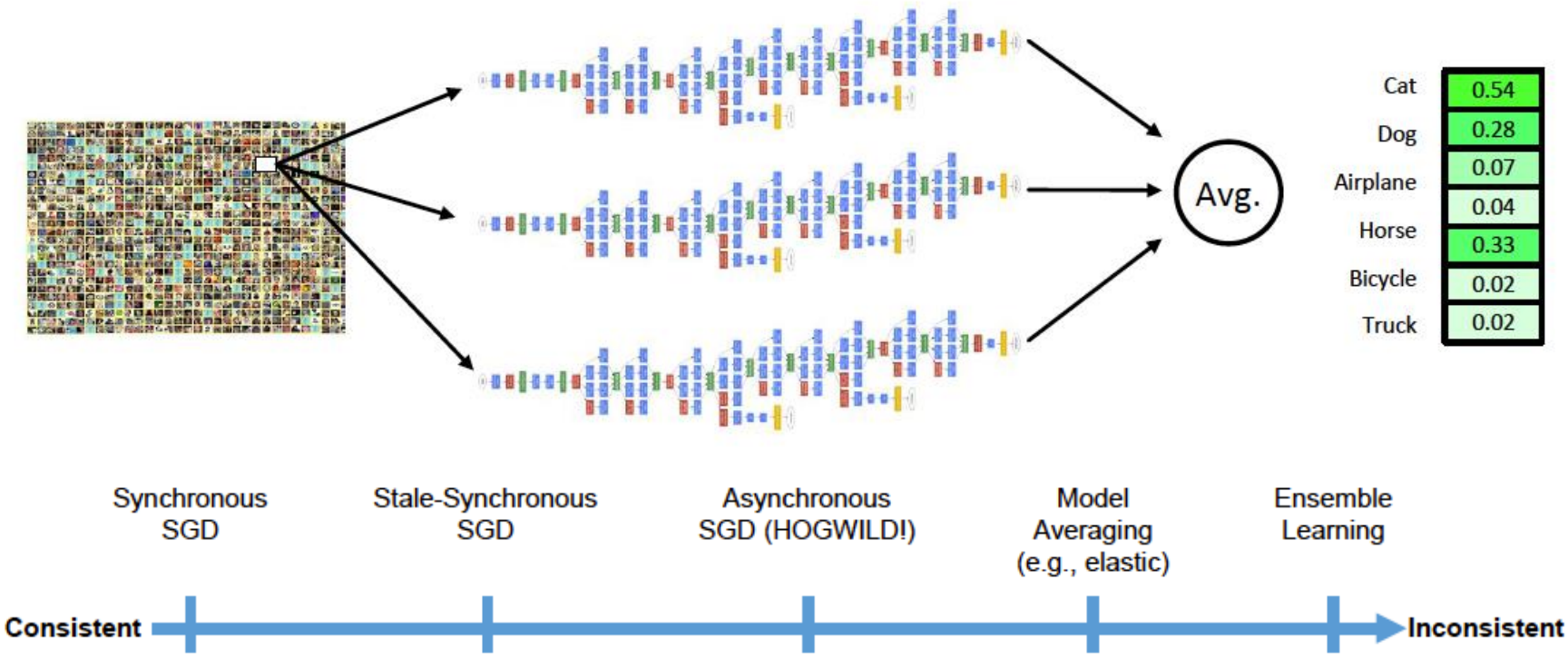


Fig. 23. Parameter Consistency Spectrum

Communication Optimization

- **Different options how to optimize updates**
  - Send ∇w, receive w
  - Send FC factors (o_{l-1}, o_l), compute ∇w on parameter server

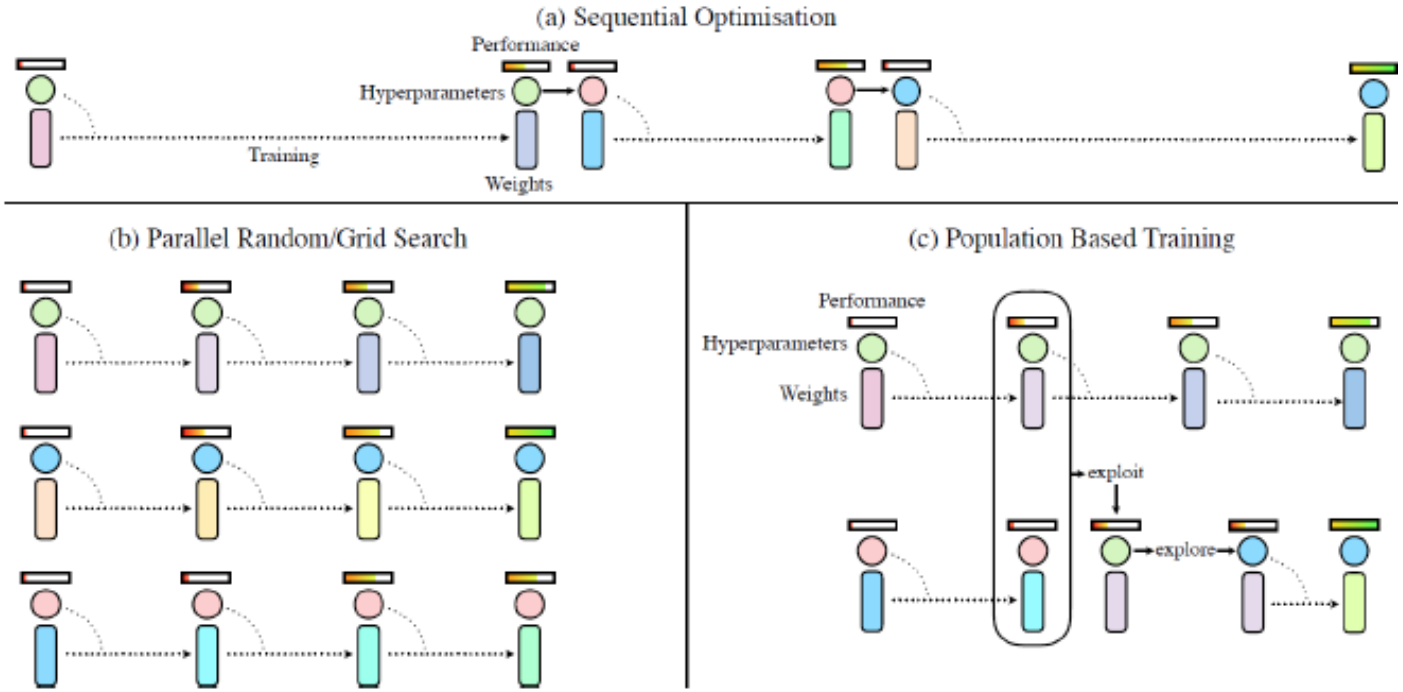
    Broadcast factors to not receive full w
  - Use lossy compression when sending, accumulate error locally!
- **Quantization**
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate

    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
  - Or to prune network

Hyper-Param Search

- Meta-optimization of hyper-params and DNN architecture
  - Common Grid Search and Sampling
  - Genetic Algorithms with modified (specialized) mutations
  - Other meta-heuristics
  - Using Reinforcement Learning (explore/exploit different configurations)



Take-away for Future

- Overview of various parallel topics in supervised DL.
- Accuracy is not the only consideration.
- Mobile DL applications are in need.
  - Require Computation, memory and energy efficient.
- Single-Operator has already been highly optimized.
  - Focus on inter-layer and overall optimization.
- Standard DL/ML interface can be redesigned.
- Multi-purpose networks / AutoML will be popular.

Thanks!

Penn Engineering