Optimization for Deep Learning

Tao LIN

Learning and INference Systems (LINs) Lab, Westlake University

March 3, 2024

WESTLAKE UNIVERSITY | SCHOOL OF ENGINEERING

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

$$h_{\mathbf{x}}(\mathbf{d}) \to y \tag{1}$$

$h_{\mathbf{x}}\Big($  $\Big) \to$ cat

Background

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

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

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

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

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

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

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 $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$

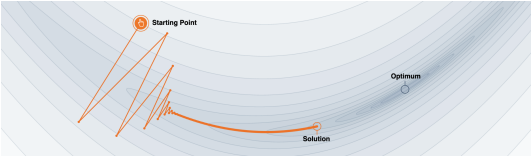
Background

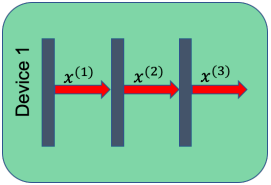
Finite-sum empirical risk minimization problem:

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

- 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

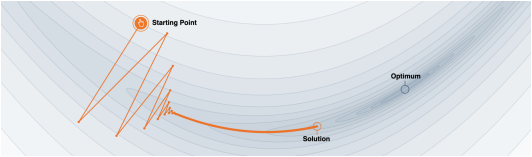
Finite-sum empirical risk minimization problem:

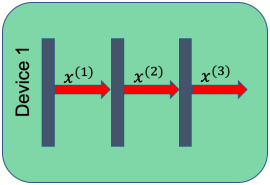
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \left(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \right) \right\} \tag{3}$$

- 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

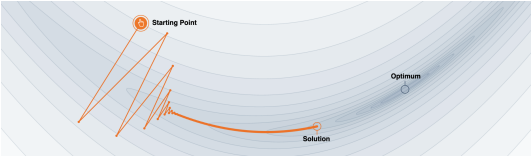
Finite-sum empirical risk minimization problem:

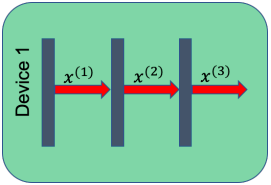
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \left(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \right) \right\} \tag{3}$$

- 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

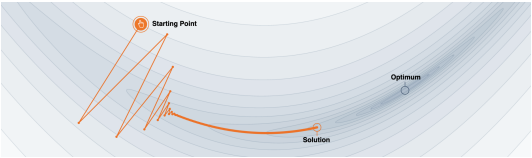
Finite-sum empirical risk minimization problem:

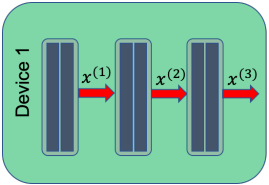
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \left(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \right) \right\} \tag{3}$$

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

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{B}\sum_{i\in\mathcal{B}} \eta \nabla f_i(\mathbf{x})$$

(Using *mini-batching*)





Background

Finite-sum empirical risk minimization problem:

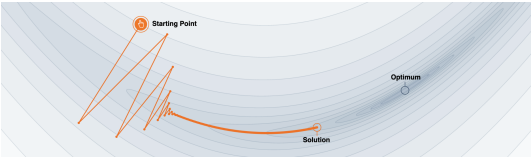
$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \left(f_i(\mathbf{x}) := F(\mathbf{x}, \boldsymbol{\xi}_i) \right) \right\} \tag{3}$$

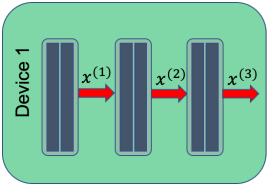
- The loss function of i-th data $\boldsymbol{\xi}_i := (\mathbf{d}_i, y_i)$
- Baseline method: Stochastic Gradient Descent (SGD)

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{B}\sum_{i\in\mathcal{B}} \eta \nabla f_i(\mathbf{x})$$

(Using *mini-batching*)

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





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 $i = 1, \ldots, N$.

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 $i = 1, \ldots, N$.

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

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

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

$\longleftrightarrow$

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

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

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

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

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

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

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

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

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

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

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* $\mathbb{E}\,\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \sigma^2$*.*

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

- L*-smoothness*

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

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*

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* $\mathbb{E}\,\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \leq \sigma^2$.

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*
- 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*

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

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

- 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*

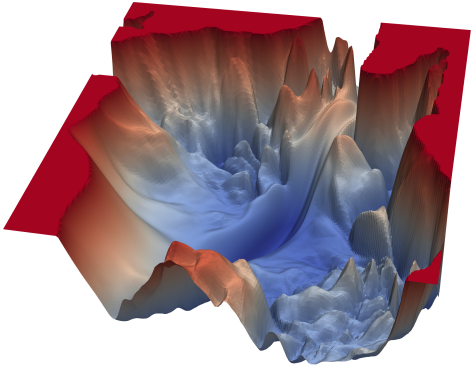
$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T} + \frac{\sigma}{\sqrt{B}}\sqrt{\frac{L\,(f(\mathbf{x}_0)-f^\star)}{T}}\right)$$

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

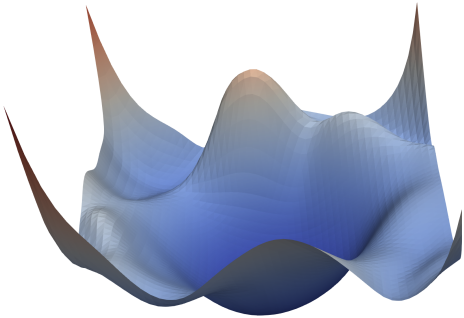
- When iterations $T \to \infty$, it holds that $\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \to 0$
- $\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \to 0$ implies the sequence converges to a stationary solution

Table of Contents

Issues of SGD—from the perspective of loss landscape



(a) ResNet w/o skip connections.

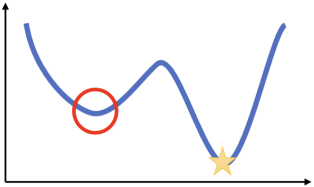


(b) ResNet w/ skip connections.

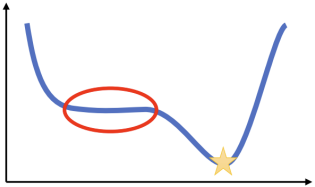
Figure: The surfaces of ResNet-56 w/ and w/o skip connections [12].

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

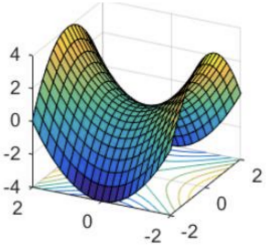
Issues of SGD—from the perspective of loss landscape



the local optimum



the plateau

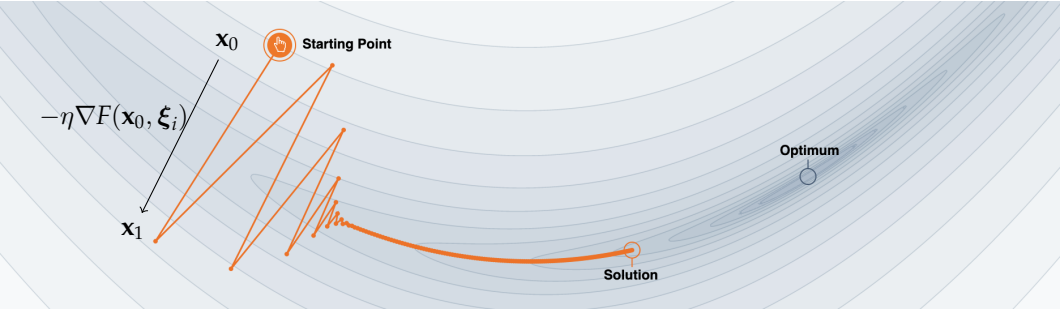


the saddle point

Challenging optimization loss landscape!

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

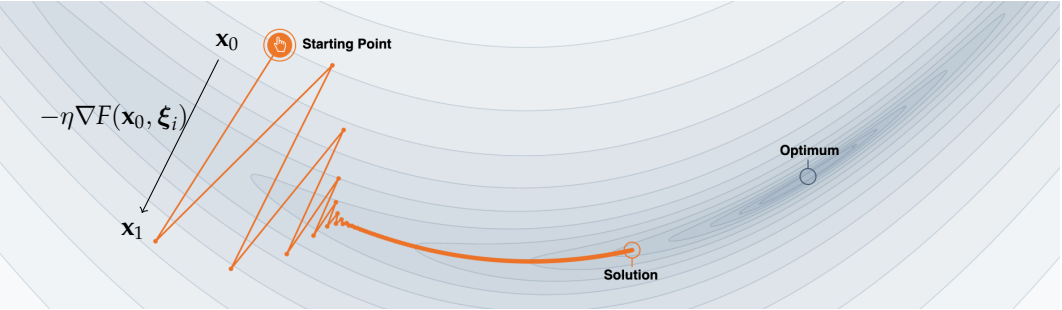
Issues of SGD—from the perspective of loss landscape



- **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/`

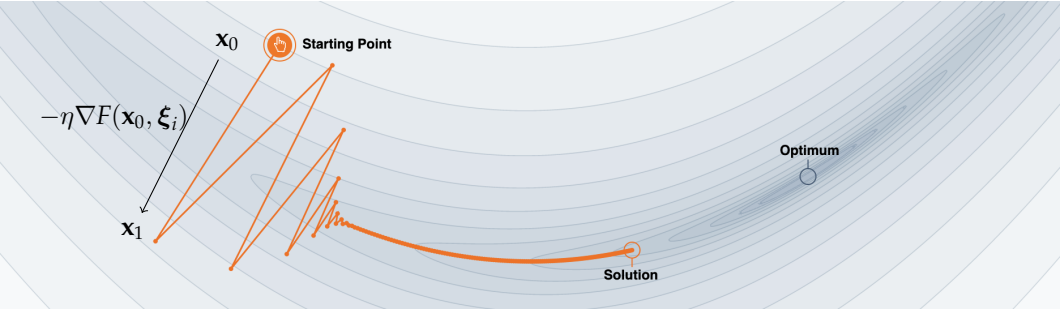
Issues of SGD—from the perspective of loss landscape



- **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/`

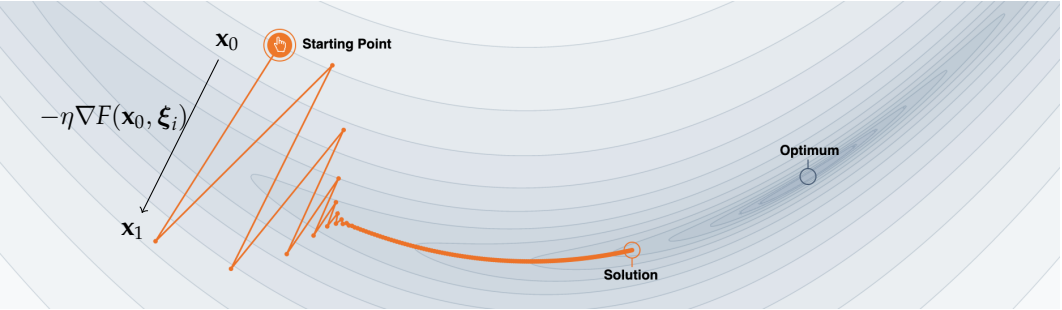
Issues of SGD—from the perspective of loss landscape



- **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/`

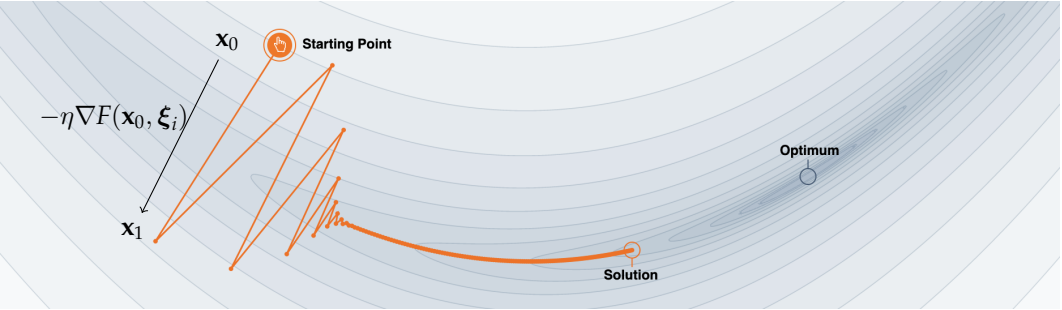
Issues of SGD—from the perspective of loss landscape



- **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.

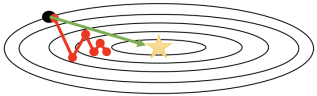
Improvement directions: leveraging the curvature information

Can we find a better descent direction in the loss landscape?

Yes! **By leveraging the curvature information** through Newton's method.

Taylor expansion:

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



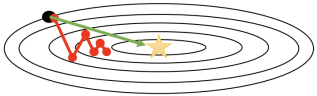
Improvement directions: leveraging the curvature information

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:

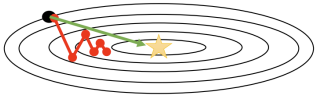
$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \underbrace{\nabla_{\mathbf{x}} f(\mathbf{x}_0)}_{\text{gradient}}(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top \underbrace{\nabla_{\mathbf{x}}^2 f(\mathbf{x}_0)}_{\text{Hessian}}(\mathbf{x} - \mathbf{x}_0) \tag{8}$$

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

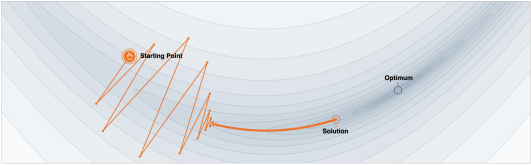
Q: Why is Newton's method not a viable way to improve neural network optimization?

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.

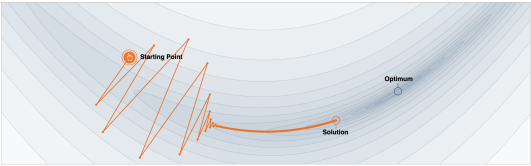
[1] if using naive approach, though fancy methods can be much faster if they avoid forming the Hessian explicitly.

Momentum method

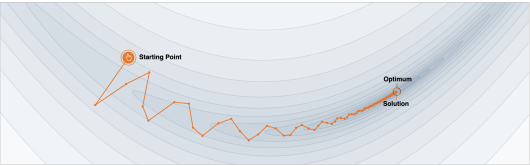


w/o momentum

Momentum method

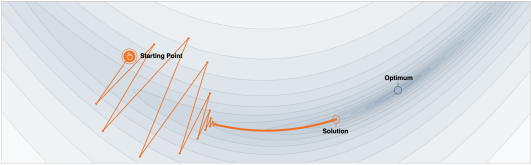


w/o momentum

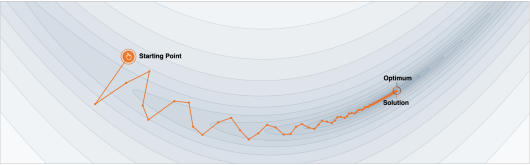


w/ momentum

Momentum method



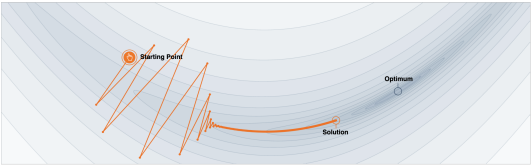
w/o momentum



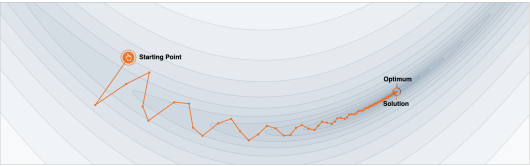
w/ momentum

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

Momentum method



w/o momentum

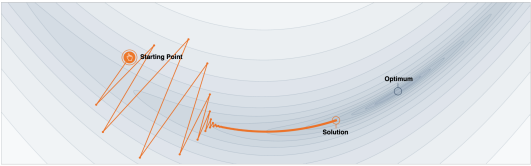


w/ momentum

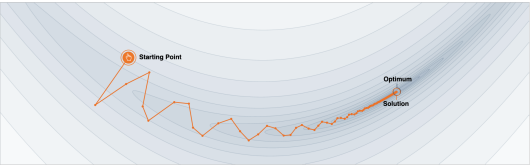
Intuition: averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions

Momentum method



w/o momentum

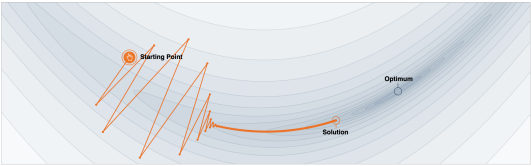


w/ momentum

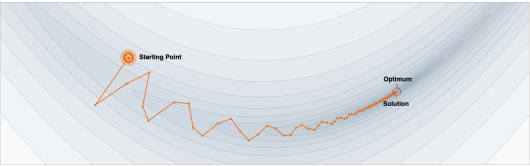
Intuition: averaging together successive gradients yield a much better direction!

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

Momentum method



w/o momentum



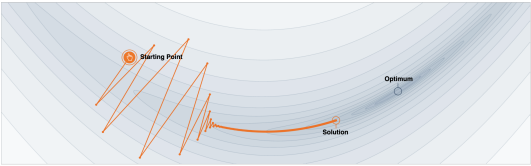
w/ momentum

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

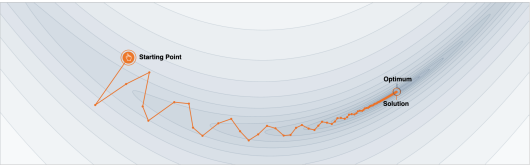
- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

- if successive gradient step point in similar directions

Momentum method



w/o momentum



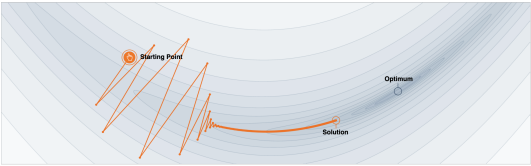
w/ momentum

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

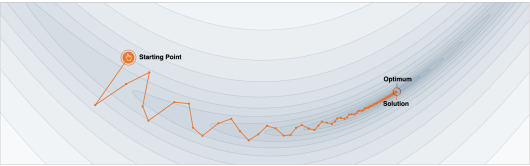
- if successive gradient step point in different directions
  $\rightarrow$ we should cancel off the directions that disagree

- 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 [2] (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.

[2] *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 [2] (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):

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

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

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

[2] *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 [2] (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):

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

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

Remarks:

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

[2] *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 [2] (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):

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

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

Remarks:

- AdaGrad has some appealing guarantees for convex problems.

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

[2] *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 [2] (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):

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

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

Remarks:

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

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

[2] *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 [2] (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):

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

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

Remarks:

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

[2] *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 [2] (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:

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

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

Adam: combining momentum and RMSProp

Idea:

[7] *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

[7] *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 $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

[7] *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 $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

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

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

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

[7] *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

[7] *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 $\frac{\eta}{\sqrt{\mathbf{v}_t+\epsilon}}\mathbf{m}_t$.

[7] *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 $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

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

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

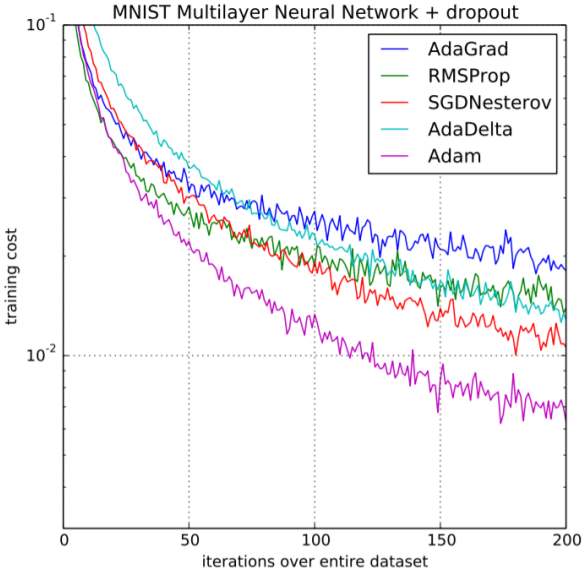
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \frac{\mathbf{m}_t}{\sqrt{\mathbf{v}_t + \epsilon}} = \mathbf{x}_t - \underbrace{\frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}}}_{\text{element-wise stepsize}} \mathbf{m}_t \qquad \text{(update step)}$$

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).

[7] *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:

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

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

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

Weight decay in SGD and Adam: why AdamW matters

On the discrepancy between L_2 regularization and weight decay:

[16] *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).

[16] *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.

[16] *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.

[16] *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)

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

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

[16] *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)

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

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

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

[16] *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.,

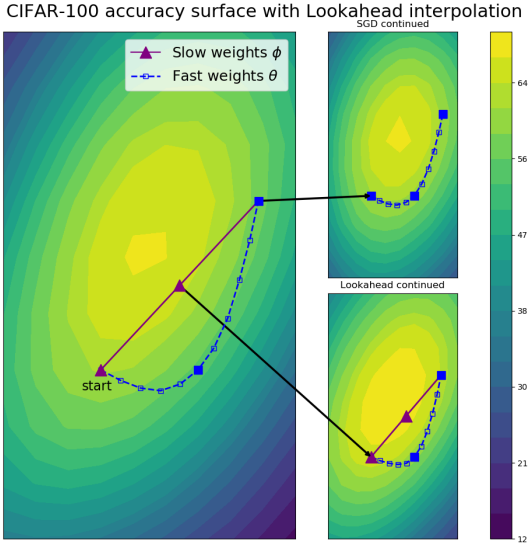
- **Adaptive element-wise learning rate**, e.g., AdaGrad and Adam
- **Accelerated optimization**, e.g., Heavy-ball momentum and Nesterov momentum.

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

[25] *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**

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

  Inner-loop optimization

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

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**

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

  Inner-loop optimization

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

[25] *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

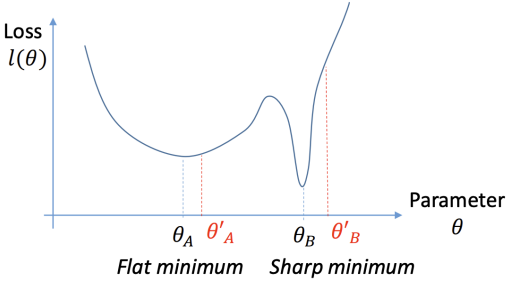
- inner-loop optimization: k steps forward in SGD & LA
- outer-loop optimization: 1 step back in LA, while no step back in SGD

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

Table of Contents

Flat Minima in Deep Learning

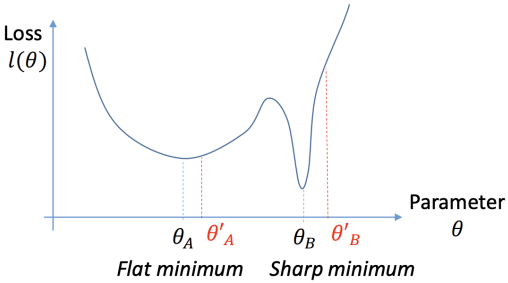
In many cases,



Flat Minima in Deep Learning

In many cases,

DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

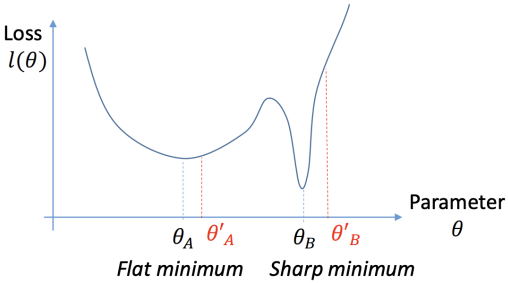


Flat Minima in Deep Learning

In many cases,

DL $\rightarrow$ minimizing a loss function $\ell(\boldsymbol{\theta})$

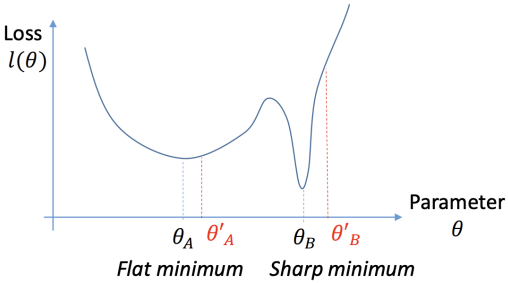
Highly non-convex (many local minima)!



Flat Minima in Deep Learning

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

Highly non-convex (many local minima)!

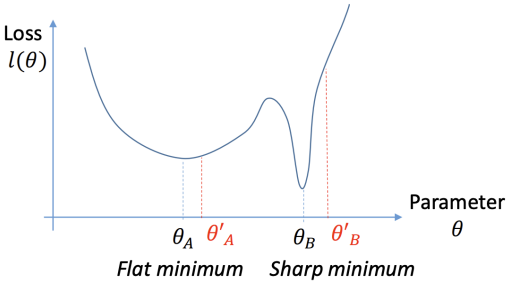


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?

Flat Minima in Deep Learning

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

Highly non-convex (many local minima)!

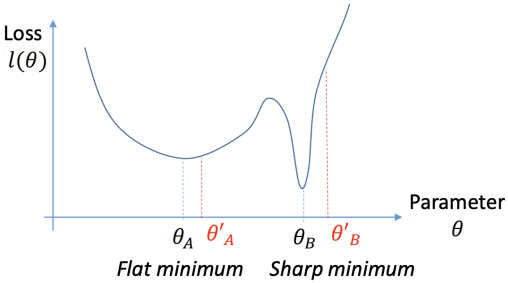


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?
- A) We prefer θ_A to $\boldsymbol{\theta}_B$ even though $\ell(\boldsymbol{\theta}_A) > \ell(\boldsymbol{\theta}_B)$

Flat Minima in Deep Learning

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

Highly non-convex (many local minima)!

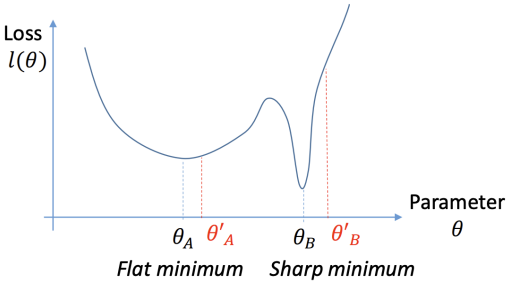


- Q) Which is better, $\boldsymbol{\theta}_A$ or $\boldsymbol{\theta}_B$?
- A) We prefer θ_A to $\boldsymbol{\theta}_B$ even though $\ell(\boldsymbol{\theta}_A) > \ell(\boldsymbol{\theta}_B)$
  - Why? Because θ_A is more robust.

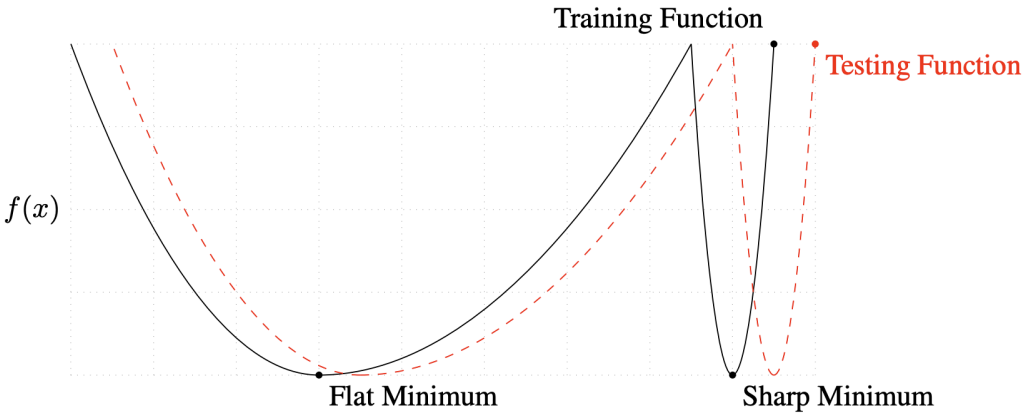
Flat Minima in Deep Learning

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

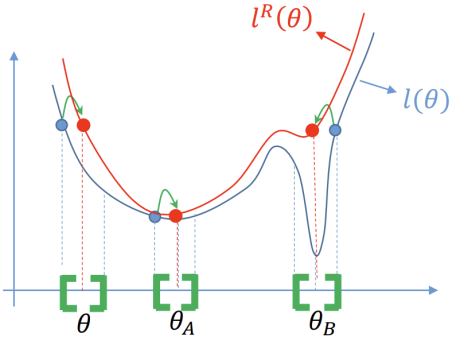
$$\begin{aligned} \text{Flat Minima} &= \text{Robust models} && (12)\\ &= \text{Resilient to data noise or model corruption} && (13)\\ &= \text{(often encountered in AI applications)} && (14) \end{aligned}$$

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 $\boldsymbol{\theta}$.



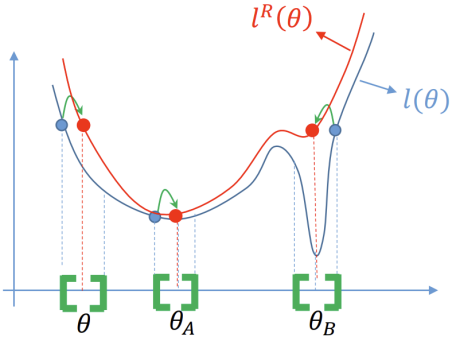
[3] *Foret et al.* Sharpness-aware Minimization for Efficiently Improving Generalization. ICLR 2021.

Sharpness-Aware Minimization (SAM)

Idea of SAM:

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

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



[3] *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 $\boldsymbol{\theta}$ that are generalizable to test samples
- **Theorem (Flatness-based generalization bounds):**

Theorem 2

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

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

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)

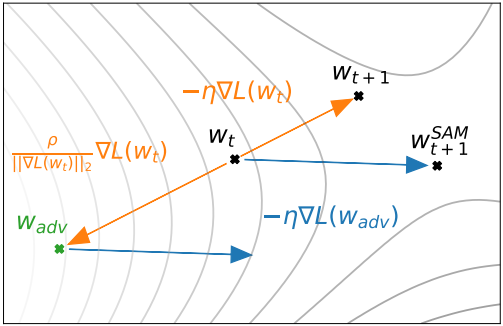
$$\hat{\boldsymbol{\epsilon}} = \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta}) = \rho \cdot \text{sign}\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})\right) \frac{|\nabla_{\boldsymbol{\theta}}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

  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:

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

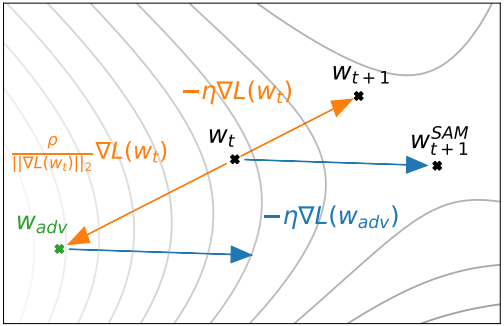


From original SAM paper.

- The gradient estimator of SAM is given by:

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

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

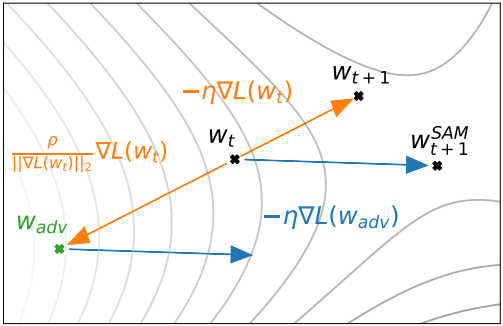


From original SAM paper.

- The gradient estimator of SAM is given by:

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

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

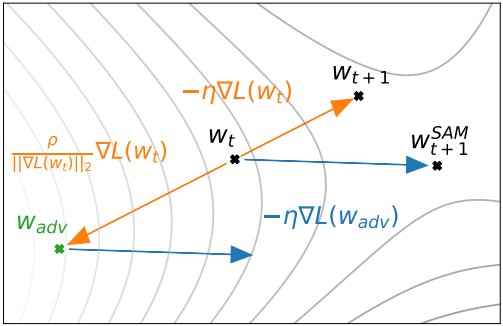


From original SAM paper.

- The gradient estimator of SAM is given by:

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

- 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 $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}}$

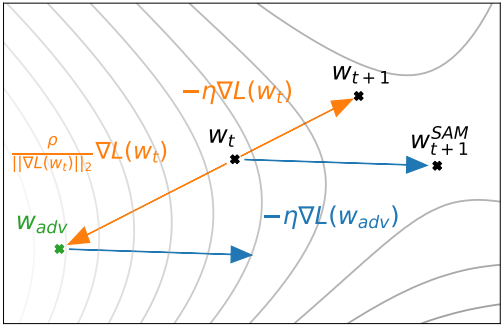


From original SAM paper.

- The gradient estimator of SAM is given by:

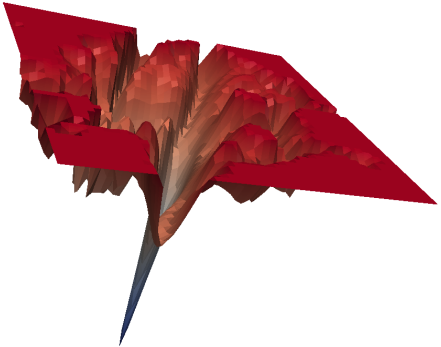
$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}^{\text{SAM}}(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{S}}(\boldsymbol{\theta})|_{\boldsymbol{\theta}+\hat{\boldsymbol{\epsilon}}} \tag{20}$$

- 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.

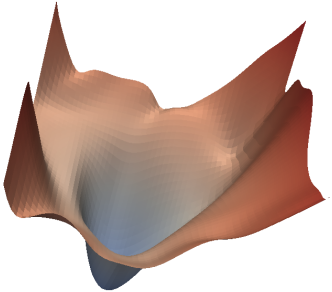


From original SAM paper.

Verification of the flatness



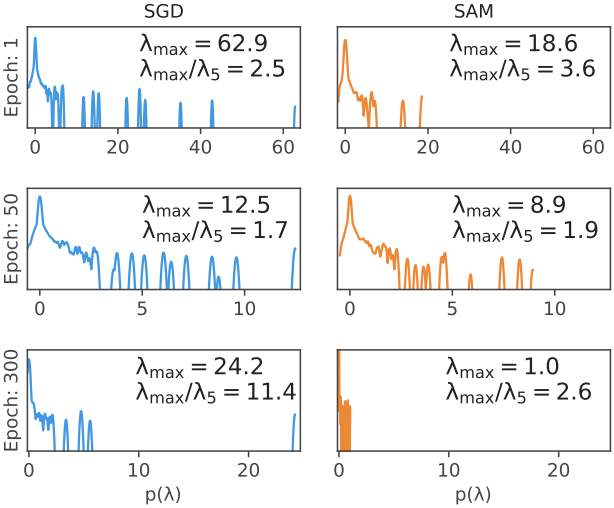
(a) ERM.



(b) SAM.

Loss surface visualization.

Verification of the flatness



Hessian spectra.

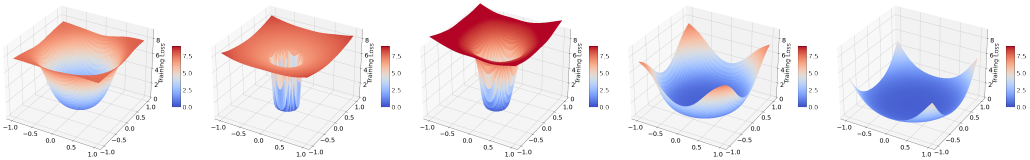
Results (i.e., SAM $>$ ERM)

- 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**

Table of Contents

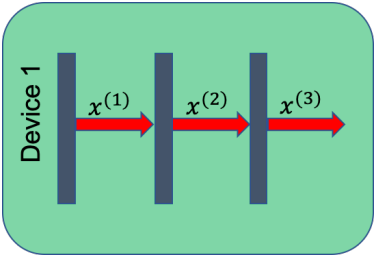
Table of Contents

Why Distributed Deep Learning Infrastructure Matters

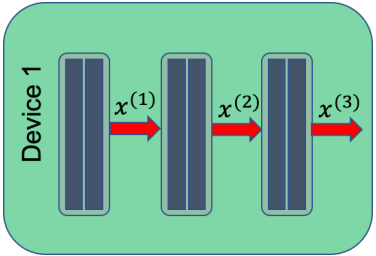
Recall that

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

To achieve an ϵ-accurate solution, i.e., $\frac{1}{T}\sum_{t=0}^{T}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \epsilon$, it requires $\mathcal{O}\left(\frac{\sigma^2}{B\epsilon^2}\right)$.



SGD



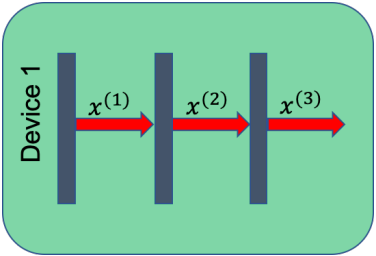
Mini-batch SGD

Why Distributed Deep Learning Infrastructure Matters

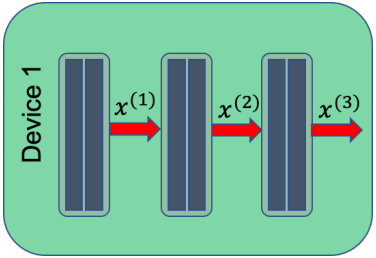
Recall that

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

To achieve an ϵ-accurate solution, i.e., $\frac{1}{T}\sum_{t=0}^{T}\mathbb{E}\left[\left\|\nabla f(\mathbf{x}^{(t)})\right\|^2\right] \leq \epsilon$, it requires $\mathcal{O}\left(\frac{\sigma^2}{B\epsilon^2}\right)$.



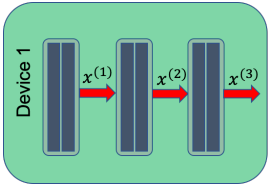
SGD



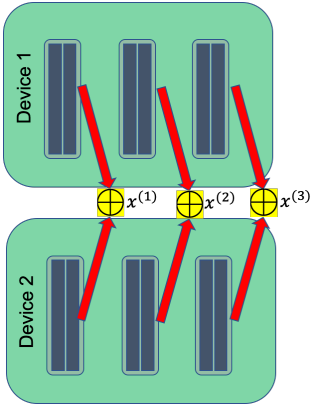
Mini-batch SGD

Increasing B may linearly reduce the required steps to reach a target performance.

Why Distributed Deep Learning Infrastructure Matters



Mini-batch SGD

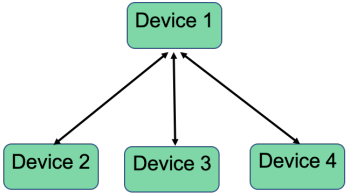


Data-parallelism

Increasing B may linearly reduce the required steps to reach a target performance.

$$B := \underbrace{B_{\text{loc}}}_{\text{normally fixed}} \quad \underbrace{n}_{\text{increasing}}$$

Various Types of Distributed Learning Infrastructure

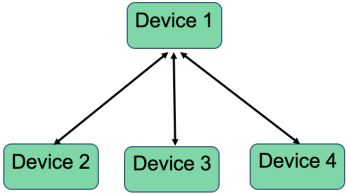


Star/Parameter server

$$\mathcal{O}\left((t_s + t_w m)n\right)$$

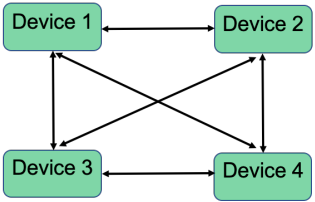
t_s is the latency, t_w is inverse bandwidth, m is the message size, and n is the number of nodes.

Various Types of Distributed Learning Infrastructure



Star/Parameter server

$$\mathcal{O}\left((t_s + t_w m)n\right)$$

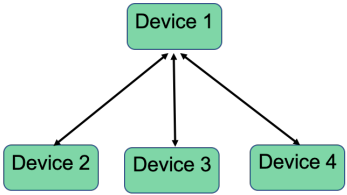


Complete

$$\mathcal{O}\left(t_s \log_2 n + t_w(n-1)m\right)$$

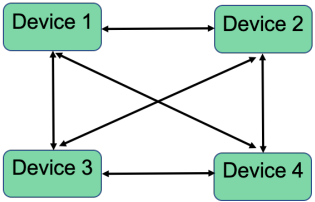
t_s is the latency, t_w is inverse bandwidth, m is the message size, and n is the number of nodes.

Various Types of Distributed Learning Infrastructure



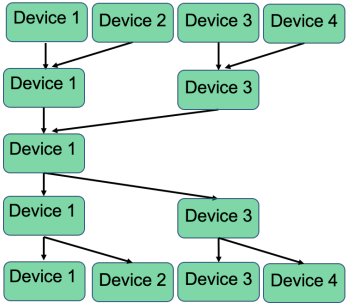
Star/Parameter server

$$\mathcal{O}\left((t_s + t_w m)n\right)$$



Complete

$$\mathcal{O}\left(t_s \log_2 n + t_w(n-1)m\right)$$

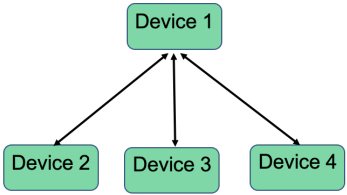


All-Reduce

$$\mathcal{O}\left((t_s + t_w m)\log_2 n\right)$$

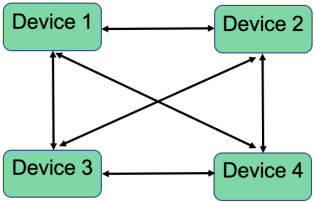
t_s is the latency, t_w is inverse bandwidth, m is the message size, and n is the number of nodes.

Various Types of Distributed Learning Infrastructure



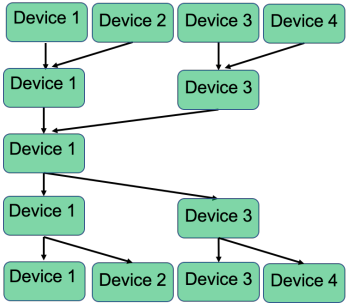
Star/Parameter server

$$\mathcal{O}\left((t_s + t_w m)n\right)$$



Complete

$$\mathcal{O}\left(t_s \log_2 n + t_w(n-1)m\right)$$



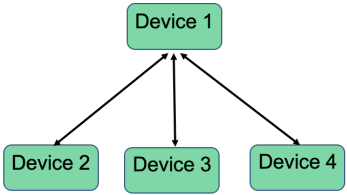
All-Reduce

$$\mathcal{O}\left((t_s + t_w m)\log_2 n\right)$$

t_s is the latency, t_w is inverse bandwidth, m is the message size, and n is the number of nodes.

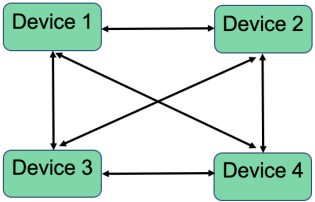
- *Parameter Server*'s bandwidth will be decreased by the number of nodes, and is sensitive to the central failures.

Various Types of Distributed Learning Infrastructure



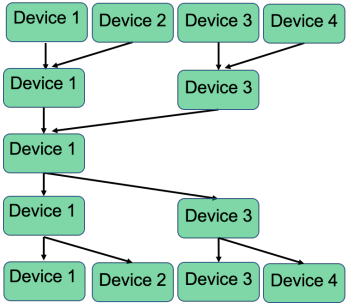
Star/Parameter server

$$\mathcal{O}\left((t_s + t_w m)n\right)$$



Complete

$$\mathcal{O}\left(t_s \log_2 n + t_w(n-1)m\right)$$



All-Reduce

$$\mathcal{O}\left((t_s + t_w m)\log_2 n\right)$$

t_s is the latency, t_w is inverse bandwidth, m is the message size, and n is the number of nodes.

- *Parameter Server*'s bandwidth will be decreased by the number of nodes, and is sensitive to the central failures.
- *All-Reduce* enables full bandwidth.

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

- The functions f_i represents the loss function on client/node i with local dataset $\mathcal{D}_i$.

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

- The functions f_i represents the loss function on client/node i with local dataset $\mathcal{D}_i$.
- Each local distribution $\mathcal{D}_i$ may be
  1. identical, e.g. data center case (achieved by shuffling across nodes)
  2. different, e.g. EdgeAI case (thus has data heterogeneity issue).

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

- The functions f_i represents the loss function on client/node i with local dataset $\mathcal{D}_i$.
- Each local distribution $\mathcal{D}_i$ may be
  1. identical, e.g. data center case (achieved by shuffling across nodes)
  2. different, e.g. EdgeAI case (thus has data heterogeneity issue).
- Each of these clients/nodes i performs 1 local update.

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

- The functions f_i represents the loss function on client/node i with local dataset $\mathcal{D}_i$.
- Each local distribution $\mathcal{D}_i$ may be
  1. identical, e.g. data center case (achieved by shuffling across nodes)
  2. different, e.g. EdgeAI case (thus has data heterogeneity issue).
- Each of these clients/nodes i performs 1 local update.
- The models are aggregated to form the new global model:

$$\mathbf{x}^{(t+1)} \leftarrow \tfrac{1}{n}\textstyle\sum_{i=1}^{n}\left(\mathbf{x}^{(t)} - \eta_l g_i(\mathbf{x}_i^{(t)})\right) , \qquad \text{(C-SGD)}$$

where η_l is the local step-size and $g_i(\mathbf{x}_i^{(t)}) := \nabla F_i(\mathbf{x}_i^{(t)};\xi_i^{(t)})$.

Extending Notations to Distributed Optimization (with a slight abuse of notation)

- To minimize a sum of stochastic functions, with only access to stochastic samples:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi\in\mathcal{D}_i}\left[F_i(\mathbf{x};\xi)\right]\right) \right\} .$$

- The functions f_i represents the loss function on client/node i with local dataset $\mathcal{D}_i$.
- Each local distribution $\mathcal{D}_i$ may be
  1. identical, e.g. data center case (achieved by shuffling across nodes)
  2. different, e.g. EdgeAI case (thus has data heterogeneity issue).
- Each of these clients/nodes i performs 1 local update.
- The models are aggregated to form the new global model:

$$\mathbf{x}^{(t+1)} \leftarrow \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \left(\mathbf{x}^{(t)} - \eta_l g_i(\mathbf{x}_i^{(t)})\right) , \qquad \text{(C-SGD)}$$

  where η_l is the local step-size and $g_i(\mathbf{x}_i^{(t)}) := \nabla F_i(\mathbf{x}_i^{(t)};\xi_i^{(t)})$.
- The convergence rate becomes (under the same assumptions as SGD)

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

Speed-up of Distributed Deep Learning (Data Parallelism)

Table: Distributed training ResNet-50 on ImageNet

	Metrics	4 nodes (32 GPUs)	8 nodes (64 GPUs)	16 nodes (128 GPUs)	32 nodes (256 GPUs)	Pattern
All-Reduce SGD	Accuracy	76.2%	76.4%	76.3%	76.2%	$\approx$
	Time	22.0 hrs.	14.0 hrs.	8.5 hrs.	5.1 hrs.	$\searrow$

[1] Assran et al. Stochastic gradient push for distributed deep learning. ICML 2019.

Speed-up of Distributed Deep Learning (Data Parallelism)

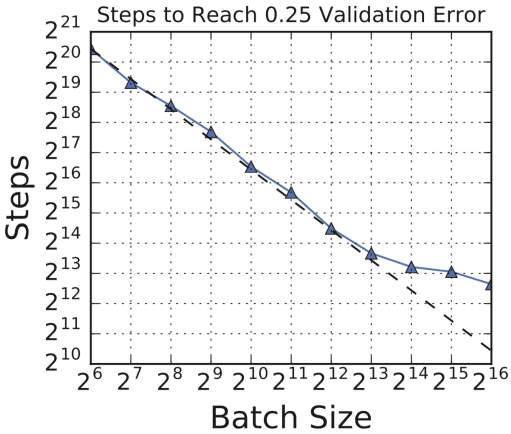
Table: Distributed training ResNet-50 on ImageNet

	Metrics	4 nodes (32 GPUs)	8 nodes (64 GPUs)	16 nodes (128 GPUs)	32 nodes (256 GPUs)	Pattern
All-Reduce SGD	Accuracy	76.2%	76.4%	76.3%	76.2%	$\approx$
	Time	22.0 hrs.	14.0 hrs.	8.5 hrs.	5.1 hrs.	$\searrow$

Scaling Deep Learning training with more GPUs!

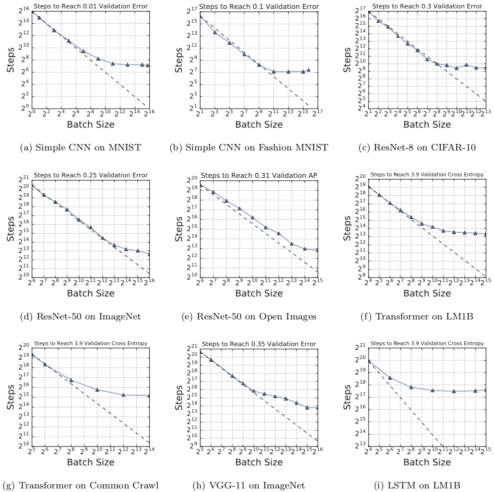
[1] Assran et al. Stochastic gradient push for distributed deep learning. ICML 2019.

No Free Lunch in Distributed Deep Learning



- **Limitation 1**: Diminishing returns of data parallelism with large mini-batch sizes.

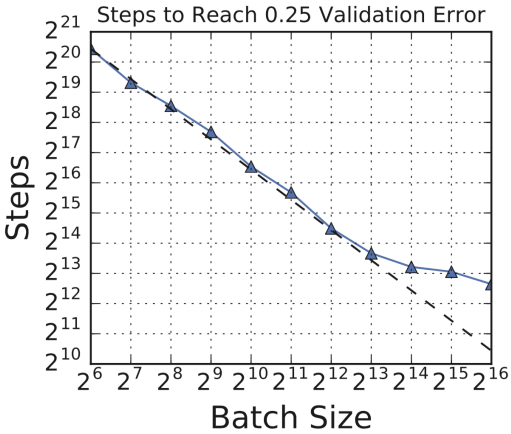
No Free Lunch in Distributed Deep Learning



(a) Simple CNN on MNIST (b) Simple CNN on Fashion MNIST (c) ResNet-8 on CIFAR-10

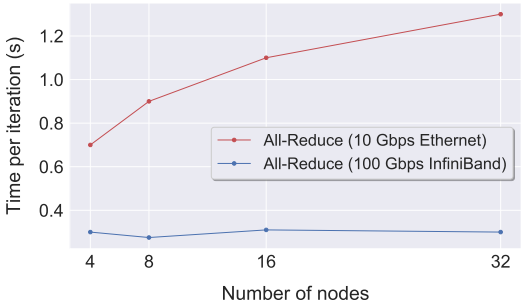
(d) ResNet-50 on ImageNet (e) ResNet-50 on Open Images (f) Transformer on LM1B

(g) Transformer on Common Crawl (h) VGG-11 on ImageNet (i) LSTM on LM1B



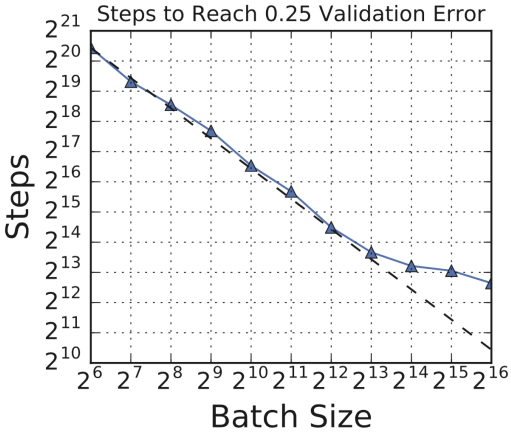
- **Limitation 1**: Diminishing returns of data parallelism with large mini-batch sizes.

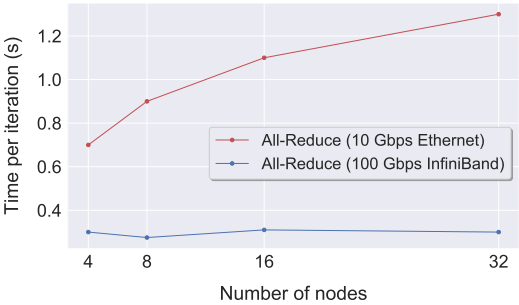
No Free Lunch in Distributed Deep Learning



- **Limitation 2**: Communication bottleneck hinders the training scalability.

No Free Lunch in Distributed Deep Learning





- **Limitation 1**: Diminishing returns of data parallelism with large mini-batch sizes.
- **Limitation 2**: Communication bottleneck hinders the training scalability.

Answers to the Aforementioned Limitations

Large-scale training in the data center has some interesting challenges:

1. The diminishing return of large-batch training [17, 24, 15, 14]

[17] Shallue et al. Measuring the effects of data parallelism on neural network training. JMLR 2019.

[24] You et al. Imagenet training in minutes. ICPP 2018.

[19] Stich et al. Sparsified sgd with memory. NeurIPS 2018.

[18] Stich et al. Local SGD converges fast and communicates little. ICLR 2019.

[6] Karimireddy et al. Error Feedback Fixes SignSGD and other Gradient Compression Schemes. ICML 2019.

[22] Vogels et al. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. NeurIPS 2019.

[21] Vogels et al. Practical Low-Rank Communication Compression in Decentralized Deep Learning. NeurIPS 2020.

[10] Koloskova et al. A unified theory of decentralized SGD with changing topology and local updates. ICML 2020.

[15] Lin et al. Don't Use Large Mini-batches, Use Local SGD. ICLR 2020.

[14] Lin et al. Extrapolation for Large-batch Training in Deep Learning. ICML 2020.

[9] Koloskova*, Lin*, et al. Decentralized Deep Learning with Arbitrary Communication Compression. ICLR 2020.

[11] Kong*, Lin*#, et al. Consensus Control for Decentralized Deep Learning. ICML 2021.

[8] Koloskova et al. An improved analysis of gradient tracking for decentralized machine learning. NeurIPS 2021.

[20] Vogels et al. Relaysum for decentralized deep learning on heterogeneous data. NeurIPS 2021.

Answers to the Aforementioned Limitations

Large-scale training in the data center has some interesting challenges:

1. The diminishing return of large-batch training [17, 24, 15, 14]
2. Communication-efficient training techniques
   - Less frequent communication: Local SGD [18, 15, 10]
   - Reducing communication cost per round—compressed communication: [19, 6, 22, 9, 21]
   - Reducing communication cost per round—decentralized communication: [10, 11, 8, 20]

---

[17] Shallue et al. Measuring the effects of data parallelism on neural network training. JMLR 2019.

[24] You et al. Imagenet training in minutes. ICPP 2018.

[19] Stich et al. Sparsified sgd with memory. NeurIPS 2018.

[18] Stich et al. Local SGD converges fast and communicates little. ICLR 2019.

[6] Karimireddy et al. Error Feedback Fixes SignSGD and other Gradient Compression Schemes. ICML 2019.

[22] Vogels et al. PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization. NeurIPS 2019.

[21] Vogels et al. Practical Low-Rank Communication Compression in Decentralized Deep Learning. NeurIPS 2020.

[10] Koloskova et al. A unified theory of decentralized SGD with changing topology and local updates. ICML 2020.

[15] Lin et al. Don't Use Large Mini-batches, Use Local SGD. ICLR 2020.

[14] Lin et al. Extrapolation for Large-batch Training in Deep Learning. ICML 2020.

[9] Koloskova*, Lin*, et al. Decentralized Deep Learning with Arbitrary Communication Compression. ICLR 2020.

[11] Kong*, Lin*#, et al. Consensus Control for Decentralized Deep Learning. ICML 2021.

[8] Koloskova et al. An improved analysis of gradient tracking for decentralized machine learning. NeurIPS 2021.

[20] Vogels et al. Relaysum for decentralized deep learning on heterogeneous data. NeurIPS 2021.

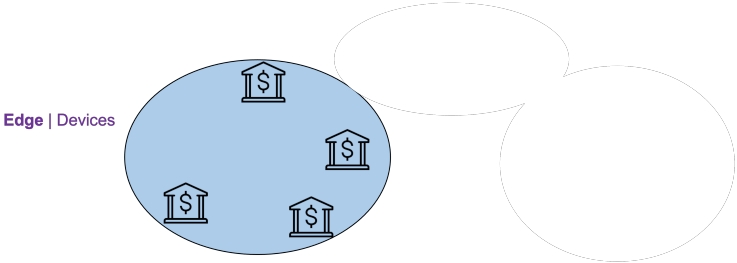
Table of Contents

All previous aspects are about efficiency!

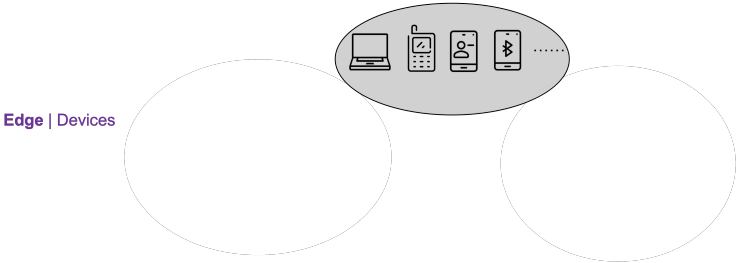
What if the privacy is a concern?

Where does ML data come from?

Where does ML data come from?

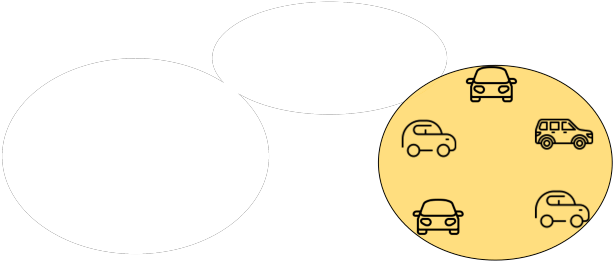


Where does ML data come from?

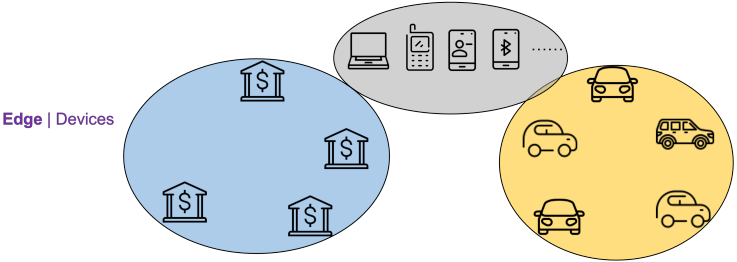


Where does ML data come from?

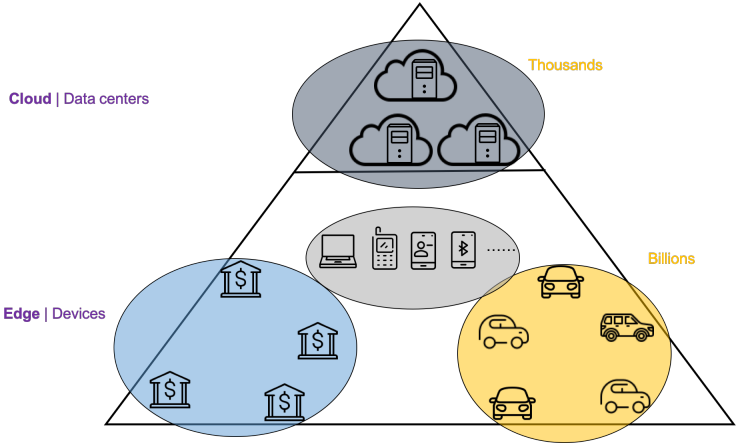
Edge | Devices



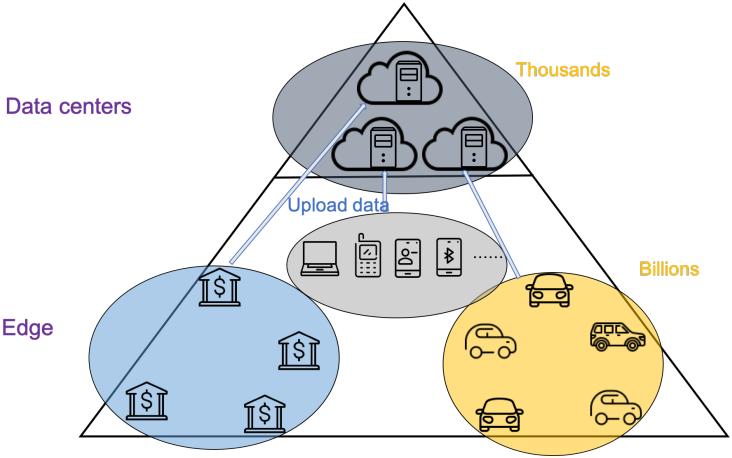
Where does ML data come from?



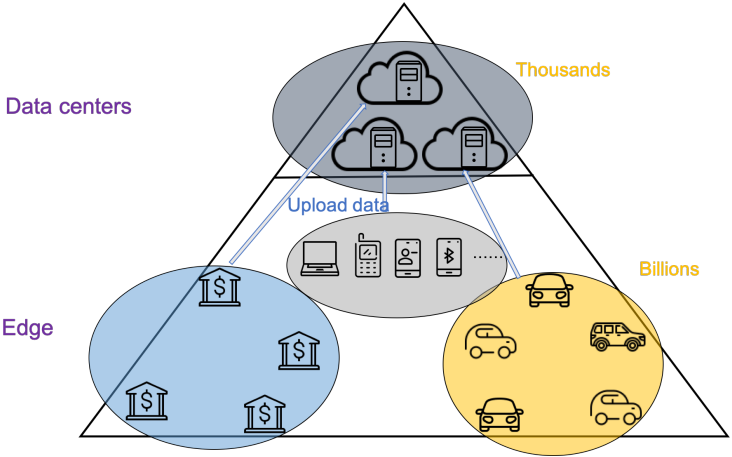
Where does ML data come from?



Where does ML data come from?

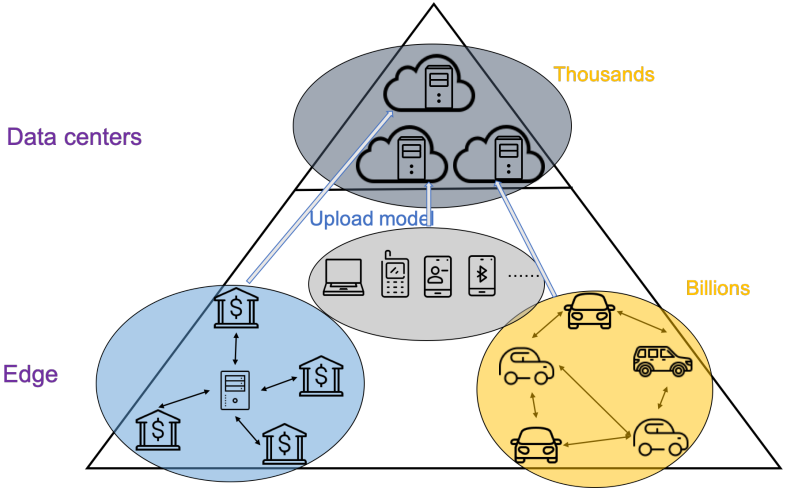


Where does ML data come from?



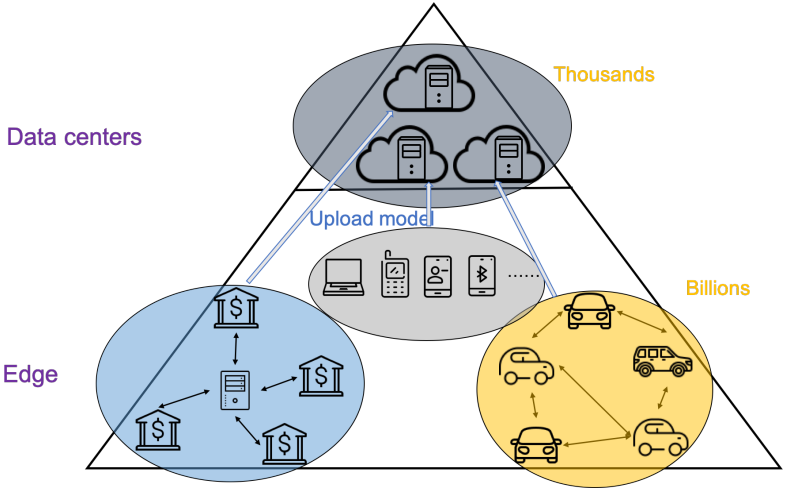
Concerns, e.g., data quality and data privacy, are rising!

Collaborative learning alleviates the *data privacy* concern



An example: Federated Learning (FL)

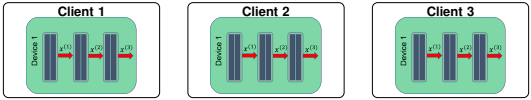
Collaborative learning alleviates the *data privacy* concern



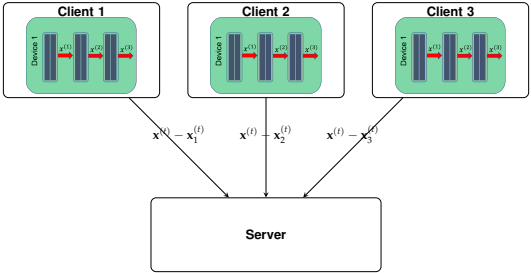
An example: Federated Learning (FL)

Instead of sending sensitive client data over the internet, just share **client models**!

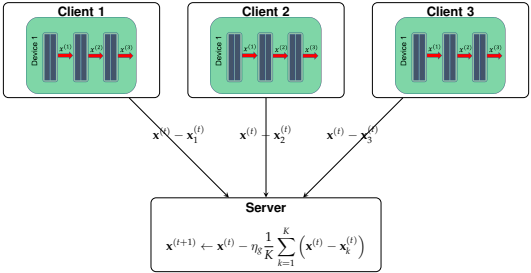
The backbone of FL: Federated Averaging (FedAvg)



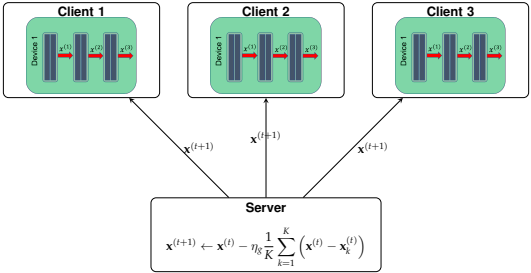
The backbone of FL: Federated Averaging (FedAvg)



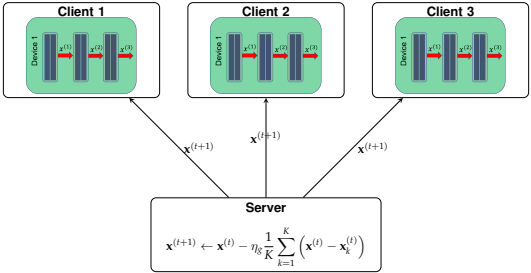
The backbone of FL: Federated Averaging (FedAvg)



The backbone of FL: Federated Averaging (FedAvg)



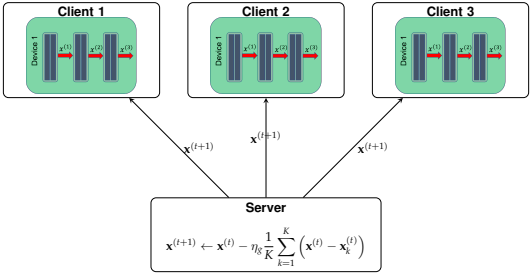
The backbone of FL: Federated Averaging (FedAvg)



Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} \Big(f_i(\mathbf{x}) \qquad \Big) \right\}.$$

The backbone of FL: Federated Averaging (FedAvg)

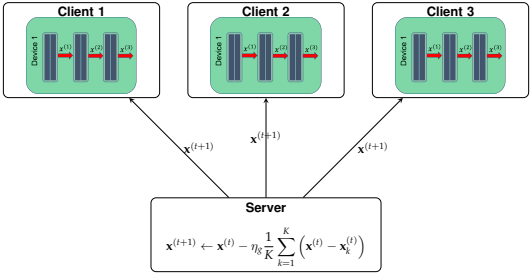


Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} \left(f_i(\mathbf{x}) \qquad \right) \right\}.$$

- The function f_i represents the loss function on client i;

The backbone of FL: Federated Averaging (FedAvg)

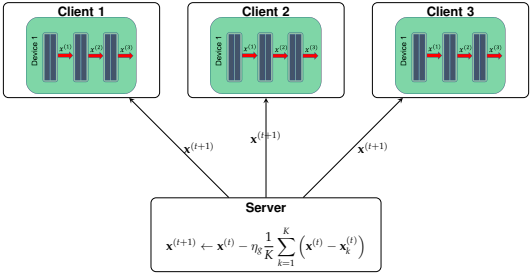


Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi \in \mathcal{D}_i} \left[F_i(\mathbf{x}; \xi) \right] \right) \right\} .$$

- The function f_i represents the loss function on client i;
- $\mathcal{D}_i$ indicates the local data distribution of client i;

The backbone of FL: Federated Averaging (FedAvg)

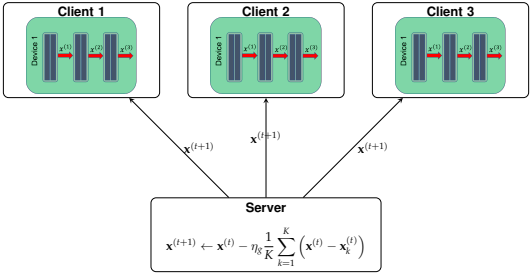


Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi \in \mathcal{D}_i} \left[F_i(\mathbf{x}; \xi) \right] \right) \right\} .$$

- The function f_i represents the loss function on client i;
- $\mathcal{D}_i$ indicates the local data distribution of client i;
- $F_i(\mathbf{x}, \xi)$ corresponds to the sample-wise loss function;

The backbone of FL: Federated Averaging (FedAvg)



Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^\star) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} \left(f_i(\mathbf{x}) := \mathbb{E}_{\xi \in \mathcal{D}_i} \left[F_i(\mathbf{x}; \xi) \right] \right) \right\}.$$

- The function f_i represents the loss function on client i;
- $\mathcal{D}_i$ indicates the local data distribution of client i;
- $F_i(\mathbf{x}, \xi)$ corresponds to the sample-wise loss function;
- FedAvg performs multiple local update steps per round.

Challenges of FL [13, 4, 23]

Communication overhead
slow & unreliable networks

Data heterogeneity
highly non-identically distributed data

Systems heterogeneity
variable hardware, power, etc

Privacy concerns
privacy leakage

[13] *Li et al.* Federated Learning: Challenges, Methods, and Future Directions. 2020.
[4] *Kairouz et al.* Advances and open problems in federated learning. 2021.
[23] *Wang et al.* A Field Guide to Federated Optimization. 2021.

Challenges of FL [13, 4, 23]

Communication overhead
slow & unreliable networks

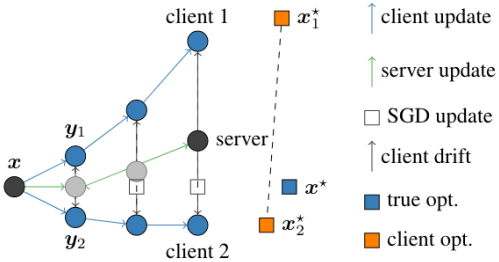
Data heterogeneity
highly non-identically distributed data

Systems heterogeneity
variable hardware, power, etc

Privacy concerns
privacy leakage

[13] *Li et al.* Federated Learning: Challenges, Methods, and Future Directions. 2020.
[4] *Kairouz et al.* Advances and open problems in federated learning. 2021.
[23] *Wang et al.* A Field Guide to Federated Optimization. 2021.

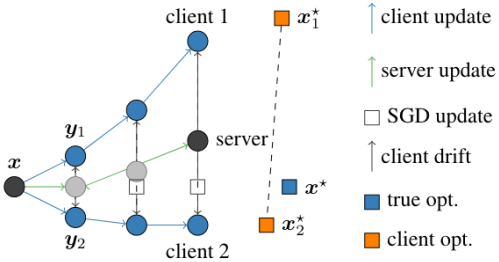
Data heterogeneity in FL



Client drift issue defined in [5].

[5] *Karimireddy et al.* SCAFFOLD: Stochastic controlled averaging for federated learning. ICML 2020.

Data heterogeneity in FL



Client drift issue defined in [5].

Data-dissimilarity $\zeta^2 > 0$ causes *drift* when doing local steps.

$$\mathbb{E}_i \left[\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2 \right] \leq \zeta^2 \tag{21}$$

[5] *Karimireddy et al.* SCAFFOLD: Stochastic controlled averaging for federated learning. ICML 2020.

Table of Contents

1. Data-parallelism is the current workhorse for large-scale deep learning training

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication
     - A line of research requires unbiased gradient estimator, which is non-trivial

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication
     - A line of research requires unbiased gradient estimator, which is non-trivial
     - Error-feedback enables the convergence for arbitrary compressors, even for biased estimator

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication
     - A line of research requires unbiased gradient estimator, which is non-trivial
     - Error-feedback enables the convergence for arbitrary compressors, even for biased estimator
   - Decentralized communication

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication
     - A line of research requires unbiased gradient estimator, which is non-trivial
     - Error-feedback enables the convergence for arbitrary compressors, even for biased estimator
   - Decentralized communication
     - Nodes only communicate with its neighborhood, reducing the cost per iteration.

1. Data-parallelism is the current workhorse for large-scale deep learning training
   - Theoretically linear speedup by adding more GPUs
   - Diminishing returns for large-batch region
   - Communication bottleneck for large-scale training
2. Communication-efficient training techniques
   - Compressed communication
     - A line of research requires unbiased gradient estimator, which is non-trivial
     - Error-feedback enables the convergence for arbitrary compressors, even for biased estimator
   - Decentralized communication
     - Nodes only communicate with its neighborhood, reducing the cost per iteration.
     - Trade-off between degraded test accuracy and improved communication efficiency.

Thanks & Question Time!

[1] M. Assran, N. Loizou, N. Ballas, and M. Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

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

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

[4] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[5] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[6] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pages 3252–3261. PMLR, 2019.

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

[8] A. Koloskova, T. Lin, and S. U. Stich. An improved analysis of gradient tracking for decentralized machine learning. *Advances in Neural Information Processing Systems*, 34:11422–11435, 2021.

[9] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi. Decentralized deep learning with arbitrary communication compression. In *International Conference on Learning Representations*, 2020.

[10] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[11] L. Kong, T. Lin, A. Koloskova, M. Jaggi, and S. U. Stich. Consensus control for decentralized deep learning. In *International Conference on Machine Learning*, 2021.

[12] 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.

[13] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.

[14] T. Lin, L. Kong, S. Stich, and M. Jaggi. Extrapolation for large-batch training in deep learning. In *International Conference on Machine Learning*, pages 6094–6104. PMLR, 2020.

[15] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi. Don't use large mini-batches, use local sgd. In *International Conference on Learning Representations*, 2020.
[16] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
[17] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.
[18] S. U. Stich. Local sgd converges fast and communicates little. In *International Conference on Learning Representations*, 2019.
[19] S. U. Stich, J.-B. Cordonnier, and M. Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.
[20] T. Vogels, L. He, A. Koloskova, S. P. Karimireddy, T. Lin, S. Stich, and M. Jaggi. Relaysum for decentralized deep learning on heterogeneous data. 2021.
[21] T. Vogels, S. P. Karimireddy, and M. Jaggi. Practical low-rank communication compression in decentralized deep learning. In *NeurIPS*, 2020.
[22] T. Vogels, S. P. Karinireddy, and M. Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances In Neural Information Processing Systems 32 (Nips 2019)*, 32(CONF), 2019.
[23] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021.

[24] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[25] 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.