Optimization Techniques

AI602: Recent Advances in Deep Learning

Lecture 1

Slide made by

Jongheon Jeong, Insu Han, Seunghyun Lee and Younggyo Seo

KAIST EE & AI

v1

Overview: Empirical Risk Minimization (ERM)

Empirical risk minimization: Find parameters that minimizes the empirical risk

- A collection of samples (or training set): $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$
- A predictive model: $f(\mathbf{x}_i; \boldsymbol{\theta}) \approx y_i$ parameterized by $\boldsymbol{\theta}$

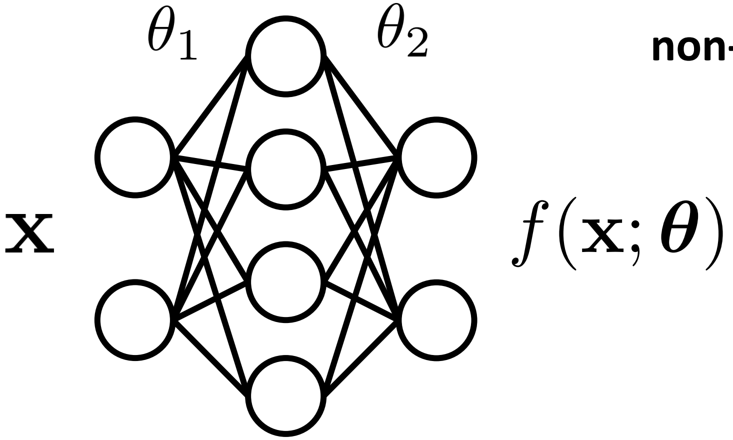
$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) := \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$$

- $\ell(\cdot, \cdot)$: A loss function - e.g., mean squared error (MSE), cross entropy

Example: Regression with a K-layer neural network via MSE

$$f(\mathbf{x}; \boldsymbol{\theta}) := \theta_K^\top \phi(\theta_{K-1}^\top \phi(\cdots \phi(\theta_1^\top \mathbf{x})))$$

non-linearity (e.g., $\phi(\cdot) = \mathrm{ReLU}(\cdot) := \max(0, \ \cdot)$)



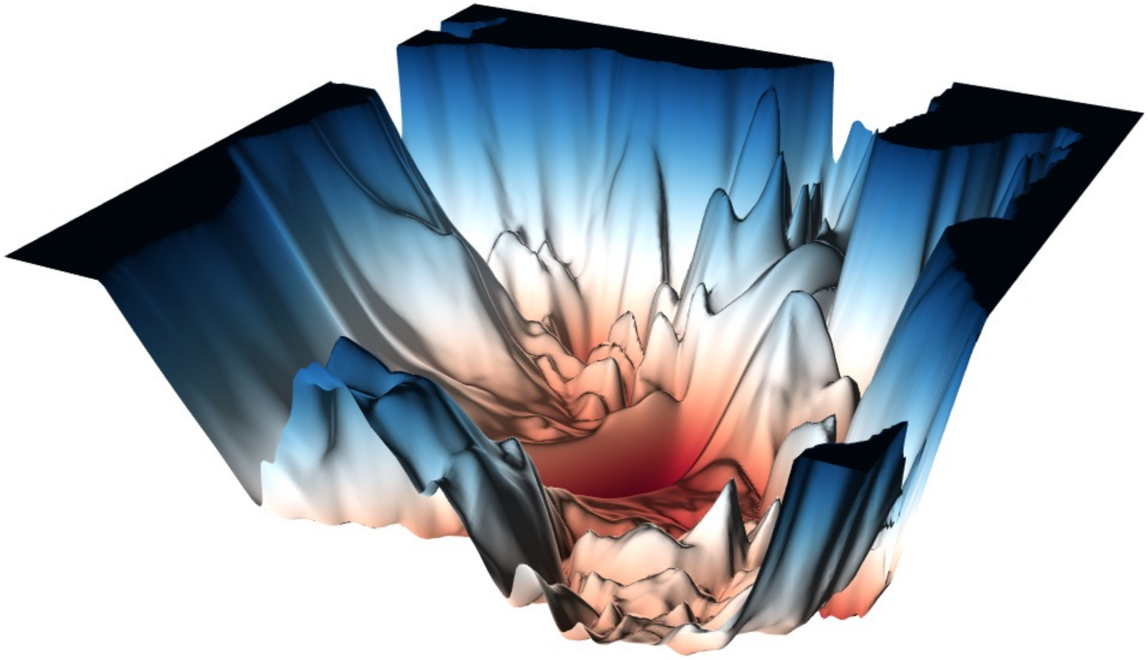
$$\ell_{\mathtt{MSE}}(\hat{y}, y) := (\hat{y} - y)^2$$

$$L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i} \ell_{\mathtt{MSE}}(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$$

Overview: Optimization Techniques for Deep Learning

Deep learning is heavily relying on large-scale, non-convex optimization

- What is the key challenges in optimizing deep neural networks?
- How to practically overcome such optimization difficulties?



Loss surface of a neural net (ResNet-50)

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

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Gradient Descent (GD)

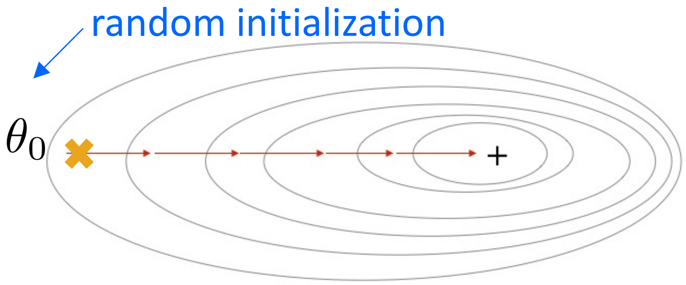
Gradient descent (GD) updates parameters iteratively by taking gradient

parameters

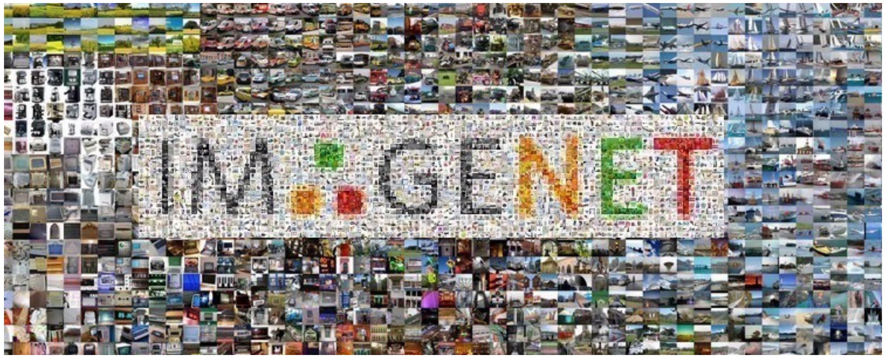
empirical risk

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

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 $\boldsymbol{n} = \mathbf{1,281,167}$ images for training



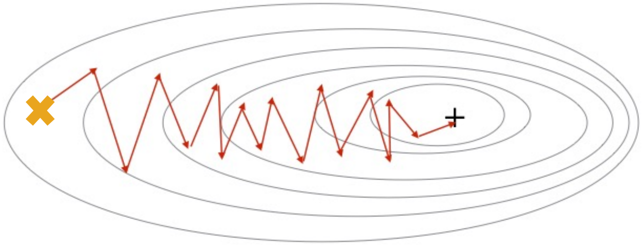
1.2M of 256 × 256 RGB images
≈ 236 GB memory

Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) use a batch of samples to approximate GD

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

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



- In practice, minibatch size $|\mathcal{B}|$ typically ranges from 32 to 512 (single machine)
- Theoretically, SGD can find the global optimum given that:
  1. The loss function is convex
  2. The gradient estimates have a bounded variance
  3. Diminishing learning rates

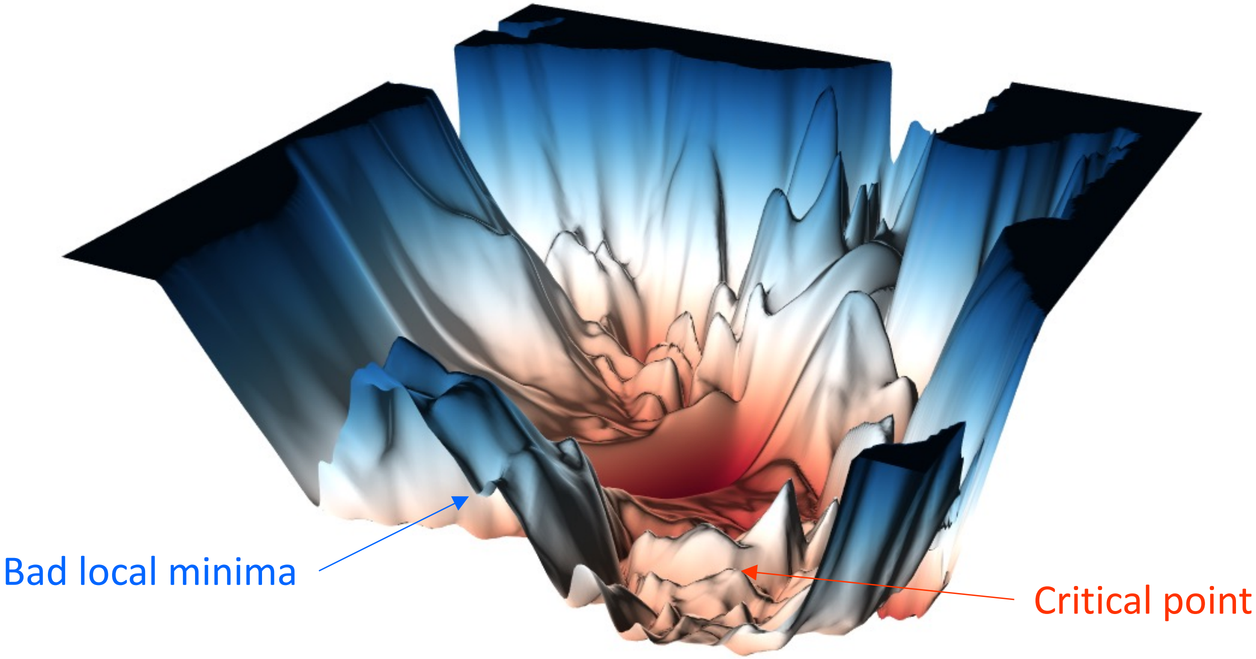
- Nevertheless, in many practical problems, SGD makes some challenges

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

Stochastic Gradient Descent (SGD)

Practical challenges in non-convex ("deep") SGD:

1. The loss function includes many local minima and critical points
2. SGD can be too noisy and might be unstable → momentum
3. Hard to find a good learning rate → adaptive learning rate
4. Gradients are often vanish/explode → normalization



Loss surface of a neural net (ResNet-50)

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

Table of Contents

Part 1. Basics
- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics
- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM
- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Adaptive Optimizers: Momentum-based SGD

1. Momentum gradient descent

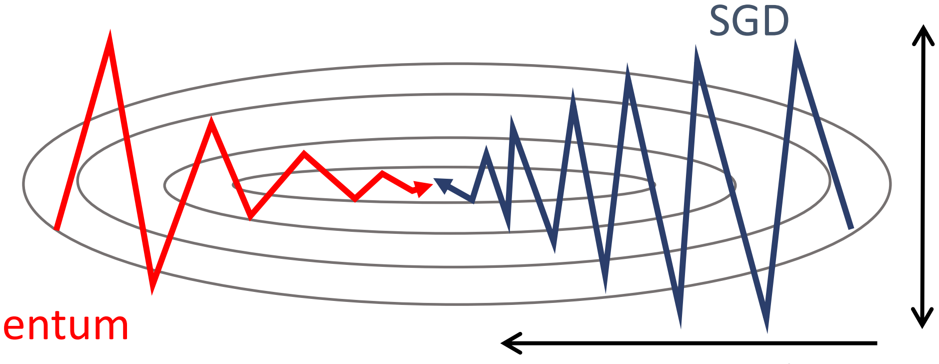
- Add a decaying term of previous gradients (momentum)

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

- 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



Adaptive Optimizers: Momentum-based SGD

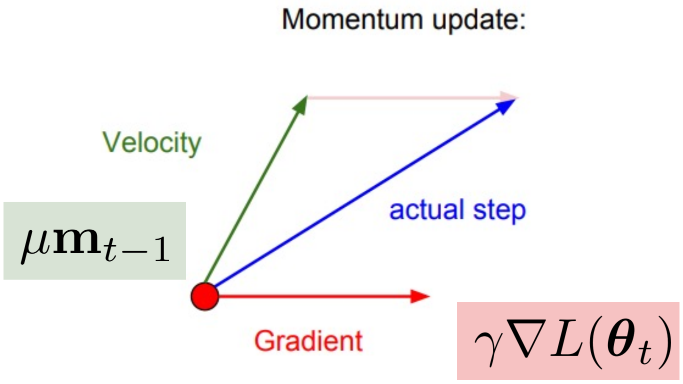
1. Momentum gradient descent

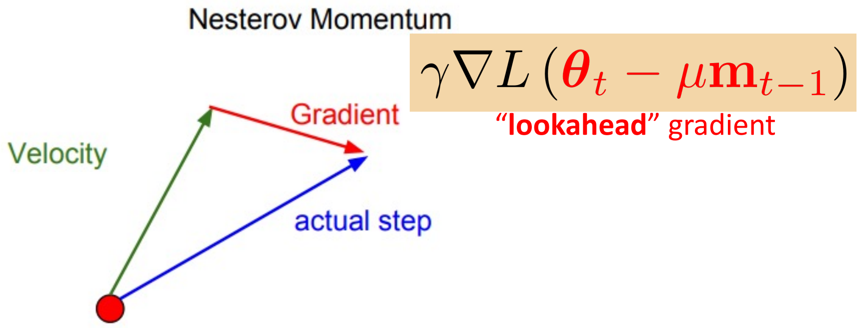
- Add a decaying term of previous gradients (momentum)

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

- (–) Momentum can fail to converge even for simple convex optimizations
- **Nesterov's accelerated gradient** (NAG) [Nesterov'83]
  - Use gradients at approximate future positions instead:

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





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

Adaptive Optimizers: Adaptive Learning Rate

2. Adaptively changing learning rate

- **AdaGrad** [Duchi'11] re-scales learning rates based on previous gradients

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

- (–) Learning rates strictly decrease and become too small for large iterations

- **RMSProp** [Tieleman'12] uses the moving average of the squared gradients

$$\boldsymbol{v}_{t+1} = \underbrace{\mu}_{\text{preservation ratio } \mu \in [0,1]} \boldsymbol{v}_t + (1-\mu) \nabla L\left(\boldsymbol{\theta}_t\right)^2$$

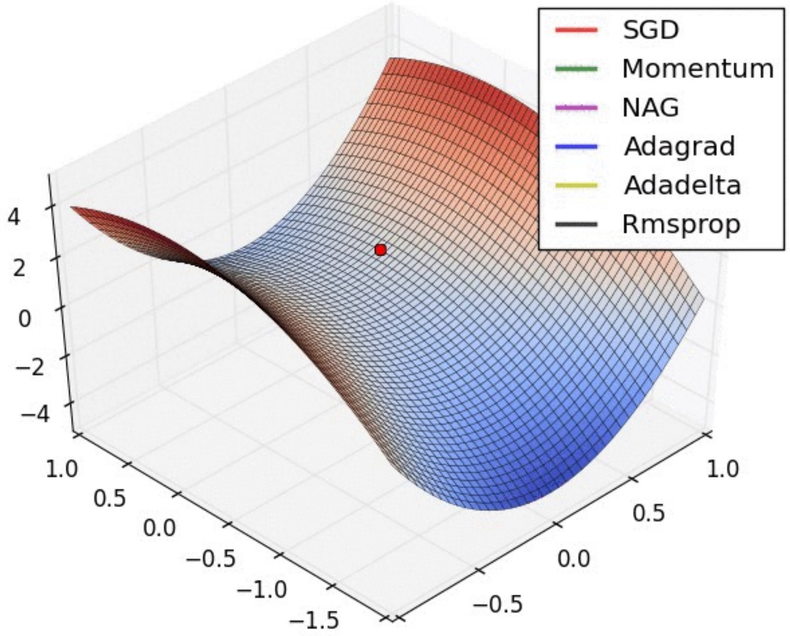
- Other variants also exist, e.g., **Adadelta** [Zeiler'12]

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

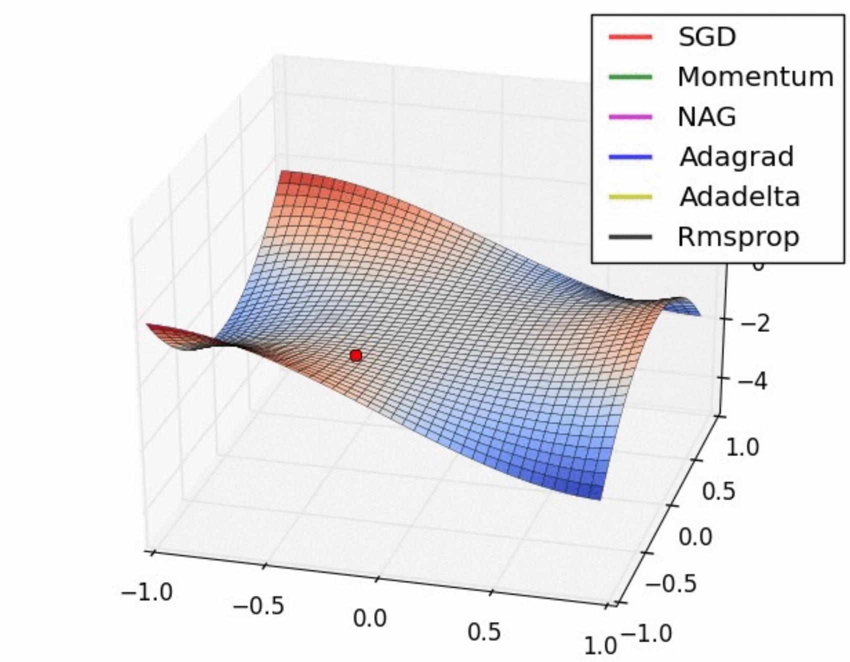
Adaptive Optimizers

Comparison of various optimizers on toy examples

optimization from saddle point



optimization from local optimum



- Adadelta and RMSprop provide the best convergence for the given scenarios

Animations from Alec Radford’s blog

Adaptive Optimizers: Adam

1 + 2. Combination of momentum and adaptive learning rate

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

momentum

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

$$\boldsymbol{m}_{t+1} \leftarrow \mu_1 \boldsymbol{m}_t + (1-\mu_1) \nabla L(\boldsymbol{\theta}_t)$$

$$\boldsymbol{v}_{t+1} \leftarrow \mu_2 \boldsymbol{v}_t + (1-\mu_2) \nabla L(\boldsymbol{\theta}_t)^2$$

avg. squared gradients (RMSProp)

- Other variants exist, e.g., **Adamax** [Kingma'15], **NAdam** [Dozat'16]

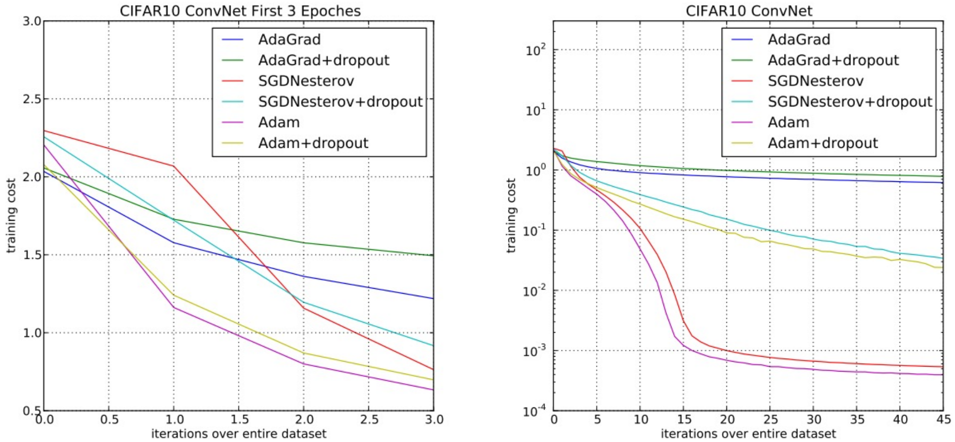


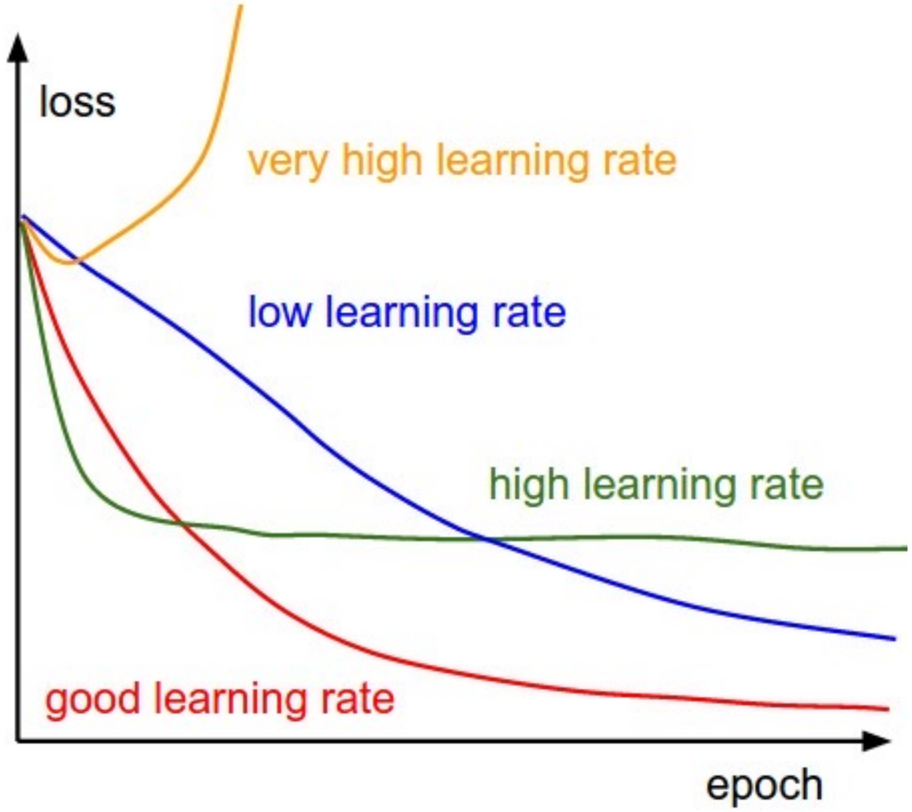
Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015
Dozat, Incorporating Nesterov Momentum into Adam, 2016

Learning Rate Scheduling

3. Learning rate scheduling

- Learning rate is critical for minimizing loss



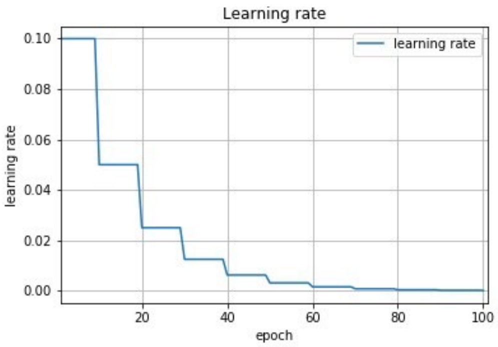
Too **high** → May ignore the narrow valley, can diverge
Too **low** → May fall into the local minima, slow converge

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

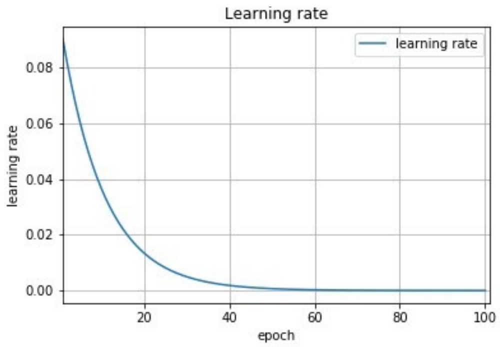
Learning Rate Scheduling

3. Learning rate scheduling: Decay schemes

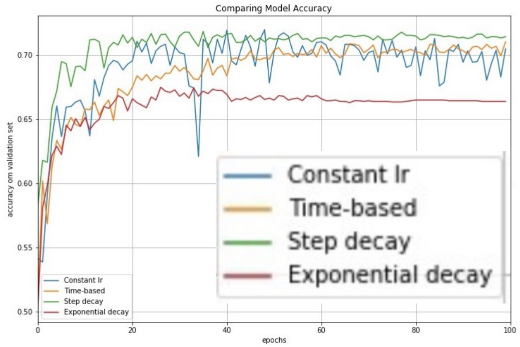
- Constant learning rate often prevents convergence → needs decaying!



Step decay

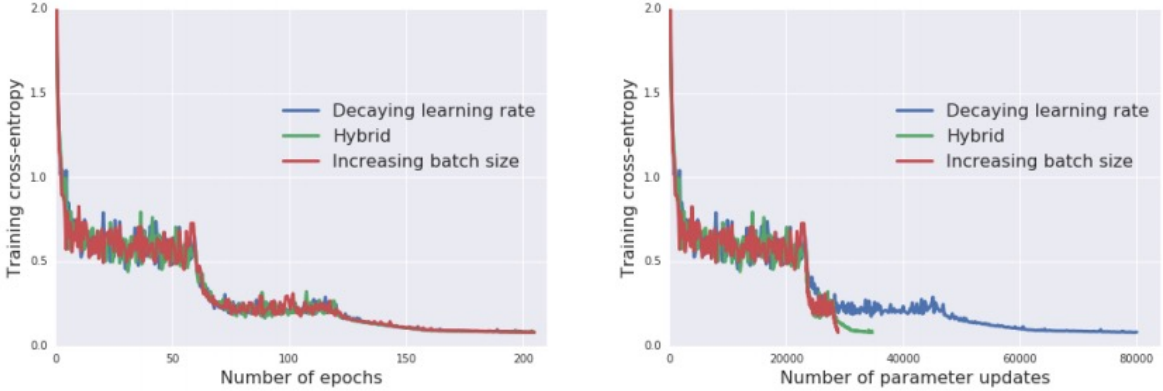


Exponential decay



Test accuracy

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



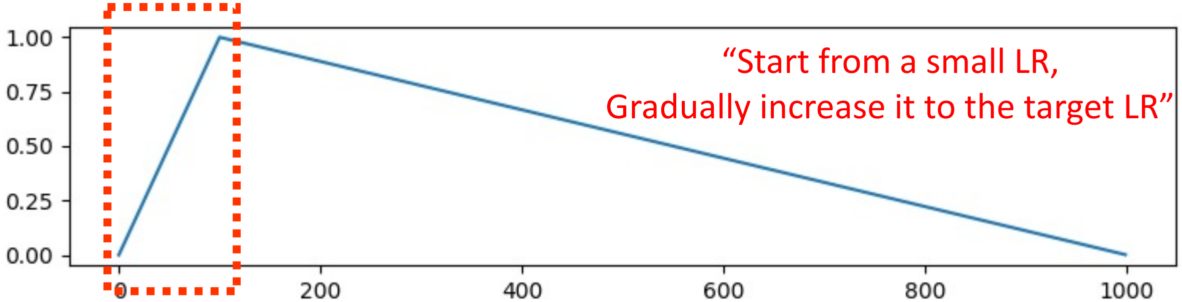
https://towardsdatascience.com/

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

Learning Rate Scheduling

3. Learning rate scheduling: Warm-up

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

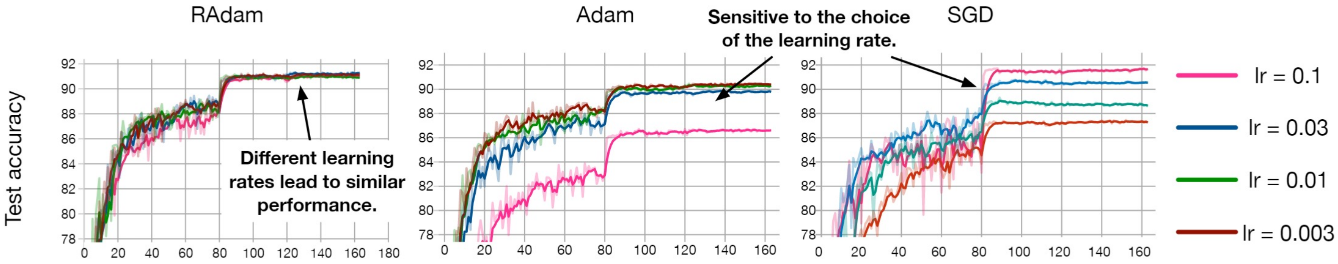


"Start from a small LR,
Gradually increase it to the target LR"

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

$$\theta_{t+1} \leftarrow \theta_t - \frac{\gamma r_t}{\sqrt{v_t}} m_t$$

Variance rectification term
(Check the paper for details)


RAdam

Adam

SGD

Sensitive to the choice of the learning rate.

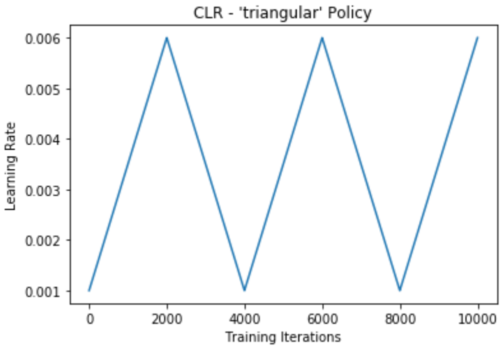
Different learning rates lead to similar performance.

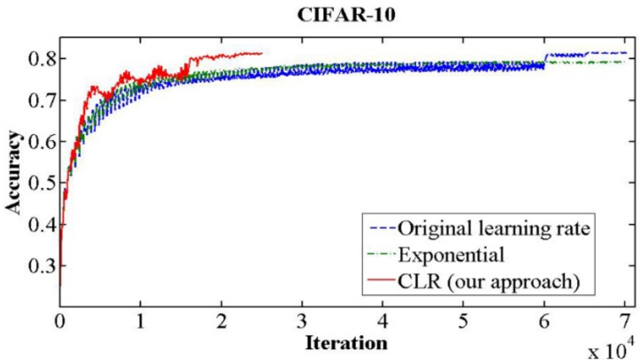
https://huggingface.co/transformers/main_classes/optimizer_schedules.html
Liu et al., "On The Variance of The Adaptive Learning Rate and Beyond.", ICLR 2020

Learning Rate Scheduling

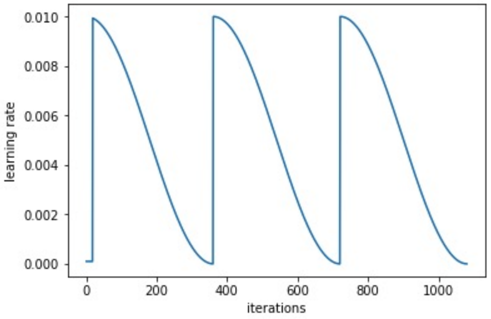
3. Learning rate scheduling: Cyclical schemes

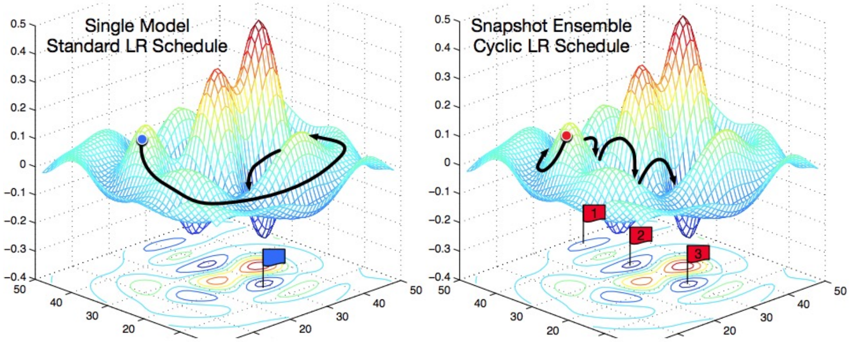
- [Smith’15] proposed **cyclical** learning rate
- **Increasing learning rate** to escape saddle points or bad local minima





- [Loshchilov’17] uses **cosine cycling** and **warm restart**
  - Traverses several local minima by moving up and down the loss surface
  - **Snapshot ensemble**: Ensemble over multiple local minima found so far





Smith., “Cyclical Learning Rates for Training Neural Networks.” 2015
Loshchilov et al., “SGDR: Stochastic Gradient Descent with Warm Restarts.” ICLR 2017

Table of Contents

Part 1. Basics
- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

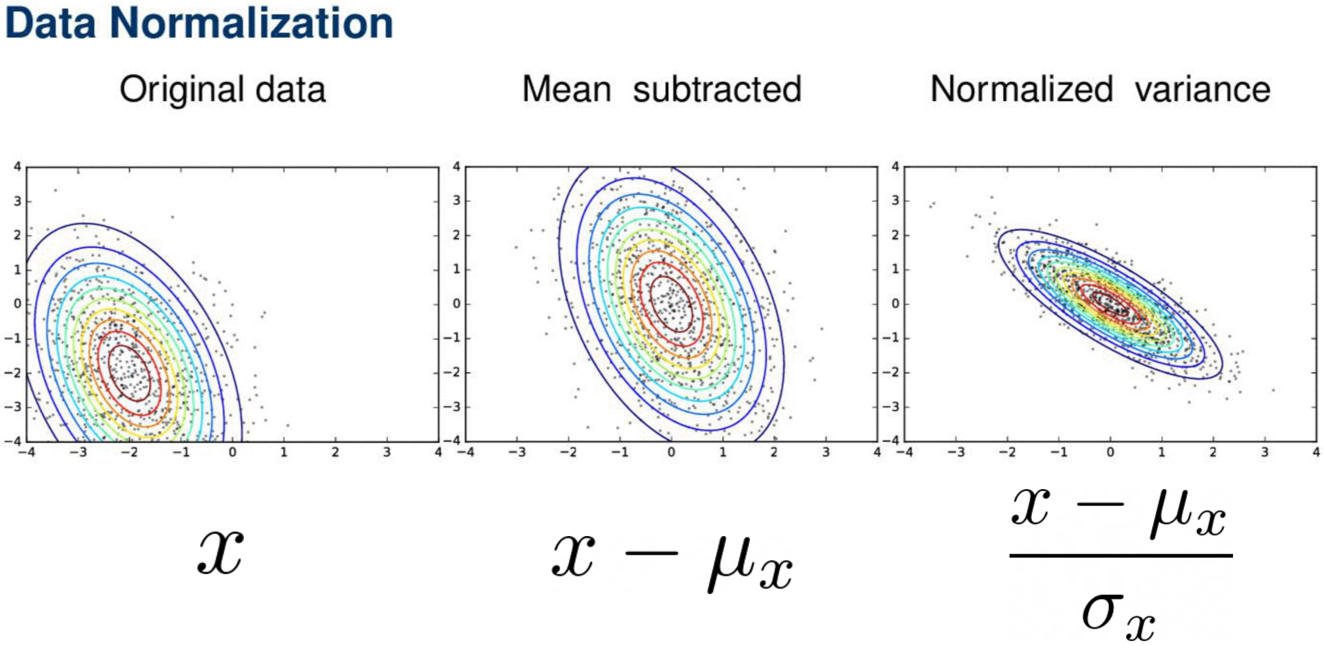
Part 2. Advanced Topics
- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM
- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Normalization

Normalization is a widely-used technique to stabilize training process

- Stabilizes training by adjusting the scale of inputs within unit variance



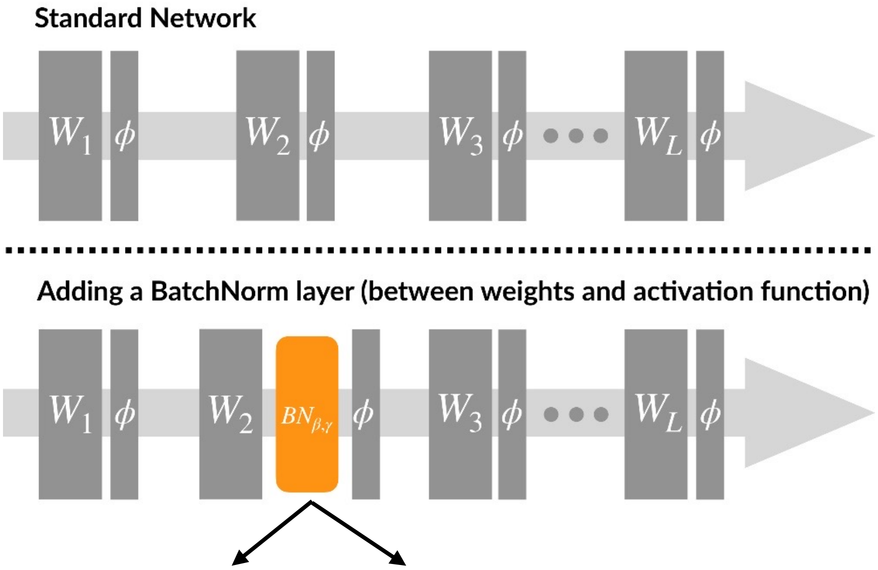
- Commonly used in training recent deep learning models

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

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

Normalization: Batch Normalization [Ioffe'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

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

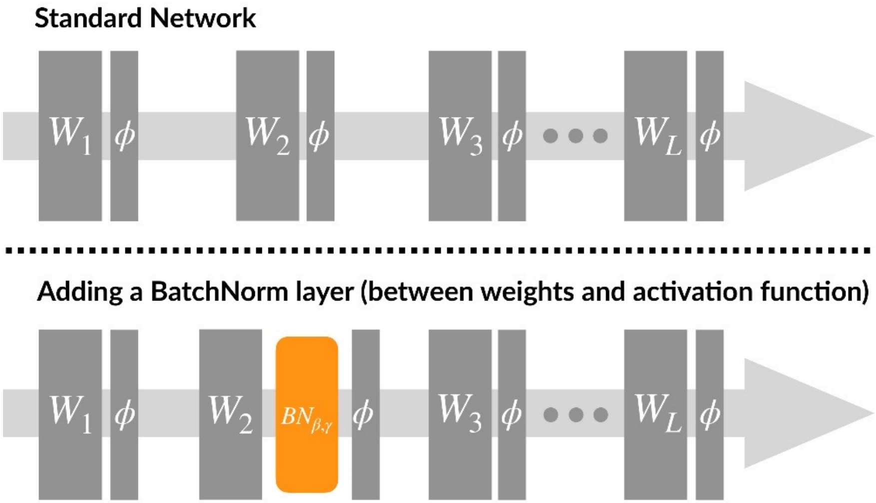
$$\mu_c = \frac{1}{NHW} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{ichw}$$

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

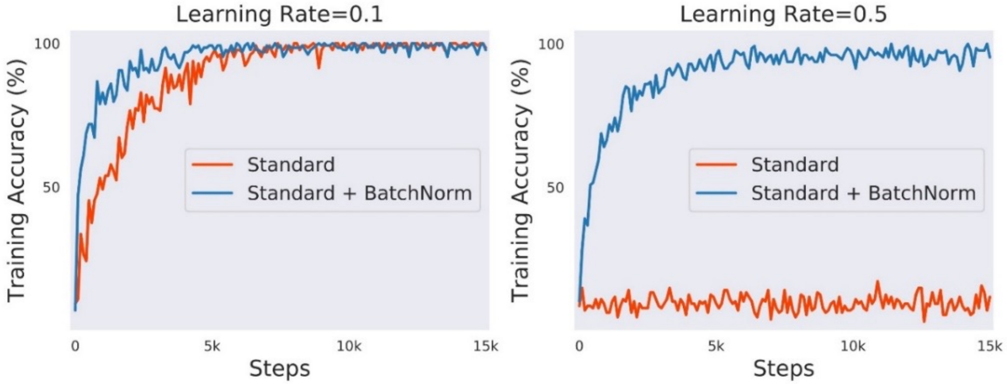
Ioffe and Szegedy., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv 2019
https://gradientscience.org/batchnorm/

Normalization: Batch Normalization [Ioffe'15]

Batch Normalization: Normalize the outputs *within the network*



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



VGG Network on CIFAR10

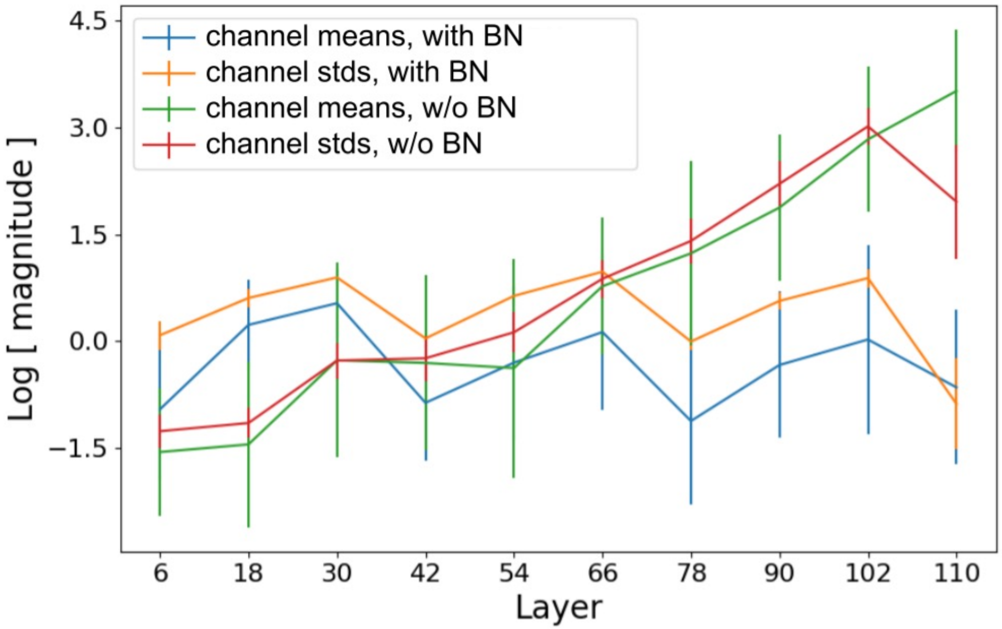
Ioffe and Szegedy., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv 2019
https://gradientscience.org/batchnorm/

Understanding Batch Normalization

Why does Batch Normalization (BN) work?

1. BN eliminates gradual 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 [Bjorck'21]

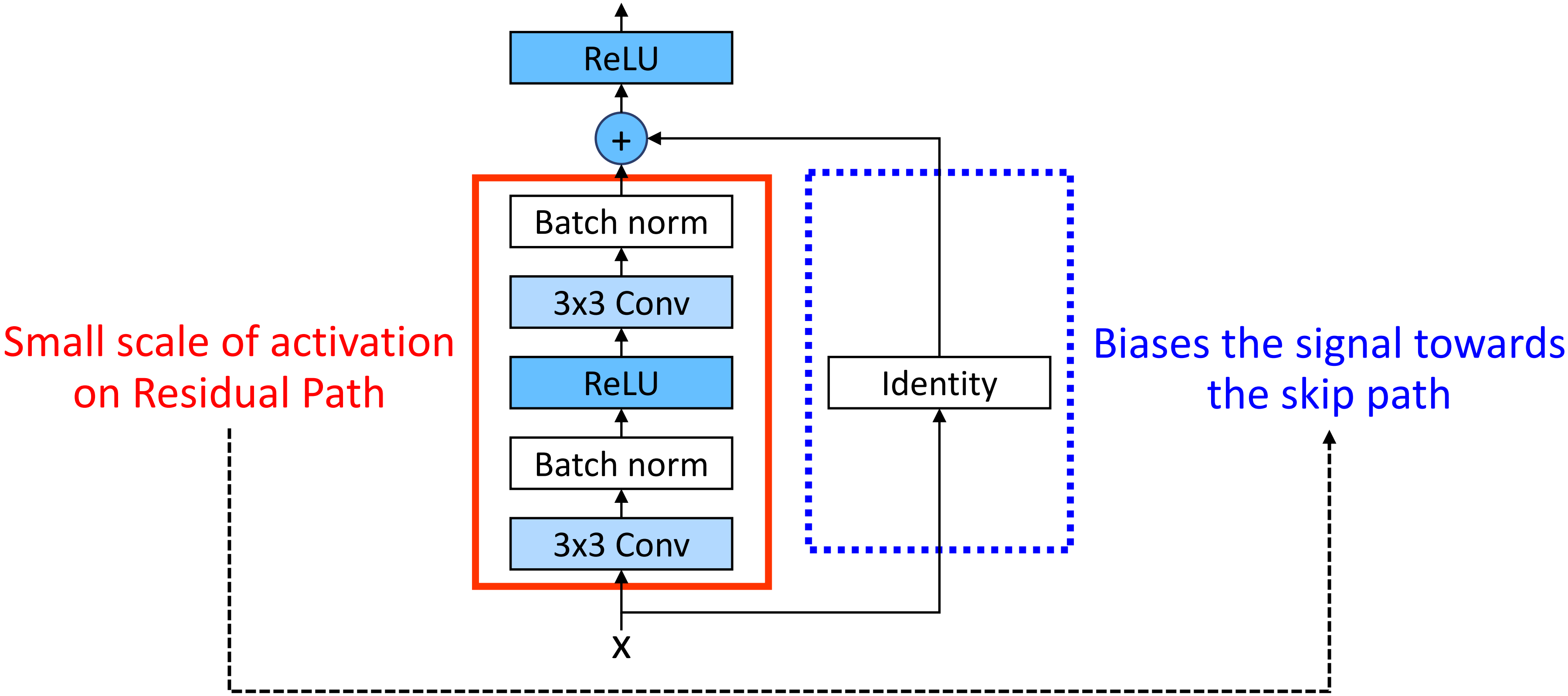
Bjorck et al., "Understanding Batch Normalization", NeurIPS 2018

Understanding Batch Normalization

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’20] → **Stable training**



Understanding Batch Normalization

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

Hoffer et al., “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”, NeurIPS 2017

Luo et al., “Towards Understanding Regularization in Batch Normalization”, ICLR 2018

Understanding Batch Normalization

Why does Batch Normalization (BN) work?

4. BN allows efficient large-batch training

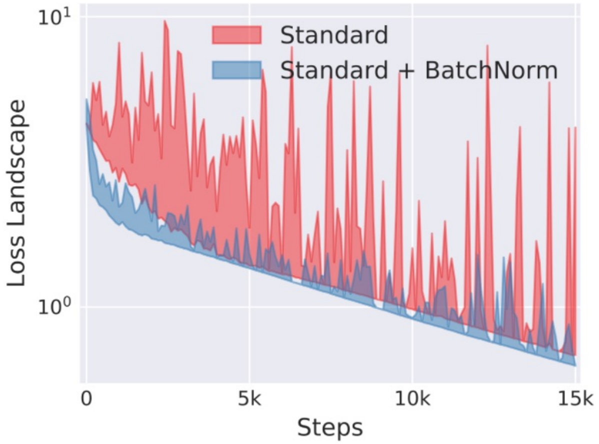
- [Santurkar'18] show that BN smoothens the loss landscape
- This increases the largest stable learning rate [Bjorck'18]
  - … which is important for large-batch training

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

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

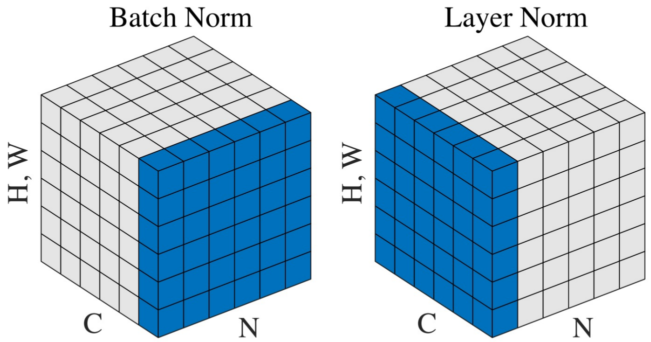


Santurkar et al., "How Does Batch Normalization Help Optimization", NeurIPS 2018
Bjorck et al., "Understanding Batch Normalization", NeurIPS 2018
https://gradientscience.org/batchnorm/

Variants of Batch Normalization

Layer Normalization [LN; Ba'16]

- LN normalizes over channels, instead of batch



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

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

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

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

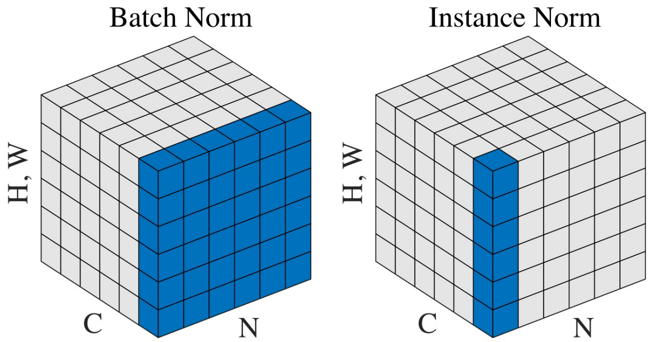
- (+) Works well for small-batch training
- (+) Effective for sequential models
  - BN requires different statistics for each time-step of RNNs

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

Variants of Batch 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 differences

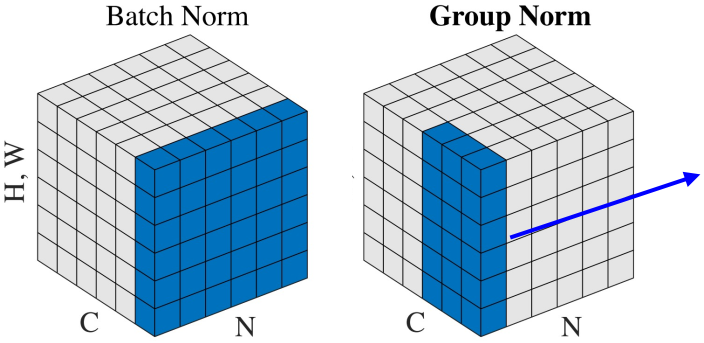
High contrast Low contrast

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

Variants of Batch Normalization

Group Normalization [GN; Wu’18]

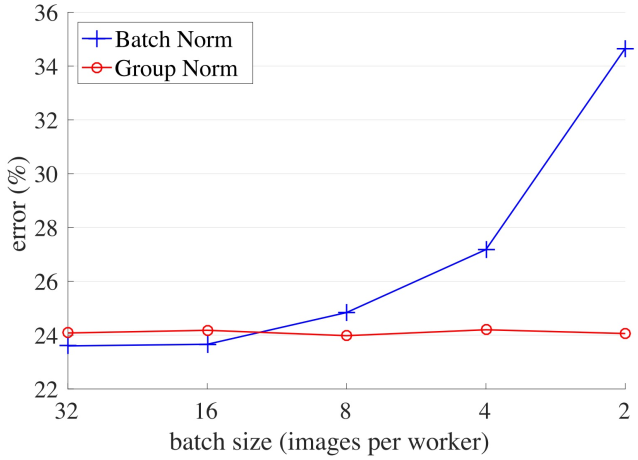
- Performance of LN and IN is limited in visual recognition tasks
- LN normalizes over **G** group of channels, instead of batch
  - Inspired by SIFT/HOG: Group-wise features and normalization



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



Wu and He., “Group Normalization”, ECCV 2018

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

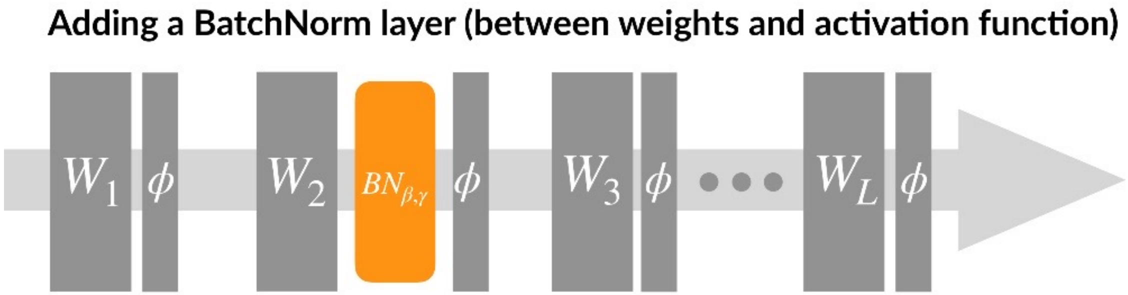
- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Normalization-free Networks

BN has been a key component to bypass optimization problems in deep nets

- Still, BN (and its variants) has significant 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
        - e.g., it makes subtle errors in distributed training

- **What component is essentially needed to stabilize deep nets without BNs?**



https://gradientscience.org/batchnorm/

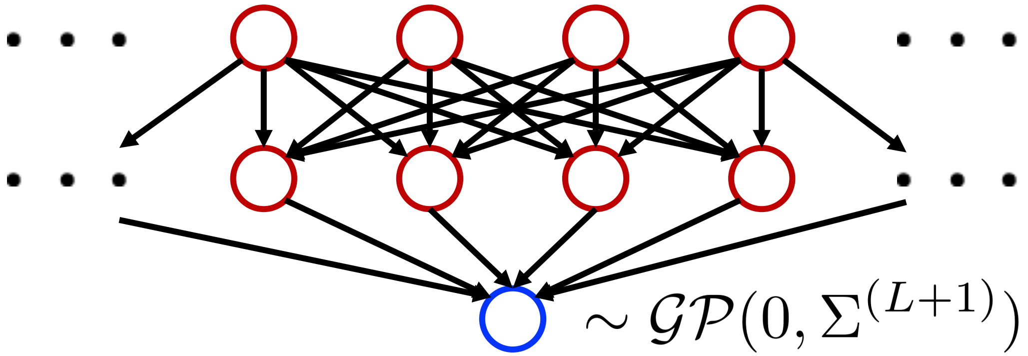
Stable Initialization from Dynamical Isometry [Pennington'17]

Good initialization can avoid gradient vanishing/exploding of deep nets

- **Motivation**: Wide neural networks as Gaussian process [Neal, 1996]
  - Consider an L-layer network: $\mathbf{W}^l \in \mathbb{R}^{N_{l-1} \times N_l}, \mathbf{h}^0 \in \mathbb{R}^{N_0}$

$$\mathbf{x}^l = \phi(\mathbf{h}^l), \quad \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l$$

  - Assume that $\mathbf{W}^l \sim \mathcal{N}(0, \sigma_w \mathbf{I}/N^{l-1}), \text{and } \mathbf{b}^l \sim \mathcal{N}(0, \sigma_b \mathbf{I})$



Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017

Stable Initialization from Dynamical Isometry [Pennington'17]

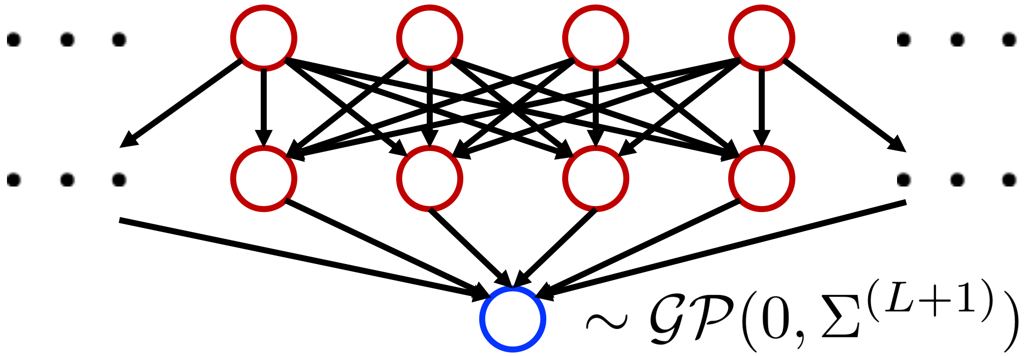
Good initialization can avoid gradient vanishing/exploding of deep nets

- **Motivation**: Wide neural networks as Gaussian process [Neal, 1996]
  - Consider an L-layer network: $\mathbf{W}^l \in \mathbb{R}^{N_{l-1} \times N_l}, \mathbf{h}^0 \in \mathbb{R}^{N_0}$

$$\mathbf{x}^l = \phi(\mathbf{h}^l), \quad \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l$$

  - Assume that $\mathbf{W}^l \sim \mathcal{N}(0, \sigma_w \mathbf{I}/N^{l-1})$, and $\mathbf{b}^l \sim \mathcal{N}(0, \sigma_b \mathbf{I})$
- **Question**: How to "optimally" set σ_w and σ_b? → **The law of large numbers**
  - **Idea**: As $N^{l-1} \to \infty$, $\mathbf{h}^l \to \mathcal{N}\left(0, q^l \mathbf{I}\right)$ where

$$q^l = \sigma_w^2 \cdot \mathbb{E}_{h \sim N(0,1)}[\phi(\sqrt{q^{l-1}}h)^2] + \sigma_b^2$$



Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017

Stable Initialization from Dynamical Isometry [Pennington'17]

Good initialization can avoid gradient vanishing/exploding of deep nets

- **Question**: How to "optimally" set σ_w and σ_b? → **The law of large numbers**
  - **Idea**: As $N^{l-1} \to \infty$, $\mathbf{h}^l \to \mathcal{N}\left(0, q^l \mathbf{I}\right)$ where

$$q^l = \sigma_w^2 \cdot \mathbb{E}_{h \sim N(0,1)}[\phi(\sqrt{q^{l-1}}h)^2] + \sigma_b^2$$

- Given **two conditions**, one can compute a set of optimal (σ_w^*, σ_b^*):

1. **Fixed point $\boldsymbol{q}^* = q^0 = \cdots = q^L$**
   - In many scenarios, $q^l \to q^*$ rapidly in few l's
   - σ_w and σ_b determines q^*: $q^* = \sigma_w^2 \cdot \mathbb{E}_{h \sim N(0,1)}[\phi(\sqrt{q^*}h)^2] + \sigma_b^2$

Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017

Stable Initialization from Dynamical Isometry [Pennington'17]

Good initialization can avoid gradient vanishing/exploding of deep nets

- **Question**: How to "optimally" set σ_w and σ_b? → **The law of large numbers**
  - **Idea**: As $N^{l-1} \to \infty$, $\mathbf{h}^l \to \mathcal{N}\left(0, q^l \mathbf{I}\right)$ where

$$q^l = \sigma_w^2 \cdot \mathbb{E}_{h \sim N(0,1)}[\phi(\sqrt{q^{l-1}}h)^2] + \sigma_b^2$$

- Given **two conditions**, one can compute a set of optimal (σ_w^*, σ_b^*):

1. **Fixed point $q^* = q^0 = \cdots = q^L$**
   - In many scenarios, $q^l \to q^*$ rapidly in few l's
   - σ_w and σ_b determines q^*: $q^* = \sigma_w^2 \cdot \mathbb{E}_{h \sim N(0,1)}[\phi(\sqrt{q^*}h)^2] + \sigma_b^2$
2. **Criticality $\chi^l = 1$** for all $l = 1, \cdots, L$
   - The mean of the squared singular values of $\mathbf{D}^l \mathbf{W}^l$

$$\boxed{D_{ij}^l = \phi'(h_i^l)\,\delta_{ij}}$$

$$\chi^l := \frac{1}{N^{l-1}} \mathbb{E}_w[\mathtt{tr}((\mathbf{D}^l\mathbf{W}^l)^T \mathbf{D}^l \mathbf{W}^l)] = \sigma_w^2 \cdot \mathbb{E}_h[\phi'(\sqrt{q^*}h)^2]$$

   - $\chi^l = 1$ makes that the input-output Jacobian to be stable:

$$\mathbf{J} = \frac{\partial \mathbf{x}^L}{\partial \mathbf{h}^0} = \prod_{l=1}^{L} \mathbf{D}^l \mathbf{W}^l.$$

Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017

Stable Initialization from Dynamical Isometry [Pennington'17]

Good initialization can avoid gradient vanishing/exploding of deep nets

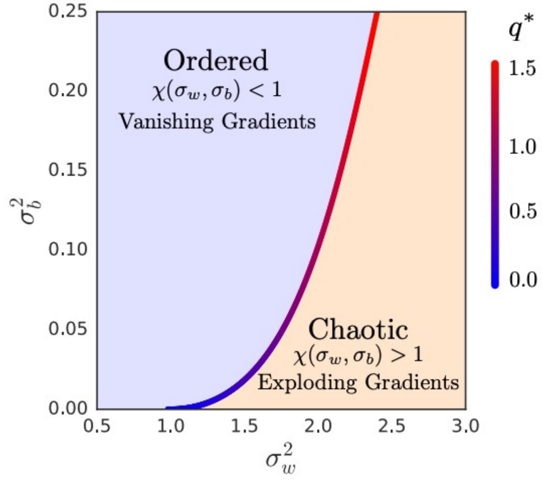
- **Question**: How to "optimally" set σ_w and σ_b? → **The law of large numbers**
- Given **two conditions**, one can compute a set of optimal (σ_w^*, σ_b^*):
  1. **Fixed point** $\boldsymbol{q^*} = q^0 = \cdots = q^L$

     $$q^* = \sigma_w^2 \cdot \mathbb{E}_{h\sim N(0,1)}[\phi(\sqrt{q^*}h)^2] + \sigma_b^2$$
  2. **Criticality** $\boldsymbol{\chi^l = 1}$ for all $l = 1, \cdots, L$

     $$\chi^l := \frac{1}{N^{l-1}}\mathbb{E}_w[\mathtt{tr}((\mathbf{D}^l\mathbf{W}^l)^T\mathbf{D}^l\mathbf{W}^l)] = \sigma_w^2 \cdot \mathbb{E}_h[\phi'(\sqrt{q^*}h)^2]$$

- **1 + 2** numerically determines $\chi(\sigma_w^*, \sigma_b^*) = 1$:
  - $\phi = \text{ReLU}$: $(\sigma_w^*, \sigma_b^*) = (2, 0)$
  - $\phi = \text{Tanh}$: See the right Figure

$\phi(h) = \text{Tanh}(h)$



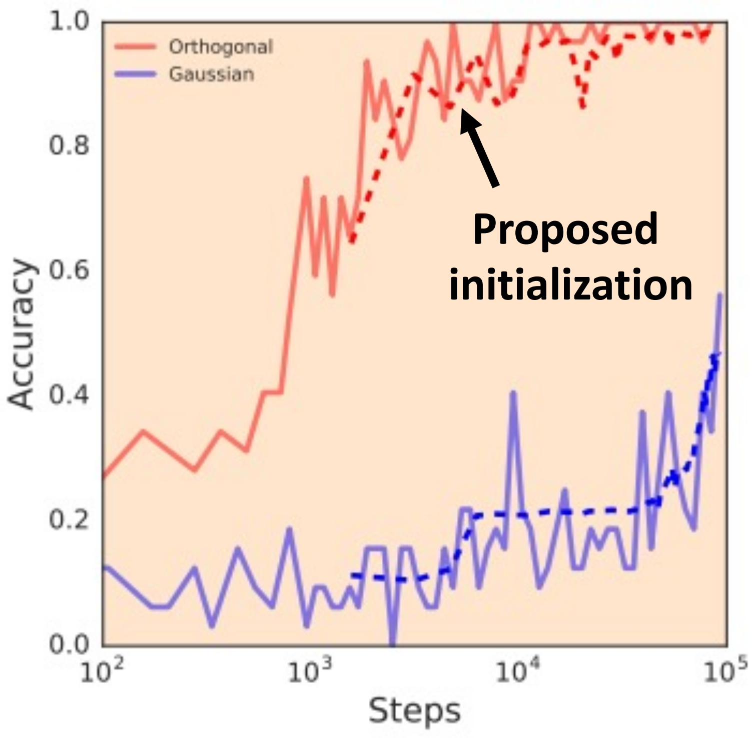
- With a deeper analysis of the spectrum of $\mathbf{J}$, one can further stabilize the training
  - For the details, see [Pennington et al., 2017]

Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017

Stable Initialization from Dynamical Isometry [Pennington'17]

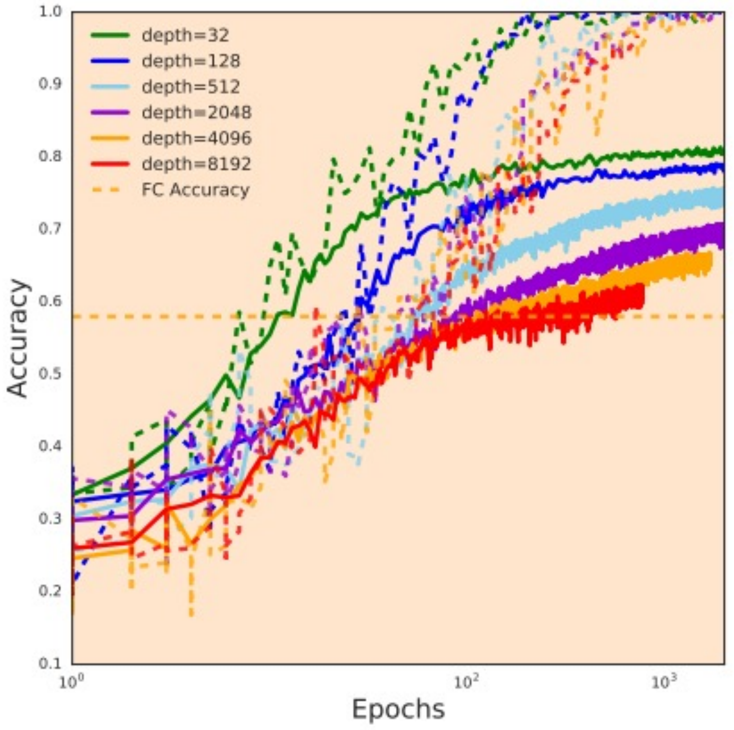
Good initialization can avoid gradient vanishing/exploding of deep nets

- The idea was later generalized to enable **10,000-layer ConvNets without BNs nor residual connections** [Xiao et al., 2019]



Proposed initialization

4000-layer ConvNet, CIFAR-10



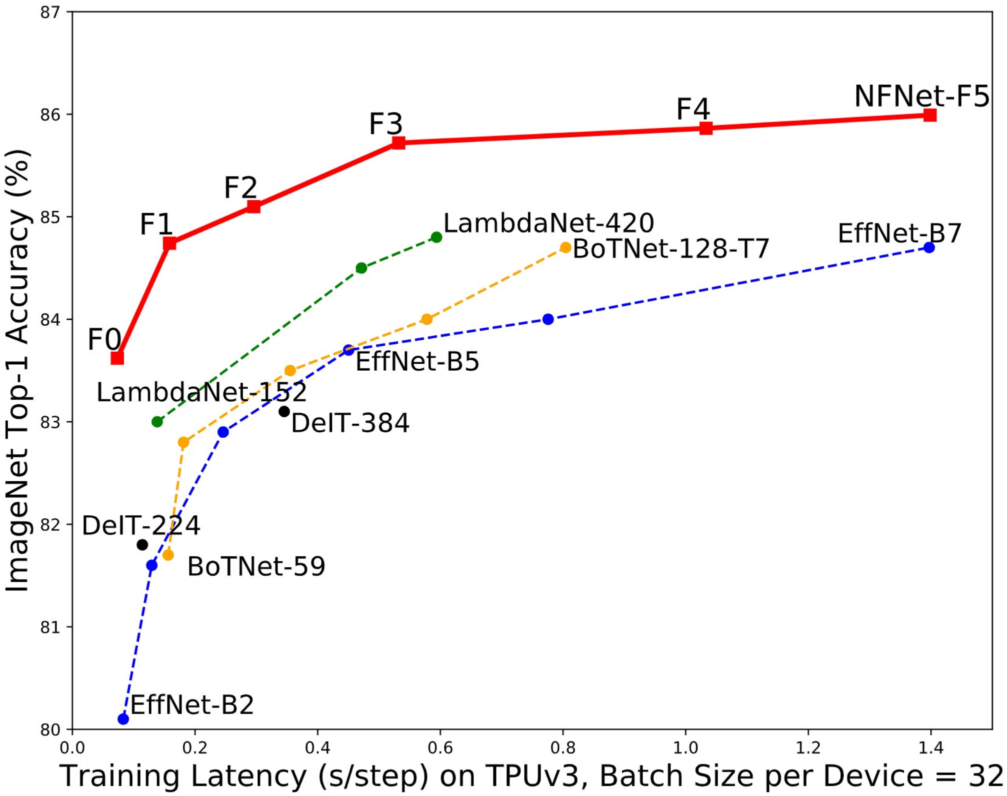
CIFAR-10
(solid: test, dashed: training)

Pennington et al., "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice", NeurIPS 2017
Xiao et al., Dynamical Isometry and a Mean Field Theory of CNNs, ICML 2019

NFNet: High-performance Normalizer-Free ResNets [Brock'21b]

Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
  - Removes BNs from ResNet while maintaining their strengths
  - Achieves strong results on ImageNet benchmark



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

NFNet: High-performance Normalizer-Free ResNets [Brock'21b]

Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
  - Removes BNs from ResNet while maintaining their strengths

1. Scaled weight standardization [Qiao'19]

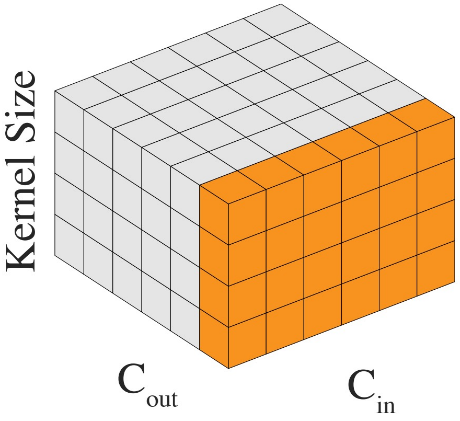
- To eliminate mean-shifts as BN does
- Re-parameterize the weights of each convolutional layer by:

$$\hat{W_{ij}} = \frac{W_{ij} - \mu_i}{\sqrt{N}\sigma_i}, \tag{1}$$

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

Weight Standardization



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

NFNet: High-performance Normalizer-Free ResNets [Brock’21b]

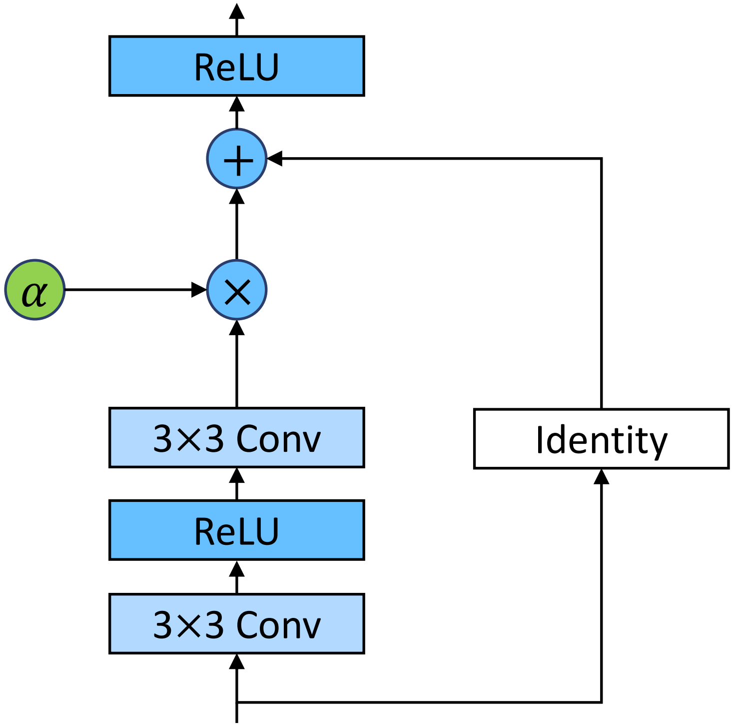
Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
  - Removes BNs from ResNet while maintaining their strengths

2. Downscaled residual branches

- A small scalar α to suppress the scale of activations from residual branches

α is set to a small value (= 0.2)



Brock et al., “High-Performance Large-Scale Image Recognition Without Normalization”, arXiv 2021

NFNet: High-performance Normalizer-Free ResNets [Brock'21b]

Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
  - Removes BNs from ResNet while maintaining their strengths

3. Adaptive gradient clipping

  - Allows efficient large-batch training
  - Robust to the clipping threshold hyperparameter λ in practice

Measures how much a single gradient update will change the weights

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

If the 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

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

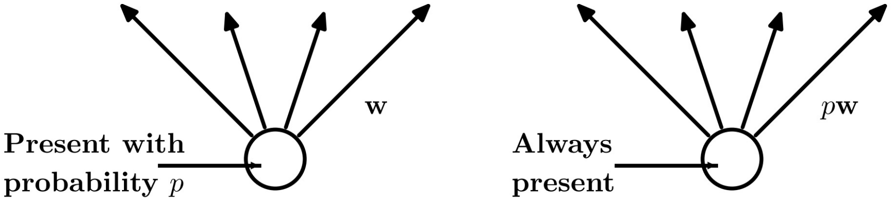
NFNet: High-performance Normalizer-Free ResNets [Brock’21b]

Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
  - Removes BNs from ResNet while maintaining their strengths

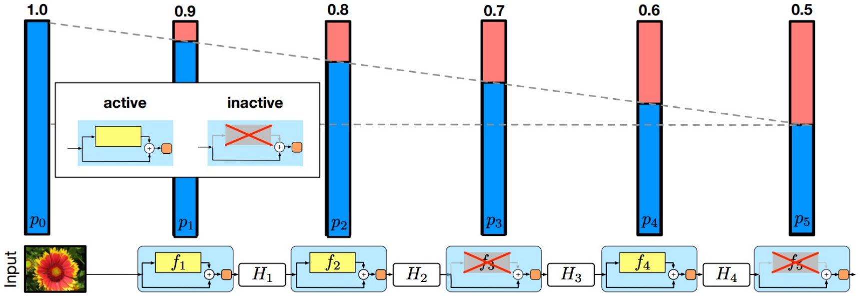
4. Additional regularizations

- Dropout [Srivastava’14] and Stochastic depth [Huang’16] during training



(a) At training time (b) At test time

Dropout
[Srivastava’14]



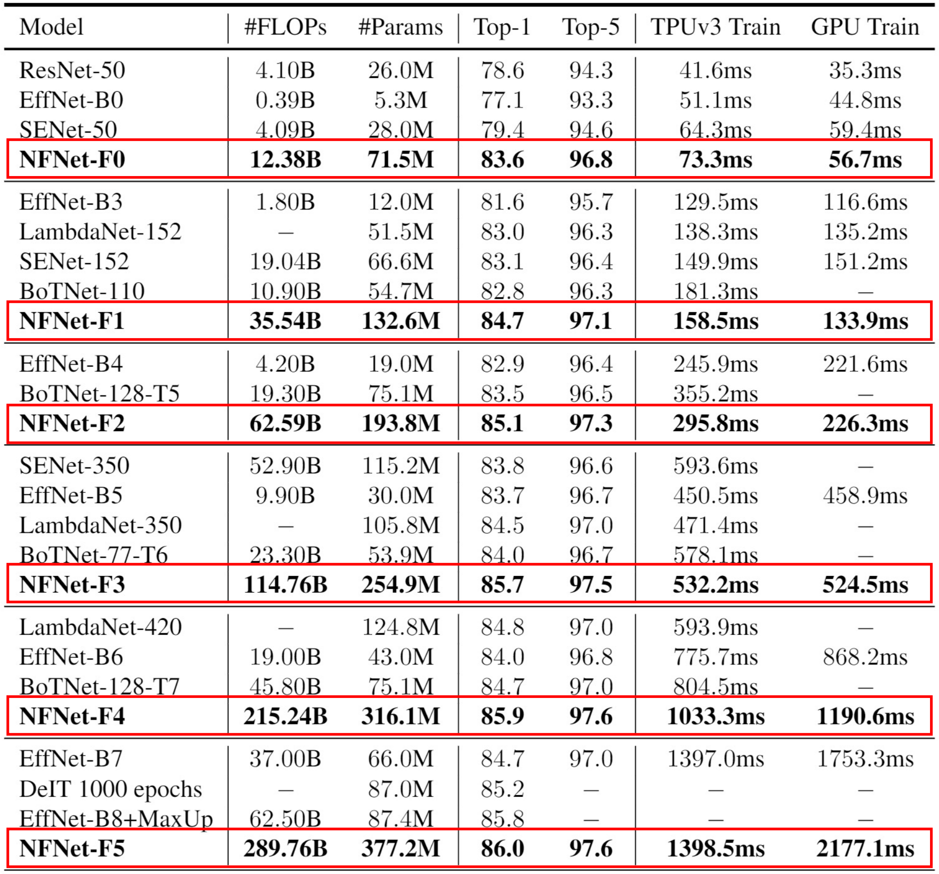
Stochastic Depth
[Huang’16]

Brock et al., “High-Performance Large-Scale Image Recognition Without Normalization”, arXiv 2021

NFNet: High-performance Normalizer-Free ResNets [Brock'21b]

Better architecture designs (+ training) can overcome the optimization difficulties

- Brock et al. (2021b): **N**ormalizer-**F**ree Res**Net** (**NFNet**)
- Experimental results on ImageNet classification
  - Achieves better accuracies compared to state-of-the-art BN-based architectures



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**

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

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

AdamW: SGD/Adam with Decoupled Weight Decay [Loshchilov'19]

Regularizing loss with ℓ_2**-norm penalty** is one of the most common practices

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

- For SGD, It is equivalent to "weight decay" since its gradient decays weight:

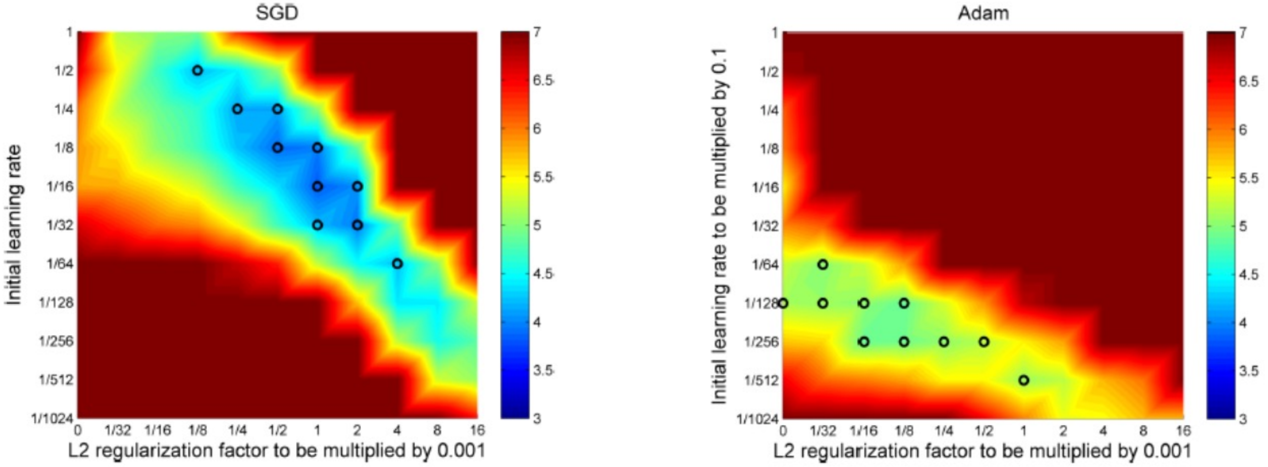
$$\boldsymbol{\theta} - \eta\nabla\left(L(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_2^2\right) \quad \Leftrightarrow \quad (1-2\eta\lambda)\boldsymbol{\theta} - \eta\nabla L(\boldsymbol{\theta})$$

SGD on ℓ_2-norm penalty

$\nabla\|\boldsymbol{\theta}\|_2^2 = 2\boldsymbol{\theta}$

weight decay

- But, this equivalence **does not hold** for momentum/adaptive methods! (check)
  - This gap supports why Adam < SGD for some tasks, e.g., image classification



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

AdamW: SGD/Adam with Decoupled Weight Decay [Loshchilov'19]

Loshchilov et al. (2019): **Decoupled** weight decay from optimizers

- For general momentum-based optimizers, ℓ_2**-regularization** $\neq$ **weight-decay**

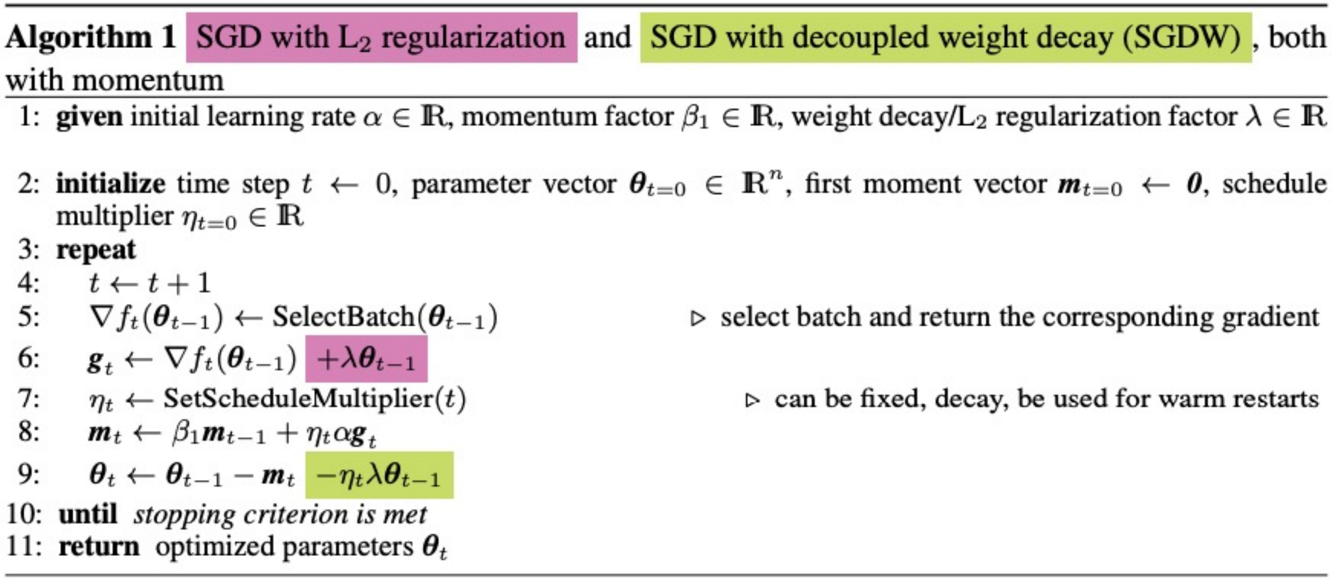
L2 penalty:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha M_t(\nabla f_t(\theta_t) + \lambda'\theta) \\ &= \boxed{\theta_t - \alpha\lambda' M_t\theta} - \alpha M_t \nabla f_t(\theta_t)\end{aligned}$$

weight decay:

$$\theta_{t+1} = \boxed{(1-\lambda)\theta_t} - \alpha M_t \nabla f_t(\theta)$$

- SGDW and AdamW aim to adjust this gap by explicitly adding the WD-term
  - **Example**: Decoupled SGD with momentum (also applicable to Adam)



Algorithm 1 SGD with L$_2$ regularization and SGD with decoupled weight decay (SGDW), both with momentum

```
given initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay/L$_2$ regularization factor $\lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    $t \leftarrow t + 1$
    $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$    ▷ select batch and return the corresponding gradient
    $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
    $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$    ▷ can be fixed, decay, be used for warm restarts
    $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t$ $-\eta_t\lambda\boldsymbol{\theta}_{t-1}$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_t$
```

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

AdamW: SGD/Adam with Decoupled Weight Decay [Loshchilov'19]

Loshchilov et al. (2019): **Decoupled** weight decay from optimizers

- For general momentum-based optimizers, ℓ_2**-regularization** $\neq$ **weight-decay**

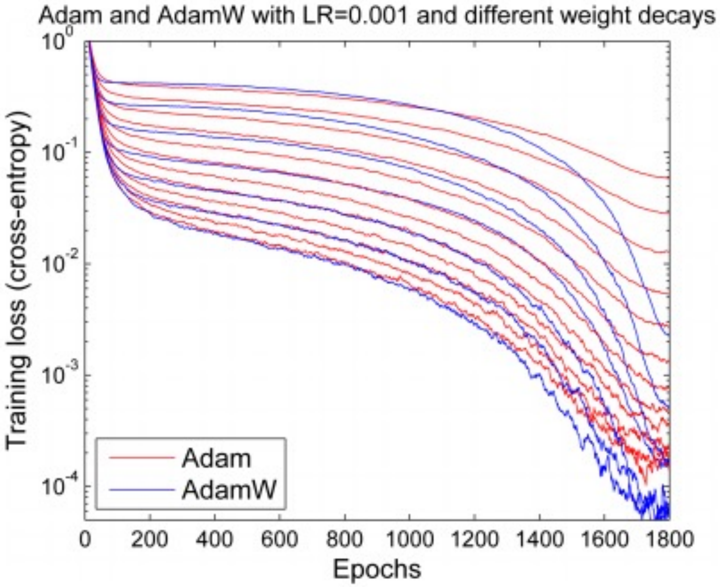
L2 penalty:

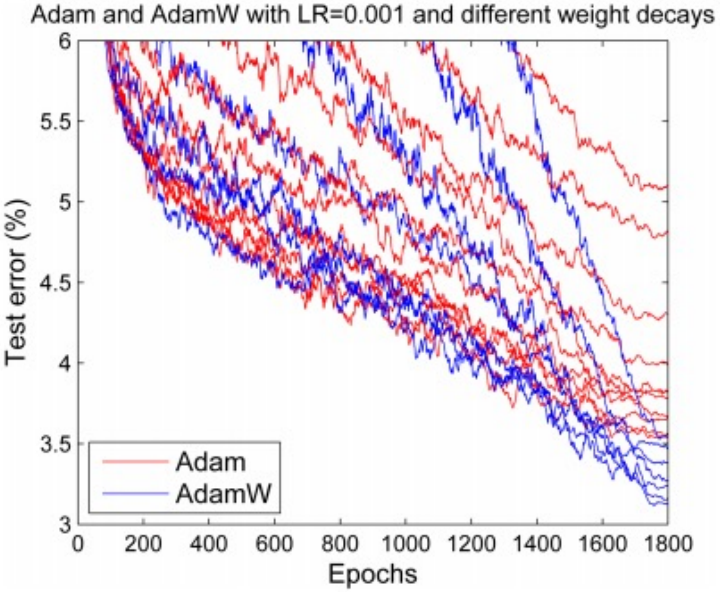
$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha M_t(\nabla f_t(\theta_t) + \lambda'\theta) \\ &= \theta_t - \alpha\lambda' M_t\theta - \alpha M_t \nabla f_t(\theta_t) \end{aligned}$$

weight decay:

$$\theta_{t+1} = (1-\lambda)\theta_t - \alpha M_t \nabla f_t(\theta)$$

- The proposed **AdamW** consistently outperforms Adam under ℓ_2-regularization





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

AdamW: SGD/Adam with Decoupled Weight Decay [Loshchilov'19]

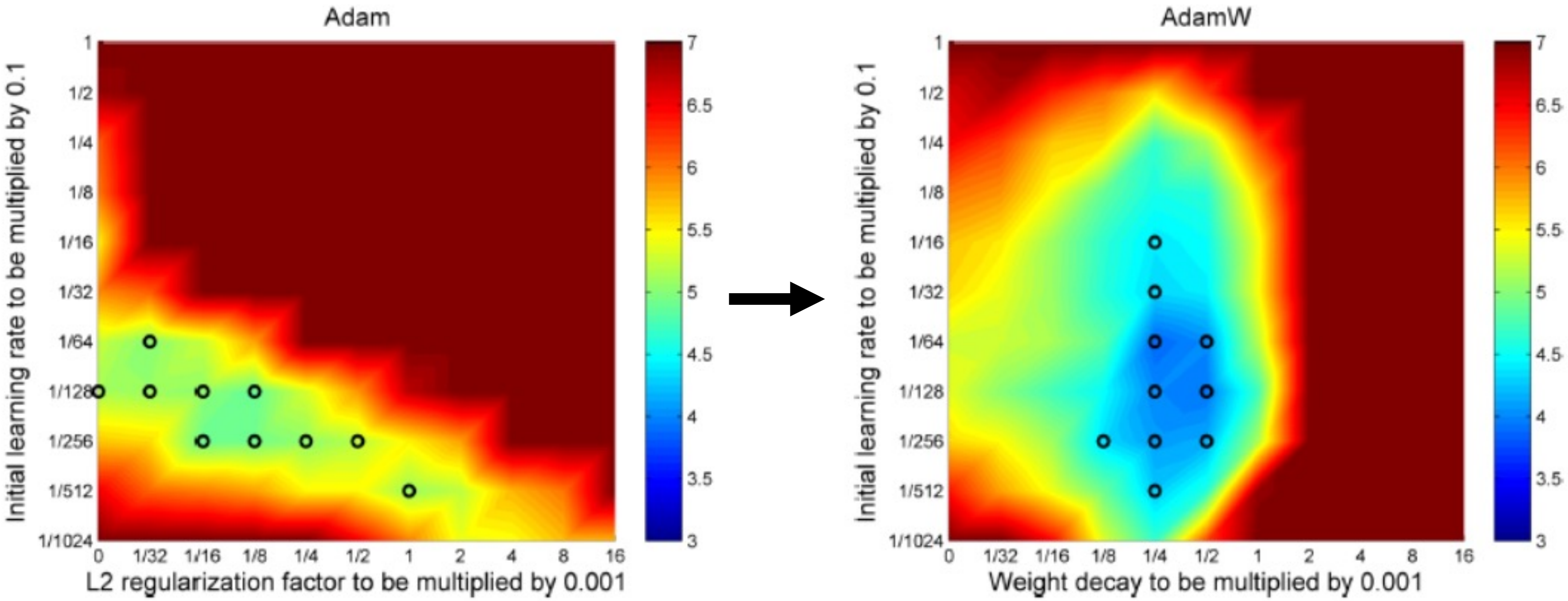
Loshchilov et al. (2019): **Decoupled** weight decay from optimizers

- For general momentum-based optimizers, ℓ_2**-regularization** $\neq$ **weight-decay**

L2 penalty:
$$\theta_{t+1} = \theta_t - \alpha M_t(\nabla f_t(\theta_t) + \lambda'\theta)$$
$$= \theta_t - \alpha\lambda' M_t\theta - \alpha M_t \nabla f_t(\theta_t)$$

weight decay:
$$\theta_{t+1} = (1-\lambda)\theta_t - \alpha M_t \nabla f_t(\theta)$$

- The proposed **AdamW** consistently outperforms Adam under ℓ_2-regularization



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

AdamW: SGD/Adam with Decoupled Weight Decay [Loshchilov'19]

Loshchilov et al. (2019): **Decoupled** weight decay from optimizers

- For general momentum-based optimizers, ℓ_2**-regularization** $\neq$ **weight-decay**

L2 penalty:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha M_t(\nabla f_t(\theta_t) + \lambda'\theta) \\ &= \theta_t - \alpha\lambda' M_t\theta - \alpha M_t \nabla f_t(\theta_t)\end{aligned}$$

weight decay:

$$\theta_{t+1} = (1-\lambda)\theta_t - \alpha M_t \nabla f_t(\theta)$$

- Currently, **AdamW** is adopted for a wide range of state-of-the-art models, especially for **Transformer-based models** those employ high weight decays
  - Data-efficient Image Transformer (DeiT) [Touvron'21]
  - Swin Transformer [Liu'21]
  - Masked Auto-encoder (MAE) [He'21]
  - ConvNeXt [Liu'22]
  - AlphaCode [Li'22]

Loshchilov et al., "Decoupled Weight Decay Regularization.", ICLR 2019
Touvron et al., "Training data-efficient image transformers & distillation through attention", 2021
Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", ICCV 2021
He et al., "Masked Autoencoders Are Scalable Vision Learners", 2021
Liu et al., "A ConvNet for the 2020s", 2022
Li et al., "Competition-Level Code Generation with AlphaCode", 2022

AdamP: Adam with Projection on Scale-invariant Weights [Heo'21]

Heo et al. (2021): "**Normalization** + momentum optimizers" can be problematic

- Normalization layers (e.g., BN) induces **scale-invariance weights**

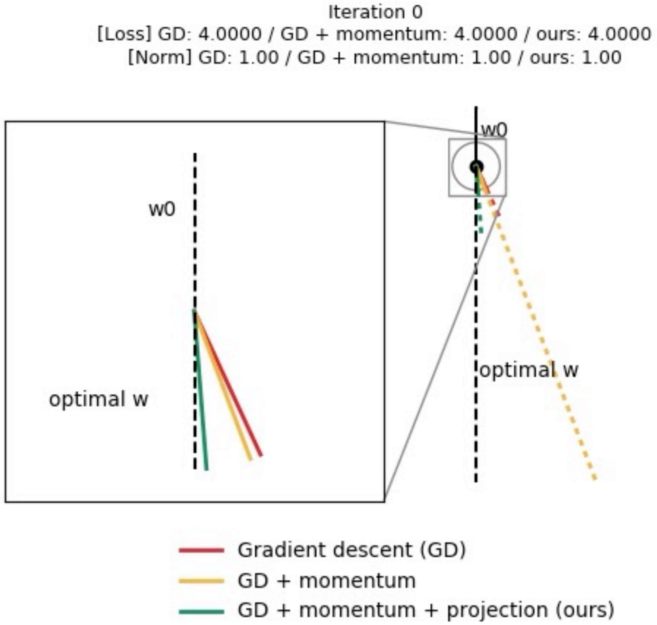
$$\mathrm{Norm}(\boldsymbol{w}^\top \boldsymbol{x}) = \mathrm{Norm}(c\boldsymbol{w}^\top \boldsymbol{x}) \quad \forall c > 0.$$

- **Problem**: Momentum induces an excessive growth of weight norms
  - **Increased weight norms** → Decreased (relative) step size

- **Example 1**: $w_0 \to w$ in $\mathbb{R}^2$

$$-\min_w \cos(w, w_0) = \max_w \frac{w \cdot w_0}{\|w\|_2 \, \|w_0\|_2}$$

  - GD + momentum "unnecessarily" increases $||w||_2$ during the optimization



Heo et al., "AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights", ICLR 2021

AdamP: Adam with Projection on Scale-invariant Weights [Heo'21]

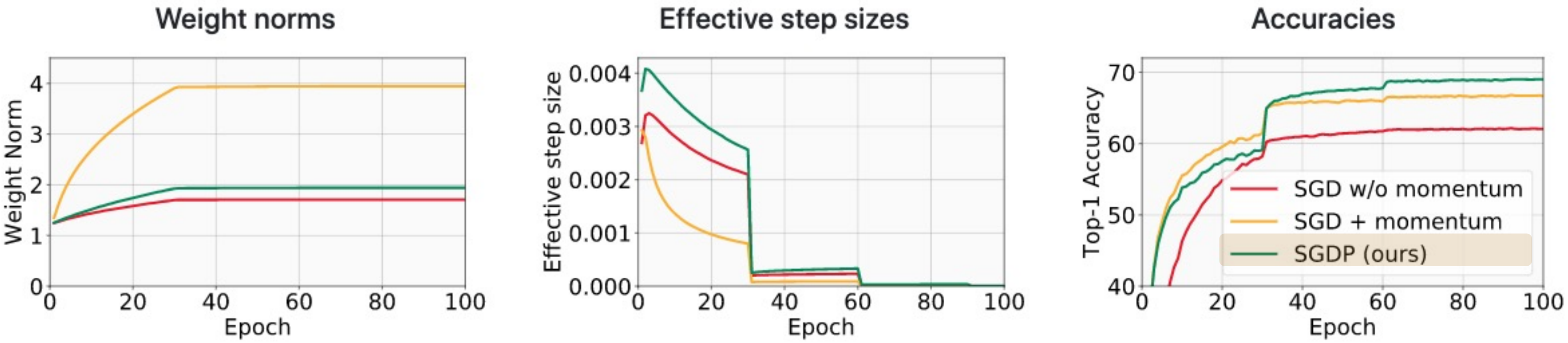
Heo et al. (2021): "**Normalization** + momentum optimizers" can be problematic

- Normalization layers (e.g., BN) induces **scale-invariance weights**

$$\mathrm{Norm}(\boldsymbol{w}^\top \boldsymbol{x}) = \mathrm{Norm}(c\boldsymbol{w}^\top \boldsymbol{x}) \quad \forall c > 0.$$

- **Problem**: Momentum induces an excessive growth of weight norms
  - **Increased weight norms** → Decreased (relative) step size

- **Example 2**: ImageNet training via SGD
  - The trend can be still observed in training deep neural networks



Heo et al., "AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights", ICLR 2021

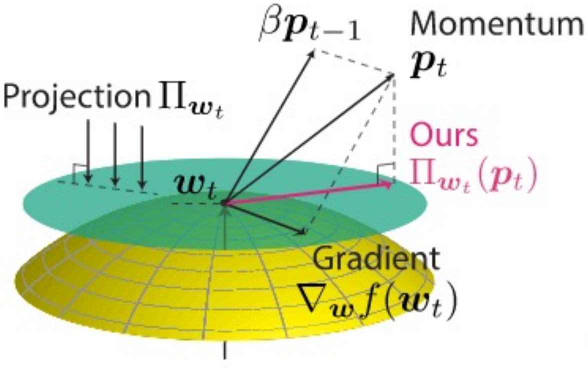
AdamP: Adam with Projection on Scale-invariant Weights [Heo'21]

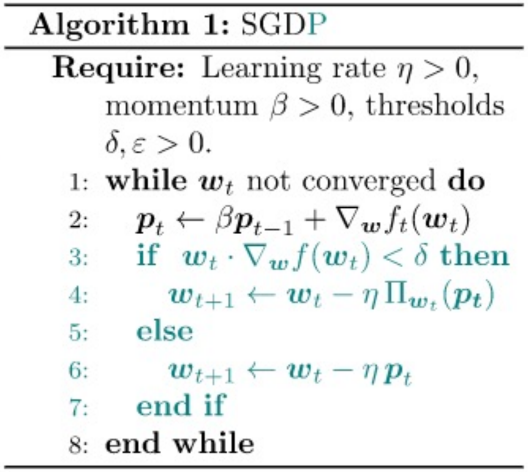
Heo et al. (2021): "**Normalization** + momentum optimizers" can be problematic

- **Adam** → **AdamP**: Adam with weight projection

$$\Pi_{\mathbf{w}}(\mathbf{x}) := \mathbf{x} - (\hat{\mathbf{w}} \cdot \mathbf{x})\hat{\mathbf{w}}, \text{ where } \hat{\mathbf{w}} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$$

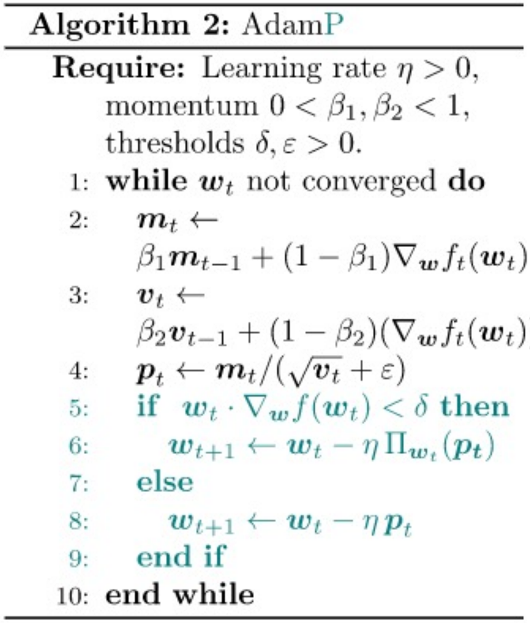
  - Apply $\boldsymbol{p}_t \longleftarrow \Pi_{\mathbf{w}}(\boldsymbol{p}_t)$ whenever $\mathbf{w}_t \cdot \nabla_{\mathbf{w}} f(\mathbf{w}_t) < \delta$ (close to orthogonal)
  - In practice, $\delta = 0.1$ widely works fine





Algorithm 1: SGDP

```
Require: Learning rate $\eta > 0$,
    momentum $\beta > 0$, thresholds
    $\delta, \varepsilon > 0$.
while $\boldsymbol{w}_t$ not converged do
    $\boldsymbol{p}_t \leftarrow \beta \boldsymbol{p}_{t-1} + \nabla_{\boldsymbol{w}} f_t(\boldsymbol{w}_t)$
    if $\boldsymbol{w}_t \cdot \nabla_{\boldsymbol{w}} f(\boldsymbol{w}_t) < \delta$ then
        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \Pi_{\boldsymbol{w}_t}(\boldsymbol{p}_t)$
    else
        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \boldsymbol{p}_t$
    end if
end while
```



Algorithm 2: AdamP

```
Require: Learning rate $\eta > 0$,
    momentum $0 < \beta_1, \beta_2 < 1$,
    thresholds $\delta, \varepsilon > 0$.
while $\boldsymbol{w}_t$ not converged do
    $\boldsymbol{m}_t \leftarrow$
    $\beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \nabla_{\boldsymbol{w}} f_t(\boldsymbol{w}_t)$
    $\boldsymbol{v}_t \leftarrow$
    $\beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)(\nabla_{\boldsymbol{w}} f_t(\boldsymbol{w}_t))^2$
    $\boldsymbol{p}_t \leftarrow \boldsymbol{m}_t / (\sqrt{\boldsymbol{v}_t} + \varepsilon)$
    if $\boldsymbol{w}_t \cdot \nabla_{\boldsymbol{w}} f(\boldsymbol{w}_t) < \delta$ then
        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \Pi_{\boldsymbol{w}_t}(\boldsymbol{p}_t)$
    else
        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \boldsymbol{p}_t$
    end if
end while
```

Heo et al., "AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights", ICLR 2021

AdamP: Adam with Projection on Scale-invariant Weights [Heo'21]

Heo et al. (2021): "**Normalization** + momentum optimizers" can be problematic

- **Adam** → **AdamP**: Adam with weight projection

$$\Pi_{\mathbf{w}}(\mathbf{x}) := \mathbf{x} - (\hat{\mathbf{w}} \cdot \mathbf{x})\hat{\mathbf{w}}, \text{ where } \hat{\mathbf{w}} := \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$$

- AdamP achieves better generalization across a wide range of tasks

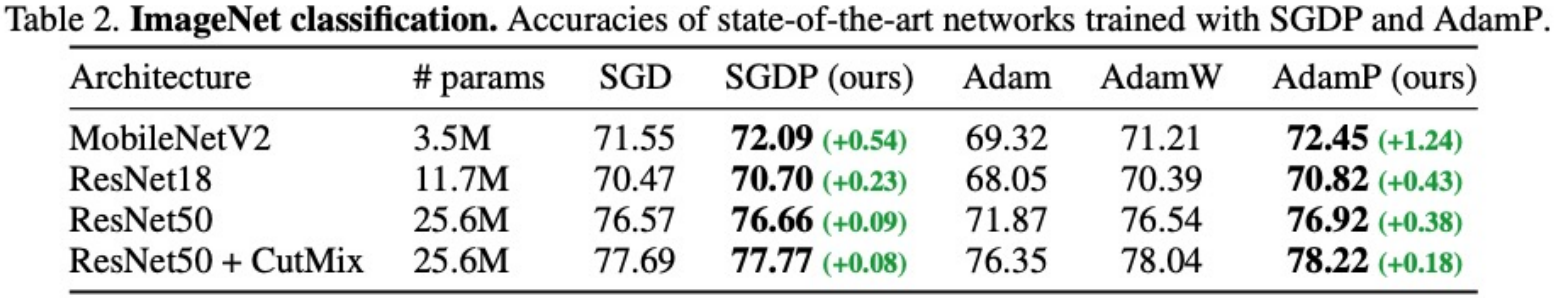


Table 2. **ImageNet classification.** Accuracies of state-of-the-art networks trained with SGDP and AdamP.

Architecture	# params	SGD	SGDP (ours)	Adam	AdamW	AdamP (ours)
MobileNetV2	3.5M	71.55	**72.09** (+0.54)	69.32	71.21	**72.45** (+1.24)
ResNet18	11.7M	70.47	**70.70** (+0.23)	68.05	70.39	**70.82** (+0.43)
ResNet50	25.6M	76.57	**76.66** (+0.09)	71.87	76.54	**76.92** (+0.38)
ResNet50 + CutMix	25.6M	77.69	**77.77** (+0.08)	76.35	78.04	**78.22** (+0.18)

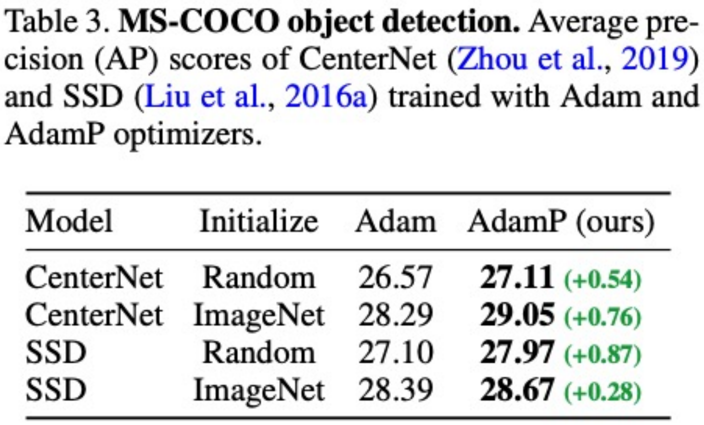


Table 3. **MS-COCO object detection.** Average precision (AP) scores of CenterNet (Zhou et al., 2019) and SSD (Liu et al., 2016a) trained with Adam and AdamP optimizers.

Model	Initialize	Adam	AdamP (ours)
CenterNet	Random	26.57	**27.11** (+0.54)
CenterNet	ImageNet	28.29	**29.05** (+0.76)
SSD	Random	27.10	**27.97** (+0.87)
SSD	ImageNet	28.39	**28.67** (+0.28)

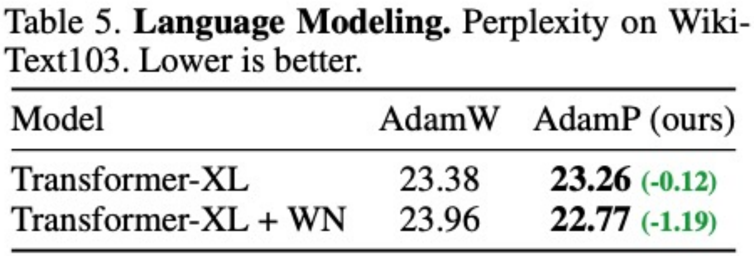


Table 5. **Language Modeling.** Perplexity on Wiki-Text103. Lower is better.

Model	AdamW	AdamP (ours)
Transformer-XL	23.38	**23.26** (-0.12)
Transformer-XL + WN	23.96	**22.77** (-1.19)

(More in the full paper, e.g.,
audio classification, robustness, …)

Heo et al., "AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights", ICLR 2021

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

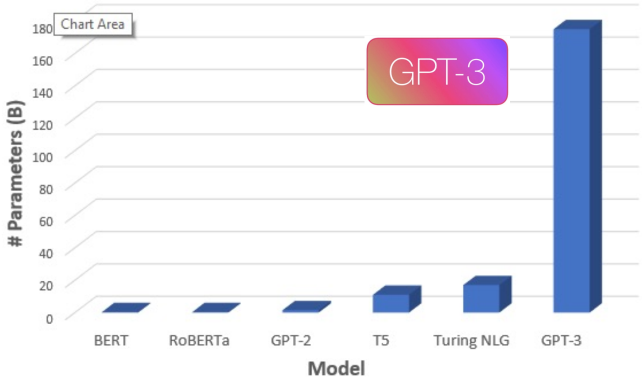
- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Large Batch Training: Motivation

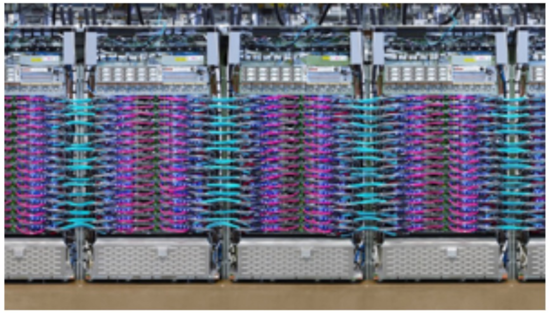
Deep learning is scaling up very quickly

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

Larger dataset



Larger model



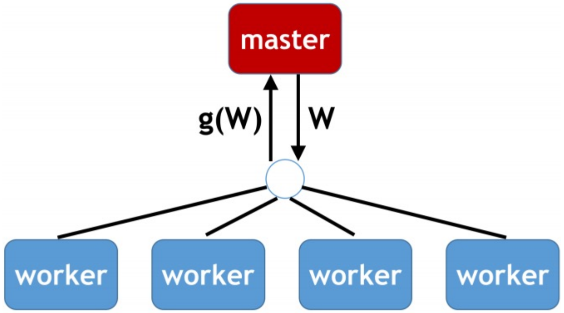
Cloud TPU v3 Pod
100+ petaflops
32 TB HBM
2-D toroidal mesh network

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

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

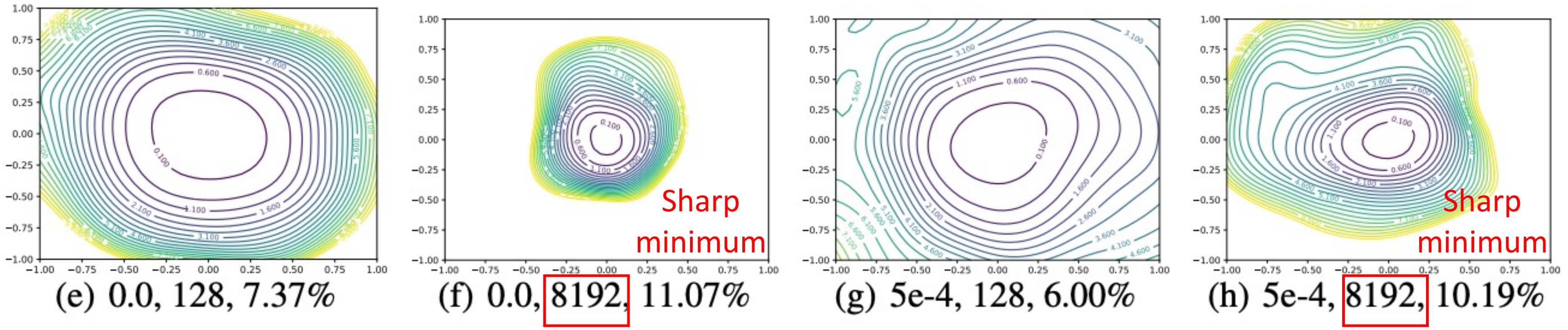
Large Batch Training: Challenge

Naïvely increasing batch size → performance degradation

- In particular, **generalization** performance suffers

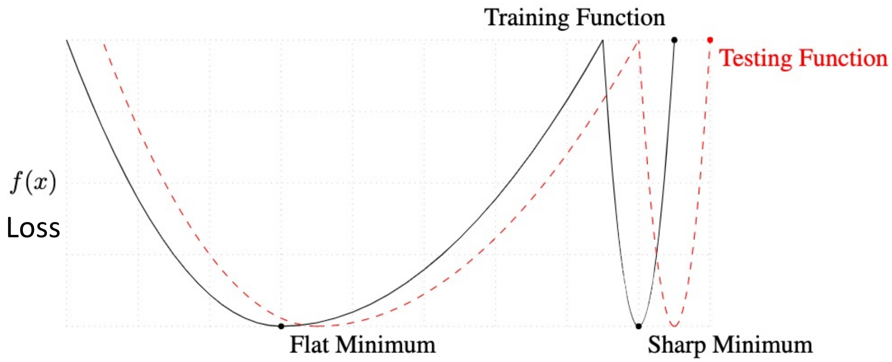
One popular explanation: **Sharp-minima problem** [Keskar'17]

- Large batch (LB) training finds a "sharp minimum"



(e) 0.0, 128, 7.37% (f) 0.0, 8192, 11.07% (g) 5e-4, 128, 6.00% (h) 5e-4, 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 $\theta^*_{\texttt{train}}$

→

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

Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017
Li et al., "Visualizing the Loss Landscape of Neural Nets", NeurIPS 2018

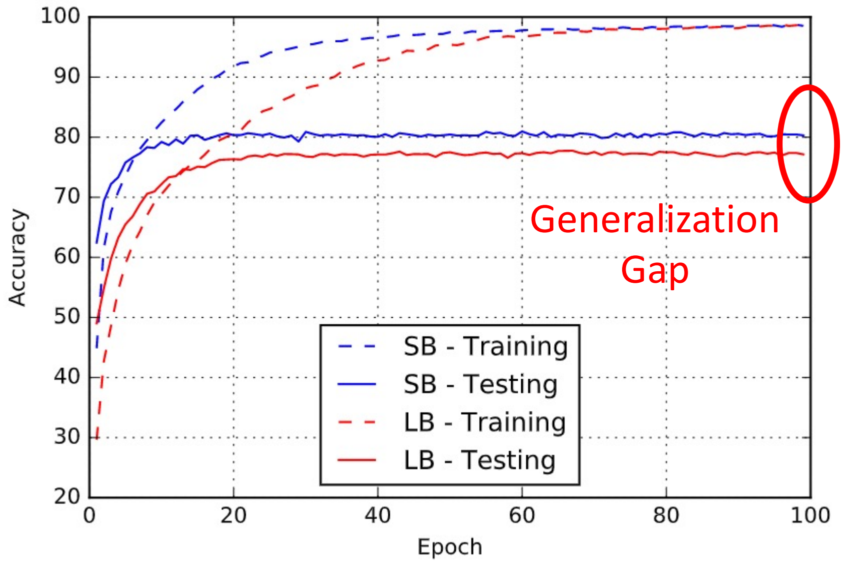
Large Batch Training: Challenge

Naïvely increasing batch size → performance degradation

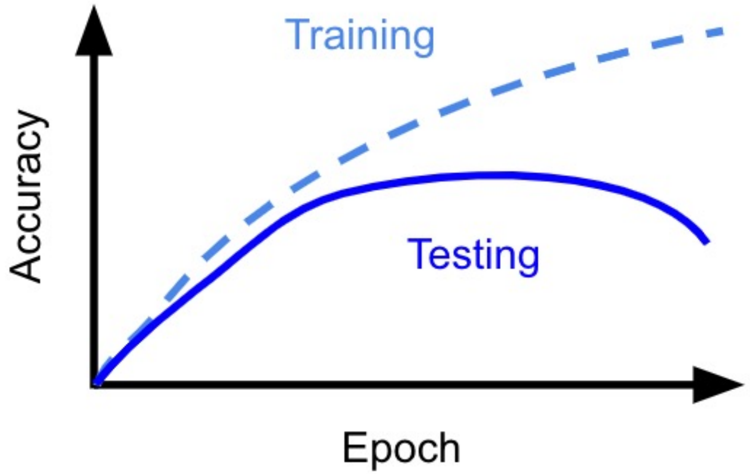
- In particular, **generalization** performance suffers

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

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

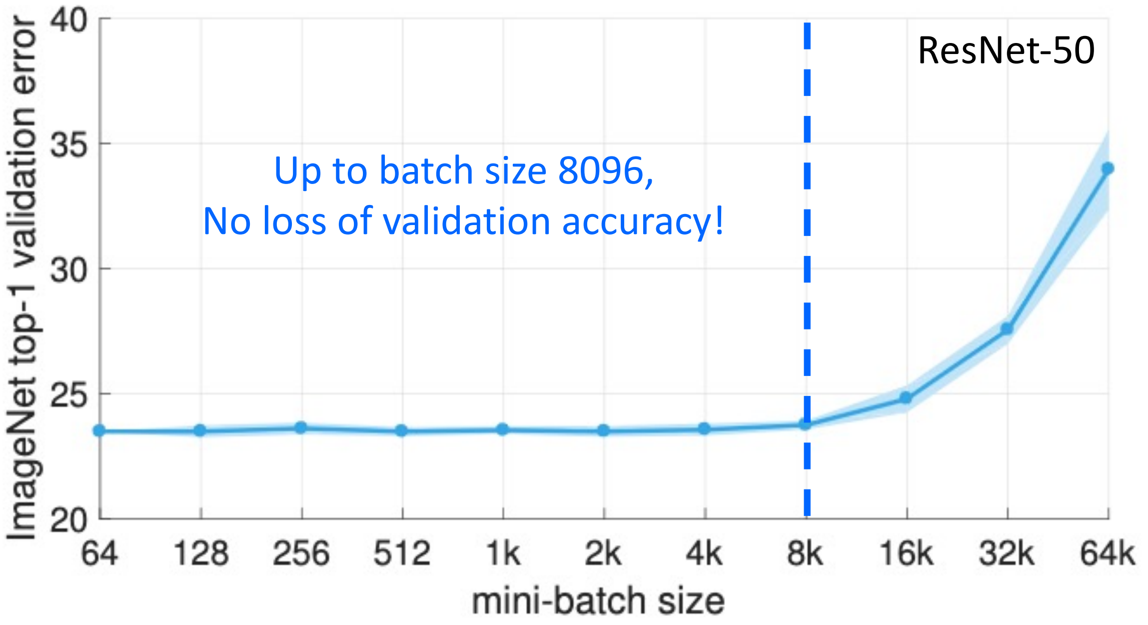


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.**

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]

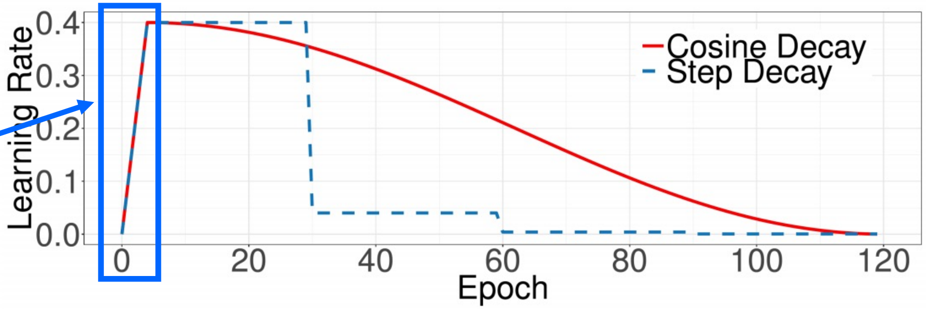
  1. **Linear scaling rule**
     - Given a fixed number of epochs, **increasing batch size** B by *k* times means *k* times **fewer training iterations**, for:

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

     - 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)

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

Large Batch Training: LARS & LAMB

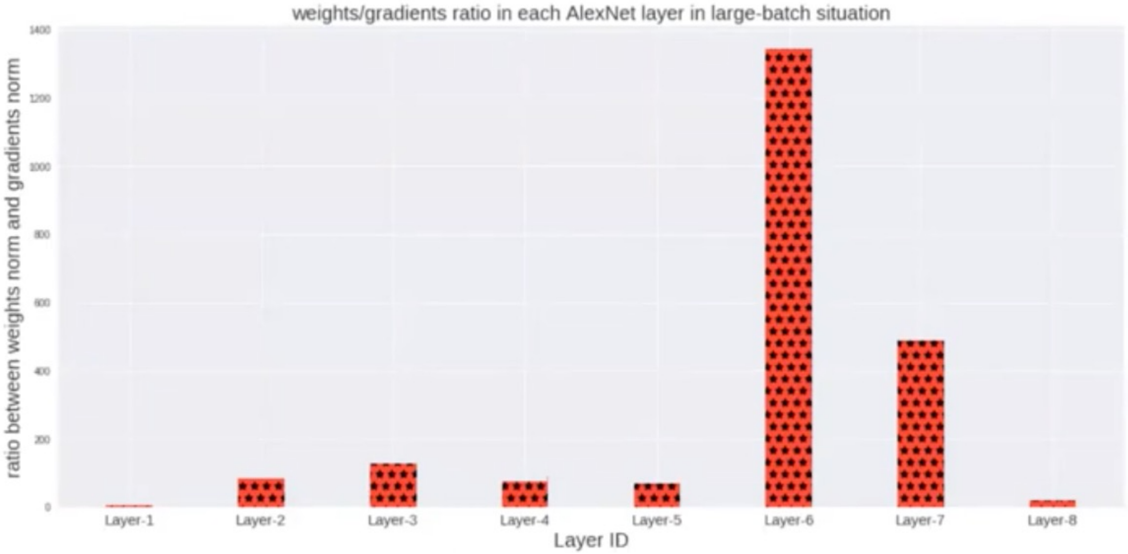
- **Layer-wise Adaptive Rate Scaling (LARS)** [You'17]
  - The ratio between weight and its gradient matters

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

Too large: slow learning

Too small: divergence

  - Note that standard SGD uses a fixed γ for all weights
  - **Observation**: For LB training, the ratios are apparently different **across layers**



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

Large Batch Training: LARS & LAMB

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

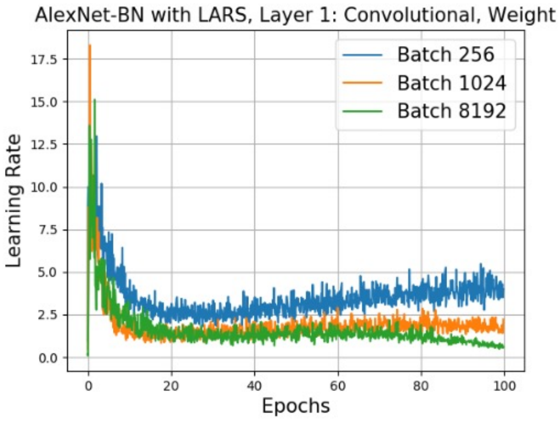
$$\boldsymbol{\theta}_{t+1}^{l} = \boldsymbol{\theta}_{t}^{l} - \gamma \cdot \lambda^{l} \cdot \nabla L(\boldsymbol{\theta}_{t}^{l})$$

γ global learning rate

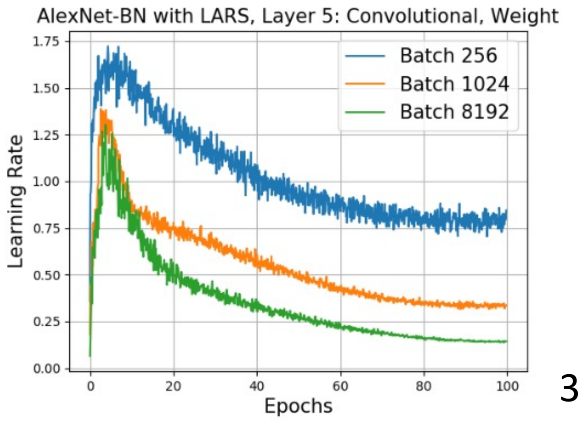
$$\lambda^{l} := \eta \frac{\|\boldsymbol{\theta}^{l}\|}{\|\nabla L(\boldsymbol{\theta}^{l})\|}$$

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)

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

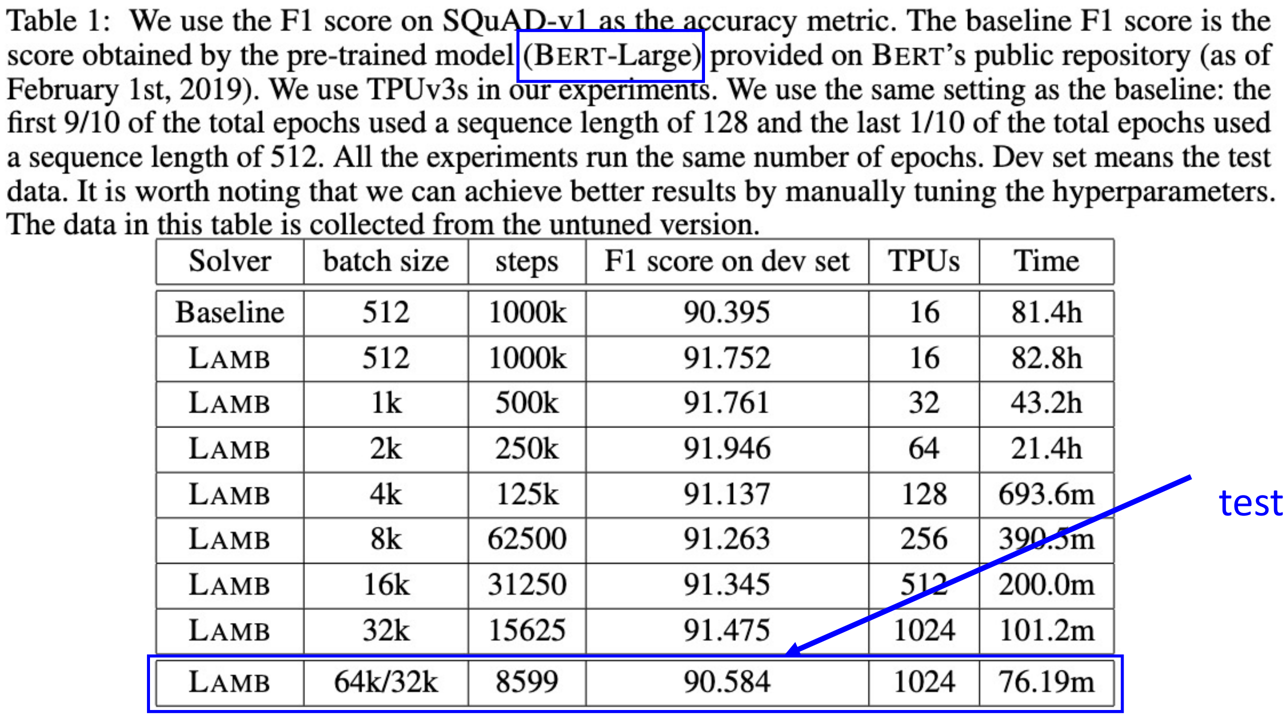


Table 1: We use the F1 score on SQuAD-v1 as the accuracy metric. The baseline F1 score is the score obtained by the pre-trained model (BERT-Large) provided on BERT's public repository (as of February 1st, 2019). We use TPUv3s in our experiments. We use the same setting as the baseline: the first 9/10 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.

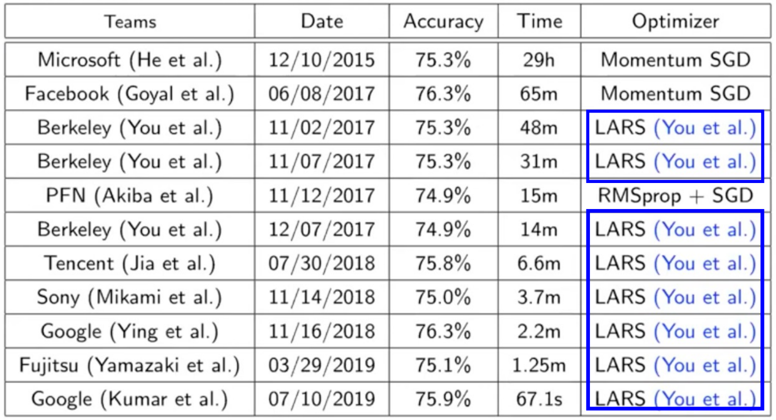
Solver	batch size	steps	F1 score on dev set	TPUs	Time
Baseline	512	1000k	90.395	16	81.4h
LAMB	512	1000k	91.752	16	82.8h
LAMB	1k	500k	91.761	32	43.2h
LAMB	2k	250k	91.946	64	21.4h
LAMB	4k	125k	91.137	128	693.6m
LAMB	8k	62500	91.263	256	390.5m
LAMB	16k	31250	91.345	512	200.0m
LAMB	32k	15625	91.475	1024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

No loss in
test performance

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

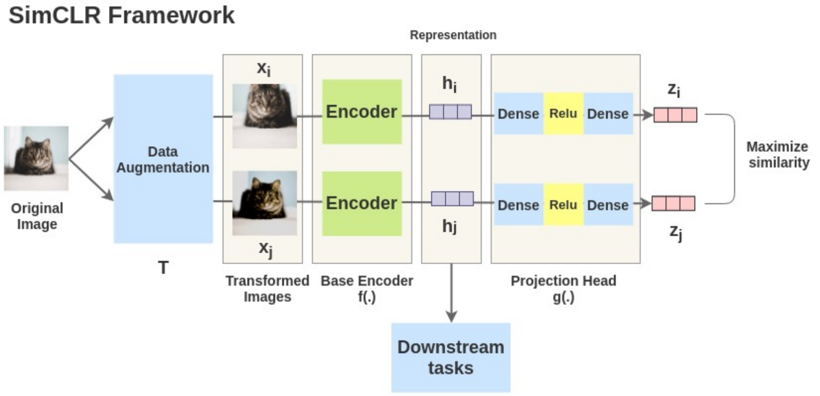
Large Batch Training: LARS & LAMB

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

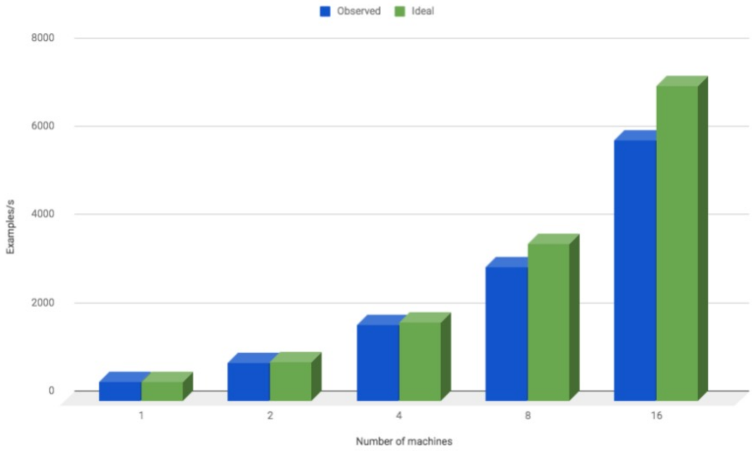


Teams	Date	Accuracy	Time	Optimizer
Microsoft (He et al.)	12/10/2015	75.3%	29h	Momentum SGD
Facebook (Goyal et al.)	06/08/2017	76.3%	65m	Momentum SGD
Berkeley (You et al.)	11/02/2017	75.3%	48m	LARS (You et al.)
Berkeley (You et al.)	11/07/2017	75.3%	31m	LARS (You et al.)
PFN (Akiba et al.)	11/12/2017	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.)

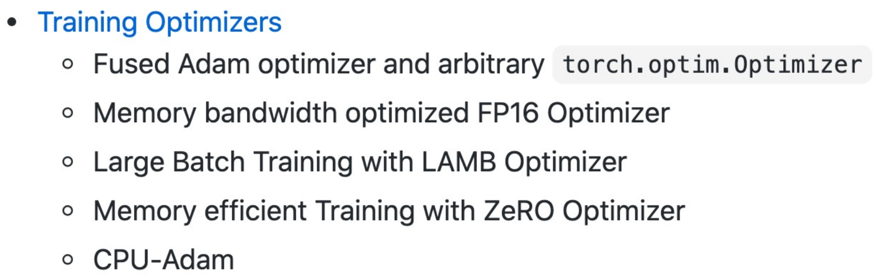
ImageNet/ResNet-50 Training Speed Records



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

But actually…

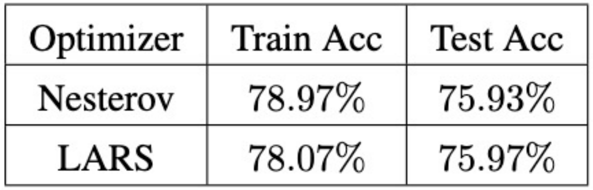
https://www.youtube.com/watch?v=kwEBP-Wbtdc

https://amitness.com/2020/03/illustrated-simclr

Large Batch Training: Sanity Check

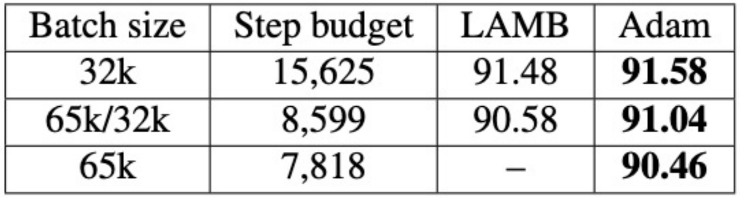
A recent paper [Nado'21] questions the effectiveness of LARS & LAMB

- Good performances are more due to **subtle implementation details**
  - For **ResNet-50**,
    - Unconventional BatchNorm hyperparameters
    - No L2-regularization on bias parameters nor on BN parameters
    - Nesterov works just as well with similar modifications
  - For **BERT**,
    - Fixing bugs in Adam and LR schedule in BERT's code → 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 useful is an open question

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

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Beyond Empirical Risk Minimization

Recall: Empirical risk minimization (ERM)

- Find parameters $\boldsymbol{\theta}$ that minimizes the **empirical risk**

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

- (+) Simple and easy-to-use
- (+) Nice statistical guarantees for *i.i.d.* data distributions

- Yet, minimizing the average loss as in ERM has known drawbacks
  - (−) Models susceptible to outliers [Jiang'18; Khetan'18]
  - (−) Unfair to a subgroup in the data [Hashimoto'18; Samadi'18]
  - (−) Brittle to shifts in distribution [Lin'17; Namkoong & Duchi'17]

Jiang et al., "MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels", ICML 2018
Khetan et al., "Learning from noisy singly-labeled data", ICLR 2018
Hashimoto et al., "Fairness without demographics in repeated loss minimization", ICML 2018
Samadi et al., "The price of fair PCA: One extra dimension", NeurIPS 2018
Lin et al., "Focal loss for dense object detection", ICCV 2017
Namkoong & Duchi, "Variance-based regularization with convex objectives", NeurIPS 2017

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

Part 3. Beyond ERM

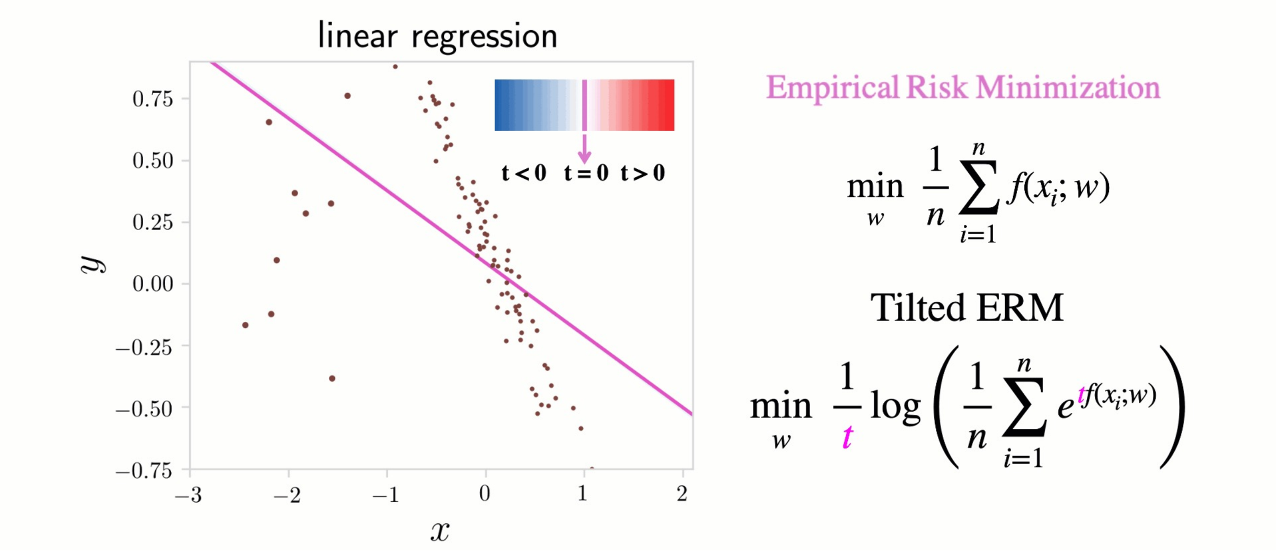
- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Tilted Empirical Risk Minimization (TERM) [Li’21]

Li et al. (2021): A “tilted” version of ERM with $t \in \mathbb{R}^{\backslash 0}$

$$\min_{\boldsymbol{\theta}} \tilde{R}_t(\boldsymbol{\theta}) := \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})} \right)$$

- $\boldsymbol{t \to 0}$: Recovers the original ERM $\bar{R}(\boldsymbol{\theta})$
- $\boldsymbol{t < 0}$: Robust regression/classification [Wang’13]
- $\boldsymbol{t > 0}$: Exponential smoothing [Kort & Bertsekas’72; Pee & Royset’11]



Empirical Risk Minimization

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} f(x_i; w)$$

Tilted ERM

$$\min_{w} \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t f(x_i; w)} \right)$$

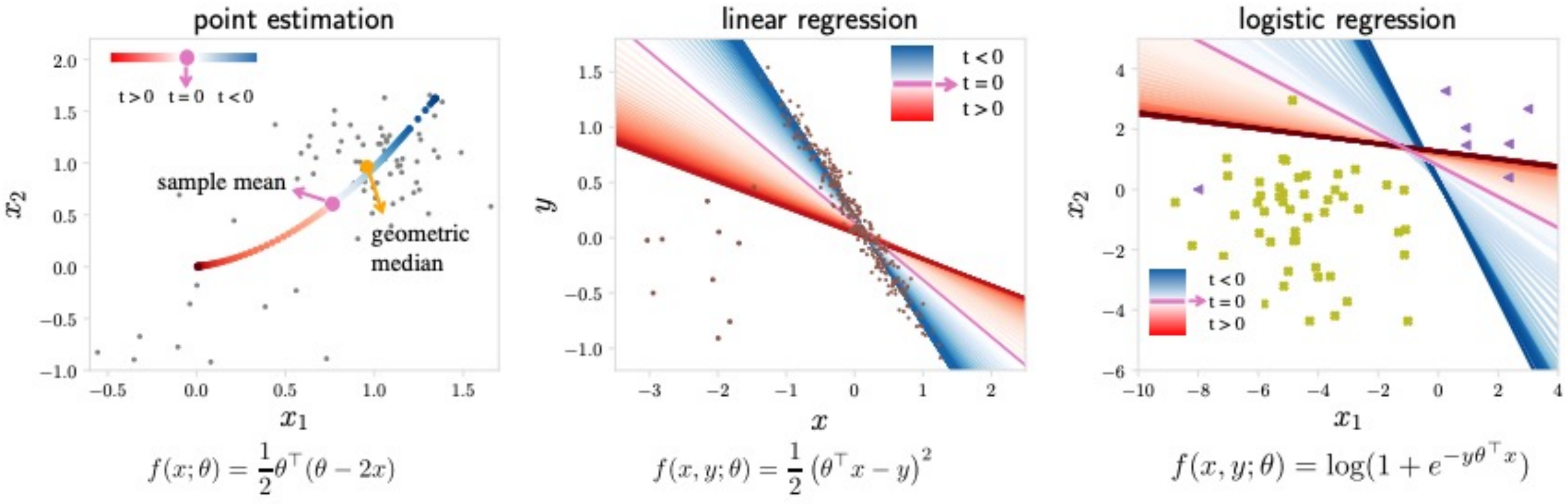
Li et al., “Tilted Empirical Risk Minimization”, ICLR 2021
Wang et al., “Robust variable selection with exponential squared loss”, 2013
Kort & Bertsekas, “A new penalty function method for constrained minimization”, 1972
Pee & Royset, “On solving large-scale finite minimax problems using exponential smoothing”, 2011

Tilted Empirical Risk Minimization (TERM) [Li'21]

Li et al. (2021): A "tilted" version of ERM with $t \in \mathbb{R}^{\backslash 0}$

$$\min_{\boldsymbol{\theta}} \tilde{R}_t(\boldsymbol{\theta}) := \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})} \right)$$

- $\boldsymbol{t} \rightarrow \mathbf{0}$: Recovers the original ERM $\bar{R}(\boldsymbol{\theta})$
- $\boldsymbol{t} < \mathbf{0}$: Robust regression/classification [Wang'13]
- $\boldsymbol{t} > \mathbf{0}$: Exponential smoothing [Kort & Bertsekas'72; Pee & Royset'11]



Li et al., "Tilted Empirical Risk Minimization", ICLR 2021
Wang et al., "Robust variable selection with exponential squared loss", 2013
Kort & Bertsekas, "A new penalty function method for constrained minimization", 1972
Pee & Royset, "On solving large-scale finite minimax problems using exponential smoothing", 2011

Tilted Empirical Risk Minimization (TERM) [Li'21]

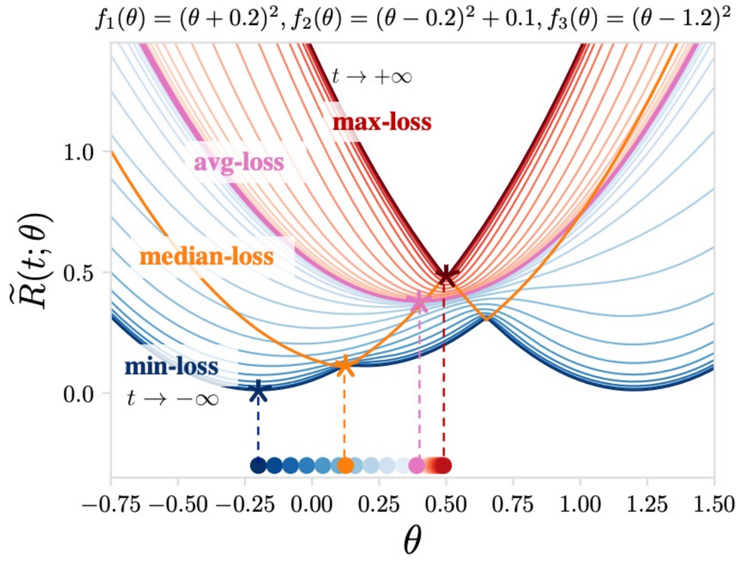
Li et al. (2021): A "tilted" version of ERM with $t \in \mathbb{R}^{\backslash 0}$

$$\min_{\boldsymbol{\theta}} \tilde{R}_t(\boldsymbol{\theta}) := \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})} \right)$$

- **Property 1**: Reweights the importance of outlier samples

$$\nabla \tilde{R}_t(\boldsymbol{\theta}) = \sum_i w_i \nabla_{\boldsymbol{\theta}} \ell(\mathbf{x}_i, y_i, \boldsymbol{\theta}), \text{ where } w_i \propto e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})}$$

- **Property 2**: Trade-off between min-/max-loss
  - $t \to -\infty$: min-loss / $t \to \infty$: max-loss
- **Property 3**: Approximates quantile losses
  - They are usually hard to directly optimize
  - **Example**: Median loss



Li et al., "Tilted Empirical Risk Minimization", ICLR 2021

Tilted Empirical Risk Minimization (TERM) [Li'21]

Li et al. (2021): A "tilted" version of ERM with $t \in \mathbb{R}^{\backslash 0}$

$$\min_{\boldsymbol{\theta}} \tilde{R}_t(\boldsymbol{\theta}) := \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})} \right)$$

- **Application 1**: Robust regression/classification
  - Simply setting $t = -2 < 0$ significantly improves ERM under label noise
  - Better than existing solutions tailored to individual tasks

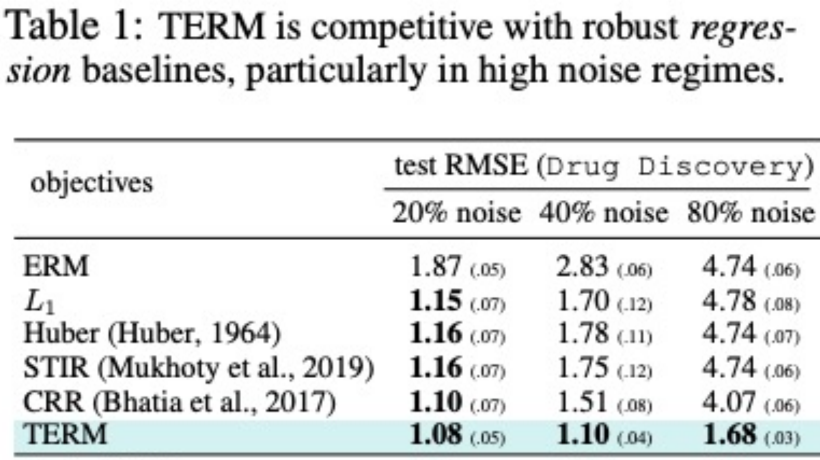


Table 1: TERM is competitive with robust *regression* baselines, particularly in high noise regimes.

objectives	test RMSE (`Drug Discovery`)		
	20% noise	40% noise	80% noise
ERM	1.87 (.05)	2.83 (.06)	4.74 (.06)
L_1	**1.15** (.07)	1.70 (.12)	4.78 (.08)
Huber (Huber, 1964)	**1.16** (.07)	1.78 (.11)	4.74 (.07)
STIR (Mukhoty et al., 2019)	**1.16** (.07)	1.75 (.12)	4.74 (.06)
CRR (Bhatia et al., 2017)	**1.10** (.07)	1.51 (.08)	4.07 (.06)
TERM	**1.08** (.05)	**1.10** (.04)	**1.68** (.03)

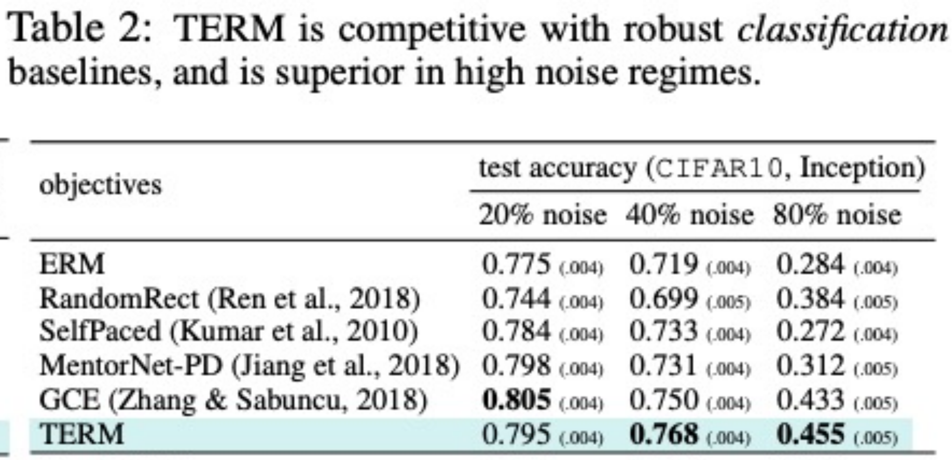


Table 2: TERM is competitive with robust *classification* baselines, and is superior in high noise regimes.

objectives	test accuracy (`CIFAR10`, Inception)		
	20% noise	40% noise	80% noise
ERM	0.775 (.004)	0.719 (.004)	0.284 (.004)
RandomRect (Ren et al., 2018)	0.744 (.004)	0.699 (.005)	0.384 (.005)
SelfPaced (Kumar et al., 2010)	0.784 (.004)	0.733 (.004)	0.272 (.004)
MentorNet-PD (Jiang et al., 2018)	0.798 (.004)	0.731 (.004)	0.312 (.005)
GCE (Zhang & Sabuncu, 2018)	**0.805** (.004)	0.750 (.004)	0.433 (.005)
TERM	0.795 (.004)	**0.768** (.004)	**0.455** (.005)

Li et al., "Tilted Empirical Risk Minimization", ICLR 2021

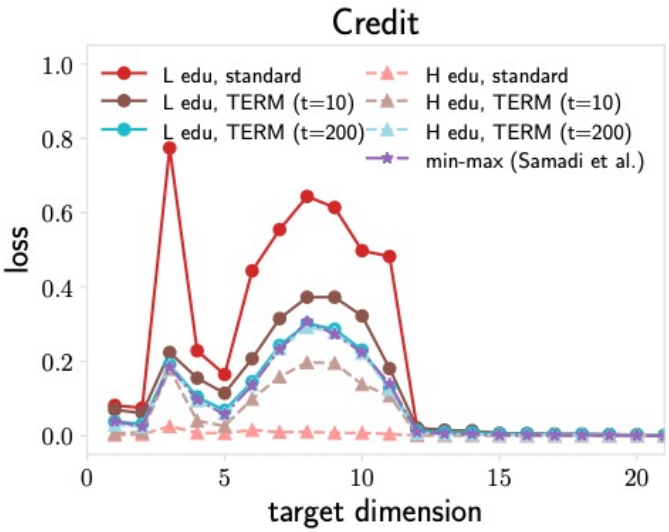
Tilted Empirical Risk Minimization (TERM) [Li'21]

Li et al. (2021): A "tilted" version of ERM with $t \in \mathbb{R}^{\backslash 0}$

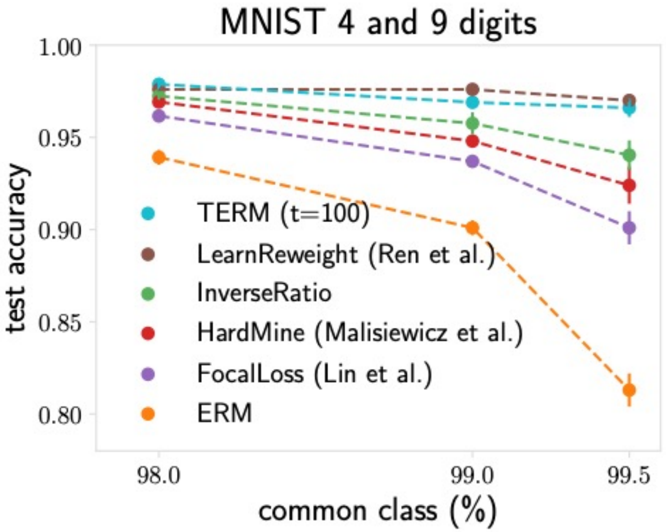
$$\min_{\boldsymbol{\theta}} \tilde{R}_t(\boldsymbol{\theta}) := \frac{1}{t} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{t \cdot \ell(\mathbf{x}_i, y_i; \boldsymbol{\theta})} \right)$$

- **Application 2**: Fairness and class-imbalance
  - TERM with $t > 0$ to improve fairness and imbalanced classification
  - Competitive with state-of-the-art methods

Fair PCA (Two groups: H/L)



Imbalanced classification



Li et al., "Tilted Empirical Risk Minimization", ICLR 2021

Table of Contents

Part 1. Basics

- Gradient descent (GD) and stochastic GD (SGD)
- Adaptive optimizers and learning rate scheduling
- Normalization layers

Part 2. Advanced Topics

- Normalization-free networks
- Pitfalls in momentum-based optimizers
- Large-batch training of deep neural networks

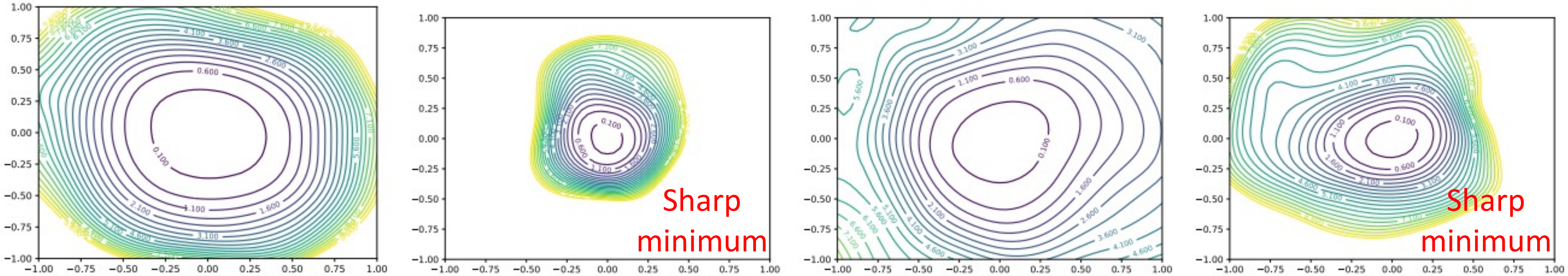
Part 3. Beyond ERM

- Tilted Empirical Risk Minimization
- Sharpness-aware Minimization

Sharpness-aware Minimization (SAM) [Foret'21]

Recall: "Flat-minima" generalize better [Keskar'17]

- Larger batch-sizes tend to make loss surface sharper, leading to worse generalization

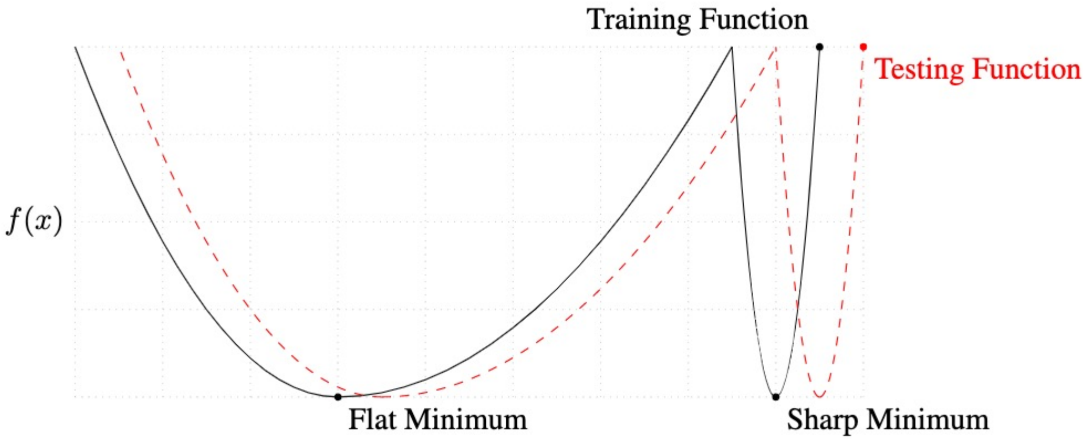


Sharp minimum

Sharp minimum

(e) 0.0, 128, 7.37% (f) 0.0, 8192, 11.07% (g) 5e-4, 128, 6.00% (h) 5e-4, 8192, 10.19%

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



Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017
Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

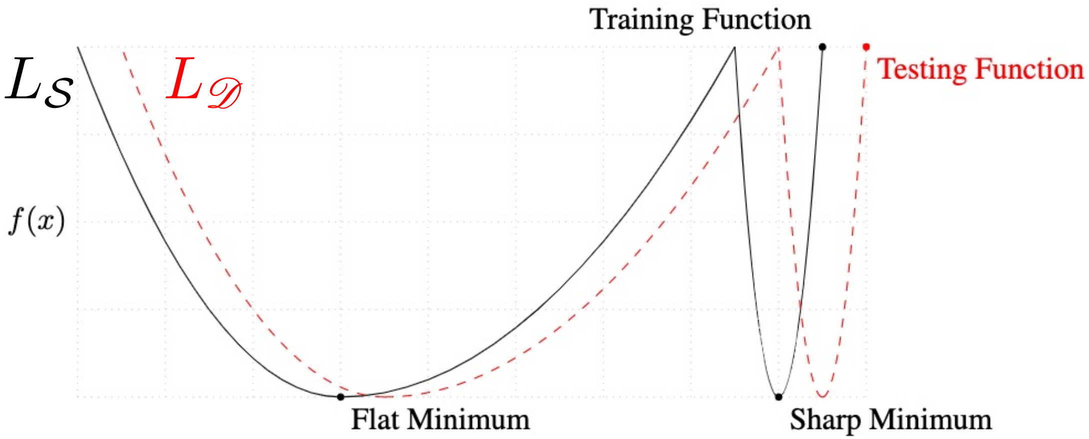
Recall: "Flat-minima" generalize better [Keskar'17]

- Motivated by this, Foret et al. (2021) shows a flatness-based bound:

Theorem (stated informally) 1. *For any* $\rho > 0$*, with high probability over training set* $\mathcal{S}$ *generated from distribution* $\mathscr{D}$*,*

$$\begin{aligned} L_{\mathscr{D}}(\boldsymbol{w}) &\leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + h(\|\boldsymbol{w}\|_2^2/\rho^2) \\ &= L_{\mathcal{S}}(\boldsymbol{w}) + \underbrace{[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) - L_{\mathcal{S}}(\boldsymbol{w})]}_{\text{“flatness”}} + h(\|\boldsymbol{w}\|_2^2/\rho^2), \end{aligned}$$

where $h : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ *is a strictly increasing function (under some technical conditions on* $L_{\mathscr{D}}(\boldsymbol{w})$*).*



Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017
Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

Recall: "Flat-minima" generalize better [Keskar'17]

$$\begin{aligned} L_{\mathscr{D}}(\boldsymbol{w}) &\leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + h(\|\boldsymbol{w}\|_2^2/\rho^2) \\ &= L_{\mathcal{S}}(\boldsymbol{w}) + \underbrace{[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) - L_{\mathcal{S}}(\boldsymbol{w})]}_{\text{“flatness”}} + h(\|\boldsymbol{w}\|_2^2/\rho^2), \end{aligned}$$

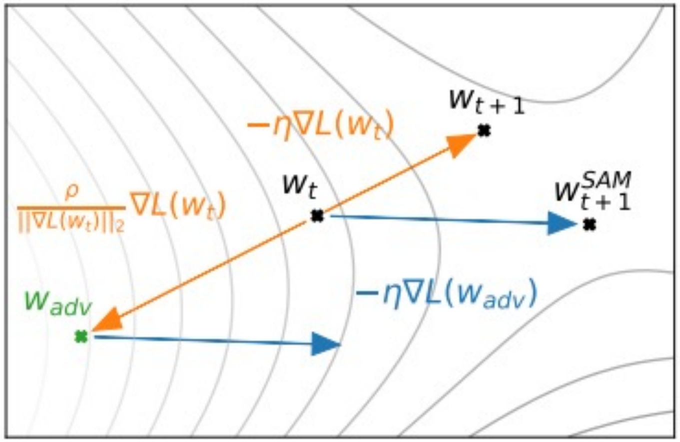
Foret et al. (2021): **Sharpness-aware Minimization (SAM)**

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}),$$

- With a linear approximation, SAM can be optimized via a 2-step gradient descent:

$$\nabla_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L_{\mathcal{S}}(w)|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}.$$

$$\hat{\boldsymbol{\epsilon}}(\boldsymbol{w}) = \rho \cdot \frac{\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})}{\|\nabla_{\boldsymbol{w}} L_{\mathcal{S}}(\boldsymbol{w})\|_2}$$



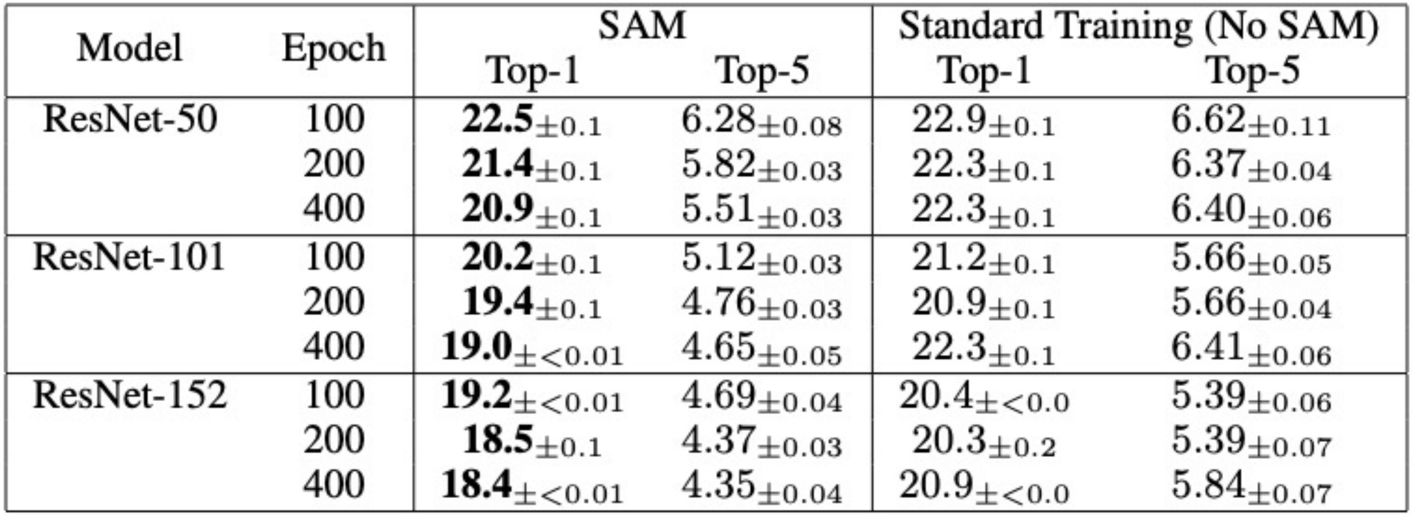
Keskar et al., "On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima", ICLR 2017
Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

Foret et al. (2021): **Sharpness-aware Minimization (SAM)**

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w}+\boldsymbol{\epsilon}),$$

- SAM consistently improves model generalization compared to SGD



Model	Epoch	SAM		Standard Training (No SAM)	
		Top-1	Top-5	Top-1	Top-5
ResNet-50	100	**22.5** $_{\pm 0.1}$	6.28 $_{\pm 0.08}$	22.9 $_{\pm 0.1}$	6.62 $_{\pm 0.11}$
	200	**21.4** $_{\pm 0.1}$	5.82 $_{\pm 0.03}$	22.3 $_{\pm 0.1}$	6.37 $_{\pm 0.04}$
	400	**20.9** $_{\pm 0.1}$	5.51 $_{\pm 0.03}$	22.3 $_{\pm 0.1}$	6.40 $_{\pm 0.06}$
ResNet-101	100	**20.2** $_{\pm 0.1}$	5.12 $_{\pm 0.03}$	21.2 $_{\pm 0.1}$	5.66 $_{\pm 0.05}$
	200	**19.4** $_{\pm 0.1}$	4.76 $_{\pm 0.03}$	20.9 $_{\pm 0.1}$	5.66 $_{\pm 0.04}$
	400	**19.0** $_{\pm <0.01}$	4.65 $_{\pm 0.05}$	22.3 $_{\pm 0.1}$	6.41 $_{\pm 0.06}$
ResNet-152	100	**19.2** $_{\pm <0.01}$	4.69 $_{\pm 0.04}$	20.4 $_{\pm <0.0}$	5.39 $_{\pm 0.06}$
	200	**18.5** $_{\pm 0.1}$	4.37 $_{\pm 0.03}$	20.3 $_{\pm 0.2}$	5.39 $_{\pm 0.07}$
	400	**18.4** $_{\pm <0.01}$	4.35 $_{\pm 0.04}$	20.9 $_{\pm <0.0}$	5.84 $_{\pm 0.07}$

Table 2: Test error rates for ResNets trained on ImageNet, with and without SAM.

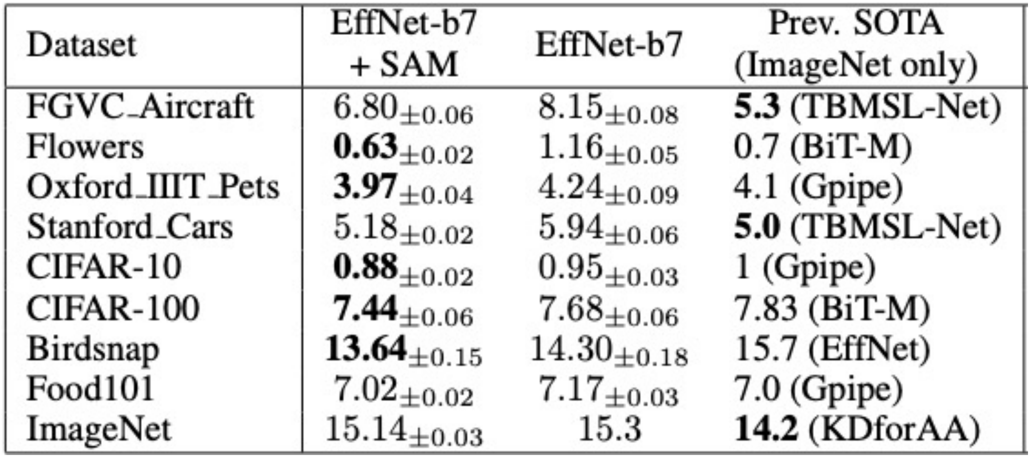
Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

Foret et al. (2021): **Sharpness-aware Minimization (SAM)**

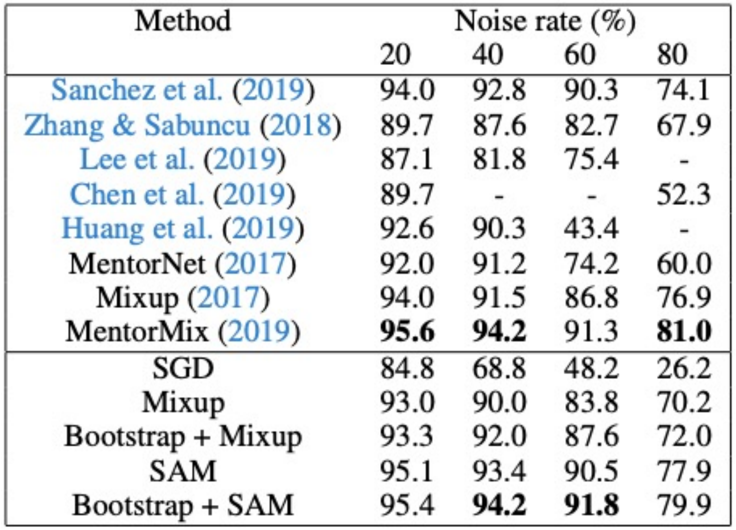
$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w}+\boldsymbol{\epsilon}),$$

- SAM consistently improves model generalization compared to SGD
- SAM also improves transfer learning and robustness to label noise



Dataset	EffNet-b7 + SAM	EffNet-b7	Prev. SOTA (ImageNet only)
FGVC_Aircraft	$6.80_{\pm0.06}$	$8.15_{\pm0.08}$	**5.3** (TBMSL-Net)
Flowers	$\mathbf{0.63}_{\pm0.02}$	$1.16_{\pm0.05}$	0.7 (BiT-M)
Oxford_IIIT_Pets	$\mathbf{3.97}_{\pm0.04}$	$4.24_{\pm0.09}$	4.1 (Gpipe)
Stanford_Cars	$5.18_{\pm0.02}$	$5.94_{\pm0.06}$	**5.0** (TBMSL-Net)
CIFAR-10	$\mathbf{0.88}_{\pm0.02}$	$0.95_{\pm0.03}$	1 (Gpipe)
CIFAR-100	$\mathbf{7.44}_{\pm0.06}$	$7.68_{\pm0.06}$	7.83 (BiT-M)
Birdsnap	$\mathbf{13.64}_{\pm0.15}$	$14.30_{\pm0.18}$	15.7 (EffNet)
Food101	$7.02_{\pm0.02}$	$7.17_{\pm0.03}$	7.0 (Gpipe)
ImageNet	$15.14_{\pm0.03}$	15.3	**14.2** (KDforAA)

Transfer learning
(pretrained on ImageNet)



Method	Noise rate (%)			
	20	40	60	80
Sanchez et al. (2019)	94.0	92.8	90.3	74.1
Zhang & Sabuncu (2018)	89.7	87.6	82.7	67.9
Lee et al. (2019)	87.1	81.8	75.4	-
Chen et al. (2019)	89.7	-	-	52.3
Huang et al. (2019)	92.6	90.3	43.4	-
MentorNet (2017)	92.0	91.2	74.2	60.0
Mixup (2017)	94.0	91.5	86.8	76.9
MentorMix (2019)	**95.6**	**94.2**	91.3	**81.0**
SGD	84.8	68.8	48.2	26.2
Mixup	93.0	90.0	83.8	70.2
Bootstrap + Mixup	93.3	92.0	87.6	72.0
SAM	95.1	93.4	90.5	77.9
Bootstrap + SAM	95.4	**94.2**	**91.8**	79.9

Label noise (CIFAR-10)

Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

Foret et al. (2021): **Sharpness-aware Minimization (SAM)**

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}),$$

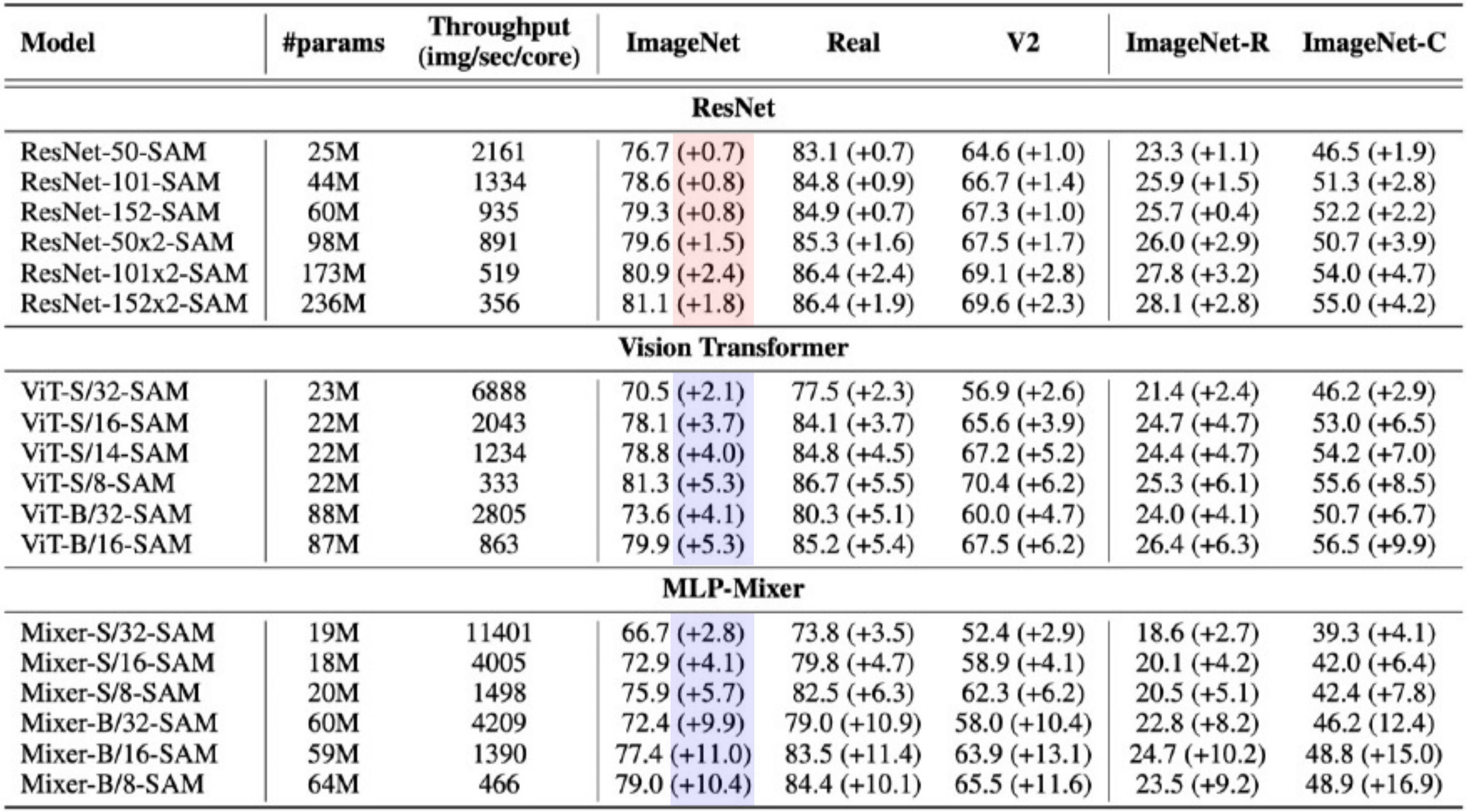
- SAM consistently improves model generalization compared to SGD
- SAM also improves transfer learning and robustness to label noise
- Visualization of loss surface on two random directions (ResNet)


SGD SAM

Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Sharpness-aware Minimization (SAM) [Foret'21]

- SAM has been getting attention due to its particular effectiveness on recent architectures, e.g., **ViT** or **Mixers** [Chen'22]
  - Accuracy gains from SAM are more significant in ViTs compared to ResNets
  - SAM closes the gap ResNet ↔ ViT in mid-sized datasets, e.g., ImageNet



Model	#params	Throughput (img/sec/core)	ImageNet	Real	V2	ImageNet-R	ImageNet-C
ResNet							
ResNet-50-SAM	25M	2161	76.7 (+0.7)	83.1 (+0.7)	64.6 (+1.0)	23.3 (+1.1)	46.5 (+1.9)
ResNet-101-SAM	44M	1334	78.6 (+0.8)	84.8 (+0.9)	66.7 (+1.4)	25.9 (+1.5)	51.3 (+2.8)
ResNet-152-SAM	60M	935	79.3 (+0.8)	84.9 (+0.7)	67.3 (+1.0)	25.7 (+0.4)	52.2 (+2.2)
ResNet-50x2-SAM	98M	891	79.6 (+1.5)	85.3 (+1.6)	67.5 (+1.7)	26.0 (+2.9)	50.7 (+3.9)
ResNet-101x2-SAM	173M	519	80.9 (+2.4)	86.4 (+2.4)	69.1 (+2.8)	27.8 (+3.2)	54.0 (+4.7)
ResNet-152x2-SAM	236M	356	81.1 (+1.8)	86.4 (+1.9)	69.6 (+2.3)	28.1 (+2.8)	55.0 (+4.2)
Vision Transformer							
ViT-S/32-SAM	23M	6888	70.5 (+2.1)	77.5 (+2.3)	56.9 (+2.6)	21.4 (+2.4)	46.2 (+2.9)
ViT-S/16-SAM	22M	2043	78.1 (+3.7)	84.1 (+3.7)	65.6 (+3.9)	24.7 (+4.7)	53.0 (+6.5)
ViT-S/14-SAM	22M	1234	78.8 (+4.0)	84.8 (+4.5)	67.2 (+5.2)	24.4 (+4.7)	54.2 (+7.0)
ViT-S/8-SAM	22M	333	81.3 (+5.3)	86.7 (+5.5)	70.4 (+6.2)	25.3 (+6.1)	55.6 (+8.5)
ViT-B/32-SAM	88M	2805	73.6 (+4.1)	80.3 (+5.1)	60.0 (+4.7)	24.0 (+4.1)	50.7 (+6.7)
ViT-B/16-SAM	87M	863	79.9 (+5.3)	85.2 (+5.4)	67.5 (+6.2)	26.4 (+6.3)	56.5 (+9.9)
MLP-Mixer							
Mixer-S/32-SAM	19M	11401	66.7 (+2.8)	73.8 (+3.5)	52.4 (+2.9)	18.6 (+2.7)	39.3 (+4.1)
Mixer-S/16-SAM	18M	4005	72.9 (+4.1)	79.8 (+4.7)	58.9 (+4.1)	20.1 (+4.2)	42.0 (+6.4)
Mixer-S/8-SAM	20M	1498	75.9 (+5.7)	82.5 (+6.3)	62.3 (+6.2)	20.5 (+5.1)	42.4 (+7.8)
Mixer-B/32-SAM	60M	4209	72.4 (+9.9)	79.0 (+10.9)	58.0 (+10.4)	22.8 (+8.2)	46.2 (12.4)
Mixer-B/16-SAM	59M	1390	77.4 (+11.0)	83.5 (+11.4)	63.9 (+13.1)	24.7 (+10.2)	48.8 (+15.0)
Mixer-B/8-SAM	64M	466	79.0 (+10.4)	84.4 (+10.1)	65.5 (+11.6)	23.5 (+9.2)	48.9 (+16.9)

Foret et al., "Sharpness-aware Minimization for Efficiently Improving Generalization", ICLR 2021

Chen et al., "When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations", ICLR 2022

Summary

Deep learning is heavily relying on large-scale, non-convex optimization

- The loss function includes many local minima and critical points
- SGD can be too noisy and might be unstable
- Hard to find a good learning rate
- Gradients are often vanish/explode

Currently, SGD is an essential ingredient for training deep neural networks

- Momentum/adaptive optimizers are widely used
- Learning rate scheduling is often important
- Normalization layers significantly improves stability with some drawbacks

Recent optimization techniques cover more scalable and realistic setups

- Large-batch SGD for distributed training
- Risks beyond ERM for out-of-distribution generalization
- Optimization practices for recent architectures, e.g., Transformers

References

[Nesterov’83] Nesterov, Y. “A method of solving a convex programming problem with convergence rate O(1/k^2).” 1983
Link: http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=dan&paperid=46009&option_lang=eng

[Duchi’11] Duchi, J., Hazan, E., Singer, Y. “Adaptive subgradient methods for online learning and stochastic optimization.” JMLR 2011
Link : http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[Tieleman’12] Geoff Hinton’s Lecture 6e of Coursera Class
Link : http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[Zeiler’12] Zeiler, M. D. “AdaDelta: An Adaptive Learning Rate Method”, 2012
Link : https://arxiv.org/pdf/1212.5701.pdf

[Kingma’15] Kingma, D., Ba, J. “Adam: A method for stochastic optimization.” ICLR 2015
Link : https://arxiv.org/pdf/1412.6980.pdf

[Dozat’16] Dozat, T. “Incorporating Nesterov Momentum into Adam.” ICLR Workshop 2016
Link : http://cs229.stanford.edu/proj2015/054_report.pdf

[Loshchilov’19] Loshchilov, I., Hutter, F. “Decoupled Weight Decay Regularization.” ICLR 2019
Link : https://arxiv.org/pdf/1711.05101.pdf

[Smith’17] Smith, L. S., Kindermans P. J., Ying, C., Le, Q. V. “Don't Decay the Learning Rate, Increase the Batch Size.” ICLR 2017
Link : https://openreview.net/pdf?id=B1Yy1BxCZ

[Liu’20] 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’15] Smith, Leslie N. "Cyclical learning rates for training neural networks.” 2015
Link : https://arxiv.org/pdf/1506.01186.pdf

[Loshchilov’17] Loshchilov, I., Hutter, F. “SGDR: Stochastic Gradient Descent with Warm Restarts.” ICLR 2017
Link : https://arxiv.org/pdf/1608.03983.pdf

References

[Mahajan’18] 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

[You’18] You, Y., Zhang, Z., Hsieh, C., Demmel, J., Keutzer, K. “ImageNet Training in Minutes.” 2018
Link: https://arxiv.org/pdf/1709.05011.pdf

[Keskar’17] 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’18] 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’18] 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’19] 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’17] You, Y., Gitman, I., Ginsburg, B. “Large Batch Training of Convolutional Networks.” 2017
Link: https://arxiv.org/pdf/1708.03888.pdf

[You’20] 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’21] 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'15] 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'16] 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'20] 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'21a] 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'18] 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'17] 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'18] 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'18] Bjorck, J., Gomes, C., Selman, B., Weinberger, K. "Understanding Batch Normalization." NeurIPS 2018
Link: https://arxiv.org/pdf/1806.02375.pdf

[Wu'18] Wu, Y., He, K. "Group Normalization." ECCV 2018
Link: https://arxiv.org/pdf/1803.08494.pdf

References

[Brock’21b] 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’19] 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’14] 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’16] 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

[Pennington’17] Pennington, J., Schoenholz, S., Ganguli, S. "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice." NeurIPS 2017
Link: https://proceedings.neurips.cc/paper/2017/file/d9fc0cdb67638d50f411432d0d41d0ba-Paper.pdf

[Xiao’19] Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S. and Pennington, J. “Dynamical Isometry and a Mean Field The ory of CNNs.” ICML 2019
Link: http://proceedings.mlr.press/v80/xiao18a/xiao18a.pdf

[Touvron’21] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A. and Jégou, H. “Training data-efficient image tran sformers & distillation through attention.” ICML 2021
Link: http://proceedings.mlr.press/v139/touvron21a/touvron21a.pdf

[Liu’21] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. and Guo, B. “Swin Transformer: Hierarchical Vision Transfor mer using Shifted Windows.” ICCV 2021
Link: https://openaccess.thecvf.com/content/ICCV2021/papers/Liu_Swin_Transformer_Hierarchical_Vision_Transformer_Using_Shifted_Windows_ICCV_2021_paper.pdf

References

[He’21] He, K., Chen, X., Xie, S., Li, Y., Dollár, P. and Girshick, R. “Masked Autoencoders Are Scalable Vision Learners.” 2021
Link: https://arxiv.org/pdf/2111.06377.pdf

[Liu’22] Liu, Z., Mao, H., Wu, C. Y., Feichtenhofer, C., Darrell, T. and Xie, S. “A ConvNet for the 2020s.” 2022
Link: https://arxiv.org/pdf/2201.03545.pdf

[Li’22] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., and others. “Competition-Level Code Generation with AlphaCode.” 2022
Link: https://storage.googleapis.com/deepmind-media/AlphaCode/competition_level_code_generation_with_alphacode.pdf

[Heo’21] Heo, B., Chun, S., Oh, S. J., Han, D., Yun, S., Kim, G., Uh, Y. and Ha, J. W. “AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights.” ICLR 2021
Link: https://arxiv.org/pdf/2006.08217.pdf

[Jiang’18] Jiang, L., Zhou, Z., Leung, T., Li, L. J. and Fei-Fei, L. “MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels.” ICML 2018
Link: http://proceedings.mlr.press/v80/jiang18c/jiang18c.pdf

[Khetan’18] Khetan, A., Lipton, Z. C. and Anandkumar, A. “Learning from noisy singly-labeled data.” ICLR 2018
Link: https://arxiv.org/pdf/1712.04577.pdf

[Hashimoto’18] Hashimoto, T., Srivastava, M., Namkoong, H. and Liang, P. “Fairness without demographics in repeated loss minimization.” ICML 2018
Link: http://proceedings.mlr.press/v80/hashimoto18a/hashimoto18a.pdf

[Samadi’18] Samadi, S., Tantipongpipat, U., Morgenstern, J. H., Singh, M. and Vempala, S. “The price of fair PCA: One extra dimension.” NeurIPS 2018
Link: https://proceedings.neurips.cc/paper/2018/file/cc4af25fa9d2d5c953496579b75f6f6c-Paper.pdf

[Lin’17] Lin, T. Y., Goyal, P., Girshick, R., He, K. and Dollár, P. “Focal loss for dense object detection.” ICCV 2017
Link: https://openaccess.thecvf.com/content_ICCV_2017/papers/Lin_Focal_Loss_for_ICCV_2017_paper.pdf

References

[Namkoong’17] Namkoong, H. and Duchi, J. C. “Variance-based regularization with convex objectives.” NeurIPS 2017
Link: https://proceedings.neurips.cc/paper/2017/file/5a142a55461d5fef016acfb927fee0bd-Paper.pdf

[Li’21] Li, T., Beirami, A., Sanjabi, M. and Smith, V. “Tilted Empirical Risk Minimization.” ICLR 2021
Link: https://arxiv.org/pdf/2007.01162.pdf

[Wang’13] Wang, X., Jiang, Y., Huang, M. and Zhang, H. “Robust variable selection with exponential squared loss.” 2013
Link: https://www.tandfonline.com/doi/pdf/10.1080/01621459.2013.766613

[Kort’72] Kort, B. W. and Bertsekas, D. P. “A new penalty function method for constrained minimization.” 1972
Link: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4044894

[Pee’11] Pee, E. Y. and Royset, J. O. “On solving large-scale finite minimax problems using exponential smoothing.” 2011
Link: https://link.springer.com/content/pdf/10.1007/s10957-010-9759-1.pdf

[Foret’21] Foret, P., Kleiner, A., Mobahi, H. and Neyshabur, B. “Sharpness-aware Minimization for Efficiently Improving Generalization.” ICLR 2021
Link: https://arxiv.org/pdf/2010.01412.pdf

[Chen’22] Chen, X., Hsieh, C. J. and Gong, B. “When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations.” ICLR 2022
Link: https://arxiv.org/pdf/2106.01548.pdf