

# Optimization for Deep Learning

**Tao LIN**, presented by Tailin Wu

Department of Artificial Intelligence, School of Engineering, Westlake University

March 11, 2026



# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods
- 4 Introduction to Distributed Deep Learning

# Background

- 1 Get data:  $\xi_1, \dots, \xi_N$ , where  $\xi_i := (\mathbf{d}, y)_i$

# Background

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

$$h_{\mathbf{x}}(\mathbf{d}) \rightarrow y$$
$$h_{\mathbf{x}} \left( \begin{array}{c} \text{🐱} \end{array} \right) \rightarrow \text{cat} \quad (1)$$

# Background

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

$$h_{\mathbf{x}}(\mathbf{d}) \rightarrow y$$
$$h_{\mathbf{x}} \left( \begin{array}{c} \text{cat} \end{array} \right) \rightarrow \text{cat} \quad (1)$$

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



Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

# Background

Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

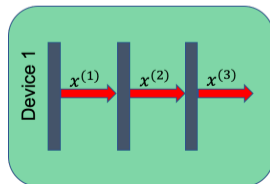
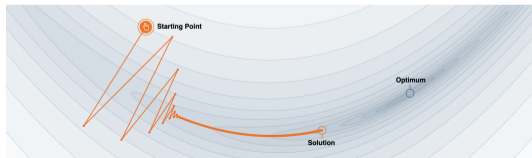
# Background

Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla F(\mathbf{x}^{(t)}, \xi_i) \quad (4)$$



# Background

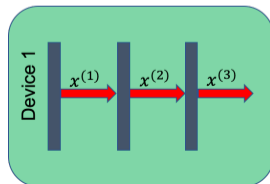
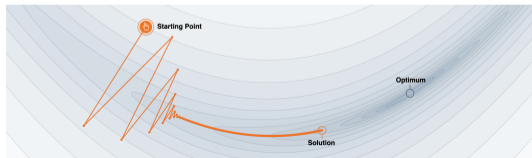
Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla F(\mathbf{x}^{(t)}, \xi_i) \quad (4)$$

- $\eta$  is step-size/learning rate



# Background

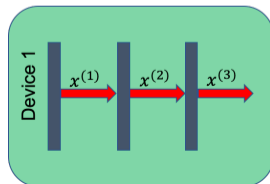
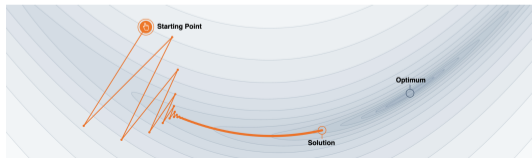
Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \nabla F(\mathbf{x}^{(t)}, \xi_i) \quad (4)$$

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



# Background

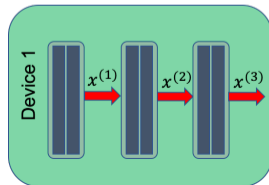
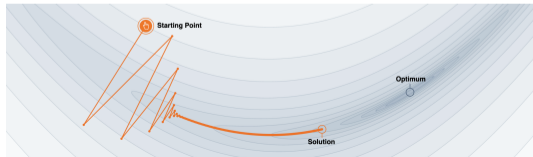
Finite-sum empirical risk minimization problem:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

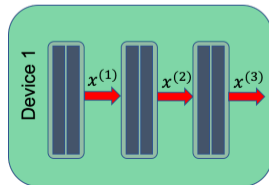
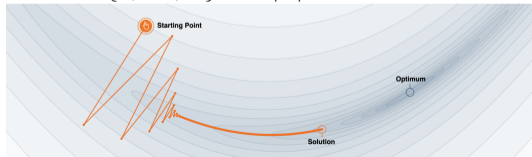
$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^d} \left\{ f(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^N \left( f_i(\mathbf{x}) := F(\mathbf{x}, \xi_i) \right) \right\} \quad (3)$$

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

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



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

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

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

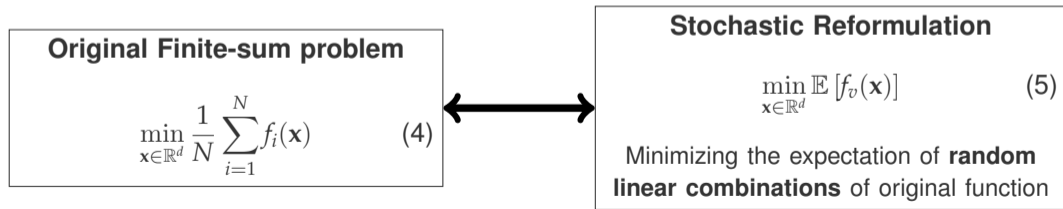
**Random sampling vector**  $\mathbf{v} = (v_1, \dots, v_N) \sim \mathcal{D}$  with  $\mathbb{E}[v_i] = 1$  for  $i = 1, \dots, N$ .

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

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

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

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



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

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

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

**Original Finite-sum problem**

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

**Stochastic Reformulation**

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

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

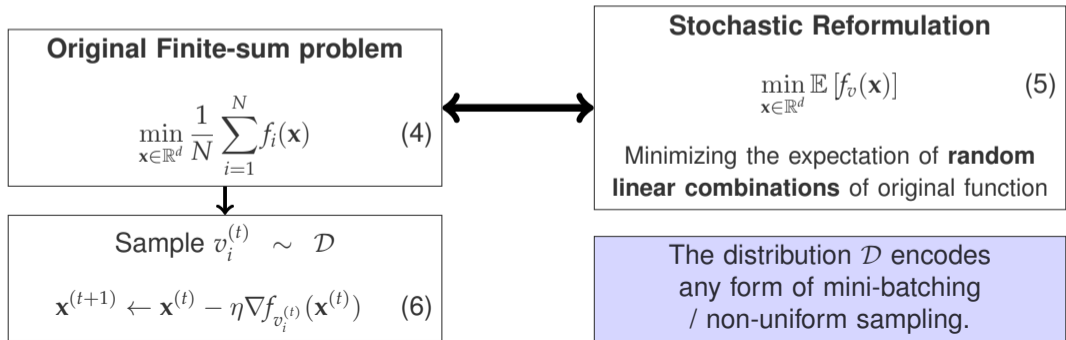
Sample  $v_i^{(t)} \sim \mathcal{D}$

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

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

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

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



# The convergence of mini-batch SGD

## Assumption 1

- *The function  $f(\mathbf{x})$  we are minimizing is lower bounded from below by  $f^* := f(\mathbf{x}^*)$ , 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  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E} [\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E}[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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)

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

# The convergence of mini-batch SGD

## Assumption 1

- The function  $f(\mathbf{x})$  we are minimizing is lower bounded from below by  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E}[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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

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

# The convergence of mini-batch SGD

## Assumption 1

- The function  $f(\mathbf{x})$  we are minimizing is lower bounded from below by  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E}[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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

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

- $T$ : 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  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E}[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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

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

- $T$ : number of iterations
- $\sigma$ : stochastic gradient variance

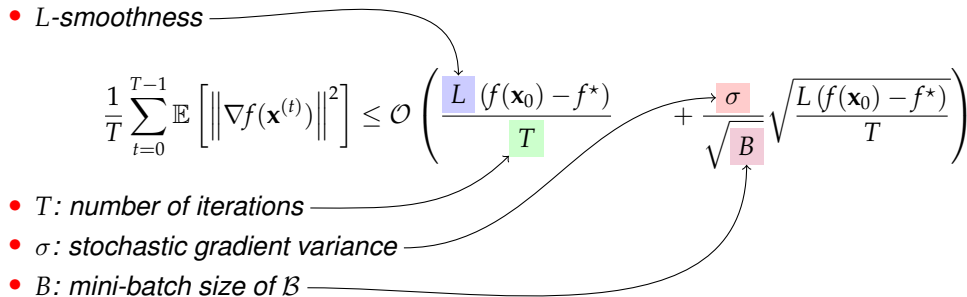
# The convergence of mini-batch SGD

## Assumption 1

- The function  $f(\mathbf{x})$  we are minimizing is lower bounded from below by  $f^* := f(\mathbf{x}^*)$ , 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  $\mathbb{E}[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$  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

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


- $T$ : number of iterations
- $\sigma$ : 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*

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

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

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

# The convergence of mini-batch SGD

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

- *L-smoothness*

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

- *T: number of iterations*

- *$\sigma$ : stochastic gradient variance*

- *B: mini-batch size of  $\mathcal{B}$*

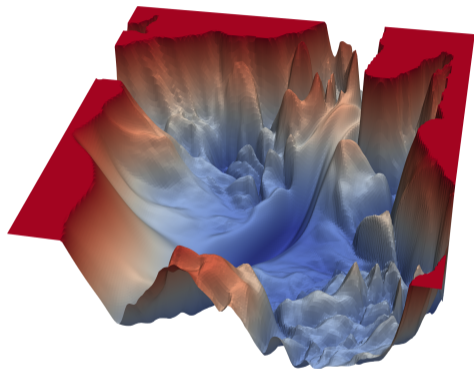
- When iterations  $T \rightarrow \infty$ , it holds that  $\mathbb{E} \left[ \left\| \nabla f(\mathbf{x}^{(t)}) \right\|^2 \right] \rightarrow 0$

- $\mathbb{E} \left[ \left\| \nabla f(\mathbf{x}^{(t)}) \right\|^2 \right] \rightarrow 0$  implies the sequence converges to a stationary solution

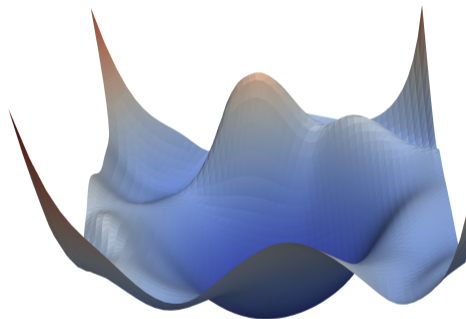
# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods**
- 3 Advanced Optimization Methods
- 4 Introduction to Distributed Deep Learning

# Issues of SGD—from the perspective of loss landscape



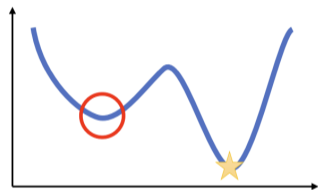
(a) ResNet w/o skip connections.



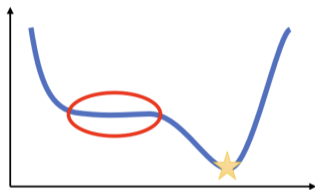
(b) ResNet w/ skip connections.

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

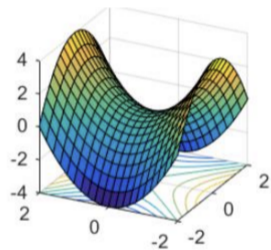
# Issues of SGD—from the perspective of loss landscape



the local optimum



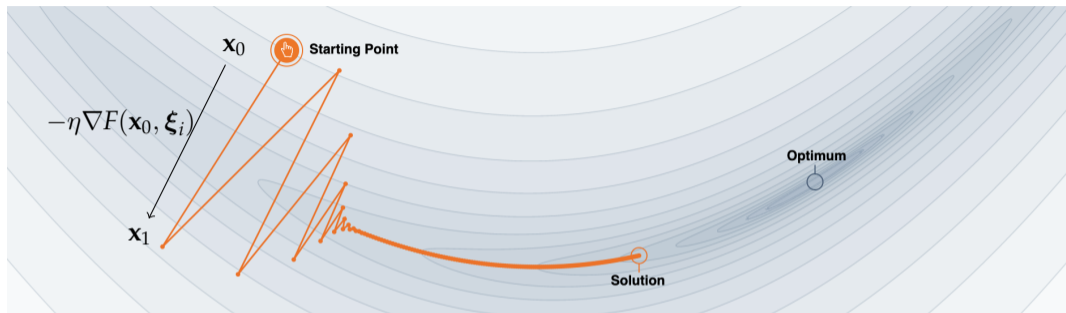
the plateau



the saddle point

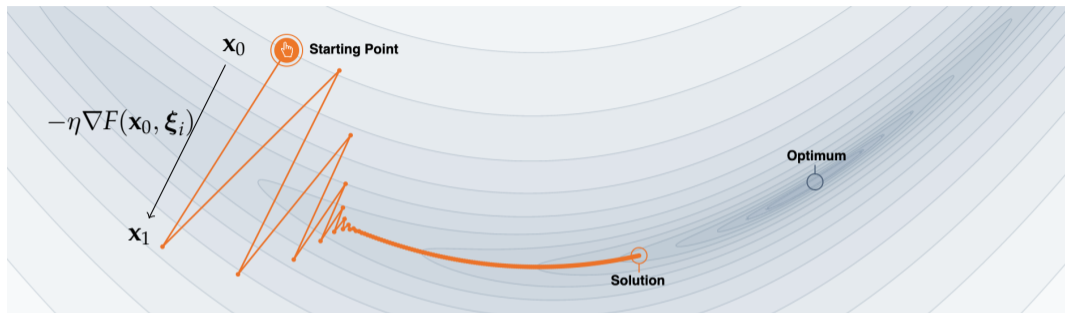
**Challenging optimization loss landscape!**

# Issues of SGD—from the perspective of loss landscape



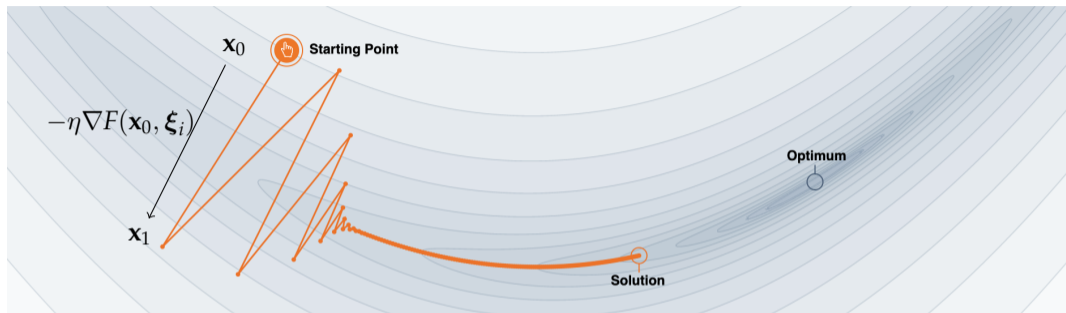
- **Challenges # 1:** loss function has high condition number.  
→ very slow progress along shallow dimension, jitter along steep direction.

# Issues of SGD—from the perspective of loss landscape



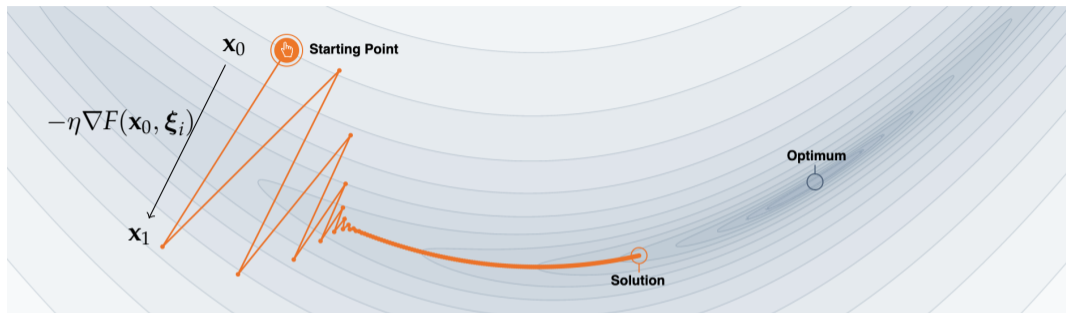
- **Challenges # 1:** loss function has high condition number.  
→ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more:** plateaus & saddle points.

# Issues of SGD—from the perspective of loss landscape



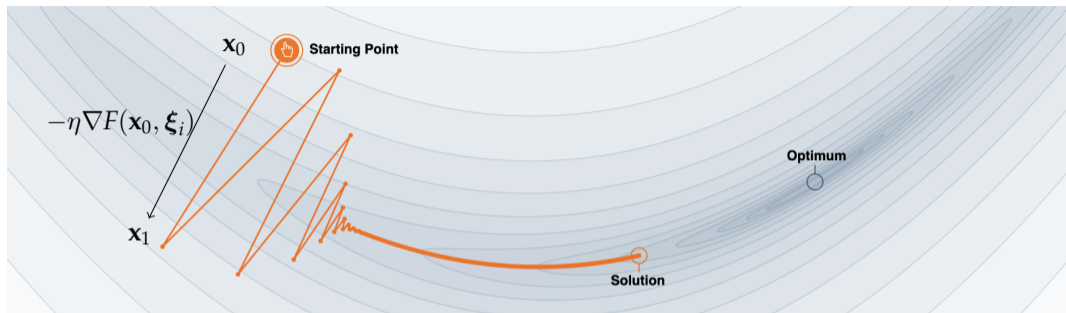
- **Challenges # 1:** loss function has high condition number.  
→ very slow progress along shallow dimension, jitter along steep direction.
- **Challenges # 2 & more:** plateaus & saddle points.  
→ cannot just choose tiny learning rates to prevent oscillation!

# Issues of SGD—from the perspective of loss landscape



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

# Issues of SGD—from the perspective of loss landscape



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

# 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.

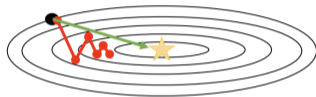
# Improvement directions: leveraging the curvature information

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

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

**Taylor expansion:**

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



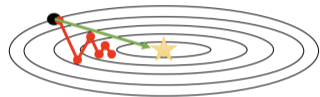
# Improvement directions: leveraging the curvature information

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

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

**Taylor expansion:**

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



**Multivariate case:**

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

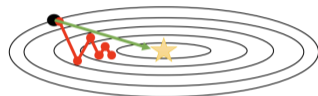
# Improvement directions: leveraging the curvature information

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

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

**Taylor expansion:**

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



**Multivariate case:**

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

**Solution** (can optimize this analytically!):

$$\mathbf{x}^* \leftarrow \mathbf{x}_0 - (\nabla_{\mathbf{x}}^2f(\mathbf{x}_0))^{-1} \nabla_{\mathbf{x}}f(\mathbf{x}_0) \quad (9)$$

## Improvement directions: trade-offs and approximations

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

---

<sup>1</sup>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)<sup>1</sup>:  $\mathcal{O}(N^3)$

---

<sup>1</sup>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.

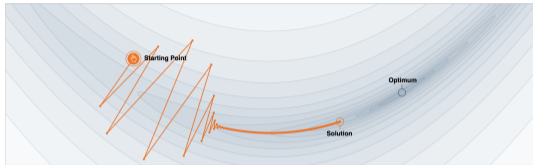
GD (w/ Hessian)<sup>1</sup>:  $\mathcal{O}(N^3)$

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

---

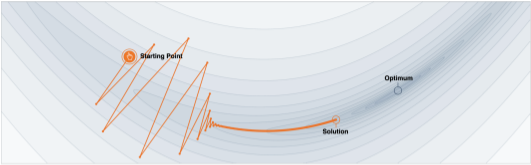
<sup>1</sup>if using naive approach, though fancy methods can be much faster if they avoid forming the Hessian explicitly.

# Momentum method

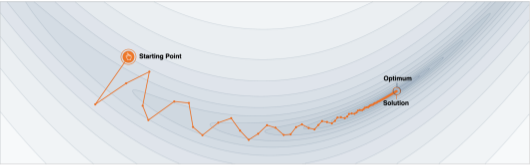


w/o momentum

# Momentum method

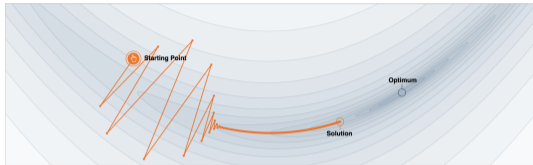


w/o momentum

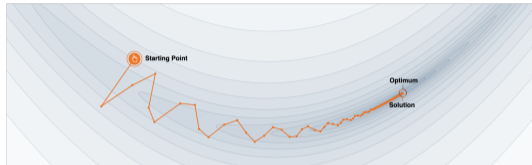


w/ momentum

# Momentum method



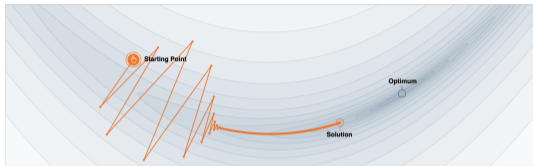
w/o momentum



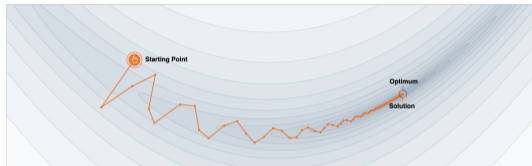
w/ momentum

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

# Momentum method



w/o momentum

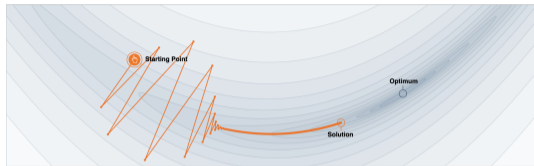


w/ momentum

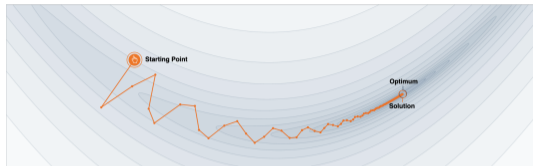
**Intuition:** averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions

# Momentum method



w/o momentum

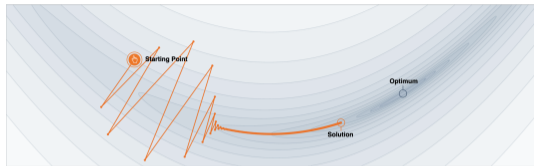


w/ momentum

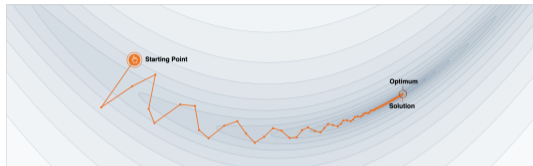
**Intuition:** averaging together successive gradients yield a much better direction!

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

# Momentum method



w/o momentum

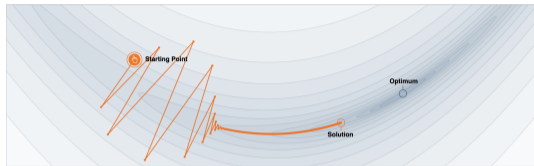


w/ momentum

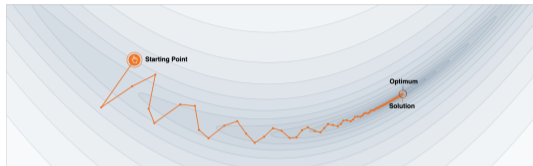
**Intuition:** averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions  
→ we should cancel off the directions that disagree
- if successive gradient step point in similar directions

# Momentum method



w/o momentum

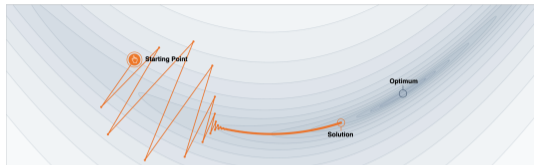


w/ momentum

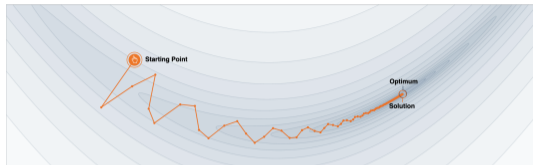
**Intuition:** averaging together successive gradients yield a much better direction!

- if successive gradient step point in different directions  
→ we should cancel off the directions that disagree
- if successive gradient step point in similar directions  
→ 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  
→ we should cancel off the directions that disagree
- if successive gradient step point in similar directions  
→ we should go faster in that direction

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

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

# Methods that manipulate gradient scale

**Intuition behind  $\nabla F(\mathbf{x}_i, \xi_i)$ :**

- *sign*:

# Methods that manipulate gradient scale

**Intuition behind  $\nabla F(\mathbf{x}_i, \xi_i)$ :**

- *sign*:
- *magnitude*:

# Methods that manipulate gradient scale

**Intuition behind  $\nabla F(\mathbf{x}_i, \xi_i)$ :**

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

# Methods that manipulate gradient scale

**Intuition behind  $\nabla F(\mathbf{x}_i, \xi_i)$ :**

- *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  $\nabla F(\mathbf{x}_i, \xi_i)$ :**

- *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:

→ overall magnitude of the gradient can change drastically during the optimization,

# Methods that manipulate gradient scale

## Intuition behind $\nabla F(\mathbf{x}_i, \xi_i)$ :

- *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:

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

# Methods that manipulate gradient scale

**Intuition behind**  $\nabla F(\mathbf{x}_i, \xi_i)$ :

- *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:
  - 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.

**AdaGrad** [2] (estimate per-dimension cumulative magnitude):

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

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

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

**AdaGrad** [2] (estimate per-dimension cumulative magnitude):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

---

RMSProp. [https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](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):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

Remarks:

---

RMSProp. [https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](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):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

Remarks:

- AdaGrad has some appealing guarantees for convex problems.

---

RMSProp. [https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](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.

**AdaGrad** [2] (estimate per-dimension cumulative magnitude):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

Remarks:

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

**AdaGrad** [2] (estimate per-dimension cumulative magnitude):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

Remarks:

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

**AdaGrad** [2] (estimate per-dimension cumulative magnitude):

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

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

**RMSProp** (estimate per-dimension magnitude):

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

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

Remarks:

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

---

RMSProp. [https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](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:**

# Adam: combining momentum and RMSProp

## Idea:

- Maintain exponential moving averages of gradient and its square

# Adam: combining momentum and RMSProp

## Idea:

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

# Adam: combining momentum and RMSProp

## Idea:

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

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

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

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

# Adam: combining momentum and RMSProp

## Idea:

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

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

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

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

where compared to RMSProp, Adam

# Adam: combining momentum and RMSProp

## Idea:

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

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

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

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

where compared to RMSProp, Adam

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

# Adam: combining momentum and RMSProp

## Idea:

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

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

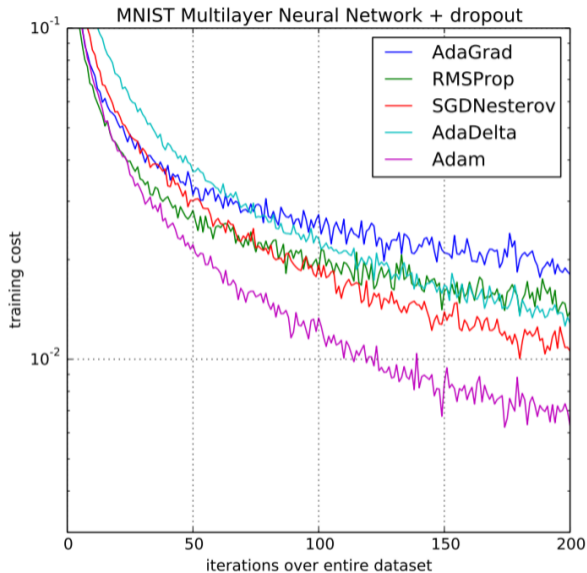
$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) (\nabla F(\mathbf{x}_t, \boldsymbol{\xi}_t))^2 \quad \text{(second moment estimate)}$$

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

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  $\beta_2$  is close to 1).

# 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:

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

# Weight decay in SGD and Adam: why AdamW matters

Many learning problems optimize the loss with  $L_2$  norm penalty:

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

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

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

# 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).

# 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 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.

# 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)

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

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t - \underset{\text{weight decay}}{2\eta\lambda\mathbf{x}_t} \quad (11)$$

# 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)

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

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbf{m}_t - \underset{\text{weight decay}}{2\eta\lambda\mathbf{x}_t} \quad (11)$$

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

# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods**
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning

# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods**
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning
  - Preliminary for Distributed Optimization
  - Federated Learning
  - Summary

# 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.

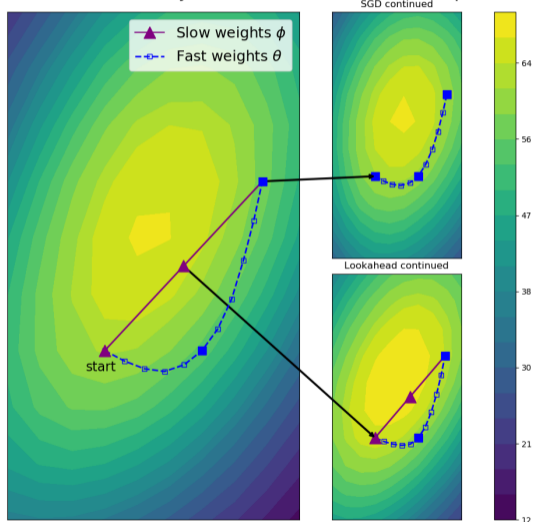
# Lookahead Optimizer: $k$ steps forward, 1 step back

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

CIFAR-100 accuracy surface with Lookahead interpolation



A natural way to explore and exploit the landscape!

---

**Algorithm 1: SGD**

---

**Input** : Objective  $F_S(\theta)$ , dataset  $S$ , 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**  $t = 1, 2, \dots, T$  **do**

$v_0^{(t)} = \theta_{t-1}$ ;

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

$v_\tau^{(t)} = \mathcal{A}(F_S(\theta), v_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, S) = v_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} g_{\tau-1}^{(t)}$

**end**

$\theta_t = v_k^{(t)} = (1 - 1)\theta_{t-1} + 1 * v_k^{(t)}$  ( $\alpha = 1$ )

**end**

**Output** :  $\theta_{\mathcal{A}, S} = \theta_T$

---

Inner-loop optimization

outer-loop optimization

---

**Algorithm 2: Lookahead**

---

**Input** : Objective  $F_S(\theta)$ , dataset  $S$ , 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**  $t = 1, 2, \dots, T$  **do**

$v_0^{(t)} = \theta_{t-1}$ ;

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

$v_\tau^{(t)} = \mathcal{A}(F_S(\theta), v_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, S) = v_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} g_{\tau-1}^{(t)}$

**end**

$\theta_t = (1 - \alpha)\theta_{t-1} + \alpha v_k^{(t)}$ .

**end**

**Output** :  $\theta_{\mathcal{A}, S} = \theta_T$

---

Inner-loop optimization

outer-loop optimization

---

**Algorithm 1: SGD**

---

**Input** : Objective  $F_S(\theta)$ , 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**  $t = 1, 2, \dots, T$  **do**

$\mathbf{v}_0^{(t)} = \theta_{t-1}$ ;

**Inner-loop optimization**

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

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

**end**

$\theta_t = \mathbf{v}_k^{(t)} = (1 - 1)\theta_{t-1} + 1 * \mathbf{v}_k^{(t)}$  ( $\alpha = 1$ )

**end**

**Output** :  $\theta_{\mathcal{A}, \mathcal{S}} = \theta_T$

---

---

**Algorithm 2: Lookahead**

---

**Input** : Objective  $F_S(\theta)$ , 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**  $t = 1, 2, \dots, T$  **do**

$\mathbf{v}_0^{(t)} = \theta_{t-1}$ ;

**Inner-loop optimization**

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

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

**end**

$\theta_t = (1 - \alpha)\theta_{t-1} + \alpha \mathbf{v}_k^{(t)}$ .

**end**

**Output** :  $\theta_{\mathcal{A}, \mathcal{S}} = \theta_T$

---

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

---

**Algorithm 1: SGD**

---

**Input** : Objective  $F_S(\theta)$ , 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  $t = 1, 2, \dots, T$  do
   $\mathbf{v}_0^{(t)} = \theta_{t-1}$ ;
  for  $\tau = 1, 2, \dots, k$  do
     $\mathbf{v}_\tau^{(t)} = \mathcal{A}(F_S(\theta), \mathbf{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \mathbf{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \mathbf{g}_{\tau-1}^{(t)}$ 
  end
   $\theta_t = \mathbf{v}_k^{(t)} = (1 - 1)\theta_{t-1} + 1 * \mathbf{v}_k^{(t)}$  ( $\alpha = 1$ )
```

end  
**Output** :  $\theta_{\mathcal{A}, \mathcal{S}} = \theta_T$  outer-loop optimization

---

---

**Algorithm 2: Lookahead**

---

**Input** : Objective  $F_S(\theta)$ , 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  $t = 1, 2, \dots, T$  do
   $\mathbf{v}_0^{(t)} = \theta_{t-1}$ ;
  for  $\tau = 1, 2, \dots, k$  do
     $\mathbf{v}_\tau^{(t)} = \mathcal{A}(F_S(\theta), \mathbf{v}_{\tau-1}^{(t)}, \eta_{\tau-1}^{(t)}, \mathcal{S}) = \mathbf{v}_{\tau-1}^{(t)} - \eta_{\tau-1}^{(t)} \mathbf{g}_{\tau-1}^{(t)}$ 
  end
   $\theta_t = (1 - \alpha)\theta_{t-1} + \alpha \mathbf{v}_k^{(t)}$ .
```

end  
**Output** :  $\theta_{\mathcal{A}, \mathcal{S}} = \theta_T$  outer-loop optimization

---

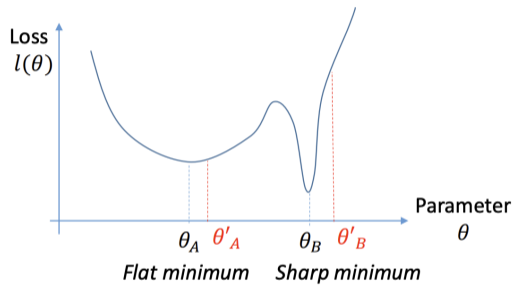
- inner-loop optimization:  $k$  steps forward in SGD & LA
- outer-loop optimization: 1 step back in LA, while no step back in SGD

# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods**
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning
  - Preliminary for Distributed Optimization
  - Federated Learning
  - Summary

# Flat Minima in Deep Learning

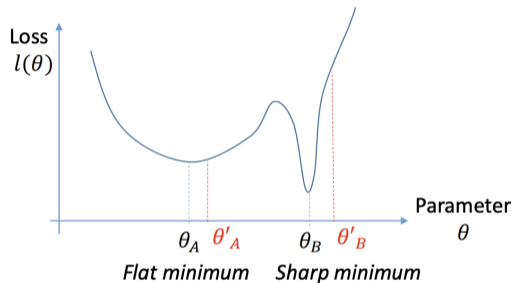
In many cases,



# Flat Minima in Deep Learning

In many cases,

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

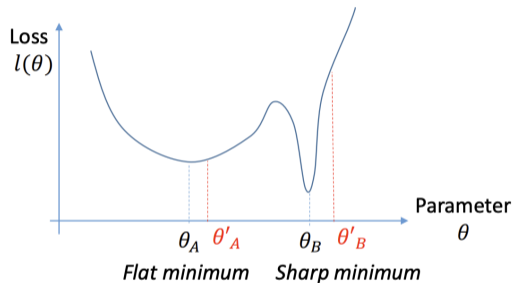


# Flat Minima in Deep Learning

In many cases,

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

*Highly non-convex (many local minima)!*

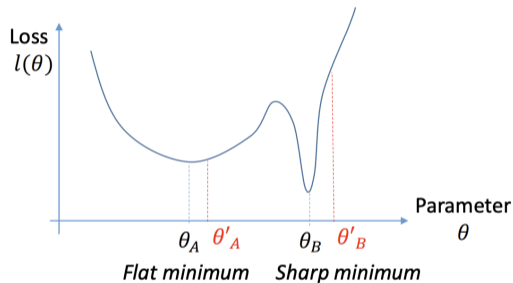


# Flat Minima in Deep Learning

In many cases,

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

*Highly non-convex (many local minima)!*



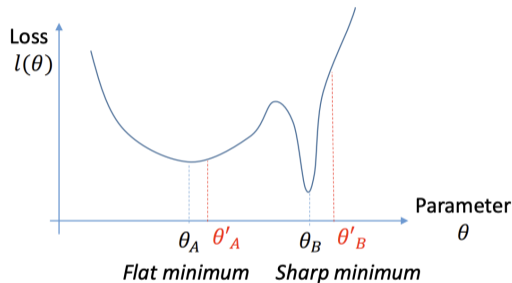
- Q) Which is better,  $\theta_A$  or  $\theta_B$ ?

# Flat Minima in Deep Learning

In many cases,

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

*Highly non-convex (many local minima)!*



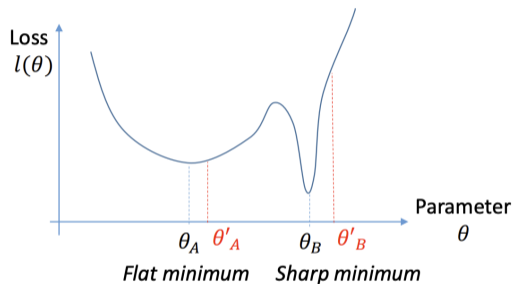
- Q) Which is better,  $\theta_A$  or  $\theta_B$ ?
- A) We prefer  $\theta_A$  to  $\theta_B$  even though  $\ell(\theta_A) > \ell(\theta_B)$

# Flat Minima in Deep Learning

In many cases,

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

*Highly non-convex (many local minima)!*



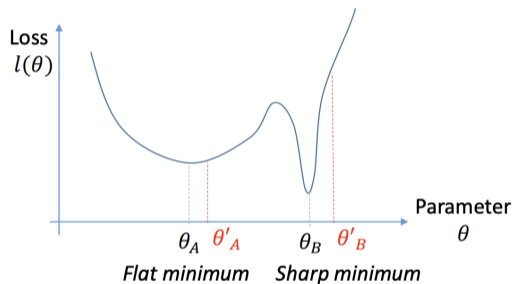
- Q) Which is better,  $\theta_A$  or  $\theta_B$ ?
- A) We prefer  $\theta_A$  to  $\theta_B$  even though  $\ell(\theta_A) > \ell(\theta_B)$ 
  - Why? Because  $\theta_A$  is more robust.

# Flat Minima in Deep Learning

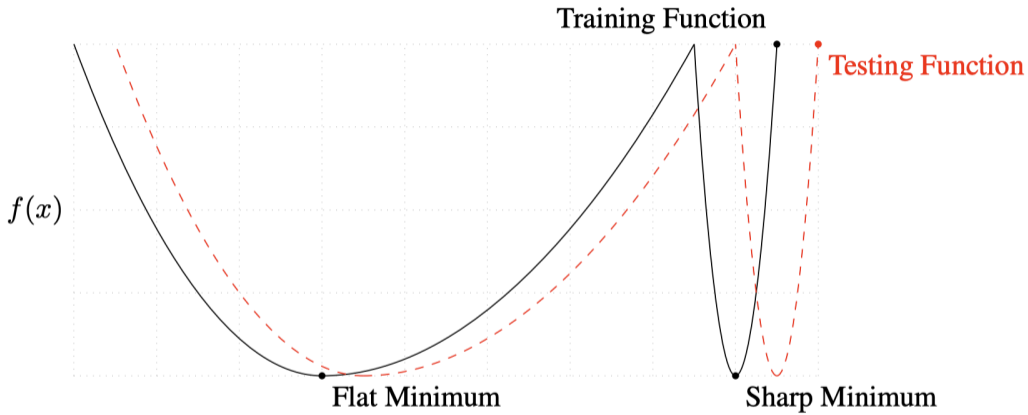
In many cases,

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

*Highly non-convex (many local minima)!*



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



# Let's seek for a Flat Minimum

Flat Minima = Robust models (12)

= Resilient to data noise or model corruption (13)

= (often encountered in AI applications) (14)

# Let's seek for a Flat Minimum

Flat Minima = Robust models (12)

= Resilient to data noise or model corruption (13)

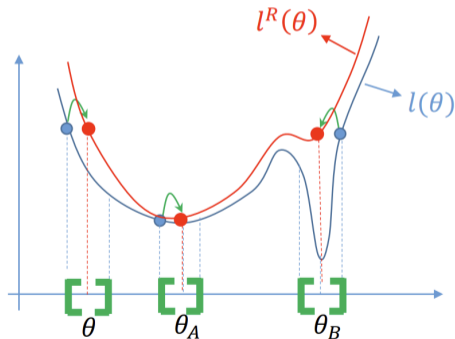
= (often encountered in AI applications) (14)

But, how?

# Sharpness-Aware Minimization (SAM)

## Idea of SAM:

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

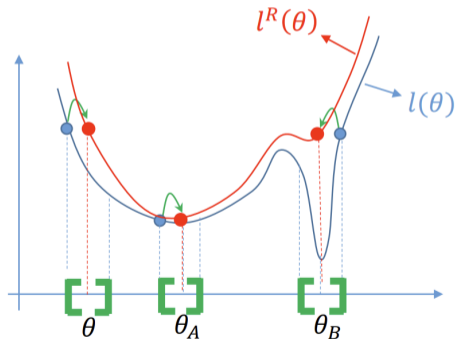


# Sharpness-Aware Minimization (SAM)

## Idea of SAM:

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

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



# Intuition behind SAM

- **Goal:** Find the local minima  $\theta$  that are generalizable to test samples

# Intuition behind SAM

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

# Intuition behind SAM

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

## Theorem 2

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

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

# Intuition behind SAM

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

## Theorem 2

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

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

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

# Sharpness-Aware Minimization (SAM)

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

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

# Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

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

# Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

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

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

# Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

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

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

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

# Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

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

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

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

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

# Sharpness-Aware Minimization (SAM)

The objective of SAM becomes:

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

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

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

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:

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

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

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

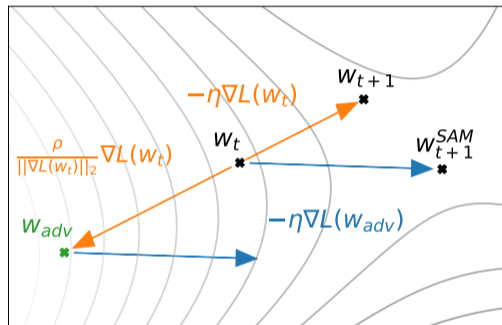
where  $1/p + 1/q = 1$  and the solution to a classical dual norm problem can solve this approximation.

- substituting  $\hat{\boldsymbol{\epsilon}}$  gives a gradient estimator

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

- The gradient estimator of SAM is given by:

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

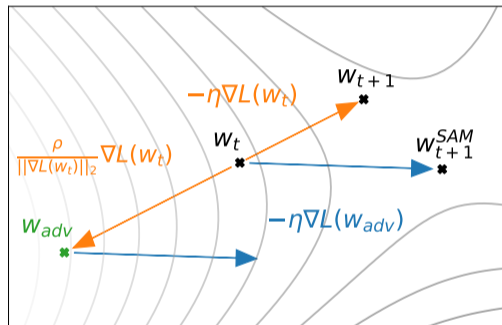


From original SAM paper.

- The gradient estimator of SAM is given by:

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

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

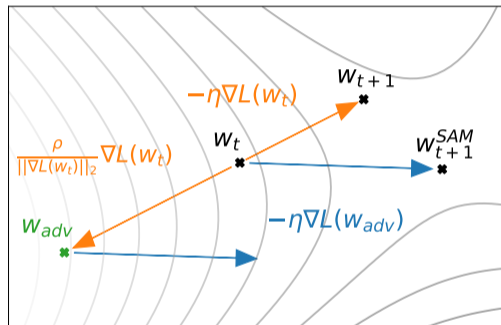


From original SAM paper.

- The gradient estimator of SAM is given by:

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

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

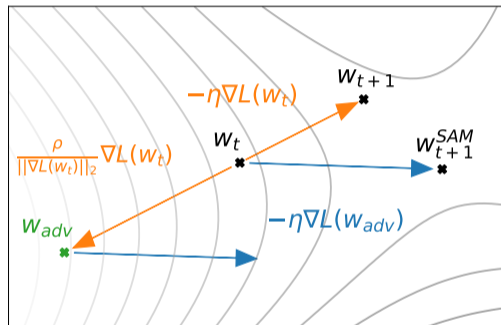


From original SAM paper.

- The gradient estimator of SAM is given by:

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

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

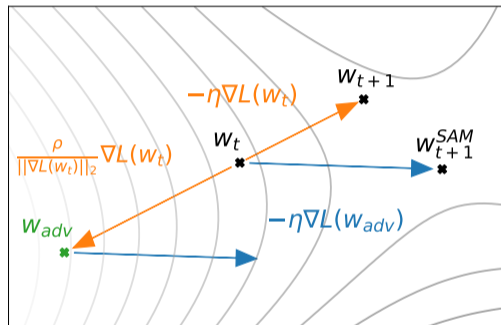


From original SAM paper.

- The gradient estimator of SAM is given by:

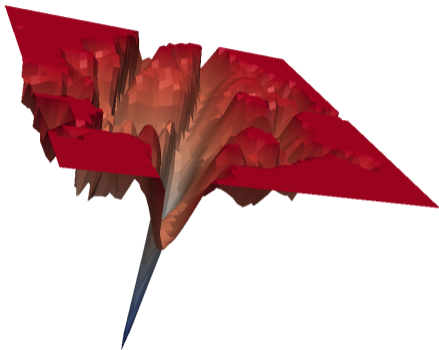
$$\nabla_{\theta} \mathcal{L}_S^{\text{SAM}}(\theta) \approx \nabla_{\theta} \mathcal{L}_S(\theta)|_{\theta+\hat{\epsilon}} \quad (20)$$

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

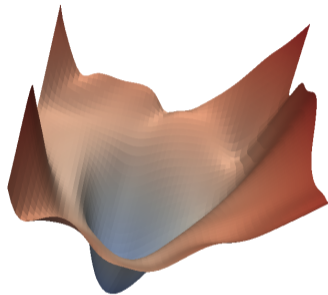


From original SAM paper.

# Verification of the flatness



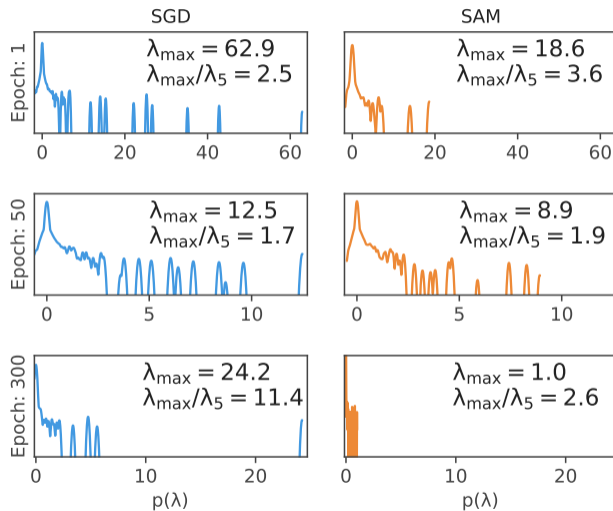
(a) ERM.



(b) SAM.

Loss surface visualization.

# Verification of the flatness



Hessian spectra.

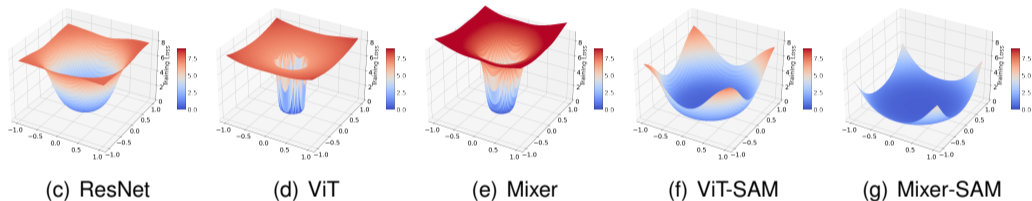
# Results (i.e., SAM > ERM)

- 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	<b>22.5</b> $\pm 0.1$	6.28 $\pm 0.08$	22.9 $\pm 0.1$	6.62 $\pm 0.11$
	200	<b>21.4</b> $\pm 0.1$	5.82 $\pm 0.03$	22.3 $\pm 0.1$	6.37 $\pm 0.04$
	400	<b>20.9</b> $\pm 0.1$	5.51 $\pm 0.03$	22.3 $\pm 0.1$	6.40 $\pm 0.06$
ResNet-101	100	<b>20.2</b> $\pm 0.1$	5.12 $\pm 0.03$	21.2 $\pm 0.1$	5.66 $\pm 0.05$
	200	<b>19.4</b> $\pm 0.1$	4.76 $\pm 0.03$	20.9 $\pm 0.1$	5.66 $\pm 0.04$
	400	<b>19.0</b> $\pm <0.01$	4.65 $\pm 0.05$	22.3 $\pm 0.1$	6.41 $\pm 0.06$
ResNet-152	100	<b>19.2</b> $\pm <0.01$	4.69 $\pm 0.04$	20.4 $\pm <0.0$	5.39 $\pm 0.06$
	200	<b>18.5</b> $\pm 0.1$	4.37 $\pm 0.03$	20.3 $\pm 0.2$	5.39 $\pm 0.07$
	400	<b>18.4</b> $\pm <0.01$	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)



**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  $\lambda_{max}$ , 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  $\kappa$  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
<b>#Params</b>		60M		87M		59M
<b>Hessian <math>\lambda_{max}</math></b>	179.8	<b>42.0</b>	738.8	<b>20.9</b>	1644.4	<b>22.5</b>
$L_{train}$	<b>0.86</b>	0.90	<b>0.65</b>	0.82	<b>0.45</b>	0.97
$L_{train}^{\mathcal{N}}$ *	2.39	<b>2.16</b>	6.66	<b>0.96</b>	7.78	<b>1.01</b>
<b>ImageNet (%)</b>	78.5	<b>79.3</b>	74.6	<b>79.9</b>	66.4	<b>77.4</b>
<b>ImageNet-C (%)</b>	50.0	<b>52.2</b>	46.6	<b>56.5</b>	33.8	<b>48.8</b>

# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods
- 4 Introduction to Distributed Deep Learning**
  - Preliminary for Distributed Optimization
  - Federated Learning
  - Summary

# Table of Contents

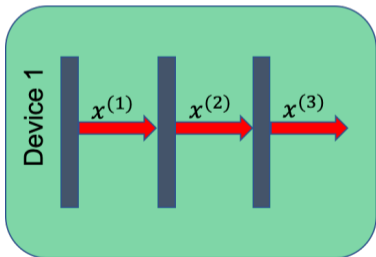
- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning
  - Preliminary for Distributed Optimization
  - Federated Learning
  - Summary

# Why Distributed Deep Learning Infrastructure Matters

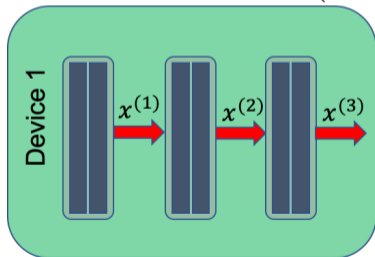
Recall that

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

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



SGD



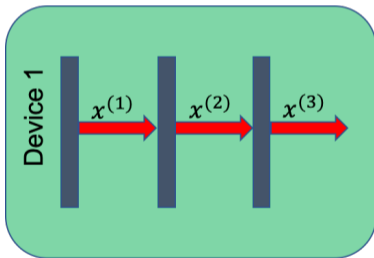
Mini-batch SGD

# Why Distributed Deep Learning Infrastructure Matters

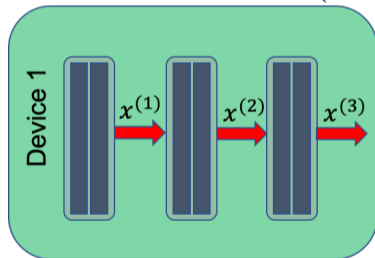
Recall that

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

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



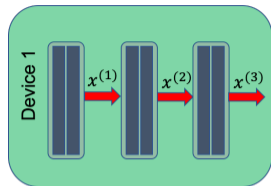
SGD



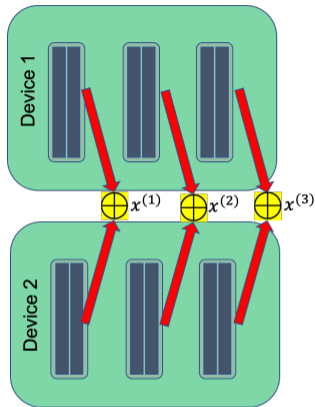
Mini-batch SGD

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

# Why Distributed Deep Learning Infrastructure Matters



Mini-batch SGD

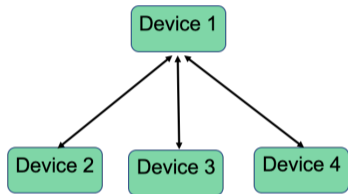


Data-parallelism

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

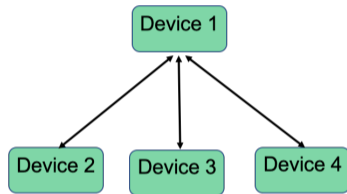


## Star/Parameter server

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

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

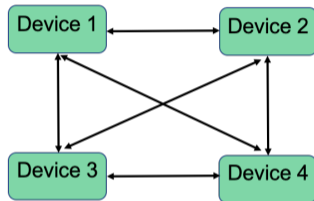
# Various Types of Distributed Learning Infrastructure



**Star/Parameter server**

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

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

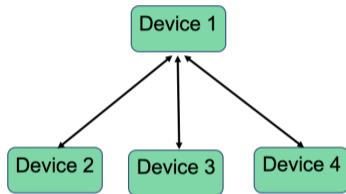


**Complete**

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

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

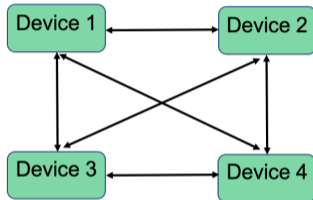
# Various Types of Distributed Learning Infrastructure



**Star/Parameter server**

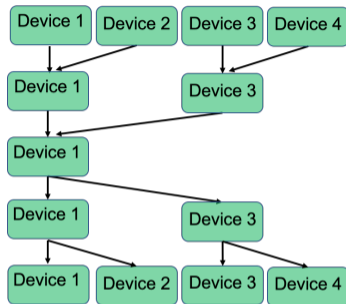
$$\mathcal{O}((t_s + t_w m)n)$$

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



**Complete**

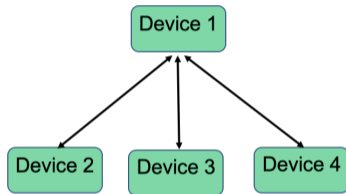
$$\mathcal{O}(t_s \log_2 n + t_w (n - 1)m)$$



**All-Reduce**

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

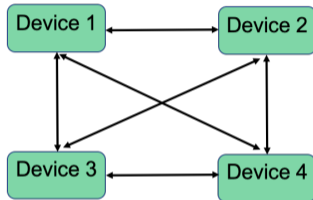
# Various Types of Distributed Learning Infrastructure



**Star/Parameter server**

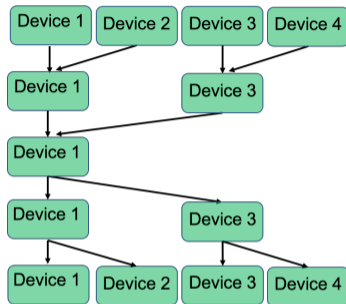
$$\mathcal{O}((t_s + t_w m)n)$$

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



**Complete**

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

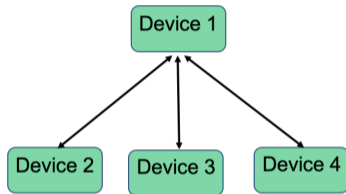


**All-Reduce**

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

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

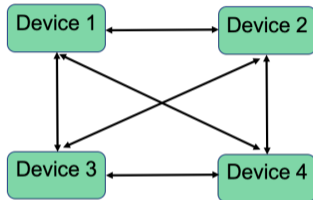
# Various Types of Distributed Learning Infrastructure



**Star/Parameter server**

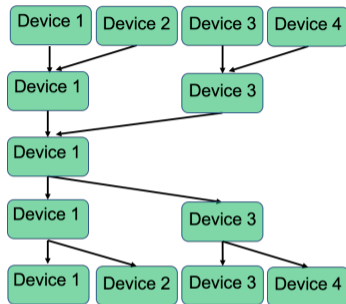
$$\mathcal{O}((t_s + t_w m)n)$$

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



**Complete**

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



**All-Reduce**

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

- *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:

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

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

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

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

- 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:

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

- 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:

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

- 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:

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

- 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:

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

where  $\eta_l$  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:

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

- 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:

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

where  $\eta_l$  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)

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

# 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.	$\searrow$

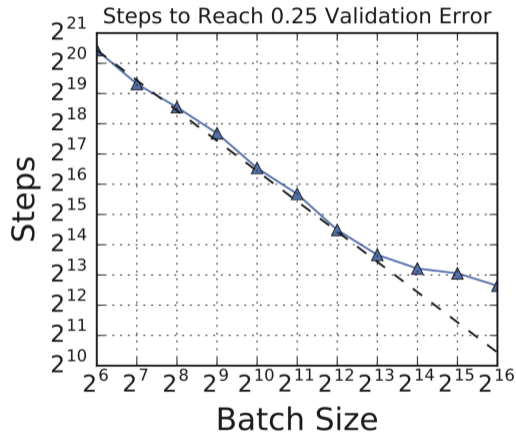
# Speed-up of Distributed Deep Learning (Data Parallelism)

**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.	$\searrow$

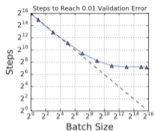
**Scaling Deep Learning training with more GPUs!**

# No Free Lunch in Distributed Deep Learning

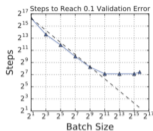


- **Limitation 1:** Diminishing returns of data parallelism with large mini-batch sizes.

# No Free Lunch in Distributed Deep Learning



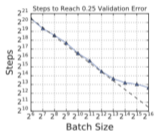
(a) Simple CNN on MNIST



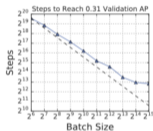
(b) Simple CNN on Fashion MNIST



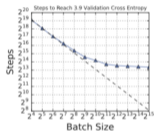
(c) ResNet-8 on CIFAR-10



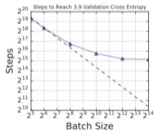
(d) ResNet-50 on ImageNet



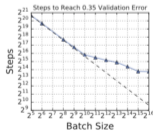
(e) ResNet-50 on Open Images



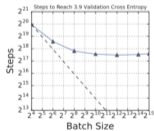
(f) Transformer on LM1B



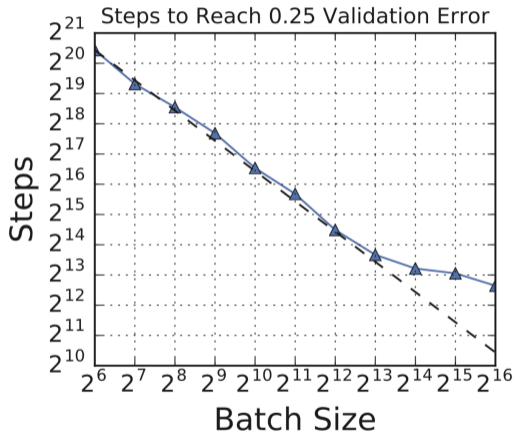
(g) Transformer on Common Crawl



(h) VGG-11 on ImageNet

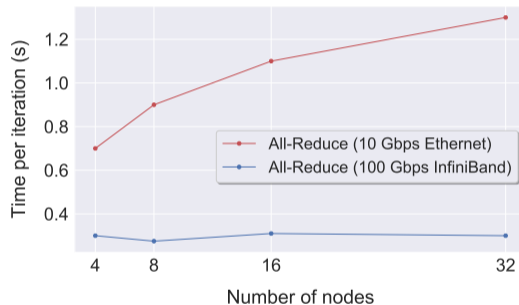


(i) LSTM on LM1B



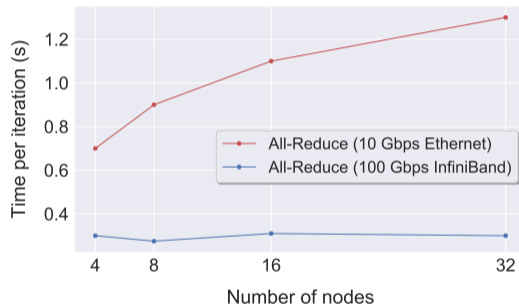
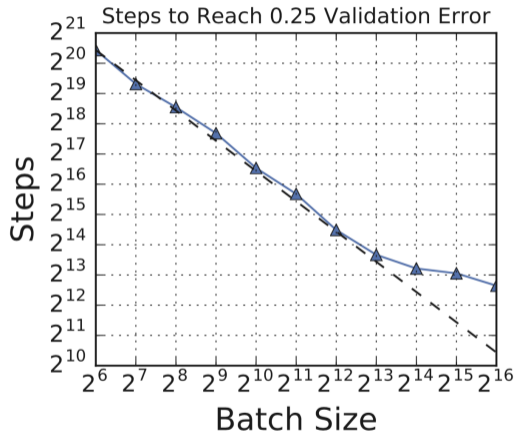
- **Limitation 1:** Diminishing returns of data parallelism with large mini-batch sizes.

# 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

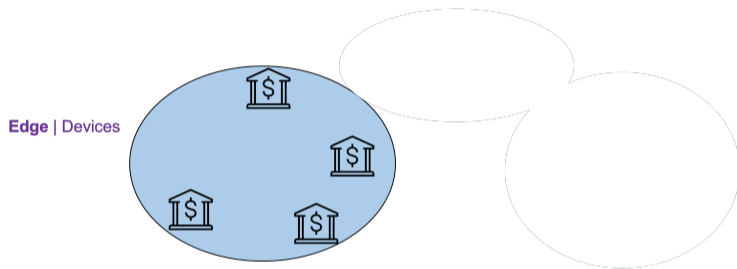
- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning
  - Preliminary for Distributed Optimization
  - **Federated Learning**
  - Summary

All previous aspects are about efficiency!

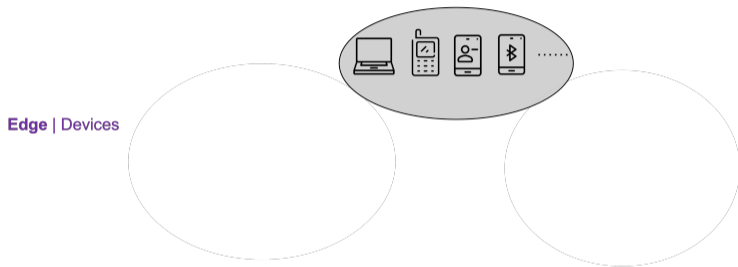
What if the privacy is a concern?

# Where does ML data come from?

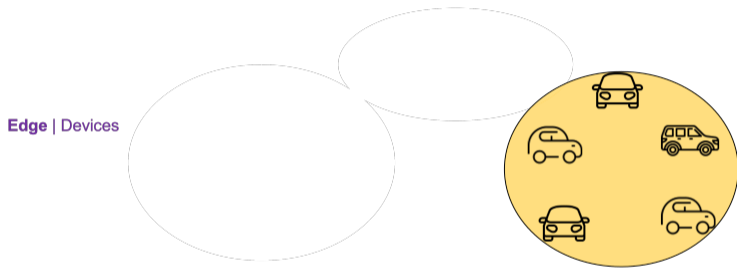
# Where does ML data come from?



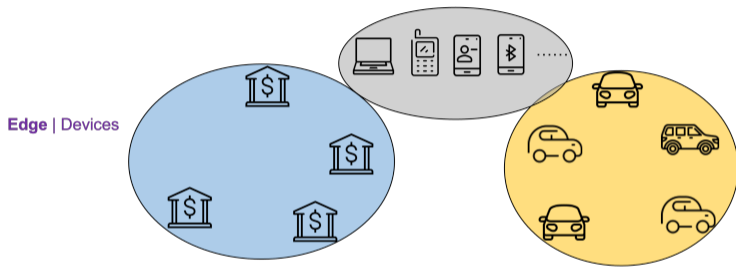
# Where does ML data come from?



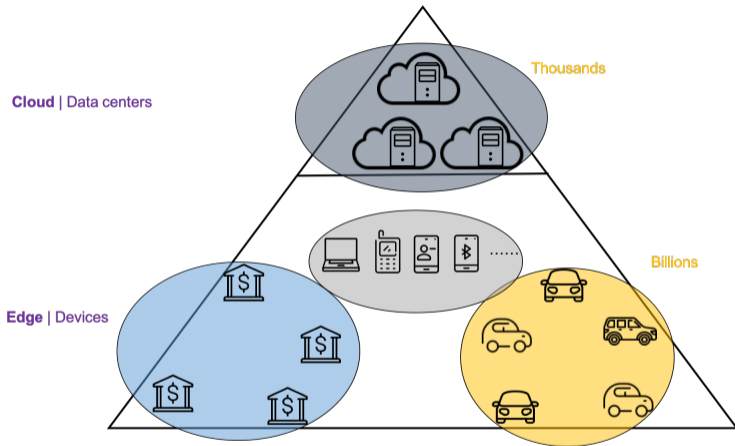
# Where does ML data come from?



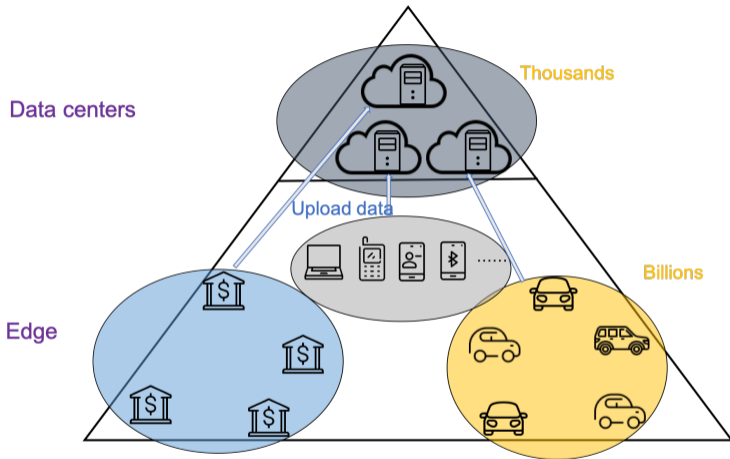
# Where does ML data come from?



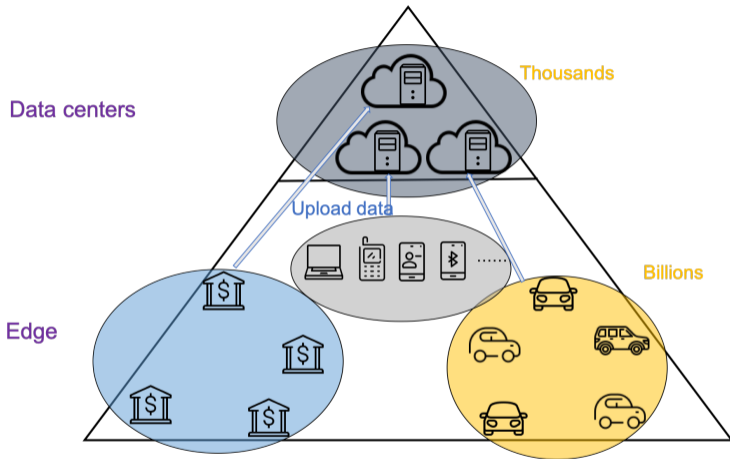
# Where does ML data come from?



# Where does ML data come from?

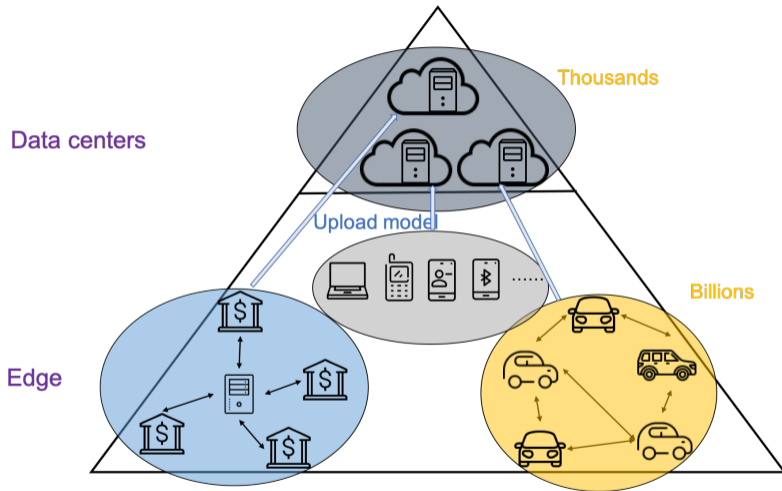


# Where does ML data come from?



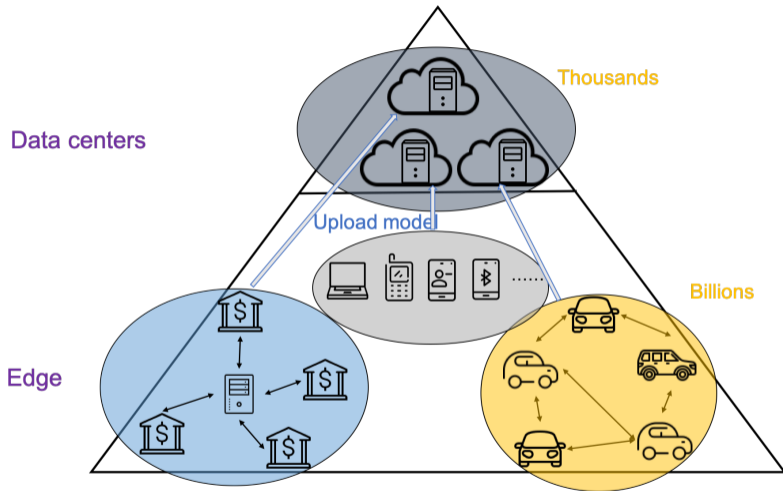
Concerns, e.g., data quality and data privacy, are rising!

# Collaborative learning alleviates the *data privacy* concern



An example: Federated Learning (FL)

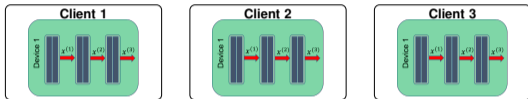
# Collaborative learning alleviates the *data privacy* concern



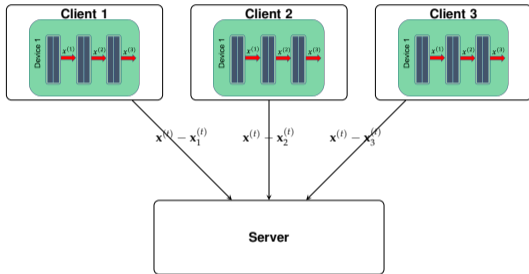
An example: Federated Learning (FL)

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

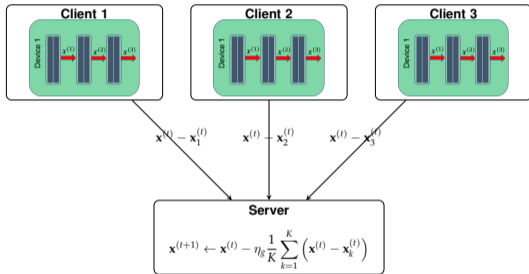
# The backbone of FL: Federated Averaging (FedAvg)



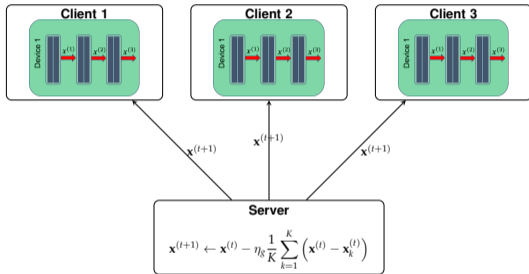
# The backbone of FL: Federated Averaging (FedAvg)



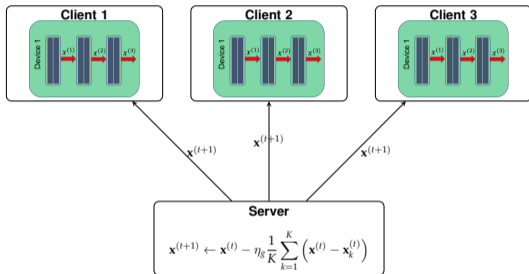
# The backbone of FL: Federated Averaging (FedAvg)



# The backbone of FL: Federated Averaging (FedAvg)



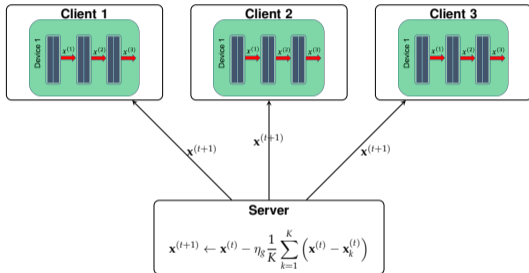
# The backbone of FL: Federated Averaging (FedAvg)



Finite-sum empirical risk minimization problem:

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

# The backbone of FL: Federated Averaging (FedAvg)

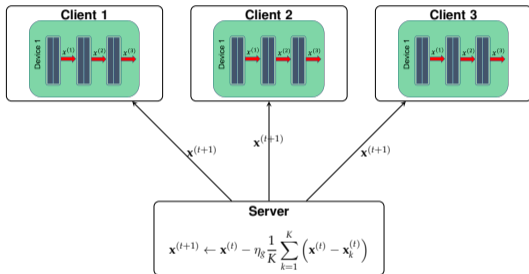


Finite-sum empirical risk minimization problem:

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

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

# The backbone of FL: Federated Averaging (FedAvg)

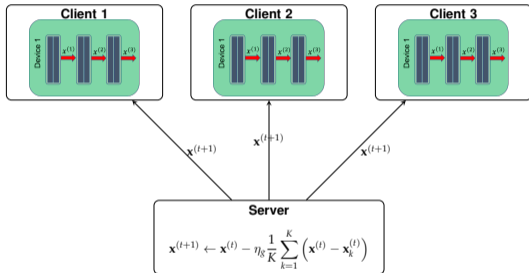


Finite-sum empirical risk minimization problem:

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

- 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)

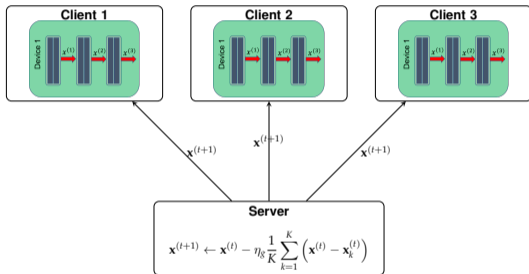


Finite-sum empirical risk minimization problem:

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

- 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:

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

- 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

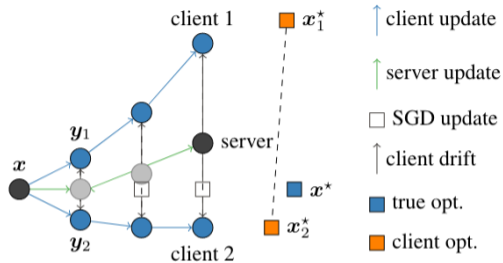
---

[13] *Li et al.* Federated Learning: Challenges, Methods, and Future Directions. 2020.

[4] *Kairouz et al.* Advances and open problems in federated learning. 2021.

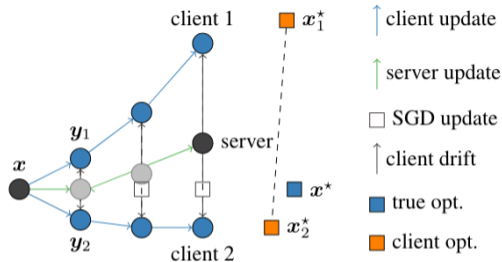
[23] *Wang et al.* A Field Guide to Federated Optimization. 2021.

# Data heterogeneity in FL



**Client drift** issue defined in [5].

# Data heterogeneity in FL



**Client drift** issue defined in [5].

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

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

# Table of Contents

- 1 Stochastic Gradient Descent (SGD) and Mini-batch SGD
- 2 Accelerated and Stabilized Optimization Methods
- 3 Advanced Optimization Methods
  - Lookahead
  - Sharpness-aware Minimization
- 4 Introduction to Distributed Deep Learning
  - Preliminary for Distributed Optimization
  - Federated Learning
  - Summary

- 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. Karimireddy, 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.