10-605/10-805: Machine Learning with Large Datasets

Fall 2022

Parallel & Distributed Deep Learning

Announcements

- HW4b due today
  - *remember to terminate your instances!!!*
  - Recitation next week will cover HW4 solutions
- HW5 released today, due November 15

Key course topics

Data preparation
- Data cleaning
- Data summarization
- Visualization
- Dimensionality reduction

Training
- Distributed ML
- Large-scale optimization
- **Scalable deep learning**
- Efficient data structures
- Hyperparameter tuning

Inference
- Hardware for ML
- Techniques for low-latency inference (compression, pruning, distillation)

Infrastructure / Frameworks
- Apache Spark
- TensorFlow
- AWS / Google Cloud / Azure

Advanced topics
- Federated learning
- Neural architecture search
- Productionizing ML

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

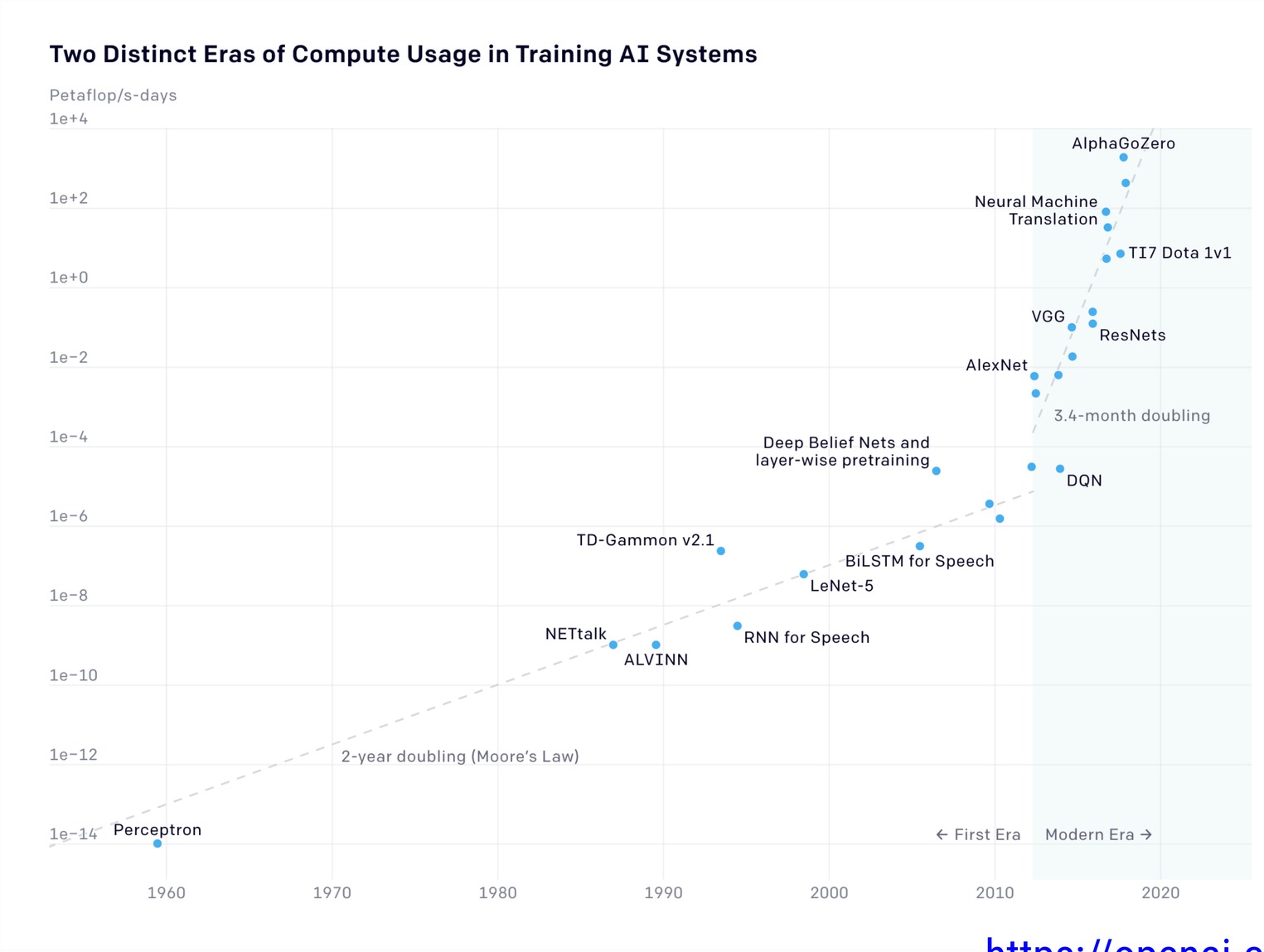
Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

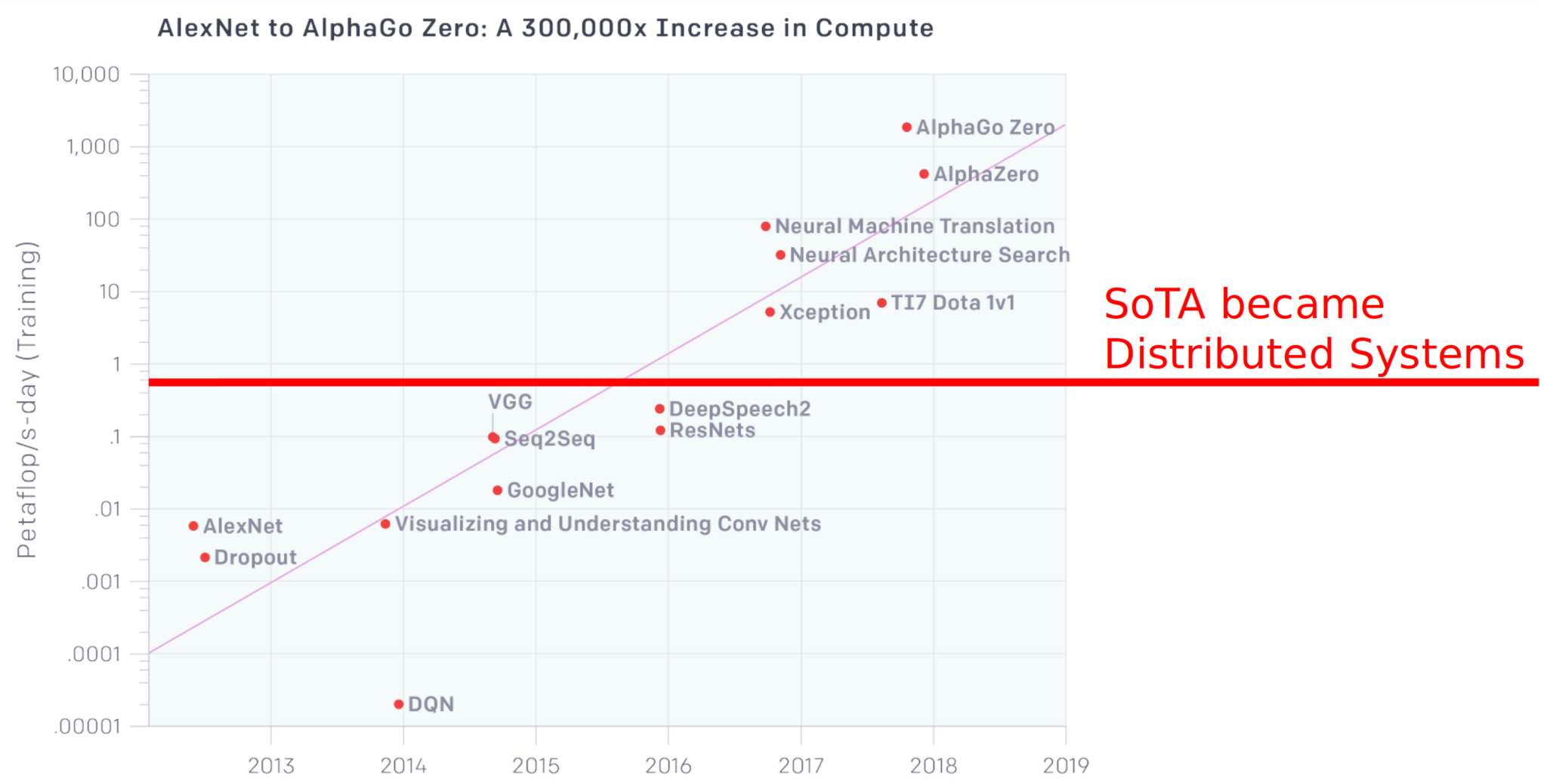
- Can be expensive to store model, perform inference

Motivation: parallel & distributed DL



https://openai.com/blog/ai-and-compute/

Motivation: parallel & distributed DL



SoTA became Distributed Systems

https://openai.com/blog/ai-and-compute/

Outline

1. Scaling up Mb-SGD: Communication schemes
2. Challenges with mini-batching

RECALL

Techniques for large-scale optimization

Reduce computation:

- Parallelize/distribute computation
- Use first-order methods (i.e., no 2nd-order or higher gradients)
- Use stochastic methods

***Reduce communication*:**

- Keep large objects local
- Reduce iterations

RECALL

Techniques for large-scale optimization

Reduce computation:

- Parallelize/distribute computation
- Use first-order methods (i.e., no 2nd-order or higher gradients)
- Use stochastic methods

***Reduce communication*:**

- Keep large objects local
- Reduce iterations

RECALL

Techniques for large-scale optimization

Reduce computation:

- Parallelize/distribute computation
- Use first-order methods (i.e., no 2nd-order or higher gradients)
- Use stochastic methods

***Reduce communication*:**

- Keep large objects local
- Reduce iterations

RECALL

Techniques for large-scale optimization

Reduce computation:

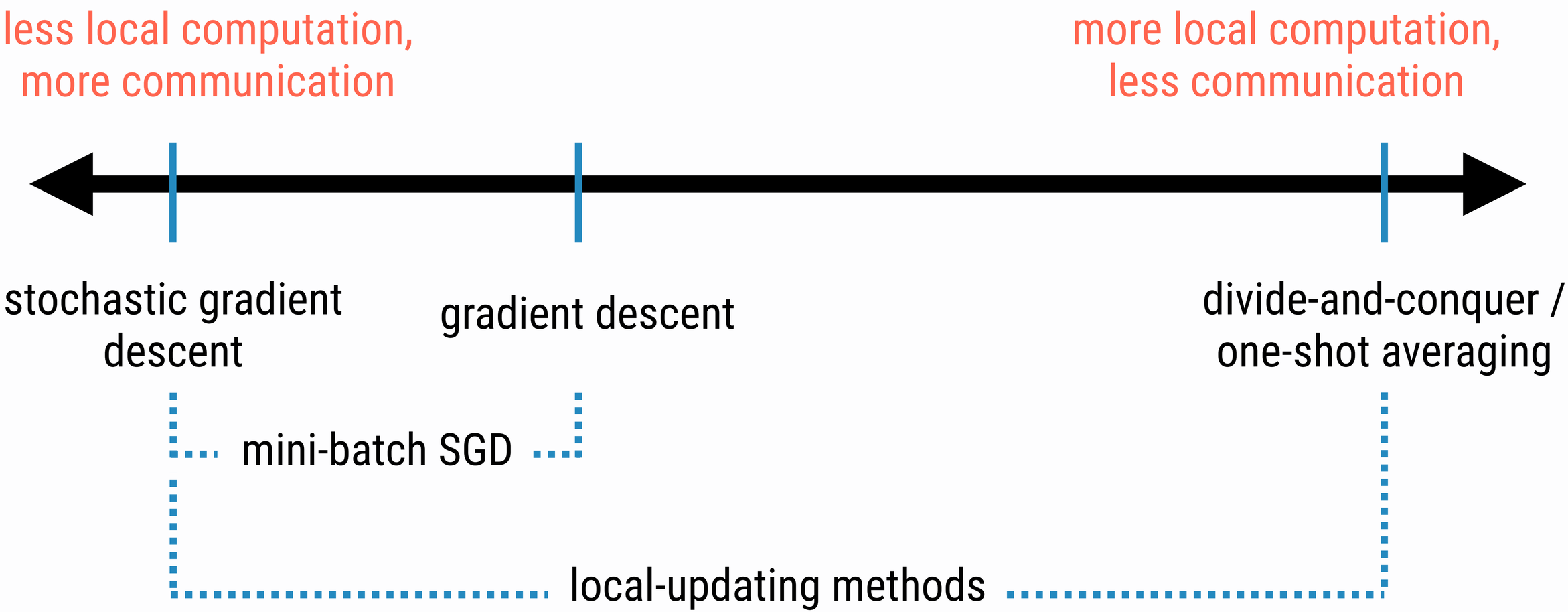
- Parallelize/distribute computation
- Use first-order methods (i.e., no 2nd-order or higher gradients)
- Use stochastic methods

***Reduce communication*:**

- Keep large objects local
- Reduce iterations
- Asynchrony, communication strategy/architecture, compression

RECALL

Methods for distributed optimization



RECALL

Mini-batch SGD

Objective we want to solve:

$$\min_{\mathbf{w}} f(\mathbf{w}) \quad \text{where} \quad f(\mathbf{w}) := \sum_{j=1}^{n} f_j(\mathbf{w})$$

Gradient Descent Update:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \nabla f(\mathbf{w}_i)$$

Stochastic Gradient Descent (SGD) Update:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \nabla f_j(\mathbf{w}_i)$$

with j sampled at random

Mini-batch SGD Update:

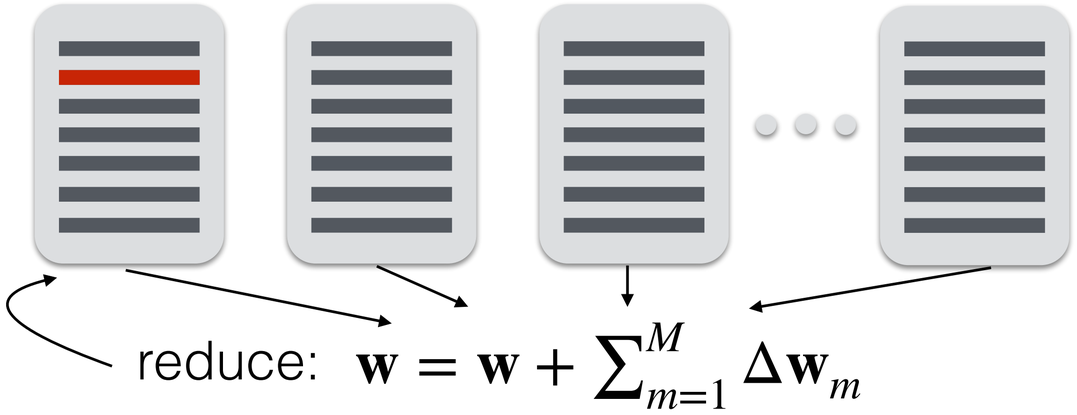
$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \nabla f_{\mathcal{B}_i}(\mathbf{w}_i)$$

with mini-batch $\mathcal{B}_i \subseteq \{1, \ldots n\}$ sampled at random

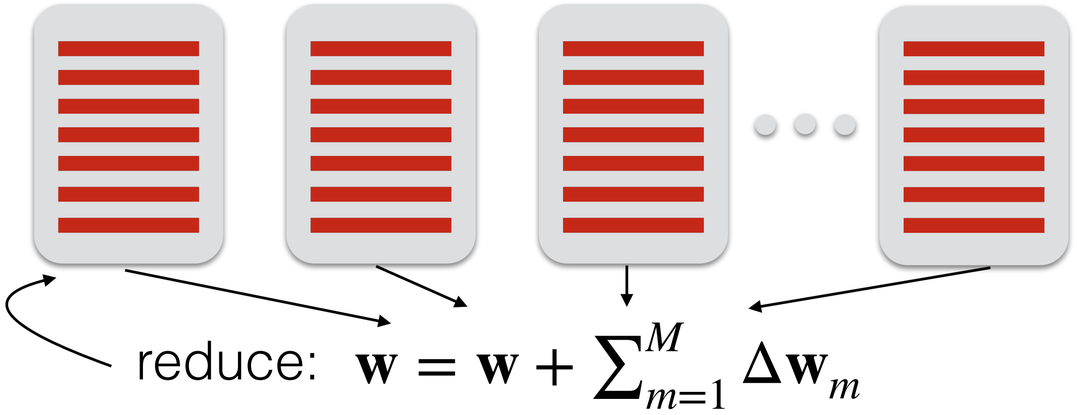
RECALL

Mini-batch SGD

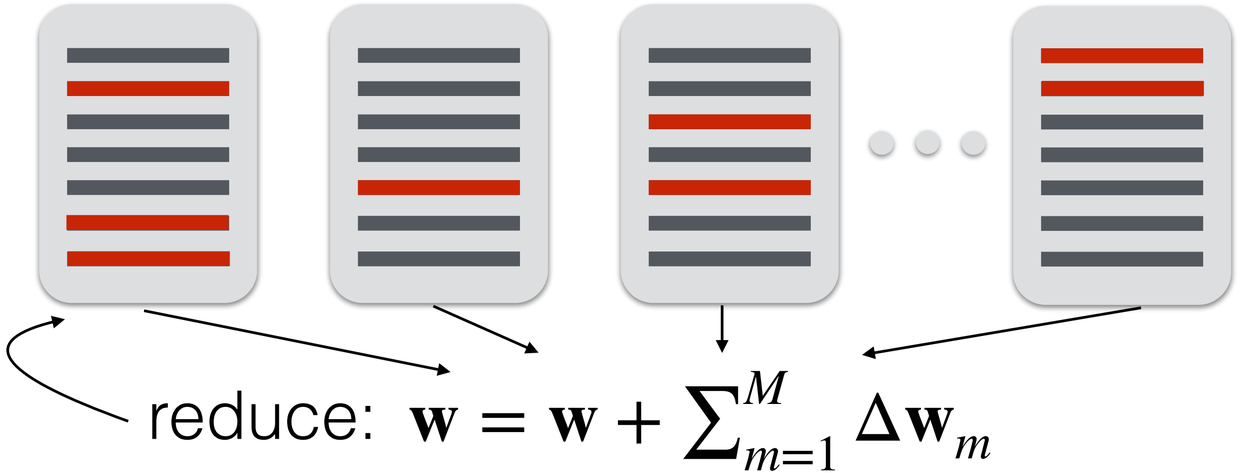
SGD: one observation



gradient descent: all observations



mini-batch SGD: some observations



QUESTION

How to parallelize mini-batch SGD?

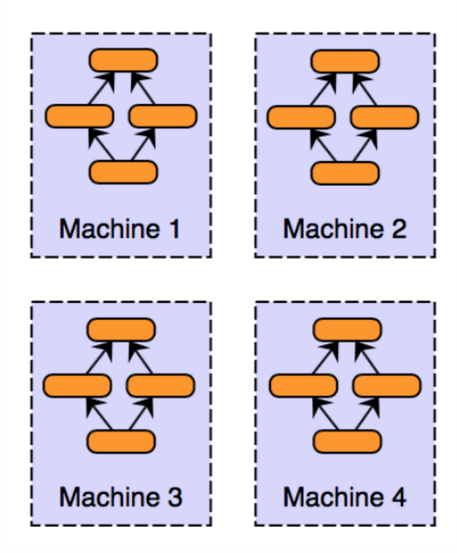
Parallelization scheme can vary based on:

- synchronization model (*when do we send updates?*)
- communication strategy (*where do we send them?*)
- compression (*what updates do we send?*)

**Note: While we motivate these approaches with mini-batch SGD, they are applicable to the other first-order methods discussed previously (AdaGrad, RMSProp, Adam, etc.)*

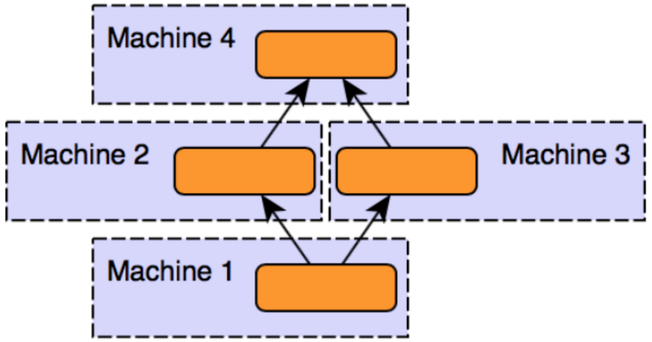
FIRST, A NOTE:

Data parallel vs. model parallel



Data parallel

- Replicate the model on each machine
- Each machine accesses a portion of the dataset
- C*ommon approach*: typically preferable in terms of implementation, fault tolerance, and parallelizability

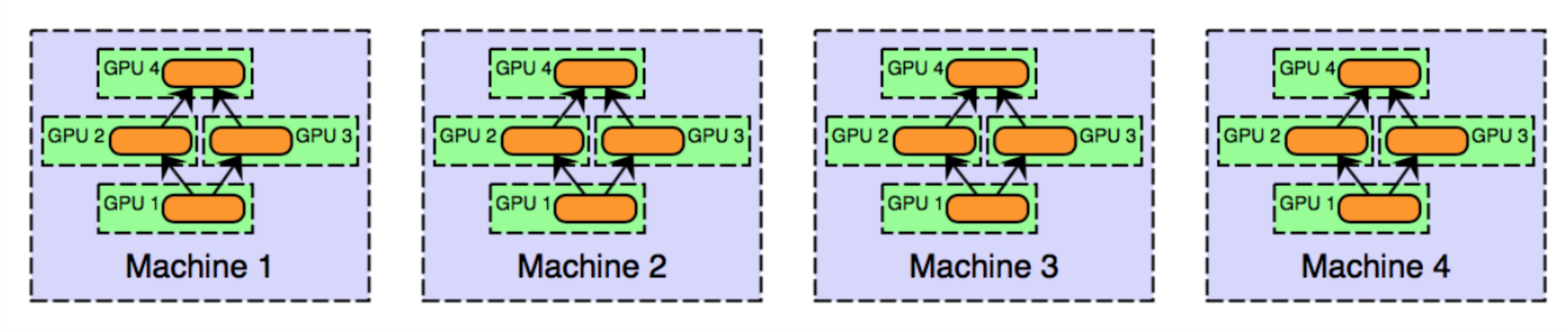


Model parallel

- Replicate the data on each machine
- Each machine accesses a portion of the model
- An attractive approach for massive models

FIRST, A NOTE:

Data parallel vs. model parallel

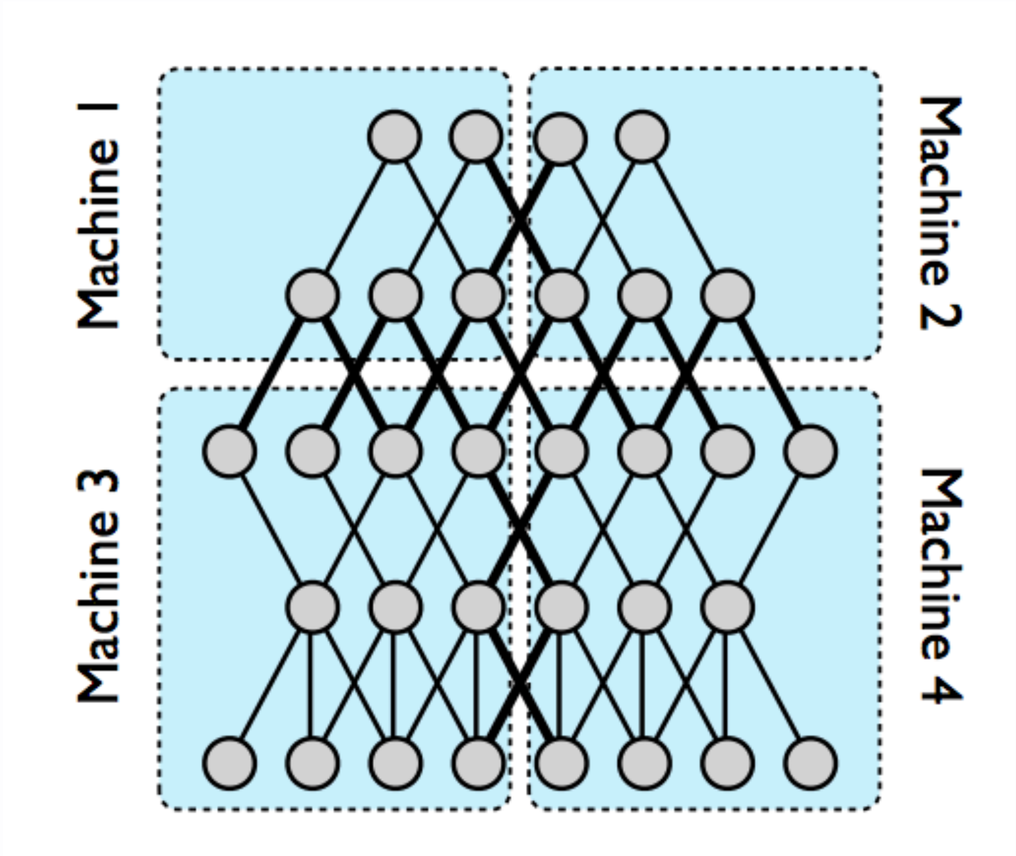


Can also combine approaches …

- Model parallel across GPUs on a single machine
- Data parallel across machines in a cluster

FIRST, A NOTE:

Model parallel



Example: *DistBelief* at Google [2012]

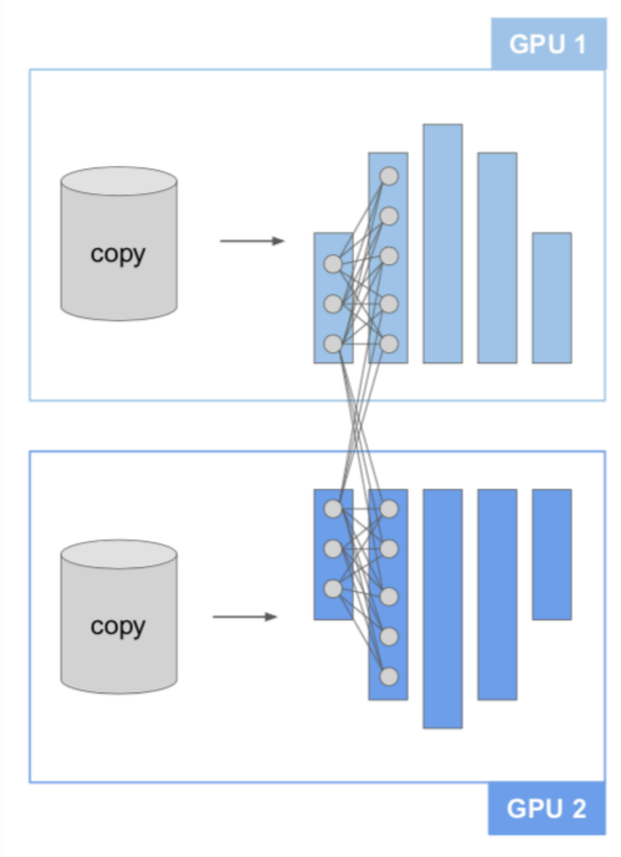
- Bold connections represent communication

Dean et. al., 2012

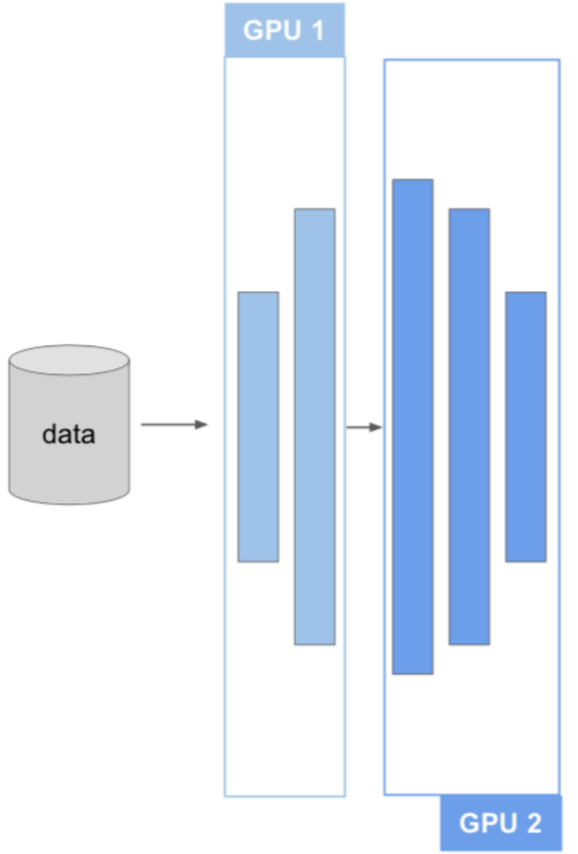
FIRST, A NOTE:

Model parallel

intra-layer parallelism: units within one layer are partitioned

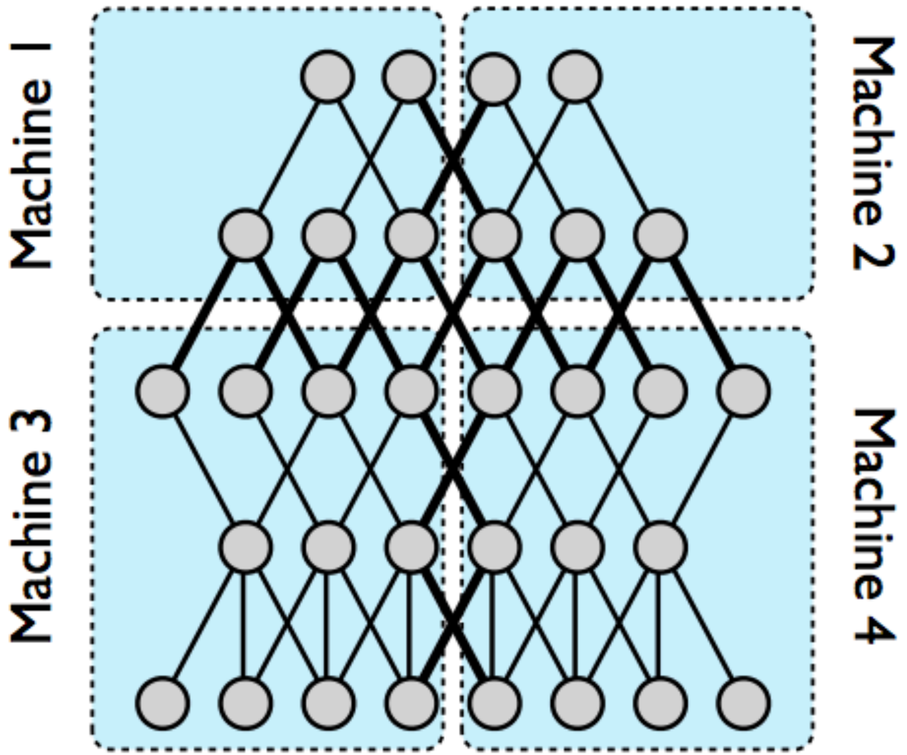


inter-layer parallelism: entire layers are partitioned



FIRST, A NOTE:

Model parallel



Example: *DistBelief* at Google [2012]

- Bold connections represent communication

Google combined this approach with data parallelism (using a parameter server)

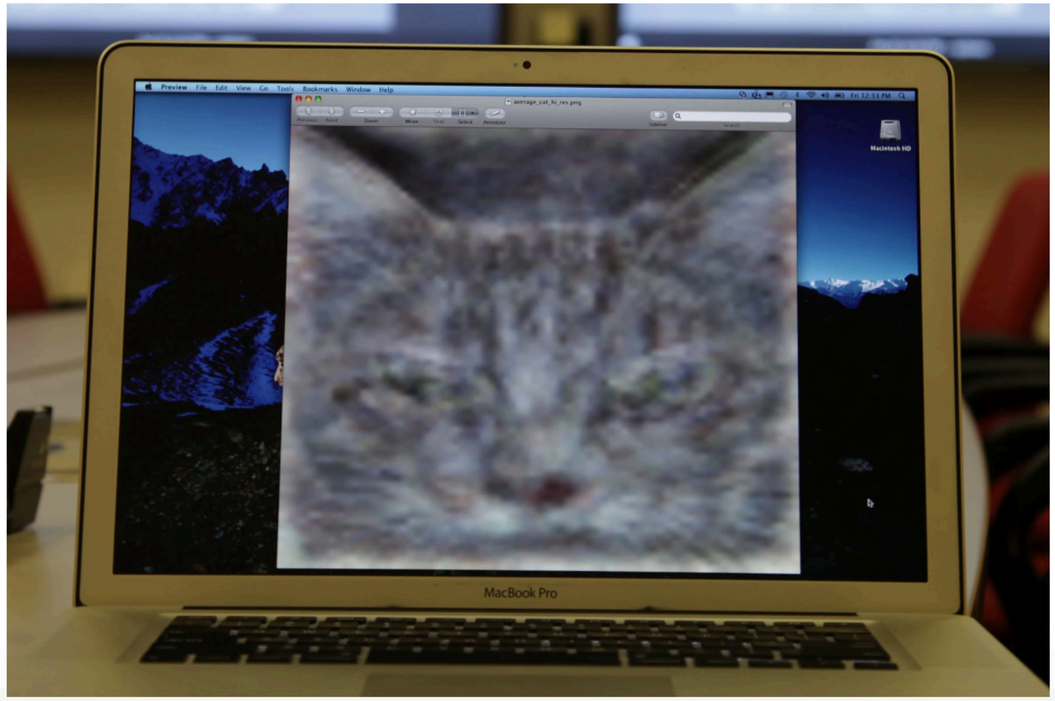
Dean et. al., 2012

FIRST, A NOTE:

Model parallel

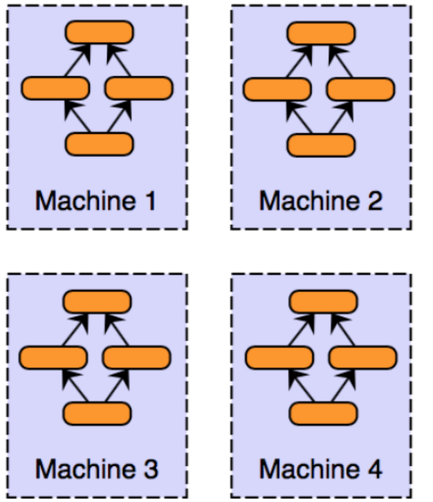
The New York Times

How Many Computers to Identify a Cat? 16,000



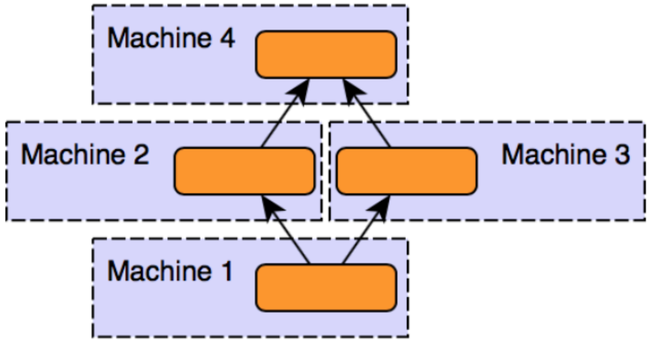
FIRST, A NOTE:

Data parallel vs. model parallel



Data parallel

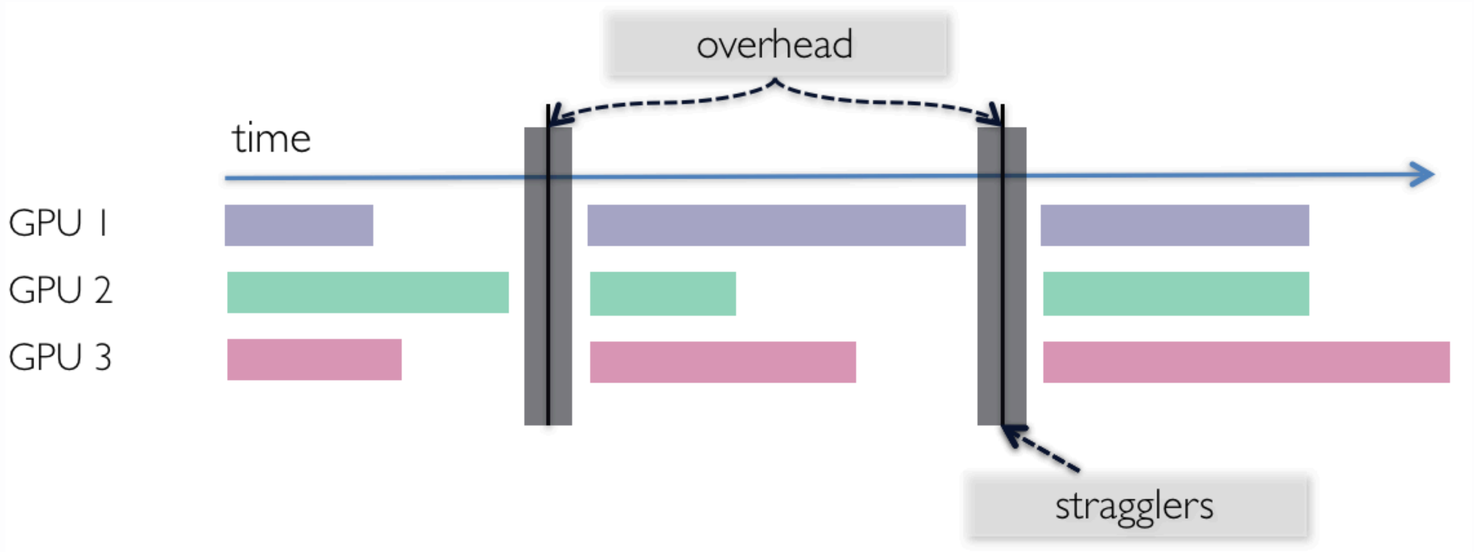
- Replicate the model on each machine
- Each machine accesses a portion of the dataset
- *Most common approach*: typically preferable in terms of implementation, fault tolerance, and parallelizability



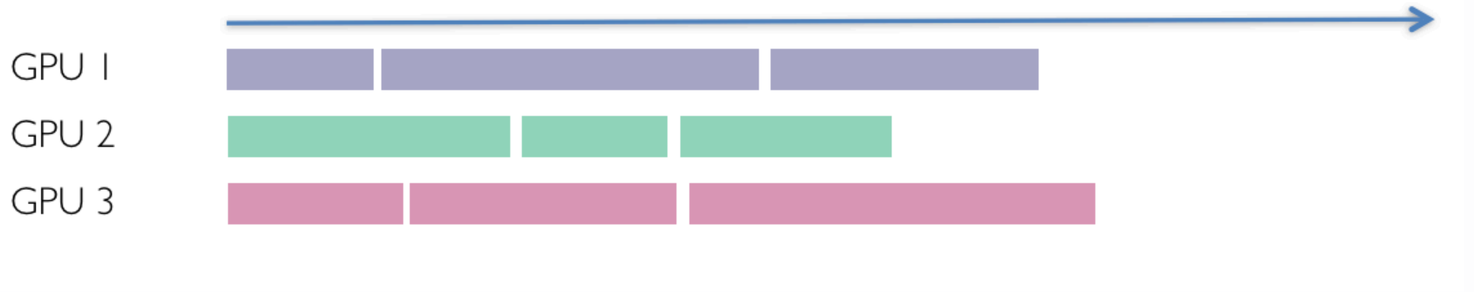
Model parallel

- Replicate the data on each machine
- Each machine accesses a portion of the model
- An attractive approach for massive models

Synchronous vs. asynchronous



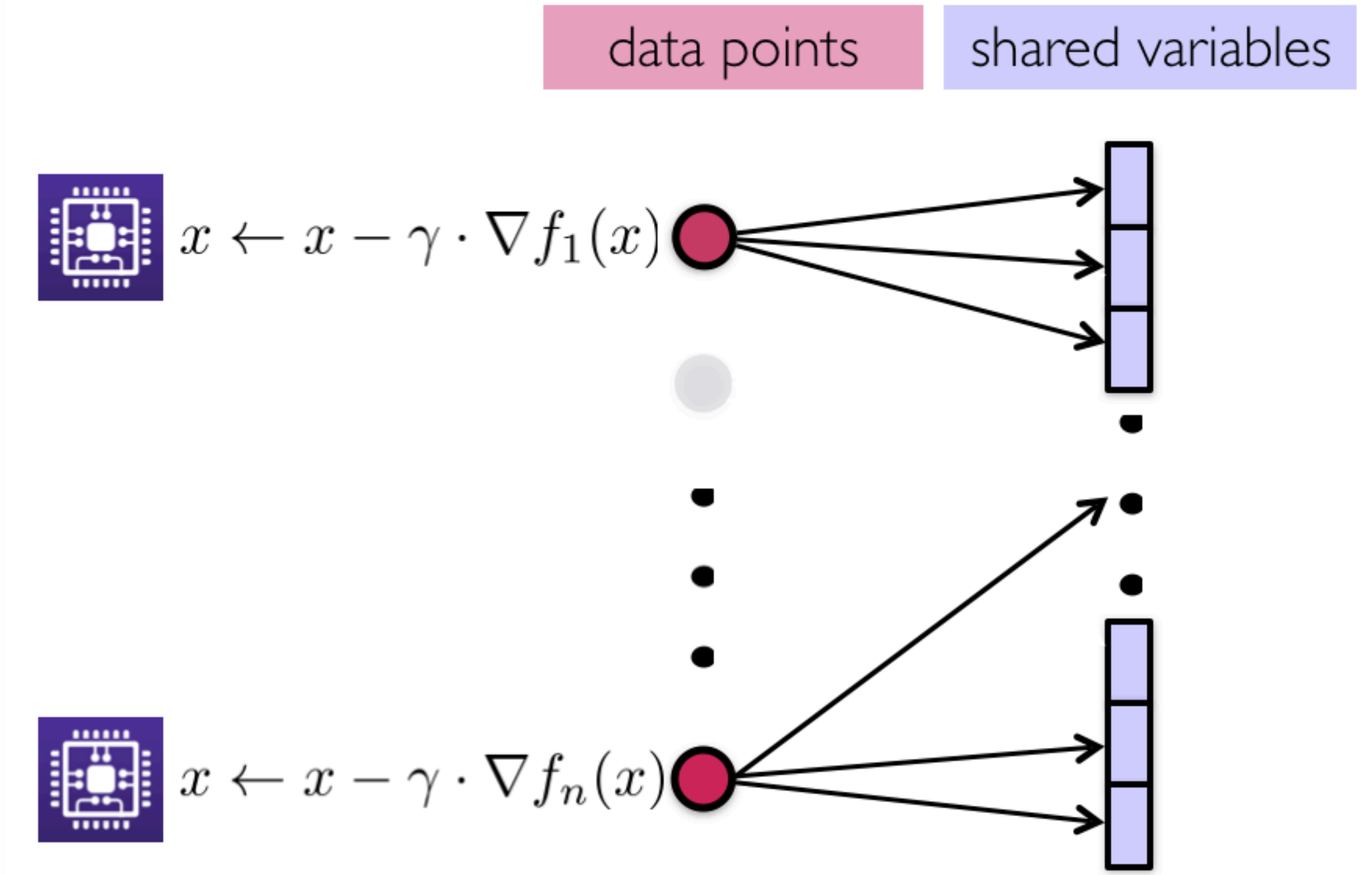
synchronous
- guaranteed to be correct
- but: can be slow due to stragglers, overhead



asynchronous
- fast—no waiting!
- but: there may be errors

Image: Dimitris Papailiopoulos

Asynchronous ML

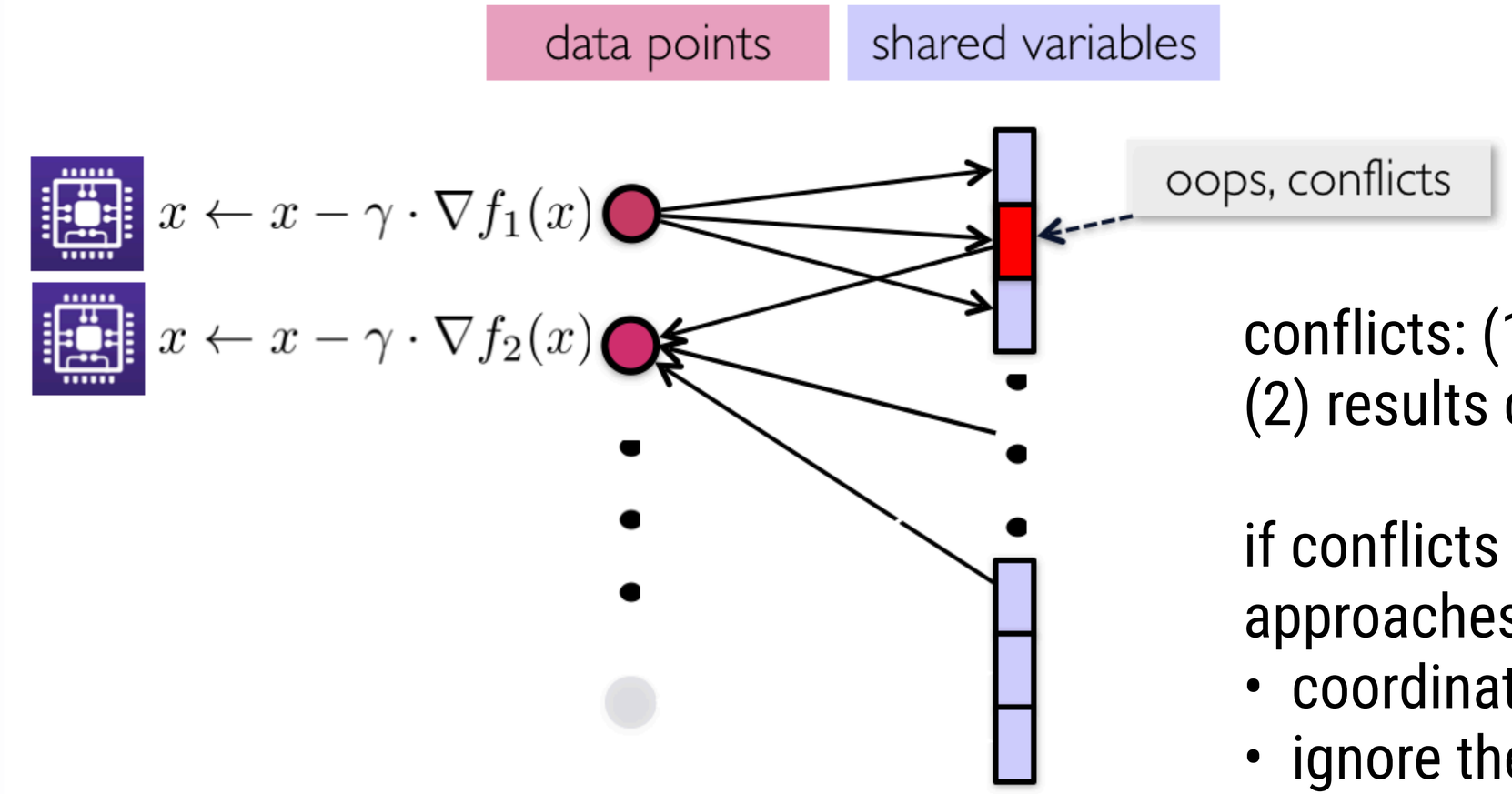


if no conflicts (e.g., when the shared parameter, *x*, is very sparse), asynchrony is great!

- 2 parallel iterations = 2 serial iterations

Image: Dimitris Papailiopoulos

Asynchronous ML



conflicts: (1) updates can be stale, (2) results can be overwritten

if conflicts occur, two general approaches:

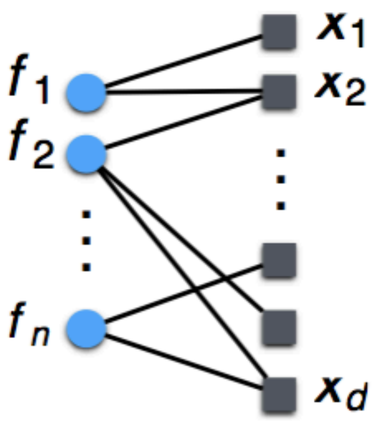
- coordinate or lock
- ignore the conflict (lock-free)

Image: Dimitris Papailiopoulos

Hogwild!

Main idea: run asynchronous parallel SGD *without locks*

- assuming problem is sufficiently sparse, can prove the method converges
- works well in practice for ML applications, which can tolerate a lot of noise



Each processor in parallel

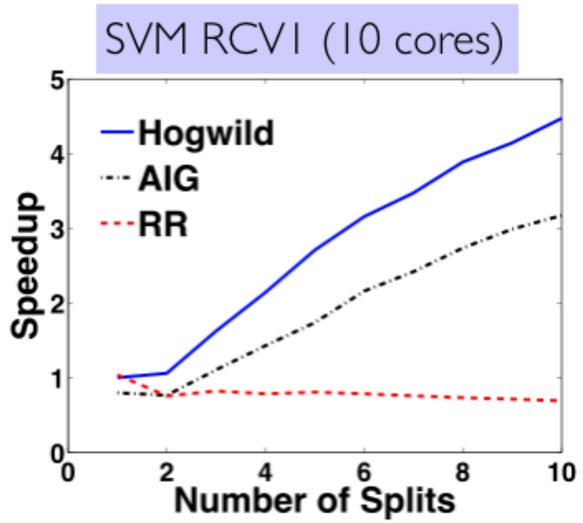
sample function f_i

$x =$ read shared memory

$g = -\gamma \cdot \nabla f_i(x)$

for v in the support of f **do**

$x_v \leftarrow x_v + g_v$



Adoption in industry: Google's Downpour SGD, Microsoft's Project Adam, etc

Niu et al, 2011; Image: Dimitris Papailiopoulos

QUESTION

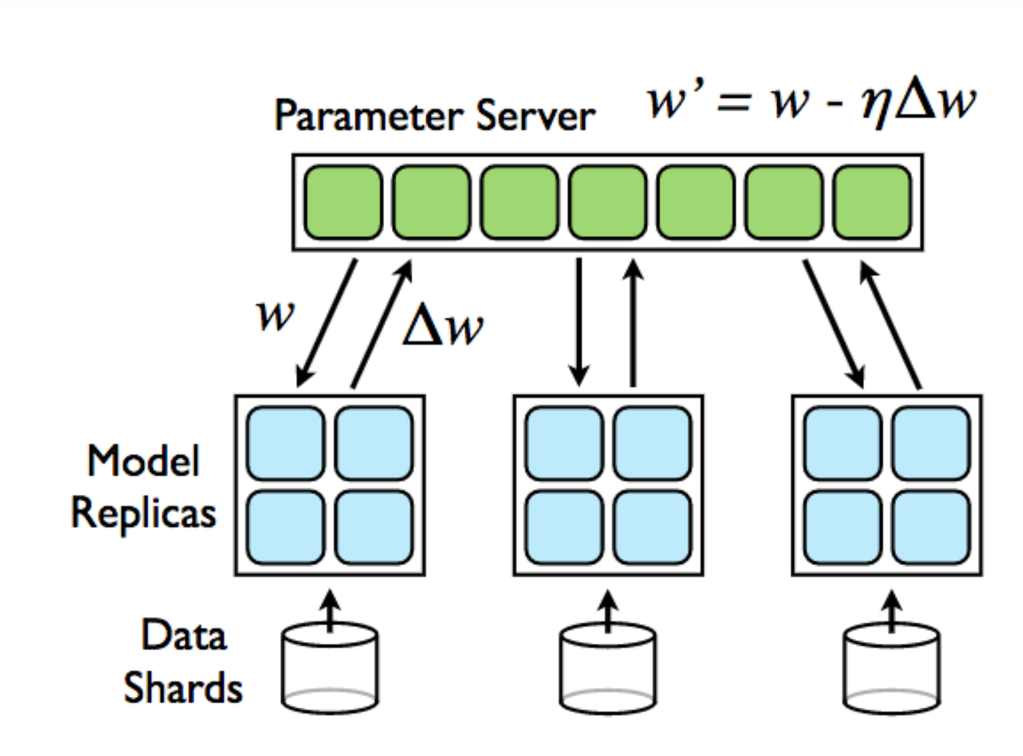
How to parallelize mini-batch SGD?

Parallelization scheme can vary based on:

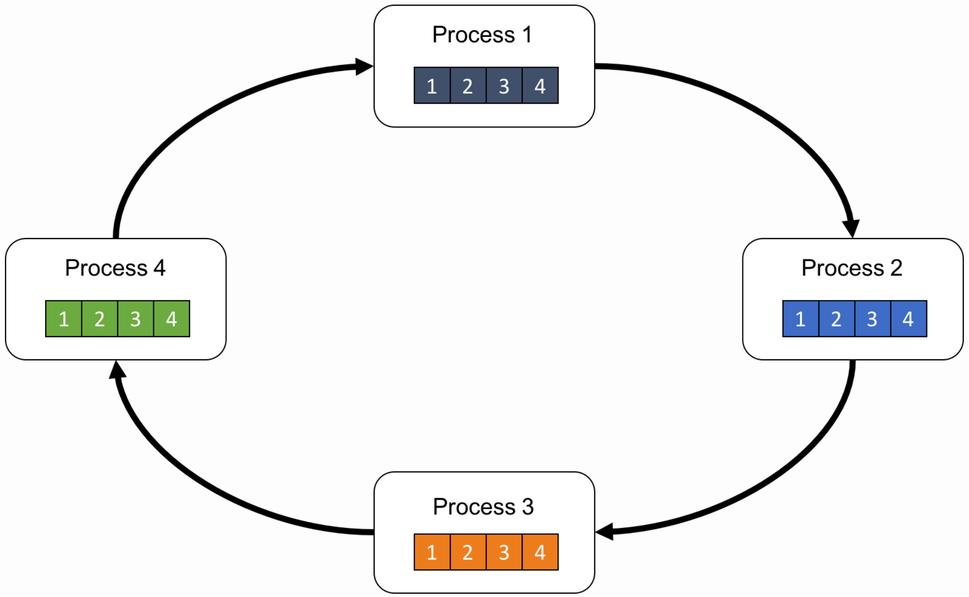
- synchronization model (*when do we send updates?*)
- communication strategy (*where do we send them?*)
- compression (*what updates do we send?*)

Communication strategies

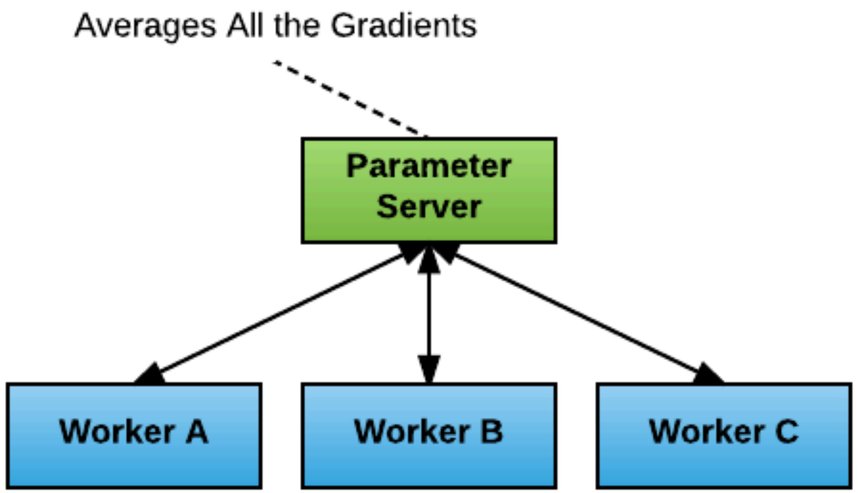
centralized: parameter server

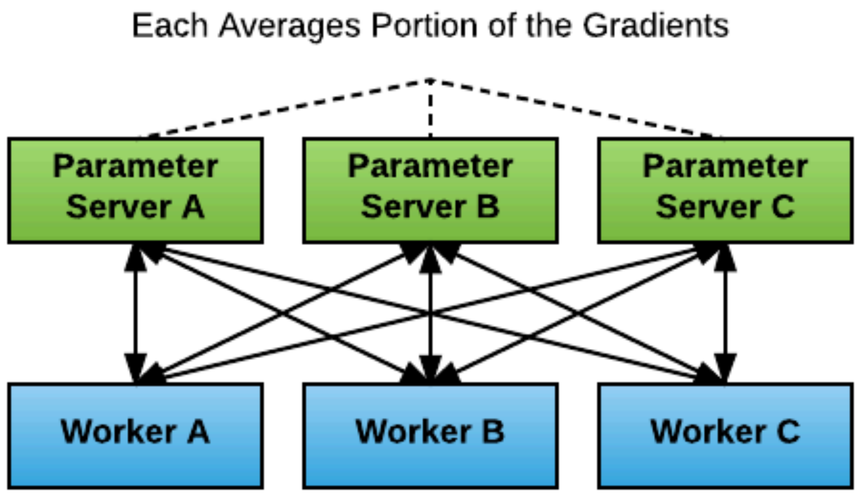


decentralized: ring all-reduce



Parameter server

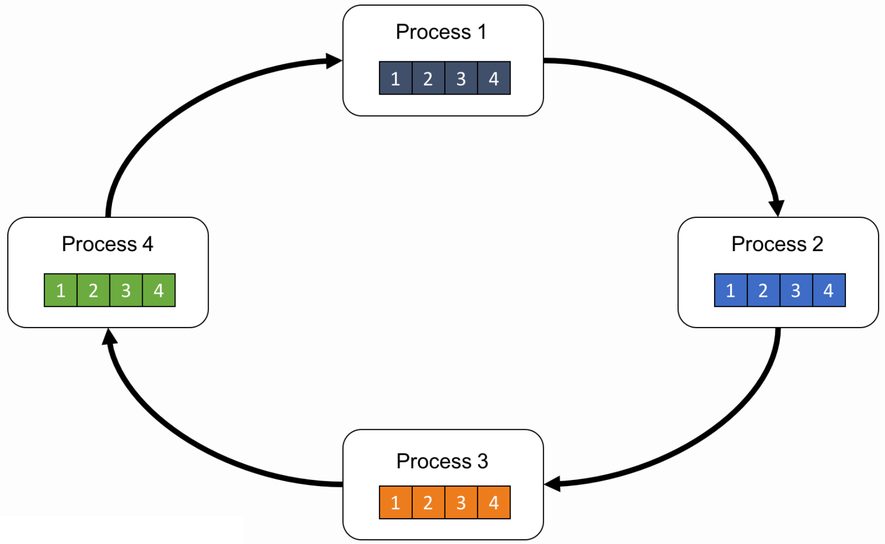




- Each worker communicates updates to parameter server, which aggregates updates and sends model back
- Can have multiple servers that partition the parameters
- Pro: works well with asynchronous updates, sparse updates, faulty nodes
- Con: communication can be a bottleneck

Ring all-reduce

- All-reduce: compute some reduction (usually a sum) of data on multiple machines $p_1, p_2, \ldots, p_m$ and materialize the result on all those machines

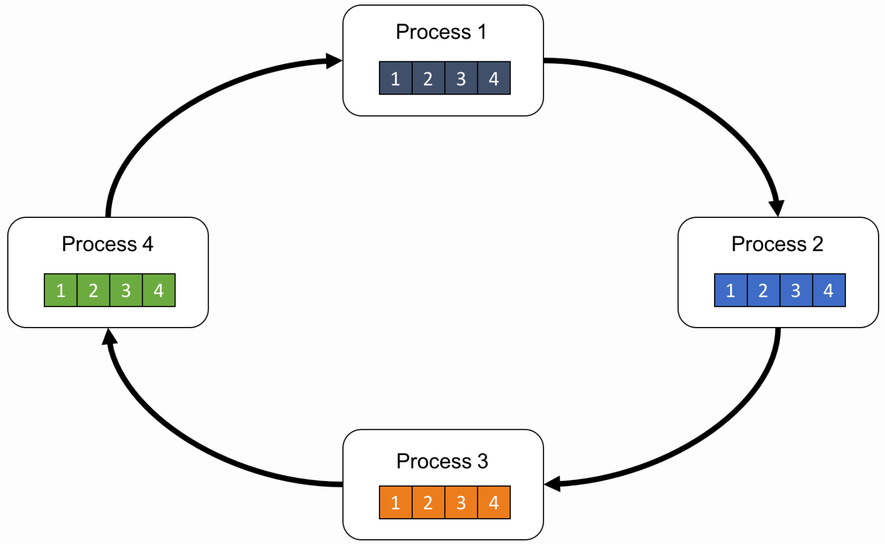



```
grad = gradient(net, w)

for epoch, data in enumerate(dataset):
  g = net.run(grad, in=data)
  gsum = comm.allreduce(g, op=sum)

  w -= lr * gsum / num_workers
```

Baidu, Horovod (Uber), 2017

Ring all-reduce

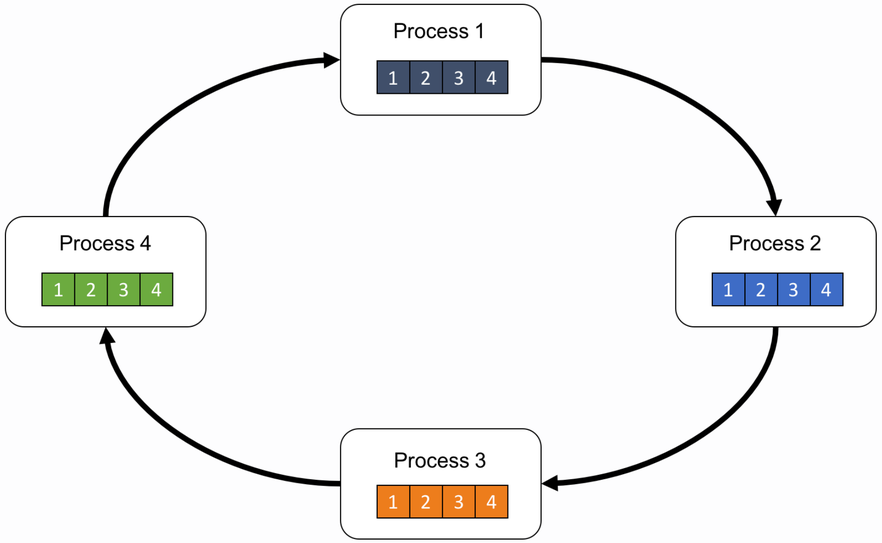
- All-reduce: compute some reduction (usually a sum) of data on multiple machines $p_1, p_2, \ldots, p_m$ and materialize the result on all those machines



- In ring all-reduce:
  - each process (worker/machine/node) passes its update to the next process, and combines its current update with what it receives
  - cycle continues until all processes have all updates
- Pro: communication-efficient—achieves optimal bandwidth usage
- Con: doesn't work as well for asynchronous or sparse updates, or faulty nodes

Baidu, Horovod (Uber), 2017

Naive all-reduce



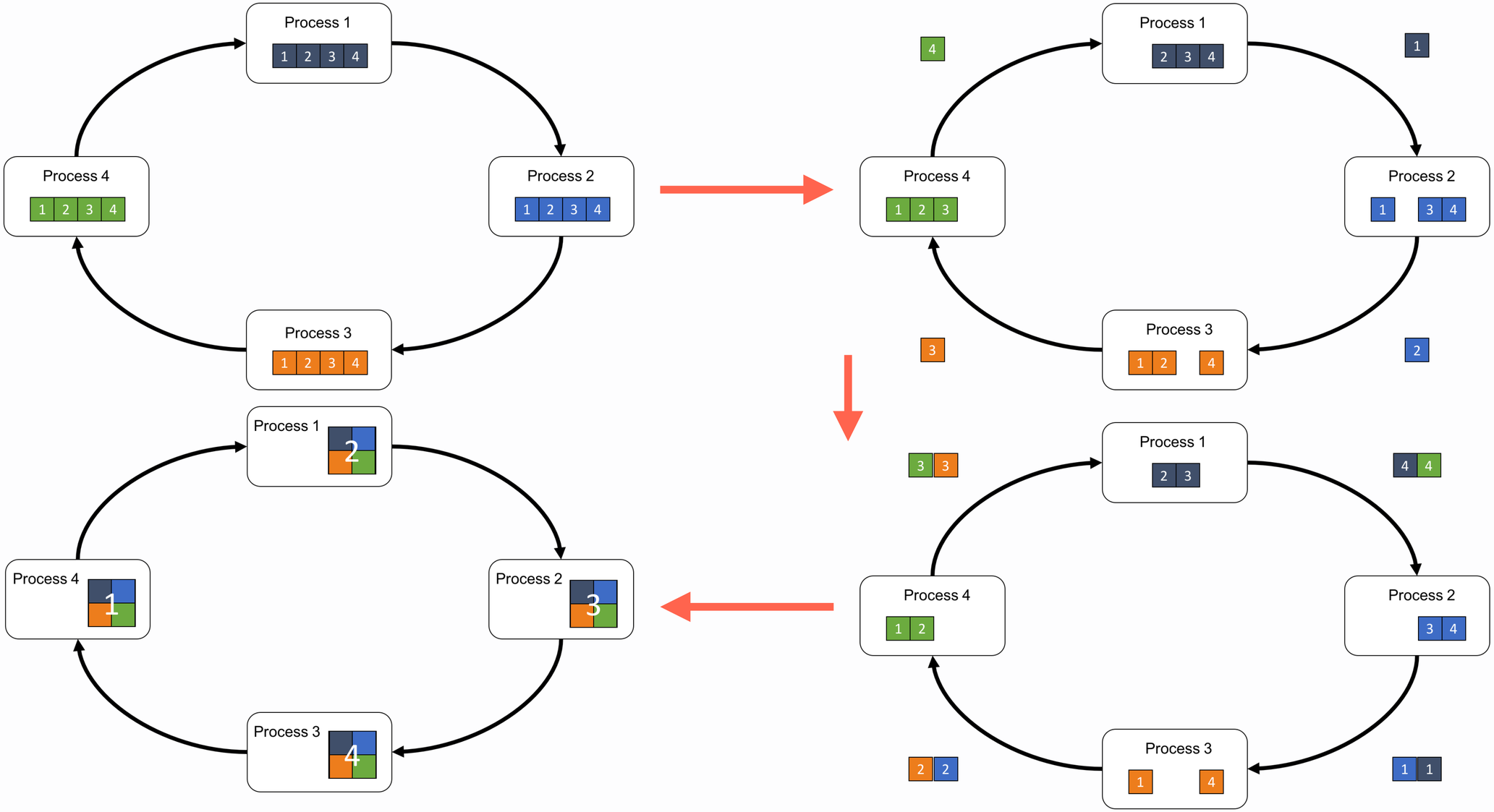
- Repeatedly share vector with neighbor

Q: how many rounds of communication given m nodes? ***m*-1**

Q: how much is communicated between nodes assuming length k vector? ***k***

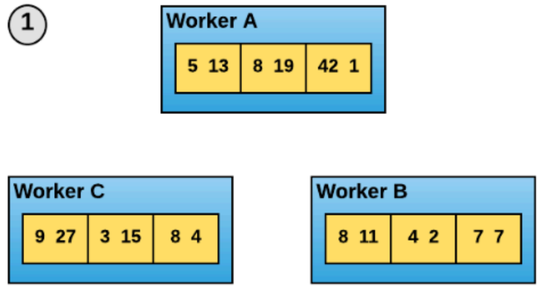
Q: total bandwidth? **O(*mk*)**

Ring all-reduce



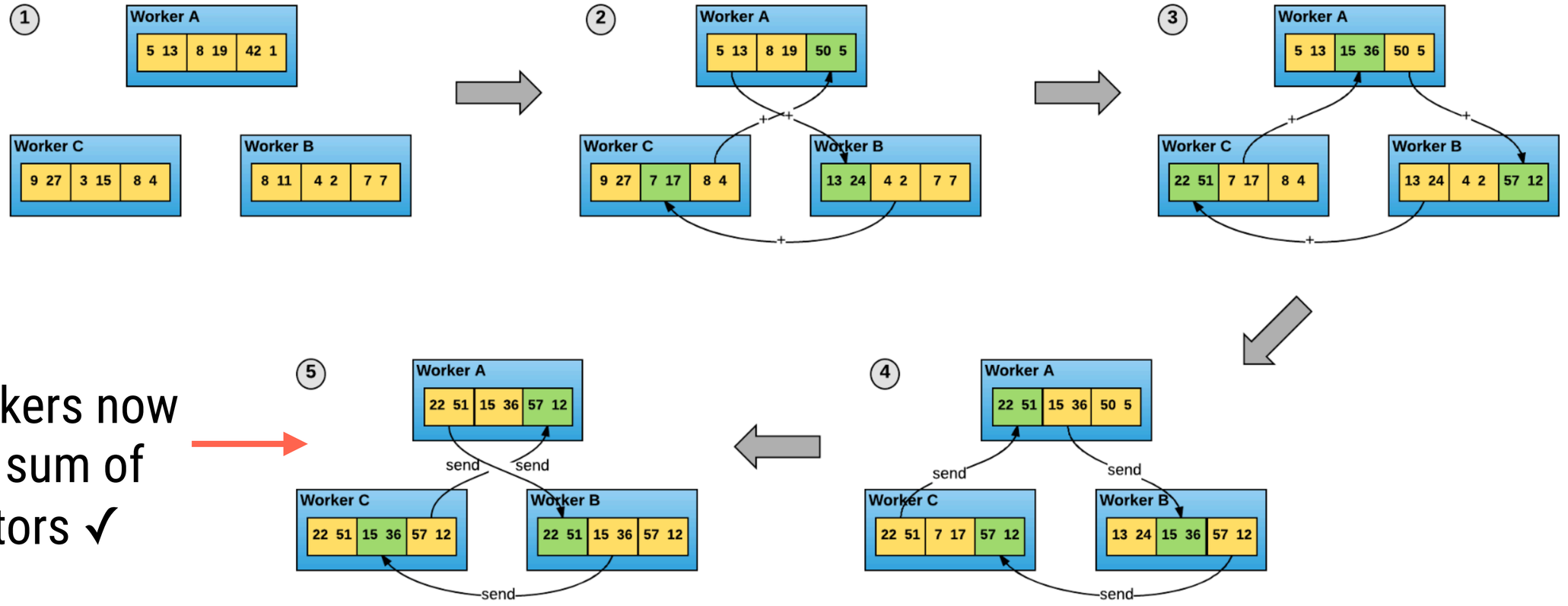
Ring all-reduce

goal: sum all *k*-length vectors (gradients) on *m* workers



Ring all-reduce

goal: sum all *k*-length vectors (gradients) on *m* workers



all workers now have sum of vectors ✓

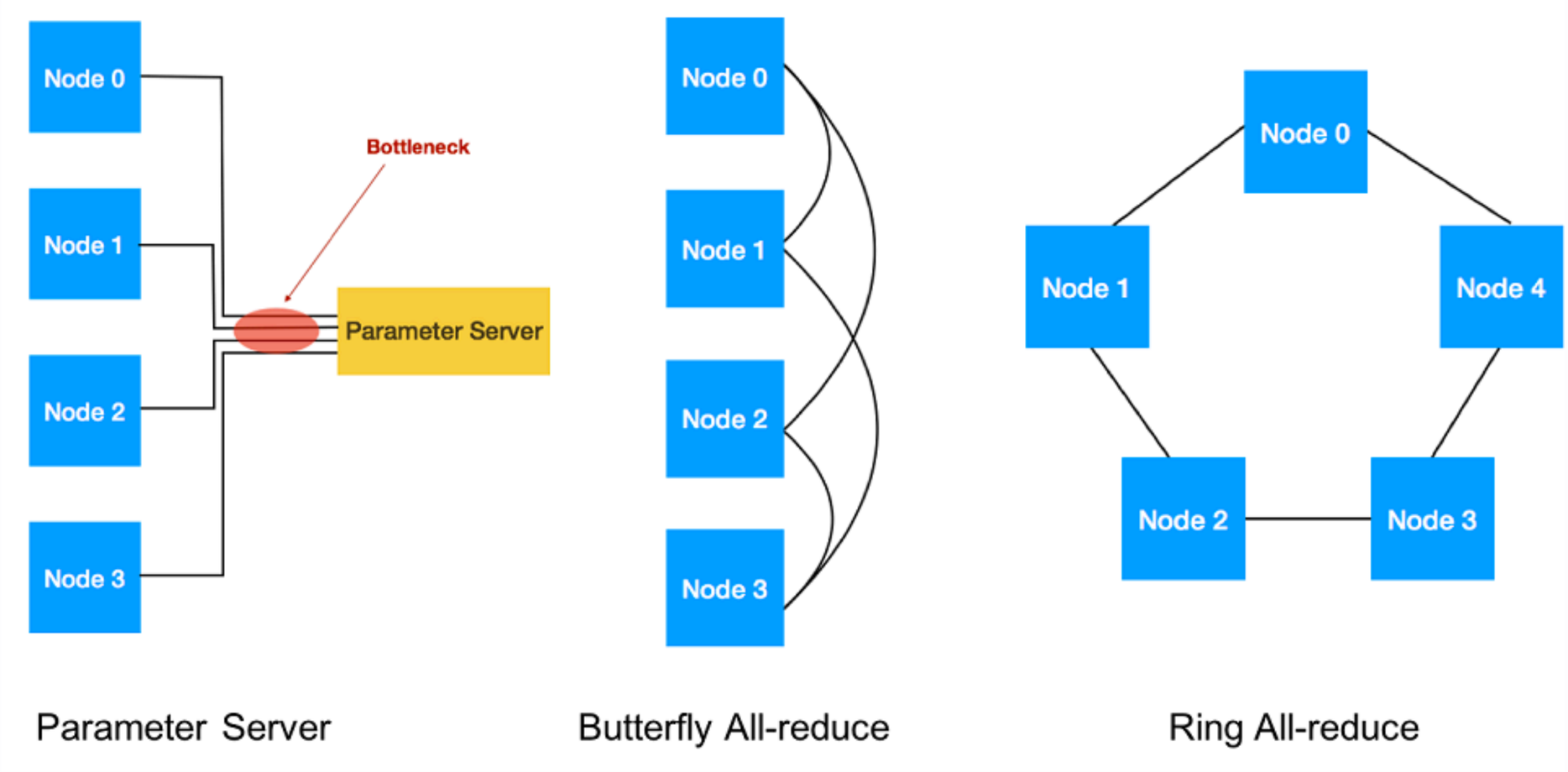
Q: how many rounds of communication? **2 (*m*-1)**

Q: how much is communicated between every node? ***k* / m**

Q: total bandwidth? **O (*k*)**

Data parallel communication strategies

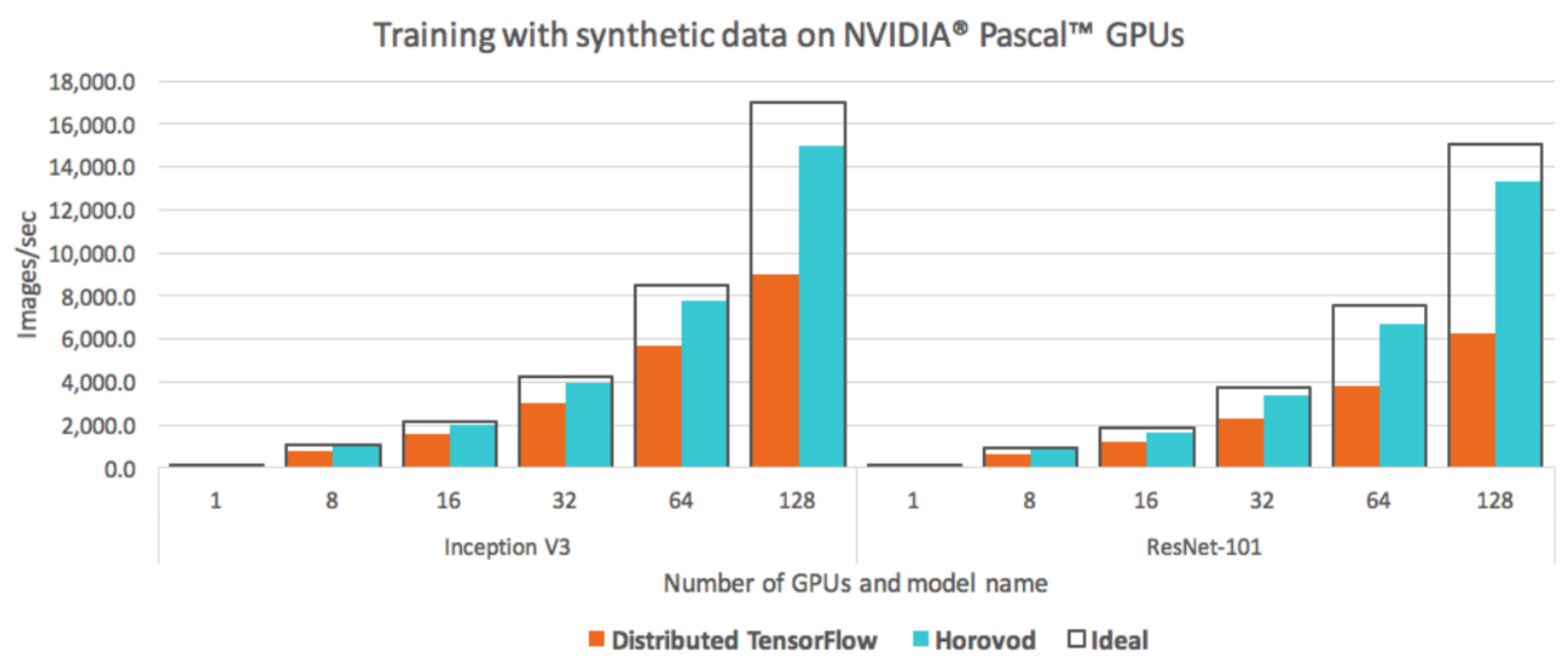
$T_{comm} = O(mk)$ $T_{comm} = O(log(mk))$ $T_{comm} = O(k)$



Parameter Server Butterfly All-reduce Ring All-reduce

Data parallel communication strategies

distributed tensorflow default (parameter server) vs. horovod (ring all-reduce)

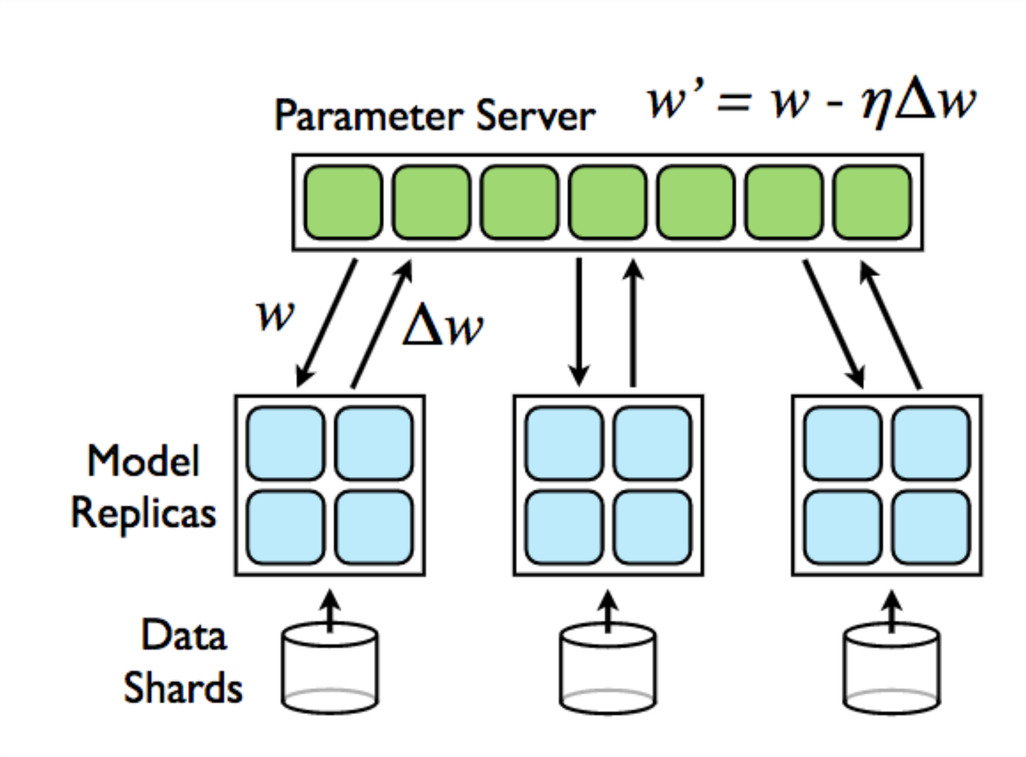


https://eng.uber.com/horovod/

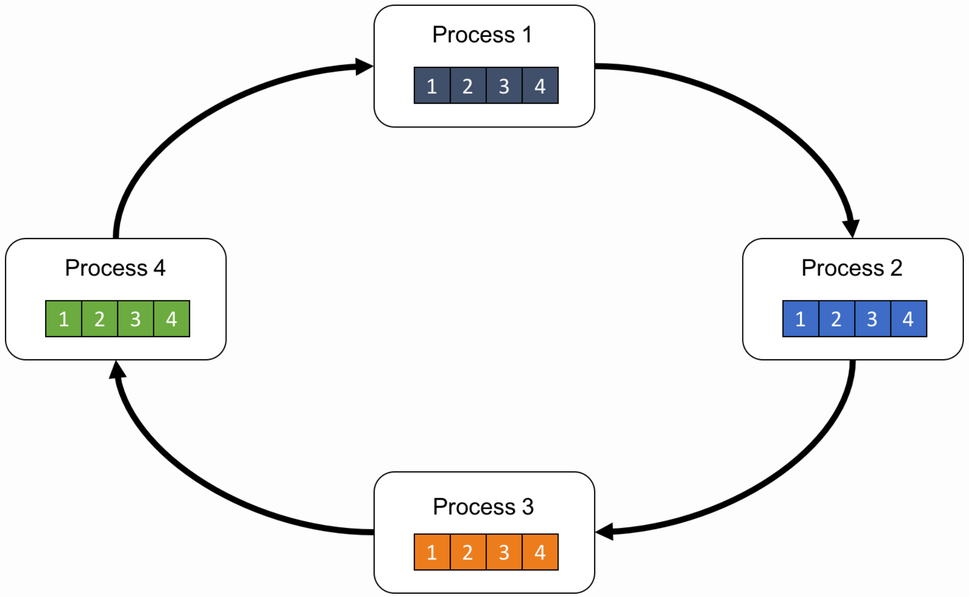
DATA PARALLEL DL:

Communication strategies

centralized: parameter server



decentralized: ring all-reduce



DATA PARALLEL DL:

Communication strategies

`tf.distribute.Strategy` intends to cover a number of use cases along different axes. Some of these combinations are currently supported and others will be added in the future. Some of these axes are:

- *Synchronous vs asynchronous training:* These are two common ways of distributing training with data parallelism. In sync training, all workers train over different slices of input data in sync, and aggregating gradients at each step. In async training, all workers are independently training over the input data and updating variables asynchronously.

Typically sync training is supported via all-reduce and async through parameter server architecture.

https://www.tensorflow.org/guide/distributed_training

QUESTION

How to parallelize mini-batch SGD?

Parallelization scheme can vary based on:

- synchronization model (*when do we send updates?*)
- communication strategy (*where do we send them?*)
- compression (*what updates do we send?*)

Gradient compression

Many strategies for compressing the updates (i.e., the gradients) in parallel SGD:

- Quantization
  - Lower the precision
  - Normal amount of communication per worker: *O(size of gradient) * 32*
  - Quantized communication per worker: *O(size of gradient) * C*, with *C << 32*
- Sparsification
  - Randomly subsample the gradient
  - Threshold small gradient entries

ATOMO

"*ATOMO: Communication-efficient Learning via Atomic Sparsification*,"
Wang, Sievert, Charles, Liu, Wright, Papailiopoulos, NeurIPS 2018

Key idea (of this and other related works):
- Quantization and sparsification can significantly reduce communication
- BUT can also hurt the convergence of SGD

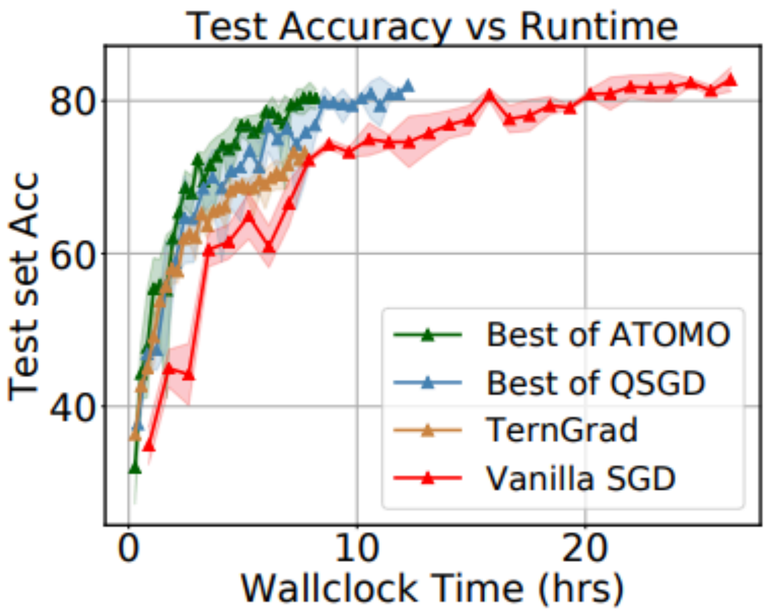
Try to remedy this by:
- Reducing the communication (number of bits communicated), while:
  - Ensuring that stochastic gradient is an unbiased estimator of the true gradient
  - Minimizing the variance of the stochastic gradient

ATOMO

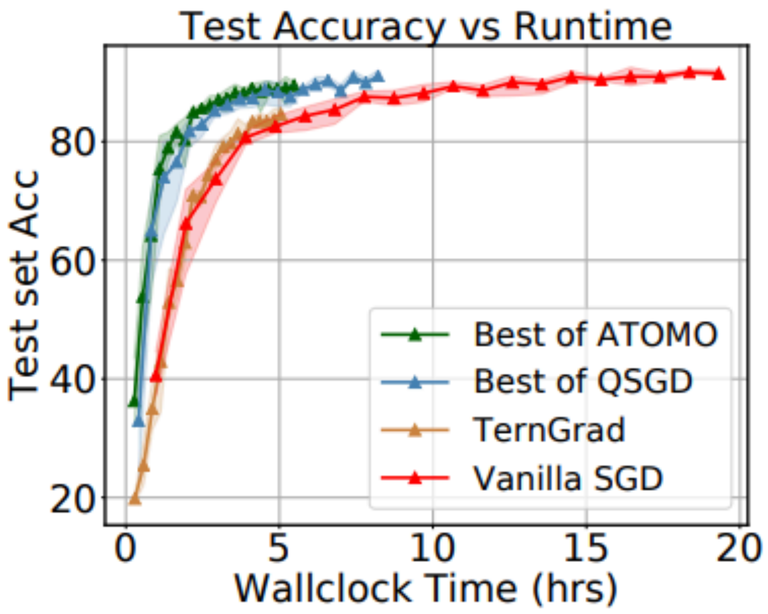
Can formulate this compression scheme via the following optimization problem:

$$\min_{g} \mathbb{E}\|\widehat{g}(w)\|^2$$
$$\text{s.t. } \mathbb{E}[\widehat{g}(w)] = g(w)$$
$$\widehat{g}(w) \text{ can be expressed with } k \text{ bits}$$

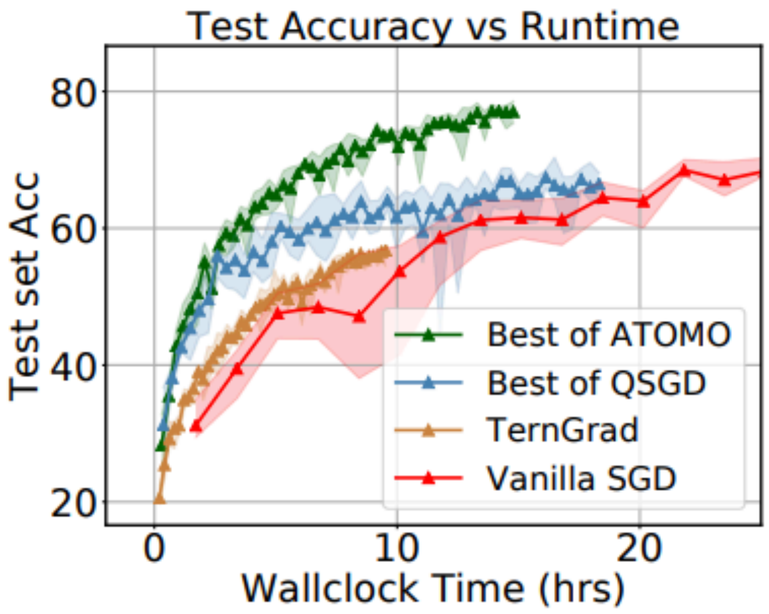
ATOMO



(a) CIFAR-10, ResNet-18, Best of QSGD and SVD



(b) SVHN, ResNet-18, Best of QSGD and SVD



(c) CIFAR-10, VGG11, Best of QSGD and SVD

2x reduction in training time

Gradient compression

Possible benefits:

- Less communication

Potential concerns:

- May slow convergence (in terms of iterations) or converge to a worse solution
- May not be suitable for all communication strategies (e.g., ring all-reduce)
- Sparsification rule may require extra computation and thus reduce benefits

QUESTION

How to parallelize mini-batch SGD?

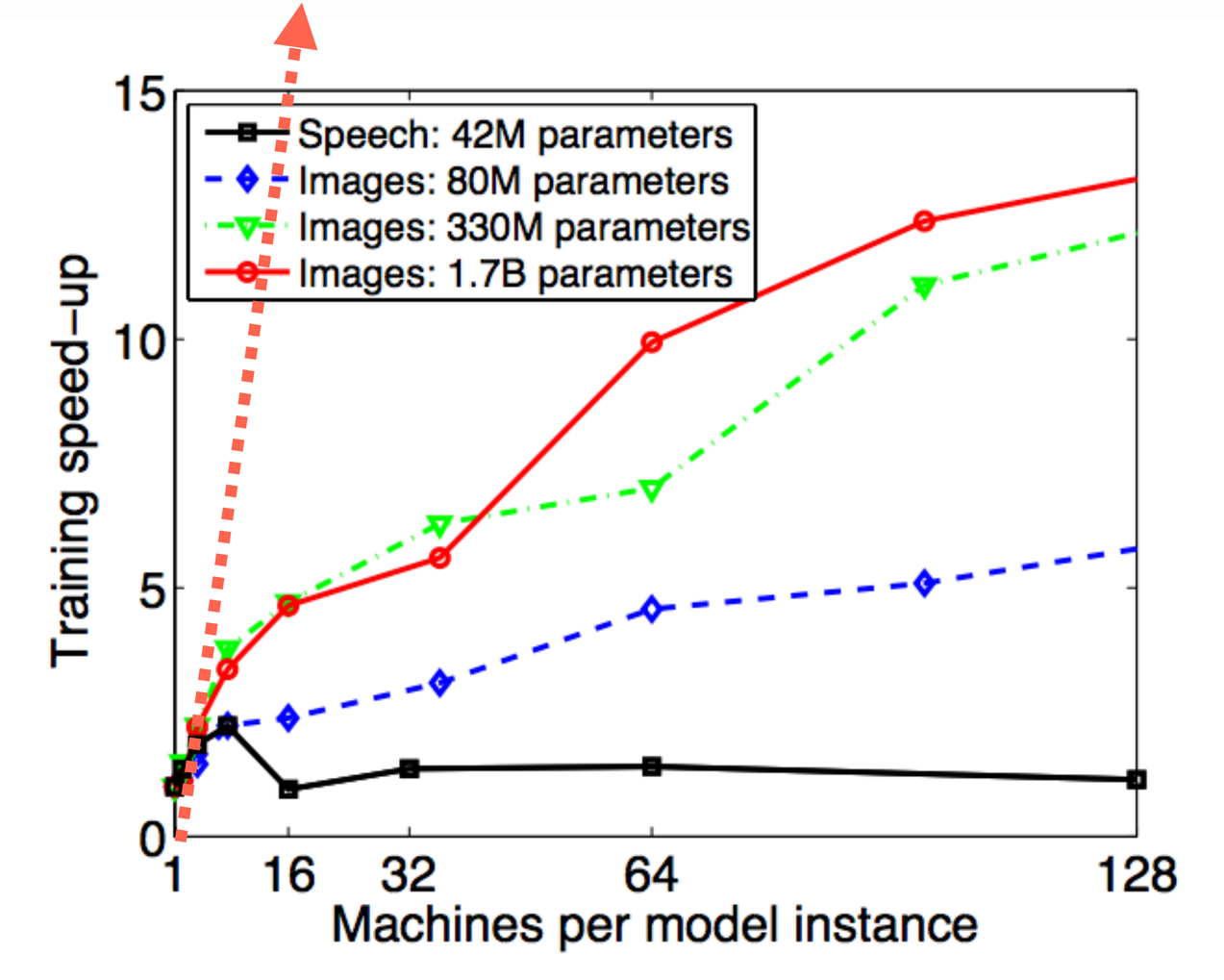
Parallelization scheme can vary based on:

- synchronization model (*when do we send updates?*)
  - synchronous vs. asynchronous
- communication strategy (*where do we send them?*)
  - parameter server vs. ring all-reduce
- compression (*what updates do we send?*)
  - quantization, sparsification

Outline

1. Scaling up Mb-SGD: Communication schemes
2. Challenges with mini-batching

One major issue with mini-batch SGD



ideally, speedups should scale linearly (i.e., you increase the # of machines by X, and you can converge X times faster)

some issues preventing speedups:

- communication between machines
- diminishing returns with larger batches

Dean et al., 2012

One major issue with mini-batch SGD

Systems perspective: large batches are good
- more computation, less communication

Statistical perspective: large batches are bad
- why??

Two problems with large batches:
- diminishing returns (after a certain point, additional gradients don’t help much)
- poor generalization (for non-convex optimization)

LOTS OF [ON-GOING] WORK IN THIS AREA …

One major issue with mini-batch SGD

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Elad Hoffer, **Itay Hubara**, **Daniel Soudry**
Technion - Israel Institute of Technology, Haifa, Israel
{elad.hoffer, itayhubara, daniel.soudry}@gmail.com

An Empirical Study of Large-Batch Stochastic Gradient Descent with Structured Covariance Noise

Yeming Wen[*,1,2], **Kevin Luk**[*,3], **Maxime Gazeau**[*,3], **Guodong Zhang**[1,2], **Harris Chan**[1,2], **Jimmy Ba**[1,2]
[1] University of Toronto, [2] Vector Institute, [3] Borealis AI

Gradient Diversity: a Key Ingredient for Scalable Distributed Learning

Dong Yin
UC Berkeley

Ashwin Pananjady
UC Berkeley

Max Lam
Stanford University

Dimitris Papailiopoulos
UW-Madison

Kannan Ramchandran
UC Berkeley

Peter L. Bartlett
UC Berkeley

Don't Use Large Mini-Batches, Use Local SGD

Tao Lin
EPFL, Switzerland
tao.lin@epfl.ch

Sebastian U. Stich
EPFL, Switzerland
sebastian.stich@epfl.ch

Kumar Kshitij Patel
IIT Kanpur, India
kumarkshitijpatel@gmail.com

Martin Jaggi
EPFL, Switzerland
martin.jaggi@epfl.ch

LOTS OF [ON-GOING] WORK IN THIS AREA …

One major issue with mini-batch SGD

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

Nitish Shirish Keskar*
Northwestern University
Evanston, IL 60208
keskar.nitish@u.northwestern.edu

Dheevatsa Mudigere
Intel Corporation
Bangalore, India
dheevatsa.mudigere@intel.com

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Elad Hoffer* **Itay Hubara*** **Daniel Soudry**
Technion - Israel Institute of Technology, Haifa, Israel
{elad.hoffer, itayhubara, daniel.soudry}@gmail.com

An Empirical Study of Large-Batch Stochastic Gradient Descent with Structured Covariance Noise

Yeming Wen*,1,2, Kevin Luk*,3, Maxime Gazeau*,3, Guodong Zhang[1,2], Harris Chan[1,2], Jimmy Ba[1,2]
[1] University of Toronto, [2] Vector Institute, [3] Borealis AI

Gradient Diversity: a Key Ingredient for Scalable Distributed Learning

Dong Yin UC Berkeley
Ashwin Pananjady UC Berkeley
Max Lam Stanford University
Dimitris Papailiopoulos UW-Madison
Kannan Ramchandran UC Berkeley
Peter L. Bartlett UC Berkeley

Don't Use Large Mini-Batches, Use Local SGD

Tao Lin
EPFL, Switzerland
tao.lin@epfl.ch

Sebastian U. Stich
EPFL, Switzerland
sebastian.stich@epfl.ch

Kumar Kshitij Patel
IIT Kanpur, India
kumarkshitijpatel@gmail.com

Martin Jaggi
EPFL, Switzerland
martin.jaggi@epfl.ch

Diminishing returns

larger batches = better potential for scaling

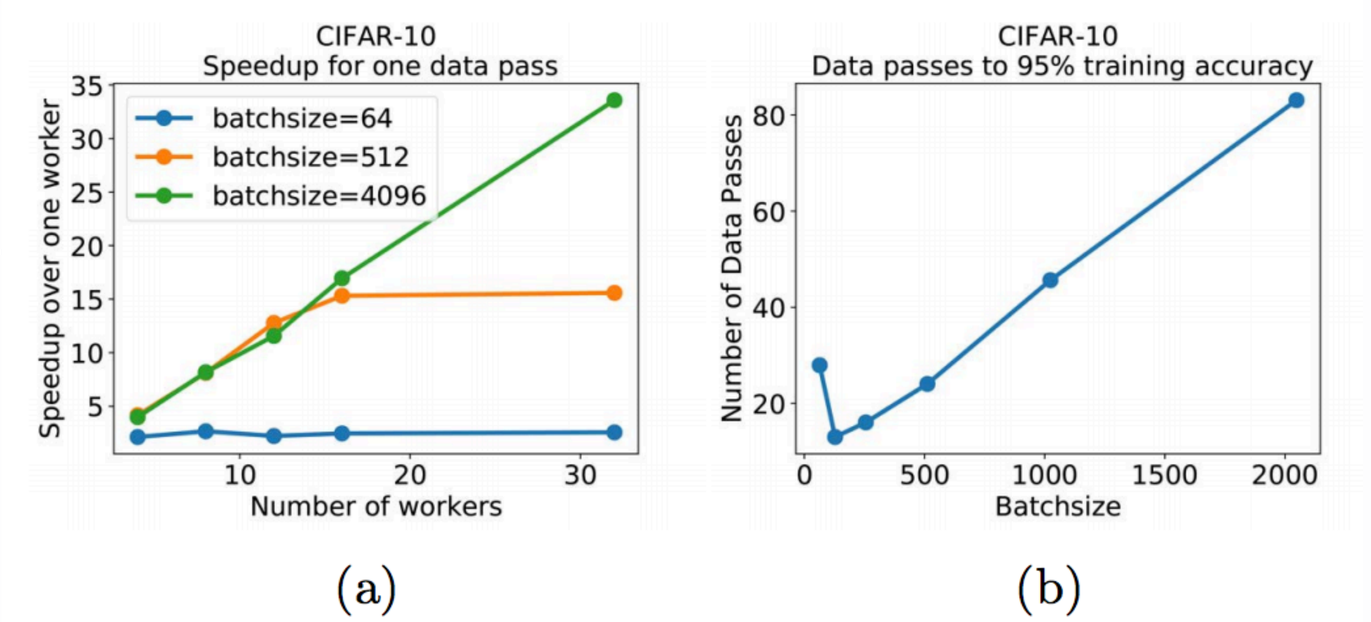


Figure 1: (a) Speedup gains for a single data pass and various batch-sizes, for a cuda-convnet model on CIFAR-10. (b) Number of data passes to reach 95% accuracy for a cuda-convnet model on CIFAR-10, vs batch-size. Step-sizes are tuned to maximize convergence speed.

BUT when you also consider training accuracy, speedups may saturate

Yin et al., AISTATS 2018

Diminishing returns

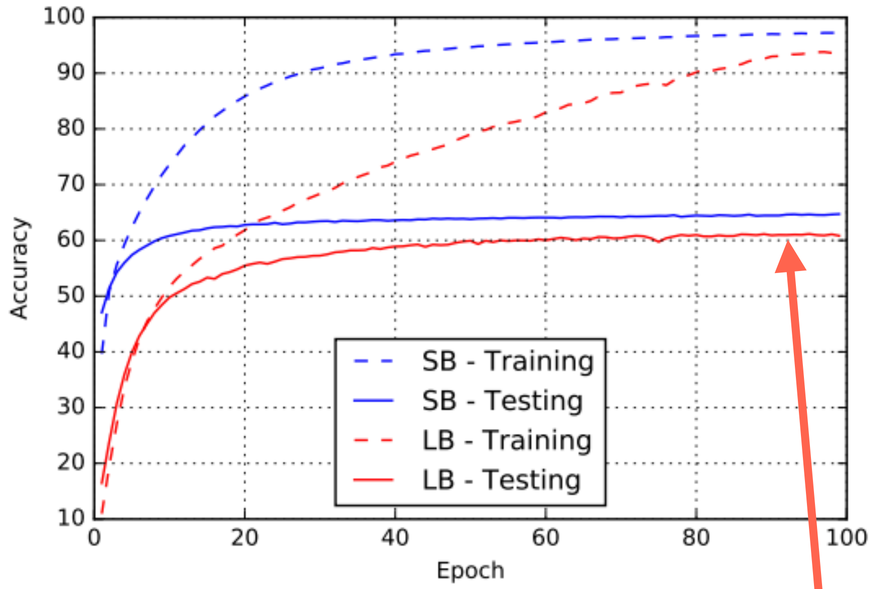
a potential explanation: gradient diversity, i.e.,
the degree to which gradients differ from one another

Claim: large batches perform worse when they lack diversity

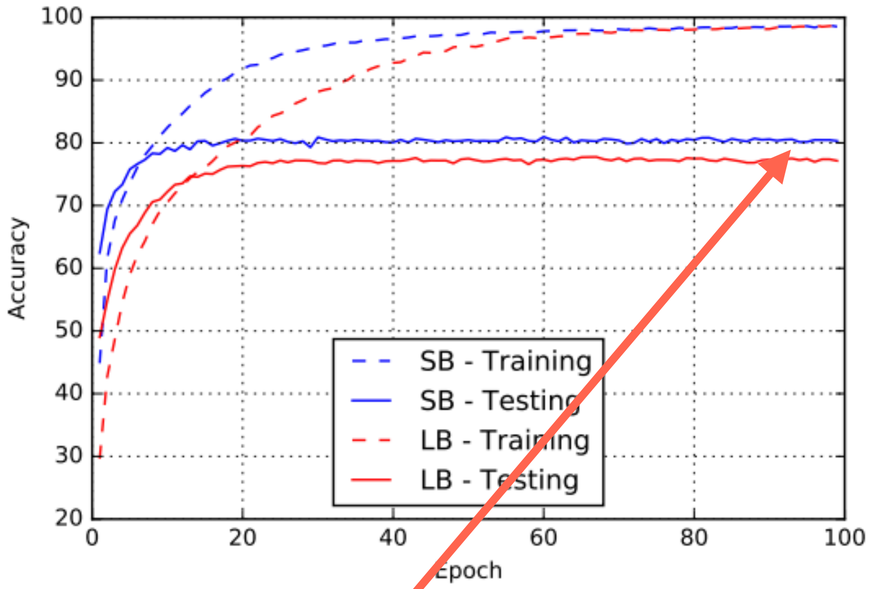
Yin et al., AISTATS 2018

EVEN WORSE

Poor generalization



(a) Network F_2



(b) Network C_1

Figure 2: Training and testing accuracy for SB and LB methods as a function of epochs.

generalization gap: large batch methods converge to lower test accuracy

Keskar, et al, ICLR 2017

EVEN WORSE

Poor generalization

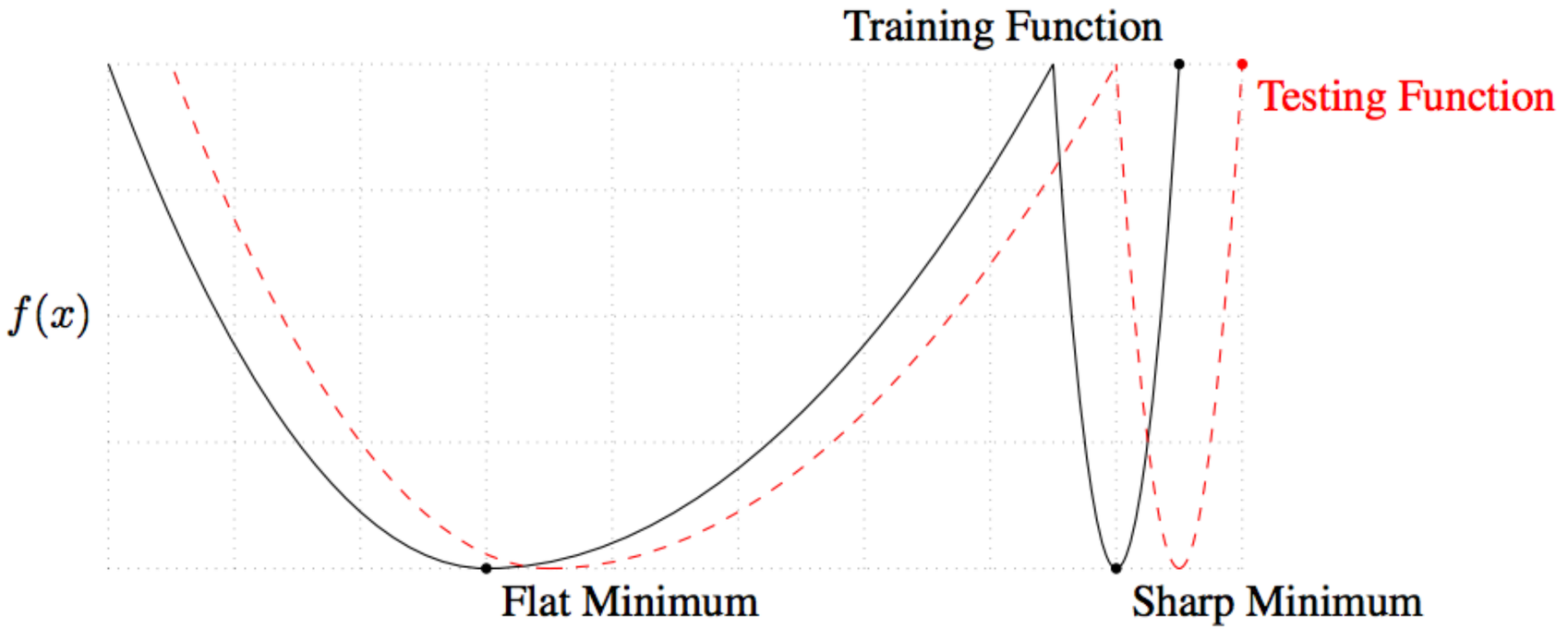


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

one hypothesis: using larger batches causes convergence to sharp minimum ?

Keskar, et al, ICLR 2017

EVEN WORSE

Poor generalization

potential solution: inject noise to help with regularization/generalization

Dataset	Model	SB	LB	LB+Fisher Trace	LB+Diag
MNIST	MLP	98.10	97.95	98.08	**98.10**
MNIST	LeNet	99.10	98.88	99.02	**99.11**
FASHION	LeNet	91.15	88.89	90.29	**90.79**
CIFAR-10	Alexnet	87.80	86.42	N/A	**87.61**
CIFAR-100	Alexnet	59.21	56.79	N/A	**59.10**
CIFAR-10	VGG16	93.25	91.81	92.91	**93.19**
CIFAR-100	VGG16	72.83	69.45	71.35	**72.11**
CIFAR-10	ResNet44	93.42	91.93	92.33	**92.88**
CIFAR-100	ResNet44x2	75.55	73.13	73.77	**74.26**

better, but still can’t completely close the gap …

Wen et al, AISTATS 2020

Issue with large batch sizes

Still an open problem!

Some ways to overcome issues with large batch sizes:

- use smaller batches and try to reduce communication bottlenecks
  - e.g., use asynchronous communication
- try to improve large batch training [*work still ongoing*]
  - e.g., gradient diversity, inject noise
- use a different distributed optimization method [*work still ongoing*]
  - will discuss additional methods in context of federated learning

DAWNBench / MLCommons

DAWNBench

An End-to-End Deep Learning Benchmark and Competition

Deep Learning benchmark introduced in 2018

Measures on end-to-end performance to achieve state-of-the-art accuracy on:

- Image classification (ImageNet, CIFAR-10)
- Question answering (SQuAD)

‘Performance’ is measured in training time / cost and inference latency / cost

Since introduced:

- Training time on ImageNet dropped from ~10 days to ~3 minutes [~5,000x speedup]
- Inference latency on ImageNet dropped from ~22 ms down to 0.07 [~300x speedup]

See retrospective here: https://arxiv.org/pdf/1806.01427.pdf

MLCommons (https://mlcommons.org/) has now replaced / expanded on DAWNBench

Additional Resources

- Distributed DL at Google [Large Scale Distributed Deep Networks]: https://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf
- Parameter server: https://www.cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf
- Ring all-reduce: https://eng.uber.com/horovod/
- Asynchrony [Hogwild!]: https://people.eecs.berkeley.edu/~brecht/papers/hogwildTR.pdf