Optimization for Deep Learning

Tao LIN

Learning and INference Systems (LINs) Lab, Westlake University

March 13, 2023

Table of Contents

Background

1. Get data: $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N$, where $\boldsymbol{\xi}_i := (\mathbf{d}, y)_i$

Background

1. Get data: $\xi_1, \ldots, \xi_N$, where $\xi_i := (\mathbf{d}, y)_i$
2. Choose a classifier

$$h_{\mathbf{x}}(\mathbf{d}) \to y$$

$$h_{\mathbf{x}}\left(\text{  } \right) \to \text{cat} \tag{1}$$

Background

1. Get data: $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N$, where $\boldsymbol{\xi}_i := (\mathbf{d}, y)_i$
2. Choose a classifier

$$h_{\mathbf{x}}(\mathbf{d}) \to y$$

$$h_{\mathbf{x}}\left(\quad \right) \to \text{cat} \tag{1}$$

3. Choose a loss function: $\ell(h_{\mathbf{x}}(\mathbf{d}, y)) \geq 0$

Background

1. Get data: $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N$, where $\boldsymbol{\xi}_i := (\mathbf{d}, y)_i$
2. Choose a classifier

$$h_{\mathbf{x}}(\mathbf{d}) \to y$$

$$h_{\mathbf{x}}\left(\text{  } \right) \to \text{cat} \tag{1}$$

3. Choose a loss function: $\ell(h_{\mathbf{x}}(\mathbf{d}, y)) \geq 0$
4. Solve the *training problem*:

$$\min_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^{N} \ell\left(h_{\mathbf{x}}(\mathbf{d}_i), y_i\right) \tag{2}$$

Background

Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

Background

Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$

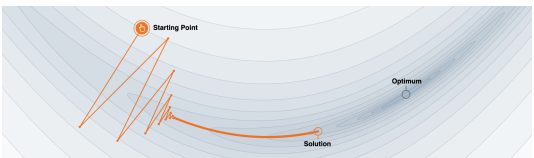
Background

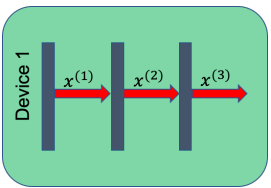
Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$
- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \, \nabla F(\mathbf{x}^{(t)}, \boldsymbol{\xi}_i) \tag{4}$$





Background

Finite-sum empirical risk minimization problem:

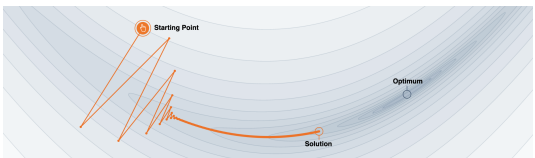
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

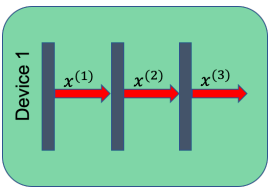
- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$

- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla F(\mathbf{x}^{(t)}, \boldsymbol{\xi}_i) \tag{4}$$

  - η is step-size/learning rate





Background

Finite-sum empirical risk minimization problem:

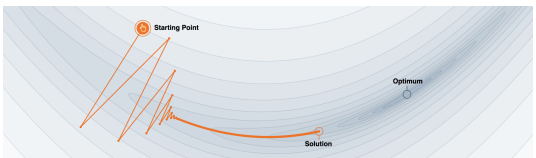
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

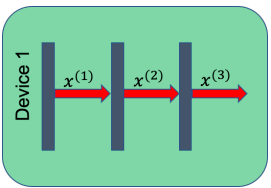
- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$

- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla F(\mathbf{x}^{(t)}, \boldsymbol{\xi}_i) \tag{4}$$

  - η is step-size/learning rate
  - sampled i.i.d. $i \in \{1, \ldots, N\}$





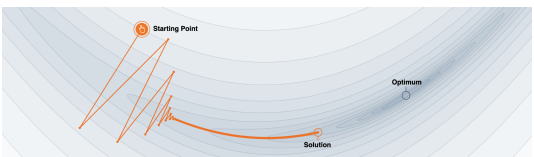
Background

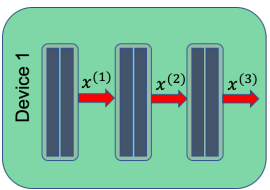
Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$
- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{B}\sum_{i\in\mathcal{B}} \eta \nabla f_i(\mathbf{x}) \qquad \text{(Using } \textit{mini-batching}\text{)}$$





Background

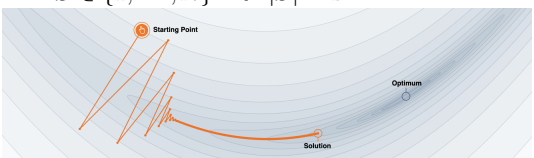
Finite-sum empirical risk minimization problem:

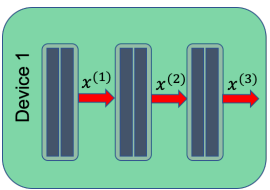
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \Big(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \Big) \right\} \tag{3}$$

- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$
- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{B}\sum_{i\in\mathcal{B}} \eta \nabla f_i(\mathbf{x}) \qquad \text{(Using } \textit{mini-batching}\text{)}$$

- $\mathcal{B} \in \{1, \ldots, N\}$ with $|\mathcal{B}| = B$.





Random sampling vector $\mathbf{v} = (v_1, \ldots, v_N) \sim \mathcal{D}$ with $\mathbb{E}[v_i] = 1$ for $i = 1, \ldots, N$.

Random sampling vector $\mathbf{v} = (v_1, \ldots, v_N) \sim \mathcal{D}$ with $\mathbb{E}[v_i] = 1$ for $i = 1, \ldots, N$.

$$f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[v_i] f_i(\mathbf{x}) = \mathbb{E}\left[\underbrace{\frac{1}{N}\sum_{i=1}^{N} v_i f_i(\mathbf{x})}_{=: f_v(\mathbf{x})}\right] \qquad \text{(Stochastic Reformulation)}$$

Random sampling vector $\mathbf{v} = (v_1, \ldots, v_N) \sim \mathcal{D}$ with $\mathbb{E}[v_i] = 1$ for $i = 1, \ldots, N$.

$$f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[v_i] f_i(\mathbf{x}) = \mathbb{E}\left[\underbrace{\frac{1}{N}\sum_{i=1}^{N} v_i f_i(\mathbf{x})}_{=: f_v(\mathbf{x})}\right] \qquad \text{(Stochastic Reformulation)}$$

Original Finite-sum problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \qquad (4)$$

Stochastic Reformulation

$$\min_{\mathbf{x}\in\mathbb{R}^d} \mathbb{E}[f_v(\mathbf{x})] \qquad (5)$$

Minimizing the expectation of **random linear combinations** of original function

Background: Stochastic reformulation of finite-sum problems: SGD with arbitrary sampling

Random sampling vector $\mathbf{v} = (v_1, \ldots, v_N) \sim \mathcal{D}$ with $\mathbb{E}[v_i] = 1$ for $i = 1, \ldots, N$.

$$f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[v_i] f_i(\mathbf{x}) = \mathbb{E}\left[\underbrace{\frac{1}{N}\sum_{i=1}^{N} v_i f_i(\mathbf{x})}_{=: f_v(\mathbf{x})}\right] \qquad \text{(Stochastic Reformulation)}$$

Original Finite-sum problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \qquad (4)$$

Sample $v_i^{(t)} \sim \mathcal{D}$

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta \nabla f_{v_i^{(t)}}(\mathbf{x}^{(t)}) \qquad (6)$$

Stochastic Reformulation

$$\min_{\mathbf{x}\in\mathbb{R}^d} \mathbb{E}[f_v(\mathbf{x})] \qquad (5)$$

Minimizing the expectation of **random linear combinations** of original function

Background: Stochastic reformulation of finite-sum problems: SGD with arbitrary sampling

Random sampling vector $\mathbf{v} = (v_1, \ldots, v_N) \sim \mathcal{D}$ with $\mathbb{E}[v_i] = 1$ for $i = 1, \ldots, N$.

$$f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[v_i] f_i(\mathbf{x}) = \mathbb{E}\left[\underbrace{\frac{1}{N}\sum_{i=1}^{N} v_i f_i(\mathbf{x})}_{=: f_v(\mathbf{x})}\right] \qquad \text{(Stochastic Reformulation)}$$

Original Finite-sum problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \qquad (4)$$

Sample $v_i^{(t)} \sim \mathcal{D}$

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta \nabla f_{v_i^{(t)}}(\mathbf{x}^{(t)}) \qquad (6)$$

Stochastic Reformulation

$$\min_{\mathbf{x}\in\mathbb{R}^d} \mathbb{E}[f_v(\mathbf{x})] \qquad (5)$$

Minimizing the expectation of **random linear combinations** of original function

The distribution $\mathcal{D}$ encodes any form of mini-batching / non-uniform sampling.

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \sigma^2$.

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \sigma^2$.

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T}}\right)$$

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\left\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\right\|^2 \leq \sigma^2$.

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

- L*-smoothness*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\left(f(\mathbf{x}_0) - f^\star\right)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\left(f(\mathbf{x}_0) - f^\star\right)}{T}}\right)$$

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\left\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\right\|^2 \leq \sigma^2$.

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

- L*-smoothness*
- T*: number of iterations*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T}}\right)$$

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \sigma^2$.

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

- L*-smoothness*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T}}\right)$$

- T*: number of iterations*
- σ*: stochastic gradient variance*

The convergence of mini-batch SGD

Assumption 1

- *The function* $f(\mathbf{x})$ *we are minimizing is lower bounded from below by* $f^\star := f(\mathbf{x}^\star)$*, and each* f_i *is* L*-smooth satisfying* $\|\nabla f_i(\mathbf{y}) - \nabla f_i(\mathbf{x})\| \leq L \|\mathbf{y} - \mathbf{x}\|$
- *The stochastic gradients satisfy* $\mathbb{E}\left[\nabla f_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$ *and* $\mathbb{E}\left\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\right\|^2 \leq \sigma^2$.

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

- L*-smoothness*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0) - f^\star)}{T}}\right)$$

- T*: number of iterations*
- σ*: stochastic gradient variance*
- B*: mini-batch size of* $\mathcal{B}$

The convergence of mini-batch SGD

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

- L-*smoothness*
- T: *number of iterations*
- σ: *stochastic gradient variance*
- B: *mini-batch size of* $\mathcal{B}$

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T}}\right)$$

- When iterations $T \to \infty$, it holds that $\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \to 0$

The convergence of mini-batch SGD

Theorem 1 (Convergence rate of mini-batch SGD for non-convex functions)

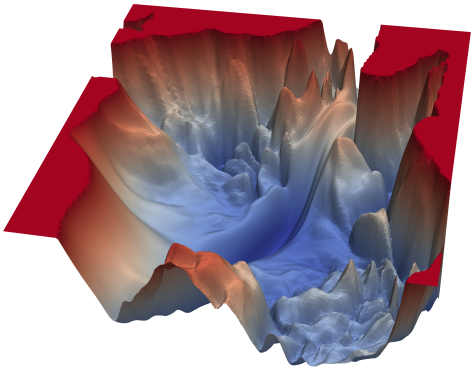
- L-*smoothness*
- T: *number of iterations*
- σ: *stochastic gradient variance*
- B: *mini-batch size of* $\mathcal{B}$

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T}}\right)$$

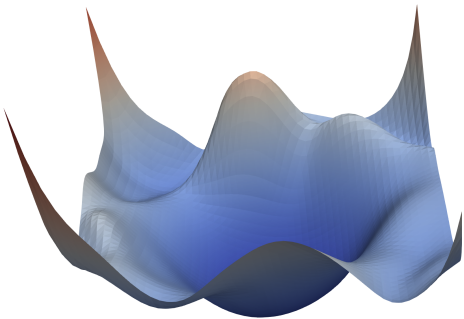
- When iterations $T \to \infty$, it holds that $\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \to 0$
- $\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \to 0$ implies the sequence converges to a stationary solution

Table of Contents

Issues of SGD—from the perspective of loss landscape



(a) ResNet w/o skip connections.

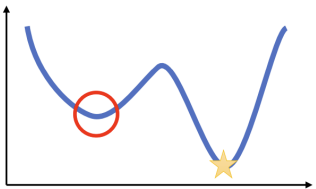


(b) ResNet w/ skip connections.

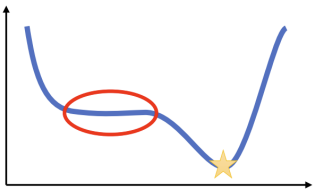
Figure: The surfaces of ResNet-56 w/ and w/o skip connections [4].

[4] *Li et al.* Visualizing the Loss Landscape of Neural Nets. NeurIPS 2018.

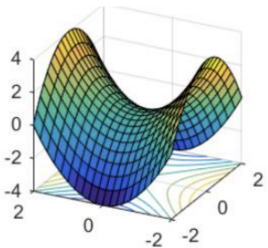
Issues of SGD—from the perspective of loss landscape



the local optimum



the plateau

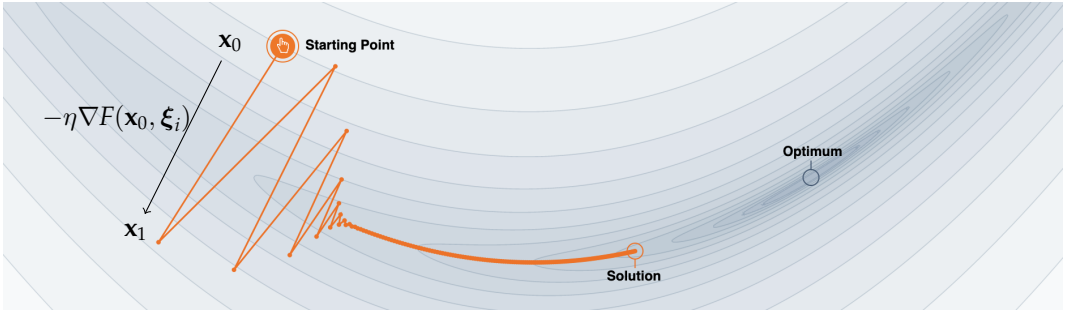


the saddle point

Challenging optimization loss landscape!

Figures borrowed from https://cs182sp21.github.io/static/slides/lec-4.pdf

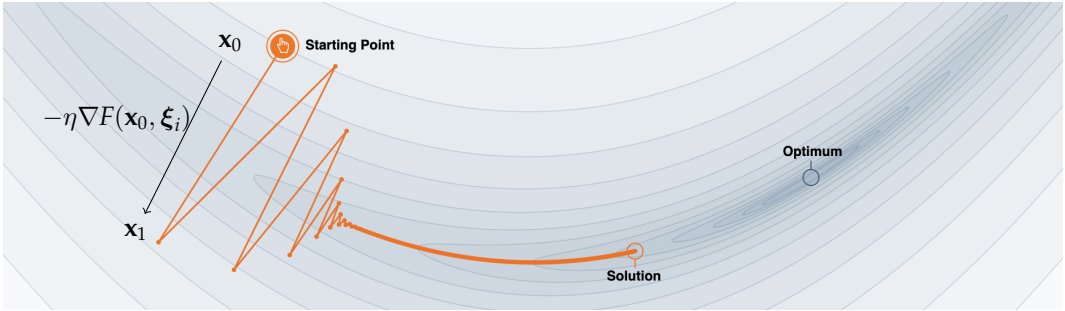
Issues of SGD—from the perspective of loss landscape



- **Challenges # 1**: loss function has high condition number.
  $\rightarrow$ very slow progress along shallow dimension, jitter along steep direction.

Visualizations based on Gabriel Goh's distill.pub article: https://distill.pub/2017/momentum/

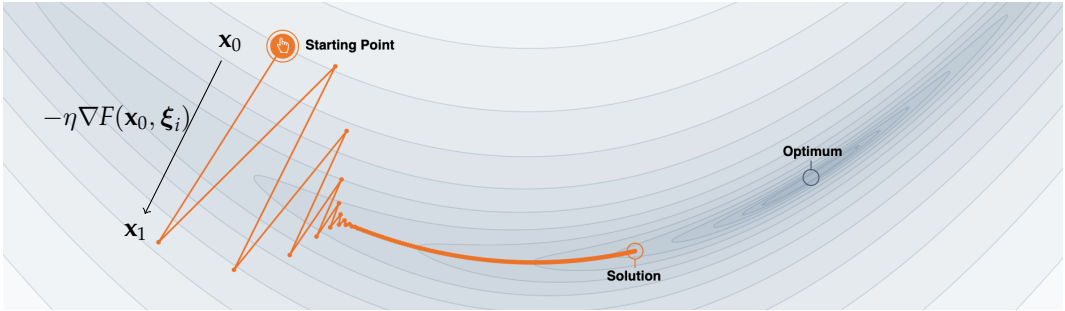
Issues of SGD—from the perspective of loss landscape



- **Challenges # 1**: loss function has high condition number.
  $\rightarrow$ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more**: plateaus & saddle points.

Visualizations based on Gabriel Goh's distill.pub article: https://distill.pub/2017/momentum/

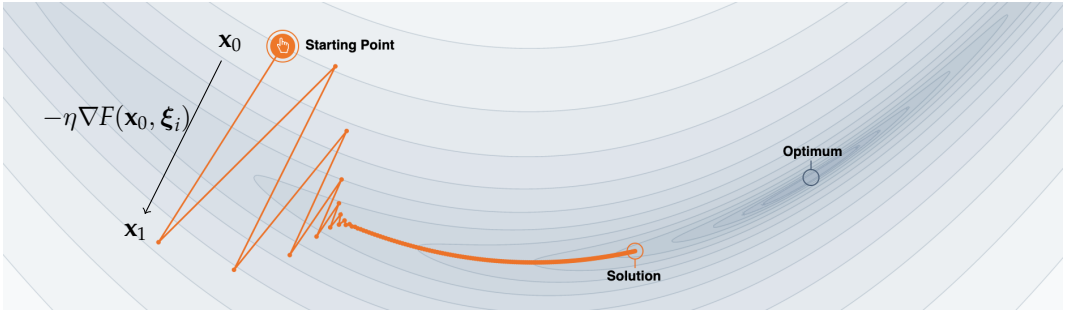
Issues of SGD—from the perspective of loss landscape



- **Challenges # 1**: loss function has high condition number.
  $\rightarrow$ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more**: plateaus & saddle points.
  $\rightarrow$ cannot just choose tiny learning rates to prevent oscillation!

Visualizations based on Gabriel Goh's distill.pub article: `https://distill.pub/2017/momentum/`

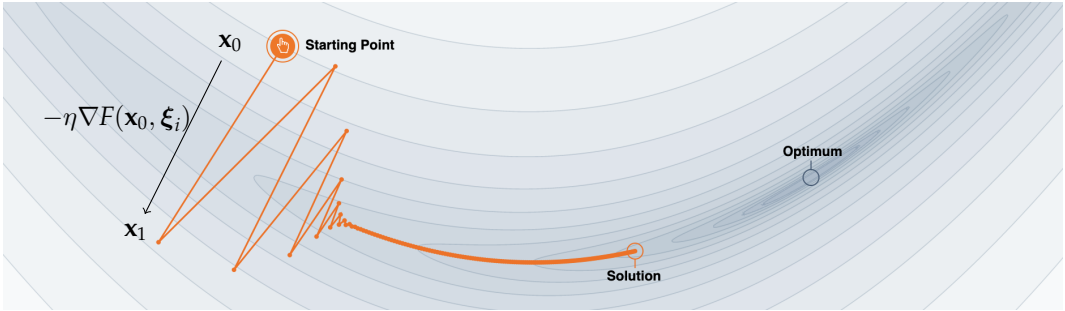
Issues of SGD—from the perspective of loss landscape



- **Challenges # 1**: loss function has high condition number.
  $\rightarrow$ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more**: plateaus & saddle points.
  $\rightarrow$ cannot just choose tiny learning rates to prevent oscillation!
  $\rightarrow$ need learning rates to be large enough not to get stuck in a plateau.

Visualizations based on Gabriel Goh's distill.pub article: `https://distill.pub/2017/momentum/`

Issues of SGD—from the perspective of loss landscape



- **Challenges # 1**: loss function has high condition number.
  $\rightarrow$ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more**: plateaus & saddle points.
  $\rightarrow$ cannot just choose tiny learning rates to prevent oscillation!
  $\rightarrow$ need learning rates to be large enough not to get stuck in a plateau.
  $\rightarrow$ saddle points have very small gradients: but much more common in high dimension.

Visualizations based on Gabriel Goh's distill.pub article: `https://distill.pub/2017/momentum/`

Improvement directions: leveraging the curvature information

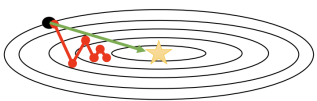
Can we find a better descent direction in the loss landscape?
Yes! **By leveraging the curvature information** through Newton's method.

Improvement directions: leveraging the curvature information

Can we find a better descent direction in the loss landscape?
Yes! **By leveraging the curvature information** through Newton's method.

Taylor expansion:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 \qquad (7)$$

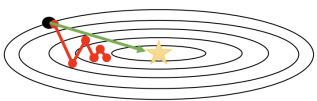


Improvement directions: leveraging the curvature information

Can we find a better descent direction in the loss landscape?
Yes! **By leveraging the curvature information** through Newton's method.

Taylor expansion:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 \tag{7}$$



Multivariate case:

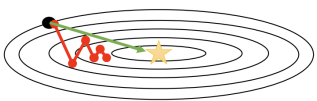
$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \underbrace{\nabla_{\mathbf{x}} f(\mathbf{x}_0)}_{\text{gradient}}(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top \underbrace{\nabla^2_{\mathbf{x}} f(\mathbf{x}_0)}_{\text{Hessian}}(\mathbf{x} - \mathbf{x}_0) \tag{8}$$

Improvement directions: leveraging the curvature information

Can we find a better descent direction in the loss landscape?
Yes! **By leveraging the curvature information** through Newton's method.

Taylor expansion:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 \tag{7}$$



Multivariate case:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \underbrace{\nabla_{\mathbf{x}} f(\mathbf{x}_0)}_{\text{gradient}}(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top \underbrace{\nabla^2_{\mathbf{x}} f(\mathbf{x}_0)}_{\text{Hessian}}(\mathbf{x} - \mathbf{x}_0) \tag{8}$$

Solution (can optimize this analytically!):

$$\mathbf{x}^\star \leftarrow \mathbf{x}_0 - \left(\nabla^2_{\mathbf{x}} f(\mathbf{x}_0)\right)^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}_0) \tag{9}$$

Improvement directions: trade-offs and approximations

Q: Why is Newton's method not a viable way to improve neural network optimization?

[1] if using naive approach, though fancy methods can be much faster if they avoid forming the Hessian explicitly.

Improvement directions: trade-offs and approximations

Q: Why is Newton's method not a viable way to improve neural network optimization?

GD (w/o Hessian): $\mathcal{O}(N)$ v.s. GD (w/ Hessian)[1]: $\mathcal{O}(N^3)$

[1] if using naive approach, though fancy methods can be much faster if they avoid forming the Hessian explicitly.

Improvement directions: trade-offs and approximations

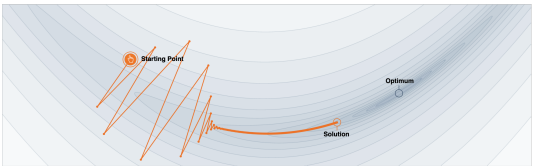
Q: Why is Newton's method not a viable way to improve neural network optimization?

GD (w/o Hessian): $\mathcal{O}(N)$ v.s. GD (w/ Hessian)[1]: $\mathcal{O}(N^3)$

We would prefer methods that don't require second derivatives, but somehow "stabilize" / "accelerate" gradient descent instead.

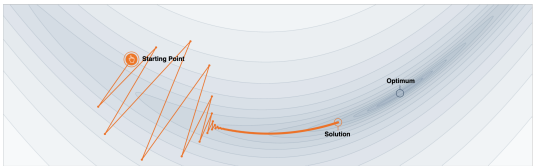
[1] if using naive approach, though fancy methods can be much faster if they avoid forming the Hessian explicitly.

Momentum method

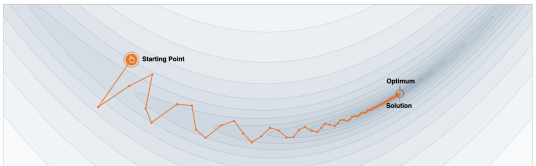


w/o momentum

Momentum method

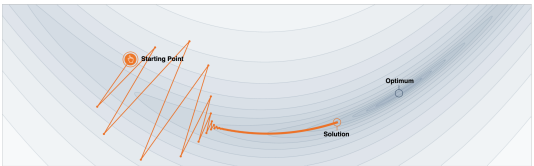


w/o momentum

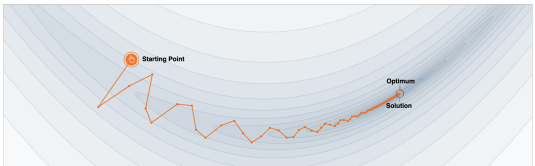


w/ momentum

Momentum method



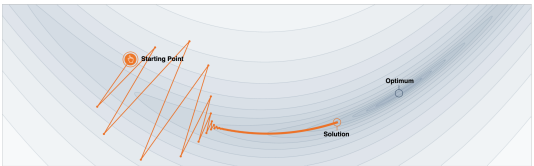
w/o momentum



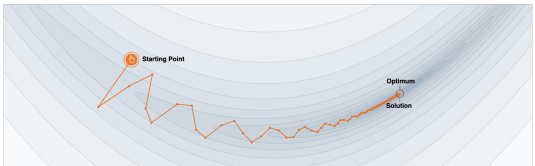
w/ momentum

Intuition: averaging together successive gradients yield a much better direction!

Momentum method



w/o momentum

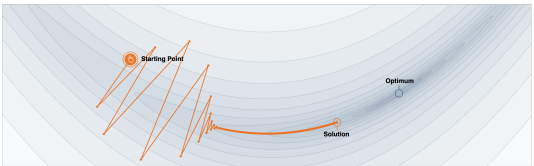


w/ momentum

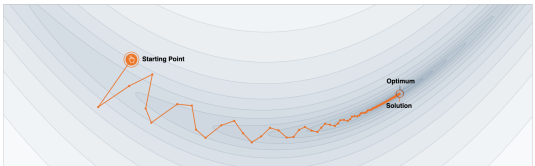
Intuition: averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions

Momentum method



w/o momentum

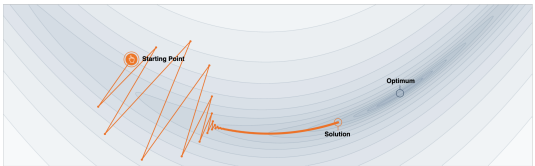


w/ momentum

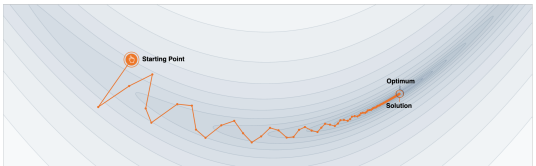
Intuition: averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

Momentum method



w/o momentum



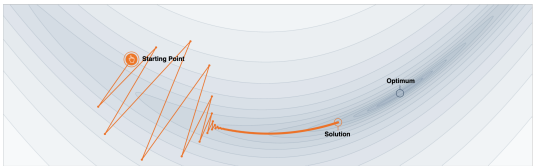
w/ momentum

Intuition: averaging together successive gradients yield a much better direction!

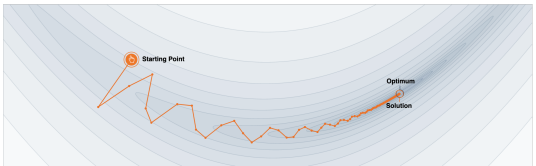
- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

- if successive gradient step point in similar directions

Momentum method



w/o momentum



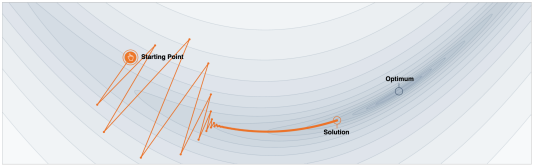
w/ momentum

Intuition: averaging together successive gradients yield a much better direction!

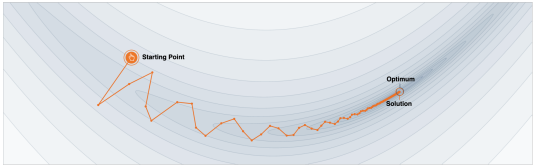
- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

- if successive gradient step point in similar directions
  $\rightarrow$ we should go faster in that direction

Momentum method



w/o momentum



w/ momentum

Intuition: averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

- if successive gradient step point in similar directions
  $\rightarrow$ we should go faster in that direction

$$\mathbf{m}_t = \beta \mathbf{m}_{t-1} + \nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\,, \mathbf{x}_{t+1} = \mathbf{x}_t - \eta \mathbf{m}_t \qquad \text{(SGD w/ momentum)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_0 - \eta \sum_{i=1}^{t} \nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i) \qquad \text{(Unroll SGD w/o momentum)}$$

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*:

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*:
- *magnitude*:

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*: the sign of the gradient tells us which way to go along each dimension;
- *magnitude*:

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*: the sign of the gradient tells us which way to go along each dimension;
- *magnitude*: the magnitude is not so great, and could be even worse:

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*: the sign of the gradient tells us which way to go along each dimension;
- *magnitude*: the magnitude is not so great, and could be even worse:

  $\rightarrow$ overall magnitude of the gradient can change drastically during the optimization,

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*: the sign of the gradient tells us which way to go along each dimension;
- *magnitude*: the magnitude is not so great, and could be even worse:

  $\rightarrow$ overall magnitude of the gradient can change drastically during the optimization, making learning rates hard to tune.

Methods that manipulate gradient scale

Intuition behind $\nabla F(\mathbf{x}_i, \boldsymbol{\xi}_i)$:

- *sign*: the sign of the gradient tells us which way to go along each dimension;
- *magnitude*: the magnitude is not so great, and could be even worse:

  $\rightarrow$ overall magnitude of the gradient can change drastically during the optimization, making learning rates hard to tune.

Idea: *normalize* out the magnitude of the gradient along each dimension.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + (\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t))^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + (\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t))^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t))^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

Remarks:

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

Remarks:

- AdaGrad has some appealing guarantees for convex problems.

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

Remarks:

- AdaGrad has some appealing guarantees for convex problems.
  → AdaGrad originally proposed to benefit from sparse data.

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

Remarks:

- AdaGrad has some appealing guarantees for convex problems.
  → AdaGrad originally proposed to benefit from sparse data.
  → Learning rate effectively "decreases" over time: good for convex (bad for non-convex).

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Methods that manipulate gradient scale: RMSProp, AdaGrad, and their differences

AdaGrad [1] (estimate per-dimension cumulative magnitude):

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

RMSProp (estimate per-dimension magnitude):

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(roughly the squared length of each dimension)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)}{\sqrt{\mathbf{v}_t}} \qquad \text{(each dimension is divided by its magnitude)}$$

Remarks:

- AdaGrad has some appealing guarantees for convex problems.
  → AdaGrad originally proposed to benefit from sparse data.
  → Learning rate effectively "decreases" over time: good for convex (bad for non-convex).
- RMSProp tends to be much better for deep learning (and most non-convex problems)

RMSProp. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[1] *Duchi et al.* Adaptive subgradient methods for online learning and stochastic optimization. COLT 2010.

Adam: combining momentum and RMSProp

Idea:

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) \qquad \text{(first moment estimate)}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(second moment estimate)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}} = \mathbf{x}_t - \underbrace{\frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}}_{\text{element-wise stepsize}} \mathbf{m}_t \qquad \text{(update step)}$$

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) \qquad \text{(first moment estimate)}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(second moment estimate)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}} = \mathbf{x}_t - \underbrace{\frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}}_{\text{element-wise stepsize}} \mathbf{m}_t \qquad \text{(update step)}$$

where compared to RMSProp, Adam

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) \qquad \text{(first moment estimate)}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(second moment estimate)}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}} = \mathbf{x}_t - \underbrace{\frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}}_{\text{element-wise stepsize}} \mathbf{m}_t \qquad \text{(update step)}$$

where compared to RMSProp, Adam

- replaces $\frac{\eta}{\sqrt{\mathbf{v}_t+\epsilon}}\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)$ by $\frac{\eta}{\sqrt{\mathbf{v}_t+\epsilon}}\mathbf{m}_t$.

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp

Idea:

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) \qquad \text{(first moment estimate)}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)\right)^2 \qquad \text{(second moment estimate)}$$

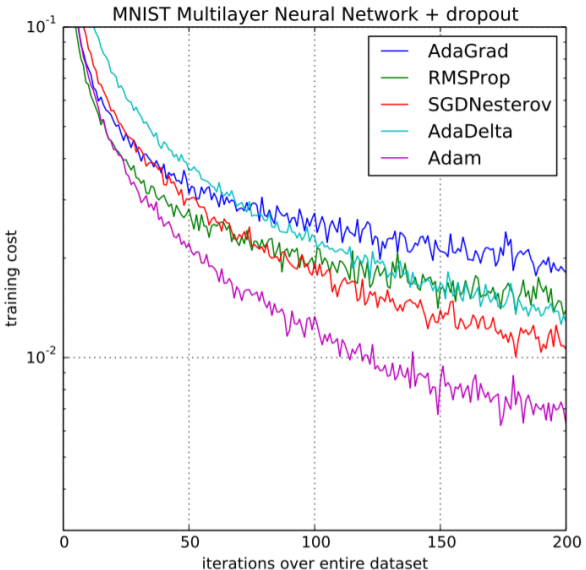
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}} = \mathbf{x}_t - \underbrace{\frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}}_{\text{element-wise stepsize}} \mathbf{m}_t \qquad \text{(update step)}$$

where compared to RMSProp, Adam

- replaces $\frac{\eta}{\sqrt{\mathbf{v}_t+\epsilon}}\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t)$ by $\frac{\eta}{\sqrt{\mathbf{v}_t+\epsilon}}\mathbf{m}_t$.
- adds bias correction (omitted in the expression above): it avoids large stepsizes in early stages of run (especially when β_2 is close to 1).

[3] *Kingma et al.* Adam: A method for stochastic optimization. ICLR 2015 (Google Scholar 12w+)

Adam: combining momentum and RMSProp



Weight decay in SGD and Adam: why AdamW matters

Many learning problems optimize the loss with L_2 norm penalty:

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + \lambda \|\mathbf{x}\|_2^2 \,, \tag{10}$$

Weight decay in SGD and Adam: why AdamW matters

Many learning problems optimize the loss with L_2 norm penalty:

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + \lambda \left\|\mathbf{x}\right\|_2^2 , \tag{10}$$

where it is sometimes called “weight decay” in SGD, since its gradient decays weight:

$$\underbrace{\mathbf{x} - \eta \nabla_{\mathbf{x}} \left(f(\mathbf{x}) + \lambda \left\|\mathbf{x}\right\|_2^2 \right)}_{\text{SGD on } L_2\text{-norm penalty}} \underset{\nabla_{\mathbf{x}} \|\mathbf{x}\|_2^2 = 2\mathbf{x}}{\Longleftrightarrow} \underbrace{(1 - 2\eta\lambda)\mathbf{x} - \eta \nabla_x f(\mathbf{x})}_{\text{weight decay}} \tag{11}$$

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

- L_2 regularization and weight decay are not identical (for momentum/adaptive SGD).

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

- L_2 regularization and weight decay are not identical (for momentum/adaptive SGD).
- L_2 regularization is not effective in Adam.

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

- L_2 regularization and weight decay are not identical (for momentum/adaptive SGD).
- L_2 regularization is not effective in Adam.
- Weight decay is equally effective in both SGD and Adam.

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

- L_2 regularization and weight decay are not identical (for momentum/adaptive SGD).
- L_2 regularization is not effective in Adam.
- Weight decay is equally effective in both SGD and Adam.

Decoupled SGD with momentum: (same trick applies to Adam)

$$\mathbf{m}_{t+1} = \beta \mathbf{m}_t + (1-\beta) \underbrace{\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) + \lambda \mathbf{x}_t \right)}_{\text{gradient of loss with } L_2 \text{ penalty}} \tag{10}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t - \underbrace{2\eta\lambda \mathbf{x}_t}_{\text{weight decay}} \tag{11}$$

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

- L_2 regularization and weight decay are not identical (for momentum/adaptive SGD).
- L_2 regularization is not effective in Adam.
- Weight decay is equally effective in both SGD and Adam.

Decoupled SGD with momentum: (same trick applies to Adam)

$$\mathbf{m}_{t+1} = \beta \mathbf{m}_t + (1-\beta) \underbrace{\left(\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t) + \lambda \mathbf{x}_t \right)}_{\text{gradient of loss with } L_2 \text{ penalty}} \tag{10}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t - \underbrace{2\eta\lambda \mathbf{x}_t}_{\text{weight decay}} \tag{11}$$

AdamW is widely used in training STOA NNs from scratch or fine-tuning on downstream tasks.

[5] *Loshchilov et al.* Decoupled weight decay regularization. ICLR 2018 (Google Scholar 4800+)

Table of Contents

Table of Contents

Lookahead Optimizer: k steps forward, 1 step back

Different from the ideas e.g.,

Lookahead Optimizer: k steps forward, 1 step back

Different from the ideas e.g.,

- **Adaptive element-wise learning rate**, e.g., AdaGrad and Adam

Lookahead Optimizer: k steps forward, 1 step back

Different from the ideas e.g.,

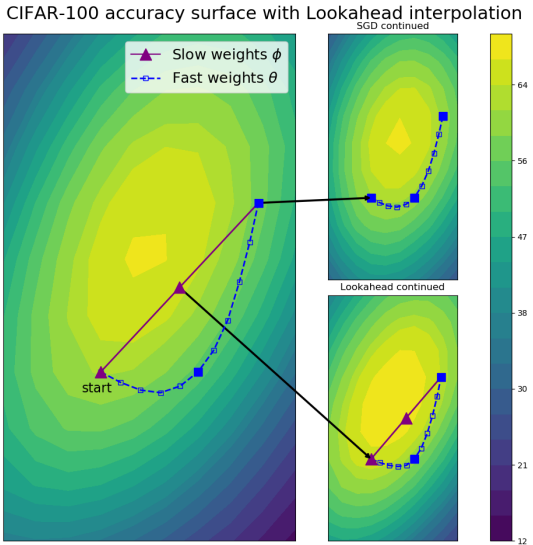
- **Adaptive element-wise learning rate**, e.g., AdaGrad and Adam
- **Accelerated optimization**, e.g., Heavy-ball momentum and Nesterov momentum.

Lookahead Optimizer: k steps forward, 1 step back

Different from the ideas e.g.,

- **Adaptive element-wise learning rate**, e.g., AdaGrad and Adam
- **Accelerated optimization**, e.g., Heavy-ball momentum and Nesterov momentum.

Lookahead Optimizer: k steps forward, 1 step back



A natural way to explore and exploit the landscape!

Algorithm 1: SGD

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = \boldsymbol{v}_k^{(t)} = (1-1)\boldsymbol{\theta}_{t-1} + 1 * \boldsymbol{v}_k^{(t)}$ $(\alpha = 1)$

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

Inner-loop optimization

outer-loop optimization

Algorithm 2: Lookahead

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = (1-\alpha)\boldsymbol{\theta}_{t-1} + \alpha \boldsymbol{v}_k^{(t)}$.

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

Inner-loop optimization

outer-loop optimization

[6] *Zhang et al.* Lookahead optimizer: k steps forward, 1 step back. NeurIPS 2019.

Algorithm 1: SGD

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = \boldsymbol{v}_k^{(t)} = (1-1)\boldsymbol{\theta}_{t-1} + 1 * \boldsymbol{v}_k^{(t)}$ $(\alpha = 1)$

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

Inner-loop optimization

Algorithm 2: Lookahead

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = (1-\alpha)\boldsymbol{\theta}_{t-1} + \alpha \boldsymbol{v}_k^{(t)}$.

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

Inner-loop optimization

- inner-loop optimization: k steps forward in SGD & LA
- outer-loop optimization: 1 step back in LA, while no step back in SGD

[6] *Zhang et al.* Lookahead optimizer: k steps forward, 1 step back. NeurIPS 2019.

Algorithm 1: SGD

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = \boldsymbol{v}_k^{(t)} = (1-1)\boldsymbol{\theta}_{t-1} + 1 * \boldsymbol{v}_k^{(t)}$ $(\alpha = 1)$

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

outer-loop optimization

Algorithm 2: Lookahead

Input : Objective $F_{\mathcal{S}}(\boldsymbol{\theta})$, dataset $\mathcal{S}$, inner-loop optimizer $\mathcal{A}$, inner-loop step number k and learning rate $\{\{\eta_\tau^{(t)}\}$, outer-loop learning rate $\alpha \in (0, 1)$.

for $t = 1, 2, ..., T$ **do**

  $\boldsymbol{v}_0^{(t)} = \boldsymbol{\theta}_{t-1}$;

  for $\tau = 1, 2, ..., k$ **do**

    $\boldsymbol{v}_\tau^{(t)} = \mathcal{A}(F_{\mathcal{S}}(\boldsymbol{\theta}), \boldsymbol{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \boldsymbol{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \boldsymbol{g}_{\tau-1}^{(t)}$

  end

  $\boldsymbol{\theta}_t = (1-\alpha)\boldsymbol{\theta}_{t-1} + \alpha \boldsymbol{v}_k^{(t)}$.

end

Output : $\boldsymbol{\theta}_{\mathcal{A},\mathcal{S}} = \boldsymbol{\theta}_T$

outer-loop optimization

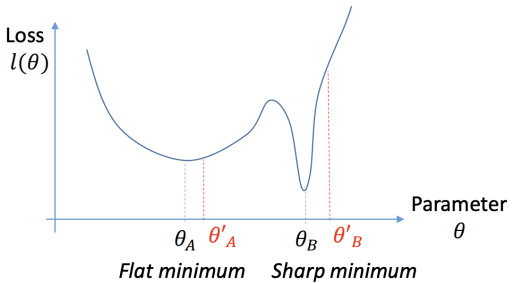
- inner-loop optimization: k steps forward in SGD & LA
- outer-loop optimization: 1 step back in LA, while no step back in SGD

[6] *Zhang et al.* Lookahead optimizer: k steps forward, 1 step back. NeurIPS 2019.

Table of Contents

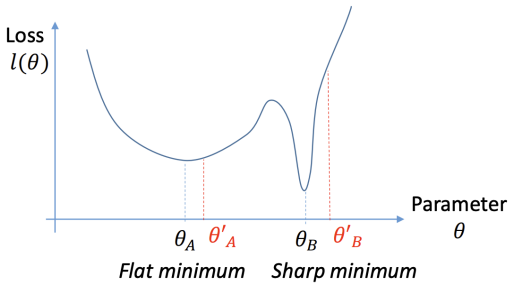
Flat Minima in Deep Learning

In many cases,



Flat Minima in Deep Learning

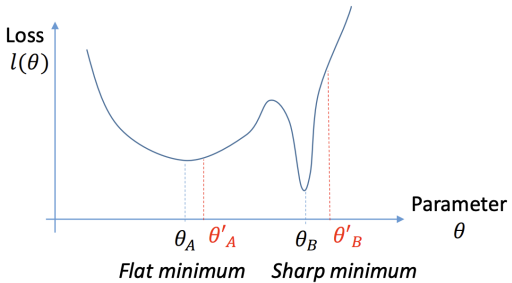
In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$



Flat Minima in Deep Learning

In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

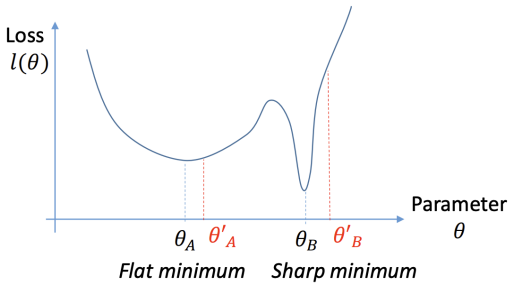
Highly non-convex (many local minima)!



Flat Minima in Deep Learning

In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

Highly non-convex (many local minima)!

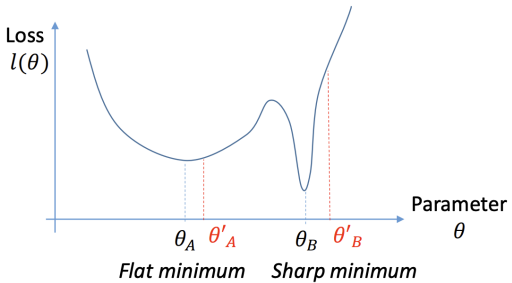


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?

Flat Minima in Deep Learning

In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

Highly non-convex (many local minima)!

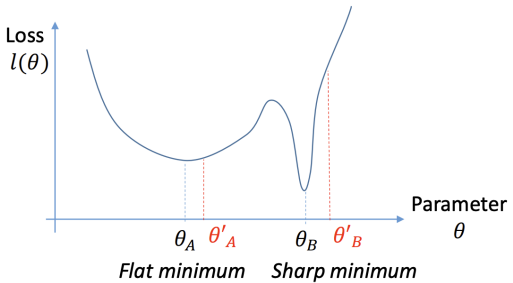


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?
- A) We prefer θ_A to $\boldsymbol{\theta}_B$ even though $\ell(\boldsymbol{\theta}_A) > \ell(\boldsymbol{\theta}_B)$

Flat Minima in Deep Learning

In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

Highly non-convex (many local minima)!

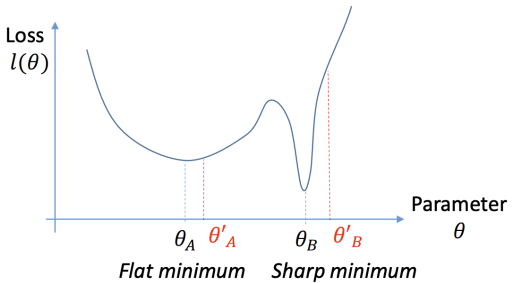


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?
- A) We prefer θ_A to $\boldsymbol{\theta}_B$ even though $\ell(\boldsymbol{\theta}_A) > \ell(\boldsymbol{\theta}_B)$
  - Why? Because θ_A is more robust.

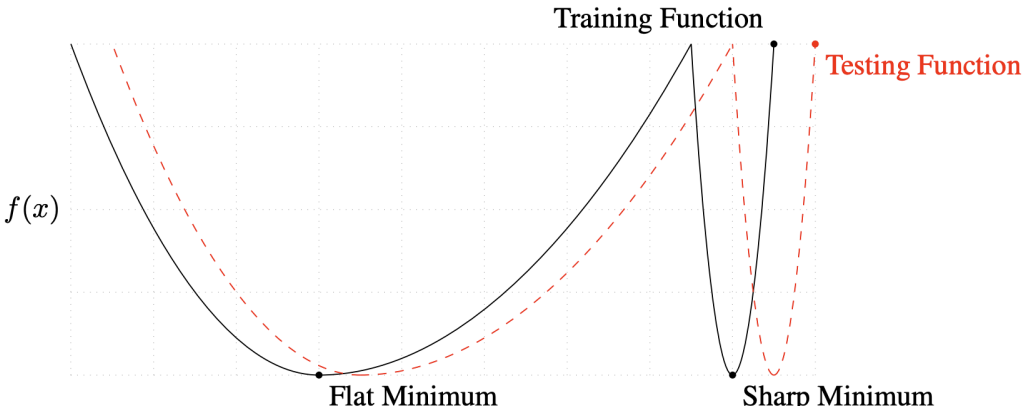
Flat Minima in Deep Learning

In many cases,
DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

Highly non-convex (many local minima)!



- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?
- A) We prefer θ_A to $\boldsymbol{\theta}_B$ even though $\ell(\boldsymbol{\theta}_A) > \ell(\boldsymbol{\theta}_B)$
  - Why? Because θ_A is more robust.
  - Imagine some perturbation: $\theta_A \rightarrow \boldsymbol{\theta}'_A, \boldsymbol{\theta}_B \rightarrow \boldsymbol{\theta}'_B \Rightarrow \ell(\boldsymbol{\theta}'_A) \ll \ell(\boldsymbol{\theta}'_B)$.


Training Function
Testing Function
$f(x)$
Flat Minimum
Sharp Minimum

Let's seek for a Flat Minimum

$$\begin{aligned} \text{Flat Minima} &= \text{Robust models} && (12)\\ &= \text{Resilient to data noise or model corruption} && (13)\\ &= \text{(often encountered in AI applications)} && (14) \end{aligned}$$

Let's seek for a Flat Minimum

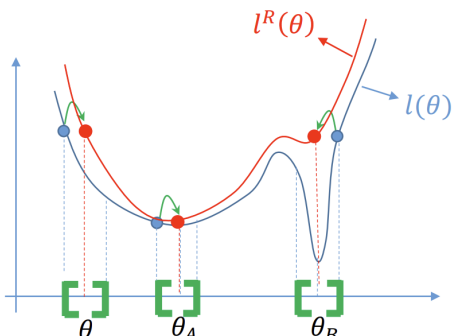
$$\begin{aligned} \text{Flat Minima} &= \text{Robust models} && (12)\\ &= \text{Resilient to data noise or model corruption} && (13)\\ &= (\text{often encountered in AI applications}) && (14) \end{aligned}$$

But, how?

Sharpness-Aware Minimization (SAM)

Idea of SAM:

Define a robust loss $\ell^R(\boldsymbol{\theta})$ as worst-case loss within a neighborhood of $\boldsymbol{\theta}$.



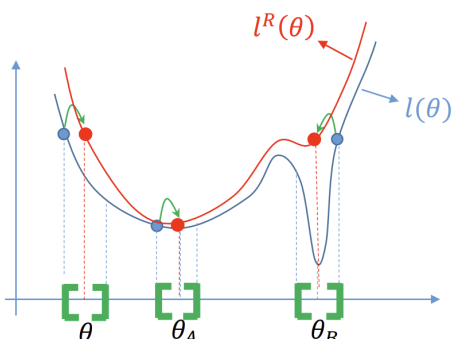
[2] *Foret et al.* Sharpness-aware Minimization for Efficiently Improving Generalization. ICLR 2021.

Sharpness-Aware Minimization (SAM)

Idea of SAM:

Define a robust loss $\ell^R(\boldsymbol{\theta})$ as worst-case loss within a neighborhood of $\boldsymbol{\theta}$.

$$\ell^R(\boldsymbol{\theta}) = \max_{\boldsymbol{\epsilon} \in N_{\boldsymbol{\theta}}} \ell(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \qquad (15)$$



[2] *Foret et al.* Sharpness-aware Minimization for Efficiently Improving Generalization. ICLR 2021.

Intuition behind SAM

- **Goal:** Find the local minima θ that are generalizable to test samples

Intuition behind SAM

- **Goal:** Find the local minima θ that are generalizable to test samples
- **Theorem (Flatness-based generalization bounds):**

Intuition behind SAM

- **Goal:** Find the local minima θ that are generalizable to test samples
- **Theorem (Flatness-based generalization bounds):**

Theorem 2

With high probability over $\mathcal{S}$, the flatness-based bound says:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \leq \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) + \underbrace{\left[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}+\boldsymbol{\epsilon}) - \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right]}_{\textit{flatness measure}} + h(\|\boldsymbol{\theta}\|_2^2/\rho^2)\,, \tag{16}$$

Intuition behind SAM

- **Goal:** Find the local minima $\boldsymbol{\theta}$ that are generalizable to test samples
- **Theorem (Flatness-based generalization bounds):**

Theorem 2

With high probability over $\mathcal{S}$, the flatness-based bound says:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \leq \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) + \underbrace{\left[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}+\boldsymbol{\epsilon}) - \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right]}_{\textit{flatness measure}} + h(\|\boldsymbol{\theta}\|_2^2/\rho^2)\,, \tag{16}$$

where $h(\|\boldsymbol{\theta}\|_2^2/\rho^2)$ is a strictly increasing function of $\boldsymbol{\theta}$. It decreases as the number of samples $n = |\mathcal{S}|$ increases.

Sharpness-Aware Minimization (SAM)

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) \leq \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) + \underbrace{\left[\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}+\boldsymbol{\epsilon}) - \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right]}_{\text{flatness measure}} + h(\|\boldsymbol{\theta}\|_2^2/\rho^2) \tag{17}$$

$$= \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}+\boldsymbol{\epsilon}) + h(\|\boldsymbol{\theta}\|_2^2/\rho^2) \tag{18}$$

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

- the optimal $\hat{\epsilon}$ is given by (linear approximation through first-order Taylor expansion)

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

- the optimal $\hat{\boldsymbol{\epsilon}}$ is given by (linear approximation through first-order Taylor expansion)

$$\hat{\boldsymbol{\epsilon}} = \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) = \rho \cdot \text{sign}\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right) \frac{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|^{q-1}}{\left(\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\|_q^q\right)^{1/p}}\,, \tag{18}$$

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

- the optimal $\hat{\boldsymbol{\epsilon}}$ is given by (linear approximation through first-order Taylor expansion)

$$\hat{\boldsymbol{\epsilon}} = \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) = \rho \cdot \text{sign}\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right) \frac{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|^{q-1}}{\left(\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\|_q^q\right)^{1/p}}\,, \tag{18}$$

where $1/p + 1/q = 1$

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

- the optimal $\hat{\boldsymbol{\epsilon}}$ is given by (linear approximation through first-order Taylor expansion)

$$\hat{\boldsymbol{\epsilon}} = \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) = \rho \cdot \text{sign}\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right) \frac{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|^{q-1}}{\left(\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\|_q^q\right)^{1/p}}\,, \tag{18}$$

where $1/p + 1/q = 1$ and the solution to a classical dual norm problem can solve this approximation.

Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

$$\boldsymbol{\theta}^\star := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \boldsymbol{\epsilon})\,. \tag{17}$$

- the optimal $\hat{\boldsymbol{\epsilon}}$ is given by (linear approximation through first-order Taylor expansion)

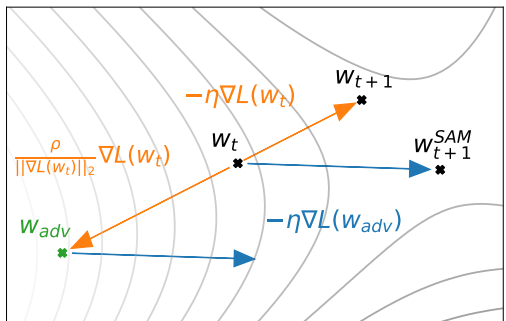
$$\hat{\boldsymbol{\epsilon}} = \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) = \rho \cdot \text{sign}\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right) \frac{|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|^{q-1}}{\left(\|\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\|_q^q\right)^{1/p}}\,, \tag{18}$$

  where $1/p + 1/q = 1$ and the solution to a classical dual norm problem can solve this approximation.
- substituting $\hat{\boldsymbol{\epsilon}}$ gives a gradient estimator

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) := \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta} + \hat{\boldsymbol{\epsilon}}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta} + \hat{\boldsymbol{\epsilon}}} \tag{19}$$

- The gradient estimator of SAM is given by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\epsilon}} \qquad (20)$$

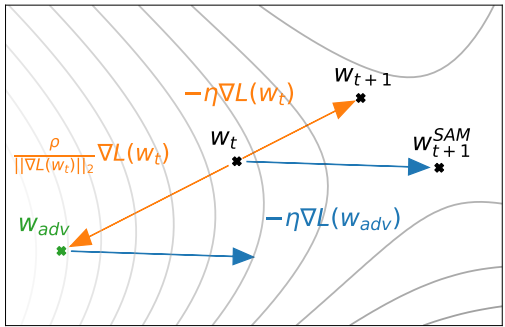


From original SAM paper.

- The gradient estimator of SAM is given by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}} \qquad (20)$$

- Recall that SAM requires *2-step* gradient descent (thus, twice slow)

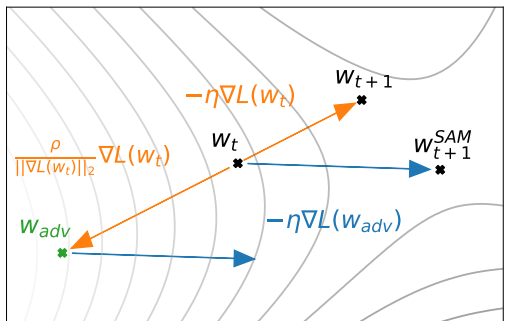


From original SAM paper.

- The gradient estimator of SAM is given by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\epsilon}} \qquad (20)$$

- Recall that SAM requires *2-step* gradient descent (thus, twice slow)
  - 1st for computing $\hat{\epsilon}$ using $\nabla_{\theta} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})$

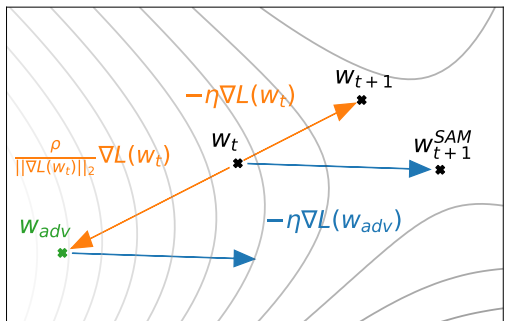


From original SAM paper.

- The gradient estimator of SAM is given by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}} \qquad (20)$$

- Recall that SAM requires *2-step* gradient descent (thus, twice slow)
  - 1st for computing $\hat{\epsilon}$ using $\nabla_{\theta} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})$
  - 2nd for computing $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}}$

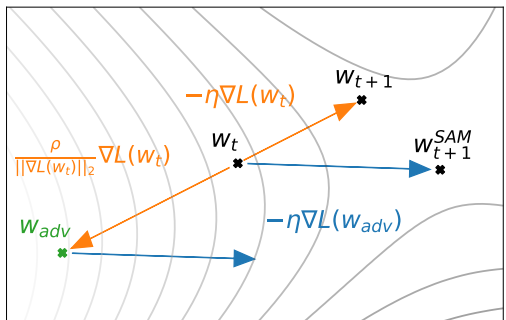


From original SAM paper.

- The gradient estimator of SAM is given by:

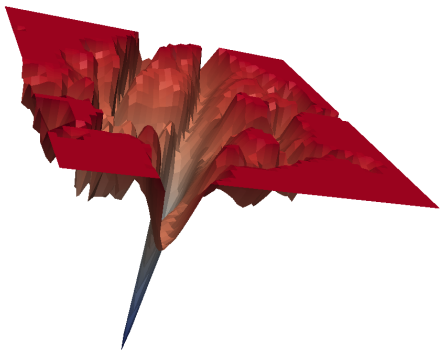
$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}} \qquad (20)$$

- Recall that SAM requires *2-step* gradient descent (thus, twice slow)
  - 1st for computing $\hat{\epsilon}$ using $\nabla_{\theta} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})$
  - 2nd for computing $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}}$
- Set $p = 2$-norm and neighborhood-size $\rho = 0.05$ as a default setup.

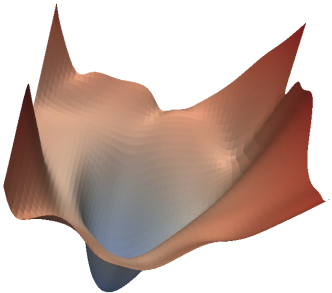


From original SAM paper.

Verification of the flatness



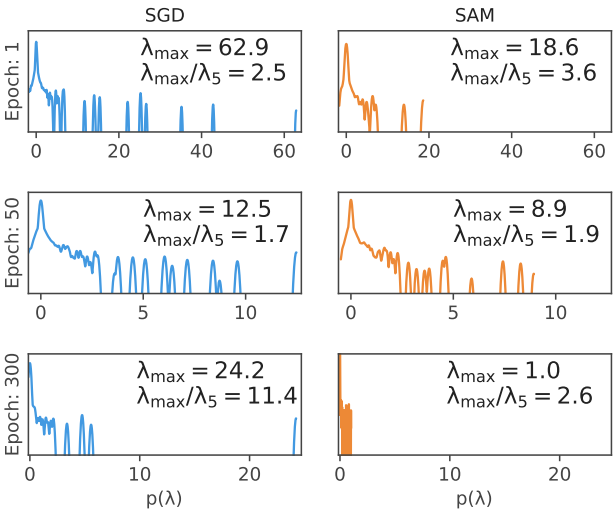
(a) ERM.



(b) SAM.

Loss surface visualization.

Verification of the flatness



Hessian spectra.

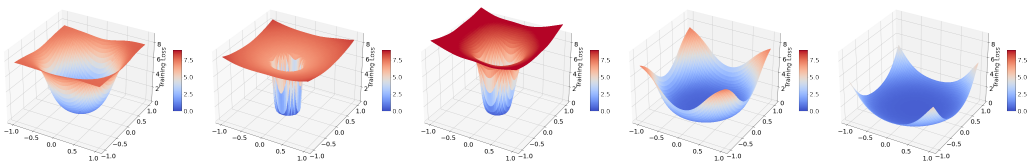
Results (i.e., SAM $>$ ERM)

- SAM *consistently improves* classification tasks, particularly with label noises

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

Results on ViT (and MLP-Mixer)



(c) ResNet (d) ViT (e) Mixer (f) ViT-SAM (g) Mixer-SAM

Figure: **Cross-entropy loss landscapes of ResNet-152, ViT-B/16, and Mixer-B/16.** ViT and MLP-Mixer converge to sharper regions than ResNet when trained on ImageNet with the basic Inception-style preprocessing. SAM significantly smooths the landscapes.

Results on ViT (and MLP-Mixer)

Table: Number of parameters, Hessian dominate eigenvalue λ_{max}, training error at convergence L_{train}, average flatness $L_{train}^{\mathcal{N}}$, accuracy on ImageNet, and accuracy/robustness on ImageNet-C. ViT and MLP-Mixer suffer divergent κ and converge at sharp regions; SAM rescues that and leads to better generalization.

	ResNet-152	ResNet-152-SAM	ViT-B/16	ViT-B/16-SAM	Mixer-B/16	Mixer-B/16-SAM
#Params	60M		87M		59M	
Hessian λ_{max}	179.8	**42.0**	738.8	**20.9**	1644.4	**22.5**
L_{train}	**0.86**	0.90	**0.65**	0.82	**0.45**	0.97
$L_{train}^{\mathcal{N}}$ *	2.39	**2.16**	6.66	**0.96**	7.78	**1.01**
ImageNet (%)	78.5	**79.3**	74.6	**79.9**	66.4	**77.4**
ImageNet-C (%)	50.0	**52.2**	46.6	**56.5**	33.8	**48.8**

Thanks & Question Time!

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[2] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[5] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[6] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32, 2019.