Large-Scale Optimization for Distributed AI

Yang You

Presidential Young Professor at National University of Singapore

NUS CS5260 2023 Spring

Outline

- **Why is deep learning slow?**
- Why is scaling deep learning difficult?
- How to scale up deep learning?
- Experimental Results.
- Our method in real-world applications.

Supercomputers are becoming popular in AI companies

- "Trends in AI systems, the first one I think about is the shift to **High Performance Computing (HPC)** type workloads" — Dr. Carlos Guestrin (UW & Apple) @ GeekWire Cloud Tech Summit
- "Investments in computer systems - and I think the bleeding-edge of AI, and deep learning specifically, is shifting to **High Performance Computing**" — Dr. Andrew Ng (Stanford & deeplearning.ai) @ International Supercomputing Conference

Deep learning is expensive

- ResNet-50 (a tiny model) training:
  - 10^{18} operations = 90 epochs × 1.3M images × 7.7B ops per image
    - 1 Intel Core i5 CPU: forever
    - 1 M40 GPUs: 14 days
    - 8 P100 GPUs: 29 hours
- Best Supercomputer in 2017: 2×10^{17} operations per sec
  - Can we finish the ResNet-50 training in 5 seconds? No!
  - Because we can't make full use of a supercomputer!
- Best Supercomputer in 2017: 2×10^{17} operations per sec
- How can we make full use of a supercomputer?
- A simple definition of Supercomputer
  - A cluster with high-speed interconnect and fast chips
  - e.g. 100+ Gbit/s infiniband and Petaflops performance

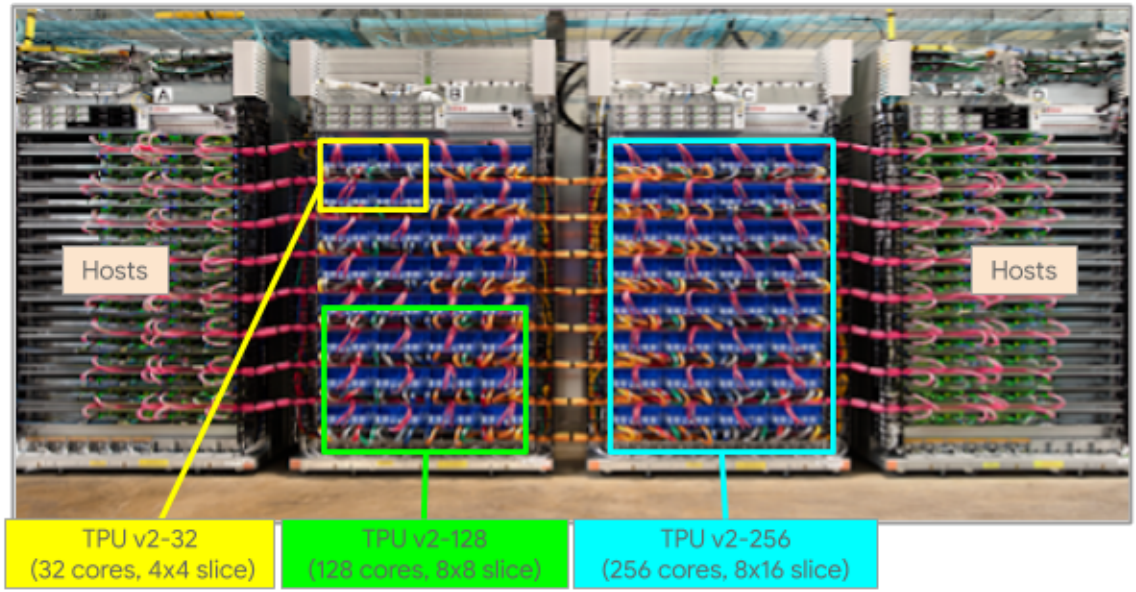
Mini-Batch SGD (Stochastic Gradient Descent)

- 1. Take B data samples x_t^i $(i \in \{1, ..., B\})$ at iteration t
- 2. Compute gradients of loss function w.r.t. weights:

$$g_t = \frac{1}{B} \sum_{i=1}^{B} \nabla f(x_t^i, w_{t-1})$$

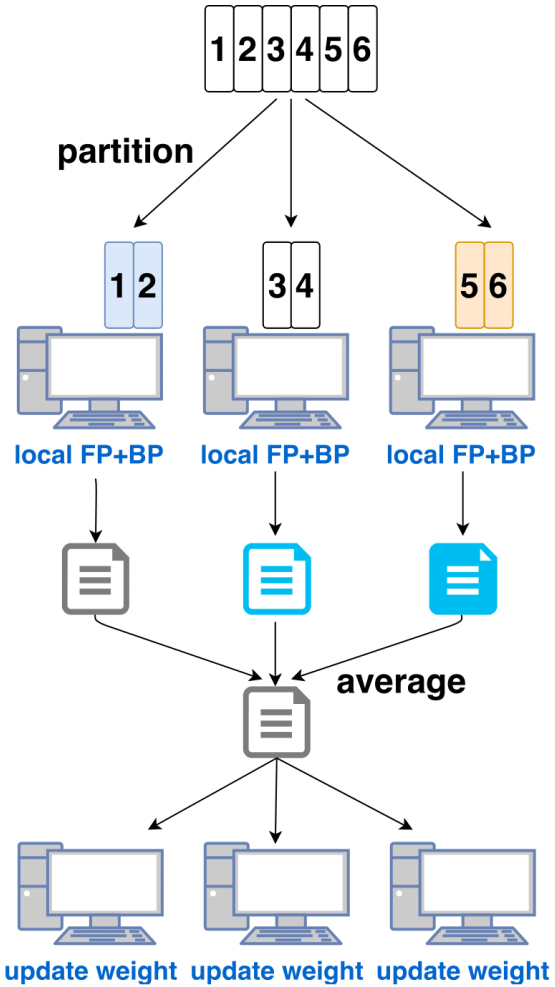
- 3. Update the weights: $w_t = w_{t-1} - \eta * g_t$
- Notations
  - B: batch size
  - η: learning rate
  - w_t: weights at iteration t
  - x_t: data samples at iteration t
  - f: loss function (e.g. least squares error)
  - g_t: gradients of the loss function at iteration t

How can we make full use of a supercomputer?



- A supercomputer has thousands of processors
  - Single processor frequency stops scaling (no free lunch)
  - Previous free lunch: we wait for 18 months and our code is $2x$ faster
- How to make full use of thousands of processors?
  - **Data Parallelism**: Parallelize the data (focus of this talk)
  - **Model Parallelism 1**: Parallelize within each layer
  - **Model Parallelism 2**: Parallelize across different layers

Data-Parallelism



1. partition the data to all the nodes

2. each node does local Forward Pass and Backward Pass on its own data

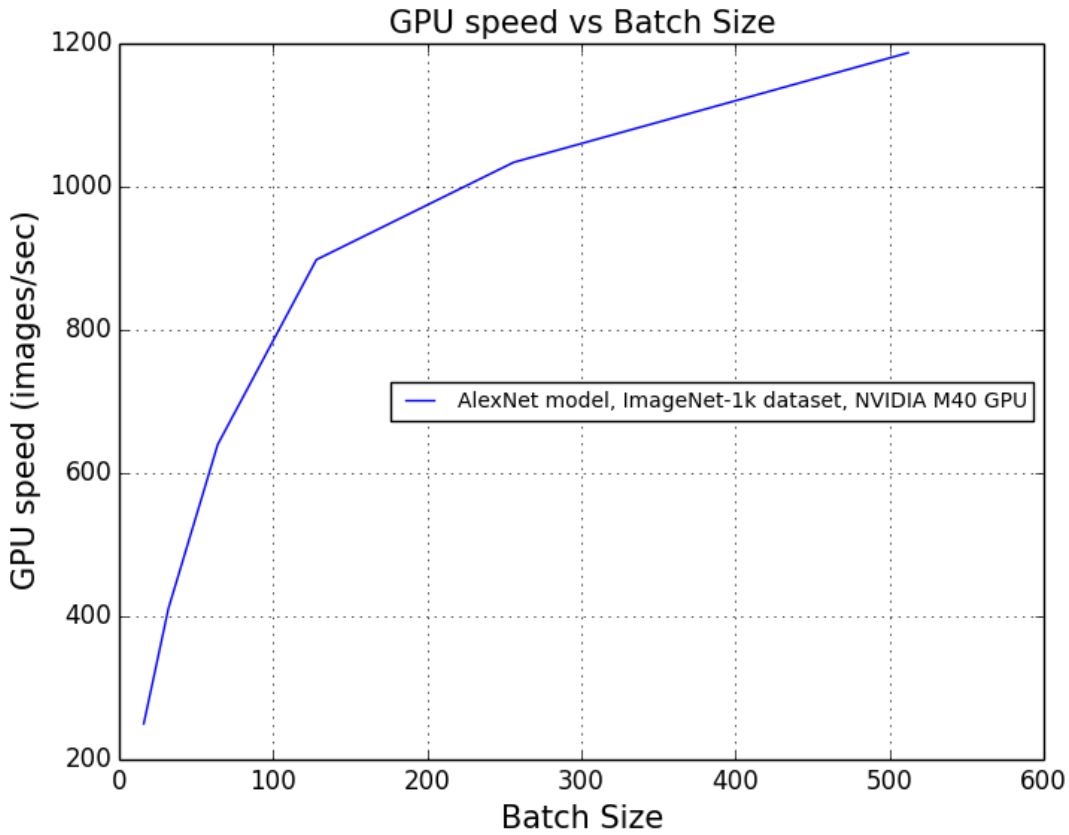
3. each node gets its local gradient

4. get the average of all the local gradients and send a copy of the global gradient to each node

5. each node uses the global gradient to update the local weight

- How to increase parallelism? Large-Batch Training!

Why Large-Batch can speed up DNN training?



- $B = 512$, the GPU achieves peak performance
- If we have 16 GPUs, we need a batch size of **8192** (16×512)
  - make sure each GPU is efficient

Why Large-Batch can speed up DNN training?

Batch Size	Epochs	Iterations
512	100	250,000
1024	100	125,000
2048	100	62,500
4096	100	31,250
8192	100	15,625
...	...	...
1,280,000	100	100

- ImageNet dataset: 1,280,000 data points
- Goal: get the same accuracy in the same epochs
  - **fixed epochs = fixed number of floating point operations**
  - needs much fewer iterations: speedup!

Fix the number of epochs and increase the batch size

Batch Size	#Epochs	#Iterations	#Nodes	Single Iteration Time
512	100	250,000	1	t_1
1024	100	125,000	2	$t_1 + (\log_2(2))t_2$
2048	100	62,500	4	$t_1 + (\log_2(4))t_2$
4096	100	31,250	8	$t_1 + (\log_2(8))t_2$
8192	100	15,625	16	$t_1 + (\log_2(16))t_2$
...	...	...		
1,280,000	100	100	2500	$t_1 + (\log_2(2500))t_2$

- Example: ImageNet dataset (1.28M data points)
- 1 epoch: statistically touches the whole dataset once
- Batch size $=$ 512 for each node (e.g. CPU with GPUs or TPUs)
- t_1: computation time, t_2: p2p communication time $(\alpha + |w|\beta)$[1]
  - $t_1 >> t_2$ is possible for ImageNet training by Inifniband[2]

[1] α is latency, β is inverse of bandwidth

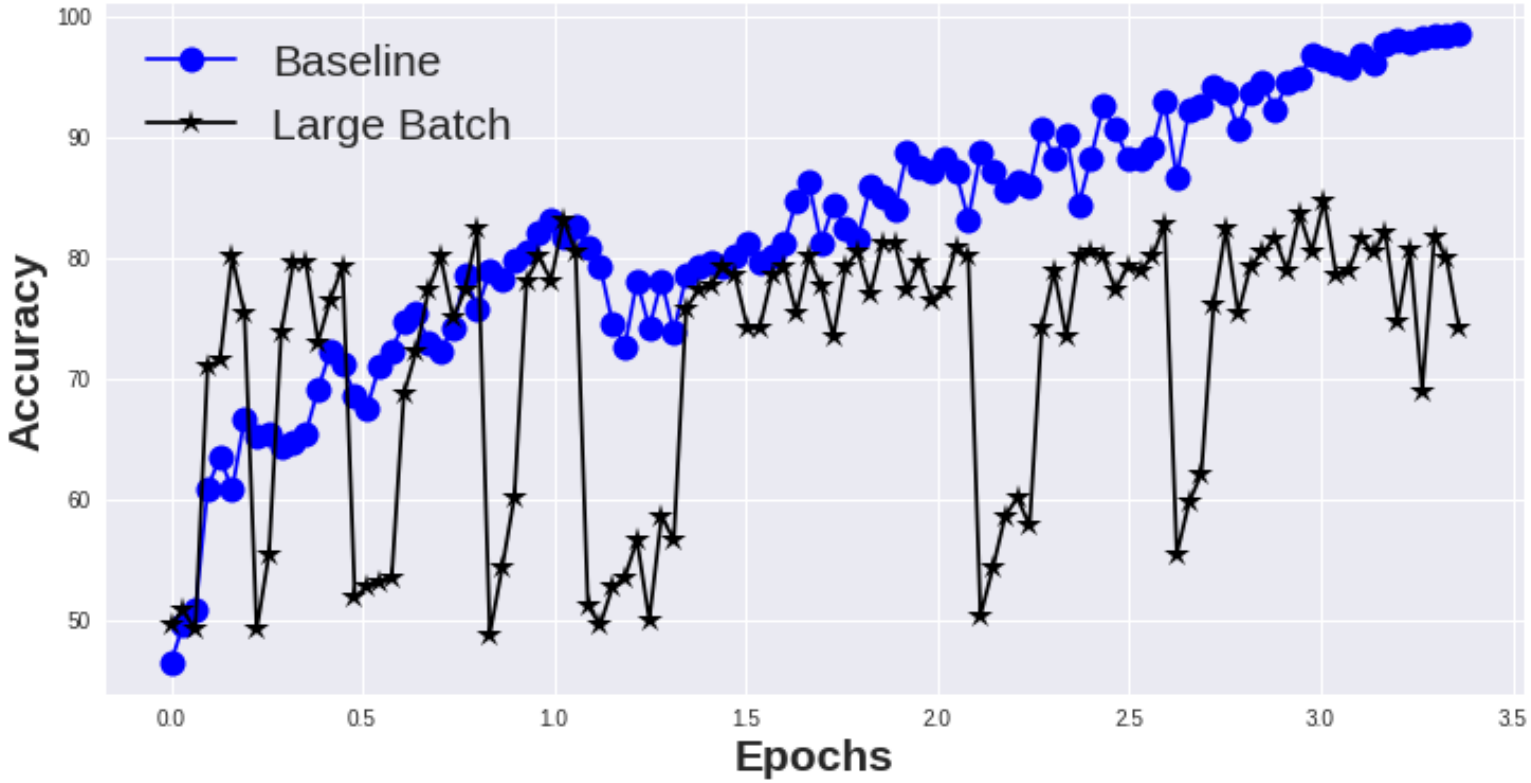
[2] Goyal, Dollar, Girshick, Noordhuis, Wesolowski, Kyrola, Tulloch, Jia, He, *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, (Facebook Report 2017)

Summary of Motivation

- Pick a Commonly-Used Approach in DNN Training?
  - Data-Parallelism Mini-Batch SGD (e.g. Caffe, Tensorflow, Torch)
  - recommended by Dr. Bryan Catanzaro (NVIDIA VP)
- How to speed up Mini-Batch SGD?
  - Use more processors (e.g. GPU)
- How to make each GPU efficient if we use many GPUs?
  - Give each GPU enough computations (find the right B)
- How to give each GPU enough computations?
  - Use large batch size (use PB)

Difficulties of Large-Batch Training

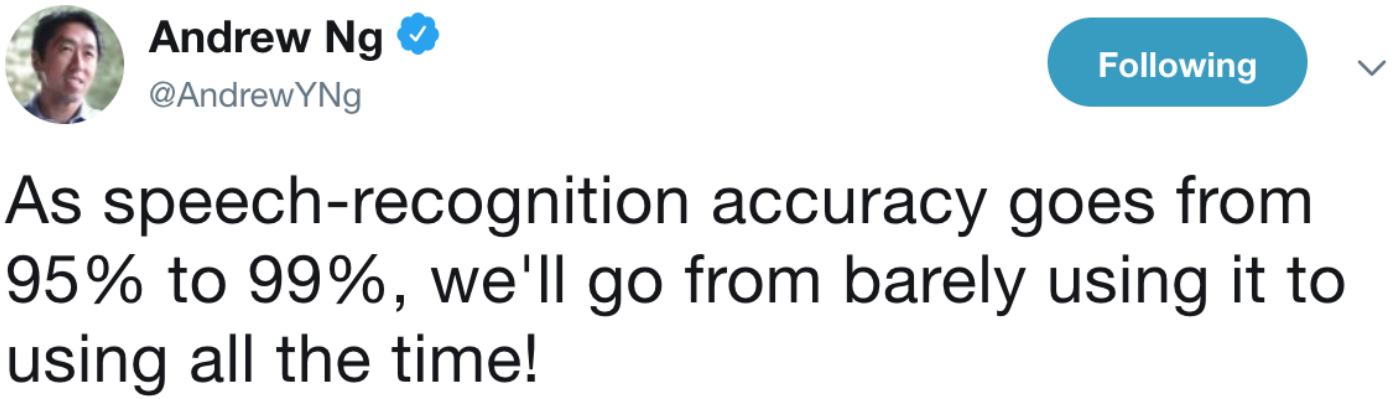
- **Lose Accuracy** (in fixed number of epochs)
- It is a hard numerical optimization problem
  - It is not sufficient to have great implementation and hardware



- Toy Application: MNIST by LSTM (baseline batch: 256; large batch: 8192)
- 1 epoch: statistically touches the whole dataset once

Challenge: can we keep the accuracy after a big speedup?

- 1000-class ImageNet dataset by ResNet-50
  - 76.3% accuracy in 90 epochs
  - 76.3% is very different from 75.3%, which only needs 60 epochs



- The last 1% accuracy is very important but hard to achieve

Outline

- Why is deep learning slow?
- **Why is scaling deep learning difficult?**
- How to scale up deep learning?
- Experimental Results.
- Our method in real-world applications.

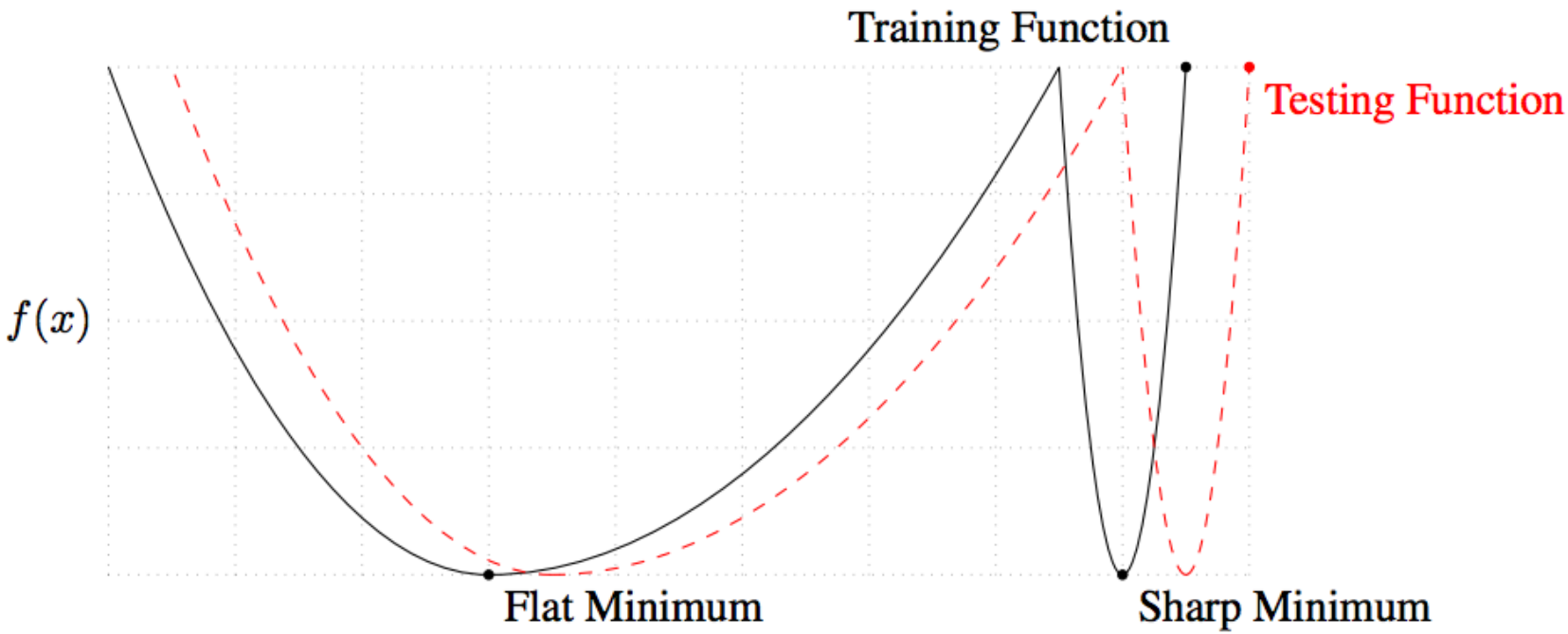
Difficulties of Large-Batch Training

- Why do we lose accuracy?
  - Generalization Problem[3]
    - High training accuracy, but low testing accuracy
  - Optimization Difficulty[4]
    - Hard to get the right hyper-parameters (even with an auto-tuner)

[3] Keskar et al., *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*, 2017 (ICLR)

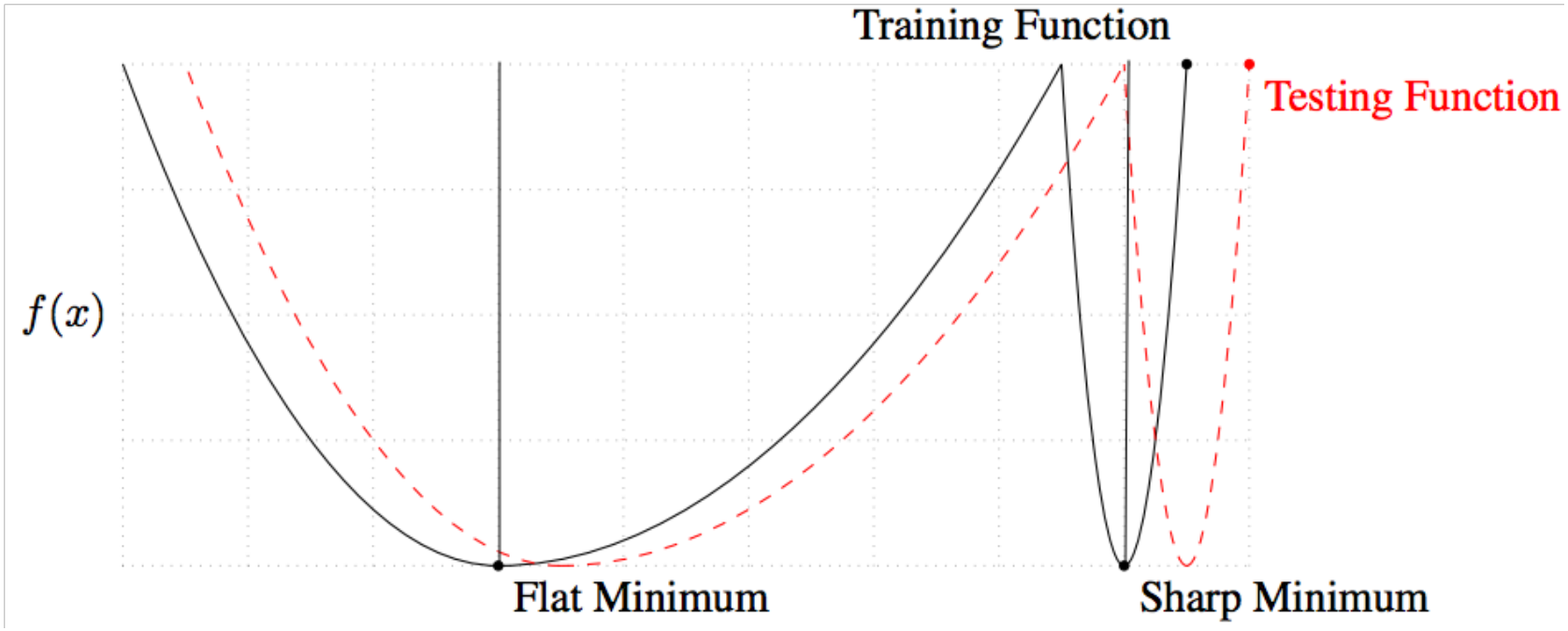
[4] Goyal et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, 2017 (Facebook Report)

Generalization Problem (Keskar et al.)



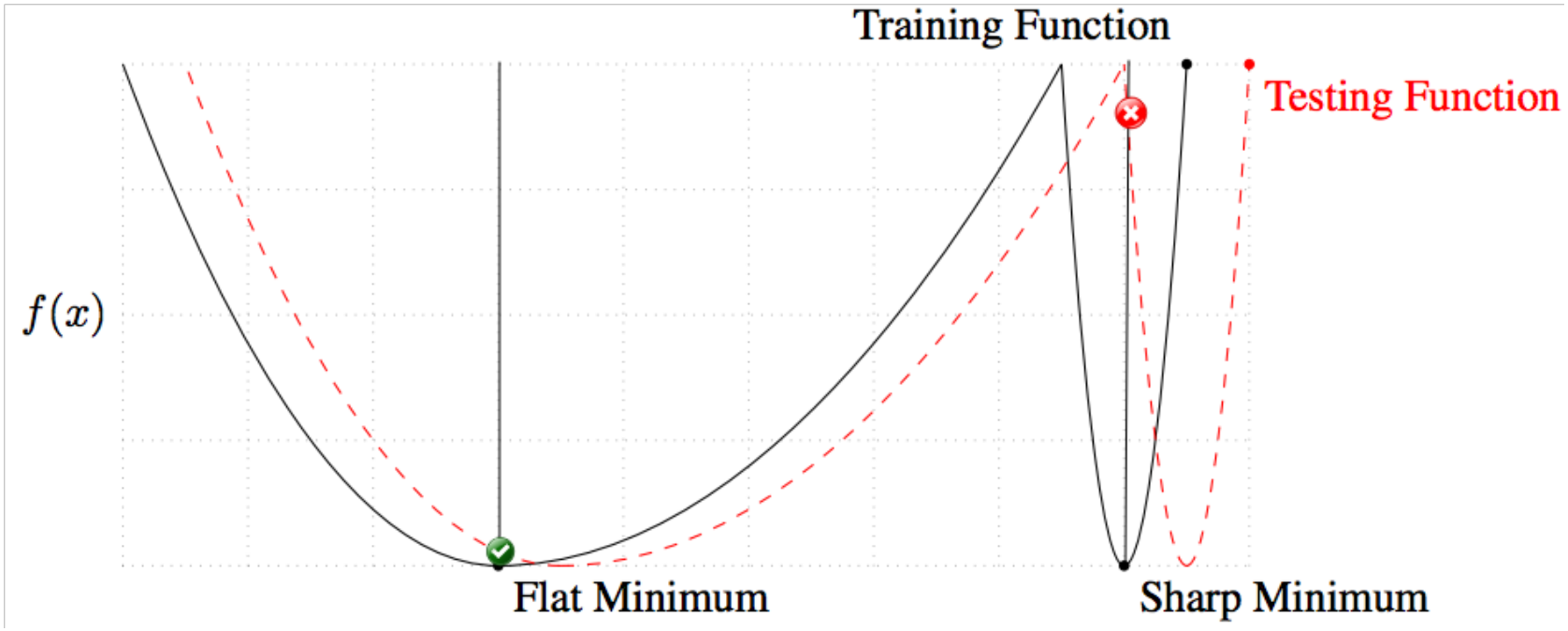
- Figure Credit: Keskar et al.
- Draw a vertical line at the minimum of the solid line (training function), where is the intersection between the vertical line and the dashed line (testing function)?
- Large-batch training is a sharp minimum problem
  - even if you can train a good model, it is hard to generalize
  - high training accuracy :-) but low testing accuracy :-(

Generalization Problem (Keskar et al.)



- Figure Credit: Keskar et al.
- Draw a vertical line at the minimum of the solid line (training function), where is the intersection between the vertical line and the dashed line (testing function)?
- Large-batch training is a sharp minimum problem
  - even if you can train a good model, it is hard to generalize
  - high training accuracy :-) but low testing accuracy :-(

Generalization Problem (Keskar et al.)



- Figure Credit: Keskar et al.
- Draw a vertical line at the minimum of the solid line (training function), where is the intersection between the vertical line and the dashed line (testing function)?
- Large-batch training is a sharp minimum problem
  - even if you can train a good model, it is hard to generalize
  - high training accuracy :-) but low testing accuracy :-(

Large-Batch Training

- Sharp Minimum: Poor Generalization Performance

Regular-Batch Training

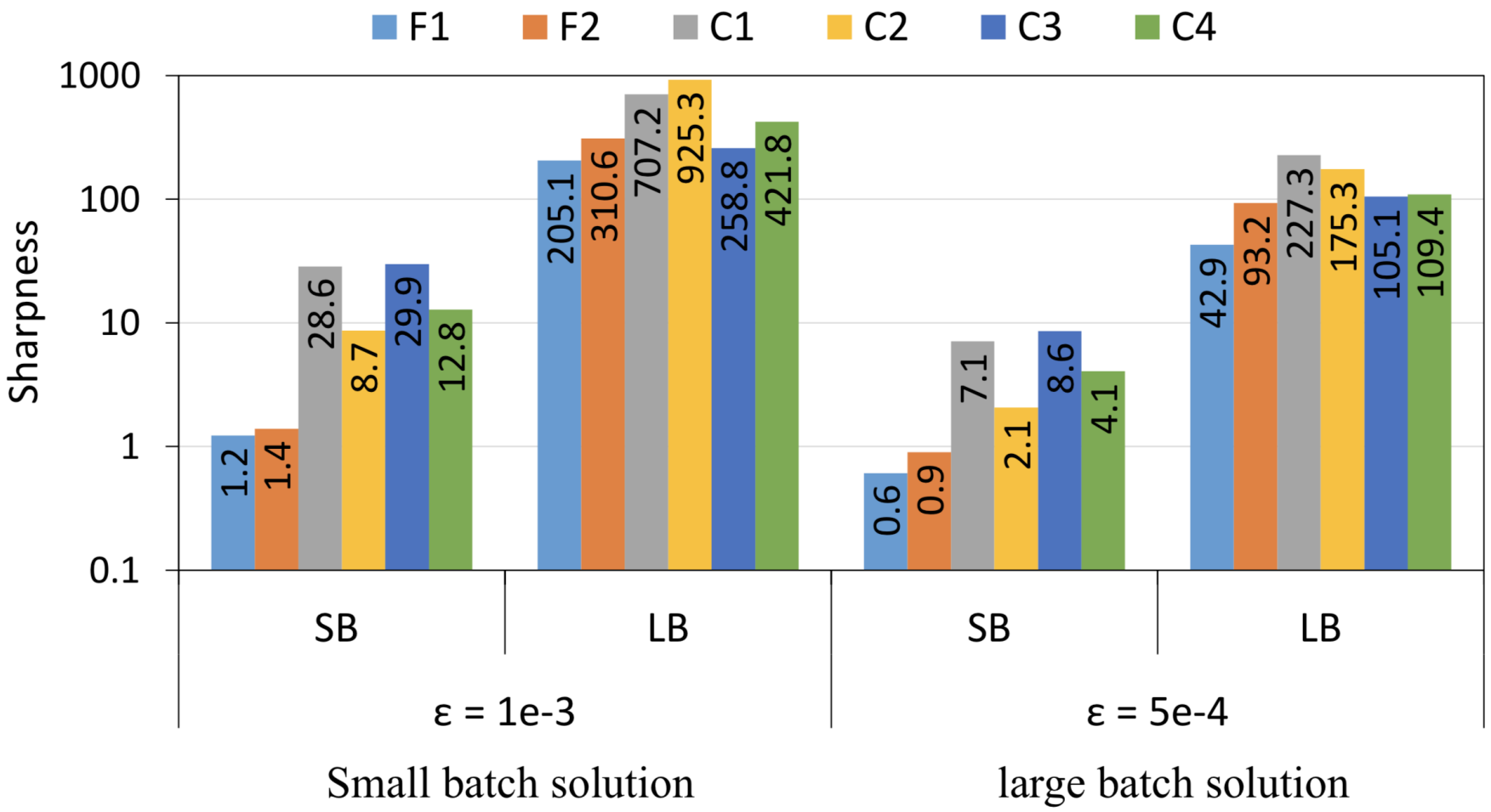
- Flat Minimum: Good Generalization Performance

How to compute the sharpness of a local minimum?

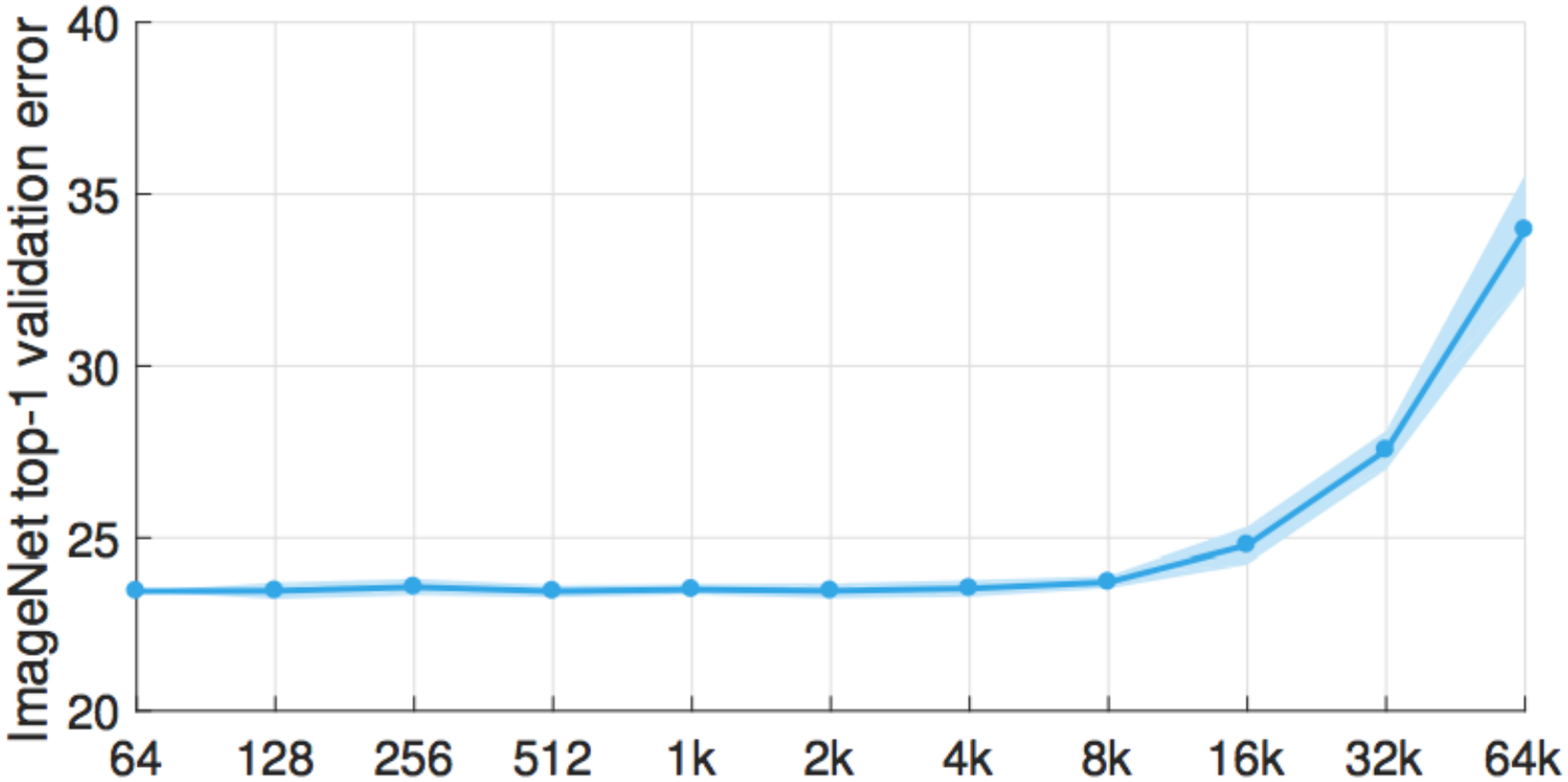
- Given a local minimum w^* and a box B of width ϵ (e.g. 10^{-3}) centered at w^*, we define the sharpness of w^* as

$$\max_{w \in B} \frac{f(w^* + w) - f(w^*)}{1 + f(w^*)}$$

Examples of the sharpness



Optimization Problem[5]



- Figure Credit: Goyal et al. (ImageNet Training with ResNet-50)
- You can keep the accuracy up to $B = 8$K, but it is hard to optimize
- Facebook scales to 8K (able to use 256 NVIDIA P100 GPUs!)

[5] Goyal et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, 2017 (Facebook Report)

Most effective techniques

- Auto-Tune the hyper-parameters like learning rate (η)
- Setting: fixed # epochs, fixed # flops
- Linear Scaling rule[6]
  - if you increase B to kB, then increase η to $k\eta$
    - # iterations reduced by $k\times$, # updates reduced by $k\times$
    - each update should enlarged by $k\times$
- Warmup rule[7]
  - start from a small η, increase η in a few epochs
    - gradients are changing quickly in the beginning
    - avoid the network diverging in the beginning

[6] Alex Krizhevsky, *One weird trick for parallelizing convolutional neural networks*, 2014 (Google Report)

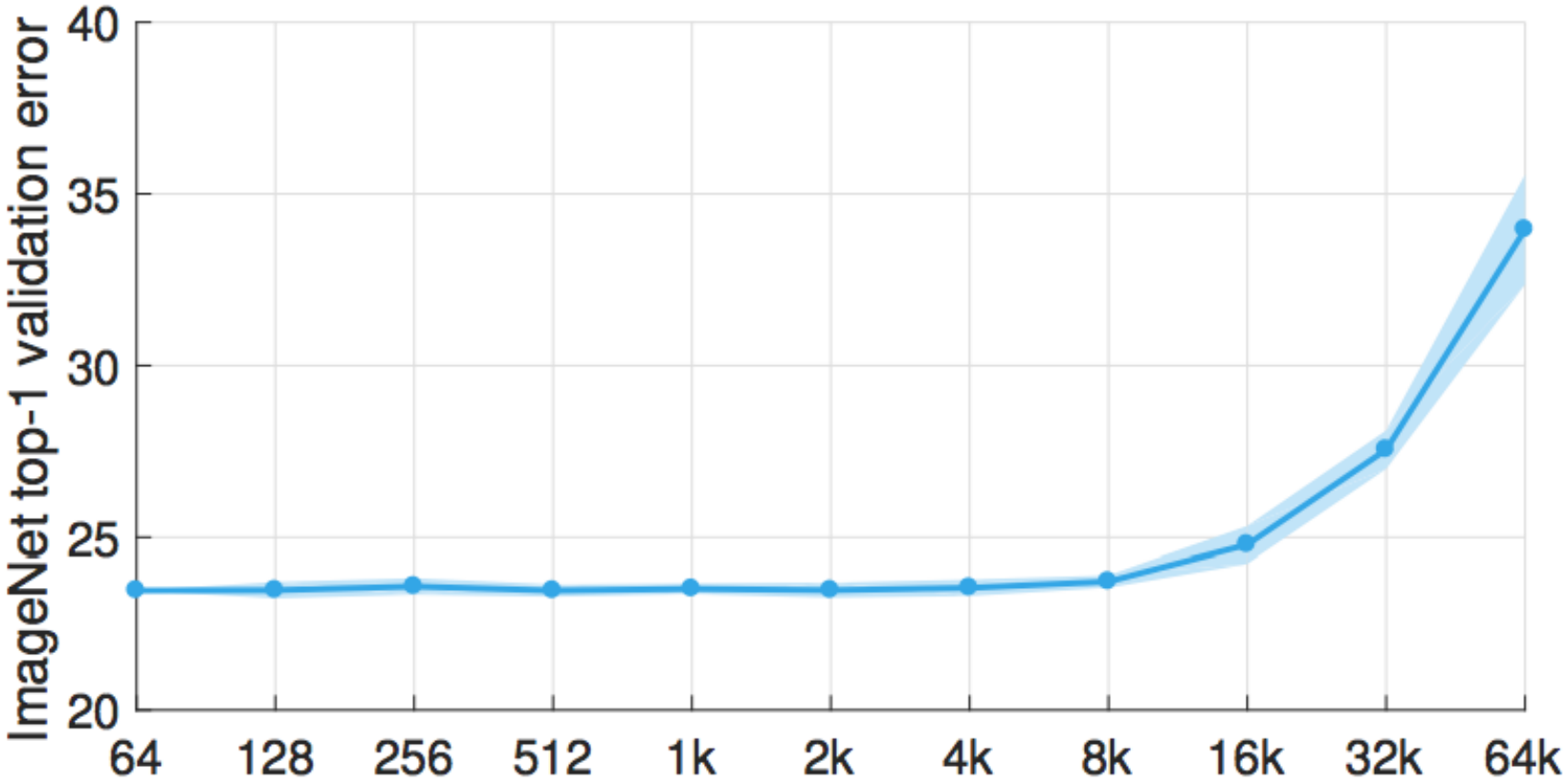
[7] Goyal et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, 2017 (Facebook Report)

Method of Goyal et al. does not work for AlexNet

- Can only scale batch size to 1024 while maintaining accuracy, tried everything:
  - Warmup + Linear Scaling
  - Tune learning rate + Tune momentum + Tune weight decay
  - data shuffle, data scaling, learning rate decay schemes, etc

Batch Size	η	momentum	epochs	test accuracy
512	0.02	0.9	100	58.3%
1024	0.02	0.9	100	58.2%
4096	0.05	0.9	100	**53.1%**
8192	0.03	0.9	100	**44.8%**

Method of Goyal et al. fails on a very large batch size



- Figure Credit: Goyal et al.
- When batch size is higher than 8K, error rate increases significantly
  - Even for ResNet-50

More challenges: BERT[8] (state-of-the-art NLP model)

Optimizer	batch size	warmup steps	LR	last step infomation	F1 score on dev set
AdamW	16K	0.05×31250	0.0001	loss=8.04471, step=28126	diverged
AdamW	16K	0.05×31250	0.0002	loss=7.89673, step=28126	diverged
AdamW	16K	0.05×31250	0.0003	loss=8.35102, step=28126	diverged
AdamW	16K	0.10×31250	0.0001	loss=2.01419, step=31250	86.034
AdamW	16K	0.10×31250	0.0002	loss=1.04689, step=31250	88.540
AdamW	16K	0.10×31250	0.0003	loss=8.05845, step=20000	diverged
AdamW	16K	0.20×31250	0.0001	loss=1.53706, step=31250	85.231
AdamW	16K	0.20×31250	0.0002	loss=1.15500, step=31250	88.110
AdamW	16K	0.20×31250	0.0003	loss=1.48798, step=31250	85.653

- The table data is a part of outputs from Google's auto-tuner
- Baseline: AdamW with a batch size of 512 (F1 score = 90.395)
- SQuAD: a question-answering challenge

[8] Devlin et al., *Bidirectional Encoder Representations from Transformers*, 2018, Google

Outline

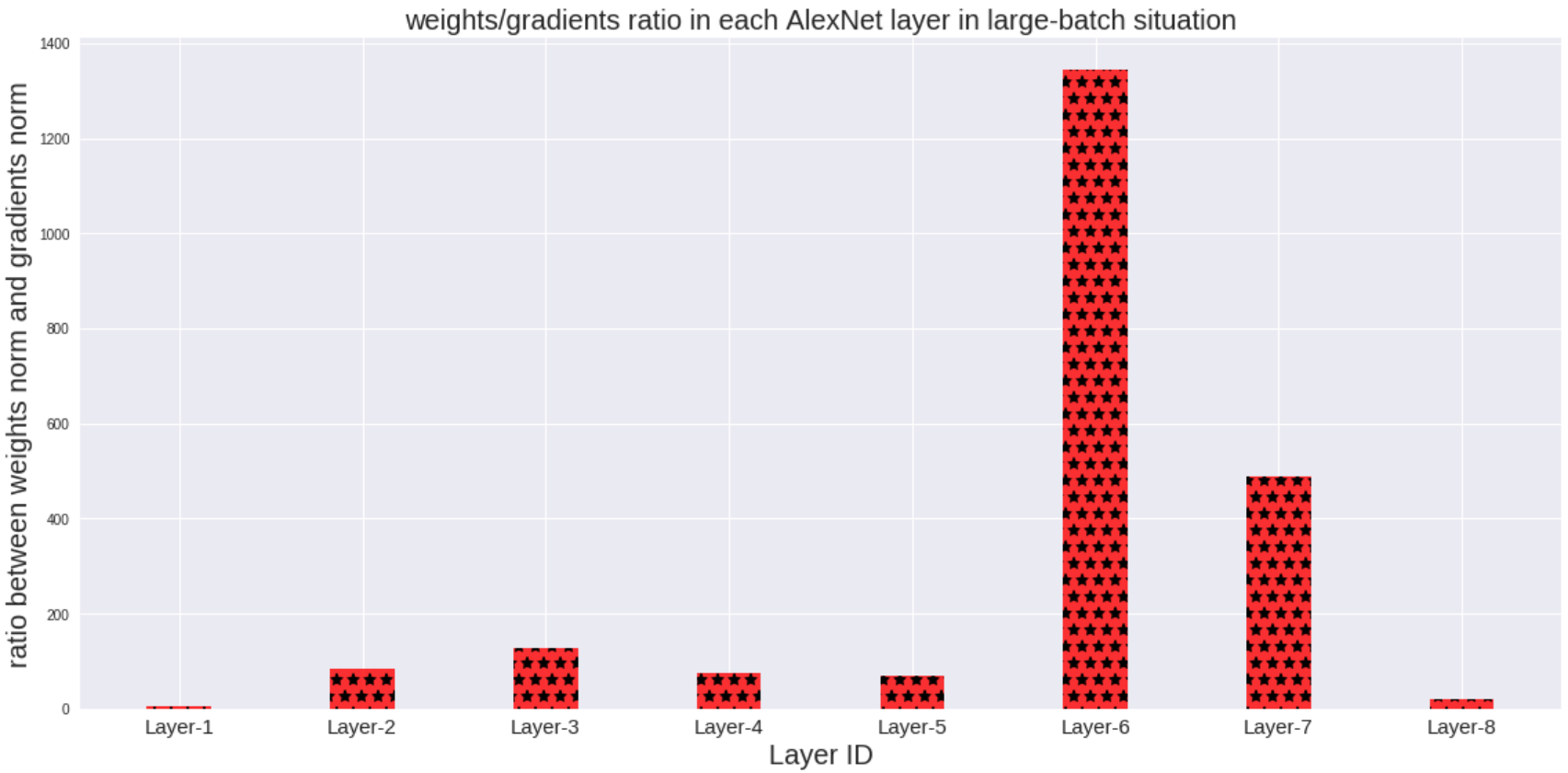
- Why is deep learning slow?
- Why is scaling deep learning difficult?
- **How to scale up deep learning?**
- Experimental Results.
- Our method in real-world applications.

Obstacle: different Weight-Gradient ($||w||/||g||$) Ratios

Layer	1	2	3	4	5	6	7	8
$\|\|w\|\|_2$	0.098	0.16	0.196	8.15	0.16	6.4	6.4	0.316
$\|\|g\|\|_2$	0.017	0.002	0.0015	0.109	0.0002	0.0048	0.013	0.016
$\frac{\|\|w\|\|_2}{\|\|g\|\|_2}$	**5.76**	83.5	127	74.6	69	**1345**	489	19

- L2 norm of layer weights and gradients of AlexNet
  - Batch Size $=$ 4096 at 1st iteration
- Bad choice: the same η for all the layers ($w = w - \eta g$)
  - Layer6's best η leads to divergence for Layer1

Obstacle: different Weight-Gradient ($||w||/||g||$) Ratios



- Suppose all the layers use a learning rate of 100
  - Layer 6: $W = 6.4 - 100 \times 0.005$ (**OK**)
  - Layer 1: $W = 0.098 - 100 \times 0.017$ (**Diverge**)

Trust Ratio

$$r_t^l = \frac{\|w_{t-1}^l\|}{\|\frac{1}{B}\sum_{i=1}^{B}\nabla f(x_t^i, w_{t-1}^l)\| + \lambda\|w_{t-1}^l\|} = \frac{\|w_{t-1}^l\|}{\|g_t^l\| + \lambda\|w_{t-1}^l\|}$$

- Notations
  - l: the layer ID; B: the batch size; t: the iteration ID
  - $g_t^l = \frac{1}{B}\sum_{i=1}^{B}\nabla f(x_t^i, w_{t-1}^l)$; λ: weight decay (e.g. $\lambda = 0.01$)
- Standard global Lipschitz-gradient condition
  - The trust ratio can be interpreted as an estimate of the inverse of the Lipschitz constant of the gradient

Layer-wise Adaptive Rate Scaling (LARS)[9]

- within layer l and iteration t
  - $g_t = \frac{1}{B}\sum_{i=1}^{B}\nabla f(x_t^i, w_{t-1}^l)$ /* compute the gradients */
  - $r_t^l = 1.0$ /* initialize the trust ratio */
  - $r_1 = \phi(\|w_{t-1}^l\|)$ /* compute the norm of the weights */
  - $r_2 = \|g_t^l\| + \lambda\|w_{t-1}^l\|$ /* layer-wise weight decay */
  - if $r_2 > 0$, then $r_t^l = r_1/r_2$ /* compute the trust ratio */
  - $m_t^l = \beta_1 m_{t-1}^l + \eta \times r_t^l \times (g_t^l + \lambda w_{t-1}^l)$ /* update the momentum */
  - $w_t^l = w_{t-1}^l - m_t^l$ /* update the weights */
- Notations
  - B: batch size; η: learning rate; w_t: weights; λ: weight decay; ϕ: scaling function;
  - x_t: data samples; f: loss function; g_t: gradients; β_1: coefficient of momentum;
- Each layer has its own unique learning rate
- The trust ratio is changing between different iterations: adaptive scaling at runtime

[9] You et al., *Scaling SGD Batch Size to 32K for ImageNet Training*, 2017

LAMB (Layer-wise Adaptive Moments for Batch training)[10]

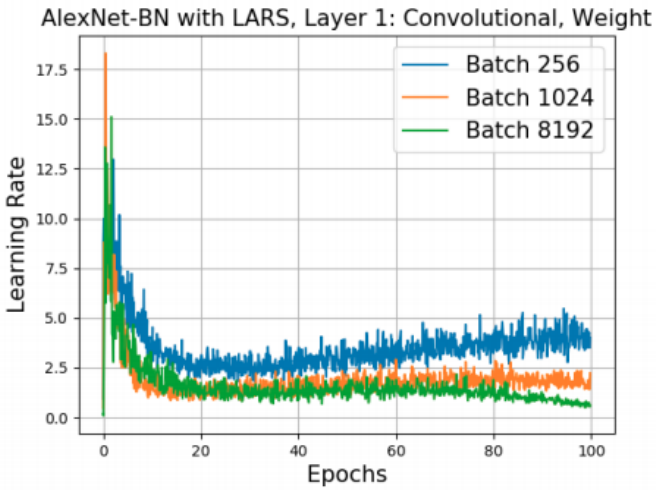
- within layer l and iteration t
  - $g_t = \frac{1}{B}\sum_{i=1}^{B} \nabla f(x_t^i, w_{t-1}^l)$ /* compute the gradients */
  - $m_t^l = \beta_1 m_{t-1}^l + (1-\beta_1) g_t^l$ /* compute the first moment */
  - $v_t^l = \beta_2 v_{t-1}^l + (1-\beta_2) g_t^l \odot g_t^l$ /* compute the second moment */
  - $\hat{m}_t^l = m_t^l / (1-\beta_1^t)$ /* bias correction for first moment */
  - $\hat{v}_t^l = v_t^l / (1-\beta_2^t)$ /* bias correction for second moment */
  - $r_t^l = 1.0$ /* initialize the trust ratio */
  - $r_1 = \phi(\|w_{t-1}^l\|)$ /* compute the norm of the weights */
  - $r_2 = \| \frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l \|$ /* element-wise weight decay */
  - if $r_2 > 0$, then $r_t^l = r_1 / r_2$ /* compute the trust ratio */
  - $w_t^l = w_{t-1}^l - \eta \times r_t^l \times (\frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l)$ /* update the weights */
- Notations
  - B: batch size; η: learning rate; w_t: weights; λ: weight decay; ϕ: scaling function;
  - x_t: data samples; f: loss function; g_t: gradients β_1/β_2: coefficient of first/second moment;
- adaptive element-wise updating + layer-wise correction
  - element-wise weight decay is more accurate (this preserves more information)

[10] You et al., *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*, 2019

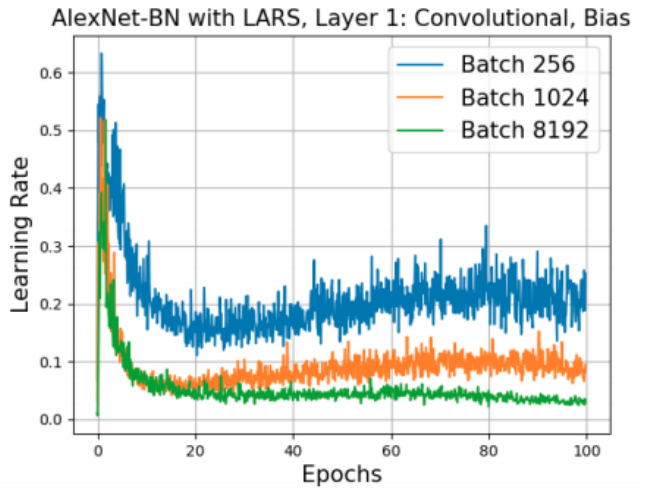
Explanation of LARS/LAMB

- Why LARS/LAMB can solve sharp minimum problem?
- Why LARS/LAMB can speed up training?
- Why LARS/LAMB can converge faster than SGD?

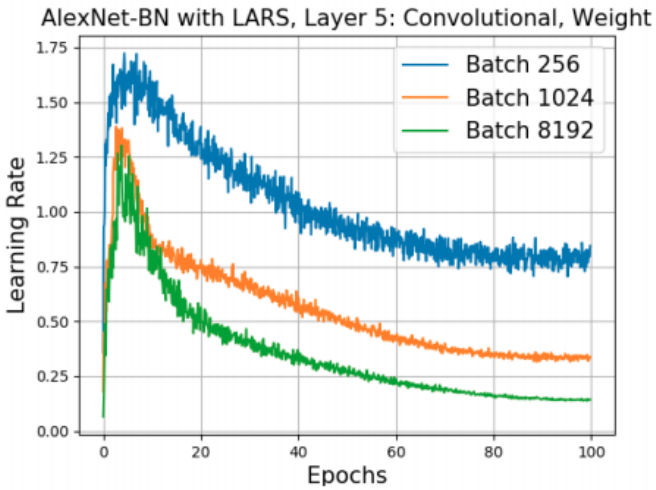
The dynamics of LARS



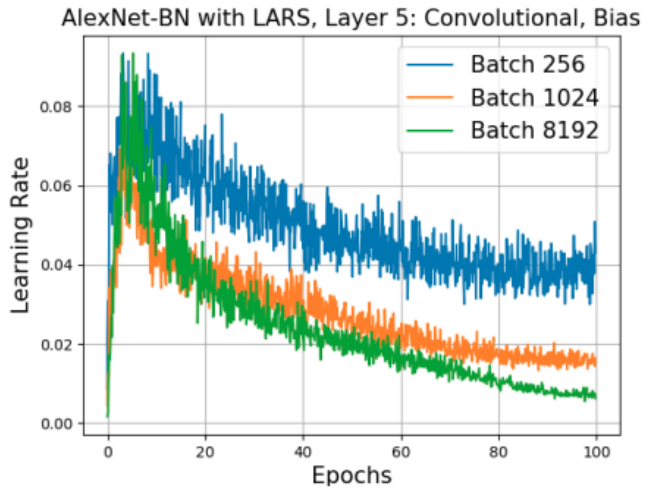
(a) Local LR, conv1-weights



(b) Local LR, conv1-bias



(c) Local LR , conv5-weights



(d) Local LR, conv5-bias

- 1 epoch: statically touches the whole dataset once

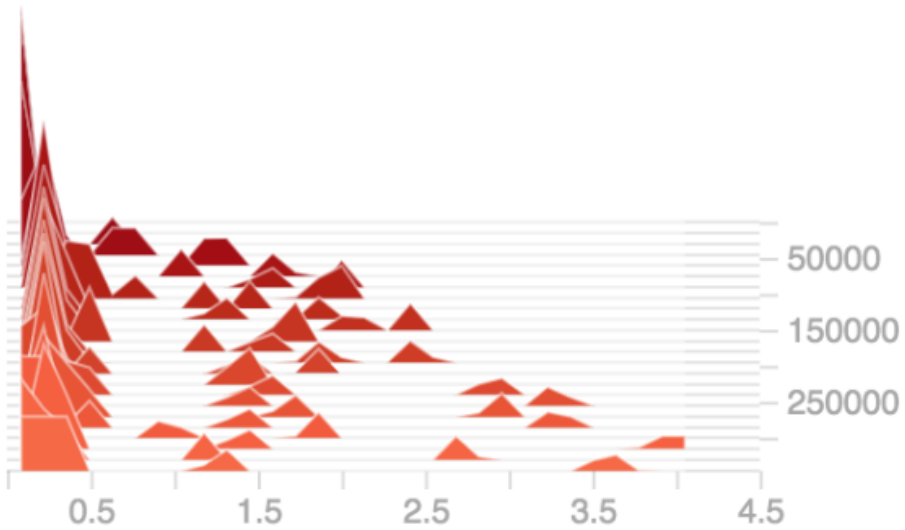
Why LARS/LAMB can solve the sharp-minimum problem?

- Regular-batch uses noisy gradients in the step computation
  - Noise pushes the iterates out of the basin of attraction of sharp minimizers
  - The noise in large-batch is not sufficient to cause ejection from the initial basin leading to convergence to a sharper minimizer
- Adding noises to large-batch training may help
  - However, how to add proper noises?
  - We tried adding the Gaussian noise and significantly tuning the hyperparameters, but it did not help
    - Add noise to activations, i.e. the outputs of each layer.
    - Add noise to weights, i.e. an alternative to the inputs.
    - Add noise to the gradients, i.e. the direction to update weights.
    - Add noise to the outputs, i.e. the labels or target variables.
  - Keskar et al. did similar experiments, but it also did not help[11]
- **The dynamics of LARS/LAMB may act as the proper noise**

[11] https://openreview.net/forum?id=H1oyRlYgg&noteId=H1oyRlYgg

Why LARS/LAMB can speed up training?

- Trust ratio starts significantly below 1
  - It creates a natural warm up period across all layers
- Some of the trust ratios are tiny across all iterations
  - This allows the more stable layers (with larger weight norm) to use a more aggressive learning rate and often converge faster
  - Without trust ratio: some layers may cause instability before others, and the weakest layers limit the overall learning rate that may be applied to the model and increases training time



- Y axis: iteration ID; X axis: trust ratio; Z axis: trust ratio frequency

Convergence Rate[12]

- LARS/LAMB can converge to a stationary point in general nonconvex settings
  - standard global Lipschitz-gradient condition
  - under standard assumptions
  - assuming gradients are bounded
- LARS/LAMB can converge faster than SGD in large-batch setting
  - the convergence rate of SGD depends on the maximum of smoothness across dimension or the maximum of all the Lipschitz constants (Ghadimi & Lan, 2013)
  - the convergence rate of LARS/LAMB only depends on the average of all the Lipschitz constants

[12] You et al., *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*, 2019

Problems and Notations

$$\min_{w \in \mathbb{R}^d} f(w) := \mathbb{E}_{s \sim \mathbb{P}}[\ell(w, s)] + \frac{\lambda}{2}\|w\|^2$$

- ℓ is a smooth (possibly nonconvex) function and $\mathbb{P}$ is a probability distribution on the domain $\mathcal{S} \subset \mathbb{R}^k$. Here, w corresponds to model parameters (i.e. ℓ is the loss function and $\mathbb{P}$ is an unknown data distribution).
- $C = (C_1, \cdots, C_L)^\top$: the L-dimensional vector of Lipschitz constants.
- C_∞ and C_{avg}: $\max_i C_i$ and $\sum_i \frac{C_i}{L}$ respectively.
- gradients are bounded: $[\nabla \ell(w, s)]_j \leq G$ for all $j \in [d]$, $w \in \mathbb{R}^d$, $s \in \mathcal{S}$.
- Other typical assumptions in stochastic first-order methods[13]

[13] Ghadimi & Lan, *Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization*, 2014

Convergence Rate of SGD

Theorem (Ghadimi & Lan 2013)

With large batch $b = T$ and using appropriate learning rate, we have the following for the iterates of :

$$\mathbb{E}\left[\|\nabla f(w_a)\|^2\right] \leq O\left(\frac{(f(w_1) - f(w^*))C_\infty}{T} + \frac{\|\sigma\|^2}{T}\right).$$

where w^* is an optimal solution and w_a is an iterate uniformly randomly chosen from $\{w_1, \cdots, w_T\}$.

Convergence Rate of LARS

Theorem

Let $\eta_t = \eta = \sqrt{\frac{2(f(w_1)-f(w^*))}{\alpha_u^2 \|C\|_1 T}}$ *for all* $t \in [T]$, $b = T$, $\alpha_l \leq \phi(v) \leq \alpha_u$ *for all* $v > 0$ *where* $\alpha_l, \alpha_u > 0$. *Then for* w_t *generated using (Algorithm 1), we have the following bound*

$$\left(\mathbb{E}\left[\frac{1}{\sqrt{L}}\sum_{i=1}^{L}\|\nabla_i f(w_a)\|\right]\right)^2 \leq O\left(\frac{(f(w_1)-f(w^*))C_{avg}}{T} + \frac{\|\sigma\|_1^2}{TL}\right),$$

where w^* *is an optimal solution and* w_a *is an iterate uniformly randomly chosen from* $\{w_1, \cdots, w_T\}$.

Convergence Rate of LAMB

Theorem

Let $\eta_t = \eta = \sqrt{\frac{2(f(w_1)-f(w^*))}{\alpha_u^2 \|C\|_1 T}}$ *for all* $t \in [T]$, $b = T$, $d_i = d/L$ *for all* $i \in [L]$, *and* $\alpha_l \leq \phi(v) \leq \alpha_u$ *for all* $v > 0$ *where* $\alpha_l, \alpha_u > 0$. *Then for* w_t *generated using (Algorithm 2), we have the following bounds:*

1. *When* $\beta_2 = 0$, *we have*

$$\left(\mathbb{E}\left[\frac{1}{\sqrt{d}}\|\nabla f(w_a)\|_1\right]\right)^2 \leq O\left(\frac{(f(w_1) - f(w^*))C_{avg}}{T} + \frac{\|\tilde{\sigma}\|_1^2}{TL}\right),$$

2. *When* $\beta_2 > 0$, *we have*

$$\mathbb{E}[\|\nabla f(w_a)\|^2] \leq O\left(\sqrt{\frac{G^2 d}{L(1-\beta_2)}}\left[\sqrt{\frac{2(f(w_1) - f(w^*))\|C\|_1}{T}} + \frac{\|\tilde{\sigma}\|_1}{\sqrt{T}}\right]\right),$$

where w^* *is an optimal solution and* w_a *is an iterate uniformly randomly chosen from* $\{w_1, \cdots, w_T\}$.

Limitation of Convergence Theory

- These bounds must work for all objective functions in the given class
  - this includes worst-case examples
  - real problems are seldom worst-case
  - thus bounds are often pessimistic or unrealistic
- They do not consider all useful structures in the real objective
  - For example, the condition number ignores:
    - clustered eigenvalues in Hessian
    - low-curvature directions that are completely flat (i.e. not important to optimize)
- Bounds only accurately describe asymptotic performance
  - We often stop before asymptotics "kick-in". Either to prevent overfitting, or because we have a fixed computational budget.
  - Early-stage behaves differently than late-stage (traveling in a roughly consistent direction vs bouncing around local minimum)
  - Provide no global guarantees for non-convex objectives
- "The design/choice of an optimizer should always be informed by practice more than anything else." — James Martens, Deepmind

Summary: why LARS/LAMB work?

- The takeaway
  - high convergence rate
  - a proper noise to get out of sharp minimizers
  - natural per-layer warm-up to stabilize the training
  - avoid network divergence even with a large learning rate
- Intuitive Understanding
  - "Training works well if the magnitudes of parameter updates are about 10^{-2} to 10^{-3} times the magnitude of the parameters". - Geoffrey Hinton
    - We can control this by the normalization of trust ratio
  - When the landscape is smooth, we should learn fast. Otherwise, we should slow down.
    - Trust ratio is the inverse of the Lipschitz constant

Outline

- Why is deep learning slow?
- Why is scaling deep learning difficult?
- How to scale up deep learning?
- **Experimental Results.**
- Our method in real-world applications.

Effects of LARS

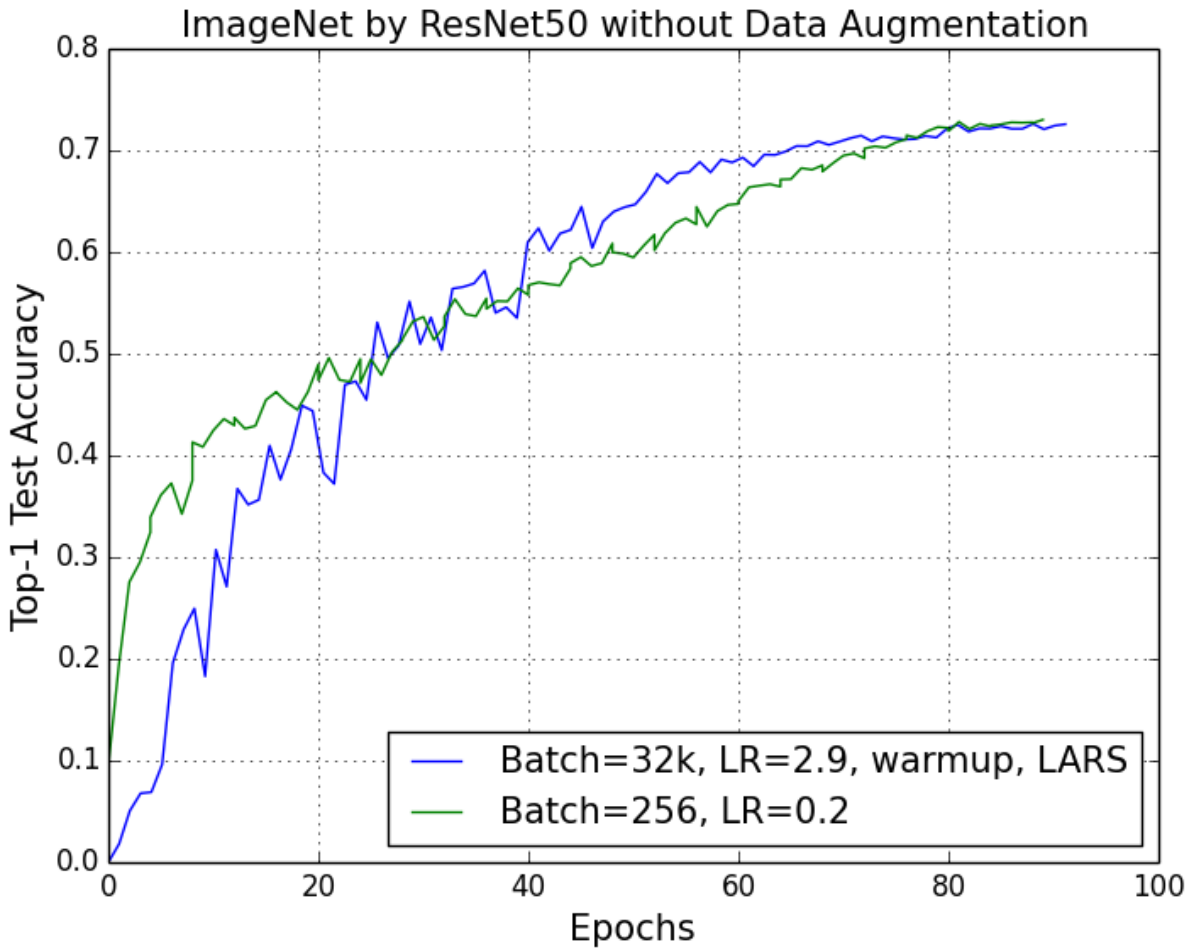
- AlexNet

Batch Size	optimizer	momentum	Epochs	test accuracy
512	Momentum	0.9	100	**58.3%**
4096	LARS	0.9	100	**58.4%**
8192	LARS	0.9	100	**58.3%**

- AlexNet-BN

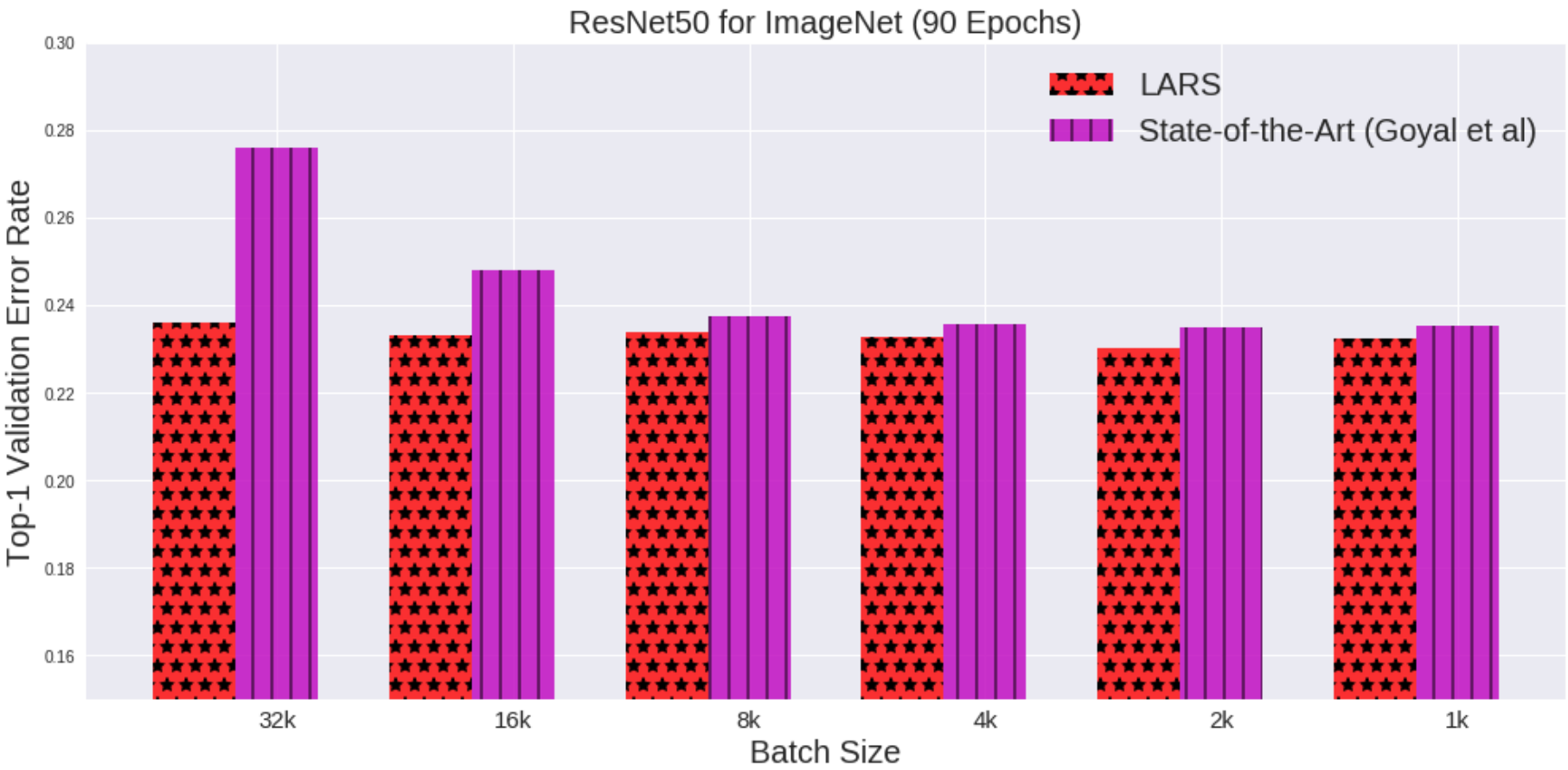
Batch Size	optimizer	momentum	Epochs	test accuracy
512	Momentum	0.9	100	**60.2%**
4096	LARS	0.9	100	**60.4%**
8192	LARS	0.9	100	**60.4%**

Scaling batch size to 32K



- previous approaches only scale to 8K batch size

Compared to State-of-the-Art

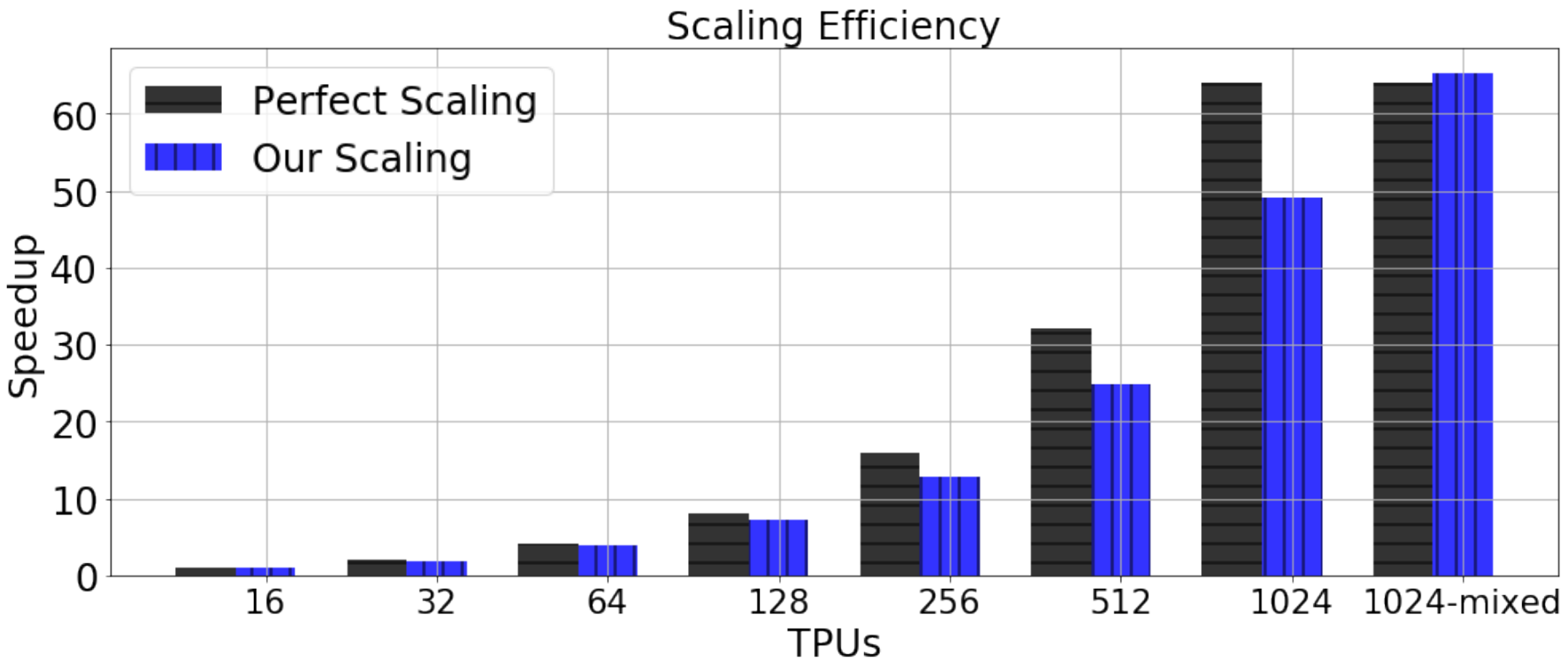


- lower is better

Performance of LAMB on SQuAD by BERT

Optimizer	batch size	steps	F1 score on dev set	TPUs	Time
Baseline	512	1000k	90.395	16	81.4h
LAMB	512	1000k	91.752	16	82.8h
LAMB	1k	500k	91.761	32	43.2h
LAMB	2k	250k	91.946	64	21.4h
LAMB	4k	125k	91.137	128	693.6m
LAMB	8k	62500	91.263	256	390.5m
LAMB	16k	31250	91.345	512	200.0m
LAMB	32k	15625	91.475	1024	101.2m
LAMB	64k/32k	8599	90.584	1024	76.19m

Performance of LAMB on BERT training



Challenge: how to auto-tune when we scale batch size?

- It is time-consuming to tune hyper-parameters when we change B
  - even for an expert or an auto-tuner

No need to re-tune hyper-parameters for scaling batch size

Table 1: SQuAD/BERT by LAMB Optimizer

Batch Size	512	1K	2K	4K	8K	16K	32K
Learning Rate	$\frac{5}{2^{3.0}\times 10^3}$	$\frac{5}{2^{2.5}\times 10^3}$	$\frac{5}{2^{2.0}\times 10^3}$	$\frac{5}{2^{1.5}\times 10^3}$	$\frac{5}{2^{1.0}\times 10^3}$	$\frac{5}{2^{0.5}\times 10^3}$	$\frac{5}{2^{0.0}\times 10^3}$
Warmup Ratio	$\frac{1}{320}$	$\frac{1}{160}$	$\frac{1}{80}$	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$
F1 score	91.752	91.761	91.946	91.137	91.263	91.345	91.475

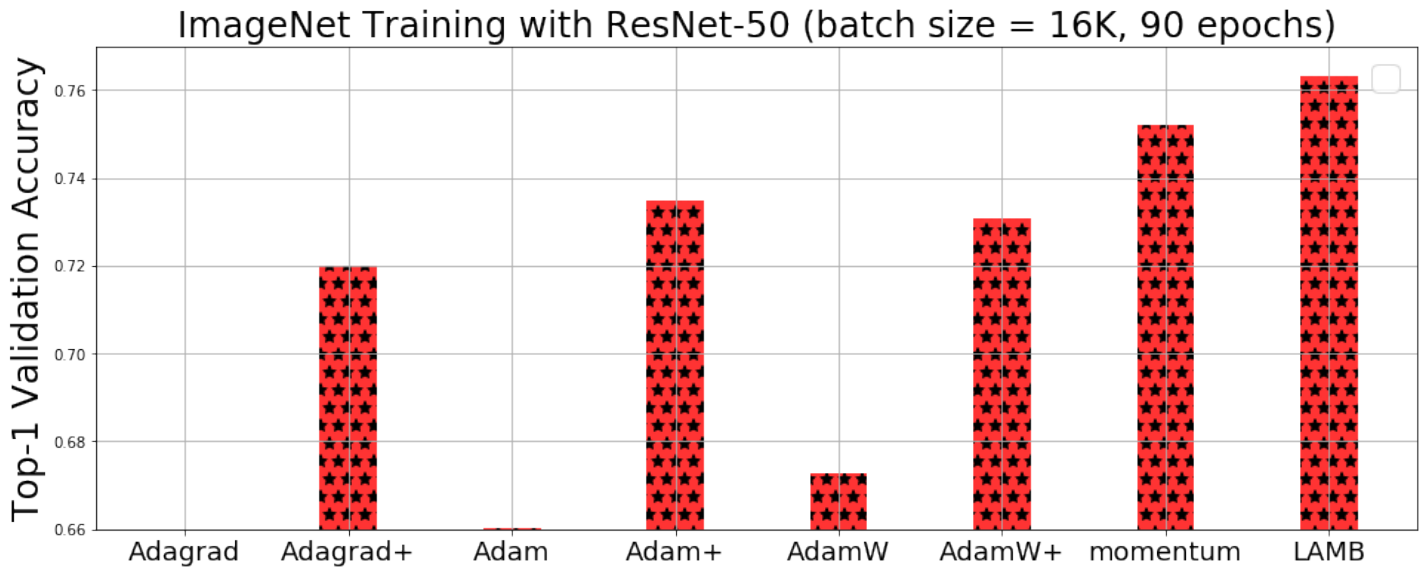
- BERT's baseline achieved a F1 score of 90.395.
- Use default values for other hyper-parameters.
  - Sqrt Learning Rate Scaling
  - Linear Epoch Warmup Scaling

Will LAMB work for ImageNet training?

solver \ application	ImageNet	NLP/RL
Momentum SGD		
Adam/Adagrad		
LAMB	?	

- We want to build a solver to work for both of them
  - Adaptive optimizers (e.g. Adam/AdaGrad) fail on ImageNet
  - Momentum optimizer works poorly on BERT training

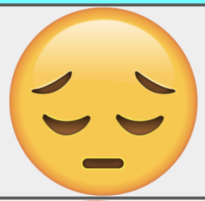
LAMB also works for ImageNet/ResNet-50 training



- The performance of momentum was reported by Goyal et al.
- $+$ means adding the Facebook scheme[14] to an optimizer
- All baselines (except LAMB) were comprehensively tuned by Google's auto tuner

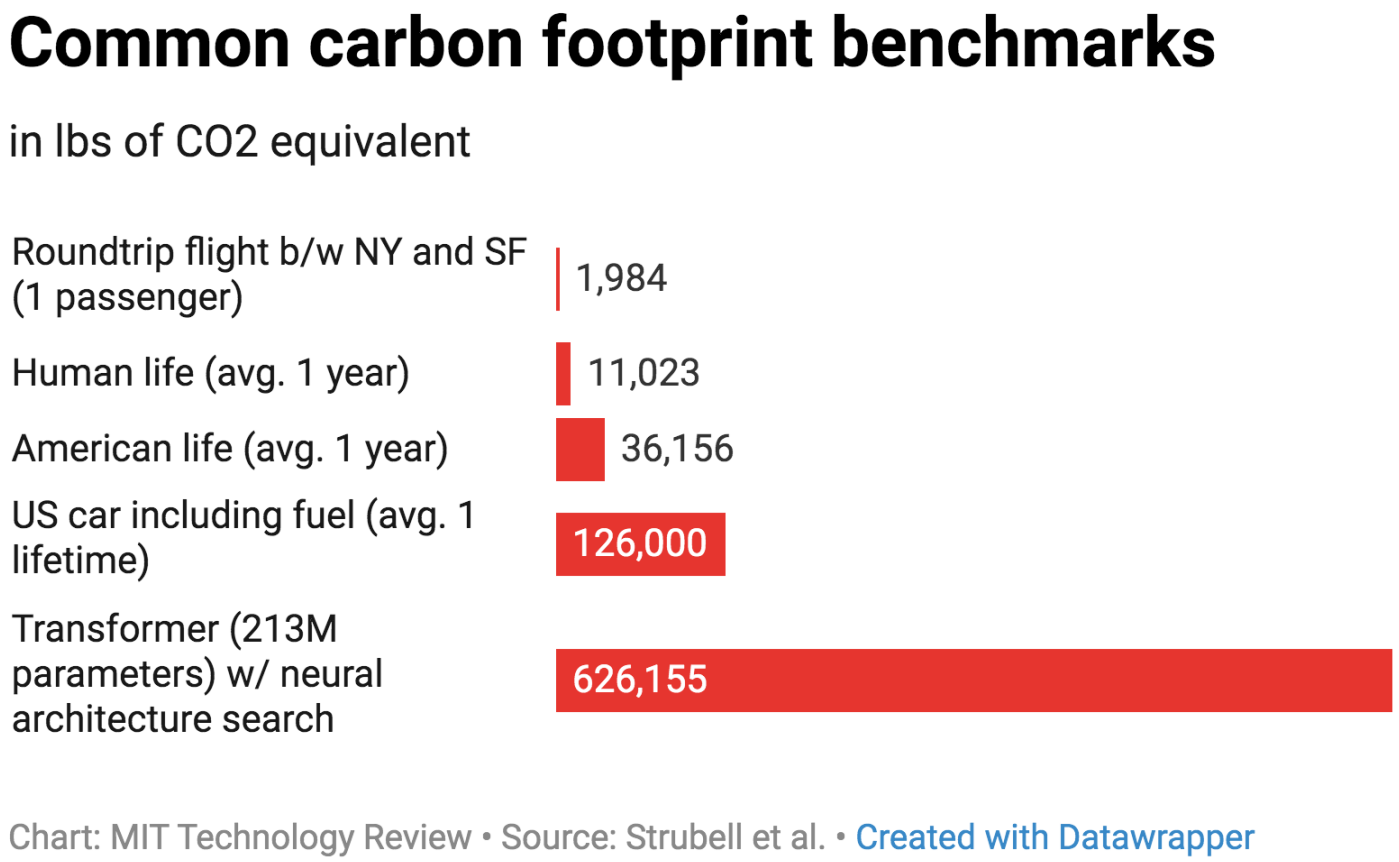
[14] Goyal et al., *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, 2017 (Facebook Report)

LAMB: a general optimizer

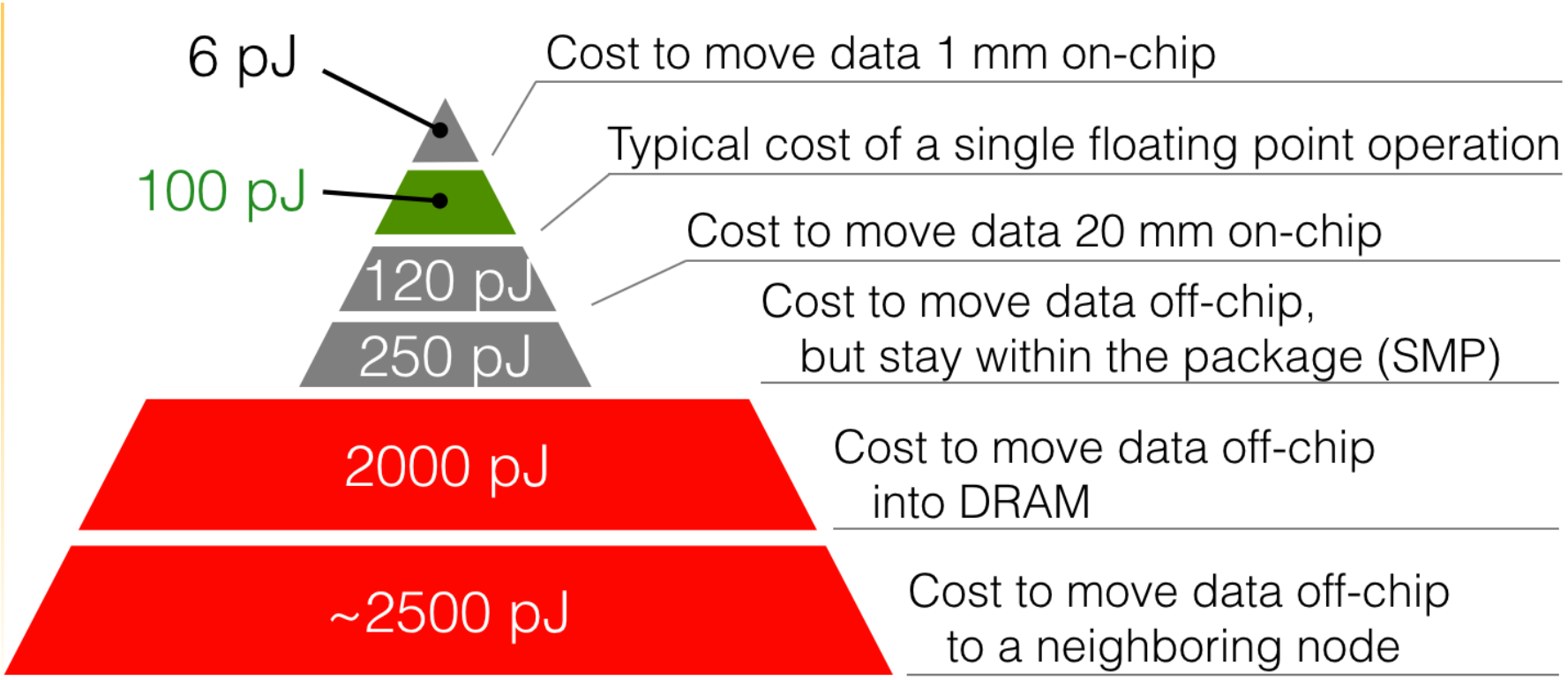
solver \ application	ImageNet	NLP/RL
Momentum SGD		
Adam/Adagrad		
LAMB		

- LAMB is the first optimizer working for both BERT and ImageNet.

An Important Research Direction: Energy-Efficient AI



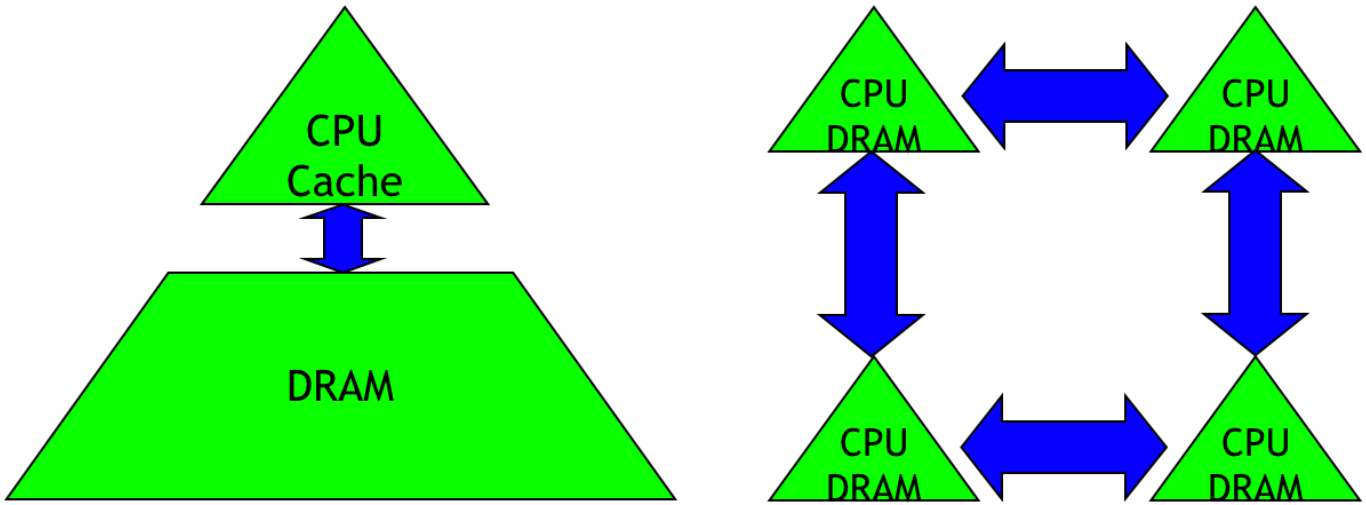
Communication costs more energy (money)



- reduce communication to save energy (money)

Scalable = Communication Efficient

- Algorithms have two costs
  - 1. Computation (floating point operations)
  - 2. Communication
    - different levels of memories (e.g. within a GPU)
    - over a network (e.g. different machines)

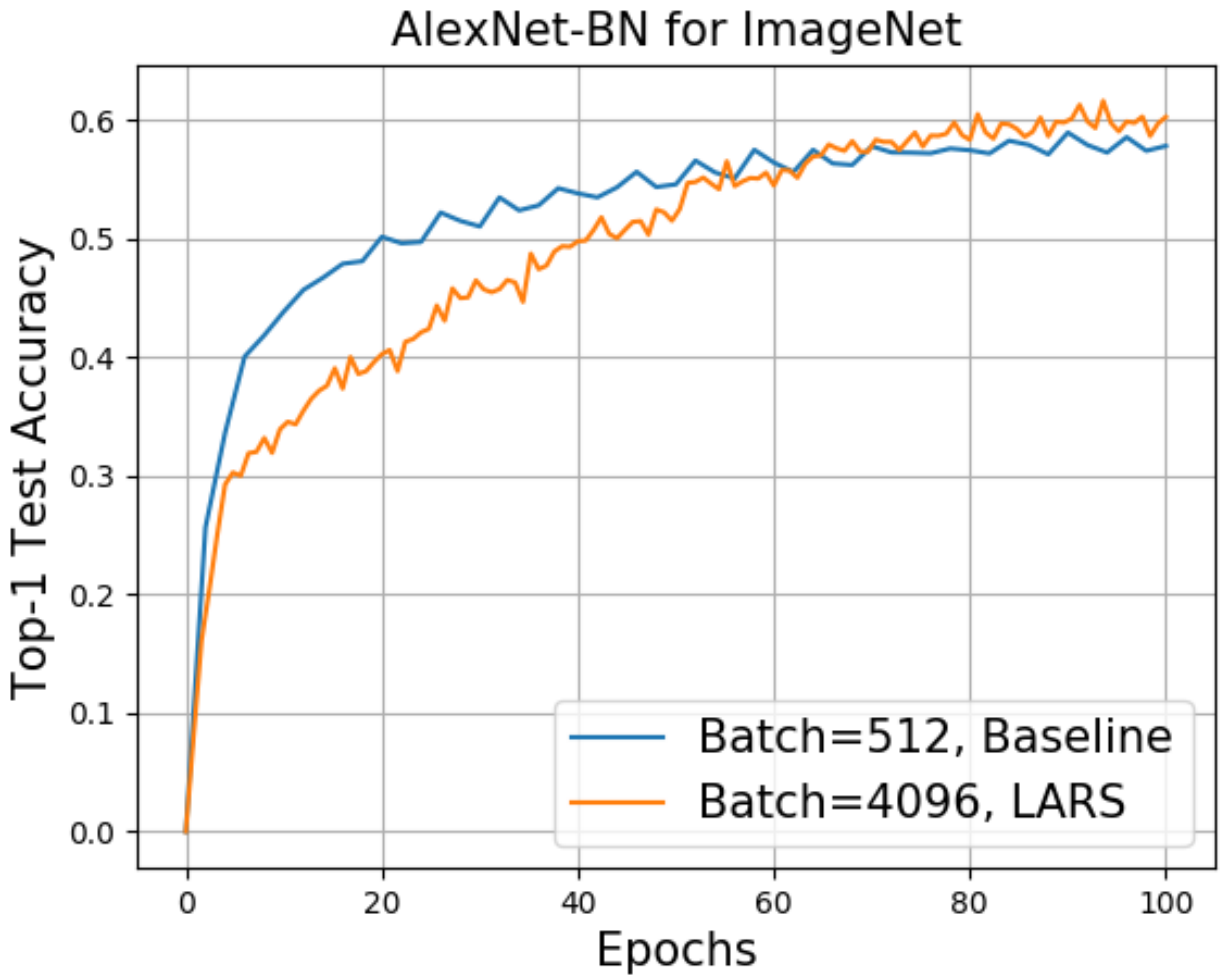


Communication is slower than Computation

- Time-per-flop (γ) $\ll$ 1/ bandwidth (β) $\ll$ latency (α)
  - $\gamma = 0.9 \times 10^{-13}$s for NVIDIA P100 GPUs

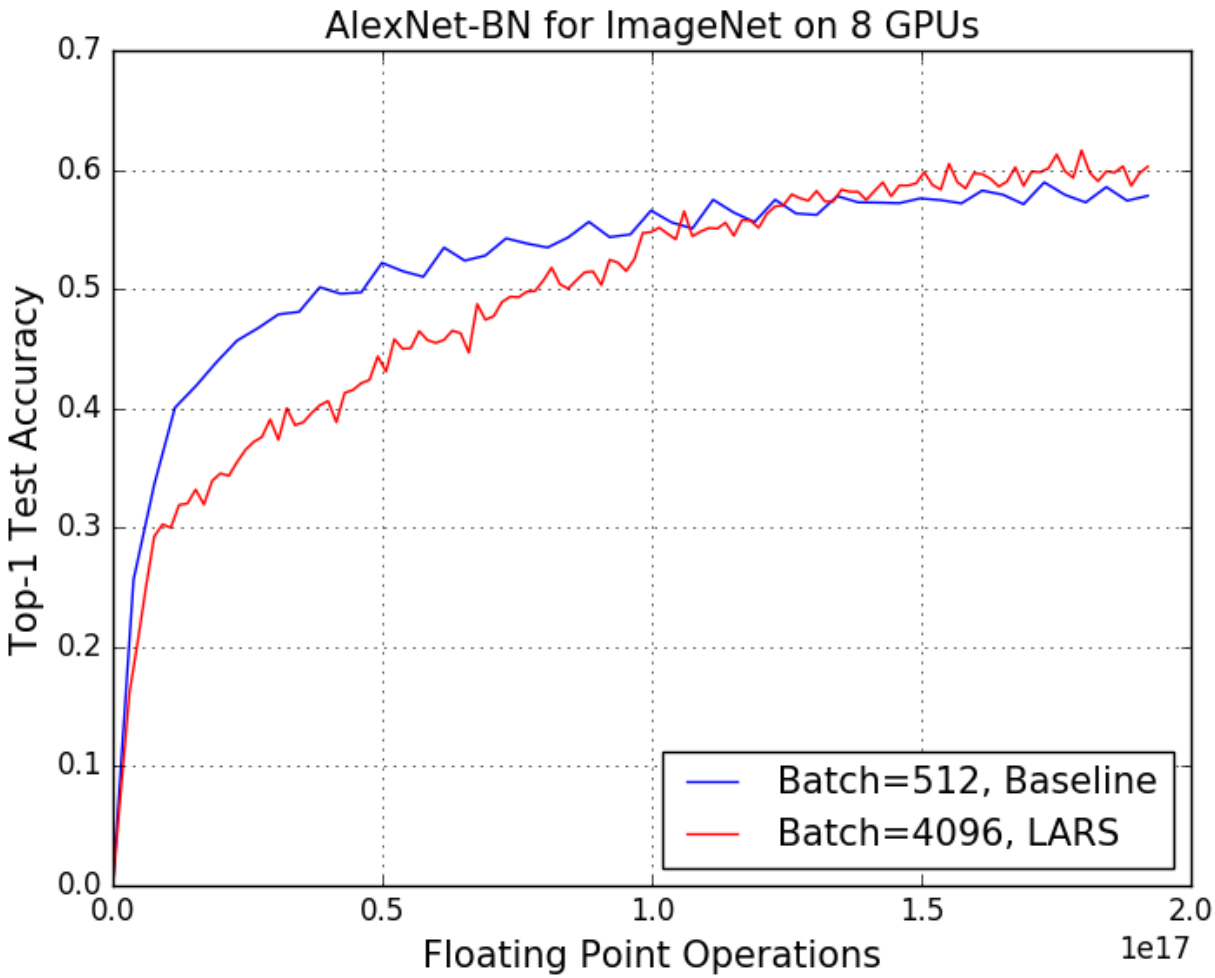
Network	α (latency)	β (1/bandwidth)
Mellanox 56Gb/s FDR IB	0.7×10^{-6}s	0.2×10^{-9}s
Intel 40Gb/s QDR IB	1.2×10^{-6}s	0.3×10^{-9}s
Intel 10GbE NetEffect NE020	7.2×10^{-6}s	0.9×10^{-9}s

Same or better result in the same epochs



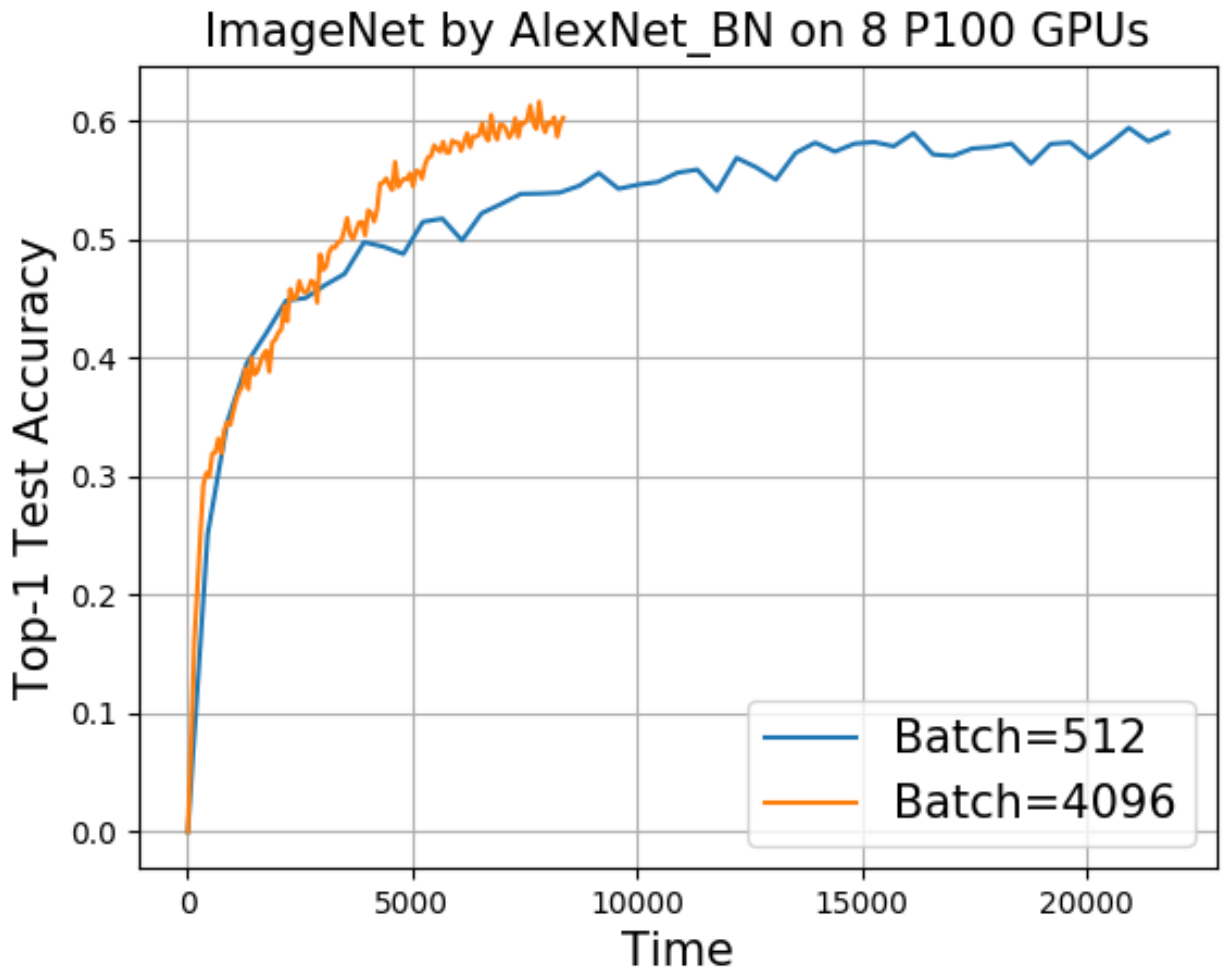
- Touch the whole dataset 100 times
  - The dataset has 1,280,000 pictures

Fixed # epochs = Fixed # floating point operations



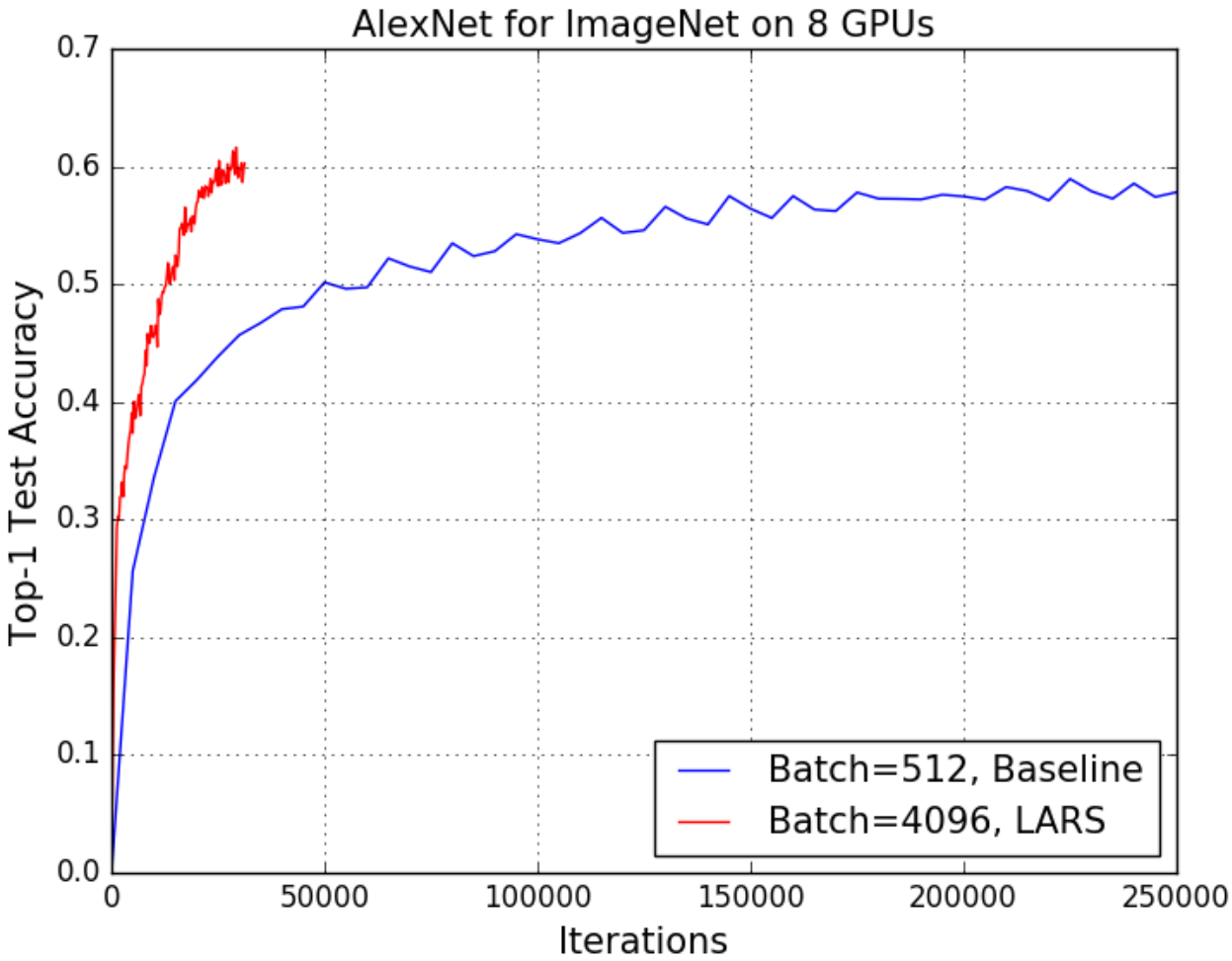
- The total number of operations is 100 × 1.28 Million × 1.5 Billion
  - 100 epochs for using AlexNet to process ImageNet-1k dataset

Larger batch is faster



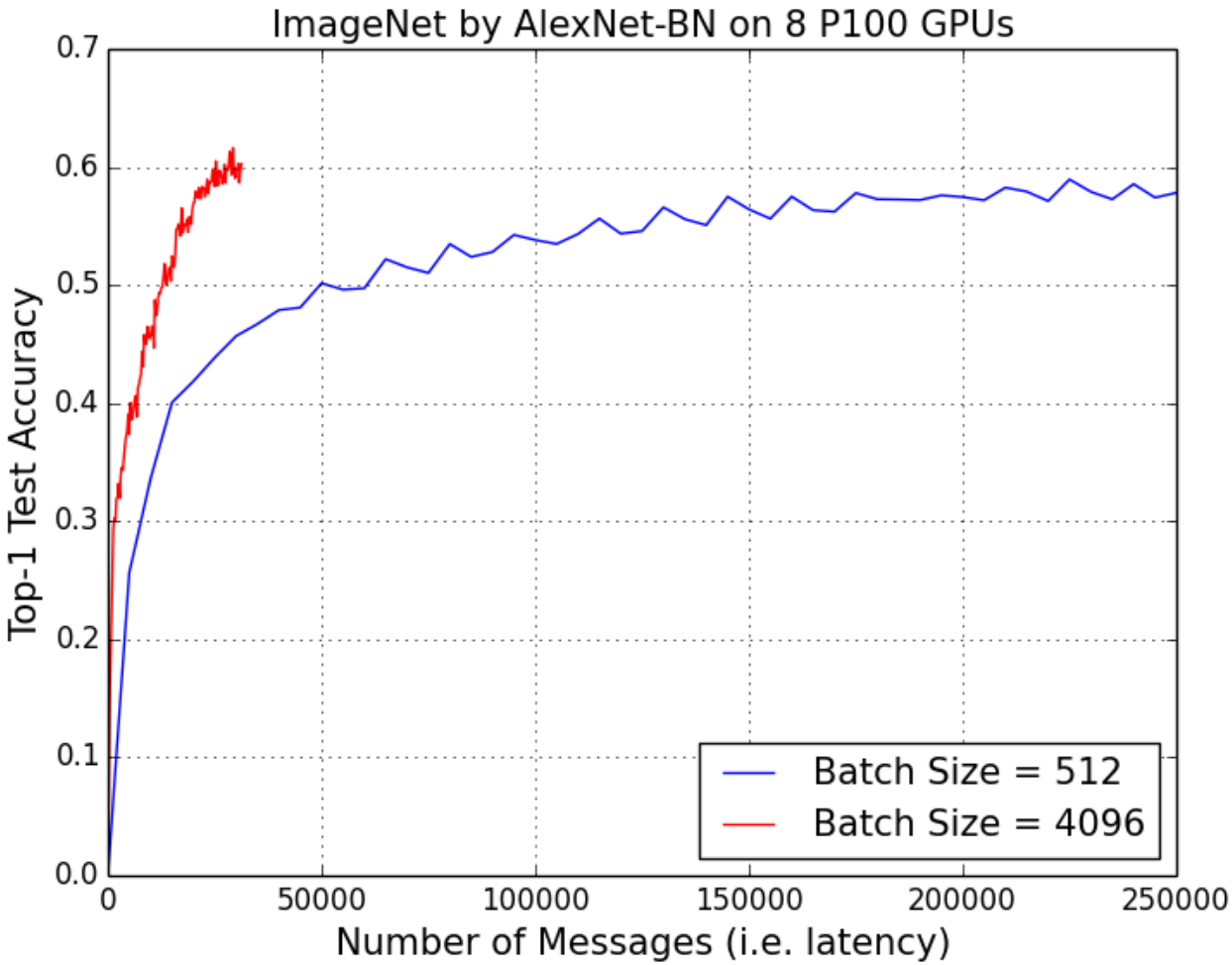
- GPU's performance is higher (less memory-level communication)
- Less GPU-GPU communication (latency and bandwidth)

Less GPU-GPU communication



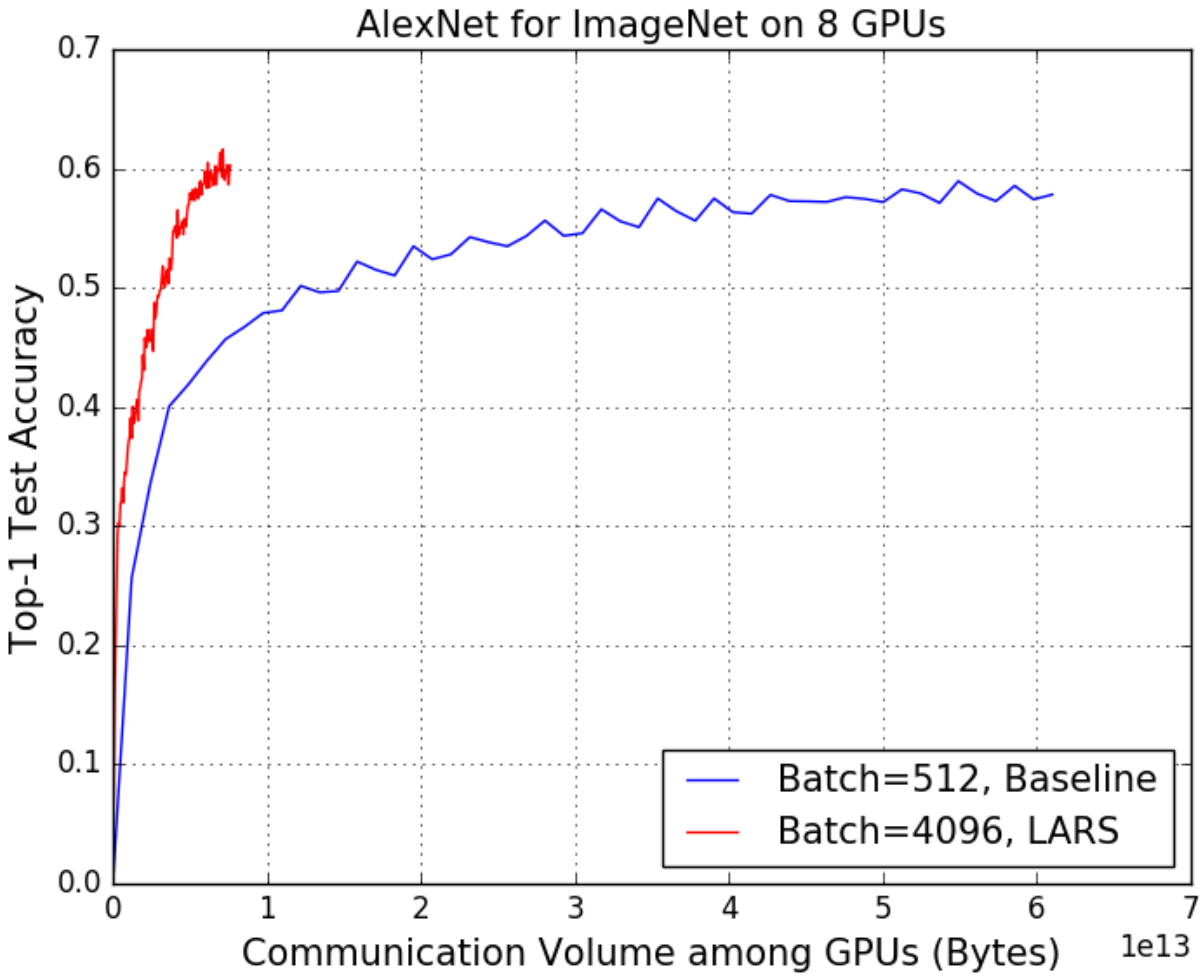
- Latency $= \Theta(\text{iterations})$
- Bandwidth $= \Theta(|W| \times \text{iterations})$

Less GPU-GPU communication



- Latency $= \Theta(\text{iterations})$
- Bandwidth $= \Theta(|W| \times \text{iterations})$

Less GPU-GPU communication



- Latency $= \Theta(\text{iterations})$
- Bandwidth $= \Theta(|W| \times \text{iterations})$

Large Batch: communication-avoiding algorithm

- large batch does not change the number of floating point operations
  - Fixed number of epochs
- large batch can reduce the communication overhead
  - $W_i = W_i - \eta/P * \sum_{i=1}^{P} \nabla W_i$ (∇W_i is on i-th machine)
  - single-iteration communication volume is proportional to the model size
  - large-batch reduced the number of iterations
- **large batch can improve the algorithm's scaling efficiency**
  - move less data
  - reduce the latency overhead

Table 2: ImageNet training with ResNet-50: communication over network.

Batch	Operations	# Messages	Data Moved	Comp/Comm
32	10^{18}	3.6 million	374TB	2673
32768	10^{18}	3686	374GB	2.7M

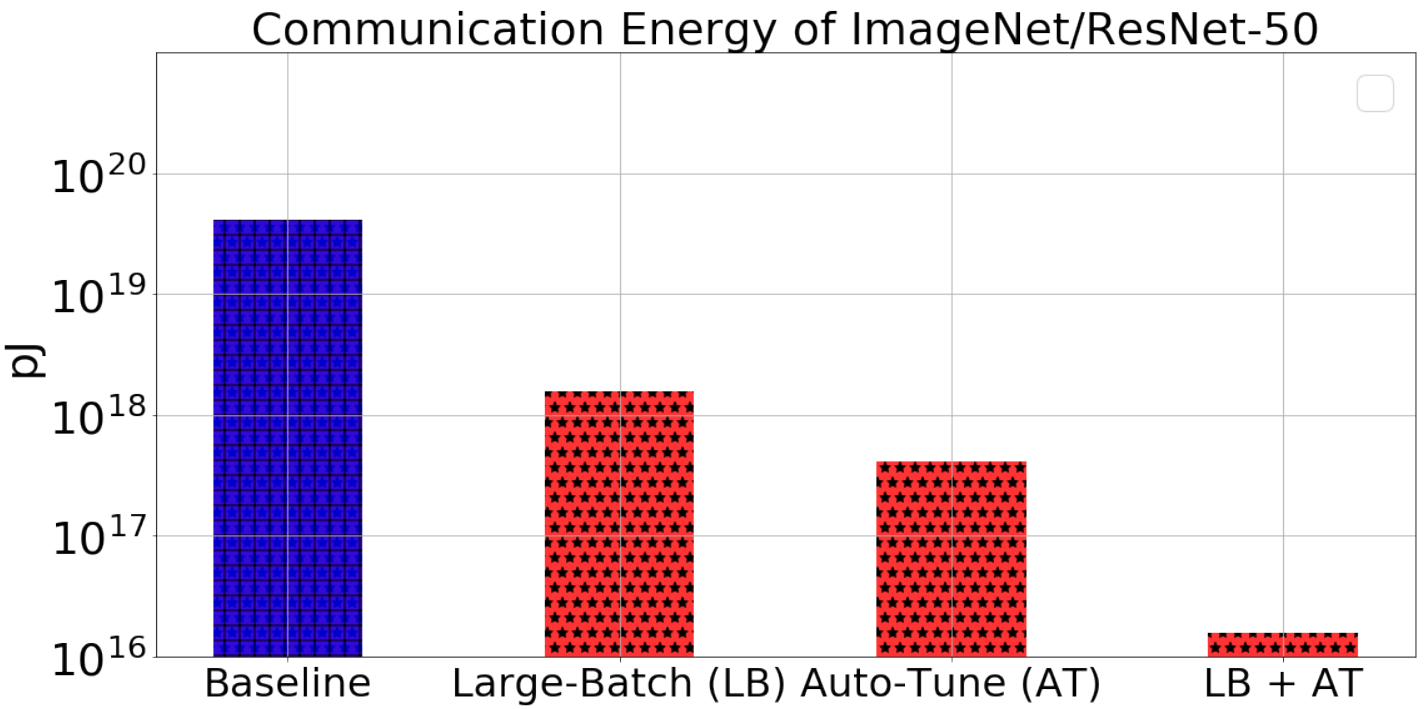
Comp/Commu Ratio and Large Batch

Table 3: Scaling Ratio for AlexNet and ResNet50.

Model	communication # parameters	computation # flops per image	flops/model ratio	comp/comm ratio
AlexNet	# 61 million	# 1.5 billion	24.6	24.6 × B
ResNet50	# 25 million	# 7.7 billion	308	308 × B

- Different models have different scaling efficiency
  - ResNet-50: 95% weak scaling efficiency
- Large batch can increase comp/comm ratio
  - very important to scaling efficiency

Energy-Efficient Communication



- B of the baseline: 256
- B of the large-batch: 32K
- the baseline tunes the hyper-parameters 100 times

Efficient System Implementations

- Mixed batch training[15]
  - improve scaling efficiency from 75.4% to 100.2% for BERT
- Mixed precision training (inspired by Micikevicius et al.)
  - reduce the memory overhead by $1.2\times$, get $1.16\times$ speedup
- Distributed Batch Normalization (inspired by Wu et al.)
  - reduce communication overhead
  - an effective batch size of 64 is OK for BN in ImageNet Training
- Input Pipeline and Data Process (inspired by Kumar et al.)
  - get $1.3\times$ speedup
- Efficient Global Summation (inspired by Ying et al.)
  - make communication time under 3% of the training time

[15] You et al., *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*, 2019

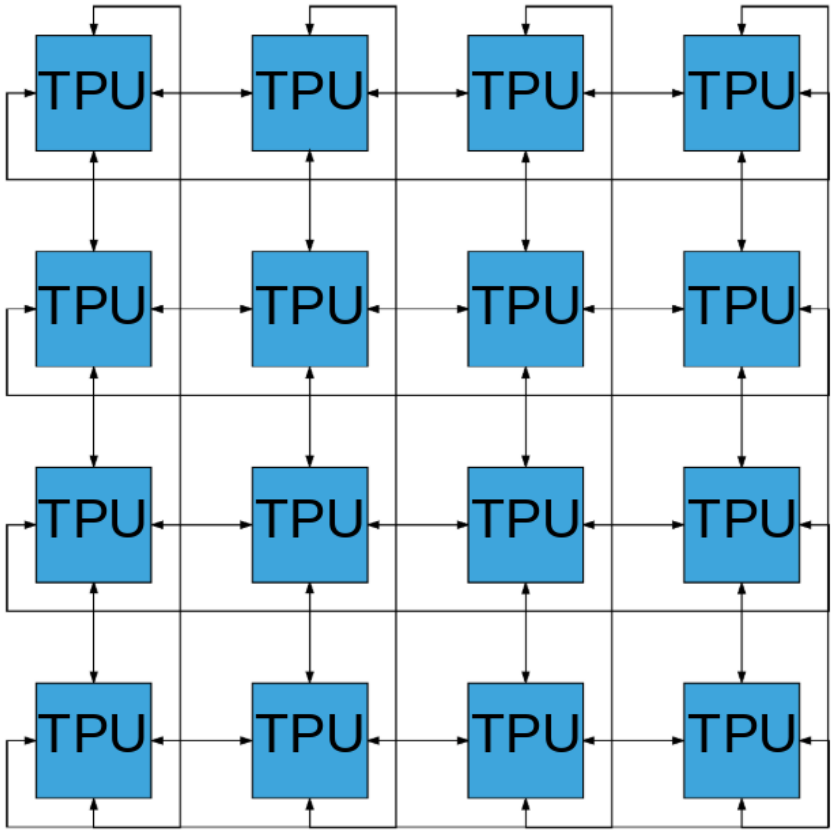
Mixed batch training[16]

- BERT training has two stages:
  - the first 9/10 of the total epochs use a sequence length of 128
  - the last 1/10 of the total epochs use a sequence length of 512
- Memory limits for BERT training on a TPUv3 Pod:
  - the maximum batch size for stage 1 is 128K
  - the maximum batch size for state 2 is 32K
- Challenge:
  - Increasing the batch size can warm-up and stabilize the optimization process, but decreasing the batch size brings chaos to the optimization process and can cause divergence (Smith et al., 2017)
- Solution:
  - Because we switched to a different optimization problem, it is necessary to warm up the optimization again.
  - Instead of decaying the learning rate at the second stage, we ramp up the learning rate from zero again in the second stage (re-warm-up).
  - As the first stage, we decay the learning rate after the re-warm-up.

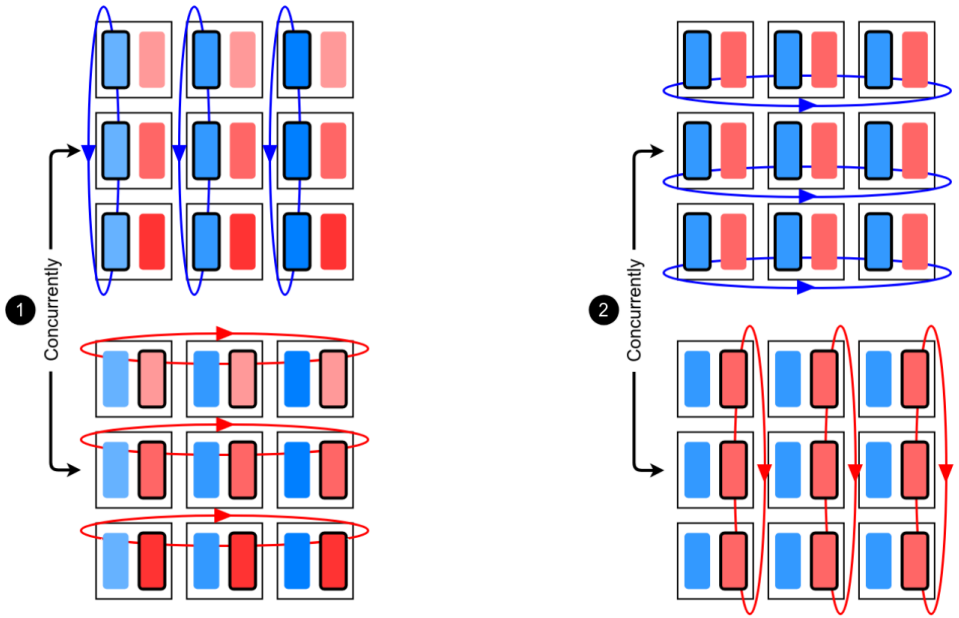
[16] You et al., *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*, 2019

Efficient Global Summation

- All-reduce algorithms
  - 1D ring: Baidu and Uber Horovod
  - halving/doubling: Facebook
- TPU Pod: 2D torus connection
  - 1D ring: limited by the latency of pushing packets in a Hamiltonian circuit across all the nodes



2D ring implementation (inspired by Ying et al.)

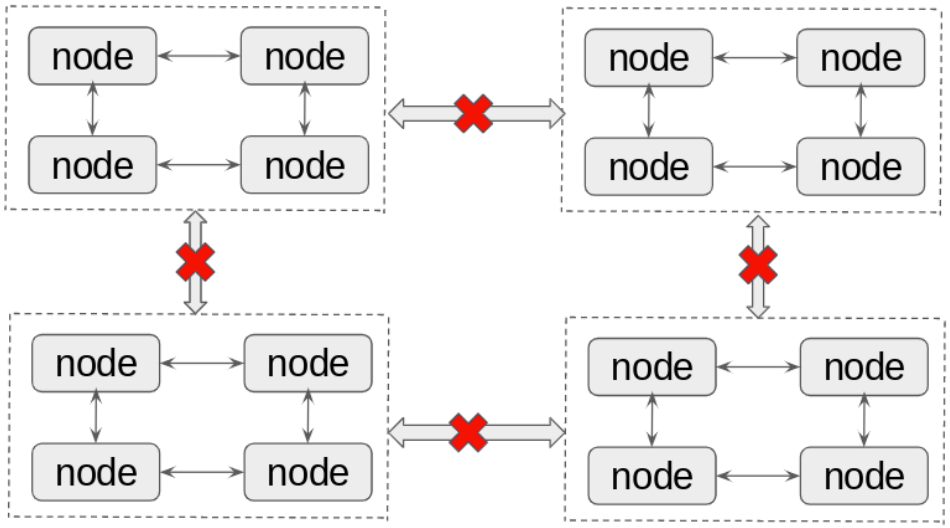


- In the first phase (left), the first half of the tensors (blue) are summed along the vertical dimension while the second half of the tensors (red) are summed concurrently along the horizontal dimension.
- In the second phase (right), the dimensions are flipped, which completes the all-reduce for both halves.
- Reduces the span of gradient summation to O(N) versus O(N^2) network hops in 1D algorithm (N is the size of torus dimension)

Mixed precision training (inspired by Micikevicius et al.)

- Majority of the computational and memory access overheads
  - convolutional operations
- Convolutions are executed using bfloat16
- Input images and intermediate activations are stored in bfloat16
  - higher training throughput
- All non-convolutional operations (e.g. batch normalization, loss computation, gradient summation) use 32-bit floating point numbers
  - maintain comparable accuracy with 32-bit floating point networks

Distributed Batch Normalization (inspired by Wu et al.)



- Batch size of effective BN is too small: can't maintain accuracy
- Batch size of effective BN is too large: high communication overhead
- Our implementation
  - each node computes local mean and variance
  - each group computes distributed mean and variance
  - each group normalizes tensors with distributed mean and variance
  - different groups don't share BN information

Input Pipeline and Data Process (inspired by Kumar et al.)

- Dataset sharding and caching
  - ideally, a dataset should be read only once and cached in memory for future access
  - practically, real-world datasets are often too large to be stored in host memory
  - we have to shard the dataset between workers and enable more efficient access patterns to the data
- Prefetch to pipeline input and compute
  - while TPUs are training on the current batch, the next batch can be processed on CPUs input pipeline concurrently
- Pipeline for load-balanced pre-processing
  - CPUs pre-process images of different sizes to the same size
    - the computation load on each TPU is the same in each iteration
  - variable image sizes caused a load-imbalance on CPUs, resulting in some of TPUs waiting
  - the pipeline of CPUs prefetches for several batches can enable sufficient headroom for TPUs when CPUs are processing large images

Outline

- Why is deep learning slow?
- Why is scaling deep learning difficult?
- How to scale up deep learning?
- Experimental Results.
- **Our method in real-world applications.**

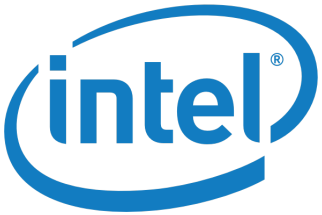
All state-of-the-art systems are using LARS

Table 4: ImageNet/ResNet-50 Training Speed Records.

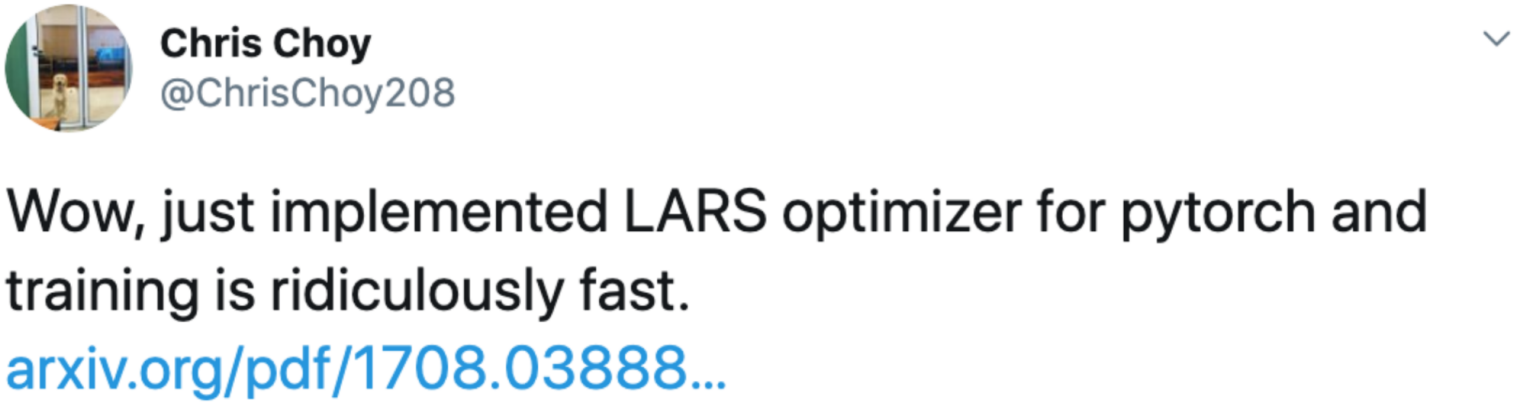
Teams	Date	Accuracy	Time	Optimizer
Microsoft (He et al.)	12/10/2015	75.3%	29h	Momentum SGD
Facebook (Goyal et al.)	06/08/2017	76.3%	65m	Momentum SGD
Berkeley (You et al.)	11/02/2017	75.3%	48m	LARS (You et al.)
Berkeley (You et al.)	11/07/2017	75.3%	31m	LARS (You et al.)
PFN (Akiba et al.)	11/12/2017	74.9%	15m	RMSprop + SGD
Berkeley (You et al.)	12/07/2017	74.9%	14m	LARS (You et al.)
Tencent (Jia et al.)	07/30/2018	75.8%	6.6m	LARS (You et al.)
Sony (Mikami et al.)	11/14/2018	75.0%	3.7m	LARS (You et al.)
Google (Ying et al.)	11/16/2018	76.3%	2.2m	LARS (You et al.)
Fujitsu (Yamazaki et al.)	03/29/2019	75.1%	1.25m	LARS (You et al.)
Google (Kumar et al.)	07/10/2019	75.9%	67.1s	LARS (You et al.)
Google (Kumar et al.)	11/07/2020	75.9%	28.8s	LARS (You et al.)

LARS is Widely Used in Real-World Applications

- ImageNet Training in Minutes
  - Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, Kurt Keutzer
  - Best Paper Award of ICPP 2018 (1 out of 313: 0.3%)
  - The most cited HPC conference paper (ICS, ICPP, IPDPS, PPOPP, SC, etc.) published since 2018 (according to academic.microsoft.com)
- Being used by many leading AI companies



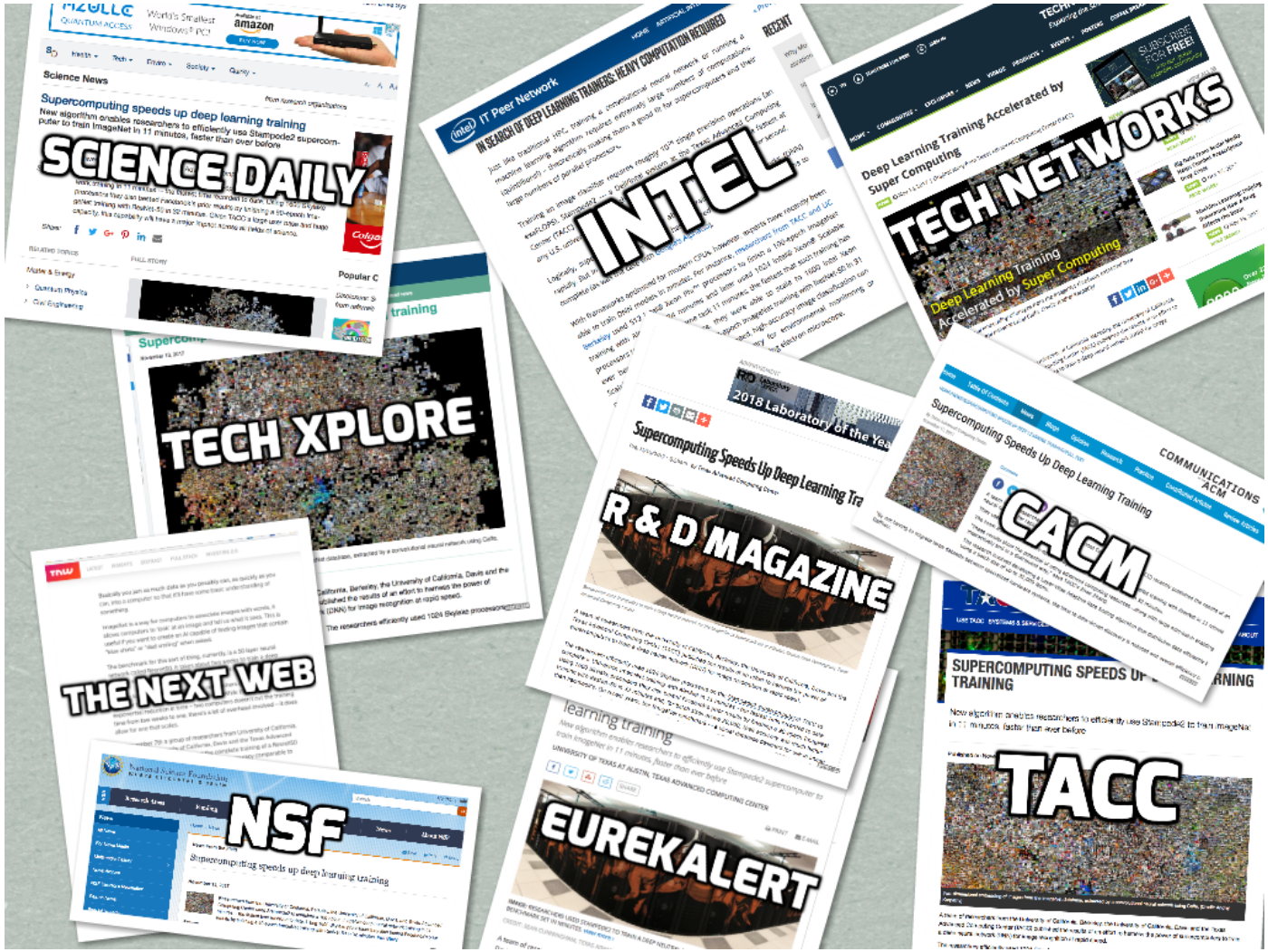
Application of LARS



- LARS helped Geoffrey Hinton's team to achieve state-of-the-art ImageNet classification results by **self-supervised learning**[17]
- LARS helped Google to achieve state-of-the-art ImageNet classification results **(reaches 90% top-1 accuracy)**[18]
- LARS was used in state-of-the-art frameworks like SimCLR (Google), BYOL (Deepmind), and SEER (Facebook)

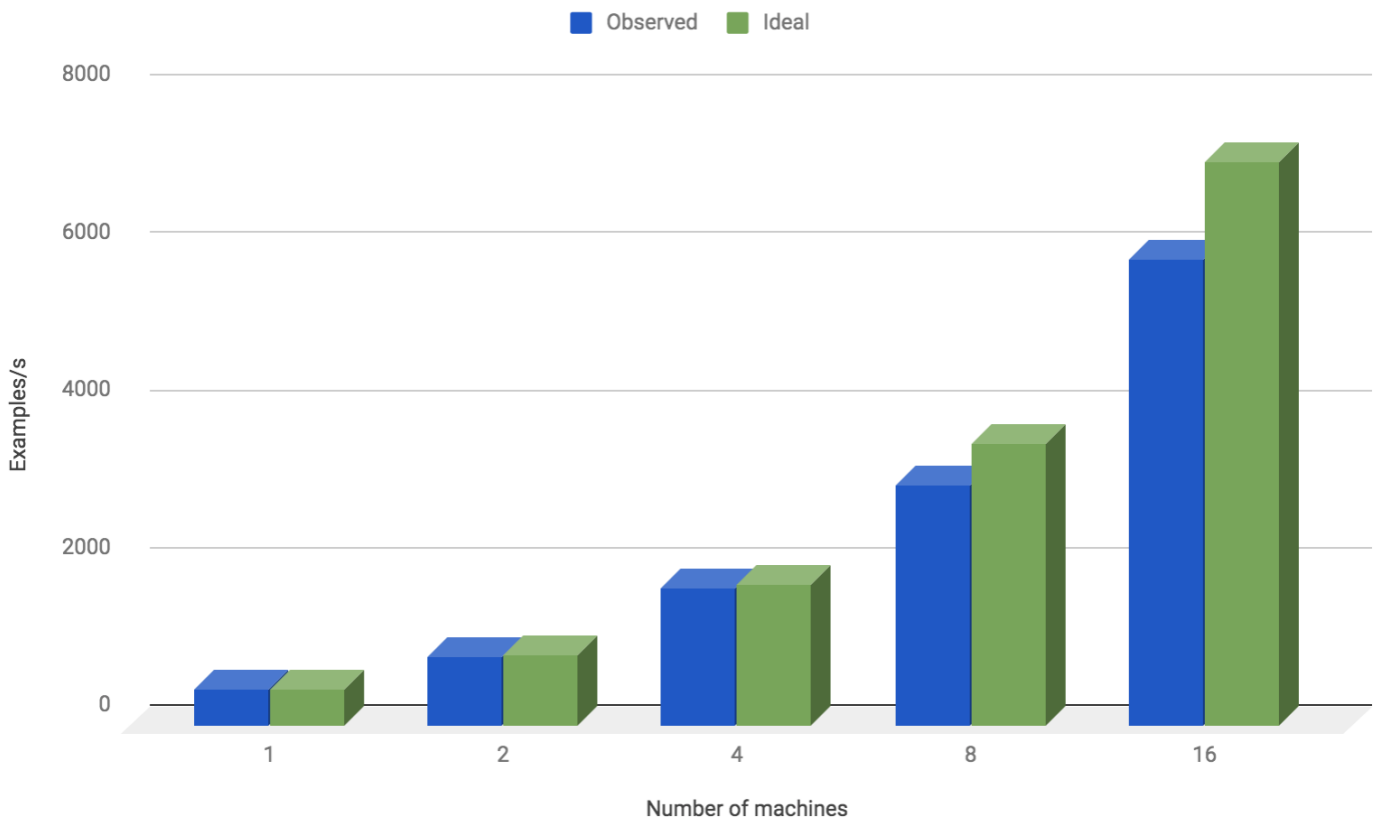
[17] Chen et al, *A Simple Framework for Contrastive Learning of Visual Representations*, 2020
[18] Pham et al, *Meta Pseudo Labels*, 2020

Media Coverage on LARS



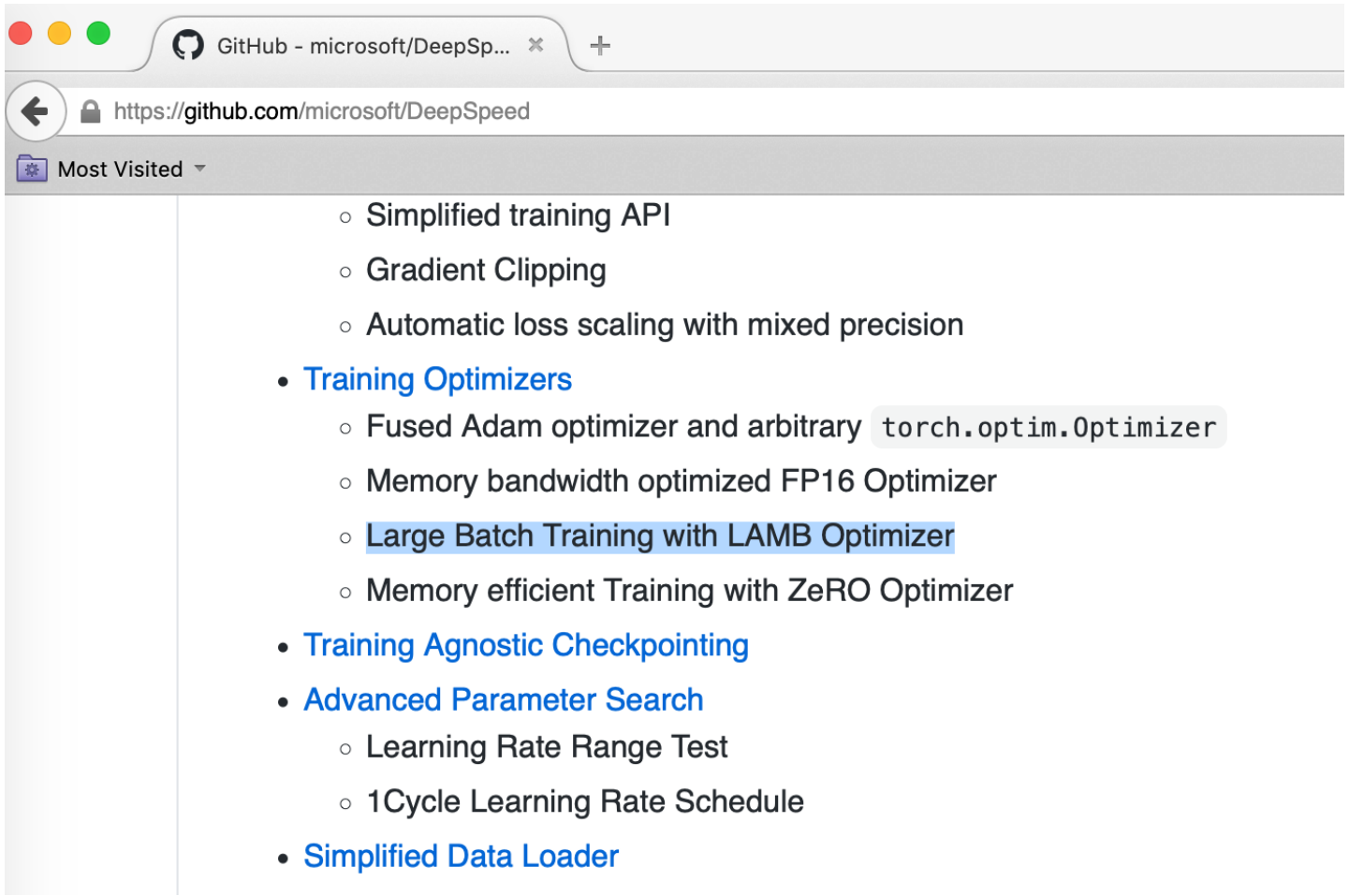
- Communication of ACM, EureKalert, fourthventricle, Get Knows, Intel, i-programmer, NSF, Parallel State, Primeur Magazine, Research & Development Magazine, Science Daily, Science NewsLine, TACC, Technology Networks, Technology News, The Next Web, Topix, UC Berkeley, World IT, etc

Early success of LAMB

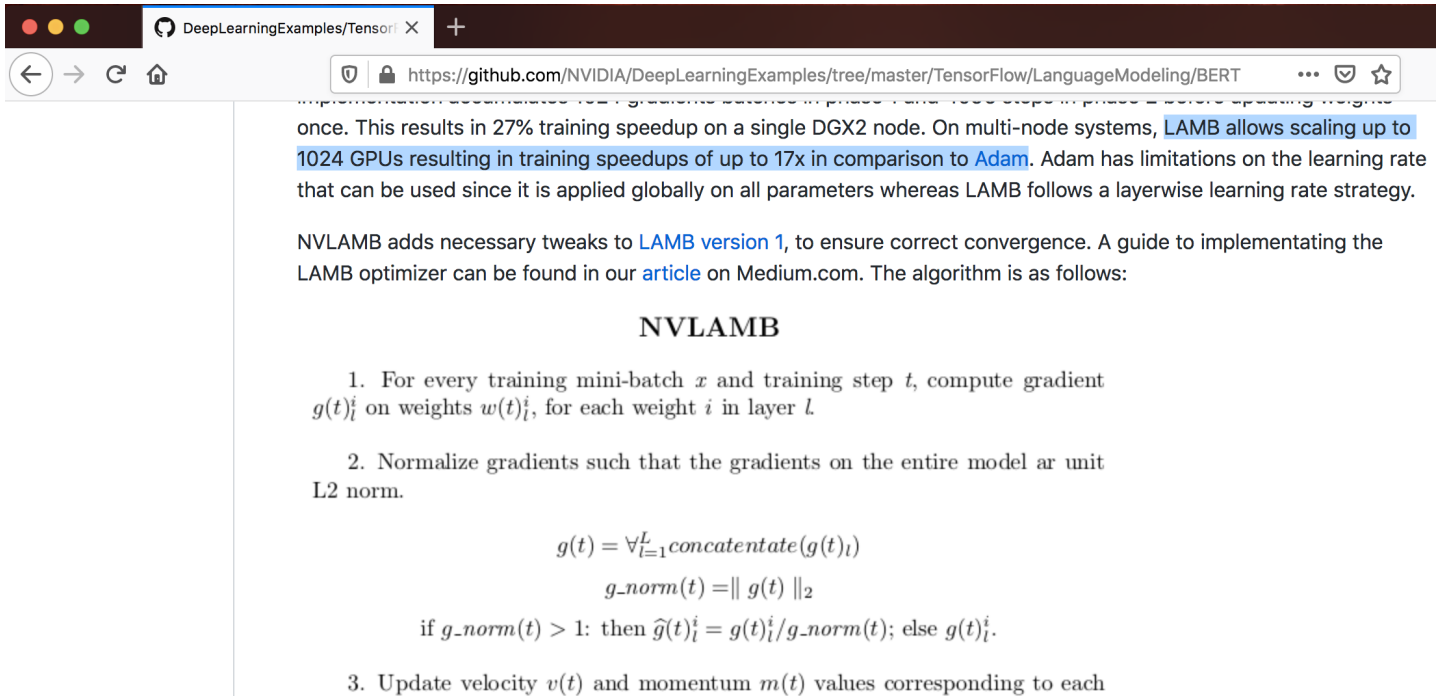


- "In our tests, we found that LAMB made a difference even on a single p3dn.24xlarge machine with global batch size 768, while Adam diverged immediately ..."
  - — by Yaroslav Bulatov, Ben Mann, Darius Lam, *Scaling Transformer-XL to 128 GPUs*

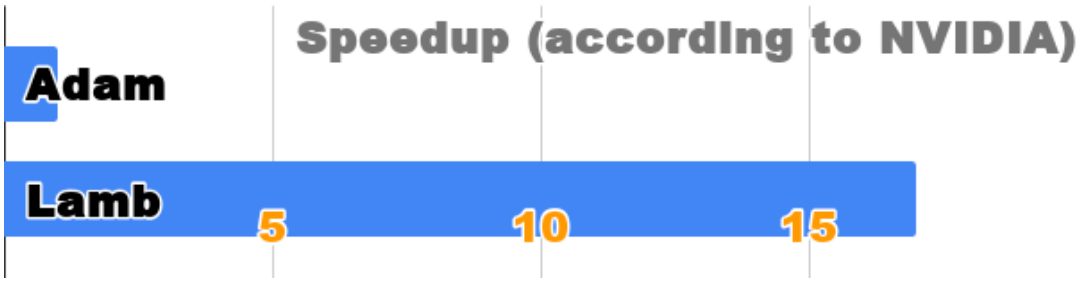
LAMB is being used by Microsoft DeepSpeed system



LAMB becomes an official optimizer of NVIDIA



- "LAMB allows scaling up to 1024 GPUs resulting in training speedups of up to 17x in comparison to Adam."
  - — NVIDIA offical github (screenshot at March 6th, 2020)



Impact of LAMB optimizer

- It helped Google achieve new state-of-the-art results on GLUE, RACE, and SQuAD benchmarks
- It is being used by state-of-the-art NLP models like Albert, ELECTRA, SHA-RNN
- LAMB achieves a 10× speedup for Mask-RCNN
  - `https://engineml.com/blog/2020-02-12-training-maskrcnn-10x-faster-with-lamb/`

"Earlier, it was all about SGD, naive-bayes and LSTM, but now it's more about LAMB, transformer and BERT."

NLP Interview Questions by Pratik Bhavsar
https://medium.com/modern-nlp/nlp-interview-questions-f062040f32f7

LARS/LAMB were added to MLPerf industry benchmark

Alibaba

AMD

Arm

Baidu

Cadence

Cerebras

Cisco

Cray

Dividiti

Enflame Tech

Esperanto

Google

Groq

Huawei

Intel

MediaTek

Mentor Graphics

Mythic

NetApp

NVIDIA

One Convergence

Rpa2ai

Sambanova

Samsung S.LSI

Sigopt

Synopsys

Tensyr

Wave Computing

Our early success

AI & MACHINE LEARNING

Google's scalable supercomputers for machine learning, Cloud TPU Pods, are now publicly available in beta

- LARS and LAMB were mentioned by Google's official product release

Beyond supercomputers: we build a better optimizer

- Our solver also works better for common hardware
  - We built a general optimizer (accurate, fast, scalable) for general data sets and architectures
- LAMB is the first optimizer working for both BERT and ResNet-50

Beyond supercomputers: we build a better optimizer

- Our solver also works better for common hardware
  - We built a general optimizer (accurate, fast, scalable) for general data sets and architectures
- LAMB is the first optimizer working for both BERT and ResNet-50

Beyond supercomputers: we build a better optimizer

- Our solver also works better for common hardware
  - We built a general optimizer (accurate, fast, scalable) for general data sets and architectures
- LAMB is the first optimizer working for both BERT and ResNet-50

Thanks for your time! Any questions?

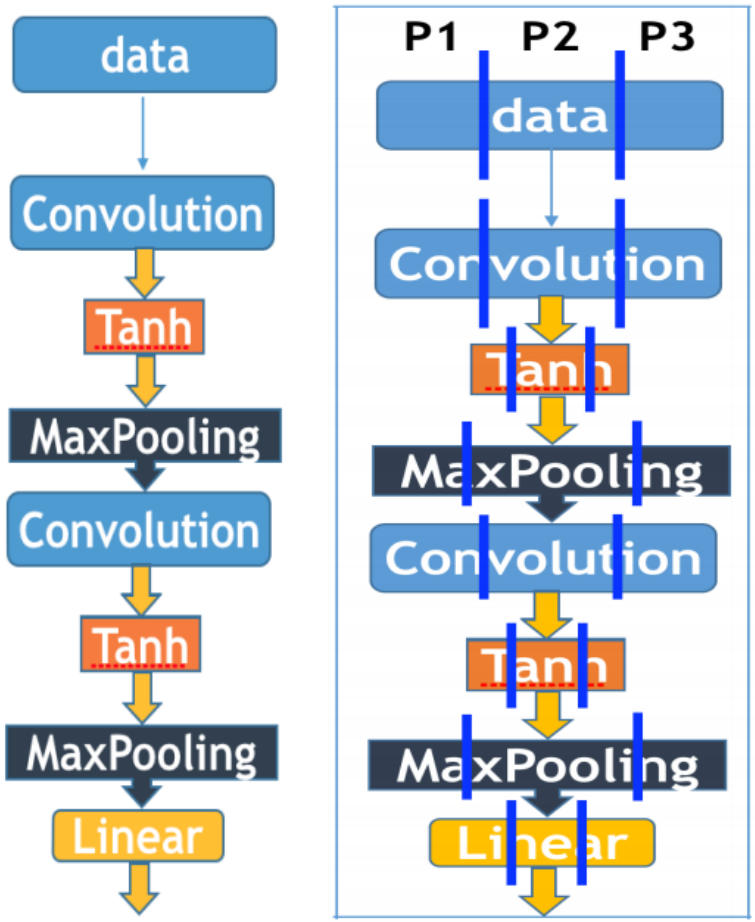
Use one sentence to sum up my talk

- To CV/NLP researchers:
  - We can help you finish the model training in minutes.
- To ML researchers:
  - We closed the generalization gap between training and testing.
- To Parallel Computing researchers:
  - We linearly reduced the critical path length.
- To Optimization researchers:
  - We solved a sharp minimum optimization problem.
- To System researchers:
  - We achieved 100% weak scaling efficiency on distributed systems.
- To Theory researchers:
  - Our algorithm can converge faster than SGD.

Summary of Recognition

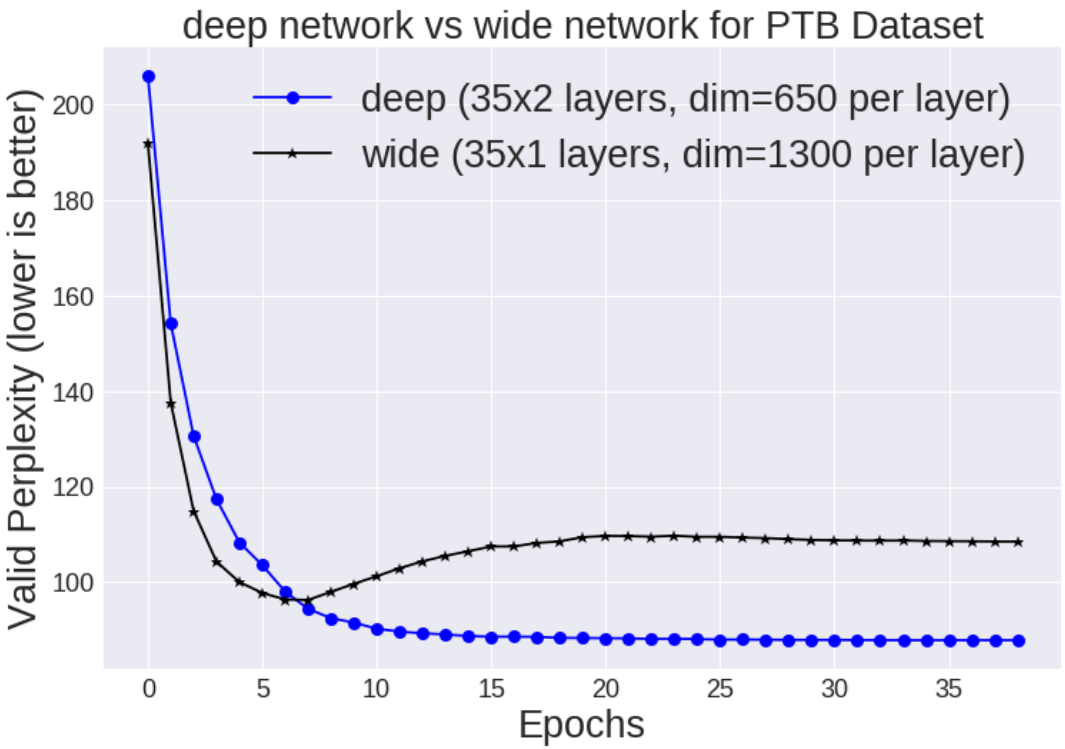
- LARS won the best paper award at ICPP'18
  - I am the first author
- LAMB is a top 10 most cited papers at ICLR'20
  - I am the first author
- CowClip won the distinguished paper award at AAAI'23
  - 12 of all 8777 submissions: 0.14% rate
  - I am the corresponding author, my Ph.D. student is the first author

Model-Parallelism 1: parallelize within each layer



- We need a very wide model

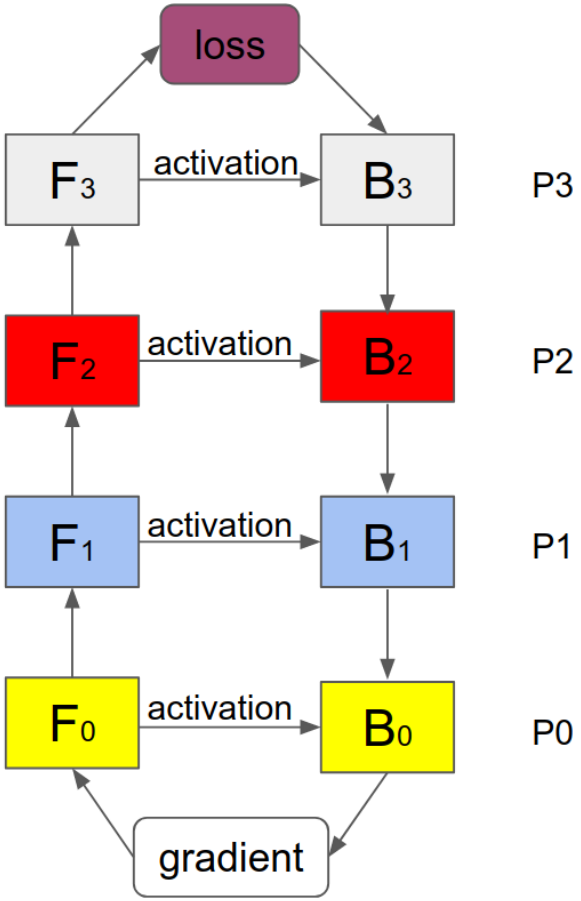
Wide model is not efficient



- Given the same number of parameters, a deep model can achieve better results than a wide model[19]
- 1 epoch: statically touches the whole dataset once

[19] Eldan and Shamir, *The power of depth for feedforward neural networks*, COLT 2016

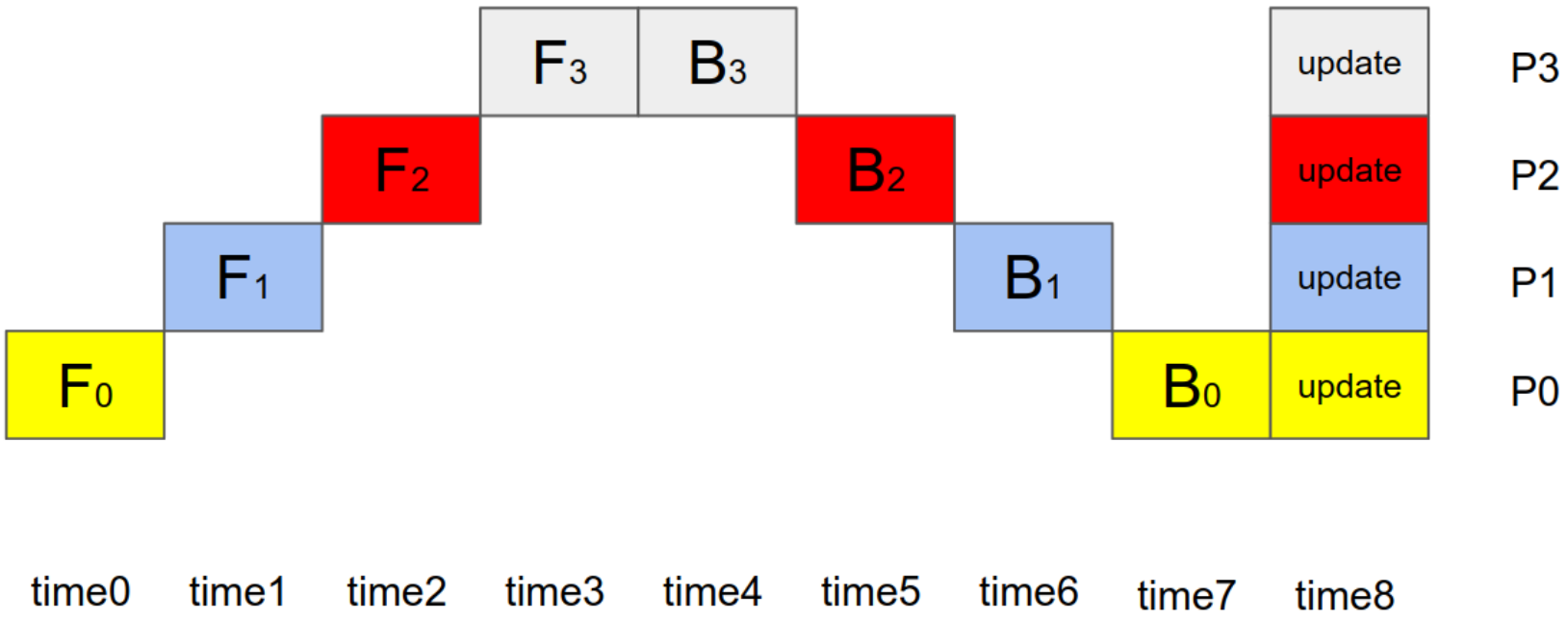
Model-Parallelism 2: Parallelize across different layers



F: forward pass; B: backward pass; P: processor

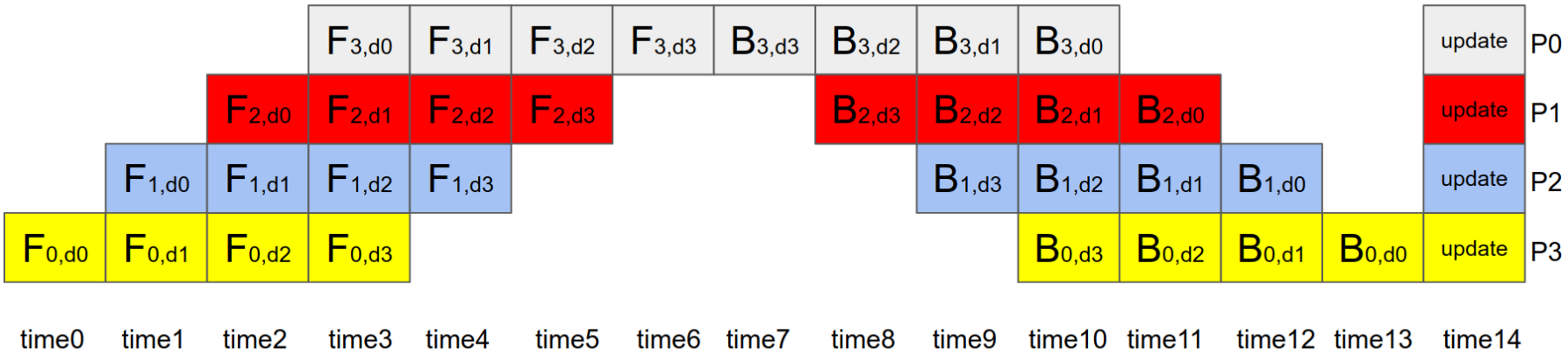
- Data dependency in forward pass and backward pass

Model-Parallelism 2: Parallelize across different layers



- The parallel efficiency is 1/P (very low)

Model-Parallelism 2: Parallelize across different layers



F: forward pass; B: backward pass; P: processor; d: data batch

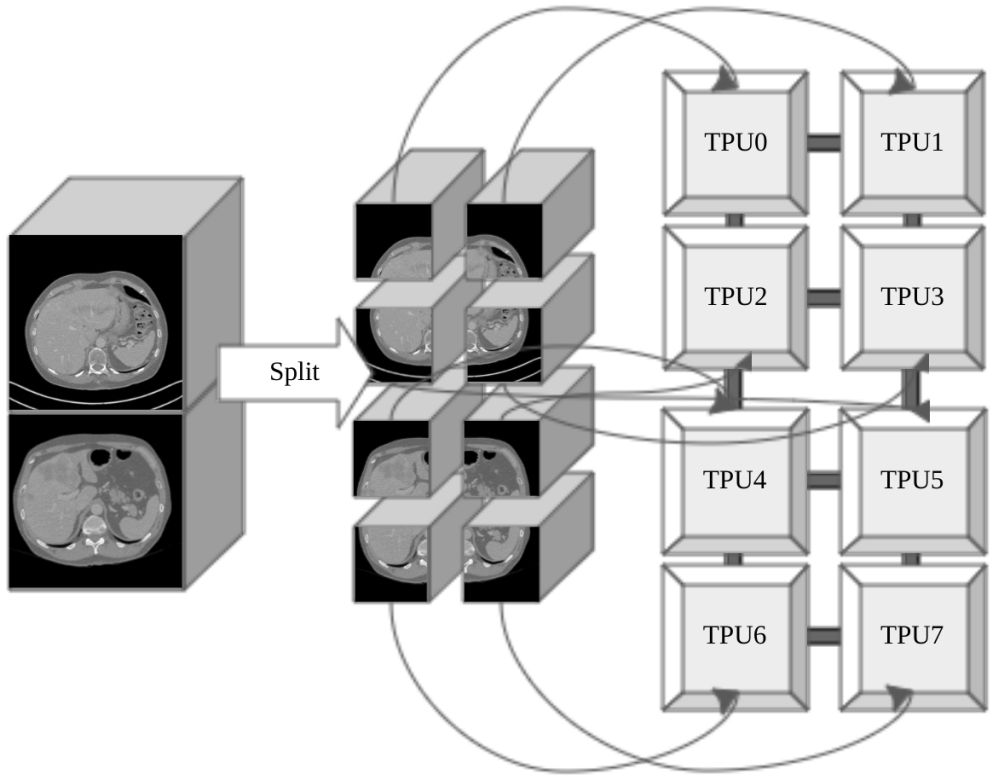
- Solution: combine pipeline with data parallelism
  - More details: GPipe[20] and PipeDream[21]

[20] Huang et al., *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism*, NeurIPS 2019

[21] Narayanan et al., *PipeDream: Generalized Pipeline Parallelism for DNN Training*, SOSP 2019

Model/Data Mixed Parallelism[22] or Domain Parallelism

- When the model or single sample (e.g. CT scan image) is huge
- Example: 1024 TPUs (Data Parallelism still dominates)
  - Regular Model: 1 model per TPU, 1024 model copies
  - Huge Model: 1/4 model per TPU, 256 model copies



[22] Shazeer et al., *Mesh-TensorFlow: Deep Learning for Supercomputers*, NeurIPS 2018

Summary

- How to make full use of thousands of processors?
  - Parallelize across different layers? (data dependency)
  - Parallelize within each layer? (build wide layers)
  - Parallelize the data? (needs a large batch size)
- No matter how we parallelize it, we will need data parallelism
  - Increase parallelism $\Rightarrow$ Increase the batch size
  - What is the challenge?

Challenges in Optimization of Federated Learning

- No successful story from asynchronous solvers for fast training
  - Bad things about asynchronous solvers[23]
    - Delayed updating: algorithm loses important information
    - Non-determinism often leads to a serious bug
    - It is hard to reproduce and debug
- Compress the gradients? It may hurt convergence

[23] Amodei et al., *Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin*, ICML 2016

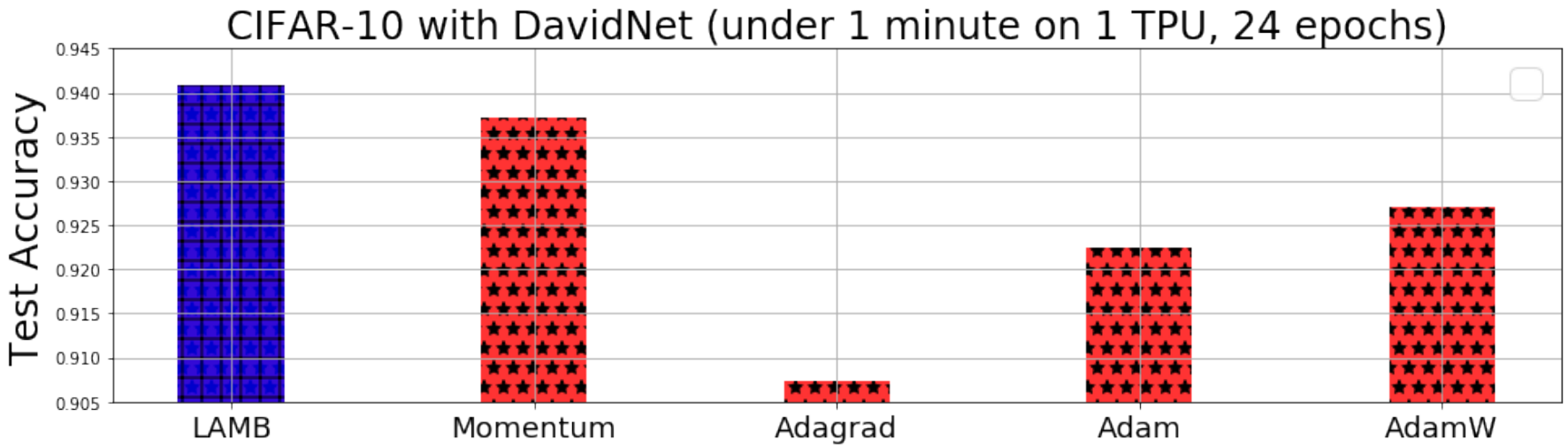
No need to re-tune hyper-parameters for scaling batch size

Table 5: ImageNet/ResNet-50 by LAMB Optimizer

Batch Size	512	1K	2K	4K	8K	16K	32K
Learning Rate	$\frac{4}{2^{3.0}\times 100}$	$\frac{4}{2^{2.5}\times 100}$	$\frac{4}{2^{2.0}\times 100}$	$\frac{4}{2^{1.5}\times 100}$	$\frac{4}{2^{1.0}\times 100}$	$\frac{4}{2^{0.5}\times 100}$	$\frac{4}{2^{0.0}\times 100}$
Warmup Epochs	0.3125	0.625	1.25	2.5	5	10	20
Top-5 Accuracy	0.9335	0.9349	0.9353	0.9332	0.9331	0.9322	0.9308
Top-1 Accuracy	0.7696	0.7706	0.7711	0.7692	0.7689	0.7666	0.7642

- ResNet-50 baseline achieved 76.3%/93% top-1/top-5 accuracy
- Use default values for other hyper-parameters.
  - Sqrt Learning Rate Scaling
  - Linear Epoch Warmup Scaling

Even on small datasets with a small batch, LAMB is better



- According to Stanford DawnBench, DavidNet is a state-of-the-art model for CIFAR-10 training

Even on small datasets with a small batch, LAMB is better

Optimizer	Momentum	Addgrad	Adam	AdamW	**LAMB**
Average accuracy over 5 runs	0.9933	0.9928	0.9936	0.9941	**0.9945**

- MNIST training with LeNet (30 epochs for Batch Size $= 1024$).
- The tuning space of learning rate: $\{0.0001, 0.001, 0.01, 0.1\}$.
- The same learning rate warmup and decay schedule for all of them.

LARS performance

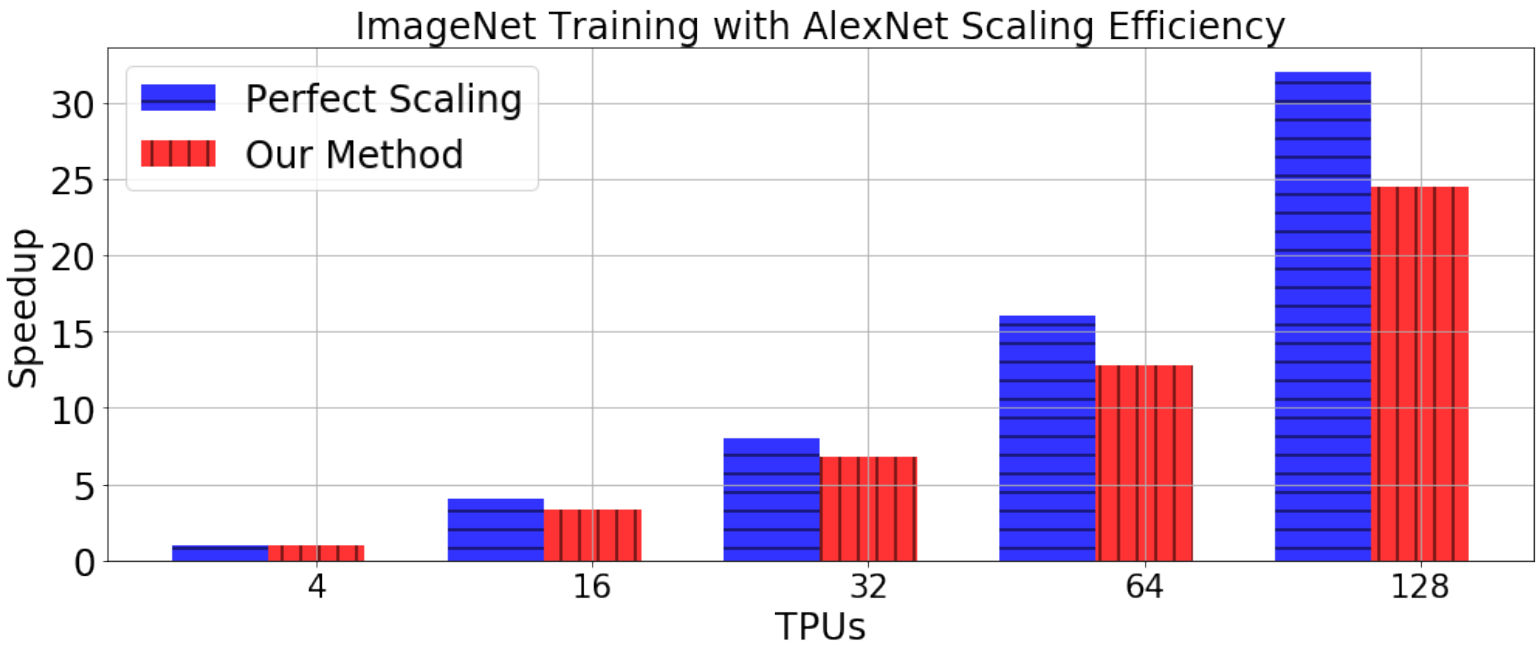
Table 6: Speedup for AlexNet.

Batch Size	epochs	Top-1 Accuracy	hardware	time
256	100	58.7%	8-core CPU + K20 GPU	144h
512	100	58.8%	DGX-1 station	6h 10m
4096	100	58.4%	DGX-1 station	2h 19m
32K	100	58.6%	512 KNLs	24m
32K	100	58.6%	1024 CPUs	11m

Table 7: Speedup for ResNet-50.

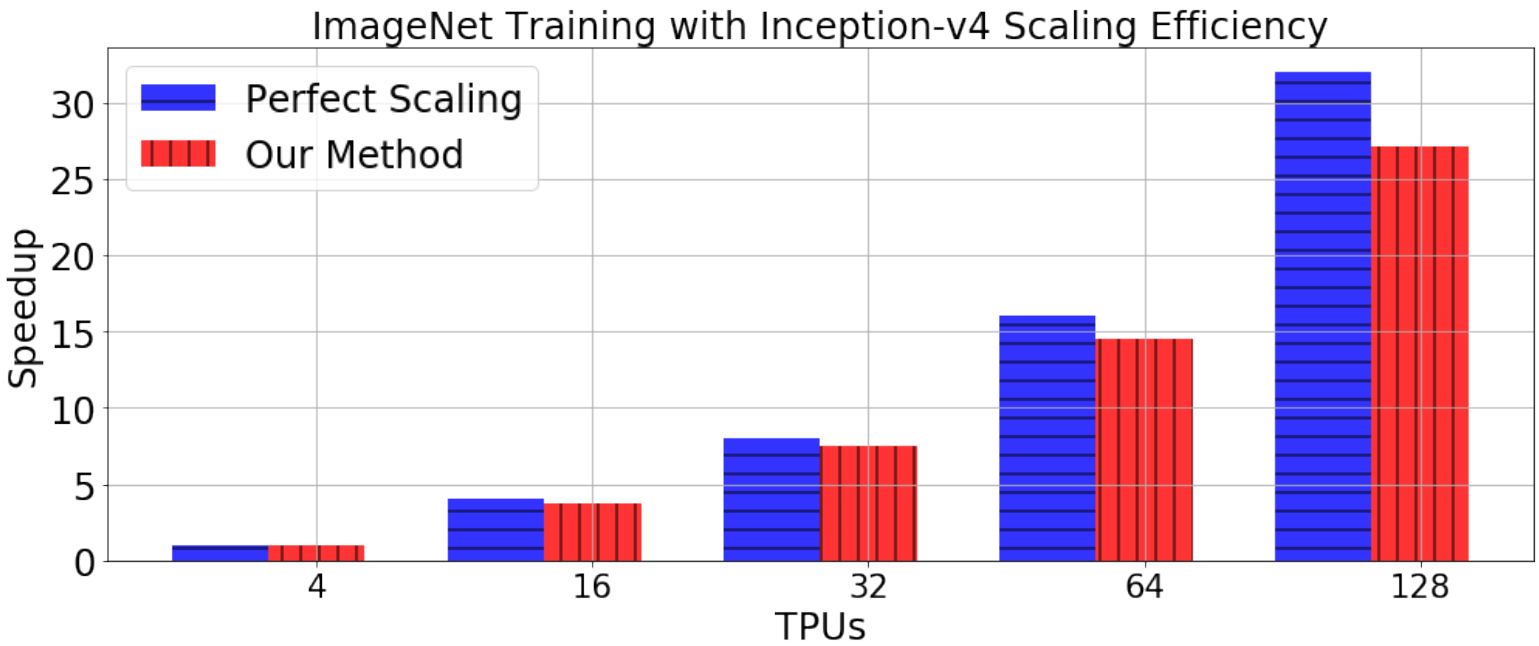
Batch Size	epochs	Top-1 Accuracy	hardware	time
256	90	75.3%	8 P100 GPUs	21h
32K	90	75.4%	512 KNLs	60m
32K	90	75.4%	1600 CPUs	32m
32K	90	75.4%	2048 KNLs	20m
32K	90	76.4%	TPU-v2 Pod	8m
64K	90	76.3%	TPU-v3 Pod	1m

Scaling on Different Models



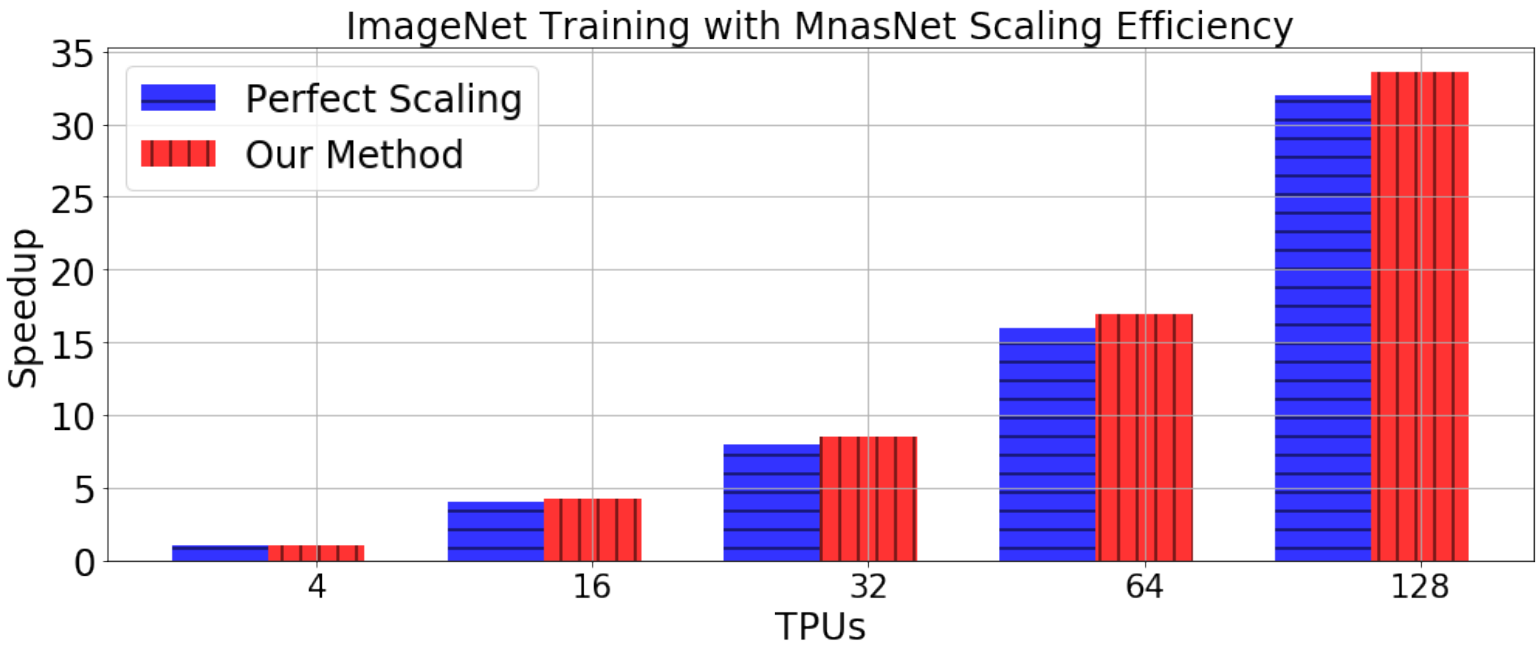
- 76.66% scaling efficiency

Scaling on Different Models



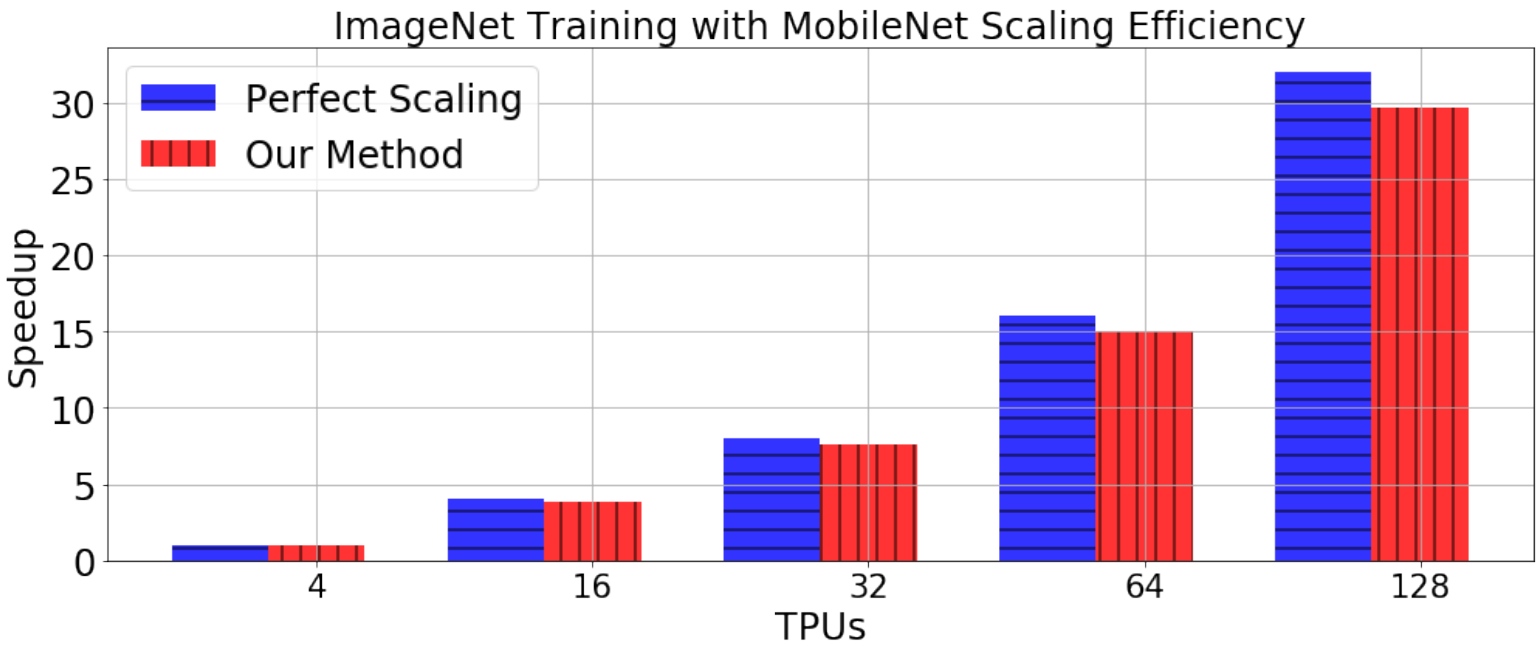
- 84.76% scaling efficiency

Scaling on Different Models



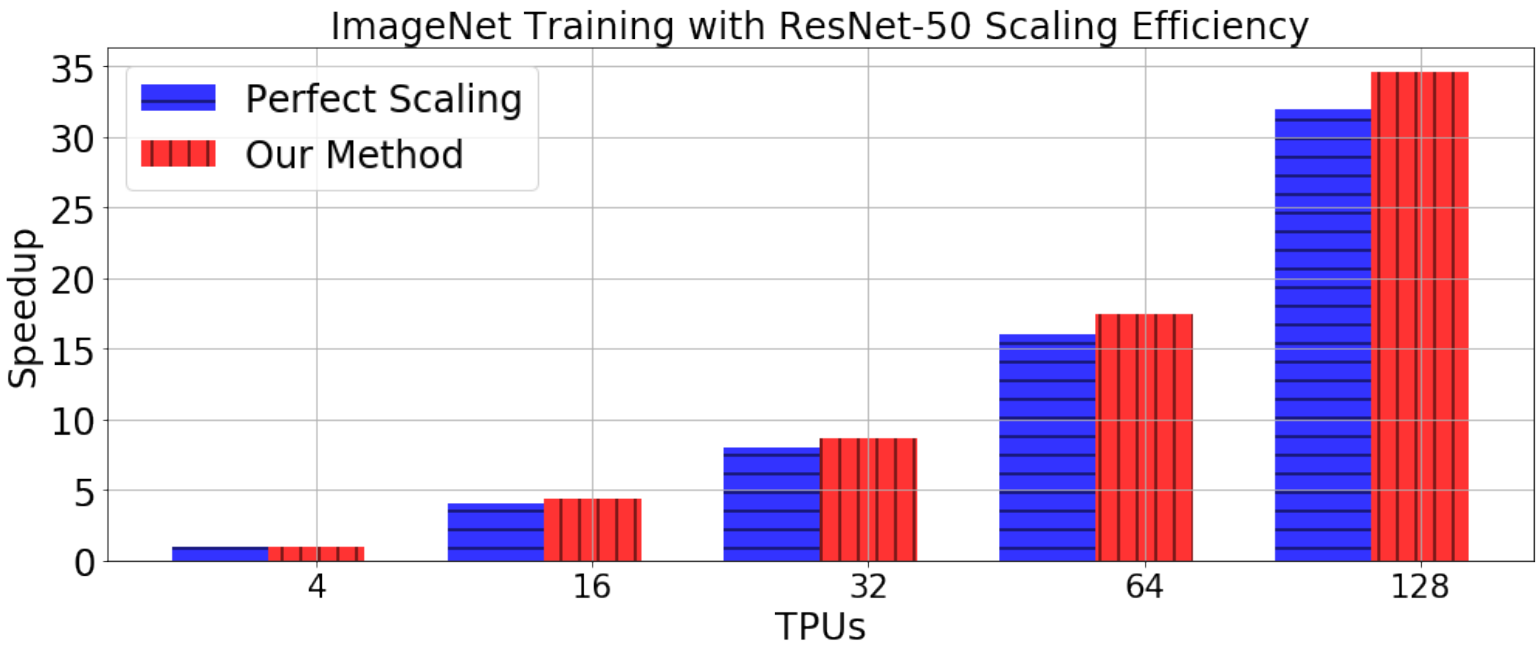
- 100.05% scaling efficiency

Scaling on Different Models



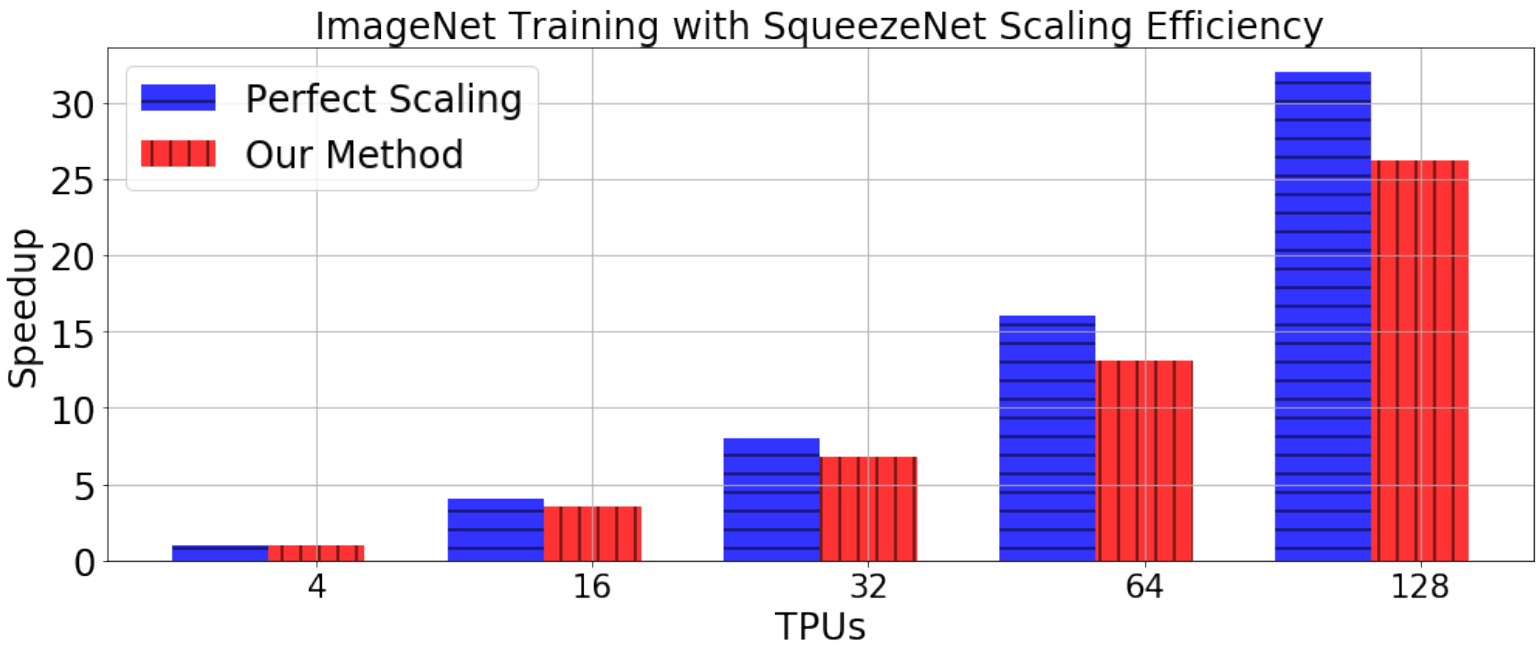
- 92.82% scaling efficiency

Scaling on Different Models



- 100.08% scaling efficiency

Scaling on Different Models



- 81.89% scaling efficiency

Linear Epoch Gradual Warmup (LEGW or Leg-Warmup)

- if we increase B to kB, then increase the warmup epochs by k times
- why LEGW works?

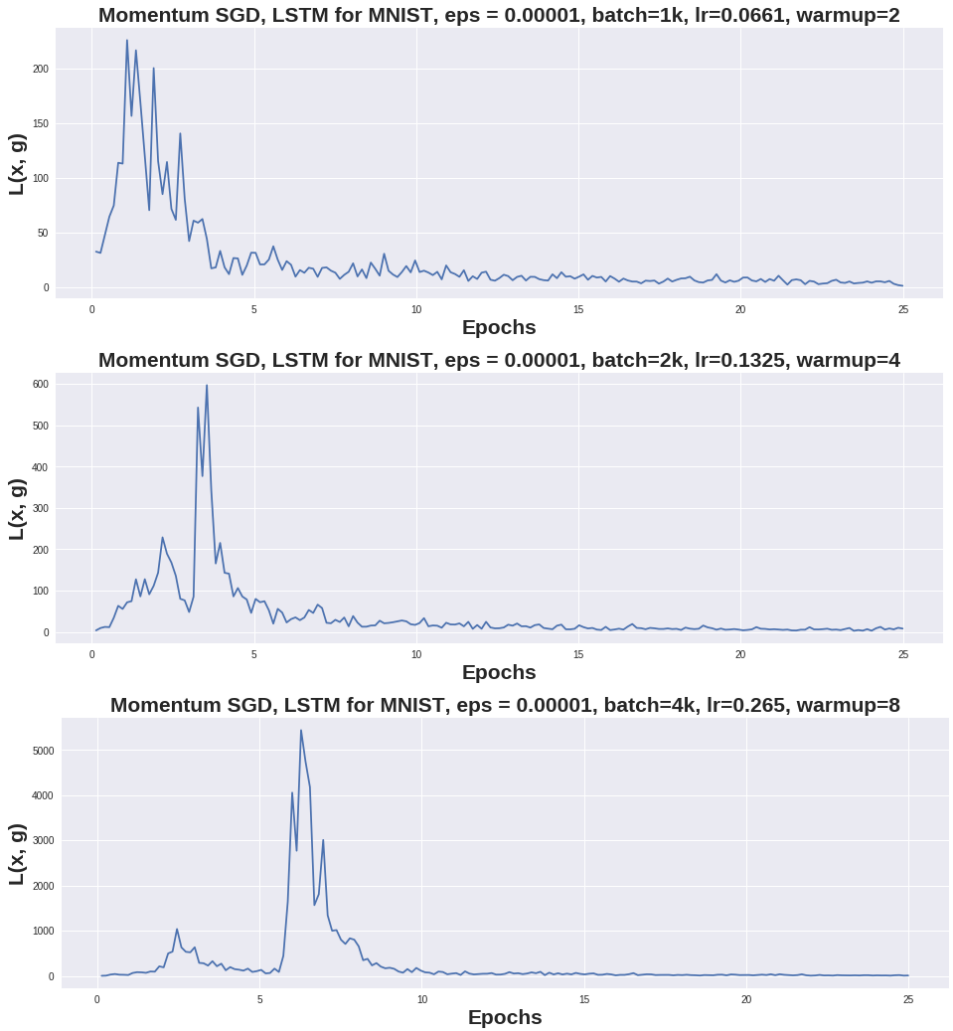
Why LEGW works?

- gradient direction $g = \nabla f(x)$
- the update is $x \leftarrow x - \eta g$
- how to choose η?
- $f(x+\Delta) \approx \tilde{f}(x+\Delta) := f(x) + \Delta^T \nabla f(x) + \frac{1}{2}\Delta^T \nabla^2 f(x) \Delta$
- we find Δ to minimize the approximation function
- if we assume Δ is in the form of $-\eta g$ and Hessian is positive definite along the direction of g ($g^T \nabla^2 f(x) g > 0$), then the optimal η^* is

$$\arg\min_{\eta} \tilde{f}(x - \eta g) = \frac{1}{g^T \nabla^2 f(x) g / \|g\|^2} := \frac{1}{L(x,g)}$$

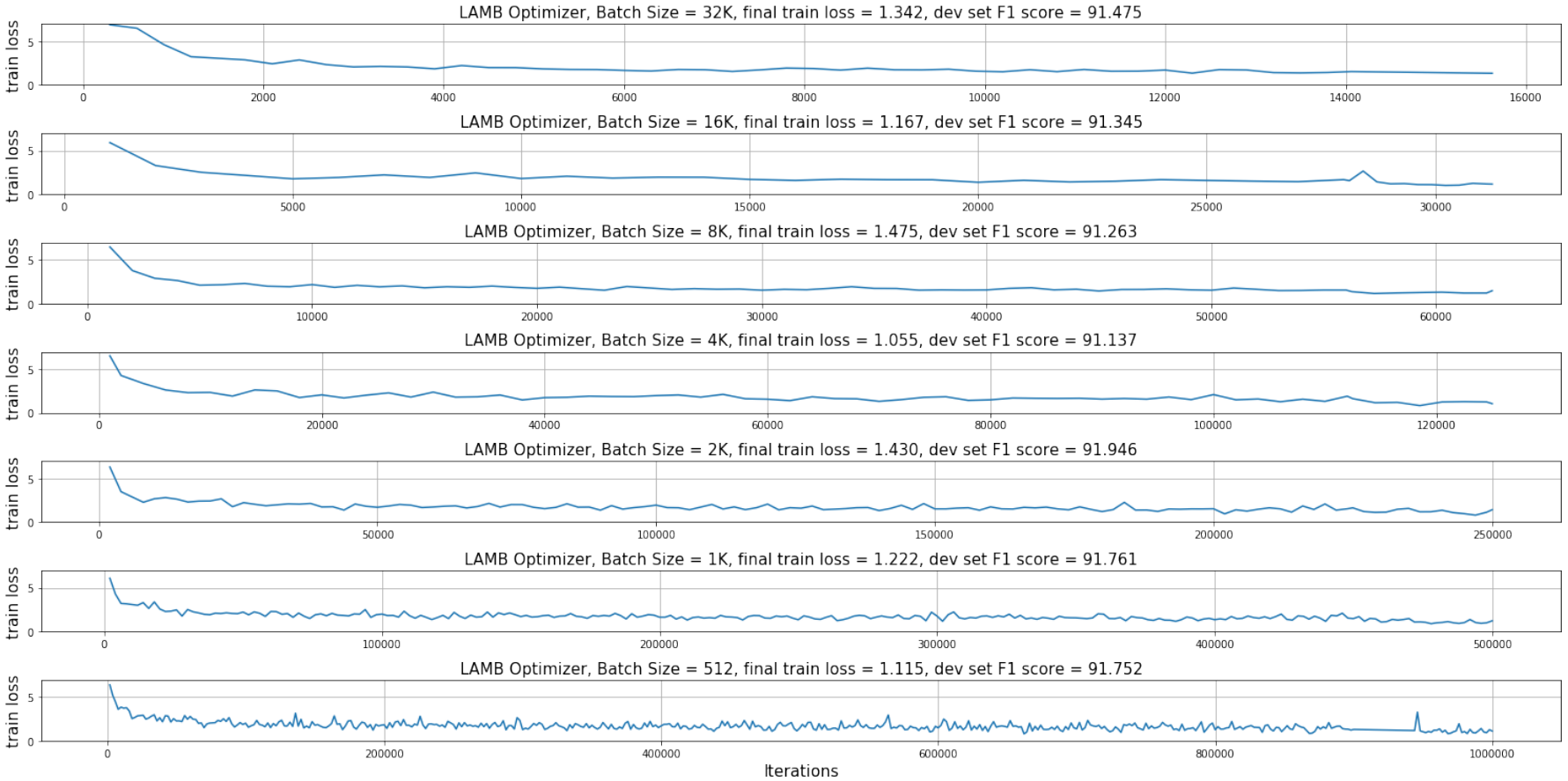
- η^* is inversely proportional to $L(x,g)$
- it is hard to get $L(x,g)$ since $\nabla^2 f(x)$ involves all the training samples
- we approximate $L(x,g)$ using a batch of data and compute the Hessian-vector product by finite difference

Why LEGW works?



- a smaller η^* needed in the beginning (which implies warmup)
- as batch size increases, a longer warmup to cover the peak region

Even the train loss curves are almost identical



- BERT training by LAMB optimizer

More explanation

- *An intuitive understanding of the LAMB* by fast.ai: