Distributed Deep Learning (part 1)

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu

What is the Problem Being Solved?

- Training models is time consuming
  - Convergence can be slow
  - Training is computationally intensive
- Not all models fit in single machine or GPU memory
- Less of a problem: big data
  - Problem for data preparation / management
  - Not a problem for training … why?

On Dataset Size and Learning

- Data is a a resource! (e.g., like processors and memory)
  - Is having lots of processors a problem?
- You don't have to use all the data!
  - Though using more data can often help
- More data *often** dominates models and algorithms

EXPERT OPINION

Contact Editor: **Brian Brannon,** bbrannon@computer.org

The Unreasonable Effectiveness of Data

Alon Halevy, Peter Norvig, and Fernando Pereira, *Google*

*More data also enables more sophisticated.

What are the Metrics of Success?

- **Marketing Team:** Maximize number of GPUs/CPUs used
  - A bad metric … why?
- **Machine Learning:** Minimize passes through the training data
  - Easy to measure, but not informative … why?
- **Systems:** minimize time to complete a pass through the training data
  - Easy to measure, but not informative … why?

Ideal Metric of Success

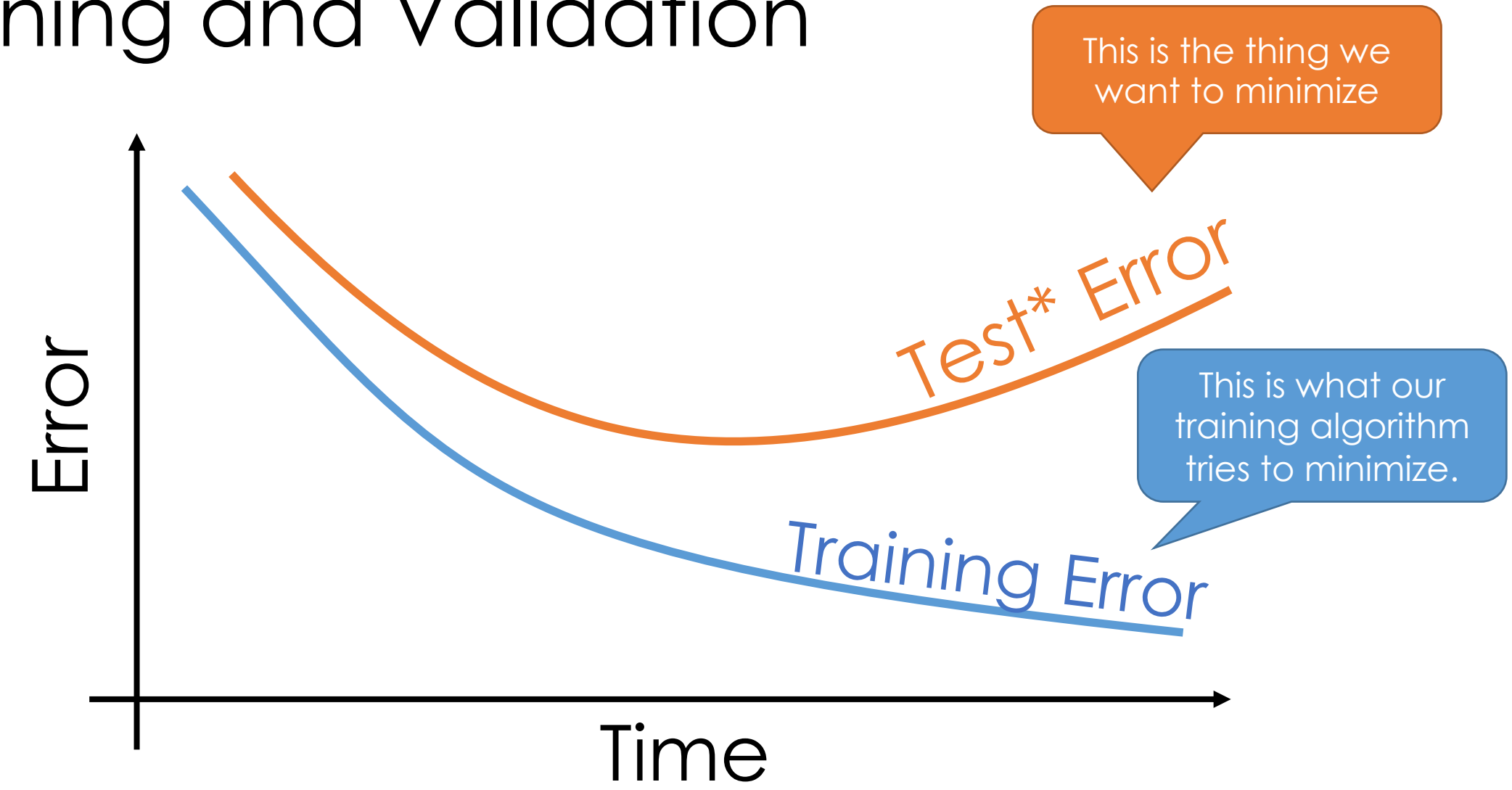
How do we measure Learning?

$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

Convergence
Machine Learning
Property

Throughput
System
Property

Training and Validation



*If you are making modeling decisions based on this then it should be called validation error.

Metrics of Success

- Minimize training time to "*best model*"
  - Best model measured in terms of test error
- Other Concerns?
  - **Complexity**: *Does the approach introduce additional training complexity (e.g., hyper-parameters)*
  - **Stability**: *How consistently does the system train the model?*

Two papers

NIPS 2012 (Same Year as AlexNet)

Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen,
Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato,
Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA

Abstract

Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with billions of parameters using tens of thousands of CPU cores. We have developed a software framework called *DistBelief* that can utilize computing clusters with thousands of machines to train large models. Within this framework, we have developed two algorithms for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas, and (ii) Sandblaster, a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS. Downpour SGD and Sandblaster L-BFGS both increase the scale and speed of deep network training. We have successfully used our system to train a deep network 30x larger than previously reported in the literature, and achieves state-of-the-art performance on ImageNet, a visual object recognition task with 16 million images and 21k categories. We show that these same techniques dramatically accelerate the training of a more modestly- sized deep network for a commercial speech recognition service. Although we focus on and report performance of these methods as applied to training large neural networks, the underlying algorithms are applicable to any gradient-based machine learning algorithm.

1 Introduction

Deep learning and unsupervised feature learning have shown great promise in many practical applications. State-of-the-art performance has been reported in several domains, ranging from speech recognition [1, 2], visual object recognition [3, 4], to text processing [5, 6].

It has also been observed that increasing the scale of deep learning, with respect to the number of training examples, the number of model parameters, or both, can drastically improve ultimate

2018 (Unpublished on Arxiv)

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

Abstract

Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD minibatch size. In this paper, we empirically show that on the ImageNet dataset large minibatches cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, we show no loss of accuracy when training with large minibatch sizes up to 8192 images. To achieve this result, we adopt a hyper-parameter-free linear scaling rule for adjusting learning rates as a function of minibatch size and develop a new warmup scheme that overcomes optimization challenges early in training. With these simple techniques, our Caffe2-based system trains ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour, while matching small minibatch accuracy. Using commodity hardware, our implementation achieves ~90% scaling efficiency when moving from 8 to 256 GPUs. Our findings enable training visual recognition models on internet-scale data with high efficiency.

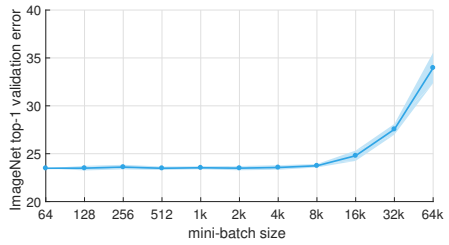


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

tation [8, 10, 28]. Moreover, this pattern generalizes: larger datasets and neural network architectures consistently yield

Large Scale Distributed Deep Networks

Described the system for the 2012 ICML Paper

NIPS 2012 (Same Year as AlexNet)

Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen,
Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato,
Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA

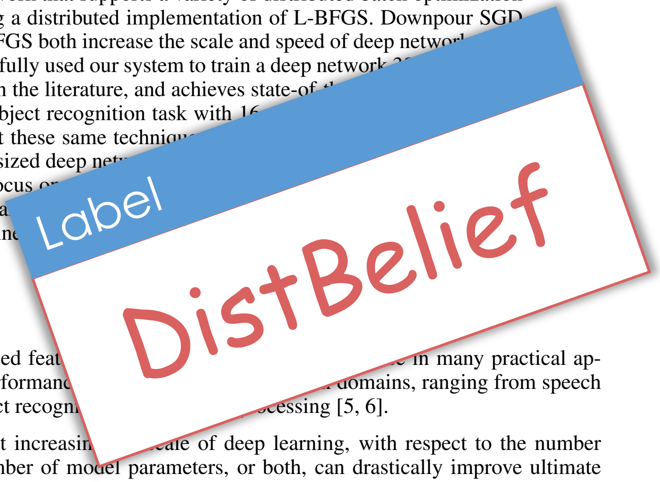
Abstract

Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with billions of parameters using tens of thousands of CPU cores. We have developed a software framework called *DistBelief* that can utilize computing clusters with thousands of machines to train large models. Within this framework, we have developed two algorithms for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas, and (ii) Sandblaster, a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS. Downpour SGD and Sandblaster L-BFGS both increase the scale and speed of deep networ
ing. We have successfully used our system to train a deep network
previously reported in the literature, and achieves state-of-
ImageNet, a visual object recognition task with 16
gories. We show that these same techniqu
of a more modestly- sized deep net
vice. Although we focus o
to training large neura
gradient-based machine

1 Introduction

Deep learning and unsupervised feat … in many practical applications. State-of-the-art performanc … domains, ranging from speech recognition [1, 2], visual object recogn … cessing [5, 6].

It has also been observed that increasin … ale of deep learning, with respect to the number of training examples, the number of model parameters, or both, can drastically improve ultimate classification accuracy [3, 4, 7]. These results have led to a surge of interest in scaling up the training and inference algorithms used for these models [8] and in improving applicable optimization procedures [7, 9]. The use of GPUs [1, 2, 3, 8] is a significant advance in recent years that makes



Building High-level Features
Using Large Scale Unsupervised Learning

Quoc V. Le
Marc'Aurelio Ranzato
Rajat Monga
Matthieu Devin
Kai Chen
Greg S. Corrado
Jeff Dean
Andrew Y. Ng

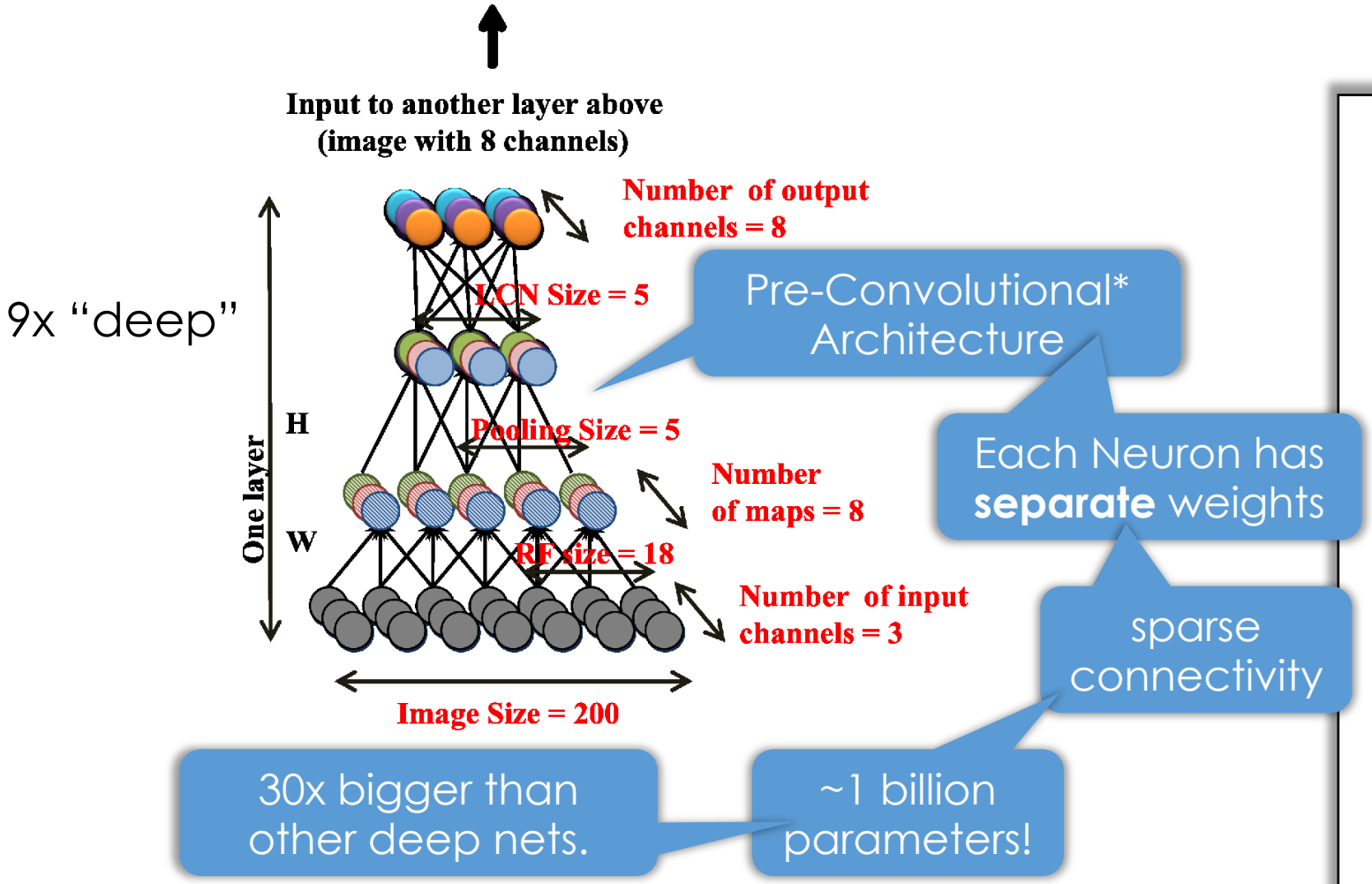
Abstrac

We consider the problem
level, class-specific featu
only unlabeled data. F
possible to learn a face d
unlabeled images using u
To answer this, we train a
connected sparse autoenc
and local contrast normal
dataset of images (the
lion connections, the dataset has 10 million

but current experimental evidence suggests the possibility that some neurons in the temporal cortex are

Building High-Level Features Using Large Scale Unsupervised Learning

ICML 2012



*This pre-dates AlexNet but is two decades after LeNet.

Building High-level Features
Using Large Scale Unsupervised Learning

Quoc V. Le QUOCLE@CS.STANFORD.EDU
Marc'Aurelio Ranzato RANZATO@GOOGLE.COM
Rajat Monga RAJATMONGA@GOOGLE.COM
Matthieu Devin MDEVIN@GOOGLE.COM
Kai Chen KAICHEN@GOOGLE.COM
Greg S. Corrado GCORRADO@GOOGLE.COM
Jeff Dean JEFF@GOOGLE.COM
Andrew Y. Ng ANG@CS.STANFORD.EDU

Abstract

We consider the problem of building high-level, class-specific feature detectors from only unlabeled data. For example, is it possible to learn a face detector using only unlabeled images using unlabeled images? To answer this, we train a 9-layered locally connected sparse autoencoder with pooling and local contrast normalization on a large dataset of images (the model has 1 billion connections, the dataset has 10 million 200x200 pixel images downloaded from the Internet). We train this network using model parallelism and asynchronous SGD on a cluster with 1,000 machines (16,000 cores) for three days. Contrary to what appears to be a widely-held intuition, our experimental results reveal that it is possible to train a face detector without having to label images as containing a face or not. Control experiments show that this feature detector is robust not only to translation but also to scaling and out-of-plane rotation. We also find that the same network is sensitive to other high-level concepts such as cat faces and human bodies. Starting with these learned features, we trained our network to obtain 15.8% accuracy in recognizing 20,000 object categories from ImageNet, a leap of 70% relative im-

1. Introduction

The focus of this work is to build *high-level*, class-specific feature detectors from *unlabeled* images. For instance, we would like to understand if it is possible to build a face detector from only unlabeled images. This approach is inspired by the neuroscientific conjecture that there exist highly class-specific neurons in the human brain, generally and informally known as "grandmother neurons." The extent of class-specificity of neurons in the brain is an area of active investigation, but current experimental evidence suggests the possibility that some neurons in the temporal cortex are highly selective for object categories such as faces or hands (Desimone et al., 1984), and perhaps even specific people (Quiroga et al., 2005).

Contemporary computer vision methodology typically emphasizes the role of *labeled* data to obtain these class-specific feature detectors. For example, to build a face detector, one needs a large collection of images labeled as containing faces, often with a bounding box around the face. The need for large labeled sets poses a significant challenge for problems where labeled data are rare. Although approaches that make use of inexpensive unlabeled data are often preferred, they have not been shown to work well for building high-level features.

This work investigates the feasibility of building high-level features from only *unlabeled* data. A positive

More Context (ML circa 2012)

- Focus of existing distributed ML research
  - Convex Optimization (e.g., SVMs, Lasso)
  - Matrix Factorization
  - Graphical Models
- Key systems at the time
  - Map Reduce → not great for iterative computation (why?)
    - Spark really wasn't visible to ML community
  - GraphLab → A truly wonderful system*
    - we worked with Quoc/Andrew to get their model running on GraphLab but wasn't as performant.

*Developed by the speaker.

Key Problems Addressed in DistBelief Paper

Main Problem

- **Speedup training** for **large models**

Sub Problems

- How to partition models and data
- Variance in worker performance → Stragglers
- Failures in workers → Fault-Tolerance

Crash Course on Stochastic Gradient Descent

The Gradient Descent Algorithm

$\theta^{(0)} \leftarrow$ initial model parameters (random, warm start)

For τ from 1 to convergence:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Learning Rate

Average Gradient of **Over Training Dataset**

Evaluated at

How do we distribute this computation?

$\theta^{(0)} \leftarrow$ initial model parameters (random, warm start)

For τ from 1 to convergence:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Learning Rate

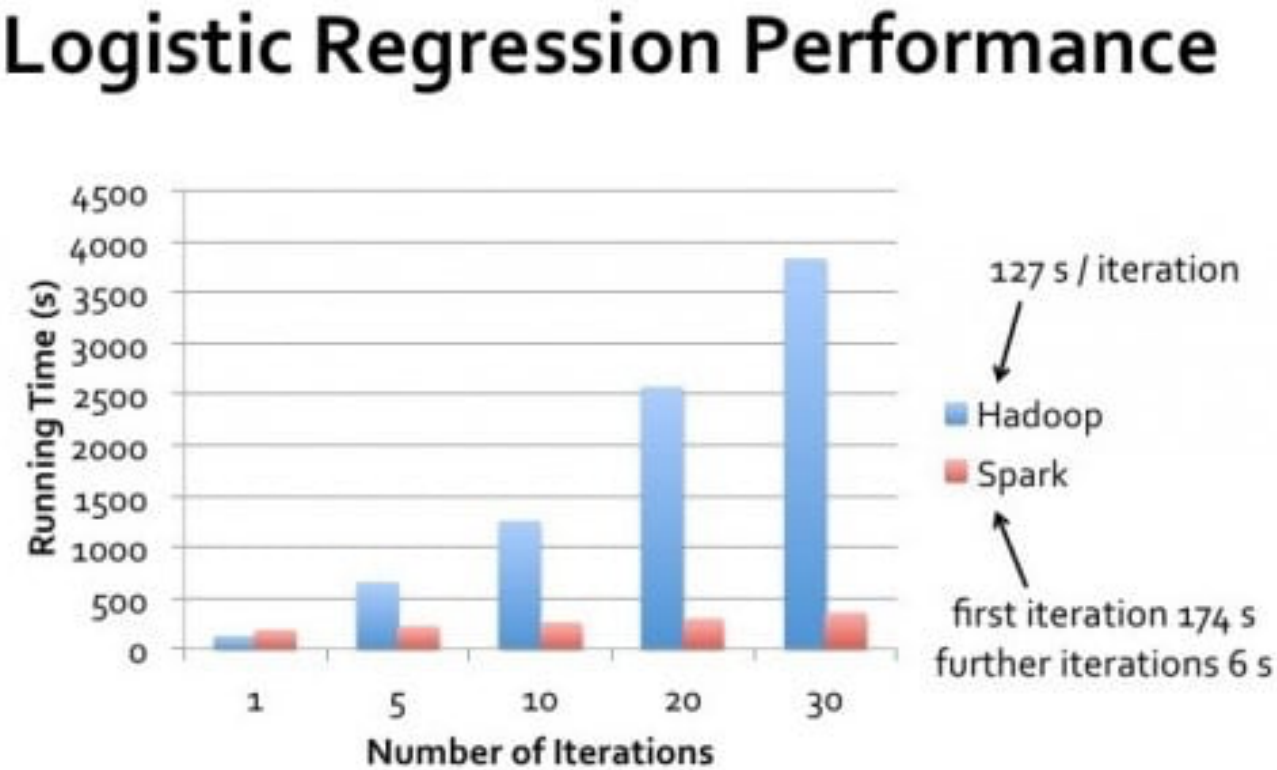
Average Gradient of **Over Training Dataset**

Evaluated at $\theta=\theta^{(t)}$

How do we distribute this computation?

Data parallelism: *divide data across machines, compute local gradient sums and then aggregate across machines. repeat.*

Issues? Repeatedly scanning the data… what if we cache it?



also substantially improved the programming API over Hadoop.

The Gradient Descent Algorithm

$\theta^{(0)} \leftarrow$ initial model parameters (random, warm start)

For t from 1 to convergence:

Can we use statistics to improve this algorithm?

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Evaluated at $\theta=\theta^{(t)}$

Learning Rate

Average Gradient of **Over Training Dataset**

The empirical gradient is an approximation of what I really want:

$$\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\nabla_\theta \mathbf{L}(y, f(x;\theta))\right]$$

Law of large numbers → more data provides a better approximation (variance in the estimator decreases linearly)

Do I really need to use all the data?

The empirical gradient is an approximation of what I really want:

$$\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\nabla_\theta \mathbf{L}(y, f(x;\theta))\right]$$

Law of large numbers → more data provides a better approximation (variance in the estimator decreases linearly)

$$\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

Random subset of the data

Small *B*: fast but less accurate
Large *B*: slower but more accurate

Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros …)

For t from 1 to convergence:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros …)

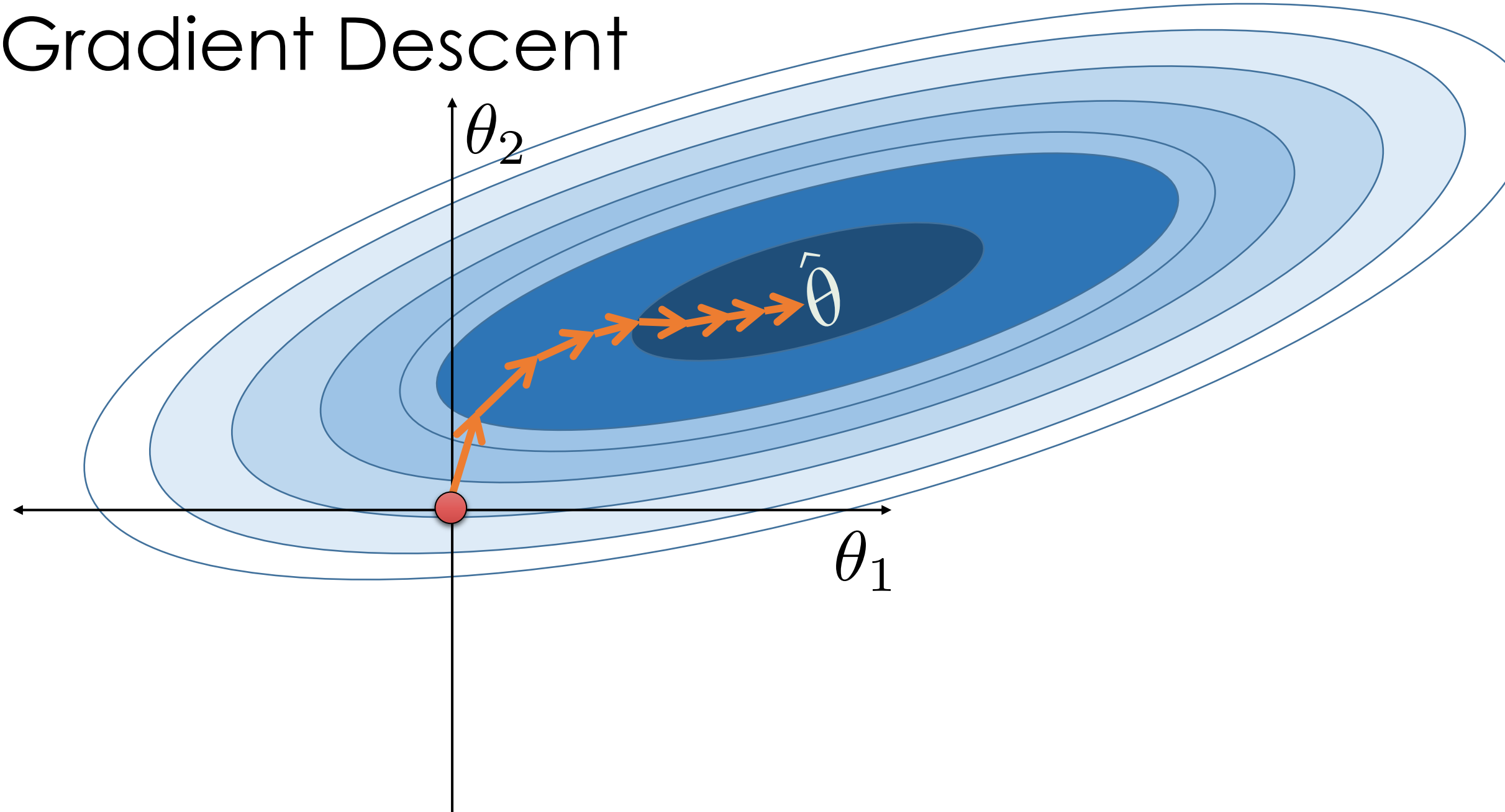
For t from 0 to convergence:

$$\mathcal{B} \sim \text{Random subset of indices}$$

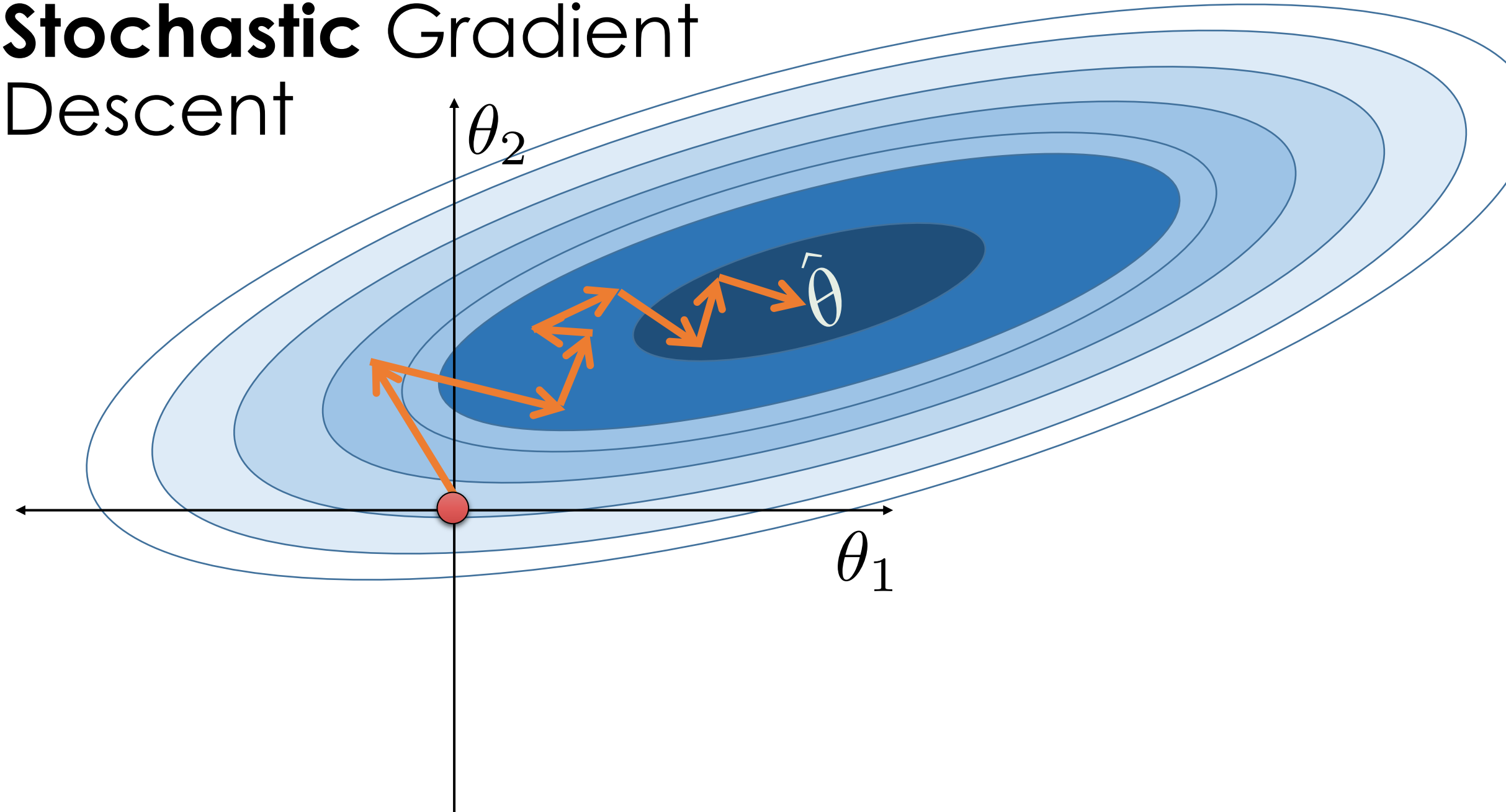
$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Assuming Decomposable Loss Functions

Gradient Descent



Stochastic Gradient Descent



How do you distribute SGD?

Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros …)

For t from 0 to convergence:

$\mathcal{B} \sim$ Random subset of indices

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Model Parallelism

Speed up Gradient.
Depends on Model

Data Parallelism

Speed up Sum.
Depends on size of B

(End digression ... for now)

Key Innovations in

Large Scale Distributed Deep Networks

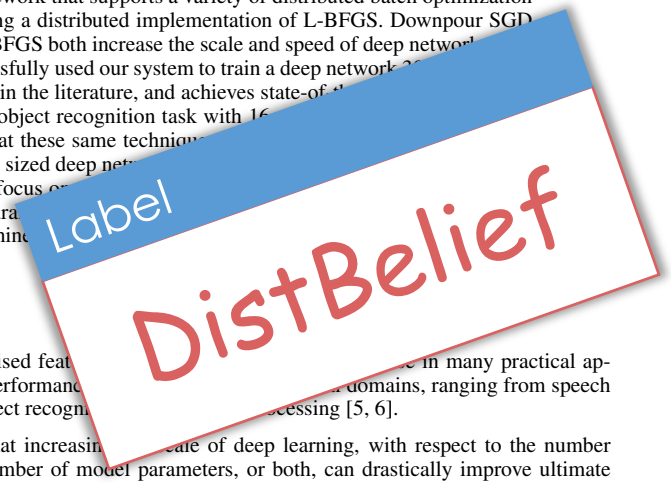
NIPS 2012 (Same Year as AlexNet)

Large Scale Distributed Deep Networks

**Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen,
Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato,
Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng**
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA

Abstract

Recent work in unsupervised feature learning and deep learning has shown that being able to train large models can dramatically improve performance. In this paper, we consider the problem of training a deep network with billions of parameters using tens of thousands of CPU cores. We have developed a software framework called *DistBelief* that can utilize computing clusters with thousands of machines to train large models. Within this framework, we have developed two algorithms for large-scale distributed training: (i) Downpour SGD, an asynchronous stochastic gradient descent procedure supporting a large number of model replicas, and (ii) Sandblaster, a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS. Downpour SGD and Sandblaster L-BFGS both increase the scale and speed of deep networ
ing. We have successfully used our system to train a deep network
previously reported in the literature, and achieves state-of-t
ImageNet, a visual object recognition task with 16
gories. We show that these same techniqu
of a more modestly- sized deep net
vice. Although we focus o
to training large neura
gradient-based machin



1 Introduction

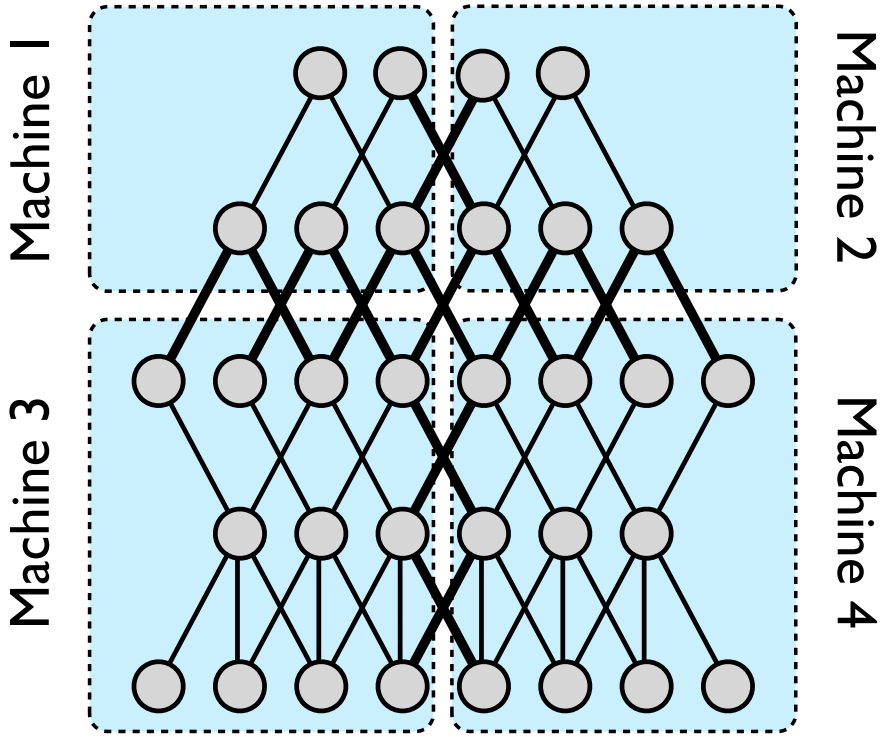
Deep learning and unsupervised feat
plications. State-of-the-art performanc
recognition [1, 2], visual object recogn

in many practical ap-
domains, ranging from speech
cessing [5, 6].

It has also been observed that increasin
of training examples, the number of model parameters, or both, can drastically improve ultimate classification accuracy [3, 4, 7]. These results have led to a surge of interest in scaling up the training and inference algorithms used for these models [8] and in improving applicable optimization procedures [7, 9]. The use of GPUs [1, 2, 3, 8] is a significant advance in recent years that makes

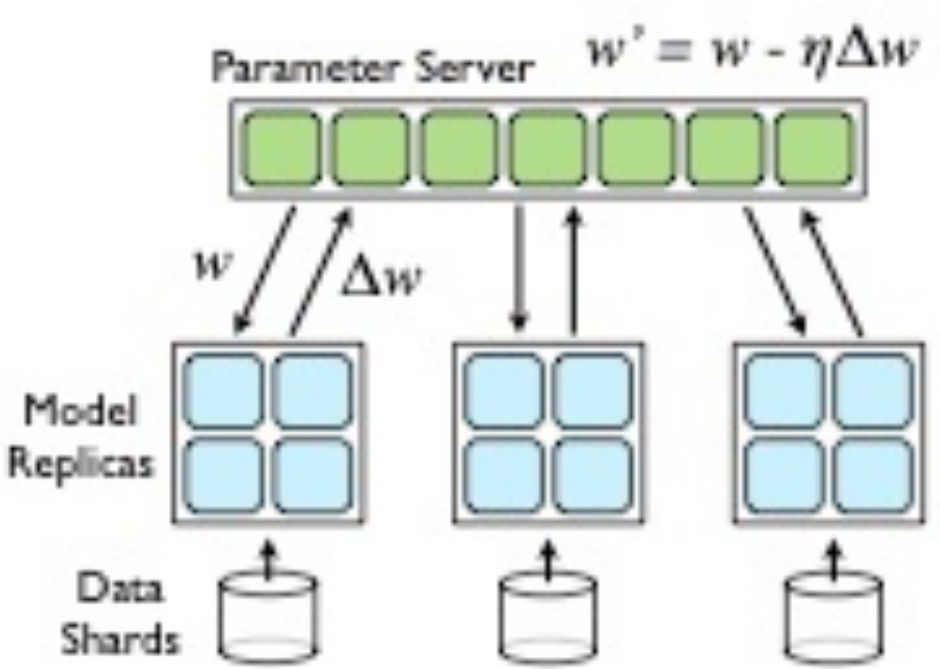
Combine Model and Data Parallelism

Model Parallelism



This appears in earlier work on graph systems …

Data Parallelism



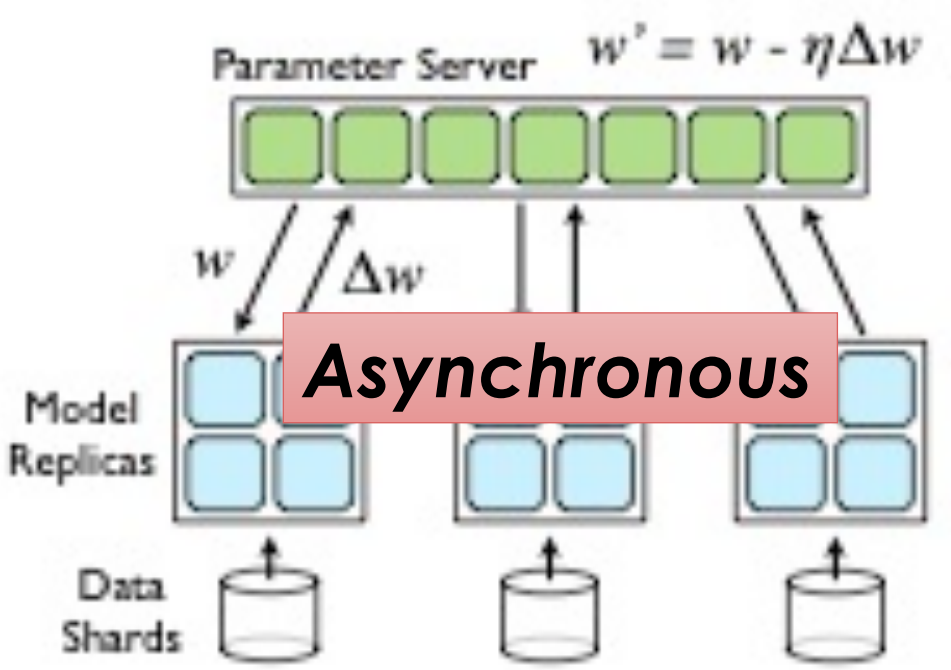
Downpour SGD

Combine Model and Data Parallelism

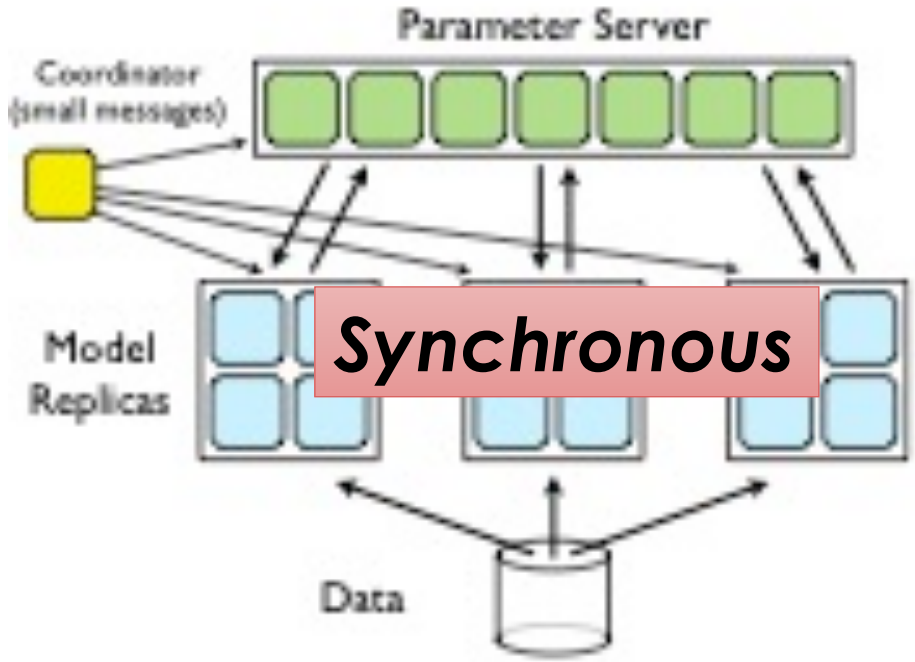
Machine 2

Machine 4

Data Parallelism



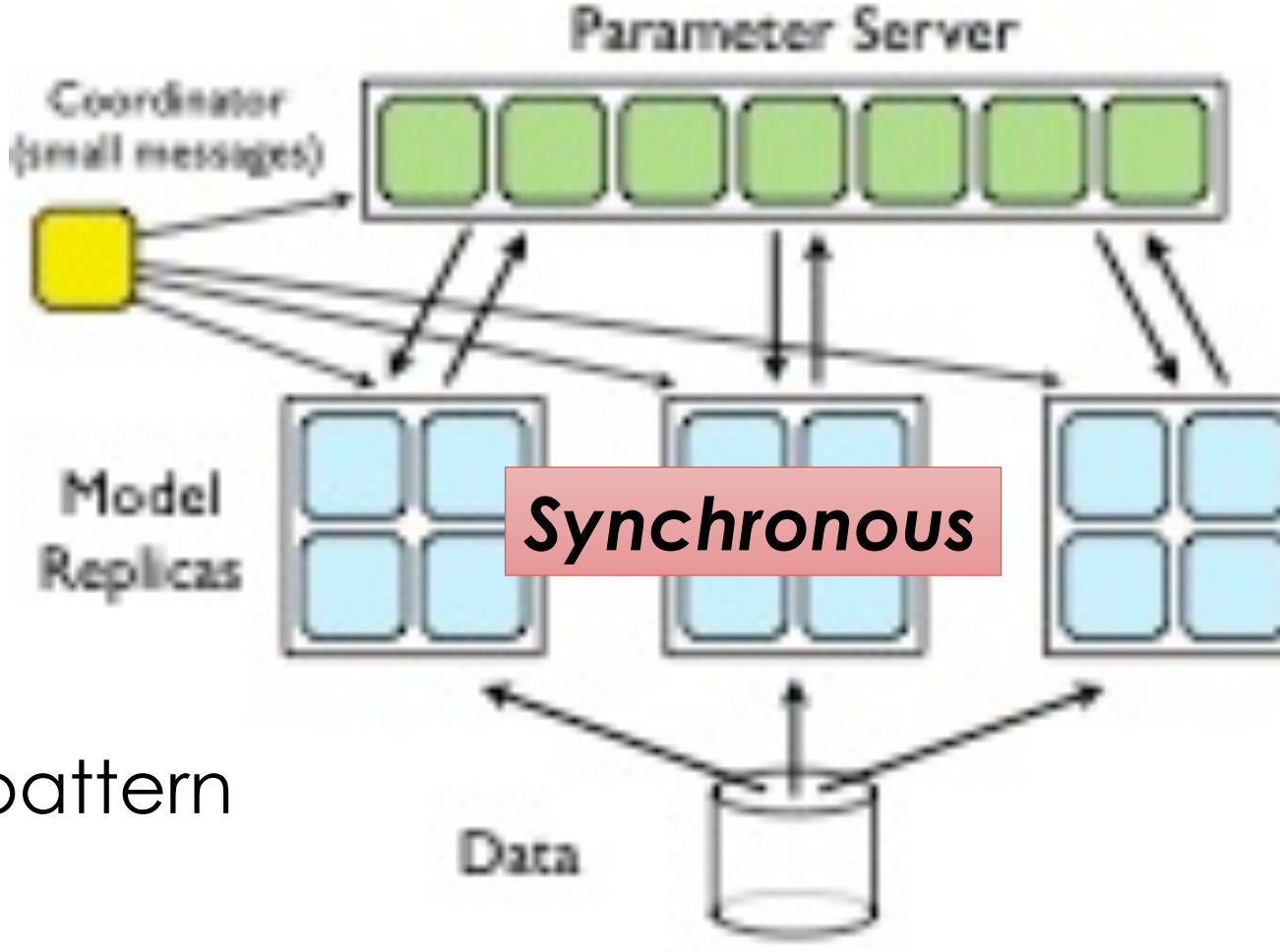
Downpour SGD



Sandblaster L-BFGS

Sandblaster L-BFGS

- L-BFGS
  - Commonly used for convex opt. problems
  - Requires repeated scans of all data
  - Robust, minimal tuning
- Naturally fits map-reduce pattern
- **Innovations:**
  - accumulate gradients and store outputs in a sharded key value store (parameter server)
  - Tiny tasks + backup tasks to mitigate stragglers

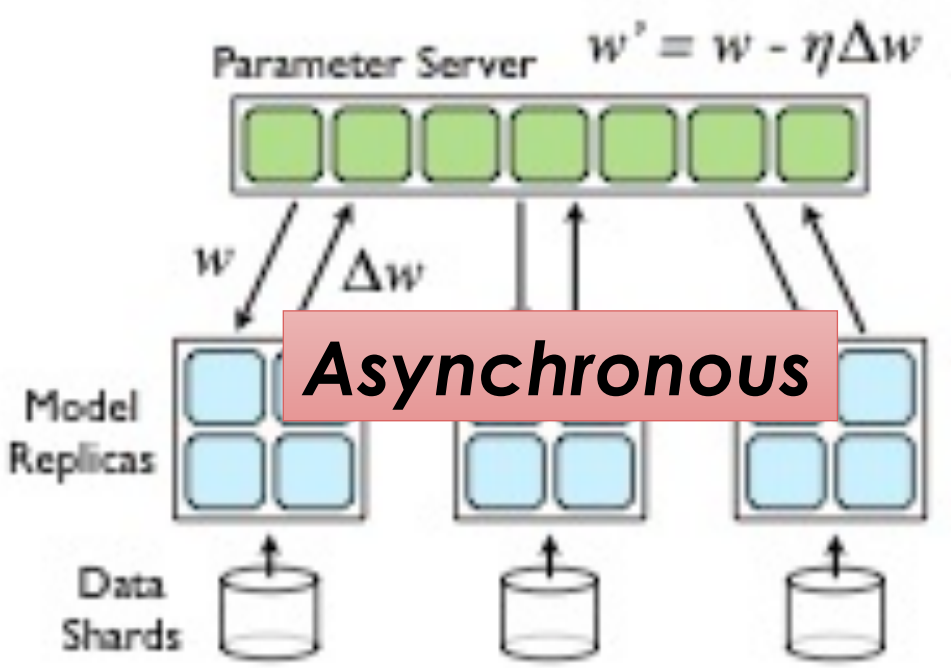


Combine Model and Data Parallelism

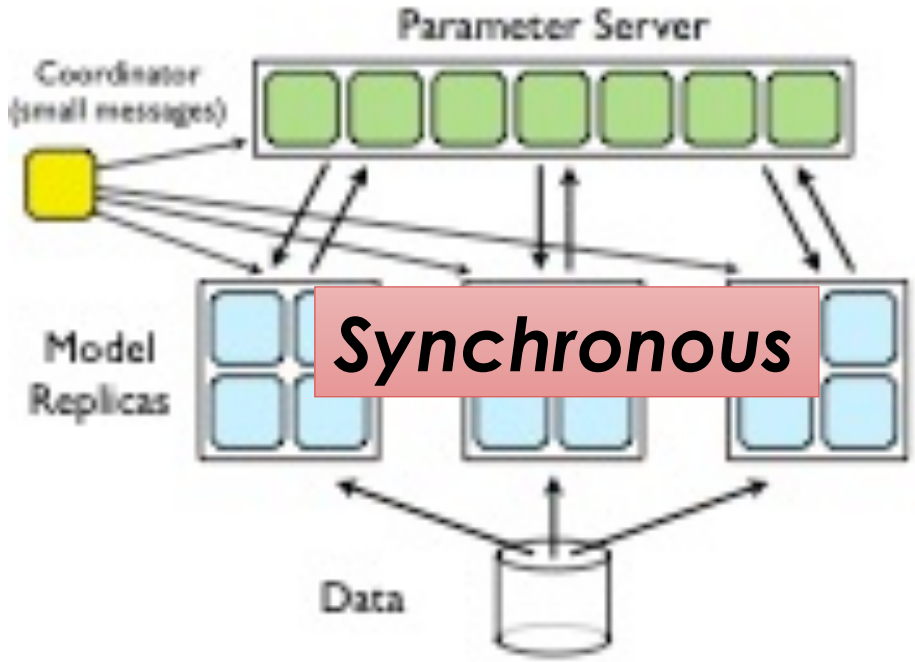
Machine 2

Machine 4

Data Parallelism



Downpour SGD

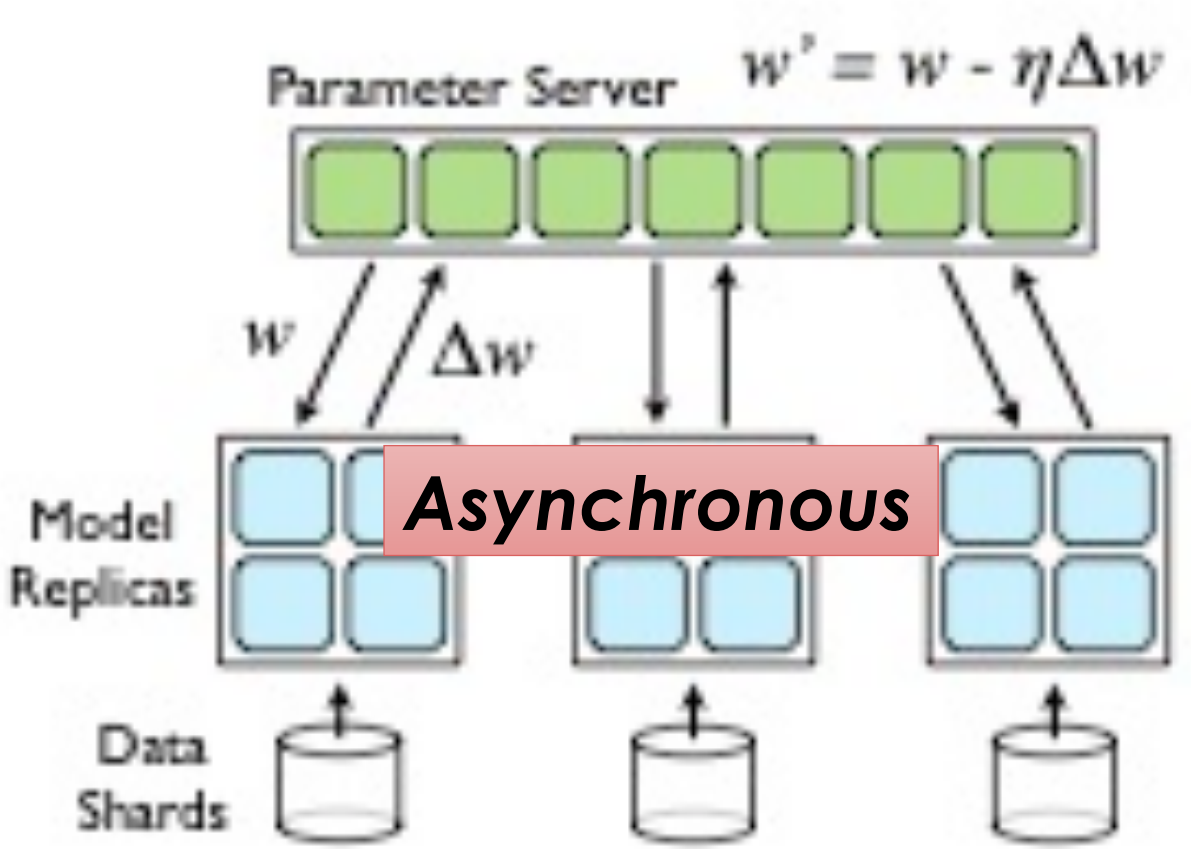


Sandblaster L-BFGS

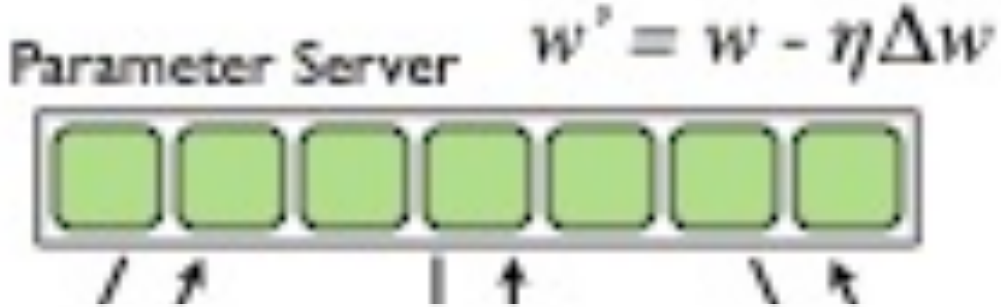
Downpour SGD

Claimed Innovations

- Parameter Server
- Combine model and data parallelism in an async. execution.
- Adagrad stabilization
- Warmstarting

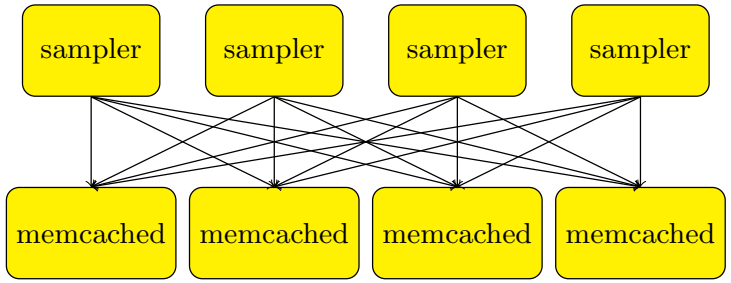


Parameter Servers

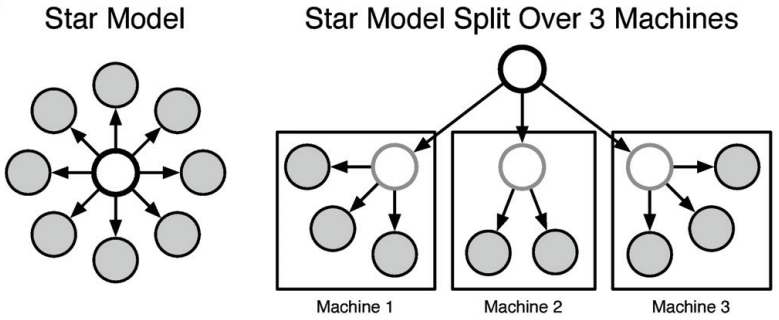


- Essentially a **sharded** key-value store
  - support for put, get, **add**
- Idea appears in earlier papers:

"*An Architecture for Parallel Topic Models*", Smola and Narayanamruthy. (VLDB'10)



"*Scalable Inference in Latent Variable Models*", Ahmed, Aly, **Gonzalez**, Narayanamruthy, and Smola. (WSDM'12)

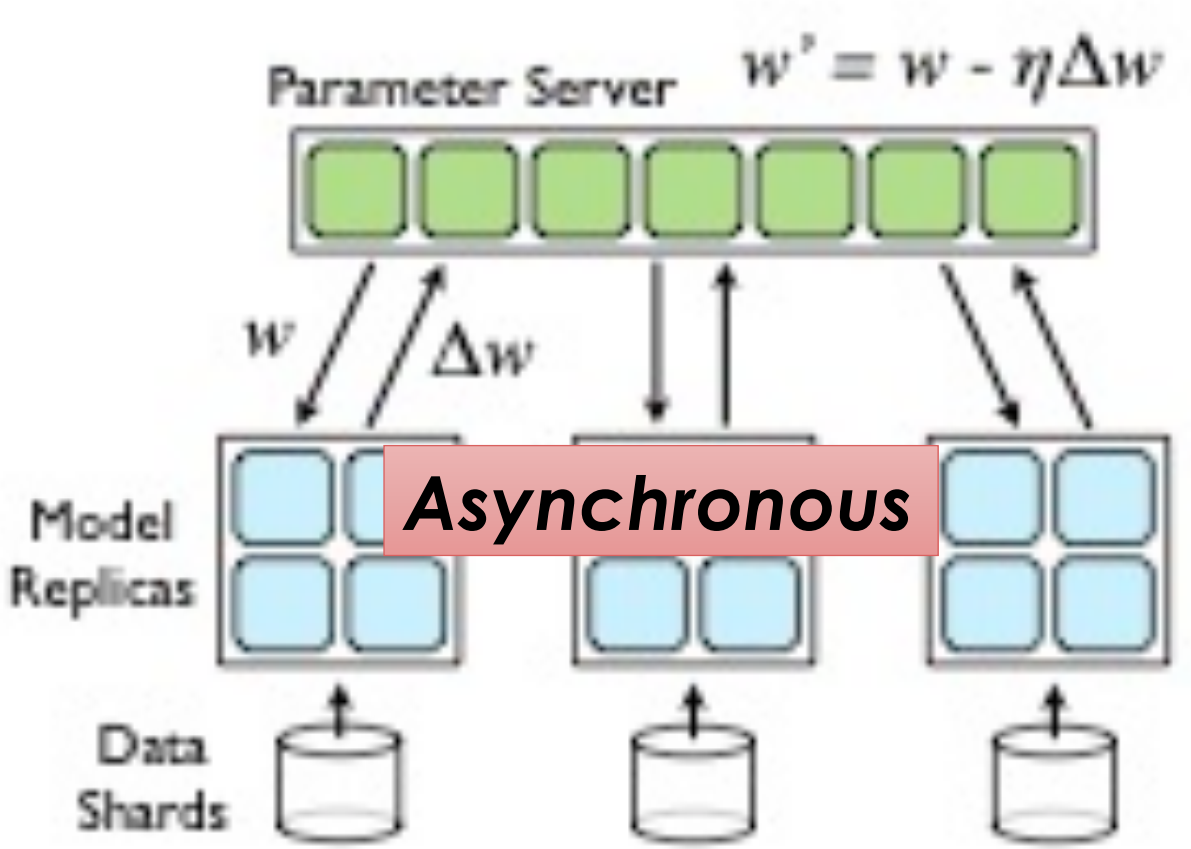


DistBelief was probably the first paper to call a sharded key-value store a Parameter Server.

Downpour SGD

Claimed Innovations

- Parameter Server
- Combine model and data parallelism in an **async. execution.**
- Adagrad stabilization
- Warmstarting



Adagrad Stabilization

- *Address large variability in magnitudes of gradients*
  - Rescale the gradients by an estimate of the diagonal variance
  - More recently superseded by Adam
- Stability is needed here to support asynchronous gradient updates

Warmstarting

Starting closer to a solution can help!

- Recall

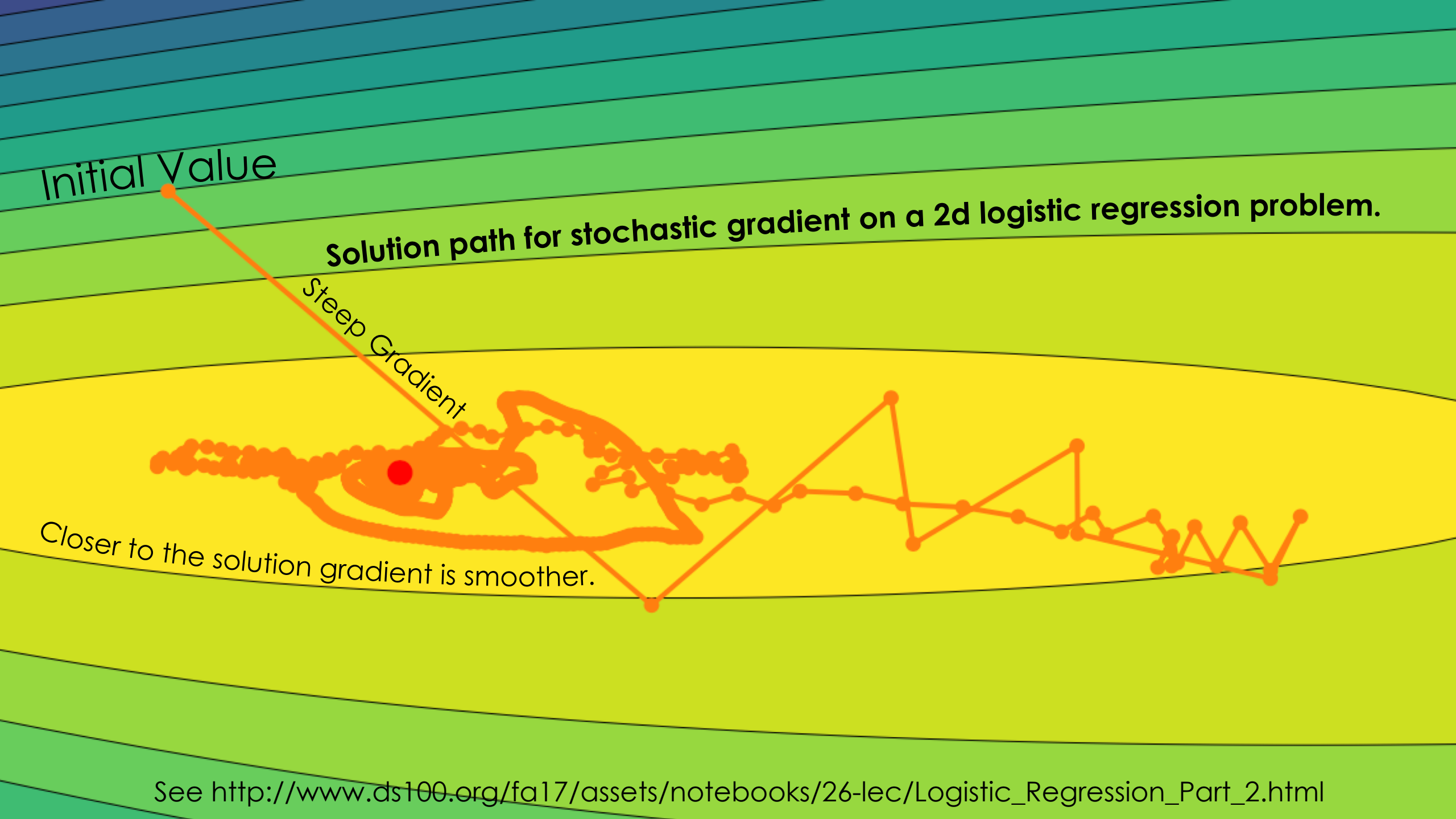
Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros ...)

For t from 0 to convergence:

$\mathcal{B} \sim$ Random subset of indices

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

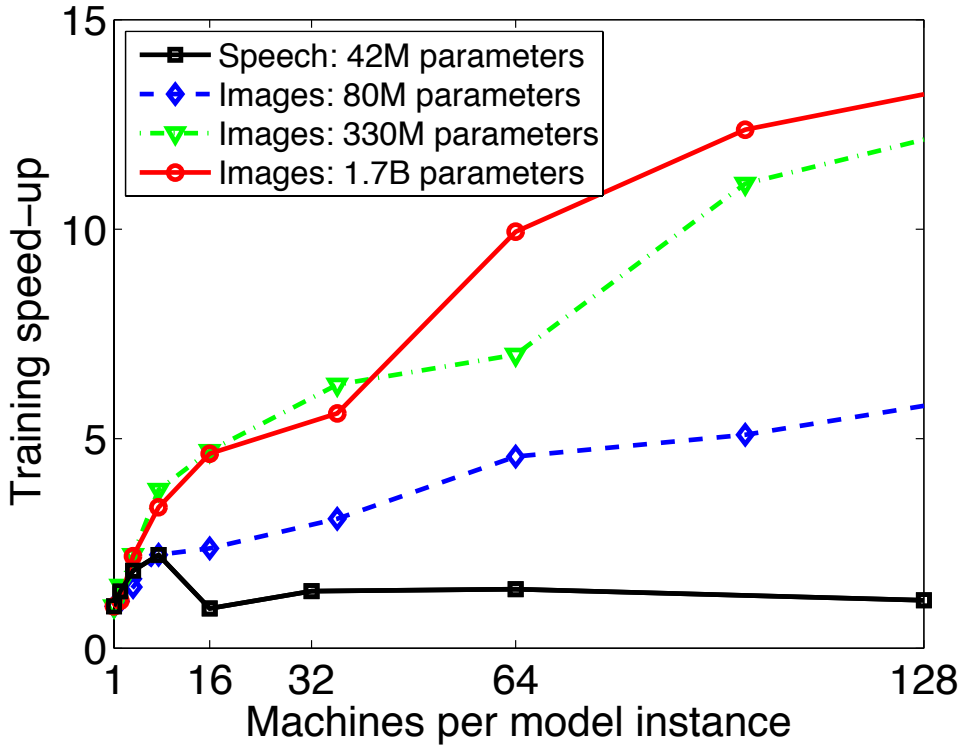


See http://www.ds100.org/fa17/assets/notebooks/26-lec/Logistic_Regression_Part_2.html

Key Results

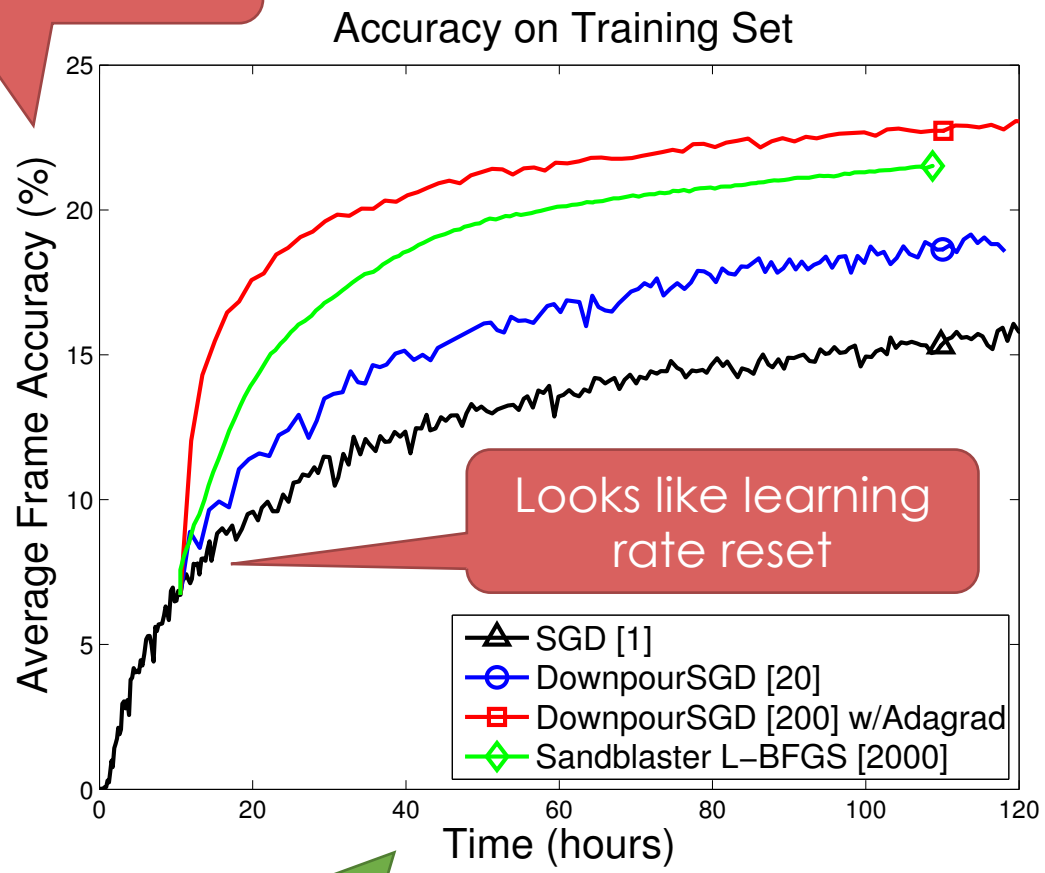
Model Parallelism

- Measured speedup to compute a single mini-batch
  - Is this a good metric?
- Results are not that strong…



Key Results: Training and Test Error

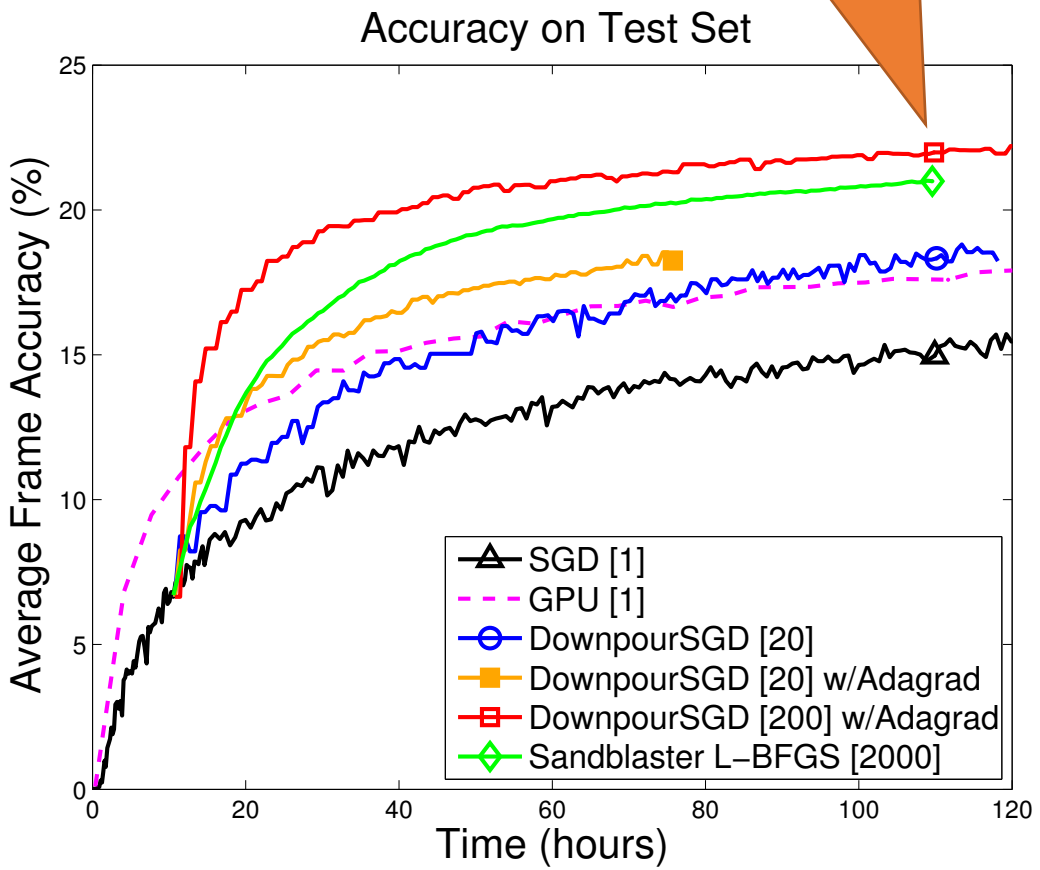
Weird 20K Error Metric



Looks like learning rate reset

Wall clock time is good.

Which would you use?



Why are they in the NY Times

- Trained a 1.7 billion parameter model (30x larger than state-of-the-art) (was it necessary?)
- Using 16,000 cores (efficiently?)
- Achieves 15.8 accuracy on ImageNet 20K (70% improvement over state of the art).
  - Non-standard benchmark
- Qualitatively interesting results

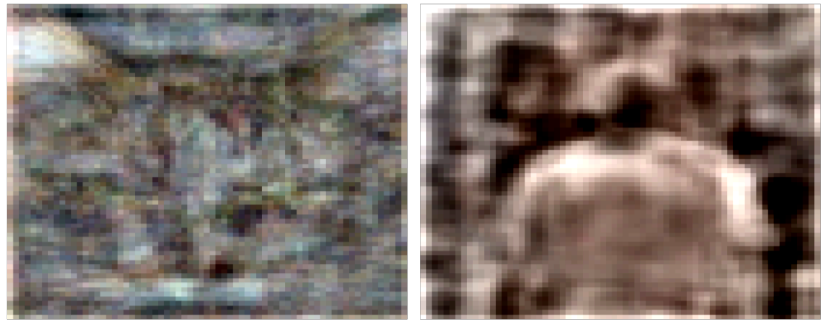


Figure 6. Visualization of the cat face neuron (left) and human body neuron (right).

Long-term Impact

- The **parameter server** appears in many later machine learning systems
- Downpour (**Asynchronous**) SGD has been largely **replaced by synchronous systems** for supervised training
  - Asynchrony is still popular in RL research
    - Why?
- Model parallelism is still used for large language models
  - Predated this work
- The neural network architectures studied here have been largely replaced by convolutional networks

Second Paper

- Generated a lot of press
  - Recently (Aug) surpassed by Fast.ai: "*Now anyone can train ImageNet in 18 minutes for $40.*" blog post

- Popularized linear learning rate scaling

2018 (Unpublished on Arxiv)

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

Abstract

Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD minibatch size. In this paper, we empirically show that on the ImageNet dataset large minibatches cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, we show no loss of accuracy when training with large minibatch sizes up to 8192 images. To achieve this result, we adopt a hyper-parameter-free linear scaling rule for adjusting learning rates as a function of minibatch size and develop a new warmup scheme that overcomes optimization challenges early in training. With these simple techniques, our Caffe2-based system trains ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour, while matching small minibatch accuracy. Using commodity hardware, our implementation achieves ~90% scaling efficiency when moving from 8 to 256 GPUs. Our findings enable training visual recognition models on internet-scale data with high efficiency.

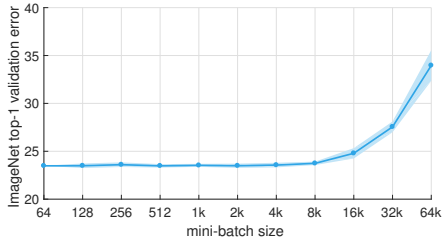


Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

1. Introduction

Scale matters. We are in an unprecedented era in AI research history in which the increasing data and model scale is rapidly improving accuracy in computer vision [22, 41, 34, 35, 36, 16], speech [17, 40], and natural language processing [7, 38]. Take the profound impact in computer vision as an example: visual representations learned by deep convolutional neural networks [23, 22] show excellent performance on previously challenging tasks like ImageNet classification [33] and can be transferred to difficult perception problems such as object detection and segmentation [8, 10, 28]. Moreover, this pattern generalizes: larger datasets and neural network architectures consistently yield improved accuracy across all tasks that benefit from pre-training [22, 41, 34, 35, 36, 16]. But as model and data scale grow, so does training time; discovering the potential and limits of large-scale deep learning requires developing novel techniques to keep training time manageable.

The goal of this report is to demonstrate the feasibility of, and to communicate a practical guide to, large-scale training with distributed *synchronous* stochastic gradient descent (SGD). As an example, we scale ResNet-50 [16] training, originally performed with a minibatch size of 256 images (using 8 Tesla P100 GPUs, training time is 29 hours), to larger minibatches (see Figure 1). In particular, we show that *with a large minibatch size of 8192, we can train ResNet-50 in 1 hour using 256 GPUs while maintaining*

Contrasting to the first paper

- **Synchronous** SGD
  - Much of the recent work has focused on synchronous setting
  - Easier to reason about
- Focus exclusively on data parallelism: **batch-size scaling**
- Focuses on the **generalization gap problem**

How do you distribute SGD?

Stochastic Gradient Descent

$\theta^{(0)} \leftarrow$ initial vector (random, zeros …)

For t from 0 to convergence:

$\mathcal{B} \sim$ Random subset of indices

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta_t \left(\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

Data Parallelism

Slow? (~150ms)
Depending on size of B

Batch Size Scaling

- Increase the batch size by adding machines

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \hat{\eta} \left(\frac{1}{k} \sum_{j=1}^{k} \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

- Each server processes a fixed batch size (e.g., n=32)
- As more servers are added (k) the effective overall batch size increases linearly
- Why do these additional servers help?

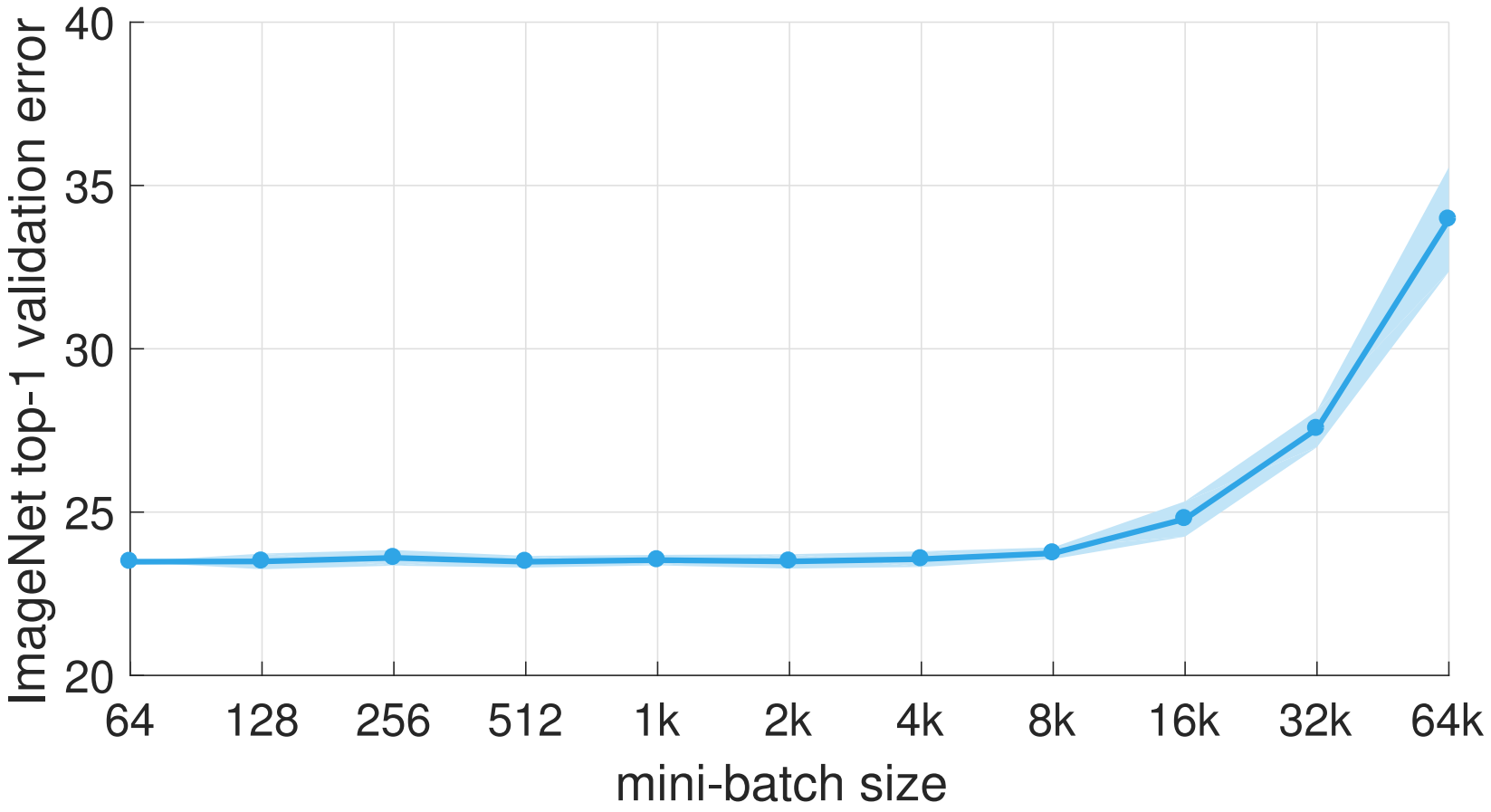
Bigger isn't Always Better

- Motivation for larger batch sizes
  - More opportunities for parallelism → but is it useful?
  - Recall (1/n variance reduction):

$$\frac{1}{n}\sum_{i=1}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta)) \approx \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\nabla_\theta \mathbf{L}(y_i, f(x_i;\theta))$$

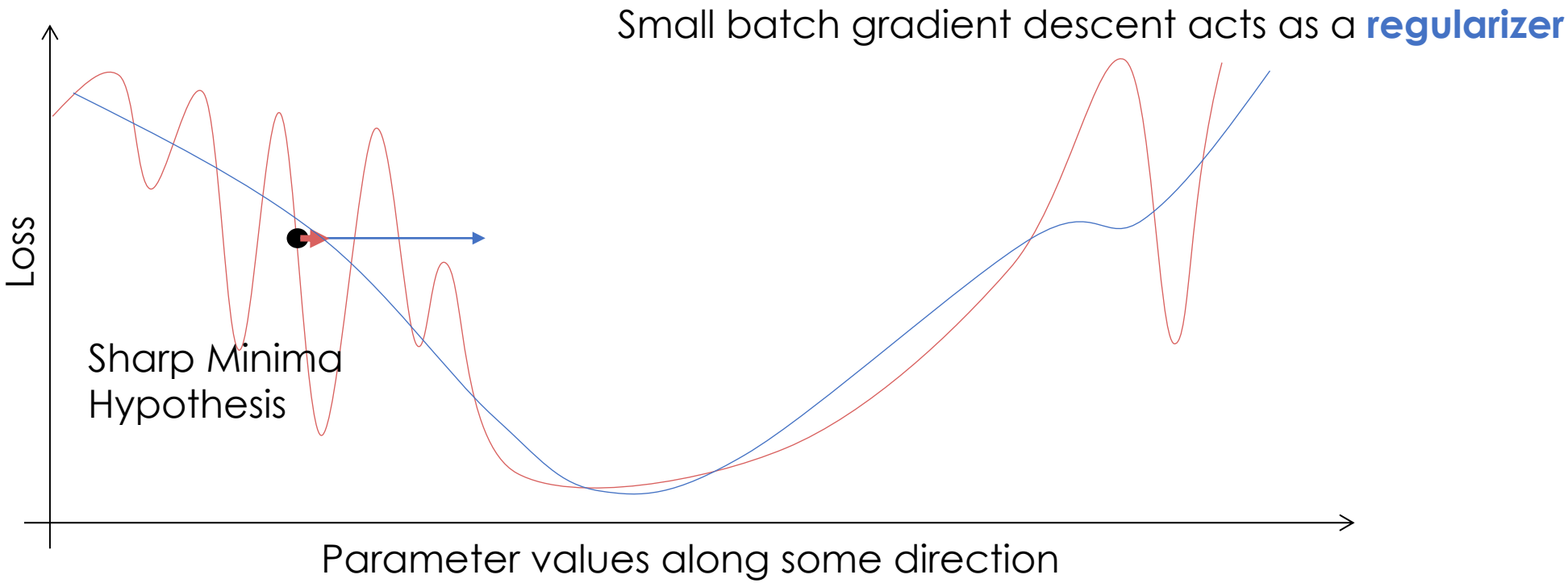
- Is a variance reduction helpful?
  - Only if it let's you take bigger steps (move faster)
  - Doesn't affect the final answer…

Generalization Gap Problem



Larger batch sizes harm generalization performance.

Rough "Intuition"



Key problem: *Addressing the generalization gap for large batch sizes.*

Solution: Linear Scaling Rule

- Scale the learning rate linearly with the batch size

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \hat{\eta} \overbrace{= \eta k}^{} \left(\frac{1}{k} \sum_{j=1}^{k} \frac{1}{|\mathcal{B}_j|} \sum_{i \in \mathcal{B}_j} \nabla_\theta \mathbf{L}(y_i, f(x_i; \theta)) \Big|_{\theta=\theta^{(t)}} \right)$$

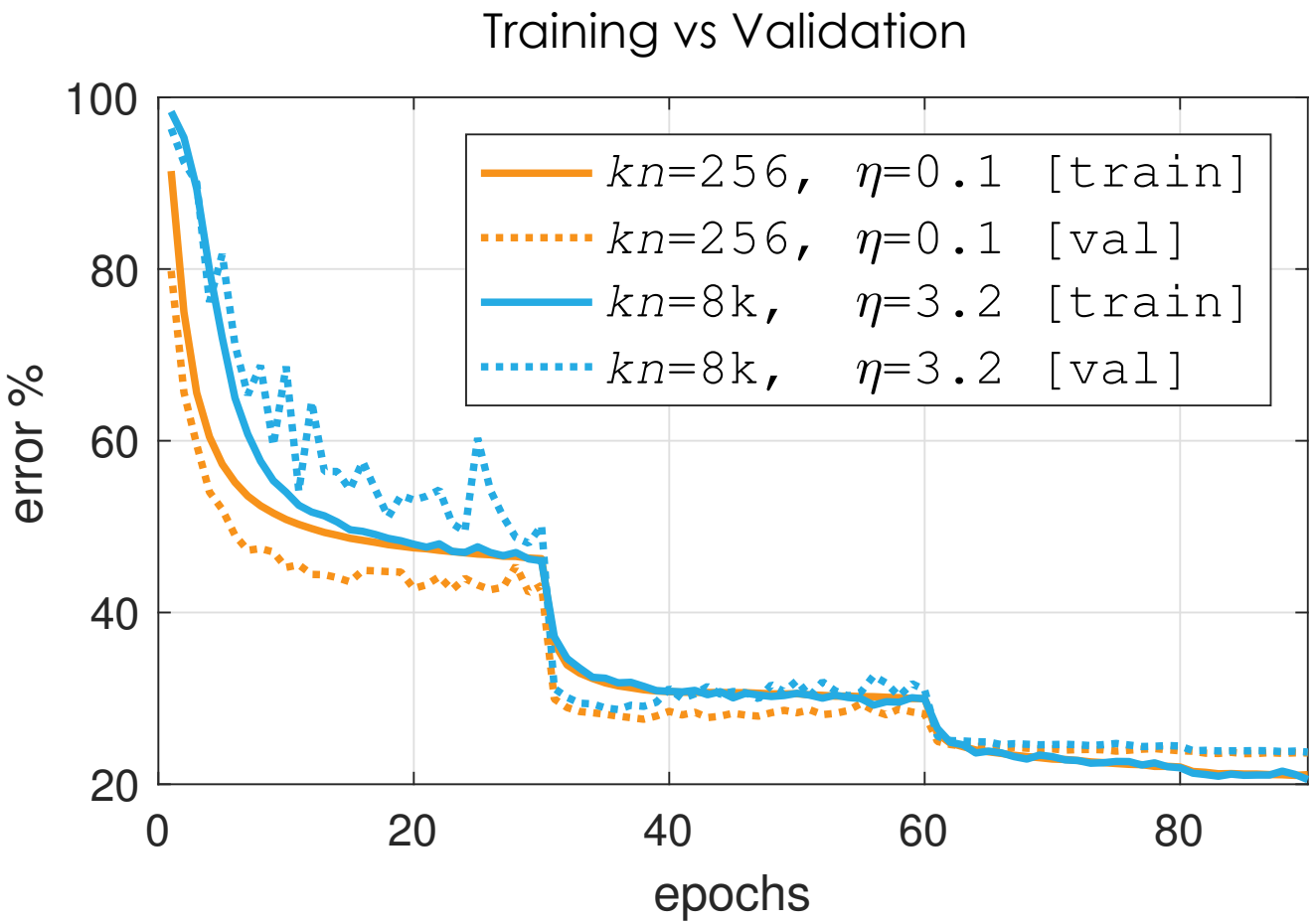
- Addresses generalization performance by **taking larger steps** (also improves training convergence)

- **Sub-problem:** *Large learning rates can be destabilizing in the beginning.* Why?
  - **Gradual warmup solution:** increase learning rate scaling from constant to linear in first few epochs
  - Doesn't help for very large k…

Other Details

- **Independent Batch Norm**: Batch norm calculation applies only to local batch size (n).
- **All-Reduce:** Recursive halving and doubling algorithm
  - Used instead of popular ring reduction (fewer rounds)
- **Gloo** a library for efficient collective communications
- **Big Basin GPU Servers:** Designed for deep learning workloads
  - Analysis of communication requirements → latency bound
- **No discussion on straggler or fault-tolerance**
  - **Why?!**

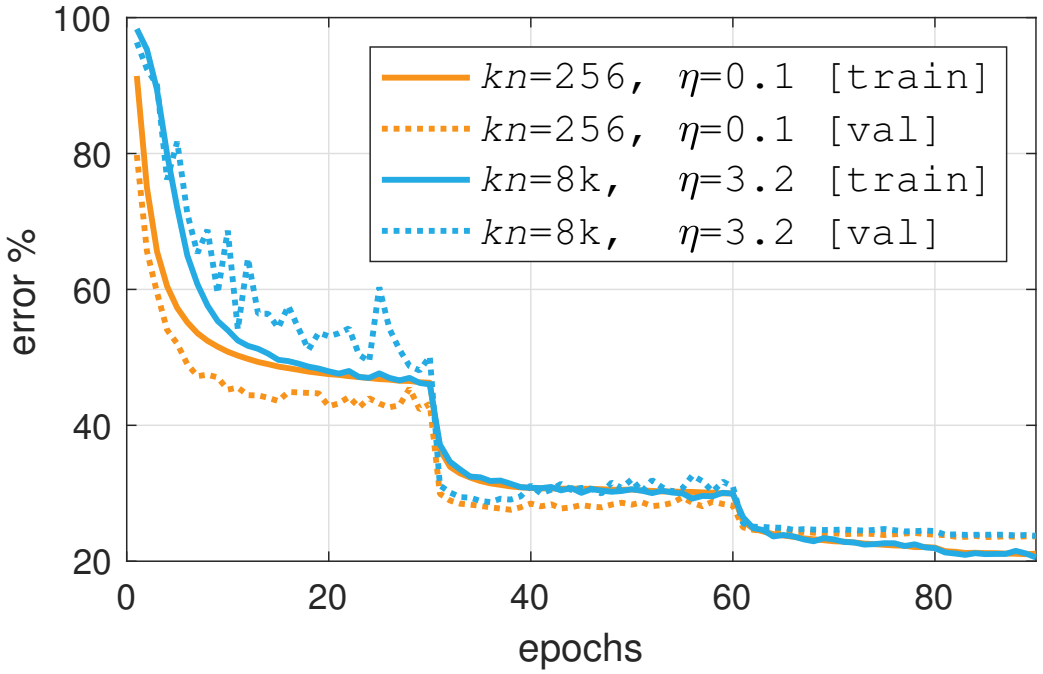
Key Results

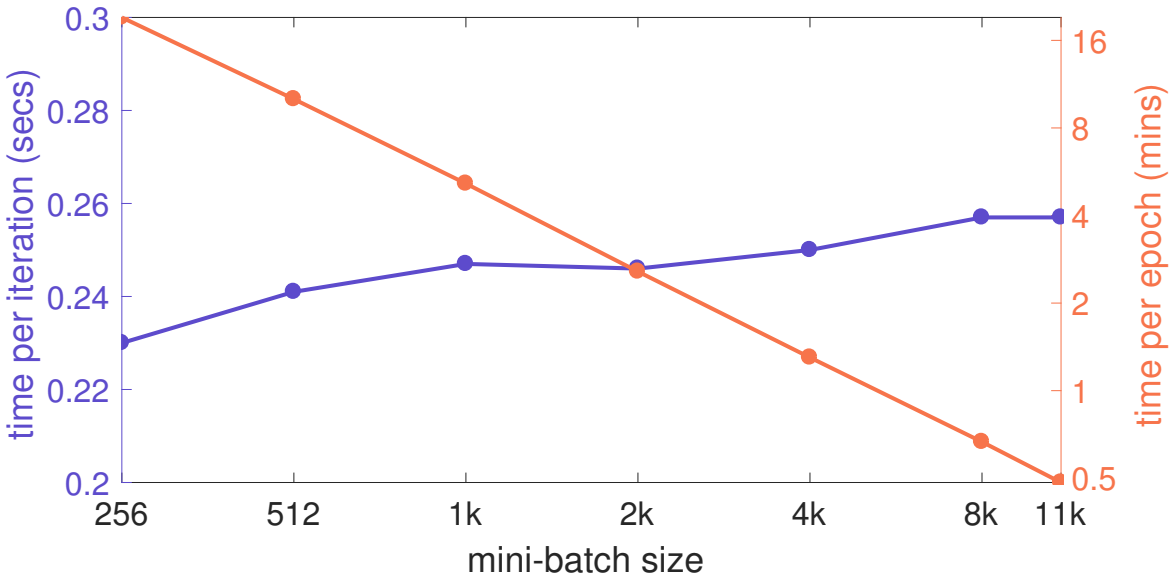


All curves closely match using the linear scaling rule.

Note learning rate schedule drops.

Key Results





$$\frac{\text{“Learning”}}{\text{Epoch}}$$

Machine Learning

$$\frac{\text{Epoch}}{\text{Second}}$$

System

Key Results

- Train ResNet-50 to state-of-the-art on 256 GPUs in 1 hour
  - 90% scaling efficiency

- Fairly careful study of the linear scaling rule
  - Observed limits to linear scaling do not depend on dataset size
  - Cannot scale parallelism with dataset size

Long-term Impact

- Still early (this paper is not yet published)
- Ideas that will appear in other papers
  - Linear rate scaling
  - Independent batch norm
- Paper points to limits of Synchronous SGD → need new methods to accelerate training
  - Eg.: Fast.ai → curriculum learning though scale variation

Questions for Discussion

- Distributed model training is not very common. Why?
- Should we return to asynchrony?
  - What is needed to address issues with asynchronous training?
- How will changes in hardware affect distributed training
  - E.g., faster GPUs → larger batches, faster networks → smaller batches …
- How will the emergence of "dynamic models" and large "mixture of expert models" affect distributed training?