Large Batch Training
(a review)

Mustafa Mustafa

Outline

- Why do we need distributed training?
- What are the challenges of training with large batches?
- Attempts at explaining the large batch generalization gap.
- Attempts to fix large batch training, practical advice.

Papers reviewed in this talk

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, ICLR 2017, Keskar et al, arXiv:1609.04836

Sharp Minima Can Generalize For Deep Nets, PMLR 2017, Dinh et al, arXiv:1703.04933

Train longer, generalize better: closing the generalization gap in large batch training of NN, NIPs 2017, Hoffer et al. arXiv:1705.08741

Why Does Large Batch Training Result in Poor Generalization? Takase et al, Neural Computation 30, 2005–2023 (2018)

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, Goyal et al. arXiv:1706.02677

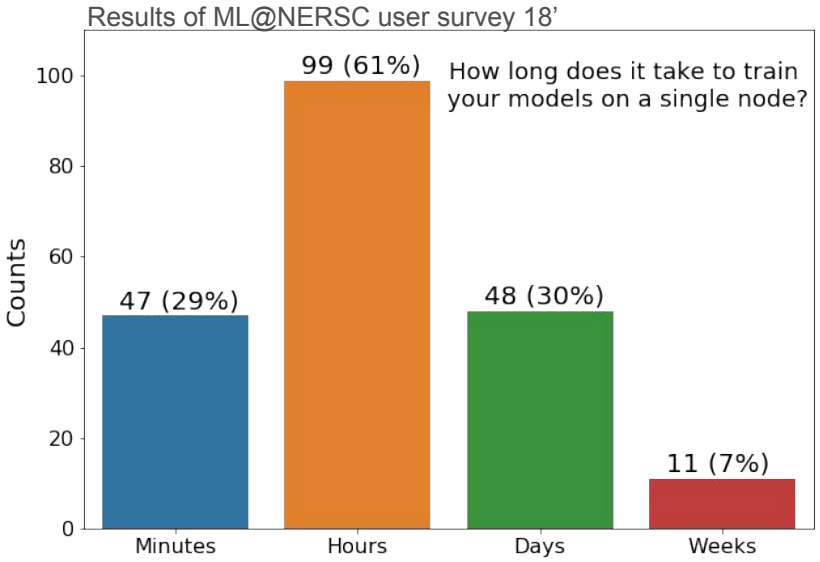
Don't Decay the Learning Rate, Increase the Batch Size, ICLR 2018, Smith et al, arXiv:1711.00489

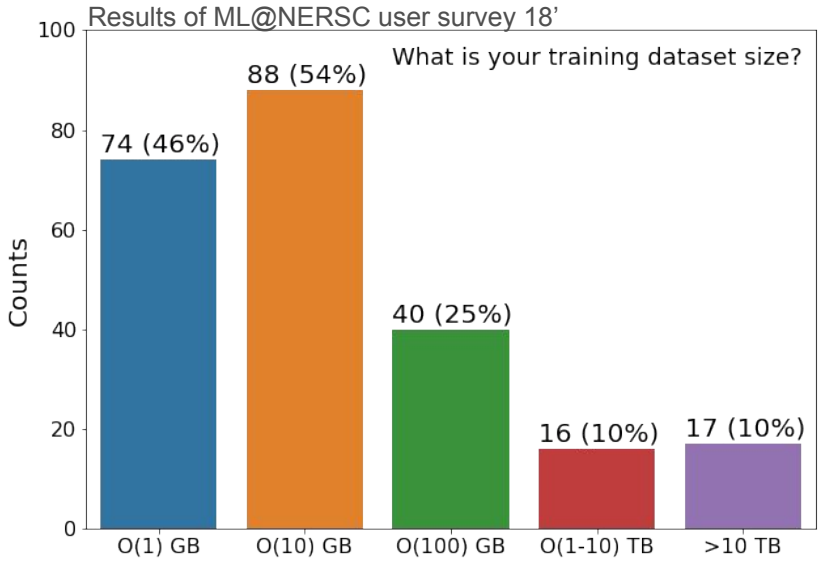
Large batch size training of neural networks with adversarial training and second-order information, Yao et al. arXiv:1810.01021

Large Batch Training of Convolutional Networks, ICLR 2018, You et al. arXiv:1708.03888

Paper in this “sky” color are not covered comprehensively.

Motivation for Scalable Deep Learning

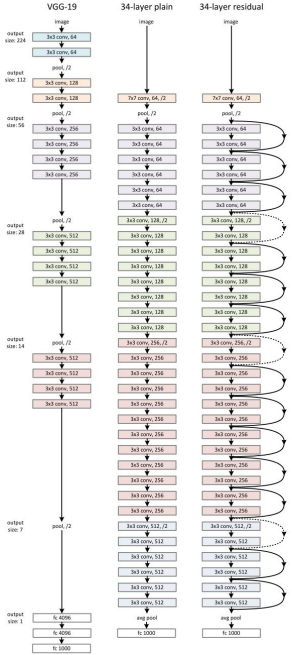




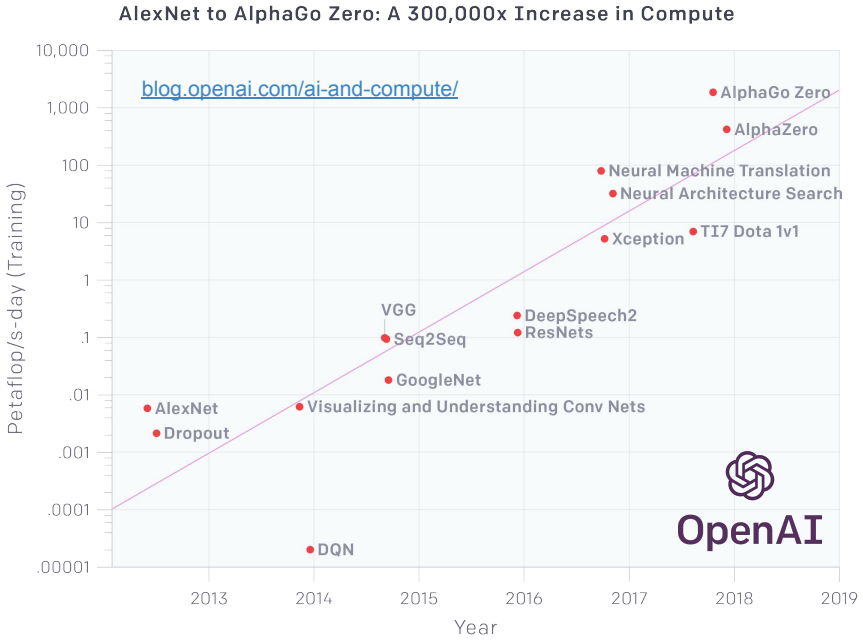
- Rapid prototyping/model evaluation
- Problem scale

- Volume of scientific datasets can be large
- Scientific datasets can be complex (multivariate, high dimensional)

Motivation for Scalable Deep Learning

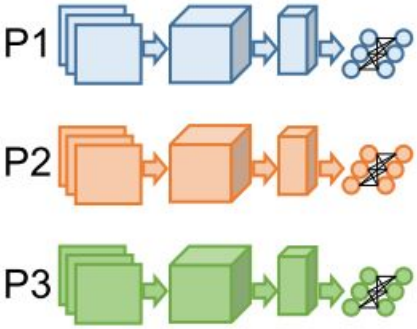


- Models get bigger and more compute intensive as they tackle more complex tasks



"... total amount of compute, in petaflop/s-days, that was used to train selected results ... A petaflop/s-day (pfs-day) = ... 10^{15} neural net operations per second for one day, or a total of about 10^{20} operations." -- OpenAI Blog

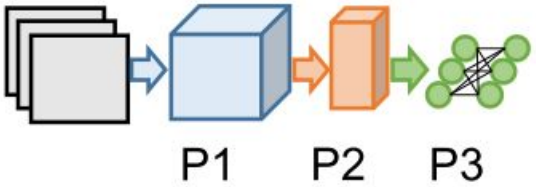
Distributed training



(a) Data Parallelism

(b) Model Parallelism

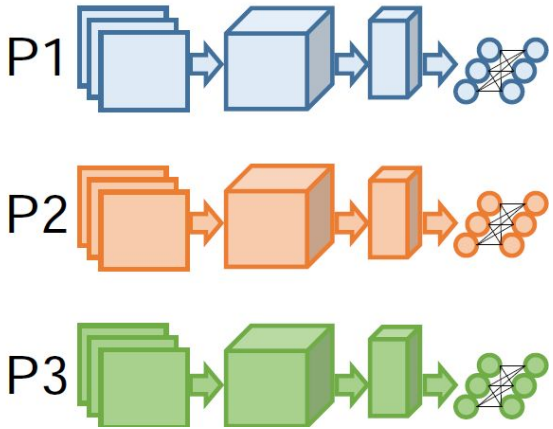
Ben-Nun and Hoefler arXiv:1802.09941



(c) Layer Pipelining

Data-parallel Training

- applies to Stochastic Gradient Descent-type algorithms
  - each node takes a data batch and computes model updates independently
  - these updates are then collectively summed and applied to the local model



Ben-Nun and Hoefler arXiv:1802.09941

Reminder: Stochastic Gradient Descent

$$w_{t+1} = w_t - \eta * \frac{1}{B} \sum_{i=1}^{B} \nabla L(x_i, w_t)$$

Notation:

N is total sample size

B is batch-size

η is learning rate

Δw is the parameter update in one gradient descent step

$\mathbf{g}_i$ is gradient on one sample

Large batch training and scaling learning rate

- In data parallelism with fully synchronous SGD one uses N workers to processes N batches of data
- One would like to scale the learning rate accordingly to accelerate the convergence
- There are two camps on how to scale the learning rate:

Large batch training and scaling learning rate

- In data parallelism with fully synchronous SGD one uses N workers to processes N batches of data
- One would like to scale the learning rate accordingly to accelerate the convergence
- There are two camps on how to scale the learning rate:
  - Linear scaling (e.g. AlexNet, Bottou et al. arXiv:1606.04838 , Goyal et al. arXiv:1706.02677): η → N * η

    The motivation can be seen if one unrolls two steps SGD:

$$w_{t+2} = w_t - \eta * \frac{1}{B}\left(\sum_{i=1}^{B} \nabla L(x_i, w_t) + \sum_{j=1}^{B} \nabla L(x_j, w_{t+1})\right)$$

$$w_{t+1} = w_t - \eta_2 * \frac{1}{2 * B} \sum_{i=1}^{2B} \nabla L(x_i, w_t)$$

    where $\eta_2 = 2 * \eta$, under the assumption that $\nabla L(x_j, w_{t+1}) \approx L(x_j, w_t)$

Large batch training and scaling learning rate

- In data parallelism with fully synchronous SGD one uses N workers to processes N batches of data
- One would like to scale the learning rate accordingly to accelerate the convergence
- There are two camps on how to scale the learning rate:
  - Linear scaling (e.g. AlexNet, Bottou et al. arXiv:1606.04838 , Goyal et al. arXiv:1706.02677): η → N * η

    The motivation can be seen if one unrolls two steps SGD:

    $$w_{t+2} = w_t - \eta * \frac{1}{B}\left(\sum_{i=1}^{B} \nabla L(x_i, w_t) + \sum_{j=1}^{B} \nabla L(x_j, w_{t+1})\right)$$

    $$w_{t+1} = w_t - \eta_2 * \frac{1}{2 * B} \sum_{i=1}^{2B} \nabla L(x_i, w_t)$$

    where $\eta_2 = 2 * \eta$, under the assumption that $\nabla L(x_j, w_{t+1}) \approx L(x_j, w_t)$
  - sqrt-scaling (e.g. AlexNet, You et al. arXiv:1708.03888, Hoffer et al. arXiv:1705.08741): η → sqrt(N) * η

    Motivated by the observation that the variance of the gradient scales with 1/batch-size:

    $$\operatorname{cov}(\Delta \mathbf{w}, \Delta \mathbf{w}) \approx \frac{\eta^2}{B}\left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i \mathbf{g}_i^{\top}\right)$$

Large batch training and scaling learning rate

- In data parallelism with fully synchronous SGD one uses N workers to processes N batches of data
- One would like to scale the learning rate accordingly to accelerate the convergence
- There are two camps on how to scale the learning rate:
  - Linear scaling (e.g. AlexNet, Bottou et al. arXiv:1606.04838 , Goyal et al. arXiv:1706.02677): η → N * η

    The motivation can be seen if one unrolls two steps SGD:

    $$w_{t+2} = w_t - \eta * \frac{1}{B}\left(\sum_{i=1}^{B} \nabla L(x_i, w_t) + \sum_{j=1}^{B} \nabla L(x_j, w_{t+1})\right)$$

    where $\eta_2 = 2 * \eta$, under the assumption that $\nabla L(x_j, w_{t+1}) \approx L(x_j, w_t)$
  - sqrt-scaling (e.g. AlexNet, You et al. arXiv:1708.03888, Hoffer et al. arXiv:1705.08741): η → sqrt(N) * η

    Motivated by the observation that the variance of the gradient scales with 1/batch-size:

    $$\mathrm{cov}\left(\Delta \mathbf{w}, \Delta \mathbf{w}\right) \approx \frac{\eta^2}{B}\left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i \mathbf{g}_i^{\top}\right)$$

In practice, we see anywhere between sub-sqrt (e.g.You et al. arXiv:1708.03888) to linear scaling (e.g. Goyal et al. arXiv:1706.02677)

Challenges with Large Batch Training

There are two main challenges with training with large batch and large learning rates:

1. Training with large learning rates is not stable in the initial stages of the training

   $\nabla L(w_{t+1}) \approx \nabla L(w_t)$ condition does not hold when the parameters are changing rapidly

Challenges with Large Batch Training

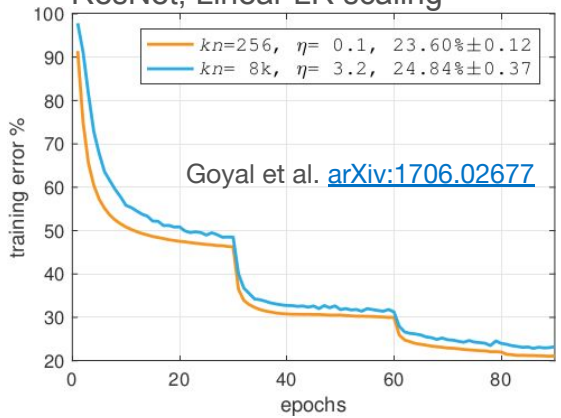
There are two main challenges with training with large batch and large learning rates:

1. Training with large learning rates is not stable in the initial stages of the training

   $\nabla L(w_{t+1}) \approx \nabla L(w_t)$ condition does not hold when the parameters are changing rapidly

2. A generalization gap appears: networks trained with small batches tend to generalize better

ResNet, Linear LR scaling



AlexNet You et al. arXiv:1708.03888

Batch	Base LR	accuracy,%
512	**0.02**	**60.2**
4096	0.16	58.1
4096	**0.18**	**58.9**
4096	0.21	58.5
4096	0.30	57.1
8192	0.23	57.6
8192	**0.30**	**58.0**
8192	0.32	57.7
8192	0.41	56.5

Explaining the generalization gap: Sharp minimizers?

Keskar et al, arXiv:1609.04836 observe:

"... large-batch methods tend to converge to sharp minimizers of the training function … (significant number of large positive eigenvalues in $\nabla^2 f(x)$) and tend to generalize less well.

In contrast, small-batch methods converge to flat minimizers … (numerous small eigenvalues of $\nabla^2 f(x)$)"

Explaining the generalization gap: Sharp minimizers?

Keskar et al, arXiv:1609.04836 observe:

"... large-batch methods tend to converge to sharp minimizers of the training function … (significant number of large positive eigenvalues in $\nabla^2 f(x)$) and tend to generalize less well.

In contrast, small-batch methods converge to flat minimizers … (numerous small eigenvalues of $\nabla^2 f(x)$)"

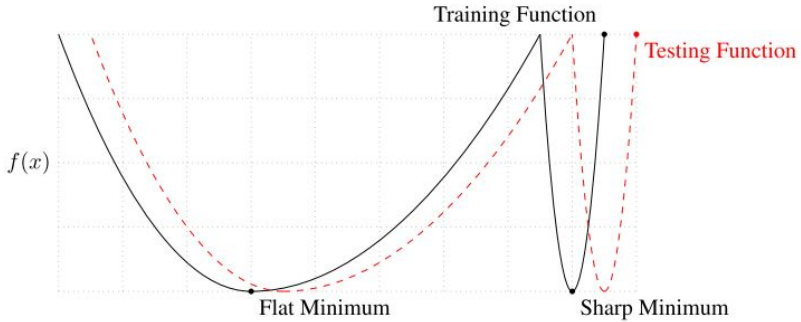


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

Explaining the generalization gap: Sharp minimizers?

Keskar et al, arXiv:1609.04836 observe:

"... large-batch methods tend to converge to sharp minimizers of the training function … (significant number of large positive eigenvalues in $\nabla^2 f(x)$) and tend to generalize less well.

In contrast, small-batch methods converge to flat minimizers … (numerous small eigenvalues of $\nabla^2 f(x)$)"

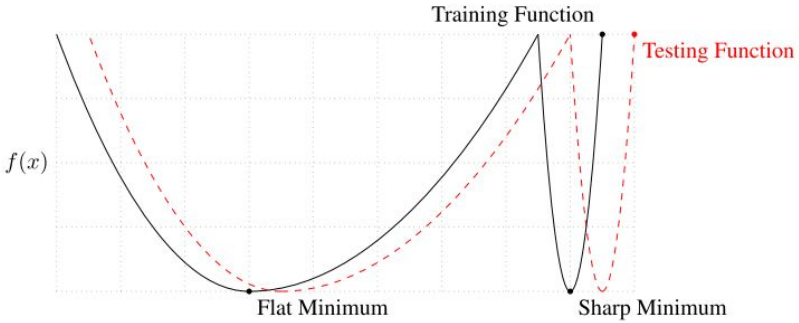
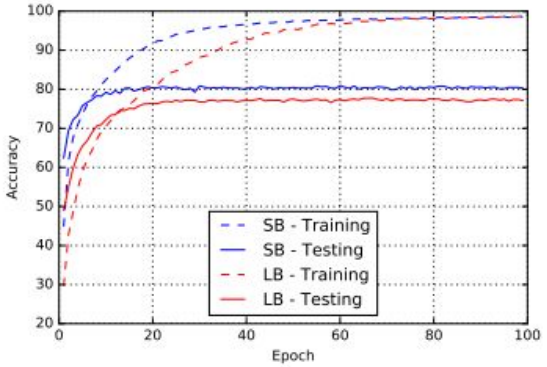


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)



(b) Network C_1

It is not a case of overfitting. So it can't be solved with early stopping.

Explaining the generalization gap: Sharp minimizers?

"We have observed that the loss function landscape of deep neural networks is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers"

Table 4: Sharpness of Minima in Random Subspaces of Dimension 100

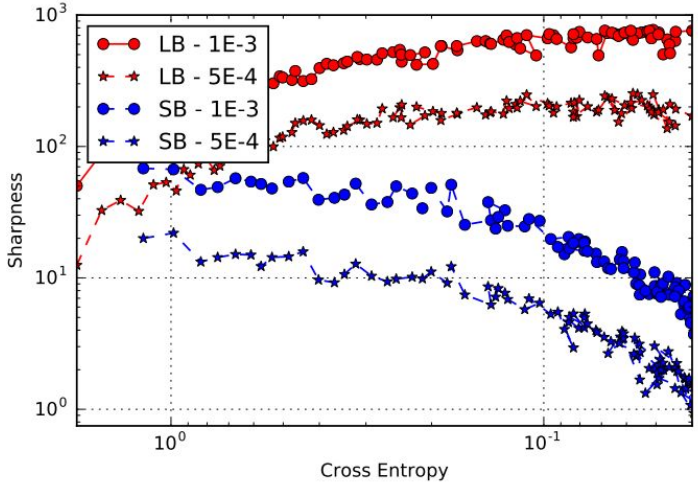
	$\epsilon = 10^{-3}$		$\epsilon = 5 \cdot 10^{-4}$	
	SB	LB	SB	LB
F_1	0.11 ± 0.00	9.22 ± 0.56	0.05 ± 0.00	9.17 ± 0.14
F_2	0.29 ± 0.02	23.63 ± 0.54	0.05 ± 0.00	6.28 ± 0.19
C_1	2.18 ± 0.23	137.25 ± 21.60	0.71 ± 0.15	29.50 ± 7.48
C_2	0.95 ± 0.34	25.09 ± 2.61	0.31 ± 0.08	5.82 ± 0.52
C_3	17.02 ± 2.20	236.03 ± 31.26	4.03 ± 1.45	86.96 ± 27.39
C_4	6.05 ± 1.13	72.99 ± 10.96	1.89 ± 0.33	19.85 ± 4.12

Explaining the generalization gap: Sharp minimizers?

"We have observed that the loss function landscape of deep neural networks is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers"

Table 4: Sharpness of Minima in Random Subspaces of Dimension 100

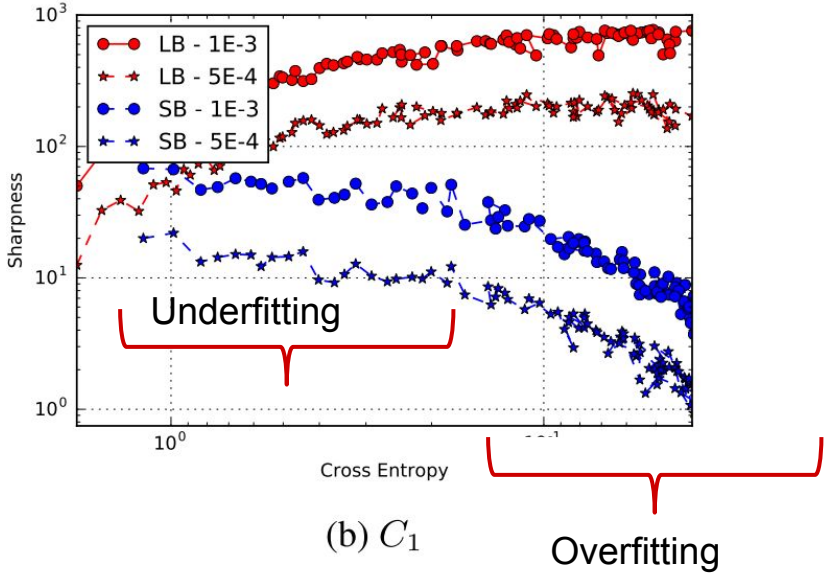
	$\epsilon = 10^{-3}$		$\epsilon = 5 \cdot 10^{-4}$	
	SB	LB	SB	LB
F_1	0.11 ± 0.00	9.22 ± 0.56	0.05 ± 0.00	9.17 ± 0.14
F_2	0.29 ± 0.02	23.63 ± 0.54	0.05 ± 0.00	6.28 ± 0.19
C_1	2.18 ± 0.23	137.25 ± 21.60	0.71 ± 0.15	29.50 ± 7.48
C_2	0.95 ± 0.34	25.09 ± 2.61	0.31 ± 0.08	5.82 ± 0.52
C_3	17.02 ± 2.20	236.03 ± 31.26	4.03 ± 1.45	86.96 ± 27.39
C_4	6.05 ± 1.13	72.99 ± 10.96	1.89 ± 0.33	19.85 ± 4.12



(b) C_1

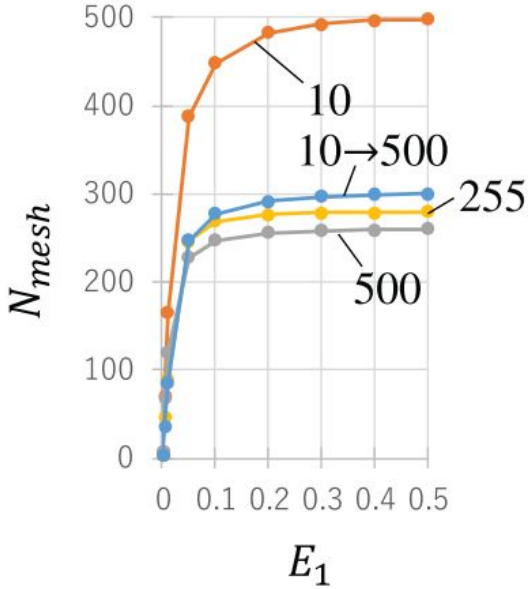
Explaining the generalization gap: Sharp minimizers?

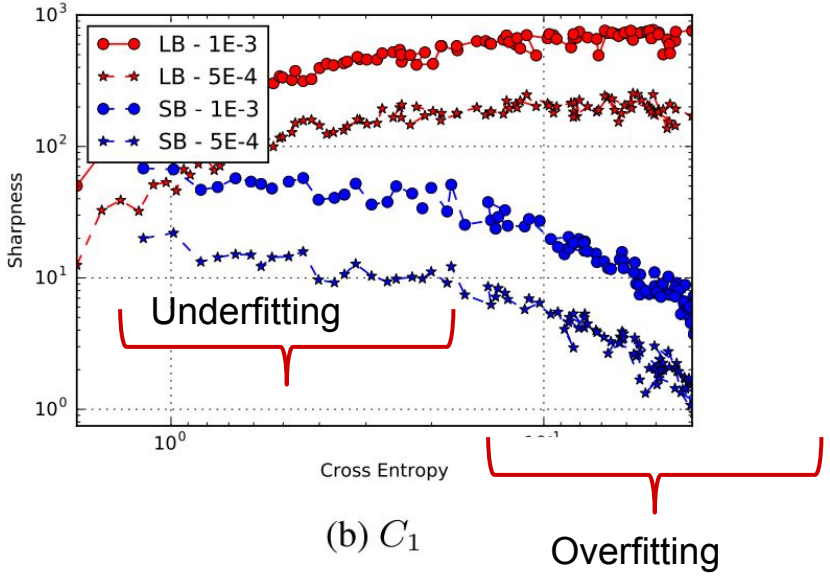
"We have observed that the loss function landscape of deep neural networks is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers"



Explaining the generalization gap: Sharp minimizers?

"We have observed that the loss function landscape of deep neural networks is such that large-batch methods are attracted to regions with sharp minimizers and that, unlike small-batch methods, are unable to escape basins of attraction of these minimizers"



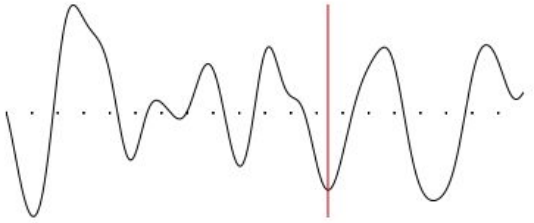


Why Does Large Batch Training Result in Poor Generalization? Takase et al, Neural Computation 30, 2005–2023 (2018)

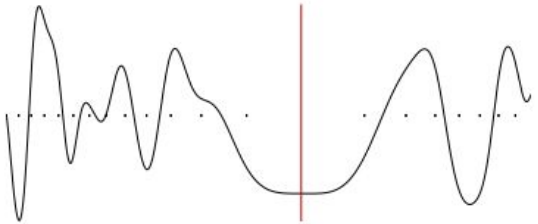
The fall of sharp minimizers picture?

Ding et al. “we demonstrate a simple observation. If we are allowed to change the parametrization of some function f, we can obtain arbitrarily different geometries without affecting how the function evaluates on unseen data. The same holds for reparametrization of the input space. The implication is that the correlation between the geometry of the parameter space (and hence the error surface) and the behavior of a given function is meaningless if not preconditioned on the specific parametrization of the model.”

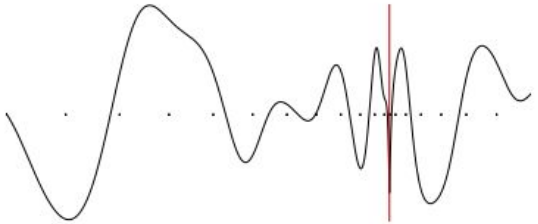
Sharp Minima Can Generalize For Deep Nets, PMLR 2017, Dinh et al, arXiv:1703.04933



(a) Loss function with default parametrization



(b) Loss function with reparametrization



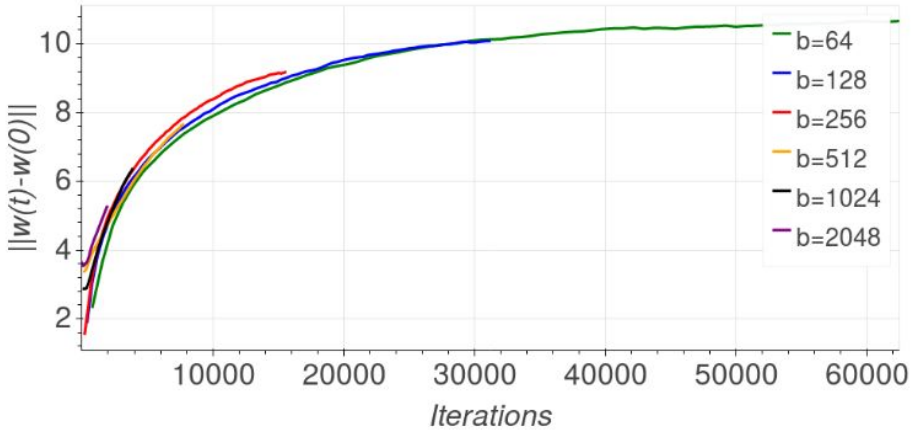
(c) Loss function with another reparametrization

Explaining the generalization gap: parameters diffusion?

Hoffer et al. show that the distance of weights from their point of initialization grows logarithmically:

$$\|\mathbf{w}_t - \mathbf{w}_0\| \sim \log t$$

they draw similarities of training deep NN to "random walk on a random landscape" statistical model which exhibits similar "ultra-slow" diffusion.

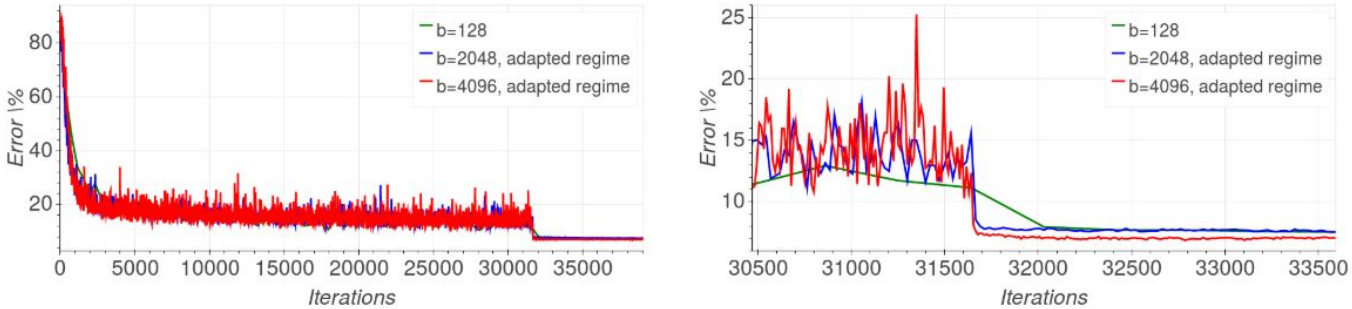


Explaining the generalization gap: parameters diffusion?

Hoffer et al. show that the distance of weights from their point of initialization grows logarithmically:

$$\|\mathbf{w}_t - \mathbf{w}_0\| \sim \log t$$

they draw similarities of training deep NN to "random walk on a random landscape" statistical model which exhibits similar "ultra-slow" diffusion.



(a) Validation error

(b) Validation error - zoomed

"Train longer, generalize better: closing the generalization gap in large batch training of NN", Hoffer et al. arXiv:1705.08741

Avoiding initial training instabilities

Learning rate warm-up:

The instability of initial training with large learning rates is remedied by having a learning rate warm-up phase. It has been shown that gradual linear warm-up works better than constant warm-up (Goyal et al. arXiv:1706.02677).

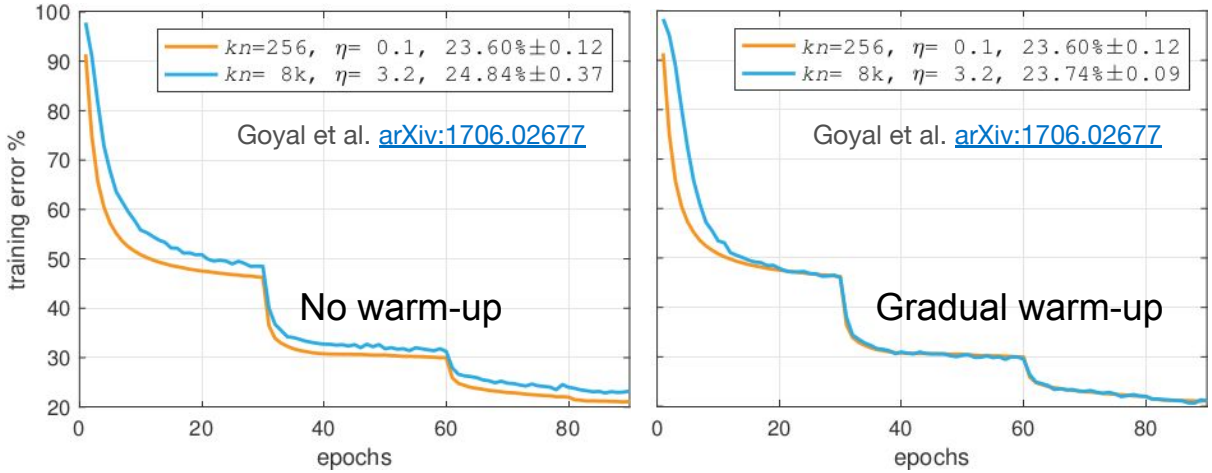
Start with small batch LR = η and linearly scale up to the target learning rate over a few epochs.

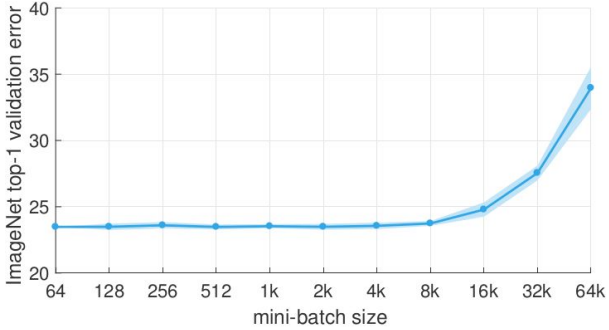
Some other works have used gradient clipping, which is typically active in the earlier stages, as an alternative to a warm-up period, see LARC version of You et al. arXiv:1708.03888 and Hoffer arXiv:1705.08741.

Closing the generalization gap

Linear warm up + LR linear scaling

Goyal et al. arXiv:1706.02677 have shown that linear warm-up over 5 epochs up to N * η followed by original learning rate decay schedule works pretty well for ResNet-50 on ImageNet up to batch-size=8k (using 256 GPUs). The paper also clarifies subtleties and common pitfalls in distributed training.





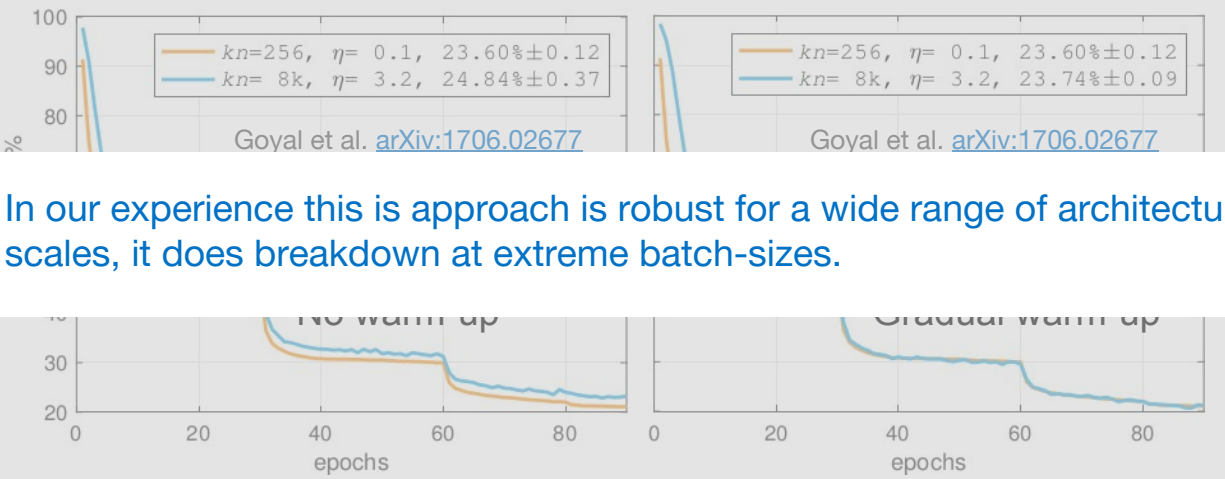
This scheme breaks down beyond batch-size = 8k for ResNet on ImageNet

In fact, the small and large batch loss curves are even matched.

Closing the generalization gap

Linear warm up + LR linear scaling

Goyal et al. arXiv:1706.02677 have shown that linear warm-up over 5 epochs up to N * η followed by original learning rate decay schedule works pretty well for ResNet-50 on ImageNet up to batch-size=8k (using 256 GPUs). The paper also clarifies subtleties and common pitfalls in distributed training.



In our experience this is approach is robust for a wide range of architectures and datasets for reasonable scales, it does breakdown at extreme batch-sizes.

This scheme breaks down beyond batch-size = 8k for ResNet on ImageNet

In fact, the small and large batch loss curves are even matched.

Closing the generalization gap

Adaptive learning rates for large batch, Layer-wise Adaptive Rate Control (LARS/LARC):

You et al. arXiv:1708.03888 observe that when the ratio between the norm of the layer weights and norm of the gradient update is too small the training becomes unstable. When it is too high, the weights don't get updated fast enough.

$$\frac{\left\|w^l\right\|}{\left\|\nabla L(w^l)\right\|}$$

Closing the generalization gap

Adaptive learning rates for large batch, Layer-wise Adaptive Rate Control (LARS/LARC):

You et al. arXiv:1708.03888 observe that when the ratio between the norm of the layer weights and norm of the gradient update ($\frac{\|w^l\|}{\|\nabla L(w^l)\|}$) is too small the training becomes unstable. When it is too high, the weights don't get updated fast enough.

They suggest introducing a layer-wise learning rate multiplier: $\lambda^l = \alpha \times \frac{||w^l||}{||\nabla L(w^l)||}$

Where α is a scaling factor dubbed "trust coefficient".

The SGD update becomes: $\triangle w_t^l = \eta * \lambda^l * \nabla L(w_t^l)$

Where η is the global learning rate.

A variation over this scheme is a clipping instead of scaling: $\triangle w_t^l = \min(\eta, \lambda^l) * \nabla L(w_t^l)$

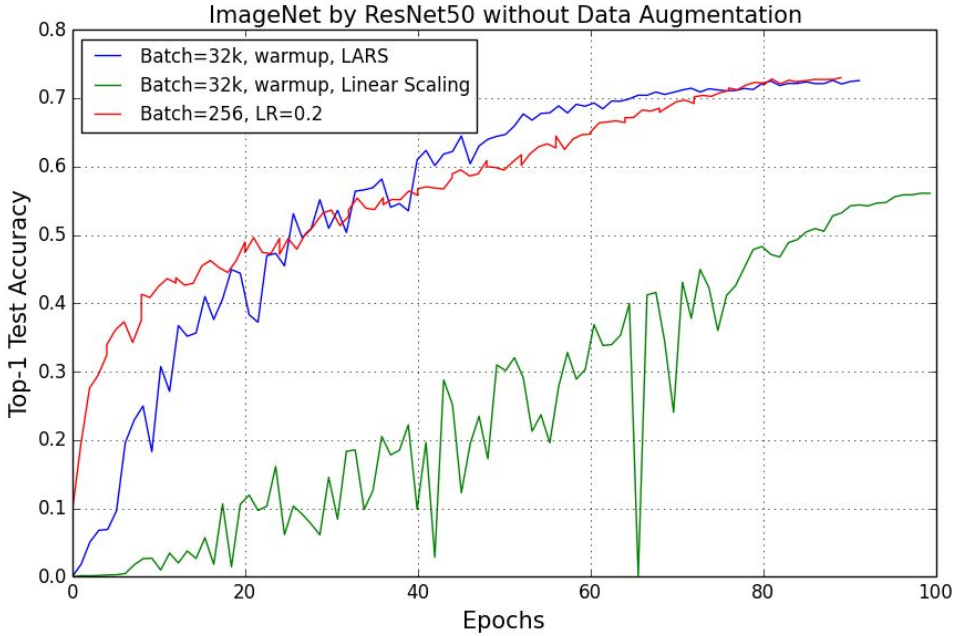
Closing the generalization gap

Adaptive learning rates for large batch, Layer-wise Adaptive Rate Control (LARS/LARC):

You et al. arXiv:1708.03888 observe that when the ratio between the norm of the layer weights and norm of the gradient update ($\frac{\|w^l\|}{\|\nabla L(w^l)\|}$) is too small the training becomes unstable. When it is too high, the weights don't get updated fast enough.

$$\lambda^l = \alpha \times \frac{||w^l||}{||\nabla L(w^l)||}$$

$$\triangle w_t^l = \eta * \lambda^l * \nabla L(w_t^l)$$



Closing the generalization gap

gradient, which can obstruct optimization. It is often stated that to reach the minimum of a strongly convex function we should decay the learning rate, such that (Robbins & Monro, 1951):

$$\sum_{i=1}^{\infty} \epsilon_i = \infty, \tag{1}$$

$$\sum_{i=1}^{\infty} \epsilon_i^2 < \infty. \tag{2}$$

"noise scale" $g = \bar{\epsilon}(\frac{N}{B} - 1)$

$$g \approx \epsilon N/B$$

Closing the generalization gap

gradient, which can obstruct optimization. It is often stated that to reach the minimum of a strongly convex function we should decay the learning rate, such that (Robbins & Monro, 1951):

$$\sum_{i=1}^{\infty} \epsilon_i = \infty, \quad (1)$$

$$\sum_{i=1}^{\infty} \epsilon_i^2 < \infty. \quad (2)$$

"noise scale" $g = \epsilon(\frac{N}{B} - 1)$

$$g \approx \epsilon N/B$$

Batch-size scaling:

Remember that we typically decay the learning rate as we train our models. Some works have suggested scaling the batch-size instead.

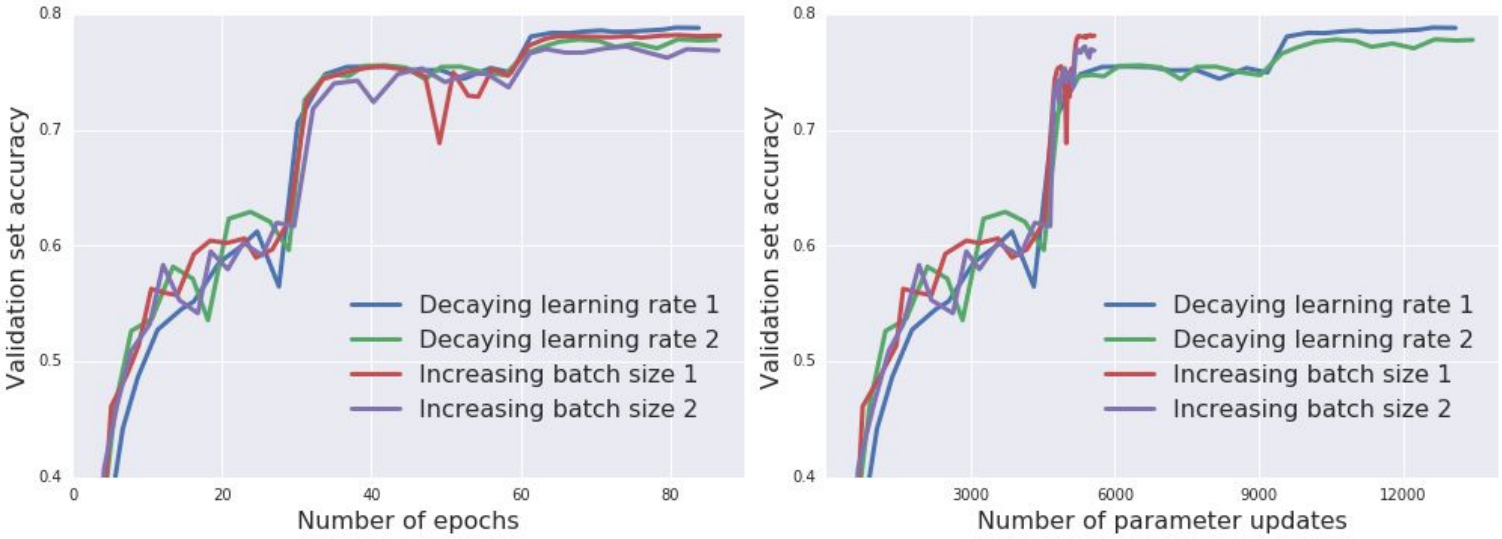
Smith et al. arXiv:1711.00489 use batch-size scaling to train Inception-ResNet V2 on ImageNet in 2500 parameter updates. Starting at batch-size 8k and scale to 80k!

Closing the generalization gap

Batch-size scaling:

Remember that we typically decay the learning rate as we train our models. Some works have suggested scaling the batch-size instead.

Smith et al. arXiv:1711.00489 use batch-size scaling to train Inception-ResNet V2 on ImageNet in 2500 parameter updates. Starting at batch-size 8k and scale to 80k!

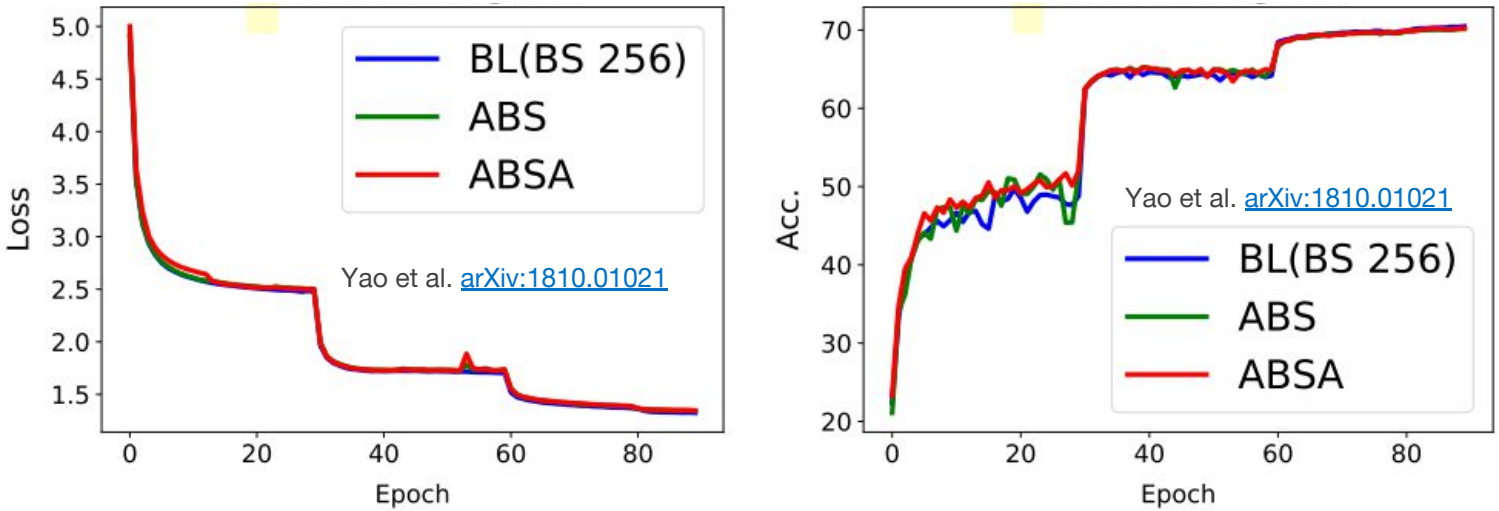


Inception-ResNet-V2 on ImageNet. Multiple runs to illustrate variance.

Closing the generalization gap

Adaptive batch-size scaling with adversarial training and 2nd-order information (ABSA):

Yao et al. arXiv:1810.01021 use 2nd-order information (to assess the loss surface curvature) to adaptively increase the batch-size. In addition, they employ adversarial training to regularize against “sharp-minima”. They show that this approach closes the generalization gap for a wide range of architectures on image classification tasks.



Example of their results using Adaptive Batch Scaling (ABS) and with Adversarial Training (ABSA). ResNet-18 on ImageNet dataset with up to 16k batch-size

Distributed training with batch-normalization

Data-parallel SGD training assumes that independence of the sample loss. However, batch-normalization computes the statistics along the minibatch. This breaks the independence assumption.

It has been shown that using a fixed batch-size to compute the batch-normalization statistic allows batch-normalization to scale well in distributed settings, see notes in Goyal et al. arXiv:1706.02677 and Ghost Batch Normalization in Hoffer et al. arXiv:1705.0874.

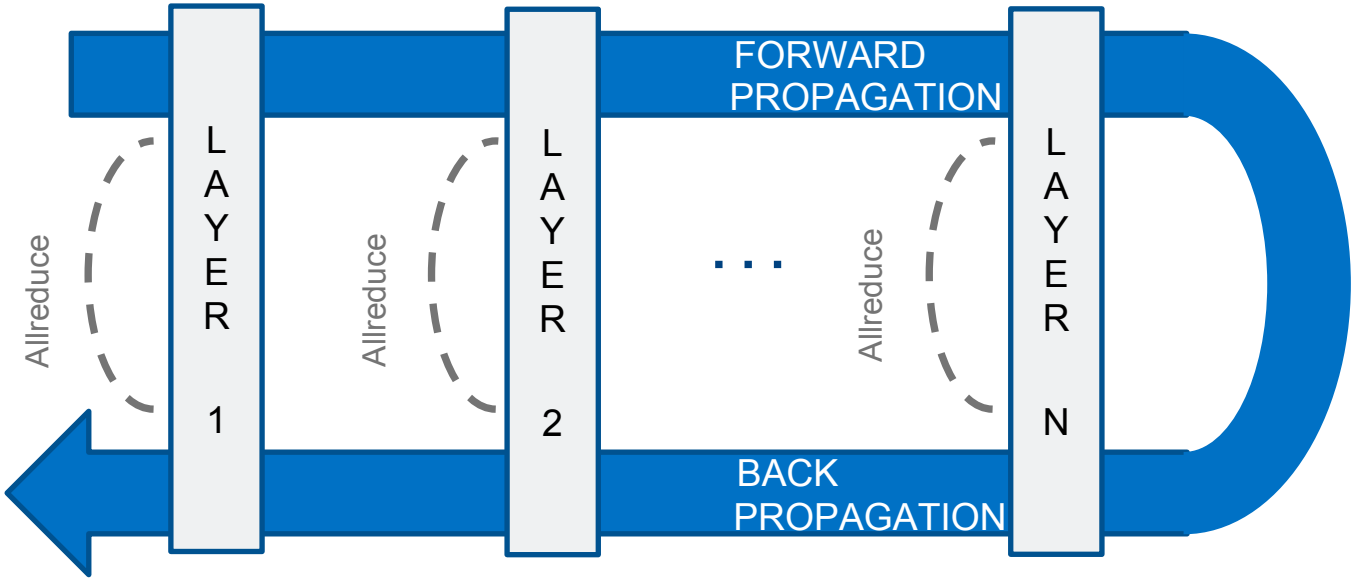
Large batch training takeaway

- Training with large batch-size requires learning rate scaling. Anywhere between sub-sqrt (e.g.You et al. arXiv:1708.03888) to linear scaling (e.g. Goyal et al. arXiv:1706.02677) have been used in practice.
- Training with large learning rates introduces instabilities in the initial stages of training. Gradual warm-up to target learning rate works well.
- Training with a large batch-size has a generalization gap w.r.t small batch-size
  - Linear warm-up + scaling LR works well for ~10x scaling of batch-size
  - Increasing the batch-size is an alternative approach to decaying the learning rate
  - Adaptive learning rate scaling methods like LARC is another alternative
- These methods are constantly pushing the limits of the largest batch-size we can use for training but they still don't eliminate the upper bound
- Use Ghost Batch Normalization (Hoffer et al. arXiv:1705.08741) for batch-norm in distributed settings.

backup

Data-parallel Training

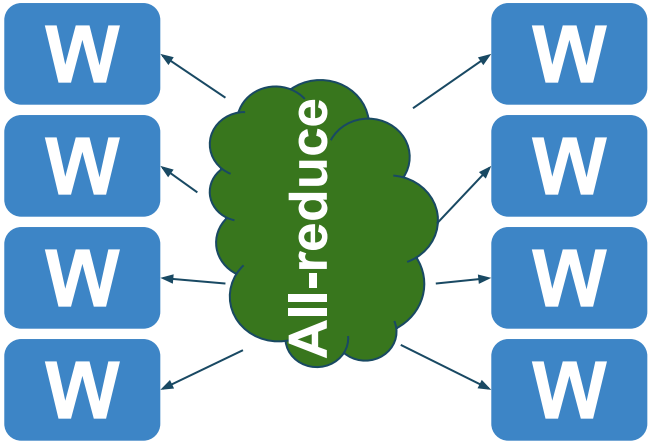
- applies to Stochastic Gradient Descent-type algorithms
  - each node takes a data batch and computes model updates independently
  - these updates are then collectively summed and applied to the local model



From Pradeep Dubey, "Scaling to Meet the Growing Needs of Artificial Intelligence (AI), IDF 2016
https://software.intel.com/en-us/articles/scaling-to-meet-the-growing-needs-of-ai

Synchronous Update

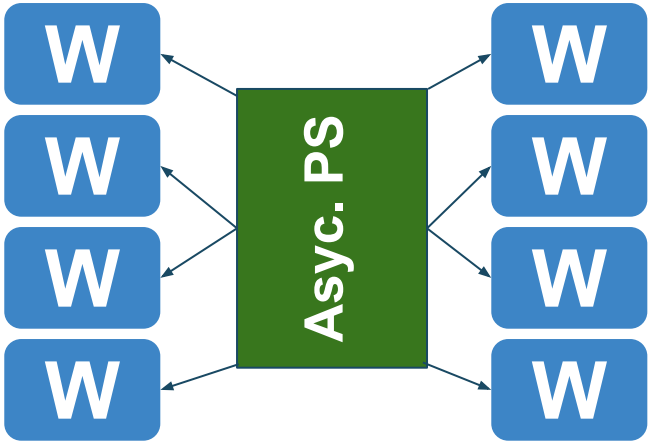
- all nodes compute gradients locally
- gradients are summed across nodes
- updates propagated to all nodes
- pros:
  - stable convergence
- cons:
  - scaling is not optimal because all nodes have to wait for reduction to complete (stragglers slow everyone down)
  - global (effective) batch size grows with number of nodes



Synchronous SGD, decentralized

Asynchronous Update

- all nodes compute gradient update locally
- gradient is sent to parameters server
- parameters servers incorporates gradients into model as they arrive and sends back the updated model to the corresponding node
- pros
  - no node waits for anybody (perfect scaling)
  - resilient
- cons
  - use of stale gradients can have impact on convergence rate (depending on #workers)
  - parameter server can be bottleneck



Asynchronous SGD, parameter-server