

# Presidential Young Professor at NUS



**Yang You**

- ✓ PhD from UC Berkeley (with Prof. Jim Demmel)
- ✓ IPDPS Best Paper Award (0.8%, first author)
- ✓ ICPP Best Paper Award (0.3%, first author)
- ✓ ACM/IEEE George Michael HPC Fellowship
- ✓ Forbes 30 Under 30 Asia list (2021)
- ✓ IEEE CS TCHPC Early Career Researchers Award
- ✓ Lotfi A. Zadeh Prize for outstanding UC Berkeley PhD
- ✓ ICML Expert reviewer
- ✓ Lab's homepage : <https://ai.comp.nus.edu.sg/>

## Research Interests:

- High Performance Computing
- Deep Learning Optimization
- Machine Learning System





# Background

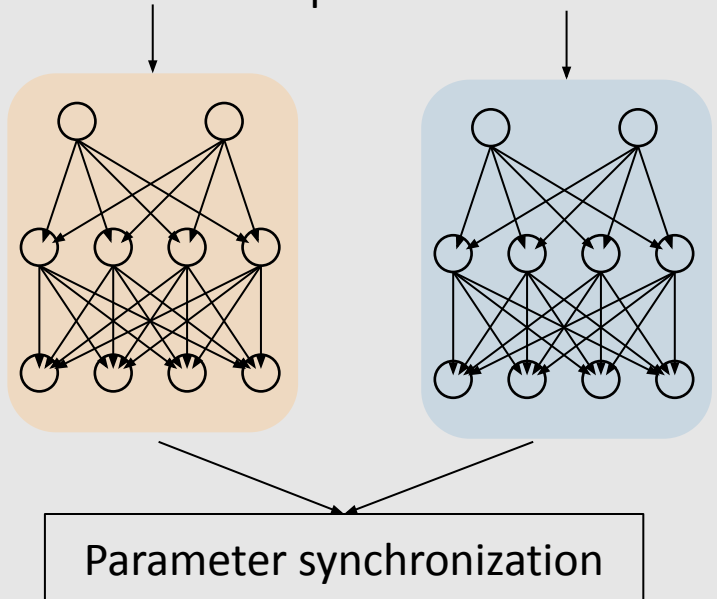
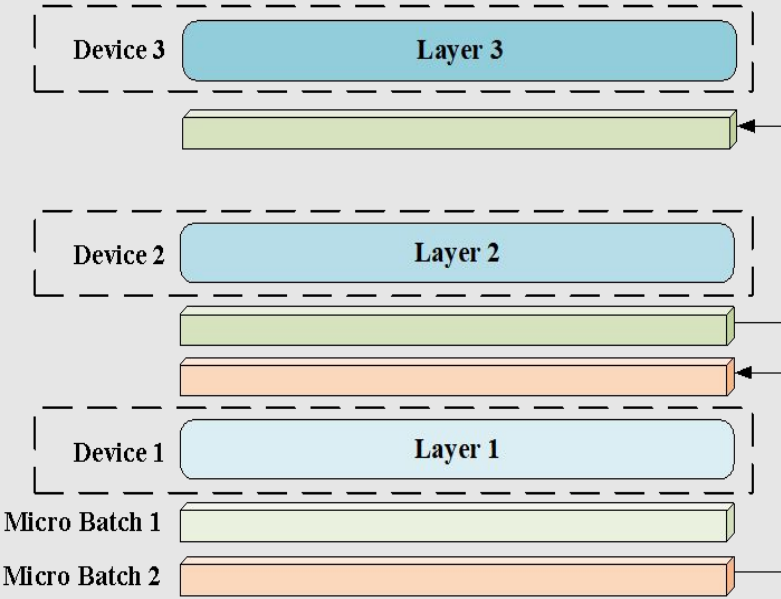
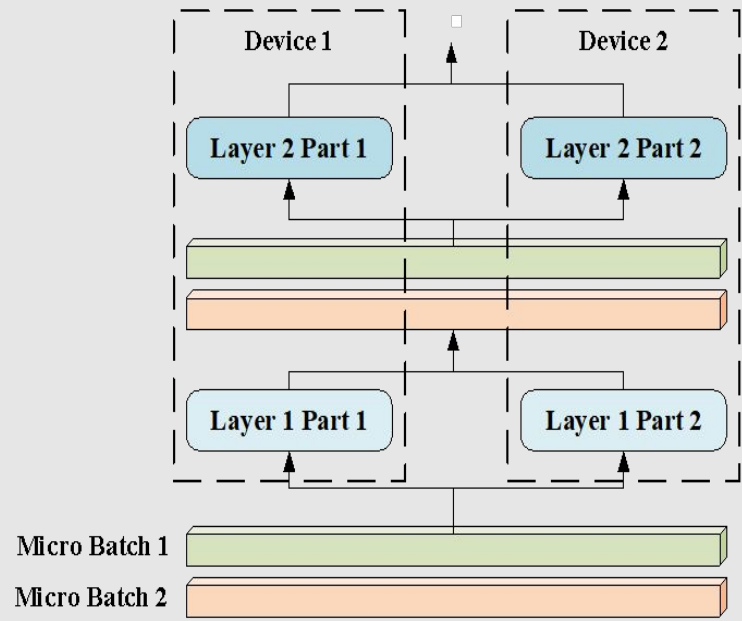
- The growth of sizes of neural networks

time	model	size
June, 2018	OpenAI GPT	0.11 billion
Oct, 2018	BERT	0.34 billion
Feb, 2019	GPT-2	1.5 billion
Sep, 2019	Megatron-LM	8.3 billion
Feb, 2020	Turing-NLP	17 billion
June, 2020	GPT-3	175 billion
June, 2020	Google MoE	600 billion
Jan, 2021	Google Switch Transformer	16000 billion

AI model size: 3.5-month doubling  
Moore's law: 18-month doubling



# Where the parallelism comes from for deep learning?

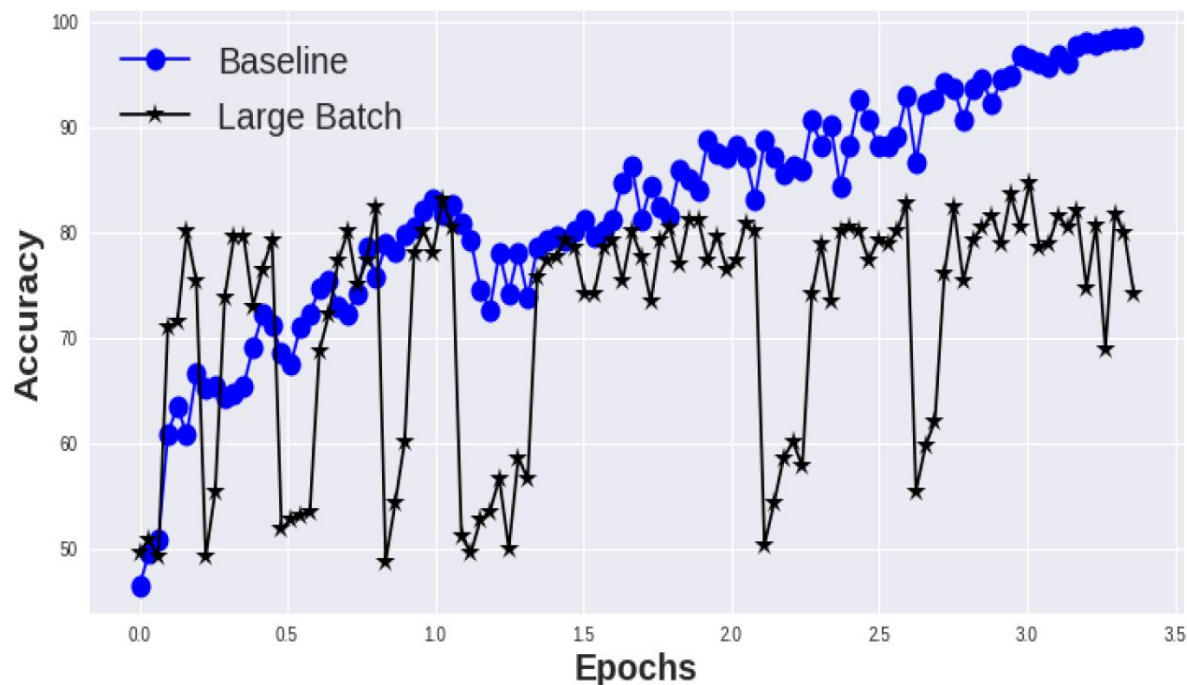
Data Parallelism	Pipeline Parallelism	Tensor Parallelism
<ul style="list-style-type: none"><li>•Distribute data across processors</li><li>•Processed in parallel, and parameters are updated synchronously.</li><li>•Communication happens at the all-reduce operations to sum the gradients from all processors</li></ul> 	<ul style="list-style-type: none"><li>• Model is split by layer</li><li>• Data is split into micro batches for pipelining</li><li>• Communication occurs between pipeline stages(devices)</li></ul> 	<ul style="list-style-type: none"><li>• Tensor is split across devices</li><li>• Example: Megatron</li></ul> 



# Problem of large-batch training (data parallelism)

## Problems of existing algorithm :

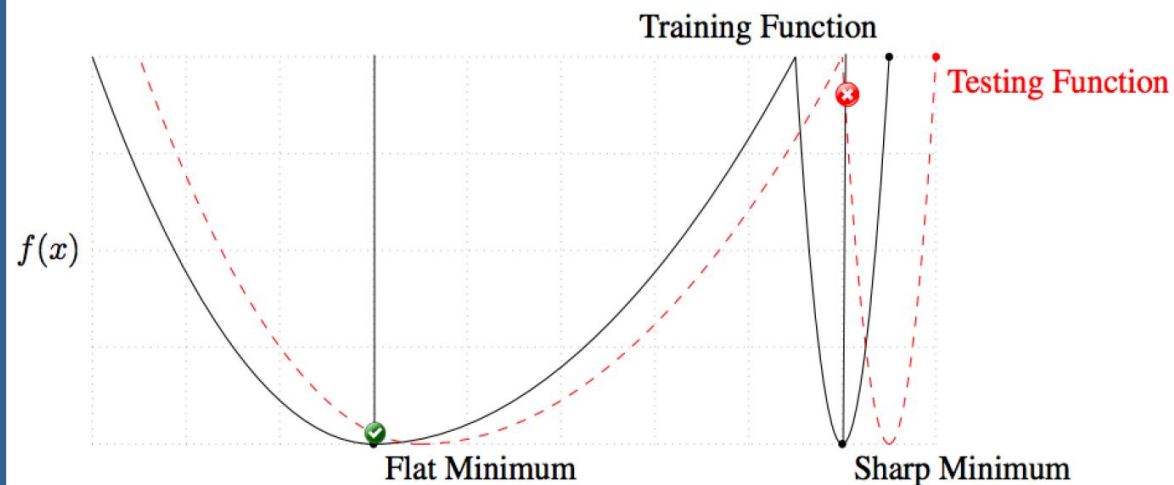
### Reduced accuracy



Reason: sharp minimum problem & poor convergence

Larger batch requires a greater learning rate. The upper limit of batch size for other methods is 8K.

### sharp minimum problem



- Large batch training = sharp minimum problem.
- For problem with shape the sharp minimum, a good performance does not mean a good performance in the test set.



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;  $\eta$ : learning rate;  $w_t$ : weights;  $\lambda$ : weight decay;  $\phi$ : scaling function;  
 $x_t$ : data samples;  $f$ : loss function;  $g_t$ : gradients;  $\beta_1$ : coefficient of momentum;

Each layer has its own unique learning rate

The trust ratio is changing between different iterations: adaptive scaling at runtime





within layer  $l$  and iteration  $t$

$$\begin{aligned}
 g_t &= \frac{1}{B} \sum_{i=1}^B \nabla f(x_t^i, w_{t-1}^l) \text{ /* compute the gradients */} \\
 m_t^l &= \beta_1 m_{t-1}^l + (1 - \beta_1) g_t^l \text{ /* compute the first moment */} \\
 v_t^l &= \beta_2 v_{t-1}^l + (1 - \beta_2) g_t^l \odot g_t^l \text{ /* compute the second moment */} \\
 \hat{m}_t^l &= m_t^l / (1 - \beta_1^t) \text{ /* bias correction for first moment */} \\
 \hat{v}_t^l &= v_t^l / (1 - \beta_2^t) \text{ /* bias correction for second moment */} \\
 r_t^l &= 1.0 \text{ /* initialize the trust ratio */} \\
 r_1 &= \phi(\|w_{t-1}^l\|) \text{ /* compute the norm of the weights */} \\
 r_2 &= \left\| \frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l \right\| \text{ /* element-wise weight decay */} \\
 \text{if } r_2 > 0, \text{ then } r_t^l &= r_1 / r_2 \text{ /* compute the trust ratio */} \\
 w_t^l &= w_{t-1}^l - \eta \times r_t^l \times \left( \frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l \right) \text{ /* update the weights */}
 \end{aligned}$$

Notations

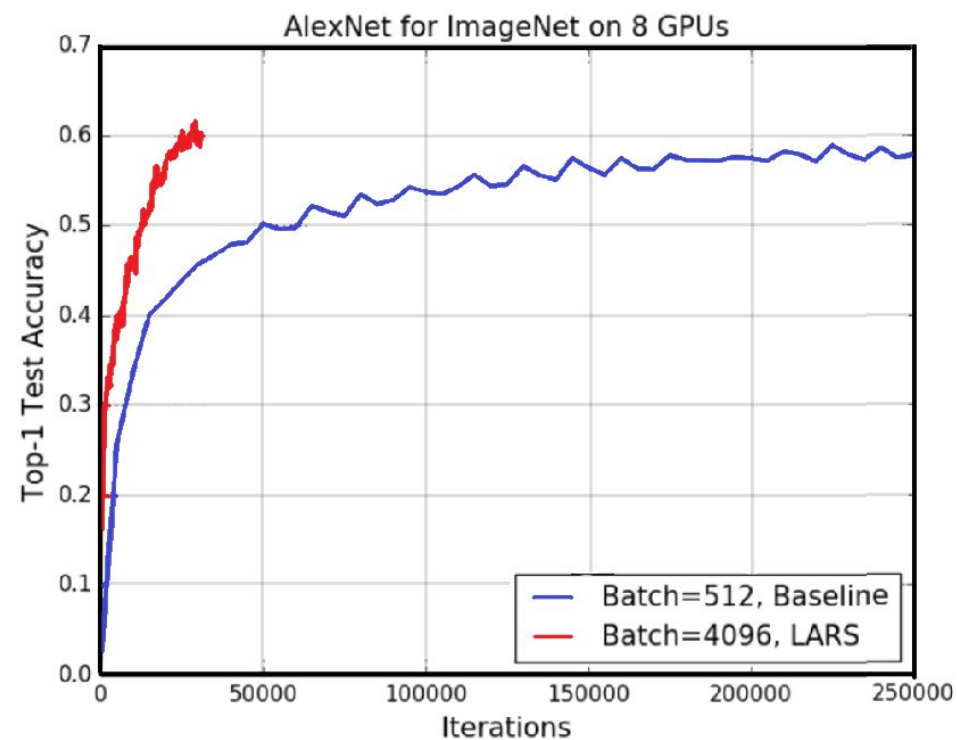
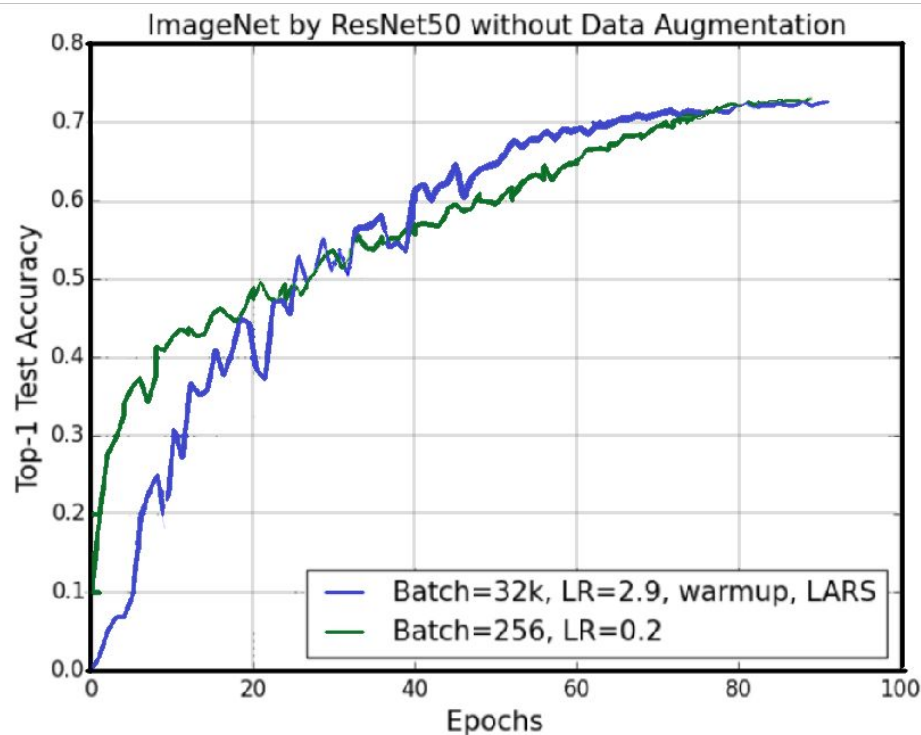
$B$ : batch size;  $\eta$ : learning rate;  $w_t$ : weights;  $\lambda$ : weight decay;  $\phi$ : scaling function;  
 $x_t$ : data samples;  $f$ : loss function;  $g_t$ : gradients  $\beta_1/\beta_2$ : coefficient of first/second moment;

adaptive element-wise updating + layer-wise correction

element-wise weight decay is more accurate (this preserves more information)



# Benefits of LARS for ImageNet Training



- LARS: Larger batch size without loss of performance. Greatly shorten training time.



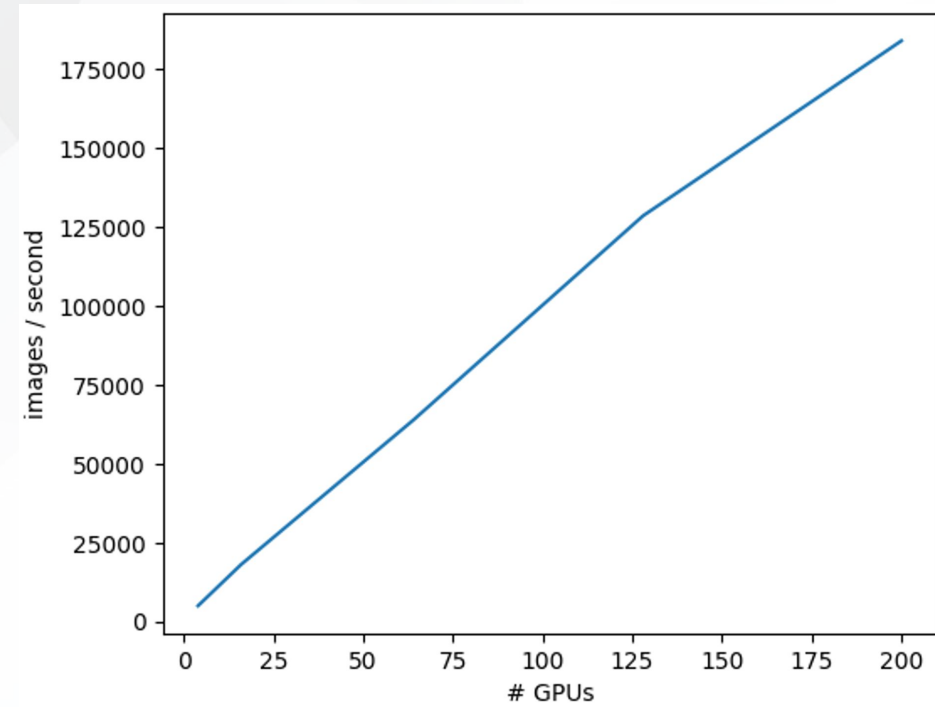


# Speedup for ViT

## Benefits

- Maximize the use of GPU resources and achieve near linear acceleration with guaranteed convergence.

# GPU	Batch size	300 epochs (hour)
1	128	73
4	512	21
16	2048	5.88
64	8192	1.67
128	16k	0.83
200	32k	0.68



ViT-B/32 ImageNet-1K





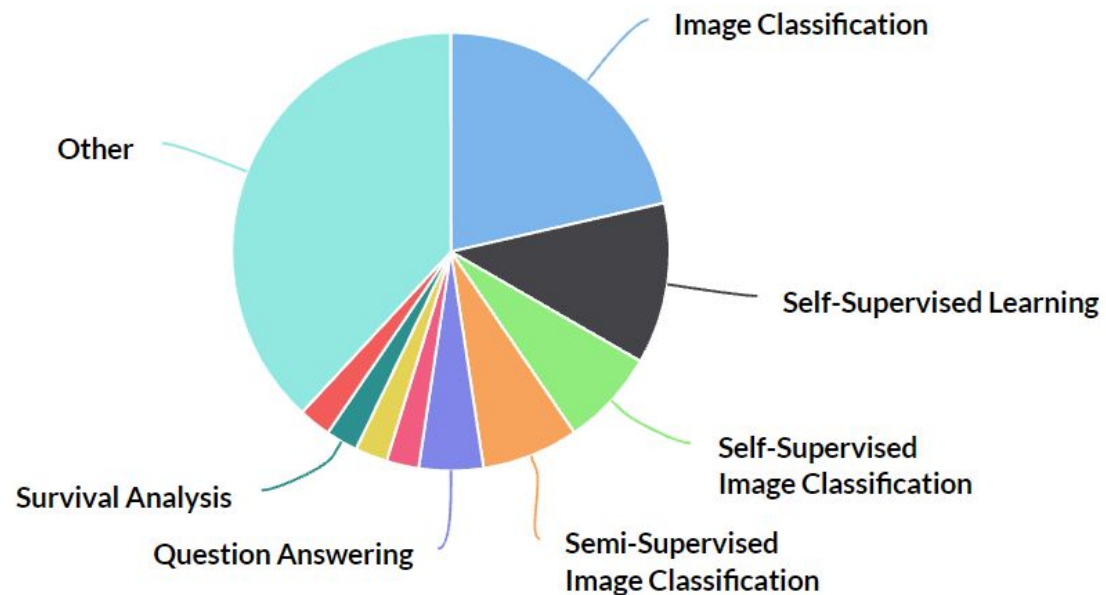
# LAMB: reducing BERT training time

LAMB :

Optimizer	batch size	steps	F1 score on dev set	TPUs	Time
<b>Baseline</b>	<b>512</b>	<b>1000k</b>	<b>90.395</b>	<b>16</b>	<b>81.4h</b>
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
<b>LAMB</b>	<b>64k/32k</b>	<b>8599</b>	<b>90.584</b>	<b>1024</b>	<b>76.19m</b>

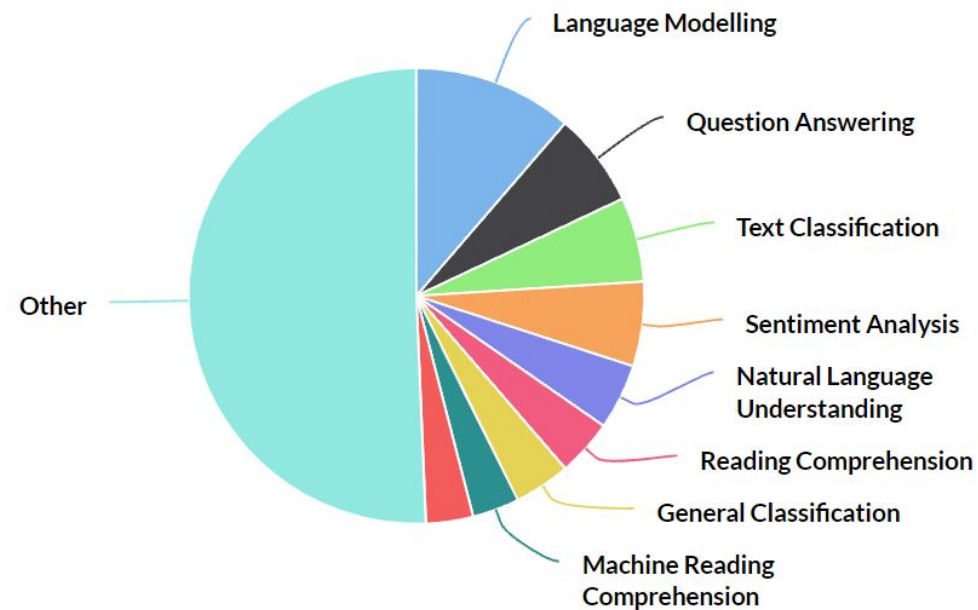


## LARS :



LARS: <https://paperswithcode.com/method/lars>

## LAMB :



LAMB: <https://paperswithcode.com/method/lamb>

- High adaptability
- Refresh the world record of training speed
- Widely used in the industry



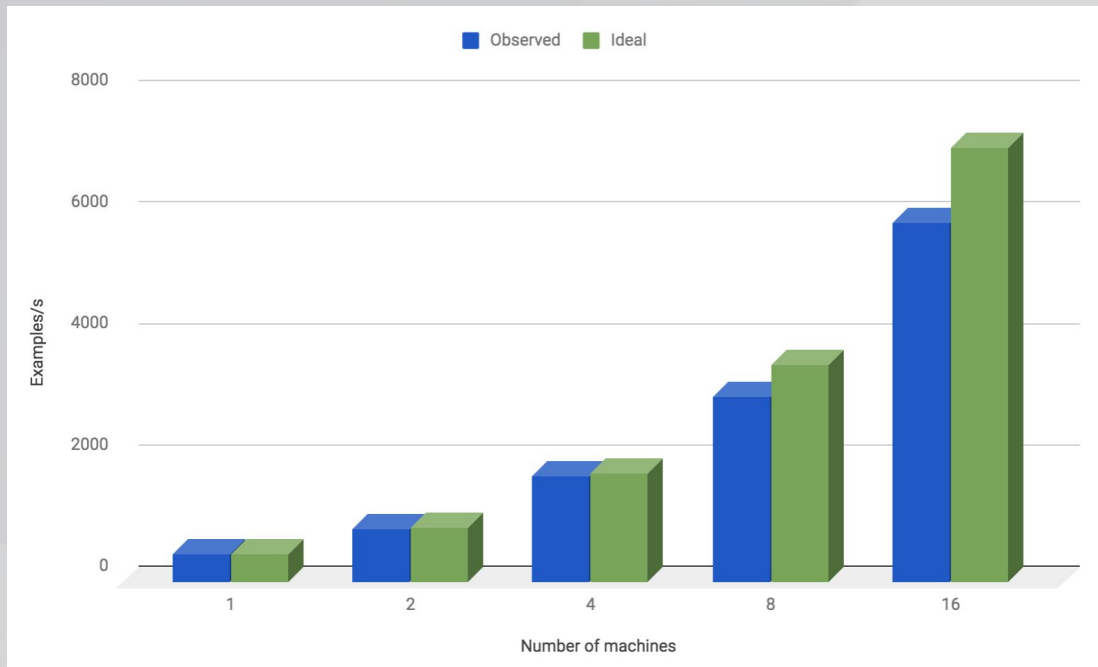
# ImageNet Training Speed World Record

Team	Date	Time	Method
Microsoft	2015/12/10	29 hours	SGD
Facebook	2017/6/8	1 hour	SGD
Berkeley	2017/12/7	14 minutes	LARS
Tencent	2018/7/30	6.6 minutes	LARS
Sony	2018/11/14	3.7 minutes	LARS
Google	2018/11/16	2.2 minutes	LARS
Fujitsu	2019/3/29	1.3 minutes	LARS
Google	2019/7/10	1.1 minutes	LARS
Google	2020/11/7	29 second	LARS

- LARS is being used in industry
  - Dr. Hinton used LARS for self-supervised learning
  - Google SimCLR, Facebook SEER, DeepMind BYOL



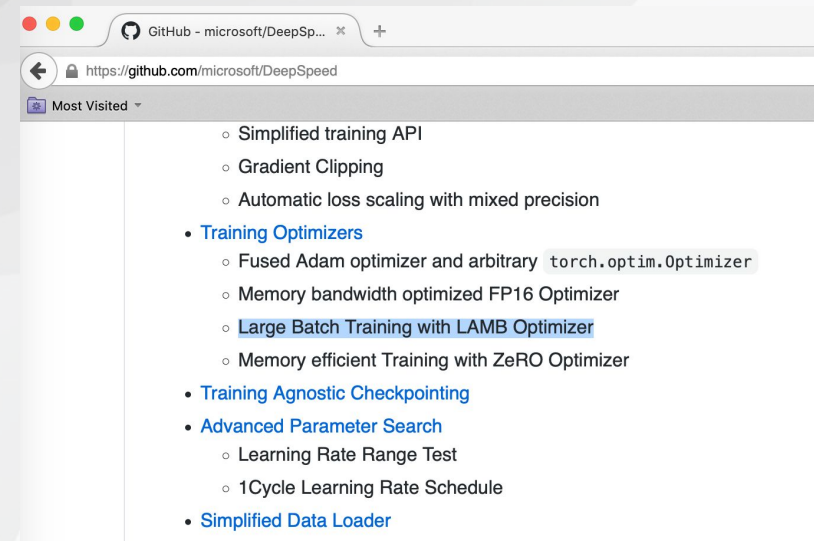
# Applications



"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 helped fast.ai to scale AI to 128 GPUs



LAMB is being used in DeepSpeed

**"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>

LAMB is becoming a commonly-used technique in industry



# Impact

- Our work was reported by many Tech medias
  - Science Daily, TACC, Intel, Tech Networks, The Next Web, USA NSF
- LARS and LAMB are included in MLPerf
  - Members of MLPerf includes Google, Alibaba, NVIDIA, Intel, AMD, Huawei, Baidu
- According to NVIDIA GitHub, LAMB is **72x faster than previous methods**

→ ↻ 🔒 <https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT> ☆

☰ README.md

LAMB stands for Layerwise Adaptive Moments based optimizer, is a large batch optimization technique that helps accelerate training of deep neural networks using large minibatches. It allows using a global batch size of 65536 and 32768 on sequence lengths 128 and 512 respectively, compared to a batch size of 256 for Adam. The optimized implementation accumulates 1024 gradient batches in phase 1 and 4096 steps in phase 2 before updating weights once. This results in 15% training speedup. On multi-node systems, LAMB allows scaling up to 1024 GPUs resulting in training speedups of up to 72x in comparison to Adam. Adam has limitations on the learning rate that can be used since it is applied globally on all parameters whereas LAMB follows a layerwise learning rate strategy.

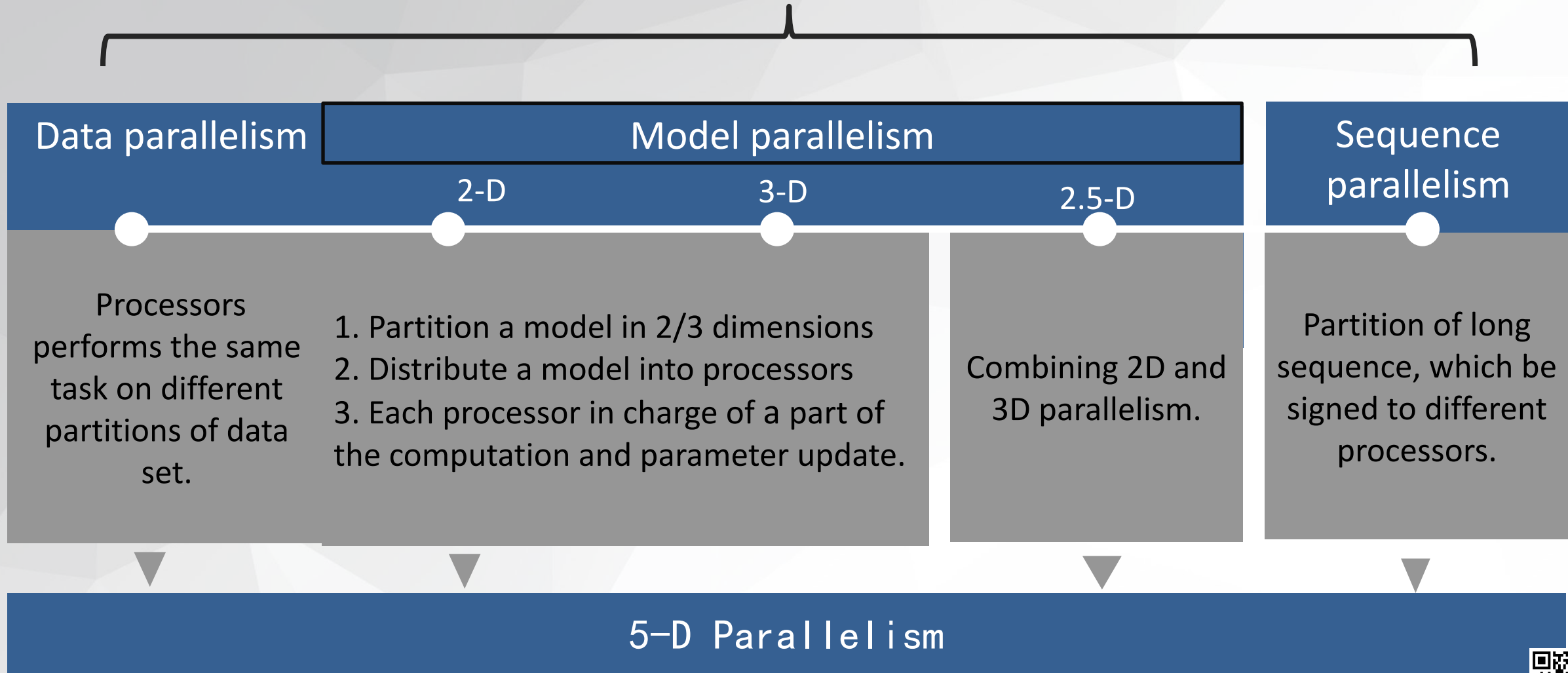
NVLAMB adds the necessary tweaks to LAMB version 1, to ensure correct convergence. The algorithm is as follows:





# Future work: 5-D Parallelism

All concluded in Colossal-AI System





# Colossal-AI System (Open-Source)

## Thanks !



<https://github.com/hpcaitech/ColossalAI>