Large-Batch Training for LSTM and Beyond

Yang You[1] (advised by James Demmel)

with James Demmel[1], Jonathan Hseu[2], Cho-Jui Hsieh[2,3], Kurt Keutzer[2], Chris Ying[2]

UC Berkeley[1], Google Brain[2], UCLA[3]

Outline

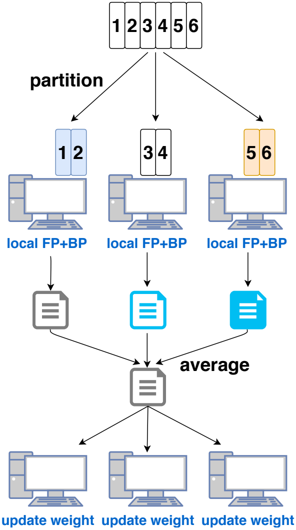
- **Problems in Distributed Deep Learning**
- Our Approach
- Experimental Results

Sync Mini-Batch SGD (Stochastic Gradient Descent)

- 1. Take B data points each iteration
- 2. Compute gradients of weights based on B data points
- 3. Update the weights: $x = x - \eta \times g$

- x: variables or weights (matrices or tensors)
- B: batch size (integer, e.g. 128)
- η: learning rate (a scalar, e.g. 0.01)
- g: gradients to the loss function (matrices or tensors)

Data-Parallelism SGD



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

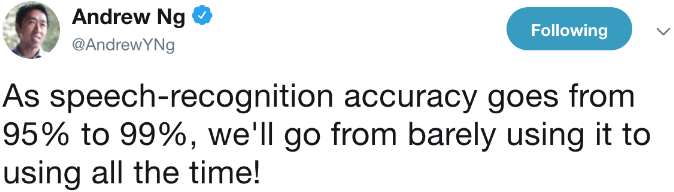
4. get the average of all the local gradient and send a copy of global gradient to each node

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

- Increase parallelism = increase the global data batch size

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

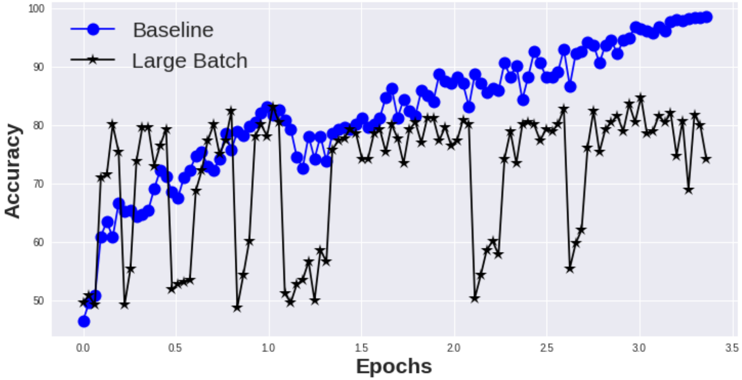
- 1000-class ImageNet dataset by AlexNet
  - 58% accuracy in 100 epochs
- 1000-class ImageNet dataset by ResNet-50
  - 76.3% accuracy in 90 epochs



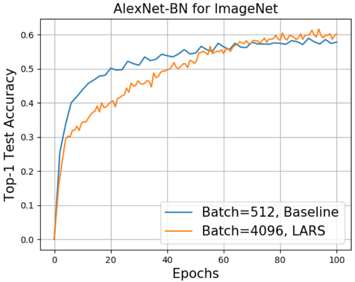
- The final 1% accuracy is very important but very hard to achieve

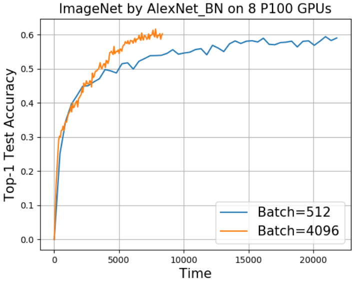
Difficulties of Large-Batch Training

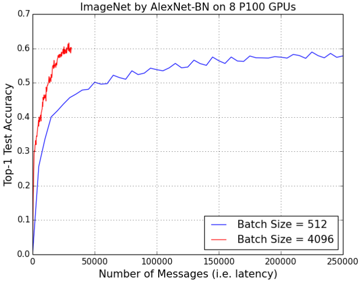
- **Large-Batch Training Loses Accuracy**
  - Even the training can be very fast
    - The solution is very bad

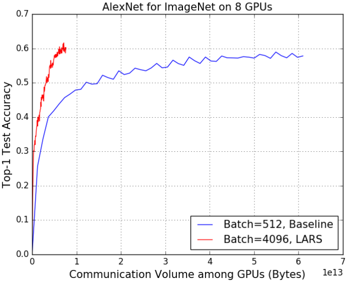


Our early success (large-batch training algorithm: LARS)









How to auto-tune when we scale batch size (B)?

- It is annoying to tune parameters every time we change the batch size

How to save energy?

Emma Strubell
@strubell

Are you interested in deep learning for NLP but also concerned about the CO2 footprint of training? You should be! Excited to share our work "Energy and Policy Considerations for Deep Learning in NLP" at @ACL2019_Italy! With @ananya__g and @andrewmccallum. Preprint coming soon.

Consumption	**CO_2e (lbs)**
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model	
SOTA NLP model (tagging)	13
w/ tuning & experimentation	33,486
Transformer (large)	121
w/ neural architecture search	394,863

8:27 AM · May 17, 2019 · Twitter Web App

1.1K Retweets **2.5K** Likes

Scaling on Various Models and Applications?

- Current Large-Batch Training is focused on CNN-based applications
  - How about RNN applications like LSTM (Long Short-Term Memory)?
- If we fix the dataset (e.g. ImageNet)
  - Can we scale on different models?

- CNN: Convolutional Neural Network
- RNN: Recurrent Neural Network

Outline

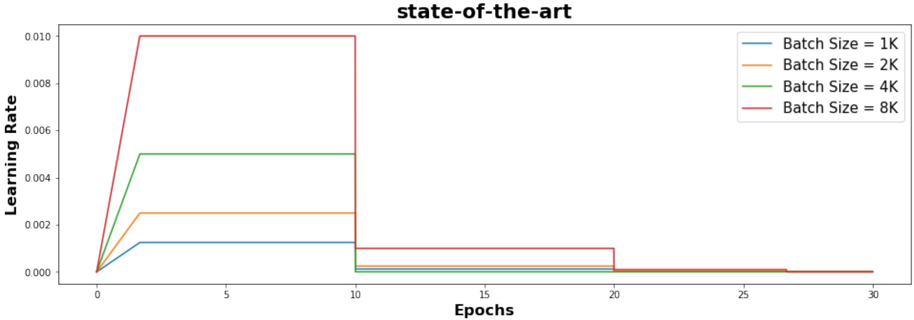
- Problems in Distributed Deep Learning
- **Our Approach**
- Experimental Results

Previous effective techniques (recipe of Goyal et al.)

- Control the learning rate (η) for large-batch training
- Linear Scaling[1]
  - if we 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[2]
  - start from a small η, increase η in a few epochs
    - avoid the divergence in the beginning
- Manual learning rate decay[3]
  - e.g. decay the η by 1/10 at 30th, 60th, 80th epoch
    - to stabilize the learning in the final stage

[1] Alex Krizhevsky, *One weird trick for parallelizing convolutional neural networks*, 2014 (Google Report)
[2] Goyal et al, *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, 2017 (Facebook Report)
[3] He et al, *Deep Residual Learning for Image Recognition*, CVPR 2017

Previous effective techniques (recipe of Goyal et al.)

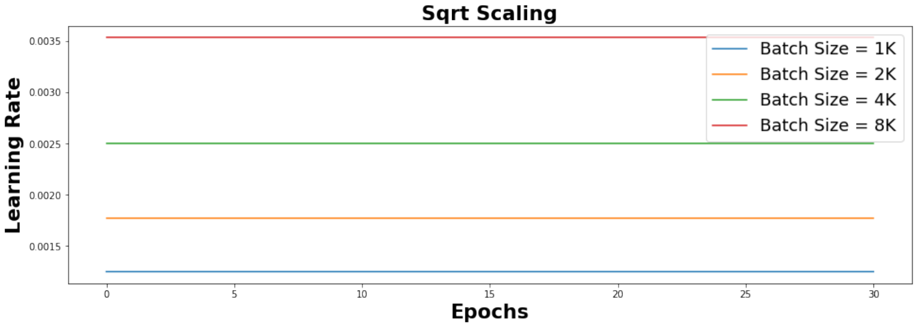


- An example for 30-epoch MNIST Training

Sqrt Learning Rate (η) Scaling

- if we increase B to kB, then we increase η by $\sqrt{k}$ times
  - not proposed by us, but we are the first to make it work
- Why do this? to keep the variance of the gradient estimator constant
- How to make it work? LEGW (Linear Epoch Gradual Warmup)

After adding optimization 1



- An example for 30-epoch MNIST Training

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?

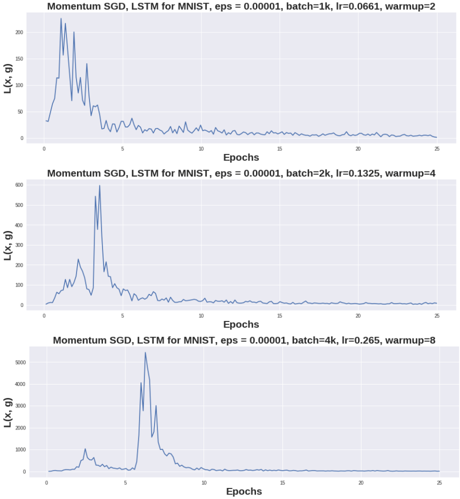
Why LEGW works?

- 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

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

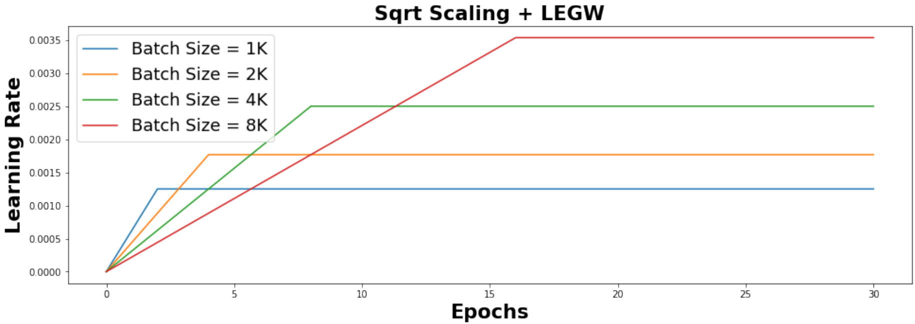
- η^* is inversely proportional to $L(x,g)$
- it is hard to get $L(x,g)$ since $\nabla^2 f(x)$ involves all the training samples
- we approximate $L(x,g)$ using a batch of data and compute the Hessian-vector product by finite difference

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

After adding optimization 2



- An example for 30-epoch MNIST Training

Learning Rate Decay

- Auto-tuning approach: AdaGrad[4]
  - use the sum of all historical gradients to decay η ($\frac{\eta}{\sqrt{\sum_t g_t \odot g_t}}$)
  - easily out of control at runtime by vanishing and exploding gradients
- State-of-the-art: discrete staircase decay
  - **a kind of manual tuning**
  - ResNet-50: reduce η by a factor of 10 at 30th, 60th, and 80th epoch[5]
  - ResNet-101: reduce η by factor of 10 at 50th and 100th epoch[6]
- Other commonly-used manually-tuning approach
  - **Needs to tune hyper-parameters**
  - Exponential decay
  - Polynomial decay

[4] Duchi et al, *Adaptive subgradient methods for online learning and stochastic optimization*

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

[6] Mu Li, *Scaling Distributed Machine Learning with System and Algorithm Co-design*

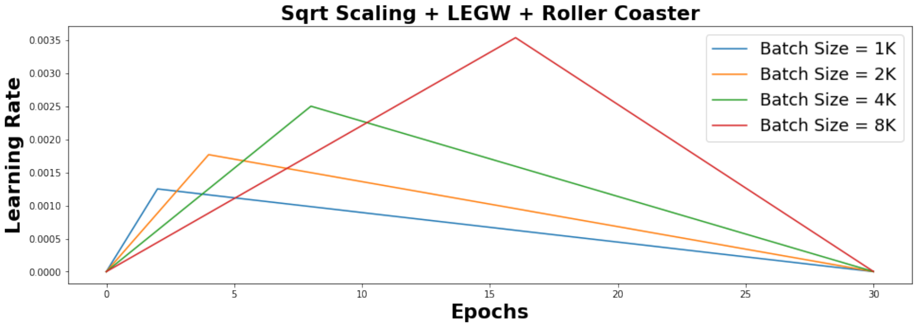
Roller Coaster Decay

- an automatic way to decay η
- use it after the warmup stage:

$$\eta = max\{\frac{(T-t)}{(1-w/E)\times T} \times \sqrt{\frac{B}{B_0}}\eta_0, \hat{\eta}\}$$

- B_0: the batch size of the baseline
- B: the target batch size
- η_0: the learning rate of the baseline
- t: the number of iterations we have finished
- T: the total number of iterations we need to finish
- $\hat{\eta}$: lower bound of η
  - no need to tune $\hat{\eta}$, use 10^{-6} as the default

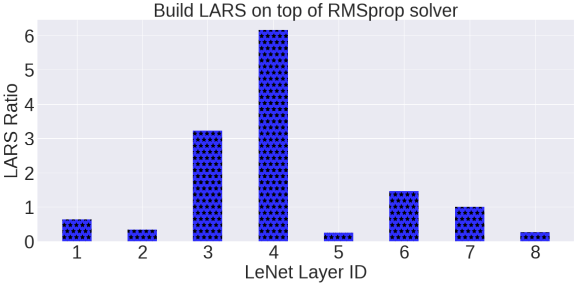
After adding optimization 3



- An example for 30-epoch MNIST Training

Dynamic Per-Layer Stabilized Learning

- Previous work: Layer-wise Adaptive Rate Scaling (LARS)[7]
  - use the trust ratio ($|w|/|g|$) to update η at runtime
  - it builds on top of Momentum SGD
  - can we apply it to adaptive solvers like RMSprop (Hinton, 2014)?
- Adding trust ratio to RMSprop (B=8K)
  - before: 2.8% error rate; after: 21.8% error rate
  - reason: some of the ratios are too large while some are too small

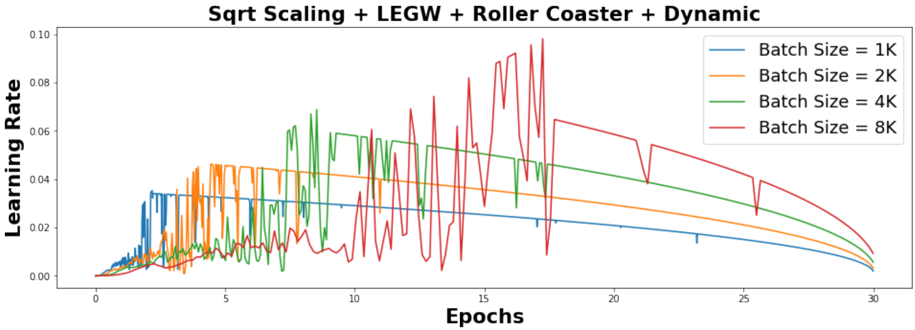


[7] You et al., *Scaling SGD Batch Size to 32K for ImageNet Training*, 2017

Dynamic Per-Layer Stabilized Learning

- Adding a dynamic lower bound and upper bound to trust ratio
- Adding trust ratio with bound to RMSprop (B=8K)
  - before: 2.8% error rate; after: 1.0% error rate

After adding optimization 4



- An example for 30-epoch MNIST Training

Dynamic Adaptive-Tuning Engine (DATE)

Input:

n labeled data points (x_i, y_i) for training;
Another k labeled data points $(\hat{x}_j, \hat{y}_j)$ for testing;
$i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, k\}$;
A baseline with Batch Size B_0, learning rate η_0, warmup epochs w_0 and total number of iterations I_0;
A target large batch size B;

Output:

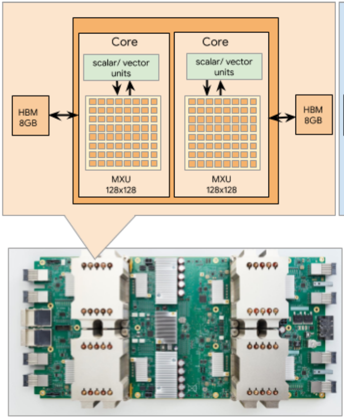
Trained Model of large batch B;
Test Accuracy of large batch B

The warmup epochs $w = \frac{B w_0}{B_0}$
The number of iterations $I = \frac{B_0 I_0}{B}$
for $i \in 1 : I$ **do**
  $E = \frac{iB}{n}$ (the current epoch)
  if $E < w$ **then**
    $\eta = \frac{E}{w}\sqrt{\frac{B}{B_0}}\eta_0$ or $(\frac{E}{w})^2\sqrt{\frac{B}{B_0}}\eta_0$
  else
    $\eta = max\{\frac{(I-i)}{(1-w/E)\times I} \times \sqrt{\frac{B}{B_0}}\eta_0, 10^{-6}\}$
  L = {the number of layers}
  for $j \in 1 : L$ **do**
    w = {the weight of layer-j}
    g = {the gradient of layer-j}
    if $||g||_2 == 0$ *or* $||w||_2 == 0$ **then**
      r = min{max{1.0, lower_limit}, upper_limit}
    else
      r = min{max{$\frac{||w||_2}{||g||_2}$, lower_limit}, upper_limit}
    $\eta = r\eta$ (runtime correction)
    apply_gradient_update(w, g, η) based on the optimizer (SGD, momentum, AdaGrad, or RMSProp)

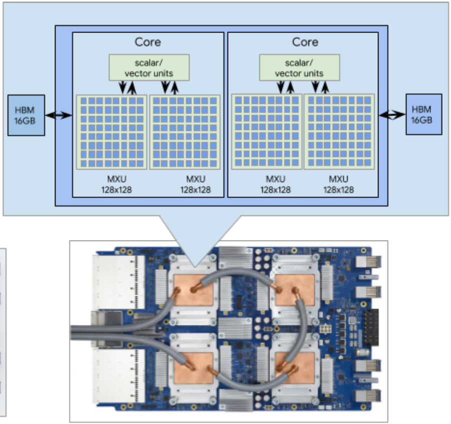
Outline

- Problems in Distributed Deep Learning
- Our Approach
- **Experimental Results**

TPU (Tensor Processing Units)



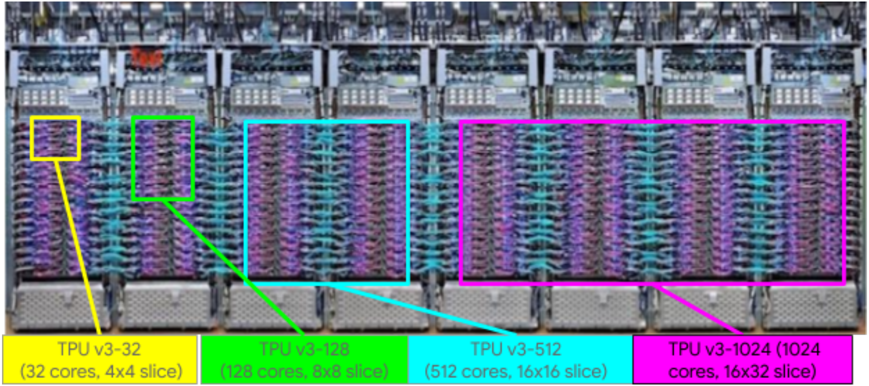
TPU v2 - 4 chips, 2 cores per chip



TPU v3 - 4 chips, 2 cores per chip

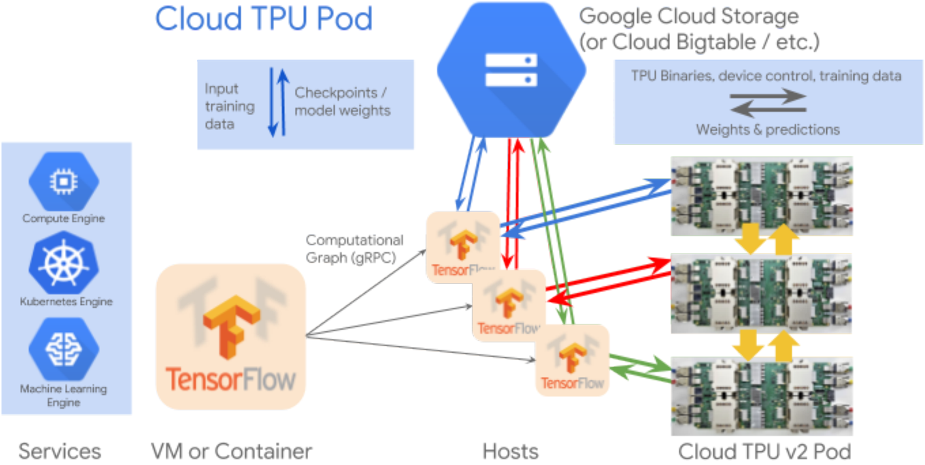
- TPU v2: 180 Tflops; 64 GB High Bandwidth Memory (HBM)
- TPU v3: 420 Tflops; 128 GB High Bandwidth Memory (HBM)

TPU Pod



- You can configure your own supercomputer!

TPU Pod on Cloud



- How to use it on Google Cloud?

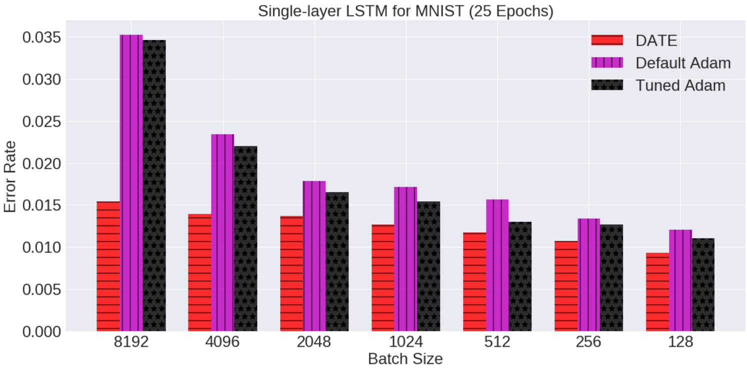
Datasets/Applications in our experiments

Table 1: The applications we used to evaluate our method.

Model	Dataset	Type	Samples	Metric & Reference
LeNet	MNIST	Small	60K/10K	99.2% accuracy[8]
1-layer LSTM	MNIST	Small	60K/10K	98.7% accuracy[9]
PTB-small	PTB	Medium	930K/82K	116 perplexity[10]
PTB-large	PTB	Medium	930K/82K	78 perplexity[11]
GNMT	wmt16	Large	3.5M/3K	21.8 BLEU[12]
ResNet50	ImageNet	Large	1.3M/5K	75.3% accuracy[13]

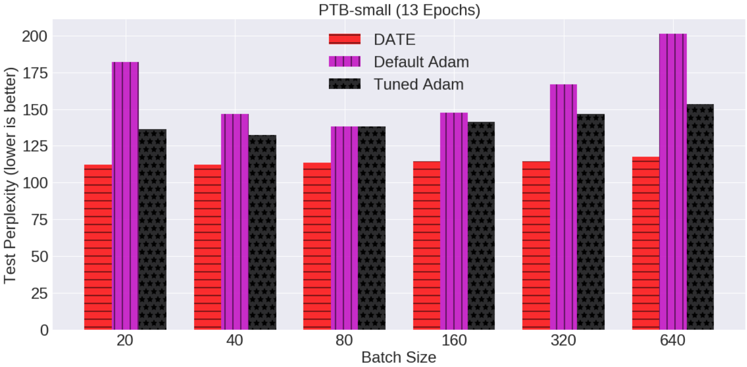
[8] https://github.com/tensorflow/models/tree/master/official/mnist
[9] https://medium.com/machine-learning-algorithms
[10] https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py
[11] https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py
[12] https://github.com/mlperf/training/tree/master/rnn_translator
[13] https://github.com/KaimingHe/deep-residual-networks

Scalable Auto-Tuning Approach



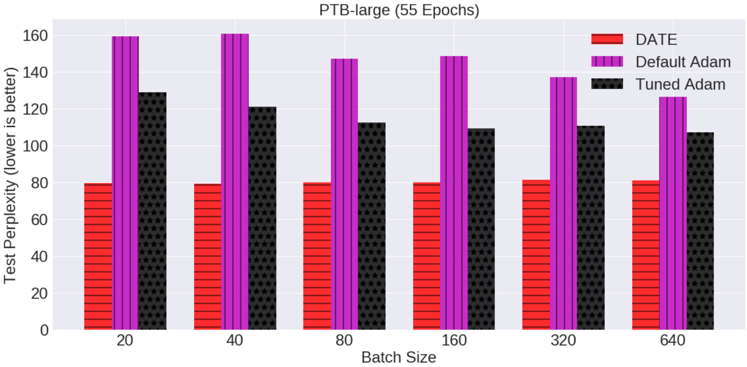
- Our approach DATE does not need tuning

Scalable Auto-Tuning Approach



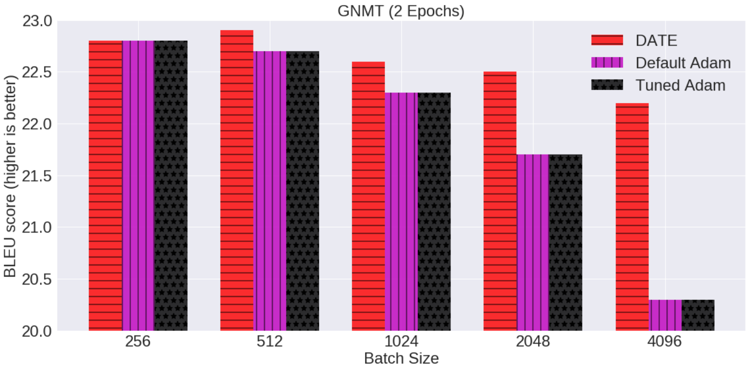
- Our approach DATE does not need tuning

Scalable Auto-Tuning Approach



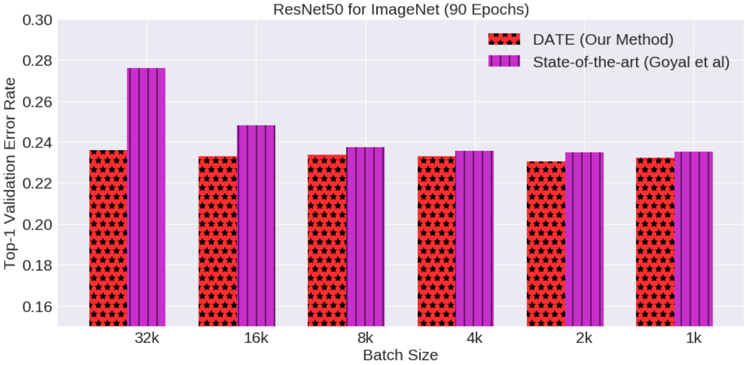
- Our approach DATE does not need tuning

Scalable Auto-Tuning Approach



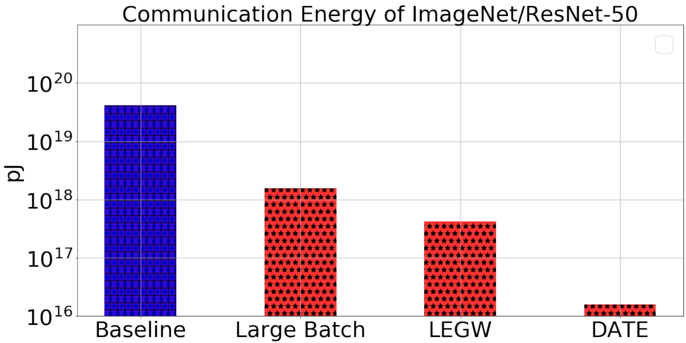
- Our approach DATE does not need tuning

Scalable Auto-Tuning Approach



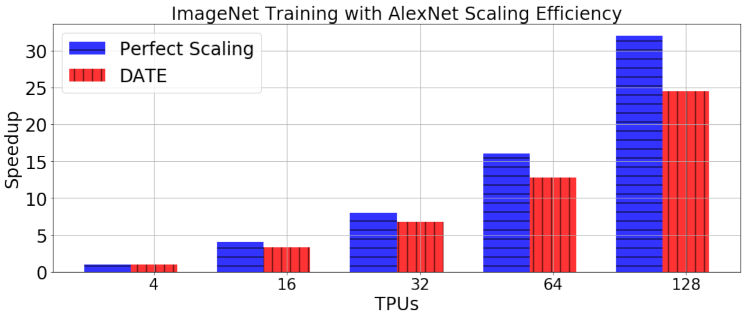
- Our approach DATE does not need tuning

Energy-Efficient Communication



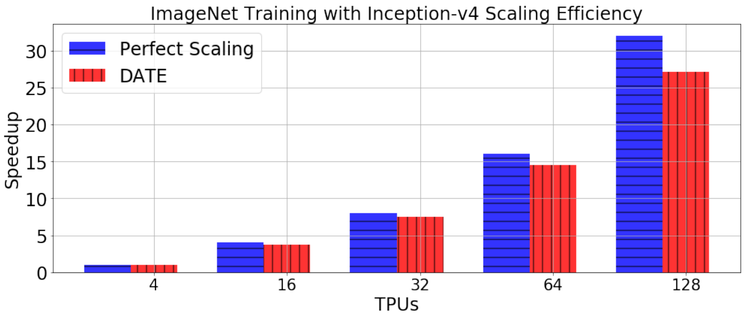
- B of the baseline: 256
- B of the large-batch: 32K
- the baseline tunes the hyper-parameters 100 times

Scaling on Different Models



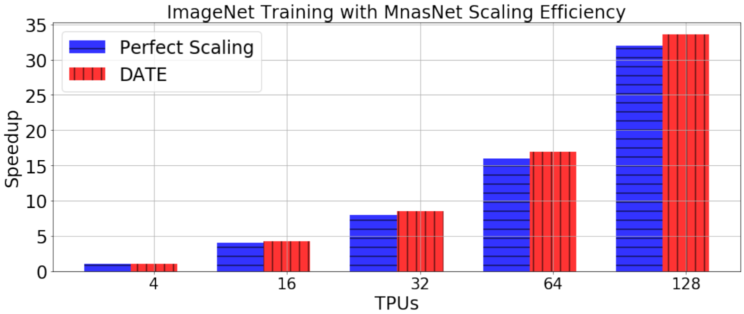
- 76.66% scaling efficiency

Scaling on Different Models



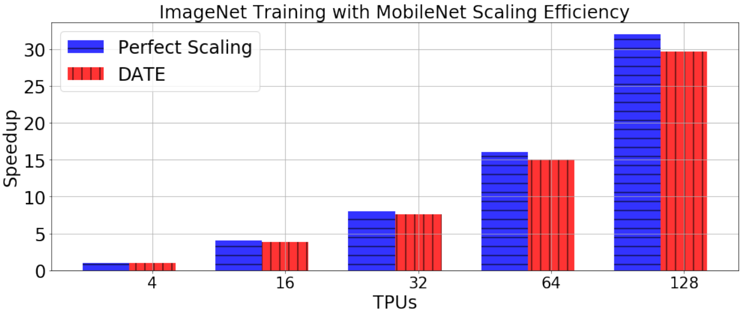
- 84.76% scaling efficiency

Scaling on Different Models



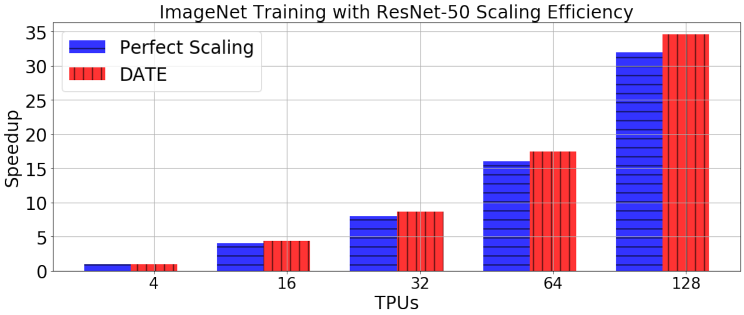
- 100.05% scaling efficiency

Scaling on Different Models



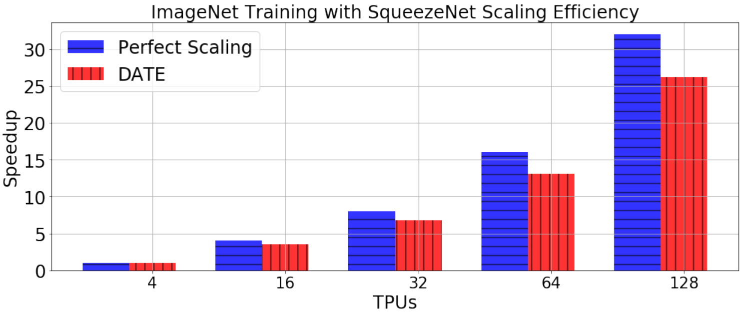
- 92.82% scaling efficiency

Scaling on Different Models



- 100.08% scaling efficiency

Scaling on Different Models



- 81.89% scaling efficiency

Our early success (featured by Google product release)

AI & MACHINE LEARNING

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

- ImageNet/ResNet-50 training in 1 minute (no tuning)
- Reduce BERT training time from 3 days to 76 minutes (no tuning)