Optimization Techniques

AI602: Recent Advances in Deep Learning

Lecture 1

Slide made by

Insu Han, Seunghyun Lee and Younggyo Seo

KAIST EE & AI

Table of Contents

Table of Contents

1. **Introduction**
   - Empirical risk minimization (ERM)

2. **Stochastic Gradient Descent**
   - Gradient descent (GD) and stochastic gradient descent (SGD)
   - Momentum and adaptive learning rate methods
   - Learning rate scheduling
   - Large batch training

3. **Normalization Techniques**
   - Batch Normalization
   - Normalization-Free Network (NFNet)

4. **Summary**

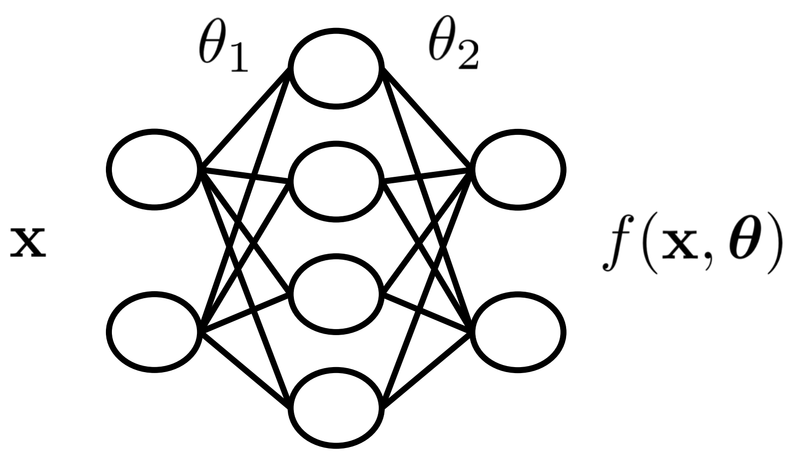
Empirical Risk Minimization (ERM)

- Given training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
- Prediction function $f(\mathbf{x}_i, \boldsymbol{\theta}) \approx y_i$ parameterized by $\boldsymbol{\theta}$
- **Empirical risk minimization:** Find a parameter that minimizes the loss function

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i, \boldsymbol{\theta}), y_i) := L(\boldsymbol{\theta})$$

where $\ell(\cdot, \cdot)$ is a loss function e.g., MSE, cross entropy,

- For example, neural network has $f(\mathbf{x}, \boldsymbol{\theta}) = \theta_k^\top \sigma\left(\theta_{k-1}^\top \sigma(\cdots \sigma(\theta_1^\top \mathbf{x}))\right)$



$$L(\boldsymbol{\theta}) = \frac{1}{n} \sum_i (f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i)^2$$

Next, how to solve ERM?

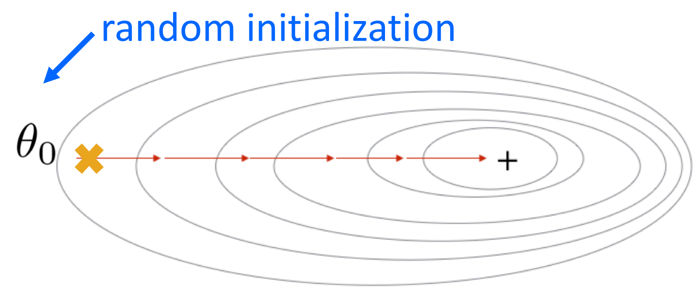
Table of Contents

- **Gradient descent (GD)** updates parameters iteratively by taking gradient.

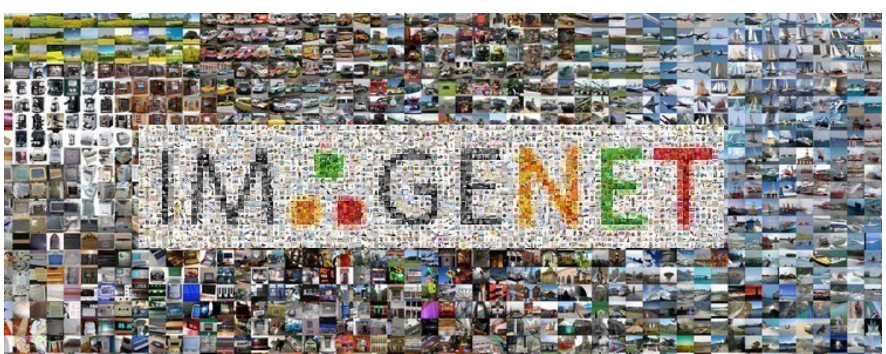
parameters loss function

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \underbrace{\nabla L\left(\boldsymbol{\theta}_t\right)}_{:= \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\boldsymbol{\theta}_t; \mathbf{x}_i, y_i)}$$

learning rate



- $(+)$ Converges to global (local) minimum for convex (non-convex) problem.
- $(-)$ Not efficient with respect to computation time and memory space for huge n.
- For example, ImageNet dataset has $\boldsymbol{n}$ =**1,281,167** images for training.



1.2M of 256x256 RGB images
≈ 236 GB memory

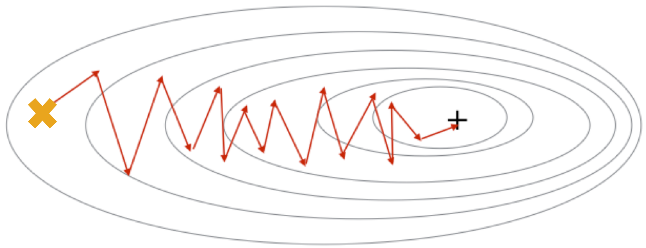
Next, efficient GD

Stochastic Gradient Descent (SGD)

- **Stochastic gradient descent (SGD)** use samples to approximate GD

$$\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\boldsymbol{\theta}; \mathbf{x}_i, y_i)$$

$$\simeq \frac{1}{|\mathcal{B}|} \sum_{\text{sample } i \in \mathcal{B}} \nabla \ell\left(\boldsymbol{\theta}; \mathbf{x}_i, y_i\right)$$



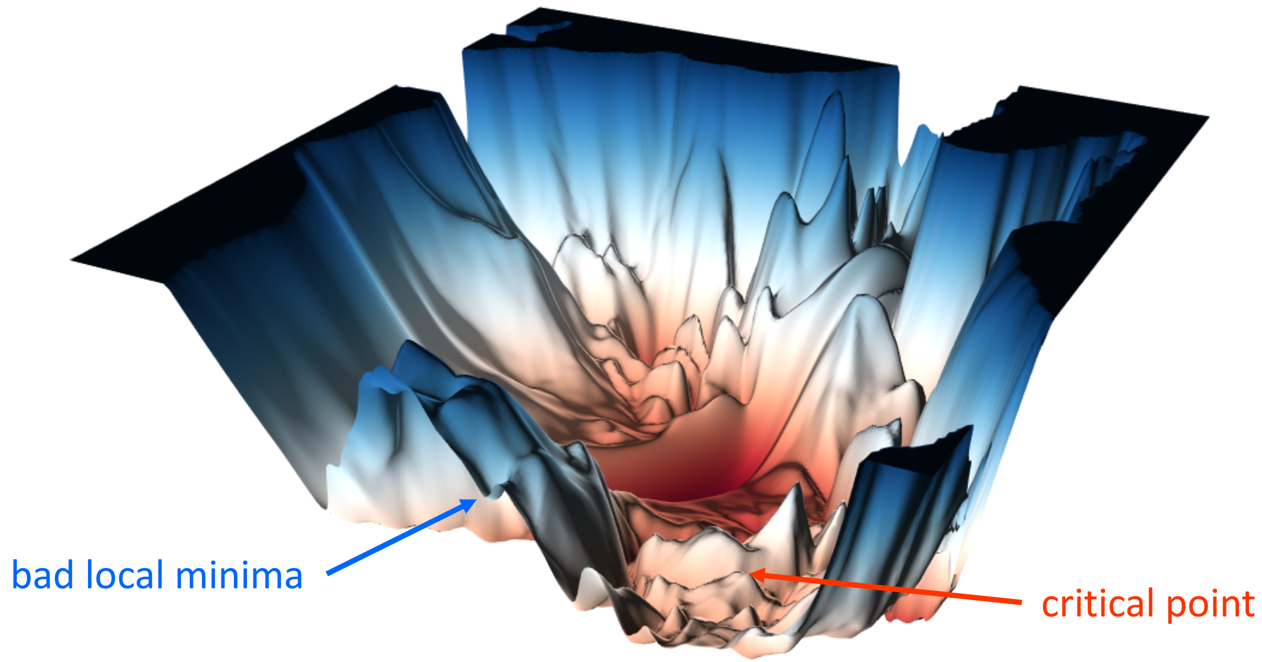
- In practice, minibatch size $|\mathcal{B}|$ typically ranges from 32 to 512 (single machine)
- SGD can find the global solution when
  1. loss function is convex
  2. bounded variance
  3. decreasing learning rate

- But, in many practical problems, SGD has some challenges

* source : https://lovesnowbest.site/2018/02/16/Improving-Deep-Neural-Networks-Assignment-2/

Hard to optimize practical problems

- Main practical challenges and current solutions:
  1. Loss function is nonconvex and includes local minima/critical points
  2. SGD can be too noisy and might be unstable → momentum
  3. Hard to find a good learning rate → adaptive learning rate



loss surface of a neural net (ResNet-50)

Next, momentum

* source : http://www.telesens.co/loss-landscape-viz/viewer.html

Table of Contents

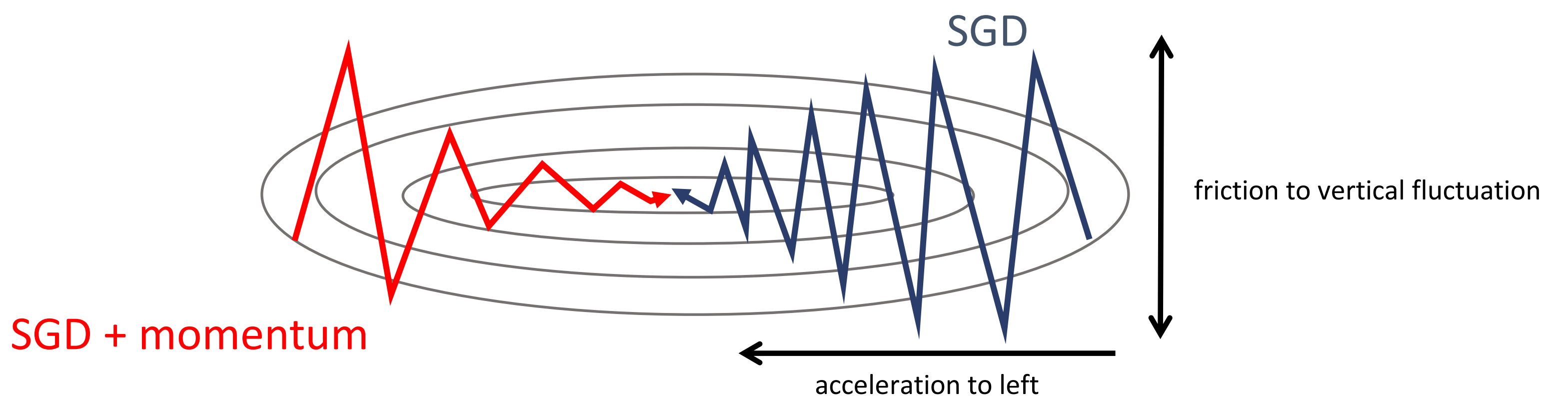
1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \underbrace{\mathbf{m}_t}_{\text{momentum}} \qquad \mathbf{m}_t = \underbrace{\mu}_{\text{preservation ratio } \mu \in [0,1]} \mathbf{m}_{t-1} + \gamma \nabla L(\boldsymbol{\theta}_t)$$

   - Equivalent to **moving average** with the fraction μ of previous update.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \left(\nabla L(\boldsymbol{\theta}_t) + \mu \nabla L(\boldsymbol{\theta}_{t-1}) + \mu^2 \nabla L(\boldsymbol{\theta}_{t-2}) + \cdots \right)$$

   - (+) Momentum reduces the oscillation and accelerates the convergence.



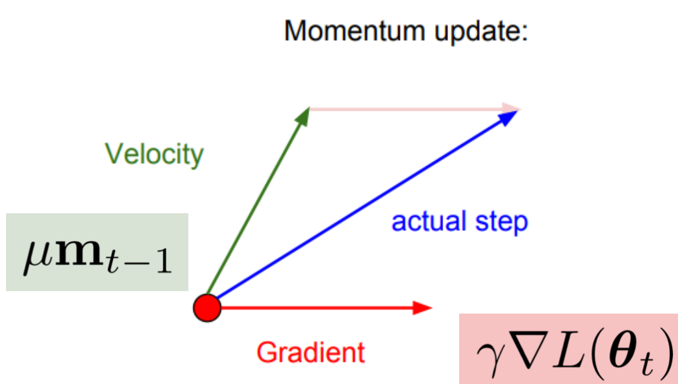
Momentum Methods: Nesterov's Momentum

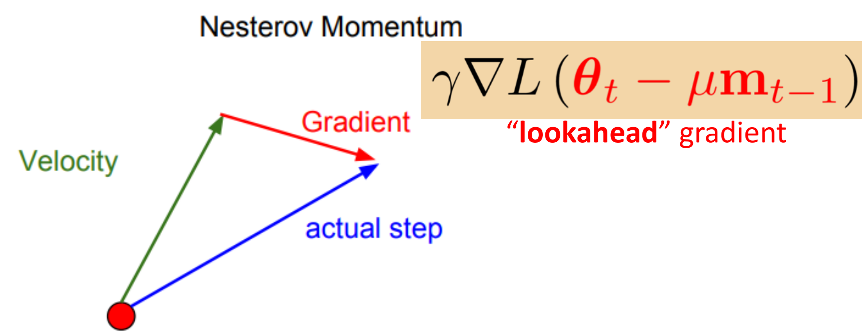
1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \underbrace{\mathbf{m}_t}_{\text{momentum}} \qquad \mathbf{m}_t = \underbrace{\mu}_{\text{preservation ratio } \mu \in [0,1]} \mathbf{m}_{t-1} + \gamma \nabla L(\boldsymbol{\theta}_t)$$

   - (−) Momentum can fail to converge even for simple convex optimizations.
   - **Nesterov's accelerated gradient** (NAG) [Nesterov'83] uses gradient for approximate future position, i.e.,

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \gamma \nabla L(\boldsymbol{\theta}_t - \mu \mathbf{m}_{t-1})$$





* source : Nesterov. "A method for solving the convex programming problem with convergence rate O(1/k^2)." 1983

1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \underset{\text{momentum}}{\underset{\downarrow}{\mathbf{m}_t}} \qquad\qquad \mathbf{m}_t = \underset{\text{preservation ratio } \mu \in [0,1]}{\underset{\downarrow}{\mu}}\mathbf{m}_{t-1} + \gamma \nabla L(\boldsymbol{\theta}_t)$$

   - **Nesterov's accelerated gradient** (NAG) [Nesterov'83] uses gradient for approximate future position, i.e.,

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \gamma \nabla L\left(\boldsymbol{\theta}_t - \mu \mathbf{m}_{t-1}\right)$$

```
while True:
    dtheta = compute_gradient(theta, batch_size)
    theta = theta - learning_rate * dtheta
```

SGD

```
m = 0
while True:
    dtheta = compute_gradient(theta, batch_size)
    m = mu * m + learning_rate * dtheta
    theta = theta - m
```

SGD + momentum

```
m = 0
while True:
    dtheta = compute_gradient(theta, batch_size)
    m_old = m
    m = mu * m + learning_rate * dtheta
    x = 
```

NAG

Quiz: fill in the pseudo code of Nesterov' accelerated gradient

Adaptive Learning Rate Methods: AdaGrad, RMSProp

2. Adaptively changing learning rate (AdaGrad, RMSProp)
   - **AdaGrad** [Duchi'11] downscales a learning rate by magnitude of previous gradients.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\gamma}{\sqrt{\boldsymbol{v}_t}} \nabla L\left(\boldsymbol{\theta}_t\right) \qquad \underbrace{\boldsymbol{v}_{t+1}}_{\text{sum of all previous squared gradients}} = \boldsymbol{v}_t + \nabla L\left(\boldsymbol{\theta}_t\right)^2$$

   - (−) the learning rate strictly decreases and becomes too small for large iterations.

   - **RMSProp** [Tieleman'12] uses the moving averages of squared gradient.

$$\boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t + (1-\mu) \nabla L\left(\boldsymbol{\theta}_t\right)^2$$

preservation ratio $\mu \in [0,1]$

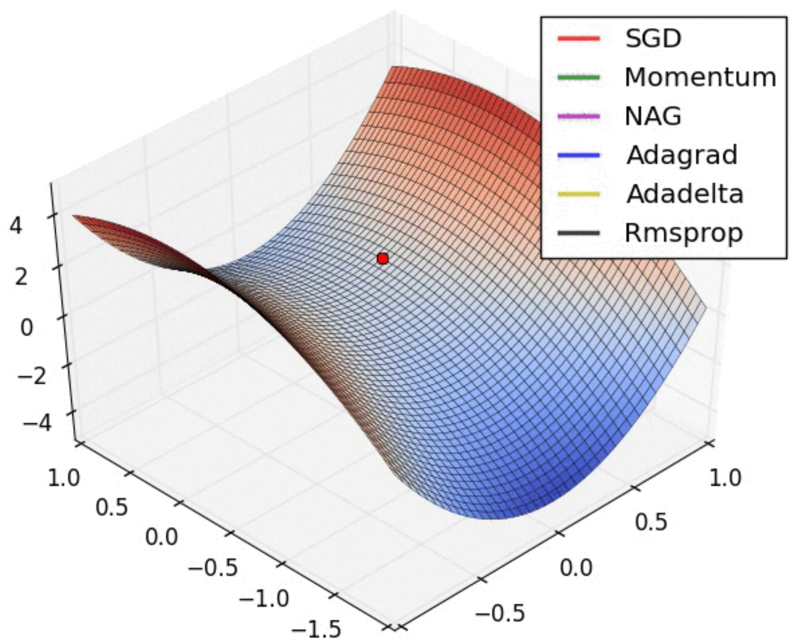
   - Other variants also exist, e.g., Adadelta [Zeiler'12]

* source : Duchi et al., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.", JMLR 2011
* source : Tieleman, Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.", 2012
* source : Zeiler, "Adadelta: An Adaptive Learning Rate Method.", 2012

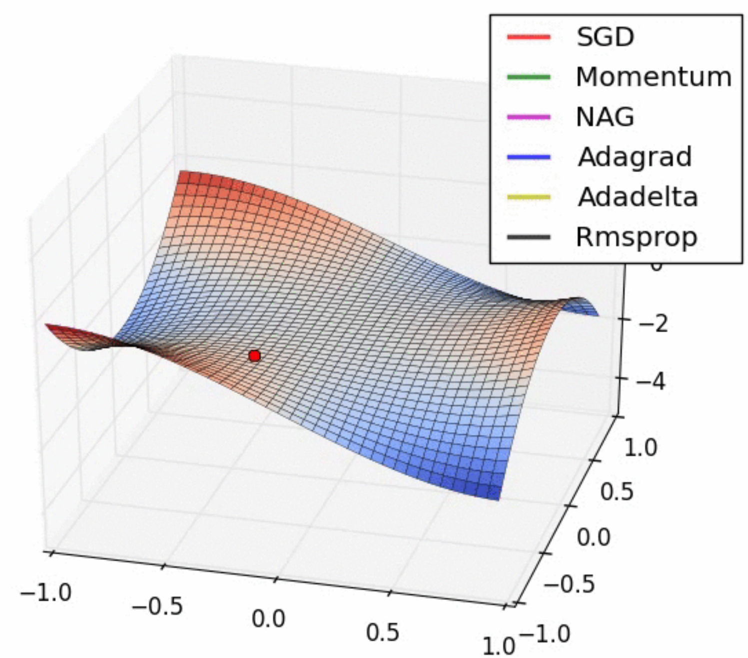
Adaptive Learning Rate Methods

- Visualization of algorithms

optimization on saddle point



optimization on local optimum



- Adaptive learning-rate methods, i.e., Adadelta and RMSprop are most suitable and provide the best convergence for these scenarios

Next, momentum + adaptive learning rate

* source: animations from from Alec Radford' blog

Adaptive Learning Rate Methods: Adam

1 + 2. Combination of momentum and adaptive learning rate

- **Adam** (ADAptive Moment estimation) [Kingma'15]

momentum

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\gamma}{\sqrt{\boldsymbol{v}_t}}\boldsymbol{m}_t \qquad \begin{aligned} \boldsymbol{m}_{t+1} &\leftarrow \mu_1 \boldsymbol{m}_t + (1-\mu_1)\nabla L\left(\boldsymbol{\theta}_t\right) \\ \boldsymbol{v}_{t+1} &\leftarrow \mu_2 \boldsymbol{v}_t + (1-\mu_2)\nabla L\left(\boldsymbol{\theta}_t\right)^2 \end{aligned}$$

average of squared gradients

- Can be seen as momentum + RMSprop update.
- Other variants exist, e.g., Adamax [Kingma'14], Nadam [Dozat'16]

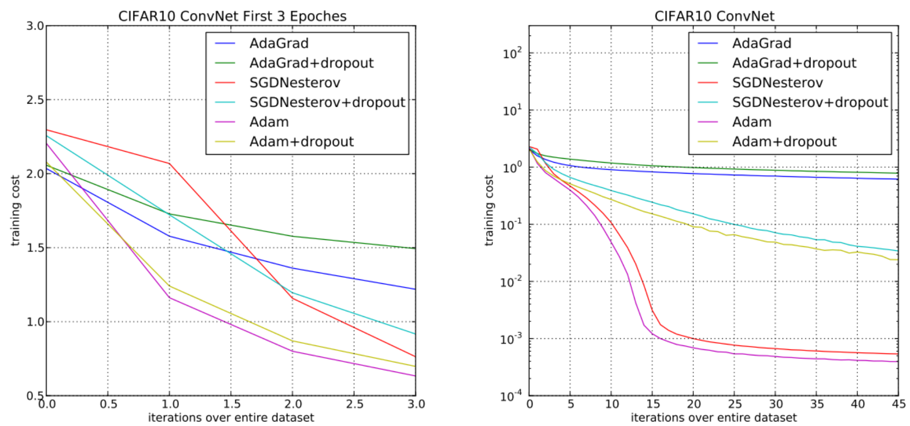


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

* source : Kingma and Ba. Adam: "A method for stochastic optimization." ICLR 2015

SGD/Adam with decoupled weight decay

- Many learning problems optimize the loss with $\boldsymbol{L^2}$ **norm penalty**

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2$$

  - It is sometimes called "weight decay" since its gradient decays weight (for SGD):

$$\underbrace{\boldsymbol{\theta} - \eta \nabla \left(L(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2 \right)}_{\text{SGD on L2-norm penalty}} \quad \underset{\nabla \|\boldsymbol{\theta}\|_2^2 = 2\boldsymbol{\theta}}{\Leftrightarrow} \quad \underbrace{(1 - 2\eta\lambda)\boldsymbol{\theta} - \eta \nabla L(\boldsymbol{\theta})}_{\text{weight decay}}$$

- This equivalence **does not hold** for momentum/adaptive methods! (check)
- [Loshchilov'19] proposes **decoupling** weight decay from optimization steps
- For example, decoupled SGD with momentum iterates (also applicable to Adam)

$$\boldsymbol{m}_{t+1} \leftarrow \mu_1 \boldsymbol{m}_t + (1 - \mu_1) \underbrace{\left(\nabla L\left(\boldsymbol{\theta}_t\right) + \lambda \boldsymbol{\theta}_t \right)}_{\text{gradient of loss with L2 penalty}}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{m}_t - \underbrace{2\eta\lambda\boldsymbol{\theta}_t}_{\text{weight decay}}$$

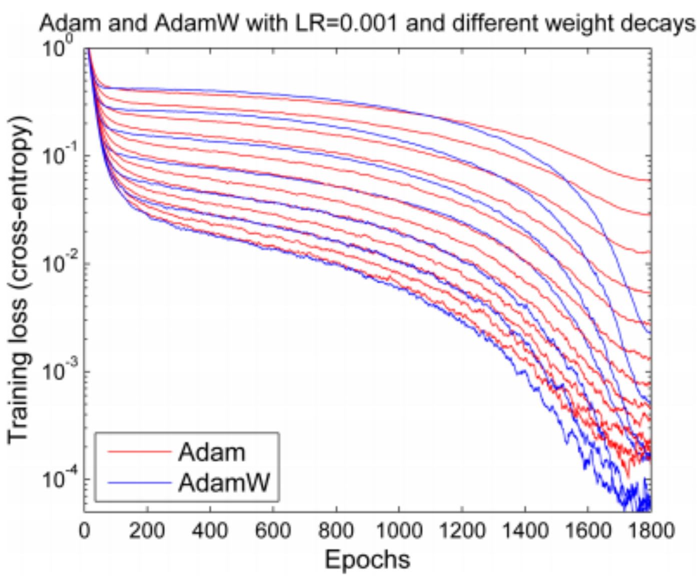
* source : Loshchilov et al., "Decoupled Weight Decay Regularization.", ICLR 2019

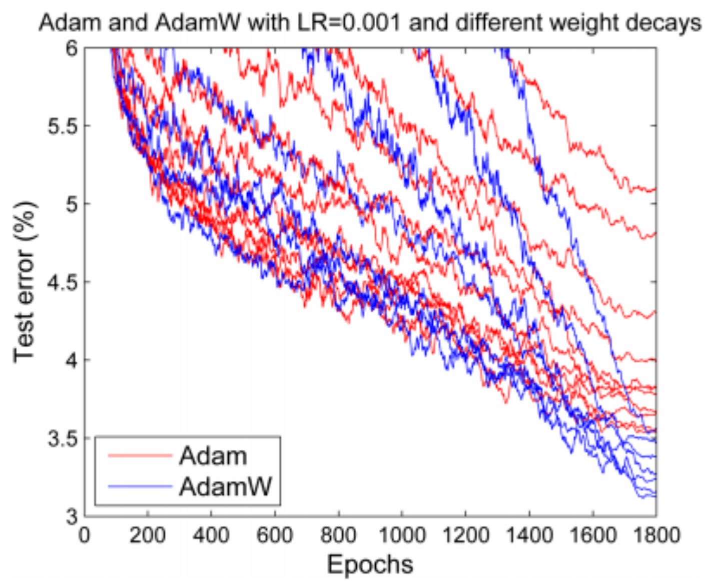
SGD/Adam with decoupled weight decay

- Many learning problems optimize the loss with L^2 **norm penalty**

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2$$

  - It is sometimes called "weight decay" since its gradient decays weight (for SGD)
- This equivalence **does not hold** for momentum/adaptive methods! (check)
- [Loshchilov'19] proposes **decoupling** weight decay from optimization steps
- The proposed **AdamW** outperforms Adam

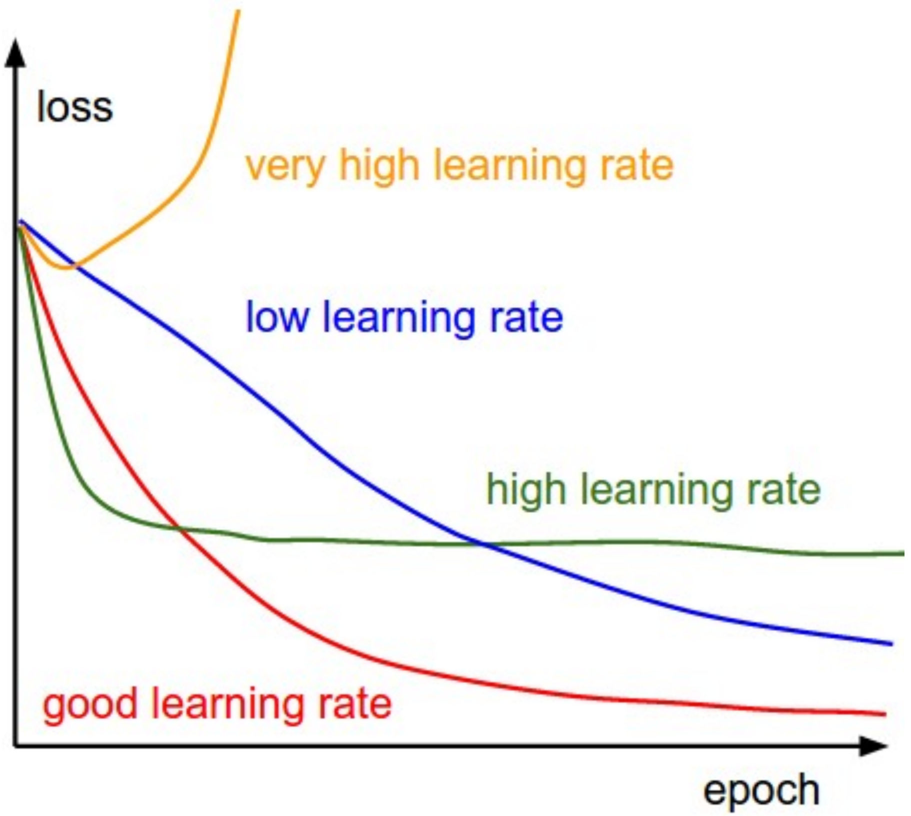




* source : Loshchilov et al., "Decoupled Weight Decay Regularization.", ICLR 2019

Table of Contents

3. Learning rate scheduling
   - **Learning rate is critical for minimizing loss !**

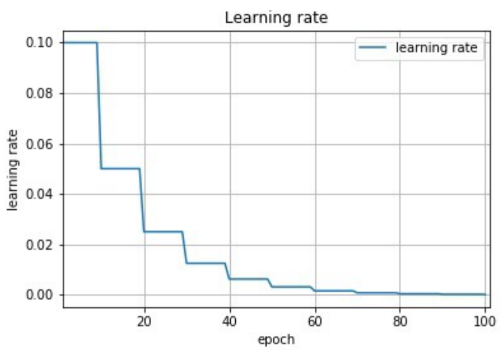


Too high → May ignore the narrow valley, can diverge
Too low → May fall into the local minima, slow converge

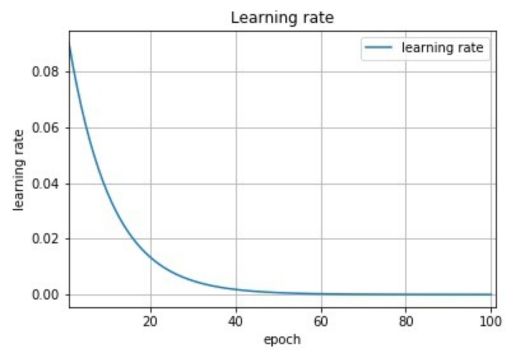
* source : http://cs231n.github.io/neural-networks-3/

Learning Rate Scheduling: Decay

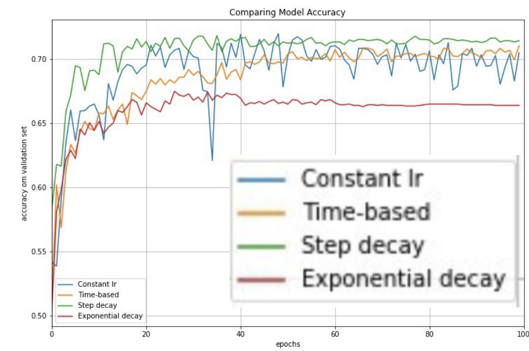
3. Learning rate scheduling : Decay methods
   - Constant learning rate often prevents convergence → needs decay!



step decay

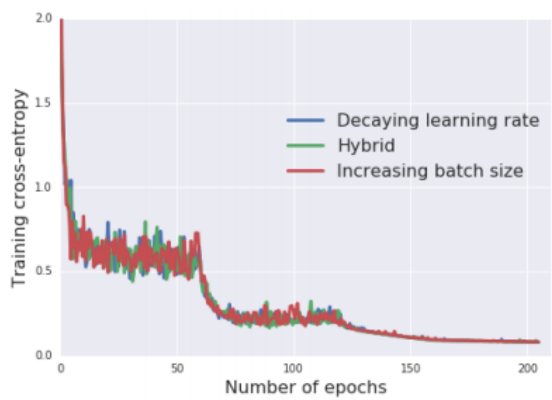


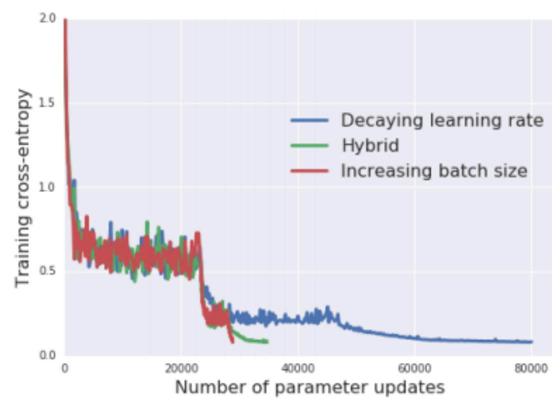
exponential decay



accuracy

- [Smith'17] showed **"Decaying the learning rate = Increasing the batch size"**

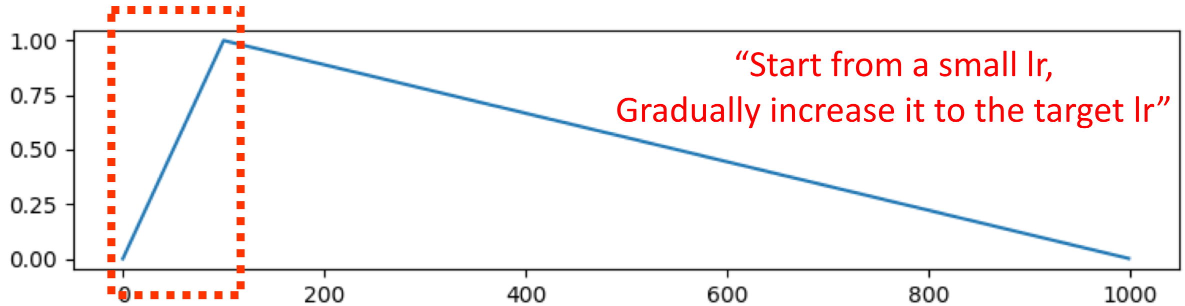




- source : https://towardsdatascience.com/

* source : Smith et al., "Don't Decay the Learning Rate, Increase the Batch Size.", ICLR 2017

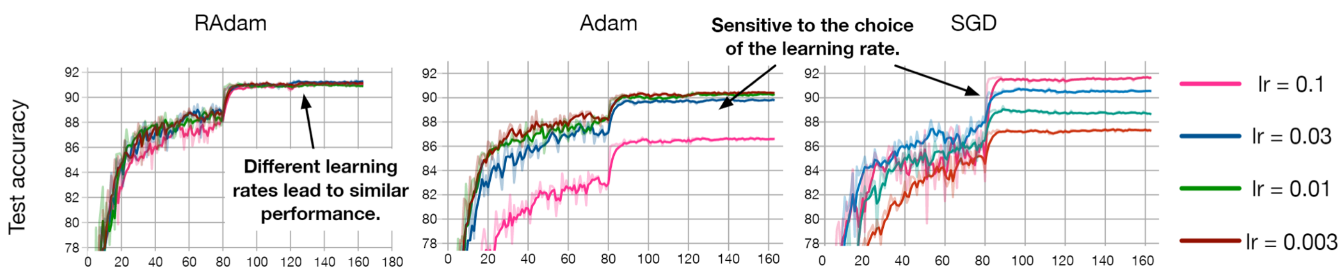
3. Learning rate scheduling : Warm-up
    - Adaptive optimizers like Adam suffer from large variance in the early training phase
    - Large batch training with momentum SGD also has similar issues
    - **Warm-up heuristic** is used to stabilize training



- **RAdam** [Liu'20] rectifies the variance of Adam lr, with theoretical justifications
    - RAdam enjoys the benefits of warm-up, but no need to search for scheduling

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\gamma r_t}{\sqrt{\boldsymbol{v}_t}} \boldsymbol{m}_t$$

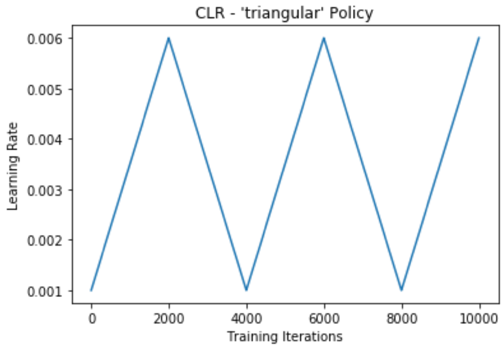
Variance rectification term
(Check the paper for details!)

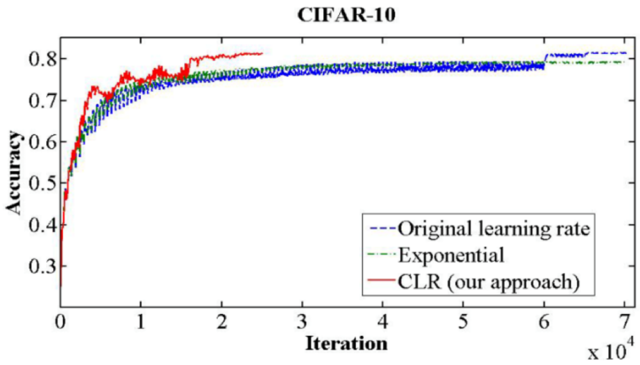


* source : https://huggingface.co/transformers/main_classes/optimizer_schedules.html
* source : Liu et al., "On The Variance of The Adaptive Learning Rate and Beyond.", ICLR 2020

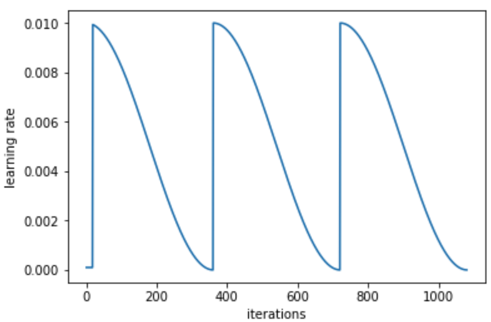
Learning Rate Scheduling: Cyclical

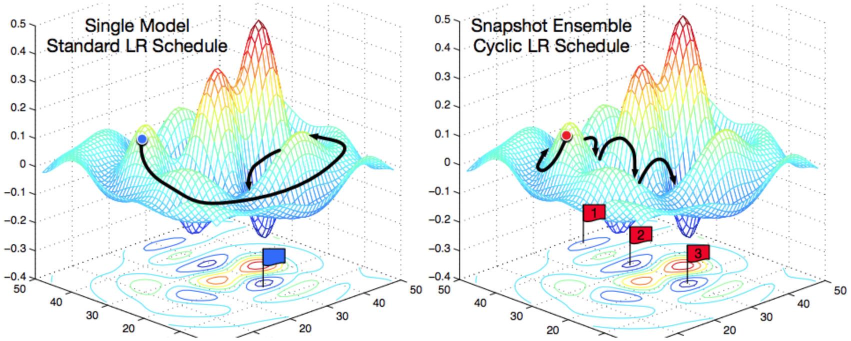
3. Learning rate scheduling : Cyclical methods
   - [Smith'15] proposed **cyclical** learning rate
   - **Increasing learning rate** can be useful for escaping saddle points / bad local minima





   - [Loshchilov'17] uses **cosine cycling** and **warm restart**
     - Traverses several local minima by moving up and down the loss surface
     - → suggests snapshot ensemble: ensemble over several local minima found





* source : Smith., "Cyclical Learning Rates for Training Neural Networks." 2015

* source : Loshchilov et al., "SGDR: Stochastic Gradient Descent with Warm Restarts." ICLR 2017

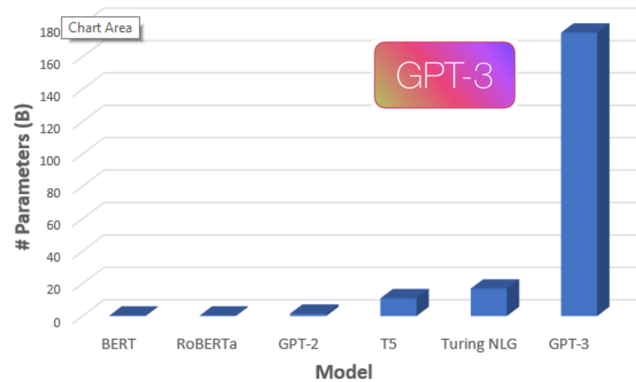
Table of Contents

Large Batch Training: Motivation

- Deep learning is scaling up very quickly

Instagram Dataset
w/ ~1bil. images [Mahajan'18]

Larger dataset



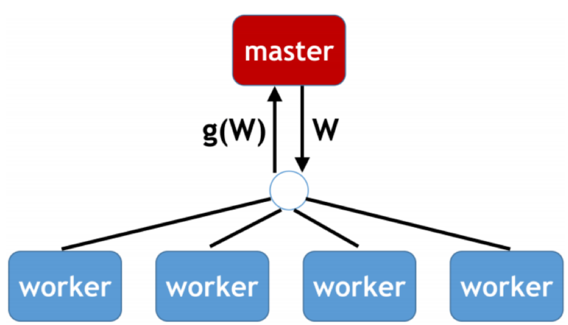
Larger model

Cloud TPU v3 Pod
100+ petaflops
32 TB HBM
2-D toroidal mesh network

More compute

- **Data parallelism** enables large-scale training
  - With *k* times more GPUs, global batch size increases by *k*
  - Ignoring communication cost, *k* times fewer iterations per epoch

1. Aggregate gradient estimates

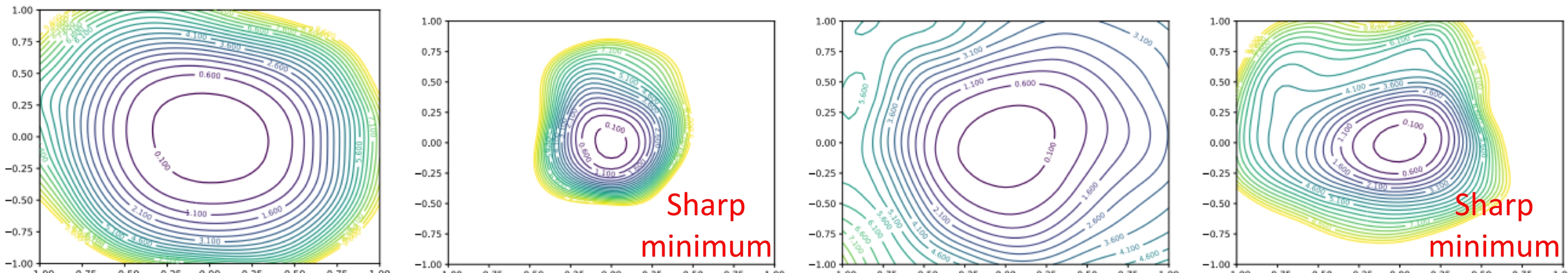


2. Synchronize updated weights across workers

* source : Mahajan, et al., "Exploring the limits of Weakly Supervised Pretraining", ECCV 2018
* source: Yu, et al., "ImageNet Training in Minutes", 2018

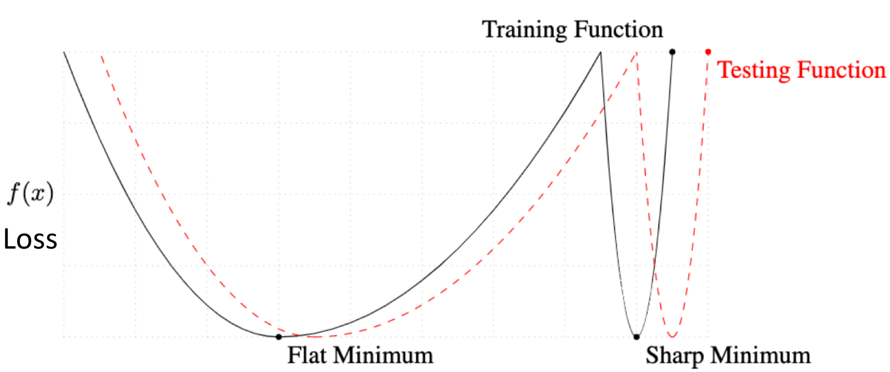
Large Batch Training: Challenge

- Naïvely increasing batch size → performance degradation
  - In particular, **generalization** performance suffers
- One popular explanation: **Sharp Minima Problem** [Keskar'17]
  - Large batch (LB) training finds a "sharp minimum"



(e) 0.0, 128, 7.37% (f) 0.0, $\boxed{8192,}$ 11.07% (g) 5e-4, 128, 6.00% (h) 5e-4, $\boxed{8192,}$ 10.19%

Loss visualization along two random directions in the parameter space (VGG-9, CIFAR-10) [Li'18]



High sensitivity of training loss around $\theta^*_{\mathtt{train}}$

→

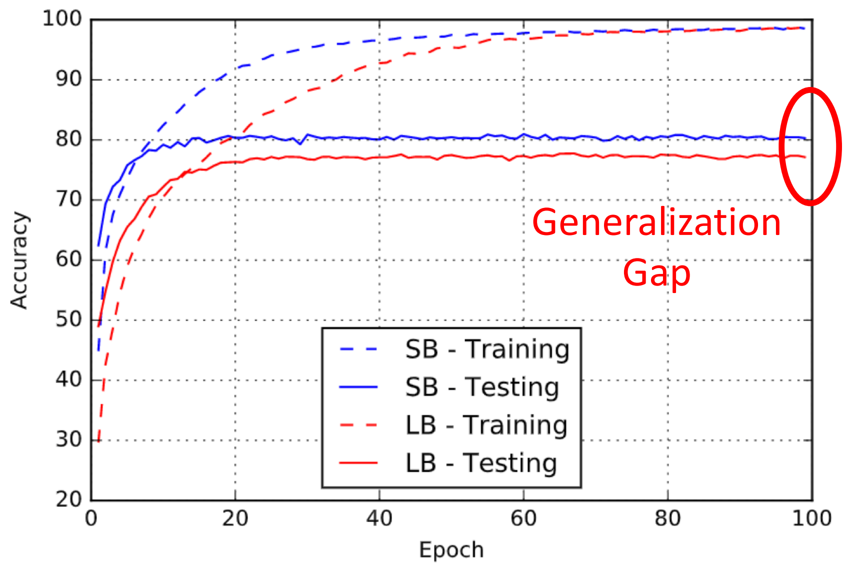
$\theta^*_{\mathtt{train}}$ is a **poor** minimizer for **test loss**

* source : Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017

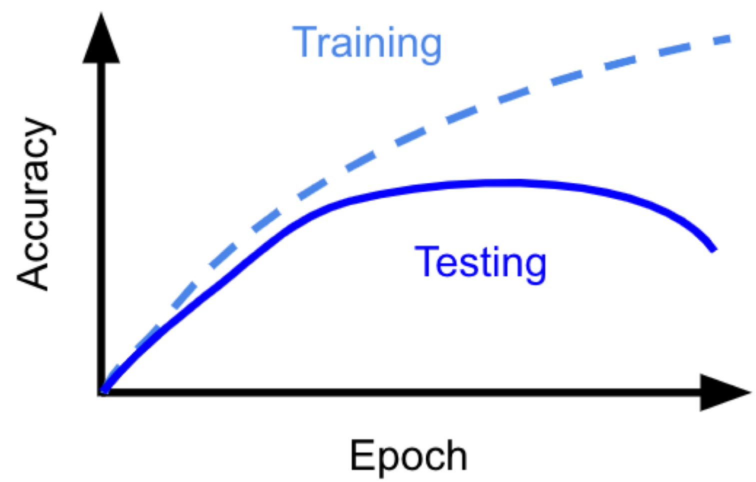
* source: Li et al., "Visualizing the Loss Landscape of Neural Nets", NeurIPS 2018

Large Batch Training: Challenge

- Naïvely increasing batch size → performance degradation
  - In particular, **generalization** performance suffers
- One popular explanation: **Sharp Minima Problem** [Keskar'17]
  - Caveat: this is **not** the same as **overfitting**!
    - In particular, cannot apply early stopping to solve the problem



LB Training



Overfitting

* source : Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017

Large Batch Training: Challenge

- Naïvely increasing batch size → performance degradation
  - In particular, **generalization** performance suffers
- Another explanation: **Optimization Difficulty** [Goyal'18]
  - [Goyal'18] suggests sharp minimum is **not** an **inherent problem** of LB training
    - With careful optimization, LB training is possible w/o loss in generalization

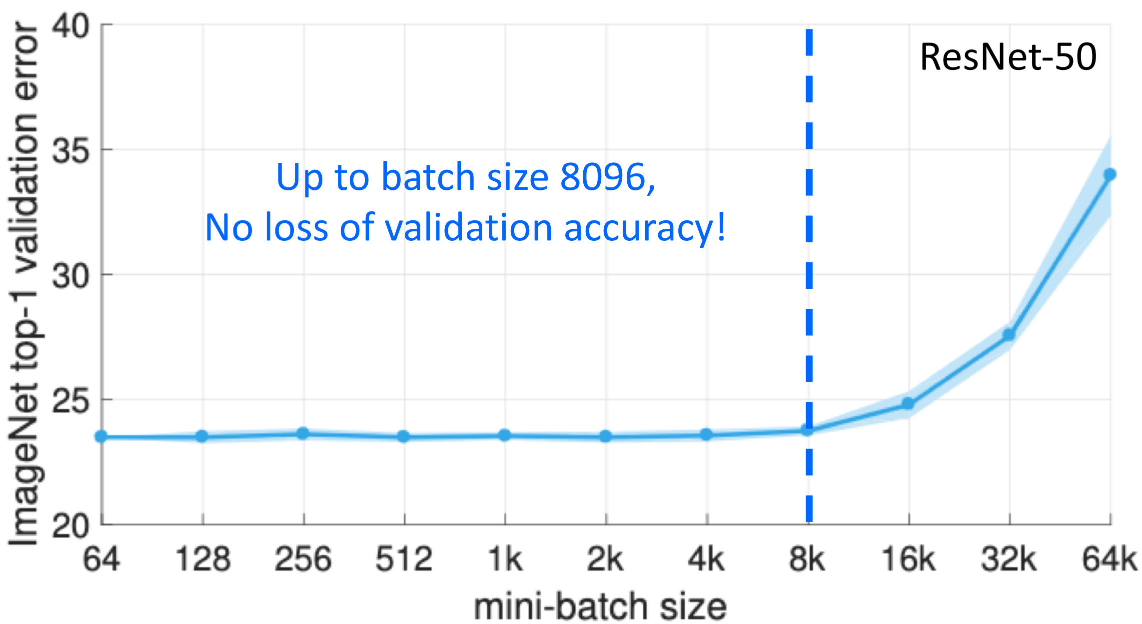


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.**

* source : Goyal et al., "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", 2018

Large Batch Training: Learning rate warm-up

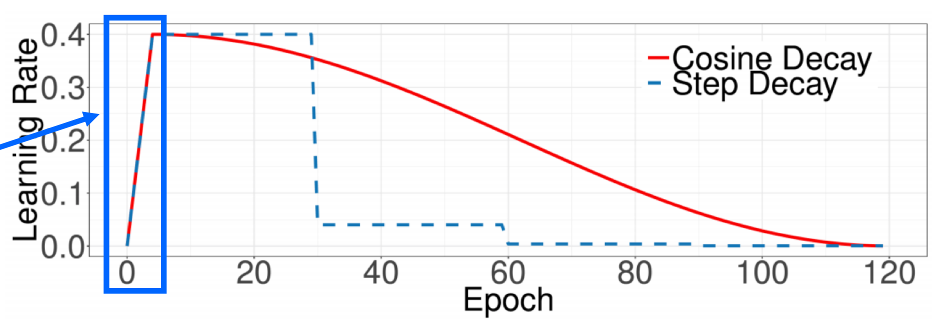
- **Learning rate warm-up** [Goyal'18]
  1. Linear scaling rule
     - Given a fixed number of epochs, **increasing batch size** B by *k* times means *k* times **fewer training iterations**, for:

$$|\mathcal{D}_{\texttt{data}}| = B \cdot (\texttt{\# iters per epoch}) = kB \cdot \frac{(\texttt{\# iters per epoch})}{k}$$

     - To make up for this, learning rate must **scale linearly** with batch size

  2. Warm-up
     - During initial training phase, neural network is changing rapidly
     - In this case, large learning rate can be destructive → **'warm up' the rate**!

Learning rate
warm-up



(a) Learning Rate Schedule

Scales up to
8096 batch size
(ImageNet, ResNet-50)

* source : Goyal et al., "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", 2018
* source: He et al., "Bag of Tricks for Image Classification with Convolutional Neural Networks", CVPR 2019

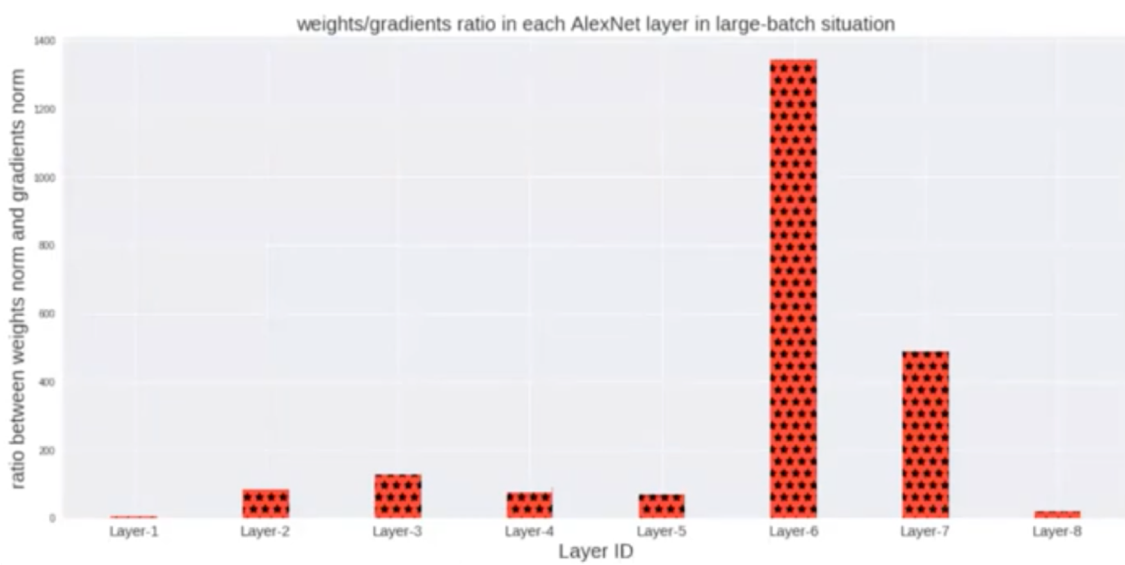
- **Layer-wise Adaptive Rate Scaling (LARS)** [You'17]
  - Ratio between weight and its gradient matters

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \nabla L(\boldsymbol{\theta}_t) \qquad \frac{||\boldsymbol{\theta}_t||}{\gamma ||\nabla L(\boldsymbol{\theta}_t)||}$$

Too large: slow learning

Too small: divergence

  - Note that standard SGD uses a fixed γ for all weights
  - Observation: for LB training, weight-gradient ratio appears **differently across layers**!



* source : You et al., "Large Batch Training of Convolutional Neural Networks", 2017

* source: https://www.youtube.com/watch?v=kwEBP-Wbtdc&ab_channel=NUSDepartmentofComputerScience

- **Layer-wise Adaptive Rate Scaling (LARS)** [You'17]
  - Solution: different learning rates for each layer

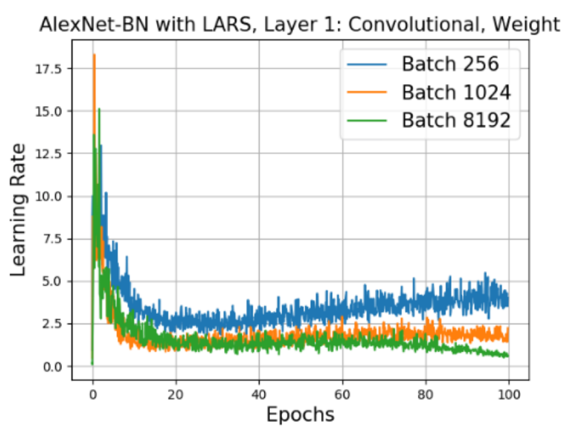
$$\boldsymbol{\theta}^l_{t+1} = \boldsymbol{\theta}^l_t - \gamma \cdot \lambda^l \cdot \nabla L(\boldsymbol{\theta}^l_t)$$

γ global learning rate

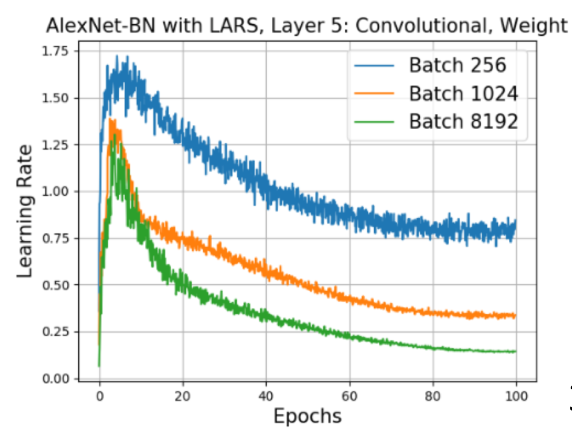
$$\lambda^l := \eta \frac{\|\boldsymbol{\theta}^l\|}{\|\nabla L(\boldsymbol{\theta}^l)\|}$$

local learning rate, where $\eta < 1$ is the trust coefficient

  - By layer-wise scaling, vanishing/exploding gradient problem can be prevented
  - Author claims **noisy** learning signal due to dynamic lr helps avoiding **sharp minima**



(a) Local LR, conv1-weights



(c) Local LR , conv5-weights

Scales up to 32768 batch size (ImageNet, ResNet-50)

* source : You et al., "Large Batch Training of Convolutional Neural Networks", 2017
* source: https://www.youtube.com/watch?v=kwEBP-Wbtdc&ab_channel=NUSDepartmentofComputerScience

Large Batch Training: LARS & LAMB

- **Layer-wise Adaptive Moments for Batch training (LAMB) [You'20]**
  - Warm-up [Goyal'18], LARS [You'17] both build on top of momentum-SGD
  - **LAMB** is an extension of LARS to the 'weight-adaptive' optimizer **Adam**
    - Successfully scales **BERT** training (batch size ~32768)
    - Trains ResNet-50 with Adam to **match** the performance of **momentum SGD**

Table 1: We use the F1 score on SQuAD-v1 as the accuracy metric. The baseline F1 score is the score obtained by the pre-trained model (BERT-Large) provided on BERT's public repository (as of February 1st, 2019). We use TPUv3s in our experiments. We use the same setting as the baseline: the first 9/10 of the total epochs used a sequence length of 128 and the last 1/10 of the total epochs used a sequence length of 512. All the experiments run the same number of epochs. Dev set means the test data. It is worth noting that we can achieve better results by manually tuning the hyperparameters. The data in this table is collected from the untuned version.

Solver	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

No loss in test performance

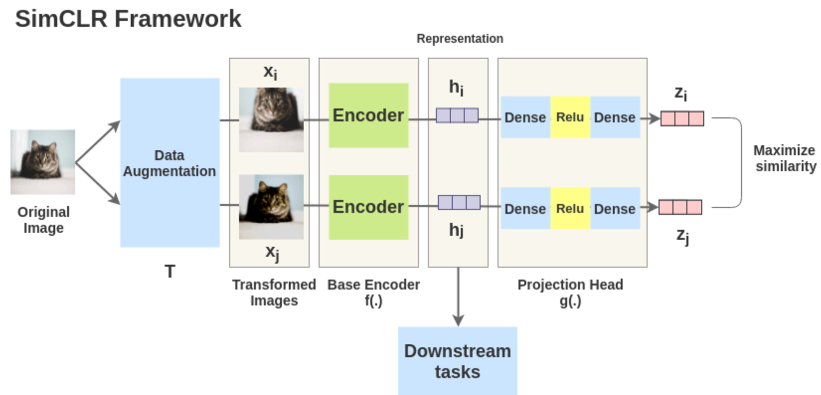
* source : You et al., "Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes", ICLR 2020

Large Batch Training: LARS & LAMB

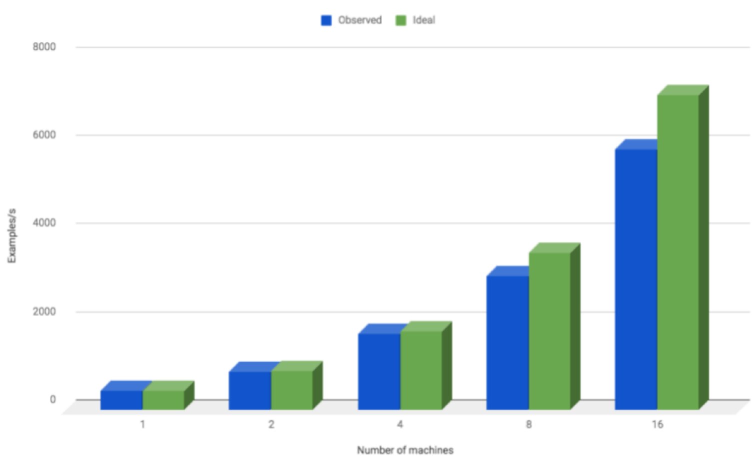
- Currently, LARS & LAMB are widely adopted in the deep learning community

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.)

ImageNet/ResNet-50 Training Speed Records



SimCLR uses LARS for training



LAMB enables scaling Transformer-XL to 128 GPUs

- Training Optimizers
  - Fused Adam optimizer and arbitrary `torch.optim.Optimizer`
  - Memory bandwidth optimized FP16 Optimizer
  - Large Batch Training with LAMB Optimizer
  - Memory efficient Training with ZeRO Optimizer
  - CPU-Adam

DeepSpeed (a large-scale DL optimization library) provides a LAMB implementation

* source : https://www.youtube.com/watch?v=kwEBP-Wbtdc

* source

But actually…

Large Batch Training: Sanity Check

- A recent paper [Nado'21] questions the effectiveness of LARS & LAMB
  - Good performance more due to **subtle implementation details**
    - For **ResNet-50**,
      - Unconventional BatchNorm hyperparameters;
      - No L2-regularization on bias parameters nor on BN parameters; etc.
      - Nesterov works just as well with similar modifications
    - For **BERT**,
      - Fixing BERT open source code's bug in Adam and learning schedule leads to good performance

Optimizer	Train Acc	Test Acc
Nesterov	78.97%	75.93%
LARS	78.07%	75.97%

Table 3. Median train and test accuracies over 50 training runs for Nesterov momentum Configuration B and LARS. (Batch size 32k)

Batch size	Step budget	LAMB	Adam
32k	15,625	91.48	**91.58**
65k/32k	8,599	90.58	**91.04**
65k	7,818	–	**90.46**

Table 4. Using Adam for pretraining exceeds the reported performance of LAMB in You et al. (2019) in terms of F1 score on the downstream SQuAD v1.1 task.

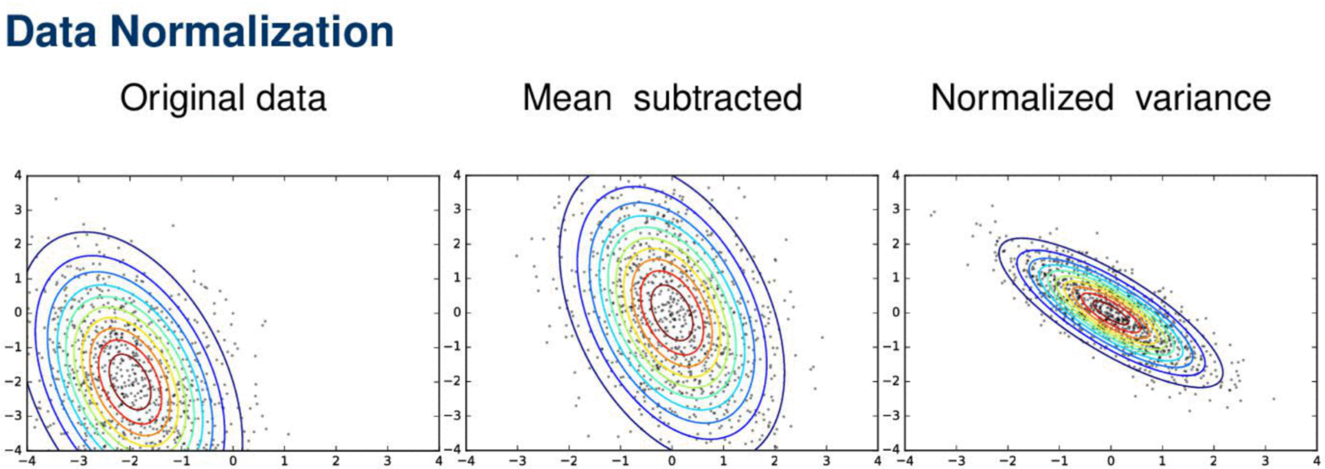
- Whether layer-wise adaptive learning rate really is useful is an open question

 * source : Nado et al., "A Large Batch Optimizer Reality Check: Traditional, Generic Optimizers Suffice Across Batch Sizes", 2021

Table of Contents

Normalization

- **Normalization** is widely-used technique to stabilize training process
  - Stabilizes training by adjusting the scale of inputs within unit variance



$$x \qquad x - \mu_x \qquad \frac{x - \mu_x}{\sigma_x}$$

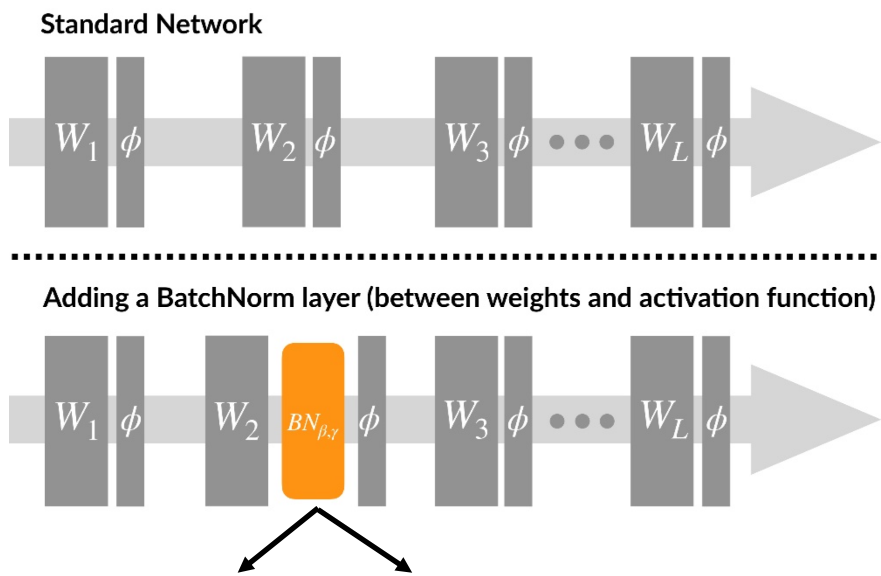
  - Commonly used in training recent deep learning models

```
transforms.Normalize(mean=[0.485, 0.456, 0.406],
                     std=[0.229, 0.224, 0.225])
```


*source : https://towardsdatascience.com/regularization-part-4-2ee8e7aa60ec

Normalization: Batch Normalization [Ioffe & Szezegy'15]

- **Batch Normalization:** Normalize the outputs *within the network*



Computed using samples within batch

$$y_{ichw} = \boxed{\gamma} \cdot \left(\frac{x_{ichw} - \boxed{\mu_c}}{\boxed{\sigma_c}} \right) + \boxed{\beta}$$

Learnable Parameters

$$x_{ichw} \in X^{N \times C \times H \times W}$$

$$\mu_c = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{ichw}$$

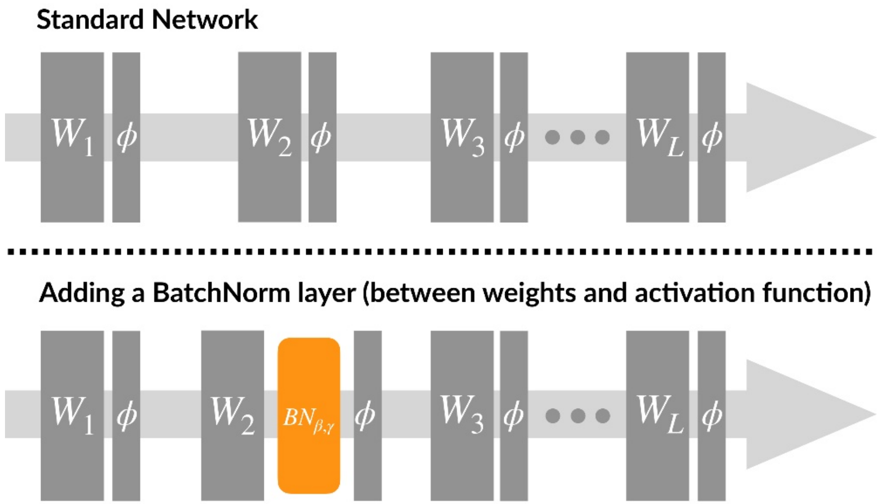
$$\sigma_c^2 = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{ichw} - \mu_i)^2$$

* source : Ioffe and Szegedy., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv 2019

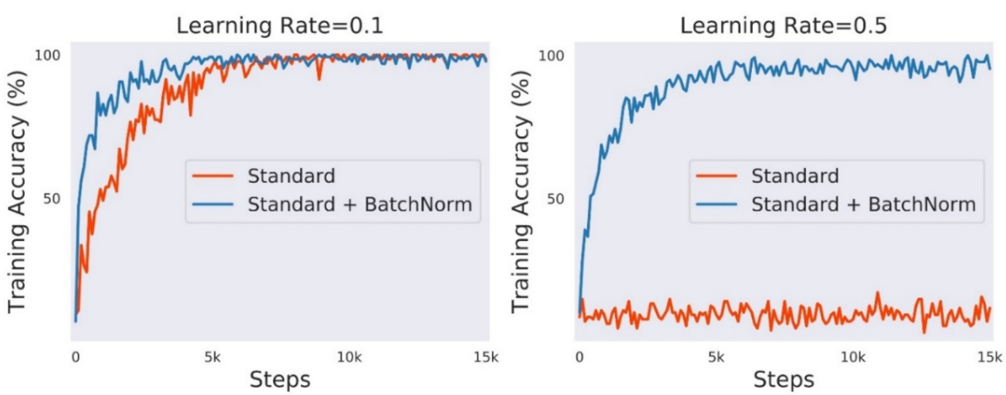
* source : https://gradientscience.org/batchnorm/

Normalization: Batch Normalization [Ioffe & Szezegy'15]

- **Batch Normalization:** Normalize the outputs *within the network*



- Batch normalization stabilizes training and **widely used in recent works**



VGG Network on CIFAR10

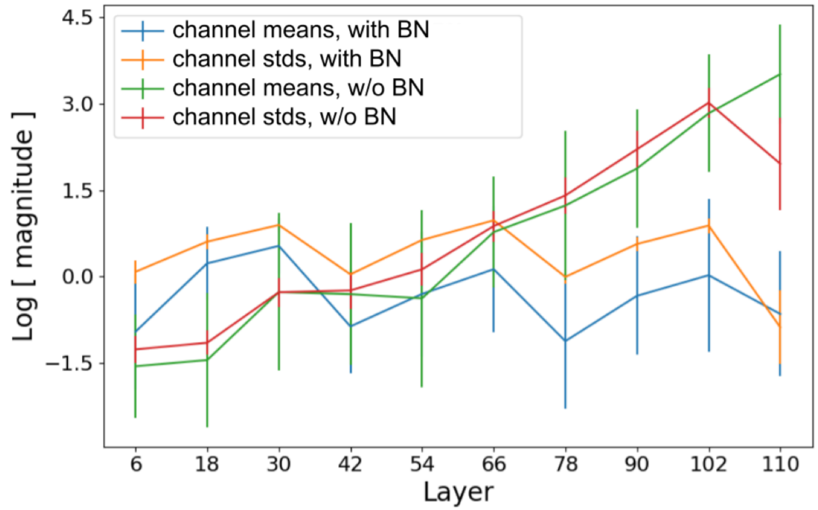
* source : Ioffe and Szegedy., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv 2019

* source : https://gradientscience.org/batchnorm/

Why does Batch Normalization (BN) work?

1. BN eliminates mean-shift

- Average channel means and variances at initialization
- Mean and variance grow exponentially in unnormalized network



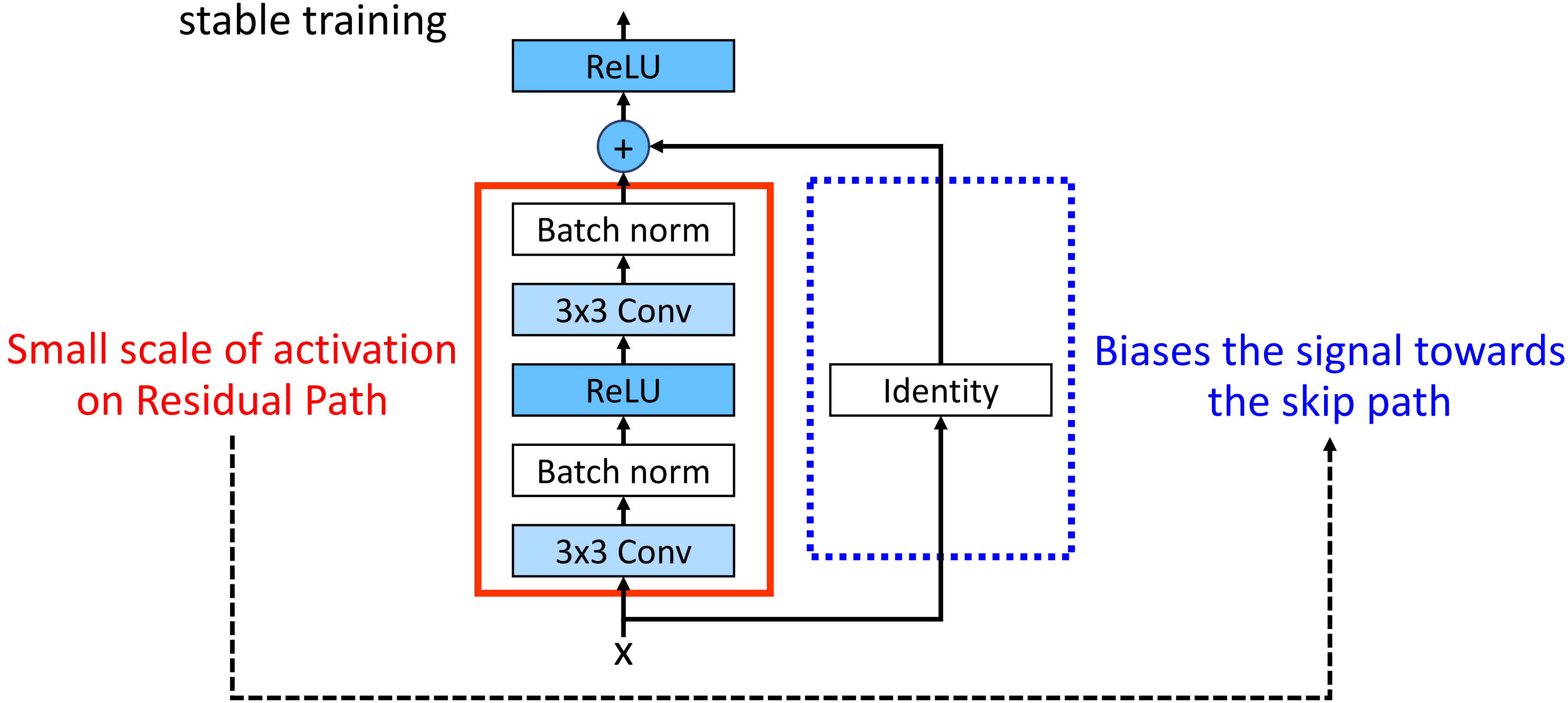
- BN eliminates mean-shift by making mean activation zero [Brock'21]

*source : Bjorck et al., "Understanding Batch Normalization", NeurIPS 2018

Why does Batch Normalization (BN) work?

2. BN downscales the residual branch

- BN is commonly applied to residual path of ResNet [He'16]
- This reduces the scale of activations on residual branches at initialization
  - Biases the signal towards the skip path [De & Smith'20], which enables stable training



Why does Batch Normalization (BN) work?

3. BN has a regularizing effect

- Noise in the batch statistics acts as a regularizer [Luo'18]
  - Using small batch for computing statistics leads to noise in statistics

Computed using samples within batch

$$y_{ichw} = \gamma \cdot \left(\frac{x_{ichw} - \boxed{\mu_c}}{\boxed{\sigma_c}} \right) + \beta$$

$$x_{ichw} \in X^{N \times C \times H \times W}$$

$$\mu_c = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{ichw}$$

$$\sigma_c^2 = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{ichw} - \mu_i)^2$$

- [Hoffer'17] show that test accuracy of batch-normalized network can further be improved by tuning batch size

*source: Hoffer et al., "Train longer, generalize better: closing the generalization gap in large batch training of neural networks", NeurIPS 2017
*source : Luo et al., "Towards Understanding Regularization in Batch Normalization", ICLR 2018

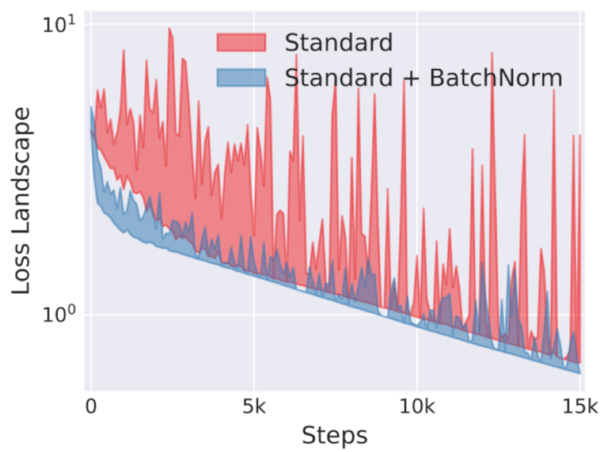
Why does Batch Normalization (BN) work?

4. BN allows efficient large-batch training

- [Santurkar'18] show that BN smoothens the loss landscape
- This increases the largest stable learning rate [Bjorck'18]
    - Which is essential to large-batch training

Sensitivity of Loss to learning rate

$$\mathcal{L}(x + \eta \nabla \mathcal{L}(x)), \eta \in [0.05, 0.4]$$



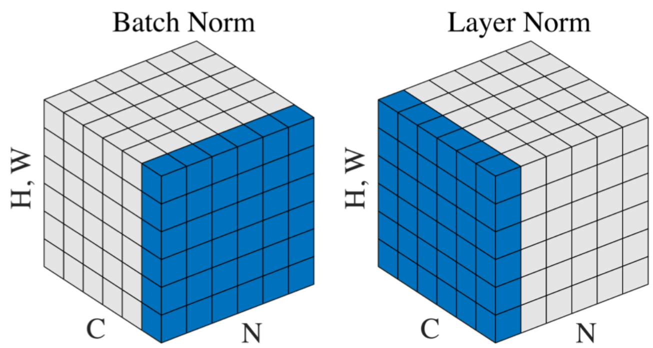
Sensitivity of Gradient to learning rate

$$||\nabla \mathcal{L}(x) - \nabla \mathcal{L}(x + \eta \nabla \mathcal{L}(x))||, \eta \in [0.05, 0.4]$$

*source: Santurkar et al., "How Does Batch Normalization Help Optimization", NeurIPS 2018
*source : Bjorck et al., "Understanding Batch Normalization", NeurIPS 2018
* source : https://gradientscience.org/batchnorm/

Variants of Batch Normalization: Layer Normalization

- **Layer Normalization** [LN; Ba'16]
  - LN normalizes over channels, instead of batch



$$x_{ichw} \in X^{N \times C \times H \times W}$$

$$y_{ichw} = \gamma \cdot \left(\frac{x_{ichw} - \mu_i}{\sigma_i} \right) + \beta$$

$$\mu_i = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{ichw}$$

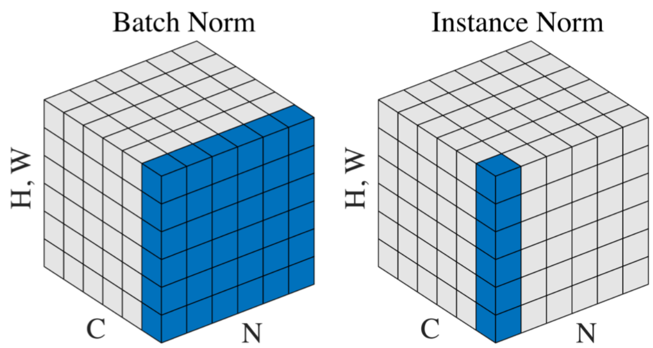
$$\sigma_i^2 = \frac{1}{CHW} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{ichw} - \mu_i)^2$$

  - (+) Works well for small-batch training
  - (+) Effective for sequential models
    - BN requires different statistics for each time-step of RNNs

* source : Ba et al., "Layer Normalization", arXiv 2016
* source : Wu and He., "Group Normalization", ECCV 2018

Variants of Batch Normalization: Instance Normalization

- **Instance Normalization** [IN; Ulyanov'16]
  - IN normalizes over each channel, instead of batch



$$x_{ichw} \in X^{N \times C \times H \times W}$$

$$y_{ichw} = \gamma \cdot \left(\frac{x_{ichw} - \mu_{ic}}{\sigma_{ic}} \right) + \beta$$

$$\mu_{ic} = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{ichw}$$

$$\sigma_{ic}^2 = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{ichw} - \mu_i)^2$$

- (+) Works well for small-batch training
- (+) Effective for generative models
  - Can remove instance-wise difference

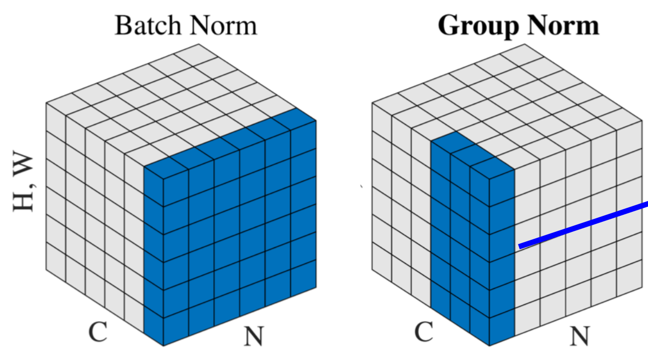
High contrast

Low contrast

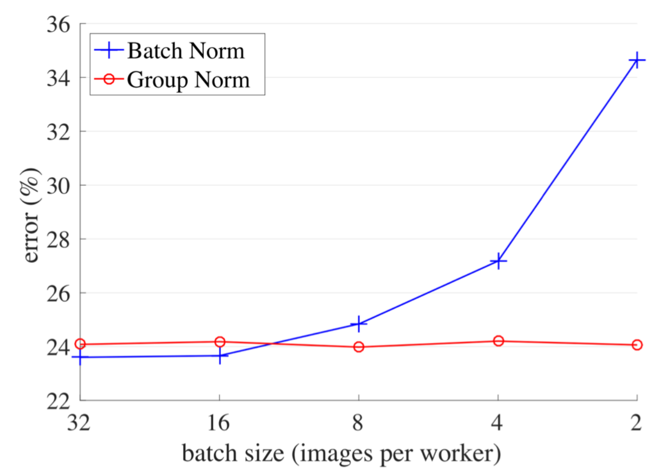
* source : Ulyanov et al., "Instance Normalization: The Missing Ingredient for Fast Stylization", arXiv 2016
* source : Wu and He., "Group Normalization", ECCV 2018

Variants of Batch Normalization: Group Normalization

- **Group Normalization** [GN; Wu'18]
  - Performance of LN and IN is limited in visual recognition tasks
  - LN normalizes over **G** group of channels, instead of batch
    - Inspired by classical approaches like SIFT/HOG that utilize group-wise features and normalization



Groups are decided by dividing C by G



- (+) Works well for small-batch training
- (+) Effective for visual recognition

- (-) Worse than BN in large-batch training

* source : Wu and He., "Group Normalization", ECCV 2018

Table of Contents

- However, BN also has practical disadvantages
    1. Sensitive to batch size
    2. Computationally expensive
    3. Discrepancy in the behavior of model during training and inference time
    4. Breaks the independence between training examples in the minibatch

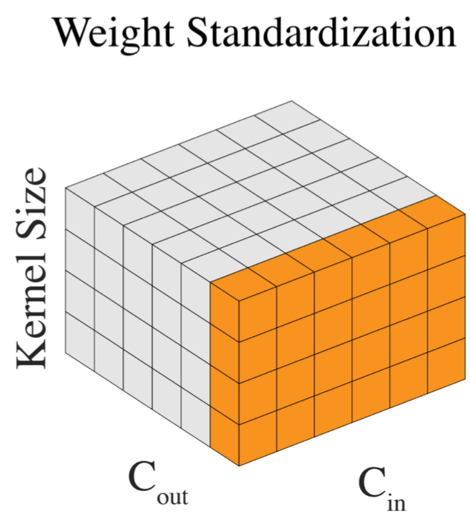
* source : Wu et al., “Group Normalization”, ECCV 2018

- Recently, [Brock'21b] proposed **NFNet that does not utilize BN** but achieves strong results on ImageNet benchmark
  - NFNet removes BN but maintains the strengths of BN

- **Strength 1. Eliminates mean-shift**
  - ➢ NFNet introduces Scaled Weight Standardization [Brock'21a] that reparameterizes the convolutional layer as

$$\hat{W}_{ij} = \frac{W_{ij} - \mu_i}{\sqrt{N}\sigma_i}, \quad (1)$$

where $\mu_i = (1/N)\sum_j W_{ij}$, $\sigma_i^2 = (1/N)\sum_j (W_{ij} - \mu_i)^2$,



(+) Computationally cheap
(+) No discrepancy in training / test behavior
(+) No dependence between batch samples

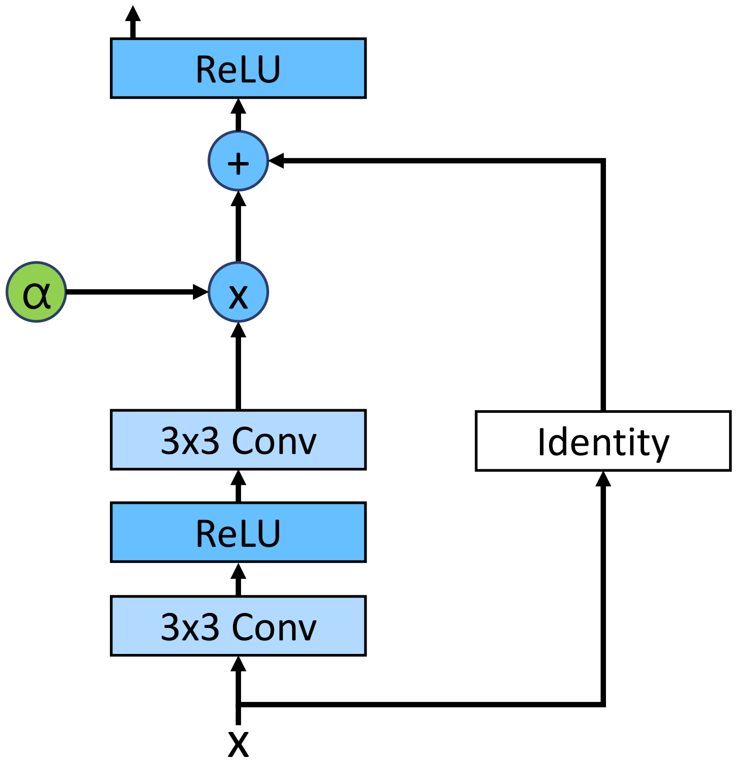
* source : Brock et al., "High-Performance Large-Scale Image Recognition Without Normalization", arXiv 2021
* source : Qiao et al., "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization", arXiv 2019

Understanding Batch Normalization

- Recently, [Brock'21b] proposed **NFNet that does not utilize BN** but achieves strong results on ImageNet benchmark
  - NFNet removes BN but maintains the strengths of BN

- **Strength 2. Downscales the residual branch**
  - ➢ NFNet introduces a small scalar to suppress the scale of activations on residual branch

α is set to small value (=0.2)

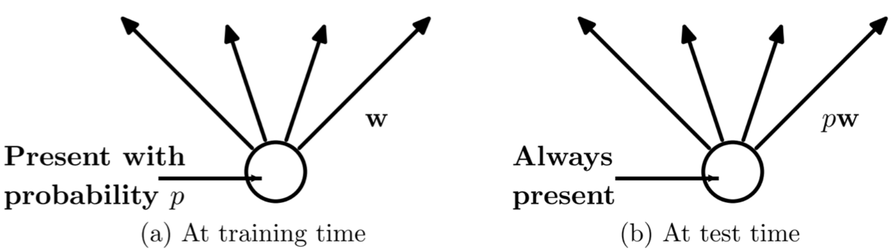


* source : Brock et al., "High-Performance Large-Scale Image Recognition Without Normalization", arXiv 2021

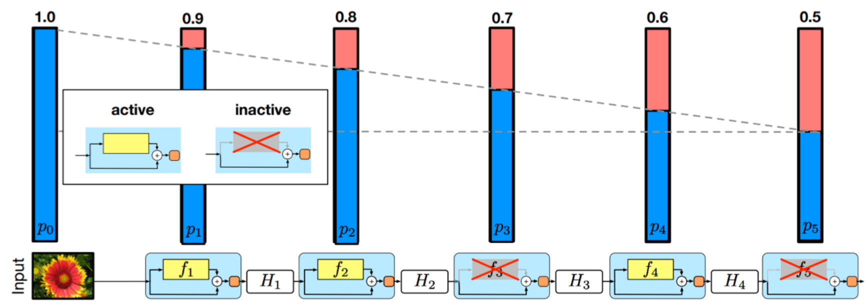
Understanding Batch Normalization

- Recently, [Brock'21b] proposed **NFNet that does not utilize BN** but achieves strong results on ImageNet benchmark
  - NFNet removes BN but maintains the strengths of BN

- **Strength 3. Has a regularizing effect**
  - ➢ NFNet utilizes additional regularizations



Dropout
[Srivastava'14]



Stochastic Depth
[Huang'16]

* source : Brock et al., "High-Performance Large-Scale Image Recognition Without Normalization", arXiv 2021
* source : Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JLMR 2014
* source : Huang et al., "Deep Networks with Stochastic Depth", ECCV 2016

- Recently, [Brock'21b] proposed **NFNet that does not utilize BN** but achieves strong results on ImageNet benchmark
  - NFNet removes BN but maintains the strengths of BN

- **Strength 4. Allows efficient large-batch training**
  - ➢ NFNet introduces Adaptive Gradient Clipping

Measures how much single gradient update will change weight

$$G_i^\ell \rightarrow \begin{cases} \boxed{\lambda \frac{\|W_i^\ell\|_F^\star}{\|G_i^\ell\|_F} G_i^\ell} & \text{if } \boxed{\frac{\|G_i^\ell\|_F}{\|W_i^\ell\|_F^\star}} > \lambda, \\ G_i^\ell & \text{otherwise.} \end{cases}$$

If update is too drastic, clip the gradient

$\|\cdot\|_F$: frobenius norm

G^l : Gradient of l-th layer

W^l : Weight of l-th layer

G_i^l : i-th row of of matrix G^l

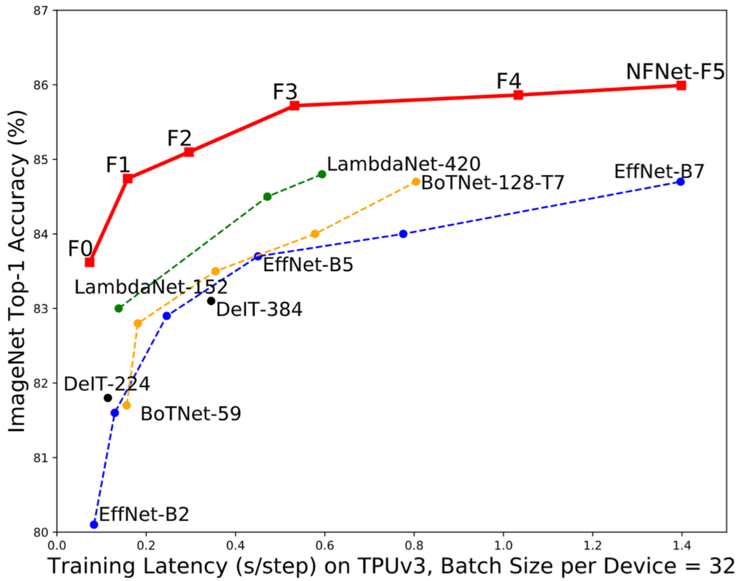
W_i^l : i-th row of of matrix W^l

- (+) Not sensitive to clipping threshold hyperparameter

* source : Brock et al., "High-Performance Large-Scale Image Recognition Without Normalization", arXiv 2021

Understanding Batch Normalization

- Recently, [Brock'21b] proposed **NFNet that does not utilize BN** but achieves strong results on ImageNet benchmark
  - NFNet removes BN but maintains the strengths of BN



Model	#FLOPs	#Params	Top-1	Top-5	TPUv3 Train	GPU Train
ResNet-50	4.10B	26.0M	78.6	94.3	41.6ms	35.3ms
EffNet-B0	0.39B	5.3M	77.1	93.3	51.1ms	44.8ms
SENet-50	4.09B	28.0M	79.4	94.6	64.3ms	59.4ms
NFNet-F0	**12.38B**	**71.5M**	**83.6**	**96.8**	**73.3ms**	**56.7ms**
EffNet-B3	1.80B	12.0M	81.6	95.7	129.5ms	116.6ms
LambdaNet-152	–	51.5M	83.0	96.3	138.3ms	135.2ms
SENet-152	19.04B	66.6M	83.1	96.4	149.9ms	151.2ms
BoTNet-110	10.90B	54.7M	82.8	96.3	181.3ms	–
NFNet-F1	**35.54B**	**132.6M**	**84.7**	**97.1**	**158.5ms**	**133.9ms**
EffNet-B4	4.20B	19.0M	82.9	96.4	245.9ms	221.6ms
BoTNet-128-T5	19.30B	75.1M	83.5	96.5	355.2ms	–
NFNet-F2	**62.59B**	**193.8M**	**85.1**	**97.3**	**295.8ms**	**226.3ms**
SENet-350	52.90B	115.2M	83.8	96.6	593.6ms	–
EffNet-B5	9.90B	30.0M	83.7	96.7	450.5ms	458.9ms
LambdaNet-350	–	105.8M	84.5	97.0	471.4ms	–
BoTNet-77-T6	23.30B	53.9M	84.0	96.7	578.1ms	–
NFNet-F3	**114.76B**	**254.9M**	**85.7**	**97.5**	**532.2ms**	**524.5ms**
LambdaNet-420	–	124.8M	84.8	97.0	593.9ms	–
EffNet-B6	19.00B	43.0M	84.0	96.8	775.7ms	868.2ms
BoTNet-128-T7	45.80B	75.1M	84.7	97.0	804.5ms	–
NFNet-F4	**215.24B**	**316.1M**	**85.9**	**97.6**	**1033.3ms**	**1190.6ms**
EffNet-B7	37.00B	66.0M	84.7	97.0	1397.0ms	1753.3ms
DeIT 1000 epochs	–	87.0M	85.2	–	–	–
EffNet-B8+MaxUp	62.50B	87.4M	85.8	–	–	–
NFNet-F5	**289.76B**	**377.2M**	**86.0**	**97.6**	**1398.5ms**	**2177.1ms**
NFNet-F5+SAM	289.76B	377.2M	86.3	97.9	1958.0ms	–
NFNet-F6+SAM	**377.28B**	**438.4M**	**86.5**	**97.9**	**2774.1ms**	–

* source : Brock et al., "High-Performance Large-Scale Image Recognition Without Normalization", arXiv 2021

Table of Contents

Summary

- Deep learning is **scaling up** fast
  - Accordingly, many optimization techniques have been proposed for scalable and efficient training

- **SGD** is an essential ingredient for training deep neural networks
  - Momentum/adaptive optimizers are widely used
  - Learning rate scheduling is often important
  - Optimization becomes more tricky when scaling to large batch sizes

- **Batch Normalization** is also extensively used
  - Chances are, there exists some variant of BN that will work for your application
  - Many explanations as to why BN works
  - There is a recent effort to do away with BN altogether (NFNet)

References

- [Nesterov' 1983] Nesterov. "A method of solving a convex programming problem with convergence rate O(1/k^2)." 1983
  link: http://mpawankumar.info/teaching/cdt-big-data/nesterov83.pdf
- [Duchi et al 2011] "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011
  link : http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf
- [Tieleman' 2012] Geoff Hinton's Lecture 6e of Coursera Class
  link : http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- [Zeiler' 2012] Zeiler, M. D. "ADADELTA: An Adaptive Learning Rate Method"
  link : https://arxiv.org/pdf/1212.5701.pdf
- [Kingma and Ba., 2015] Kingma, D., Ba, J. "Adam: A method for stochastic optimization." ICLR 2015
  link : https://arxiv.org/pdf/1412.6980.pdf
- [Dozat' 2016] Dozat, T. "Incorporating Nesterov Momentum into Adam." ICLR Workshop 2016
  link : http://cs229.stanford.edu/proj2015/054_report.pdf
- [Loshchilov et al., 2019] Loshchilov, I., Hutter, F. "Decoupled Weight Decay Regularization." ICLR 2019.
  link : https://arxiv.org/pdf/1711.05101.pdf
- [Smith et al., 2017] Smith, Samuel L., Pieter-Jan Kindermans, Quoc V. Le. "Don't Decay the Learning Rate, Increase the Batch Size." ICLR 2017.
  link : https://openreview.net/pdf?id=B1Yy1BxCZ
- [Liu et al., 2020] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., Han, J. "On the Variance of the Adaptive Learning Rate and Beyond." ICLR 2020.
  link: https://arxiv.org/pdf/1908.03265.pdf
- [Smith' 2015] Smith, Leslie N. "Cyclical learning rates for training neural networks."
  link : https://arxiv.org/pdf/1506.01186.pdf
- [Loshchilov et al., 2017] Loshchilov, I., Hutter, F. "SGDR: Stochastic Gradient Descent with Warm Restarts." ICLR 2017.
  link : https://arxiv.org/pdf/1608.03983.pdf

References

- [Mahajan et al., 2018] Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., Maaten, L. "Exploring the Limits of Weakly Supervised Pretraining." ECCV 2018.
  link: https://arxiv.org/pdf/1805.00932.pdf
- [Yu et al., 2018] You, Y., Zhang, Z., Hsieh, C., Demmel, J., Keutzer, K. "ImageNet Training in Minutes." 2018
  link: https://arxiv.org/pdf/1709.05011.pdf
- [Keskar et al., 2017] Keskar, N., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." ICLR 2017
  link: https://arxiv.org/pdf/1609.04836.pdf
- [Li et al., 2018] Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T. "Visualizing the Loss Landscape of Neural Nets." NeurIPS 2018
  link: https://arxiv.org/pdf/1712.09913.pdf
- [Goyal et al., 2018] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Aapo, K., Tulloch, A., Jia, Y., He, K. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour." 2018
  link: https://arxiv.org/pdf/1706.02677.pdf
- [He et al., 2019] He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Li, M. "Bag of Tricks for Image Classification with Convolutional Neural Networks." CVPR 2019
  link: https://arxiv.org/pdf/1812.01187.pdf
- [You et al., 2017] You, Y., Gitman, I., Ginsburg, B. "Large Batch Training of Convolutional Networks." 2017
  link: https://arxiv.org/pdf/1708.03888.pdf
- [You et al., 2020] You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., Hsieh, C. "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes." ICLR 2020
  link: https://arxiv.org/pdf/1904.00962.pdf
- [Nado et al., 2021] Nado, Z., Gilmer, J., Shallue, C., Anil, R., Dahl, G. "A Large Batch Optimizer Reality Check: Traditional, Generic Optimizers Suffice Across Batch Sizes." 2021
  link: https://arxiv.org/pdf/2102.06356.pdf

References

- [Ioffe and Szegedy., 2015] Ioffe, S., Szegedy, C. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” 2015.
  link: https://arxiv.org/pdf/1502.03167.pdf
- [He et al., 2016] He, K., Zhang, X., Ren, S., Sun, J. “Deep Residual Learning for Image Recognition.” CVPR 2016.
  link: https://arxiv.org/pdf/1512.03385.pdf
- [De and Smith., 2020] De, S., Smith, S. “Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks.” NeurIPS 2020.
  link: https://arxiv.org/pdf/2002.10444.pdf
- [Brock et al., 2021a] Brock, A., De, S., Smith, S. “Characterizing Signal Propagation to Close the Performance Gap in Unnormalized ResNets.” ICLR 2021.
  link: https://arxiv.org/pdf/2101.08692.pdf
- [Luo et al., 2018] Luo, P., Wang, X., Shao, W., Peng, Z. “Towards Understanding Regularization in Batch Normalization.” ICLR 2018
  link: https://arxiv.org/pdf/1809.00846.pdf
- [Hoffer et al., 2017] Hoffer, E., Hubara, I., Soudry, D. “Train longer, generalize better: closing the generalization gap in large batch training of neural networks.” NeurIPS 2017
  link: https://arxiv.org/pdf/1705.08741.pdf
- [Santukar et al., 2018] Santurkar, S., Tsipras, D., Ilyas, A., Madry, A. “How Does Batch Normalization Help Optimization?” NeurIPS 2018
  link: https://arxiv.org/pdf/1805.11604.pdf
- [Bjorck et al., 2018] Bjorck, J., Gomes, C., Selman, B., Weinberger, K. “Understanding Batch Normalization.” NeurIPS 2018
  link: https://arxiv.org/pdf/1806.02375.pdf
- [Wu and He, 2018] Wu, Y., He, K. “Group Normalization.” ECCV 2018.
  link: https://arxiv.org/pdf/1803.08494.pdf

References

- [Brock et al., 2021b] Brock, A., De, S., Smith, S., Simonyan, K. “High-Performance Large-Scale Image Recognition Without Normalization.” 2021.
  link: https://arxiv.org/pdf/2102.06171.pdf
- [Qiao et al., 2019] Qiao, S., Wang, H., Liu, C., Shen, W., Yuille, A. “Micro-Batch Training with Batch-Channel Normalization and Weight Standardization.” 2019.
  link: https://arxiv.org/pdf/1903.10520.pdf
- [Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” JLMR 2014.
  link: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
- [Huang et al. 2016] Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K. “Deep Networks with Stochastic Depth.” ECCV 2016.
  link: https://arxiv.org/pdf/1603.09382.pdf