AI Summer School

Stochastic Optimization for Large-Scale Machine Learning

I-Hsiang Wang
National Taiwan University
ihwang@ntu.edu.tw

2019.08.13 @NTU

ML: from Data to Intelligence

$$\begin{tikzcd}\text{data} \arrow[r] & \boxed{\text{Machine Learning}} \arrow[r] & \text{AI}\end{tikzcd}$$

"AI" = "skills"
in most cases

- Many successful applications needless to say …

- How well a machine can learn depends on many factors:

Amount of data	How much data should be used?
Feature selection	What features should be chosen?
Model selection	What learning model should we use?
Training algorithm	How to train the learning model?

- Focus today: *training* large-scale ML models with big data

Abstraction of ML (1/3)

Data generated from a black box

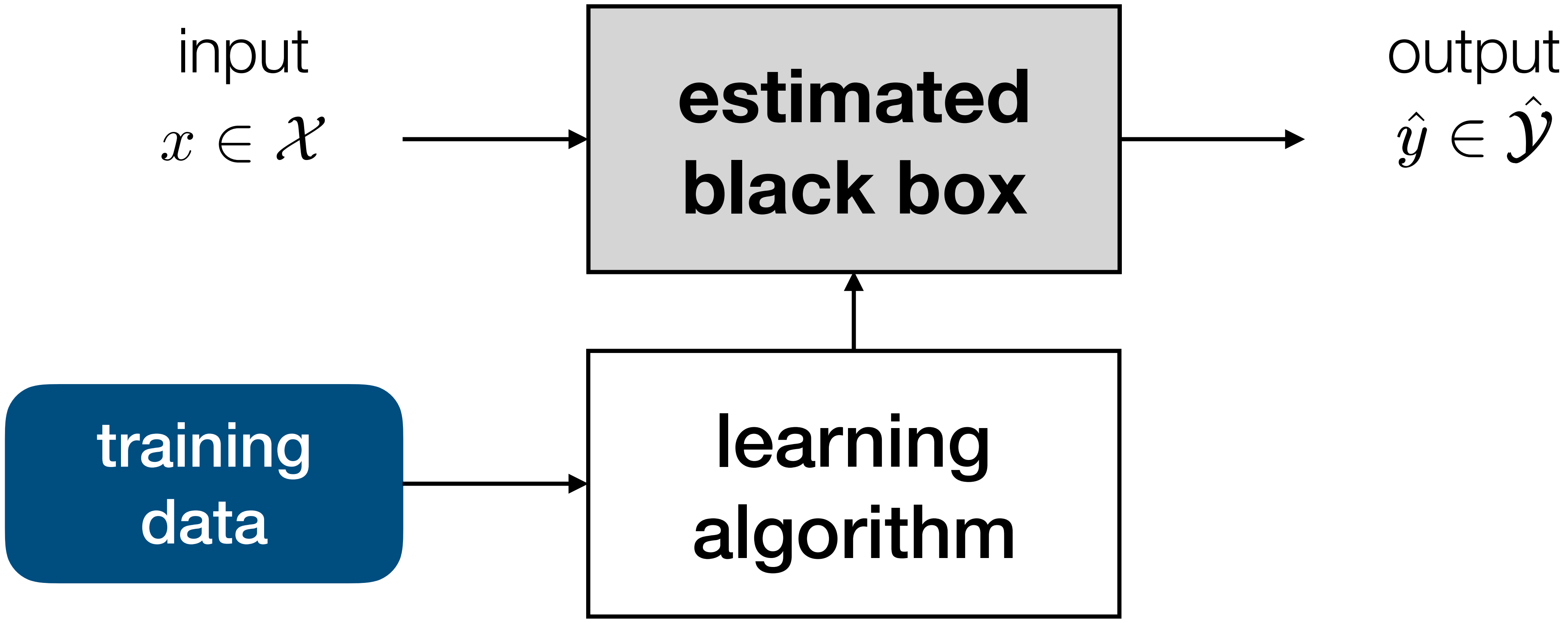
$$\begin{tikzcd}
\substack{\text{input} \\ x \in \mathcal{X}} \arrow[r] & \boxed{\textbf{black box}} \arrow[r] & \substack{\text{output} \\ y \in \mathcal{Y}}
\end{tikzcd}$$

*supervised learning

keywords in an article	topic
image of a handwritten digit	the digit
fMRI image	status of disease
text of a message	spam or not

Abstraction of ML (2/3)

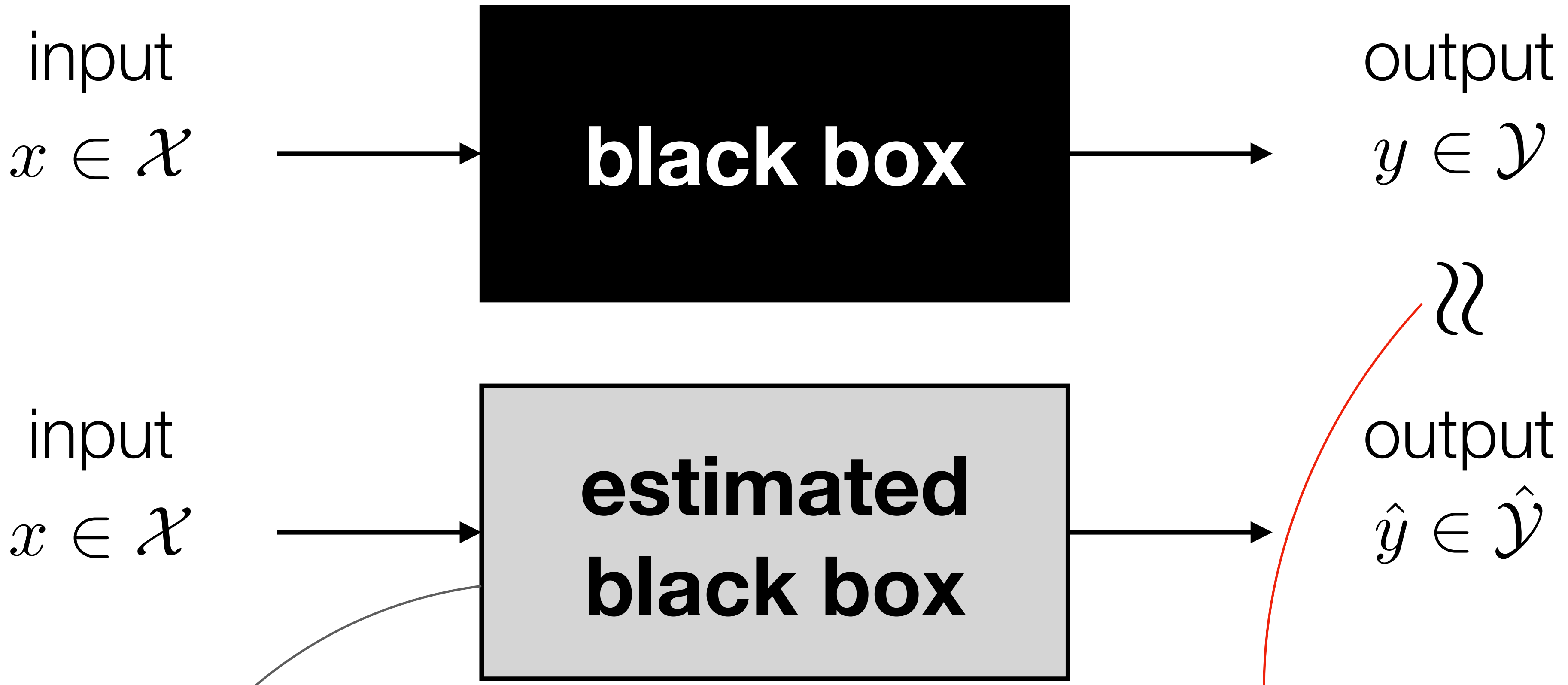
But we do not completely know how the black box works



so we try to learn how it works from **data**!

Abstraction of ML (3/3)

Goal: the estimated box can predict the actual output well.



belongs to some *learning model* parametrized by $w \in \mathcal{W}$

how to quantify how good the prediction is?

Quantifying the Effectiveness

- Loss function: $\ell : \hat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R},\ (\hat{y}, y) \mapsto \ell(\hat{y}, y)$
    - Quantify the cost of predicting $\hat{y}$ when the actual outcome is y

- Examples:
    - 0-1 loss (classification) $\ell(\hat{y}, y) = \mathbb{1}\{\hat{y} \neq y\}$
    - hinge loss (binary classification)

      $$\ell(\hat{y}, y) = (1 - \hat{y}y)^+ \quad \mathcal{Y} = \{\pm 1\},\ \hat{\mathcal{Y}} = \mathbb{R}$$
    - cross entropy loss (prediction/classification)

      $$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{d} y_i \log \hat{y}_i \qquad \mathcal{Y} = \hat{\mathcal{Y}} = \mathcal{P}_d$$
    - norm loss (regression) $\ell(\hat{y}, y) = \|\hat{y} - y\|$

Learning Model

- Typically the estimated blackbox takes the following form:
  - Raw data x is mapped to a high-dimensional feature $\boldsymbol{x} \in \mathbb{R}^{d_{\mathrm{F}}}$:

$$x \to \boxed{\phi(\cdot)} \to \phi(x) =: \boldsymbol{x} \in \mathbb{R}^{d_{\mathrm{F}}}$$

  - Feature fed to a learning model to produce an output

$$\boldsymbol{x} \to \boxed{h(\cdot\,;\boldsymbol{w})} \to h(\boldsymbol{x};\boldsymbol{w}) =: \hat{y}$$

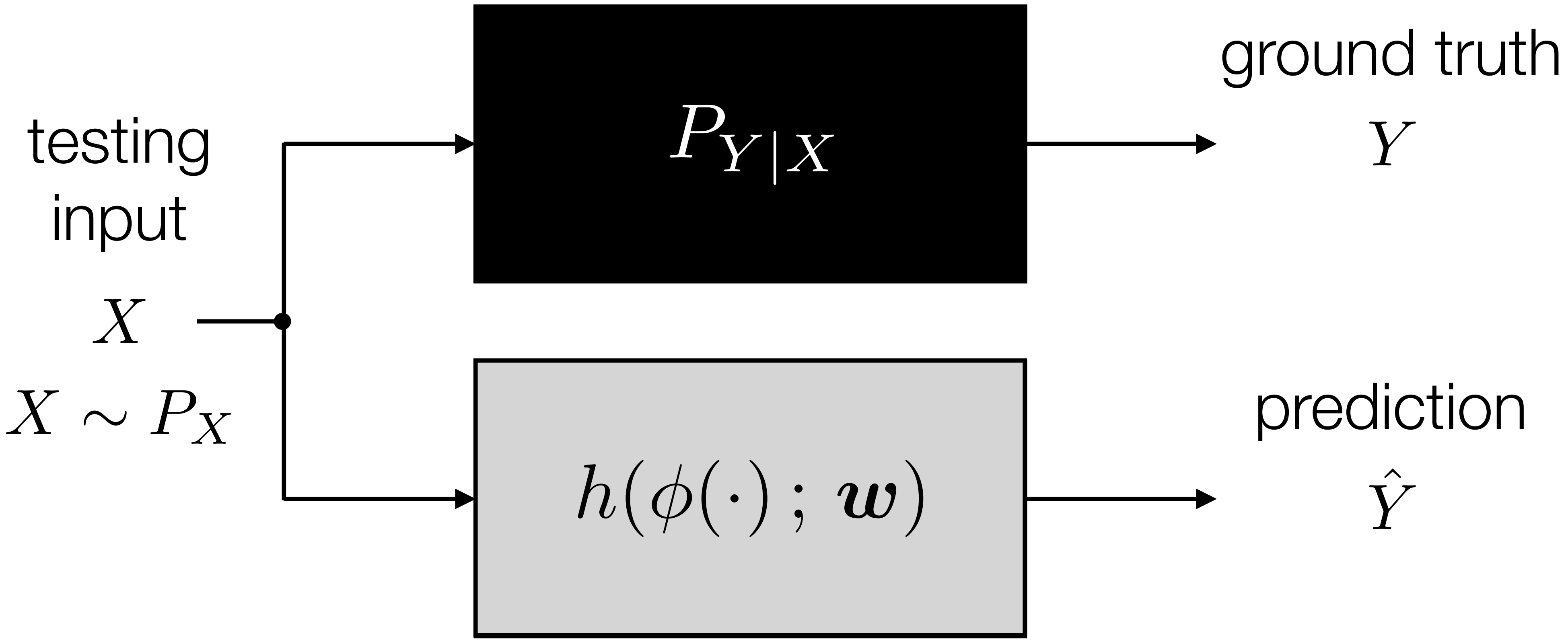
  - The model is configured by parameter $\boldsymbol{w} \in \mathbb{R}^{d_{\mathrm{M}}}$

- The simplified goal of ML: finding a "good" $\boldsymbol{w} \in \mathbb{R}^{d_{\mathrm{M}}}$

- How to define "good"?

Expected Loss (Statistical Risk)

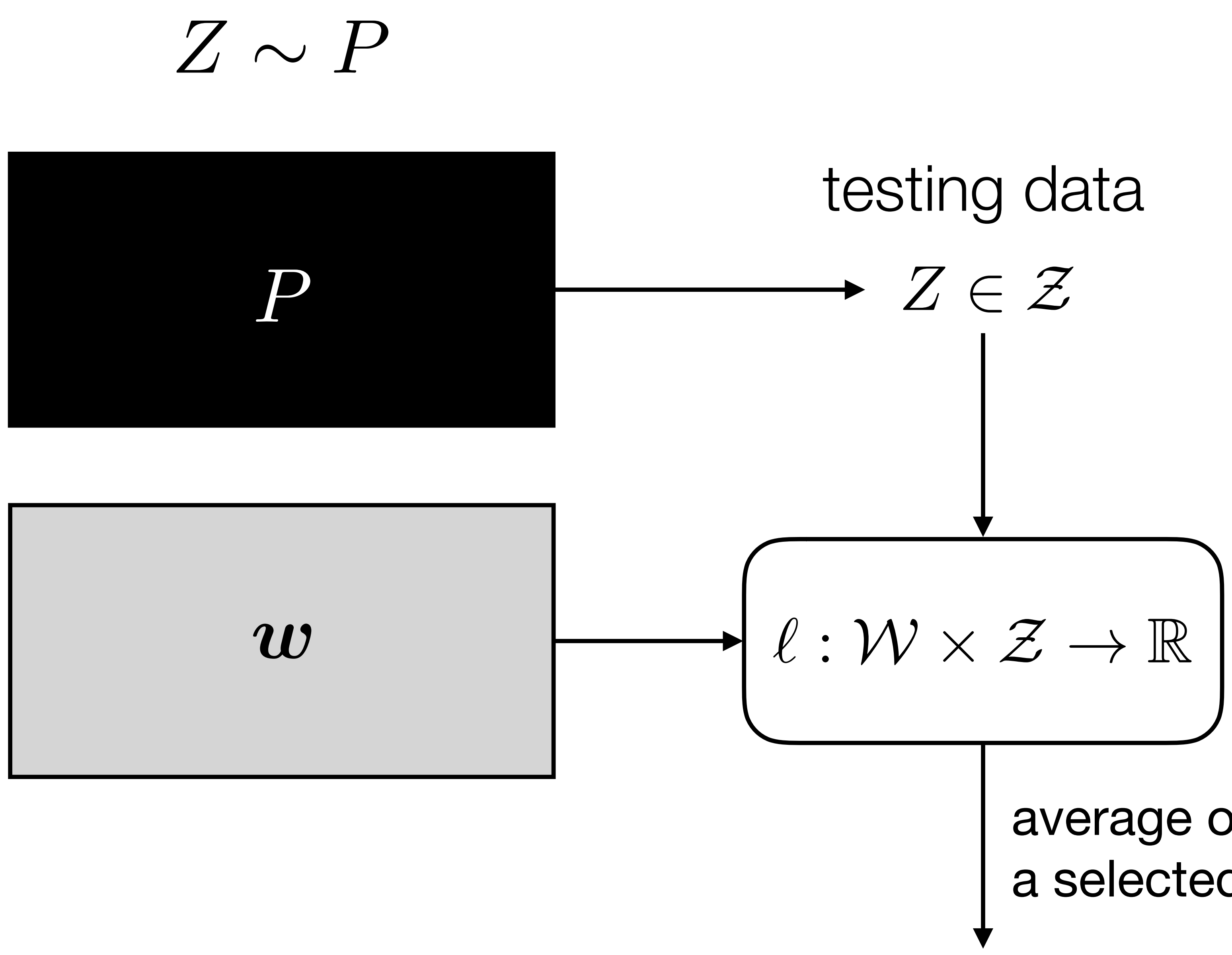
$$(X, Y) \sim P \equiv P_{X,Y} = P_{Y|X} P_X$$



$$L(\boldsymbol{w}; P) \triangleq \mathsf{E}_{(X,Y)\sim P}\left[\ell(h(\phi(X); \boldsymbol{w}), Y)\right]$$

It quantifies how well the selected $\boldsymbol{w} \in \mathbb{R}^{d_{\mathrm{M}}}$ performs *on the average*

General Learning Framework



Correspondence

(X, Y)	$\longleftrightarrow$	Z
$\ell(h(\phi(X); \boldsymbol{w}), Y)$	$\longleftrightarrow$	$\ell(\boldsymbol{w}, Z)$

$$L(\boldsymbol{w}; P) \triangleq \mathsf{E}_{Z \sim P} \left[\ell(\boldsymbol{w}, Z) \right]$$

Risk Minimization is Challenging

- Simplified goal of ML: find the best $\boldsymbol{w} \in \mathcal{W}$ by solving an optimization problem:

$$\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w}} \{L(\boldsymbol{w}; P) \,|\, \boldsymbol{w} \in \mathcal{W}\}$$

- Unfortunately ill-posed since ground truth P is unknown
- Training data come to rescue –

  approximate the statistical risk by the empirical risk.

statistical risk $L(\boldsymbol{w}; P) \triangleq \mathsf{E}_{Z\sim P}\left[\ell(\boldsymbol{w}, Z)\right]$

empirical risk $\hat{L}(\boldsymbol{w}; \underbrace{z_1, \ldots, z_n}_{\text{training data}}) \triangleq \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{w}, z_i)$

- Computationally challenging if n is huge.

Stochastic Optimization

$$\text{minimize}_{\boldsymbol{w}\in\mathbb{R}^d} \quad \underbrace{F(\boldsymbol{w}) = \mathsf{E}_Z\left[f(\boldsymbol{w};Z)\right]}_{\text{statistical risk, empirical risk, etc.}}$$

- Z: randomness in the problem
  - Statistical risk: $Z \sim P$ unknown distribution
  - Empirical risk: $Z \sim \hat{P}_{z_1,\dots,z_n} = \frac{1}{n}\sum_{i=1}^{n}\delta_{z_i}$ empirical distribution

- Evaluation of the objective function, its gradient, Hessian, etc., can be quite challenging, because:
  - The distribution may be unknown
  - Even it is known, taking the expectation can be expensive

- Idea: stochastically approximate these quantities.

First-Order Methods

- In this lecture we only focus on first-order iterative optimization methods.
- Per-iteration computation cost is $O(d)$ for $\boldsymbol{w} \in \mathbb{R}^d$, because it only uses *gradient* information in each iterate.
- Second-order methods using Hessian etc. are much more expensive if the feature/model is very high-dimensional.

Stochastic Gradient Method

Stochastic Gradient Method [Robbins and Monro, 1951]:

```
Initialize:  start with $\boldsymbol{w}^{[1]}$
Update:      in each iteration $t \geq 1$:
                 1) Generate a realization of a random $\xi_t$
                 2) Compute a stochastic vector $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$
                 3) Choose a step size $\eta_t > 0$
                 4) Update $\boldsymbol{w}^{[t+1]} \leftarrow \boldsymbol{w}^{[t]} - \eta_t \boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$
Terminate:   after $\tau$ iterations, output
```

$$\boldsymbol{w}^{\mathrm{o}} \overset{\mathrm{f}}{=} \{\boldsymbol{w}^{[1]}, \ldots, \boldsymbol{w}^{[\tau+1]}\}$$

- Here $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$ is some update direction ***estimating*** the true gradient $\nabla_{\boldsymbol{w}} F(\boldsymbol{w}) = \mathsf{E}_Z \left[\nabla_{\boldsymbol{w}} f(\boldsymbol{w}; Z)\right]$

(unbiased estimate)

SGD for Risk Minimization

For machine learning problems, estimating the gradient of the (empirical) risk function can be done very naturally.

Statistical Risk

$$F(\boldsymbol{w}) \equiv L(\boldsymbol{w}; P) = \mathsf{E}_{Z \sim P}\left[\ell(\boldsymbol{w}, Z)\right]$$

Choose the random variable in SGD

$$\xi_t \overset{\text{i.i.d.}}{\sim} P$$

Although P is unknown, if there is sufficient amount of homogeneous training data, we can pick each ξ_t to be a *fresh* training sample Z_t, and get $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \leftarrow \nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}^{[t]}, Z_t)$

Empirical Risk

$$F(\boldsymbol{w}) \equiv \frac{1}{n} \sum_{i=1}^{n} \underbrace{\ell(\boldsymbol{w}, z_i)}_{f_i(\boldsymbol{w})}$$

Choose the random variable in SGD

$$\xi_t \overset{\text{i.i.d.}}{\sim} \text{Unif}\{1, \ldots, n\}$$

At each time, uniformly at random pick one out of n training samples (let ξ_t be its index) and get

$$\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \leftarrow \underbrace{\nabla f_{\xi_t}(\boldsymbol{w}^{[t]})}_{\nabla_{\boldsymbol{w}} \ell(\boldsymbol{w}^{[t]}, z_{\xi_t})}$$

Stochastic vs. Full-Batch in Training

- Training, in the context of this lecture, is equivalent to solving the *empirical risk minimization* (ERM) problem:

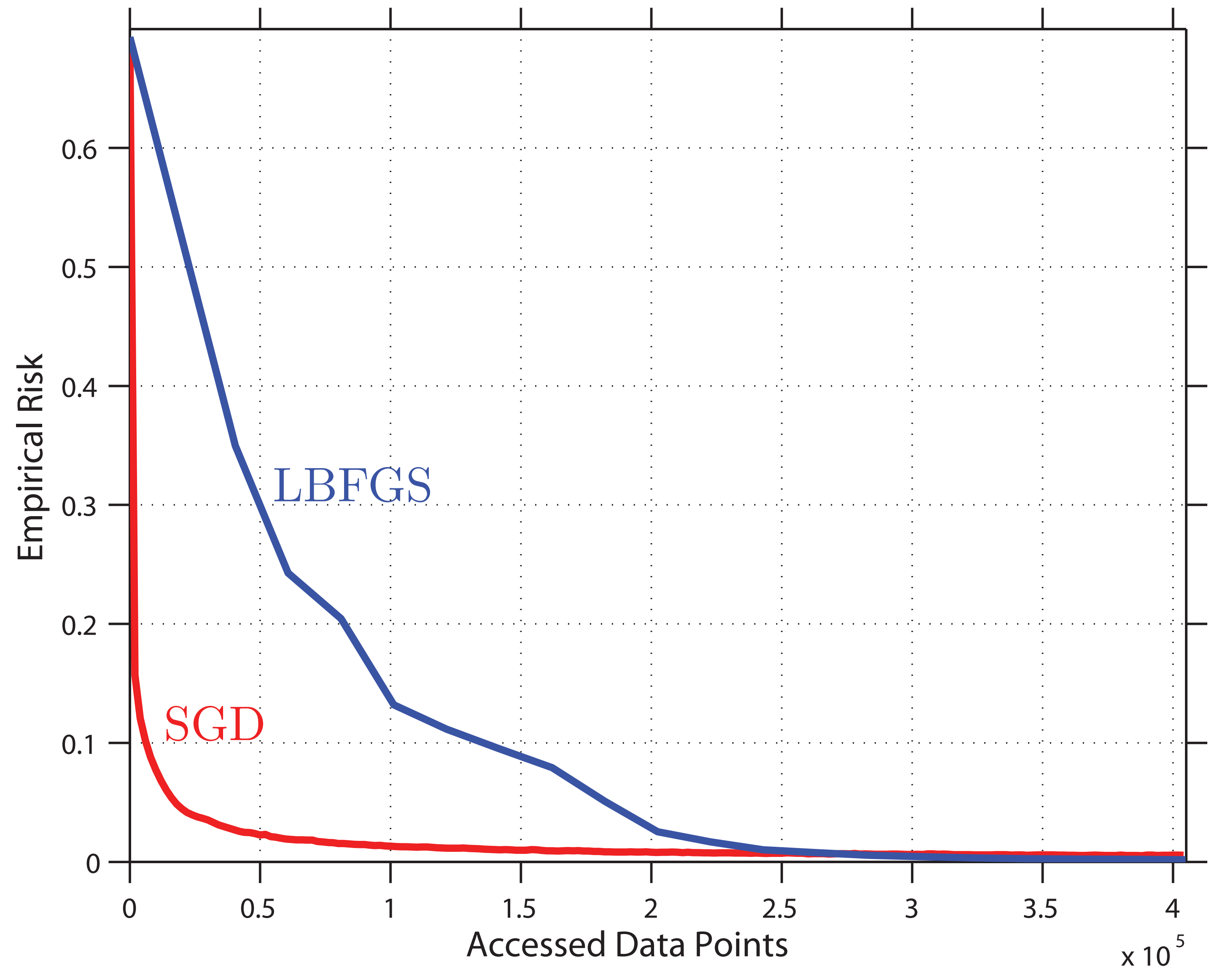
$$\text{minimize}_{\boldsymbol{w}\in\mathcal{W}} \; \underbrace{\hat{L}(\boldsymbol{w}; z_1, \ldots, z_n)}_{F(\boldsymbol{w})} \triangleq \frac{1}{n}\sum_{i=1}^{n} \underbrace{\ell(\boldsymbol{w}, z_i)}_{f_i(\boldsymbol{w})}$$

  Everything is deterministic, nothing random. Why SGD?

- Baseline – gradient descent (GD): in each iteration, the update direction is the full gradient

$$\nabla F(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{w})$$

- Stochastic gradient vs. Full Gradient:
  - Much cheaper per-iteration cost.
  - Exploit the information in training data more efficiently.

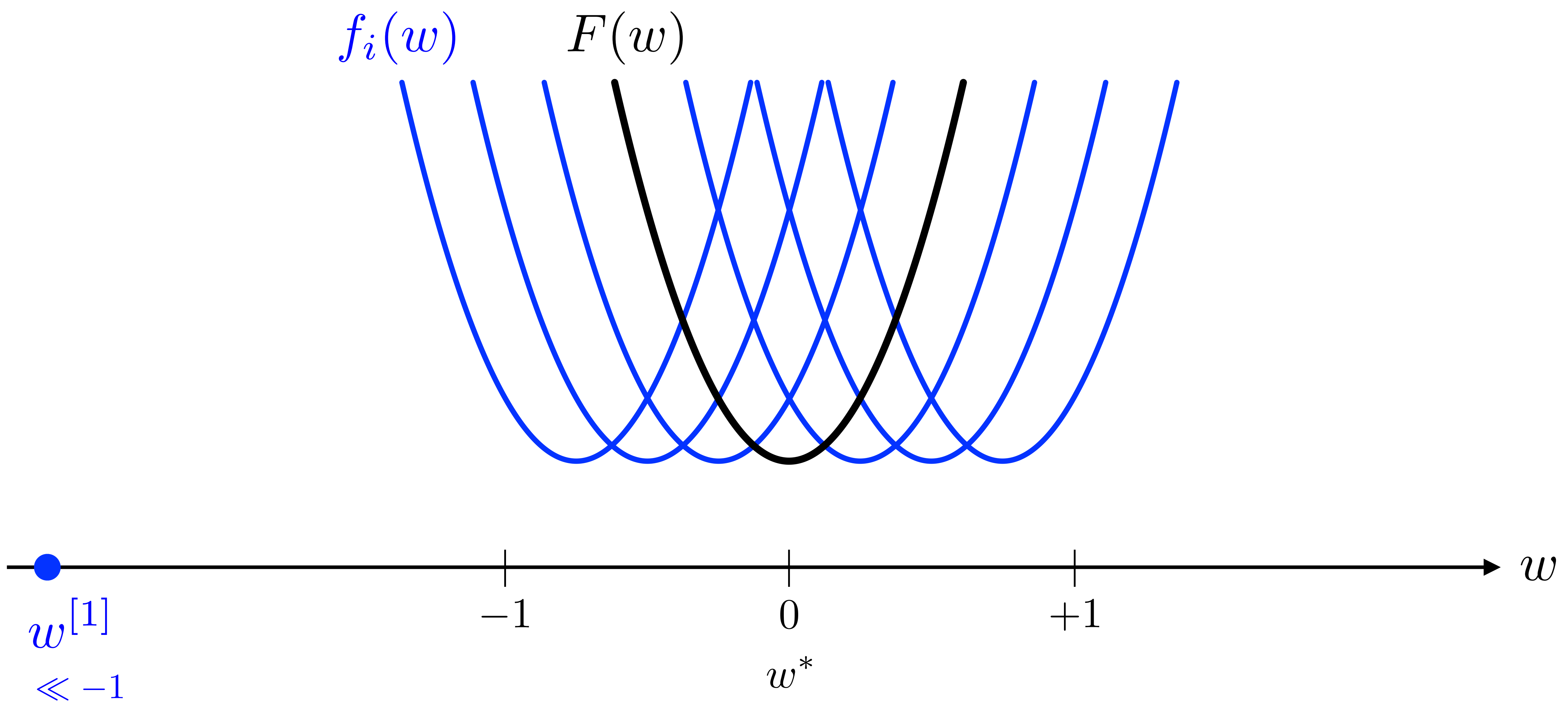


- Binary classification
- RCV1 dataset
- Logistic loss
- SGD learning rate $\eta_t = 4$
- Batch method: limited memory BFGS, a quasi-Newton method

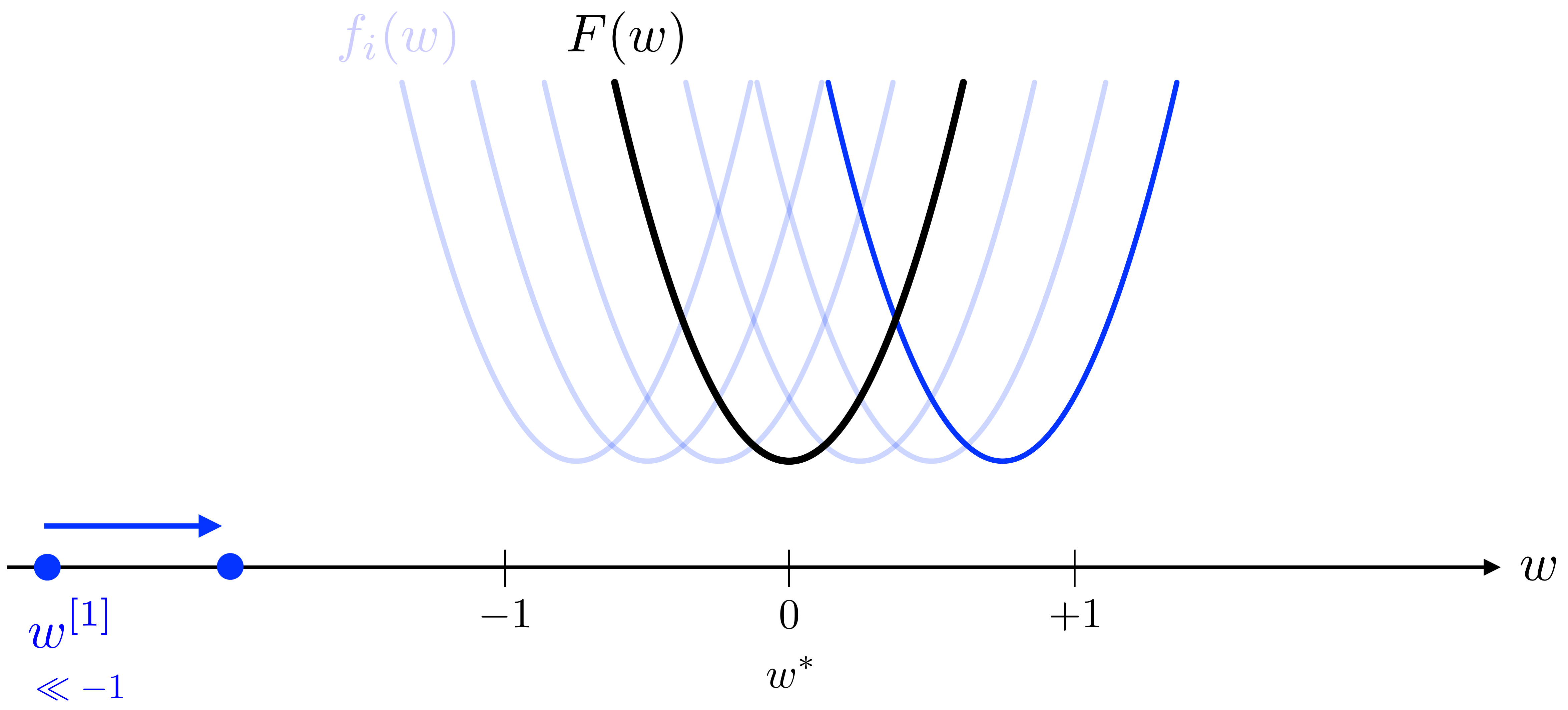
[Bottou, Curtis, and Nocedal, 2018]

- Intuition: a lot of redundancy in training data; no need to use the entire batch in each iteration.
- Empirical observation: SGD exploits the training data efficiently in the beginning, but quickly saturates.

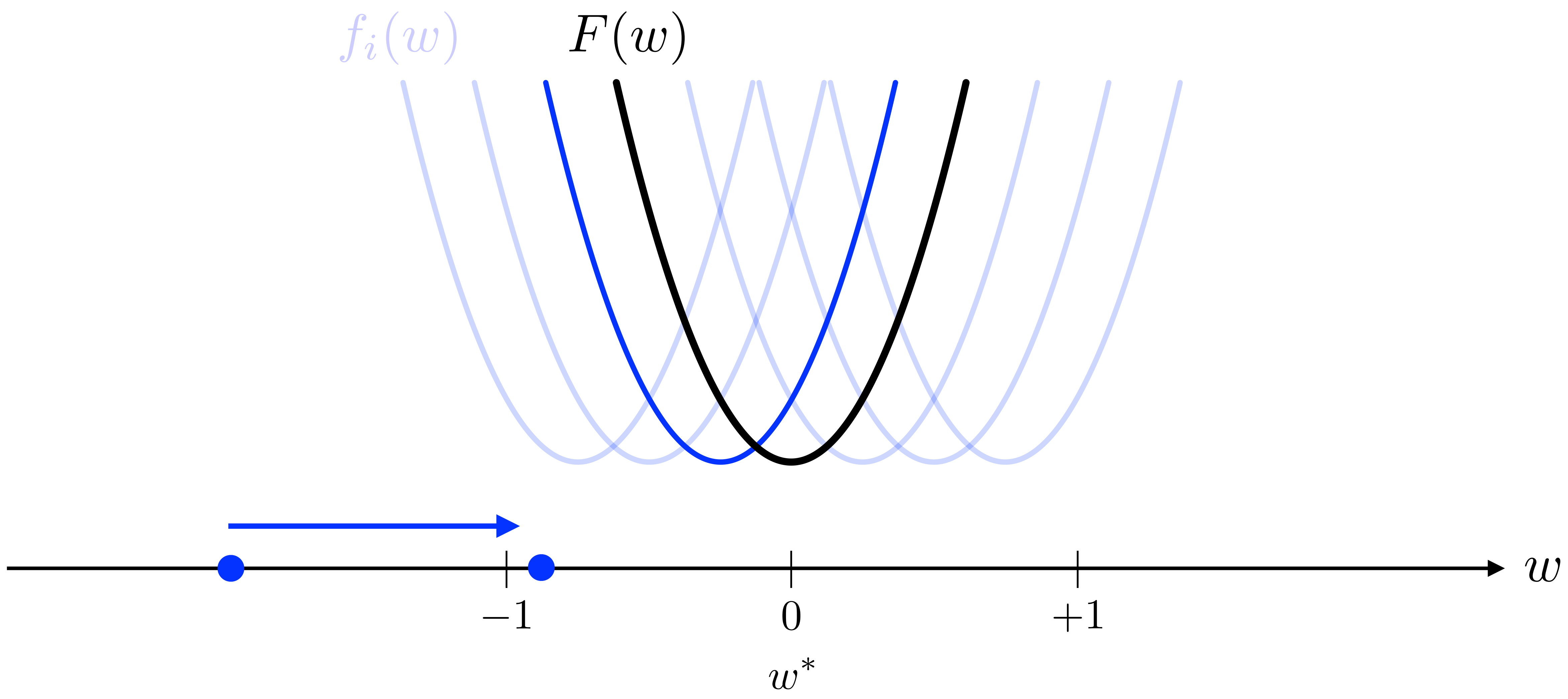
Intuitive Explanation



Intuitive Explanation

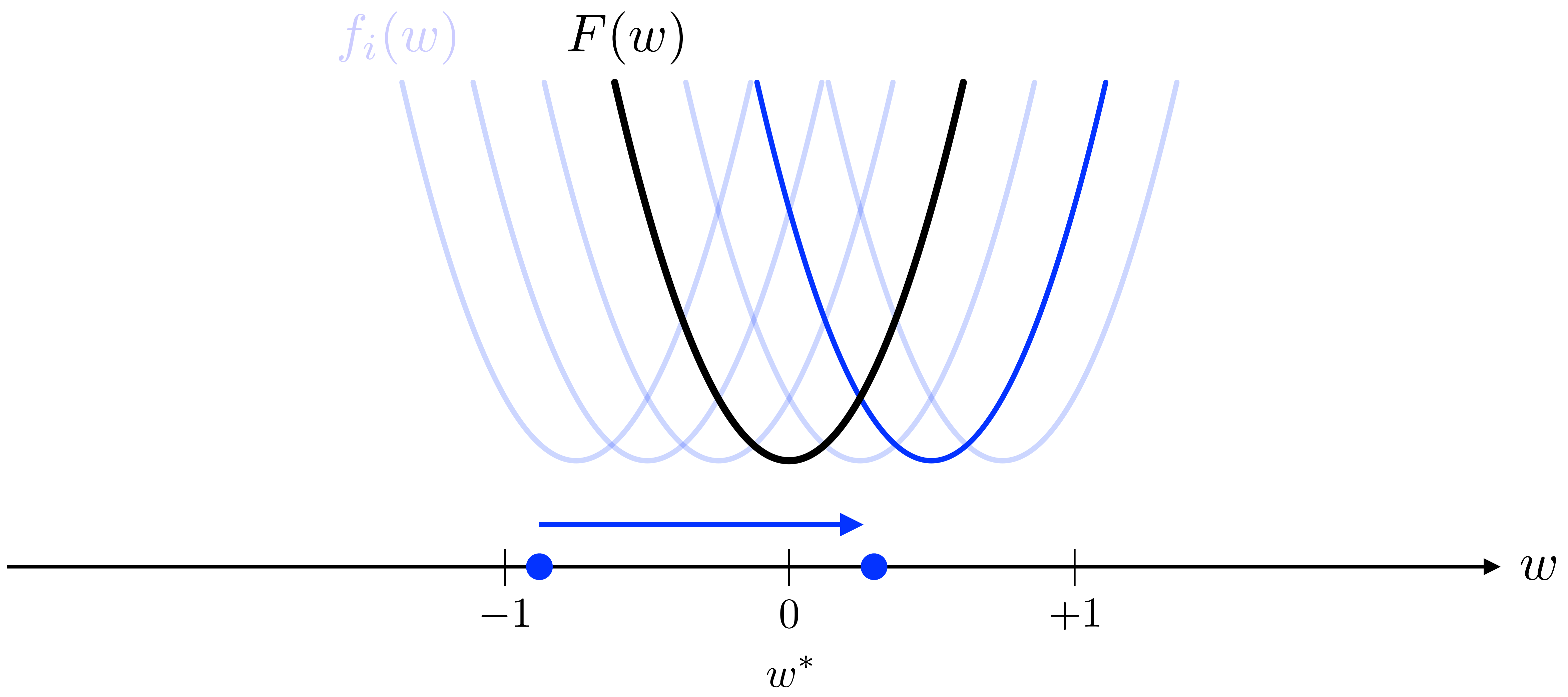


Intuitive Explanation

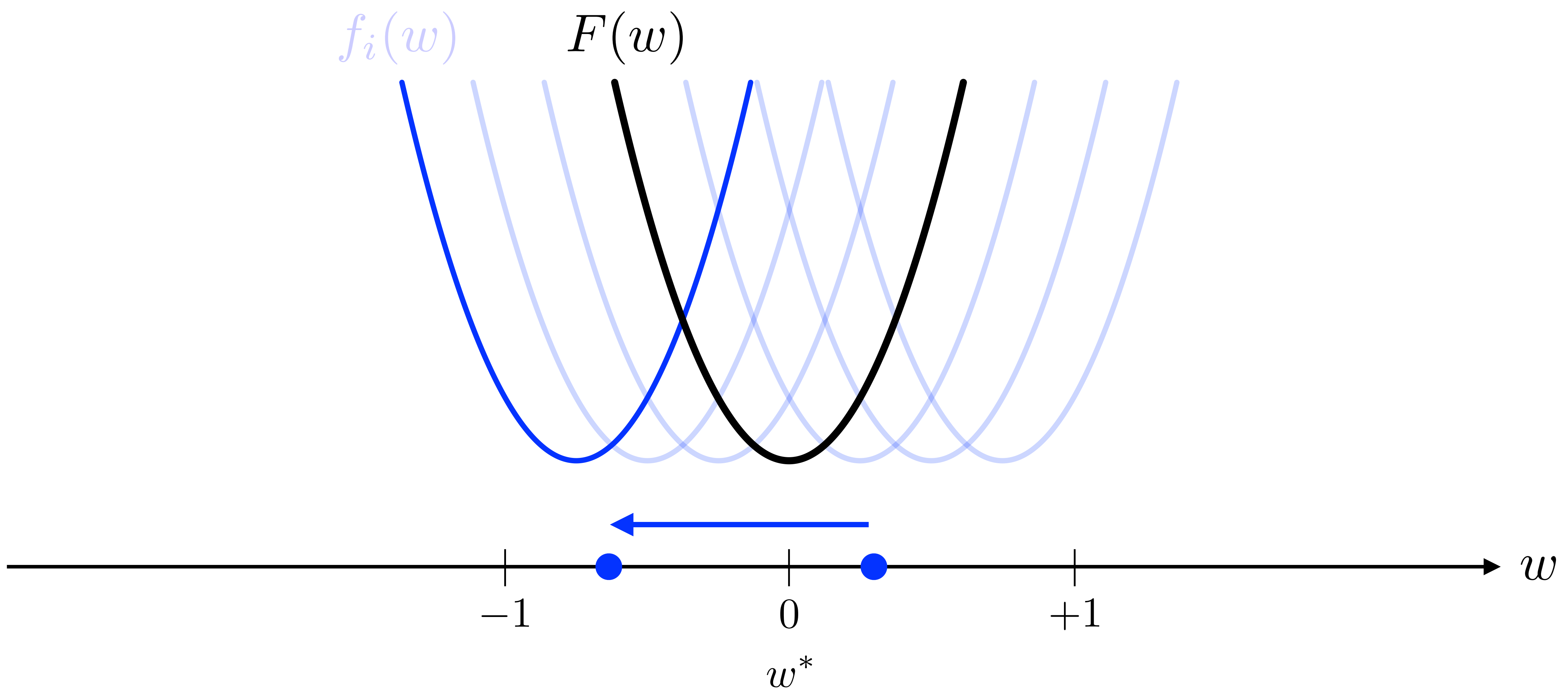


Initially, when far away from w^*, the point moves fast and the direction is consistent (with fixed step size).

Intuitive Explanation



Intuitive Explanation



When getting close to w^*, the point becomes "confused" and moves to the optimum very slowly (with fixed step size).

Emerging Questions

- Convergence guarantees of SGD?
- How does it depend on the problem structure?
- How does it depend on how well the stochastic gradient estimates the true gradient?
- Better choice of step sizes (learning rate)?
- Acceleration?

Next: some theoretical results to answer these questions in a rigorous and systematic way.

Begin with *convex* problems.

Why Convexity

Without further structural conditions, it is hard to obtain global convergence (no assumption on initialization) to global (or even local) optimum

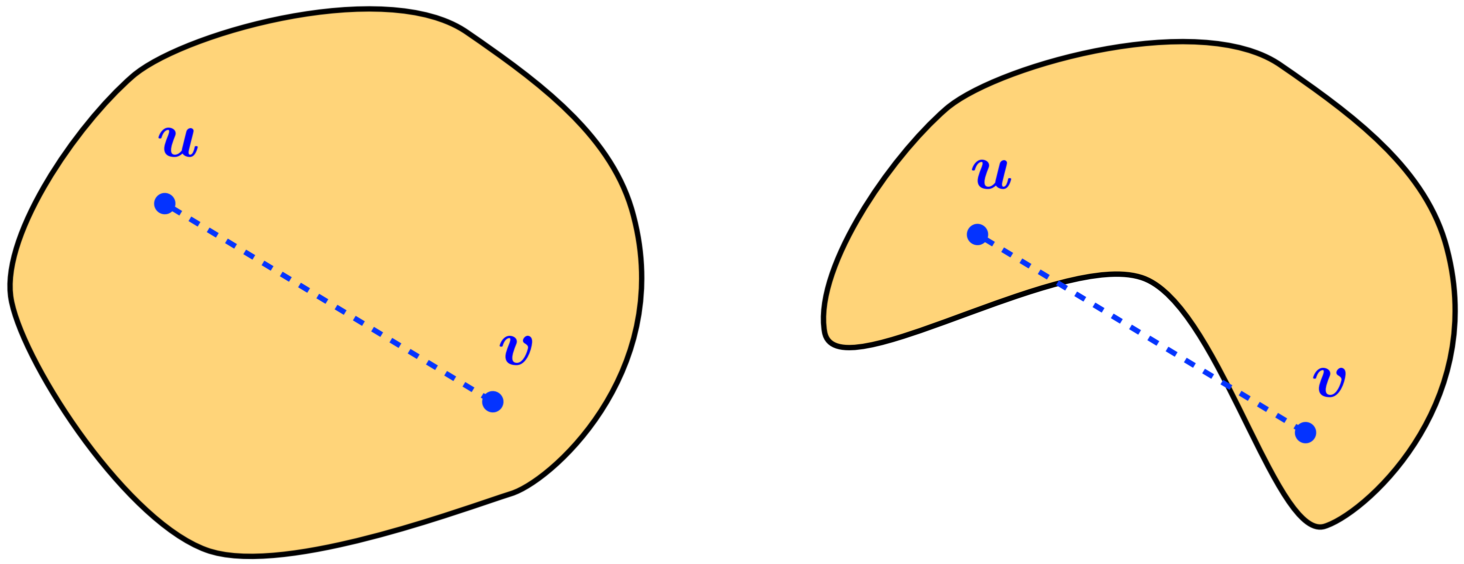
Convexity guarantees global convergence to optimum.
Local optimum = global optimum.

Note:

- Most algorithms still work without convexity
- Landscape of non-convex problems: local convexity around a local minimum

Convexity – Convex Set

- Convex set: a set $\mathcal{W} \subseteq \mathbb{R}^d$ is convex if $\forall\, \boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ and $\forall\, \lambda \in [0,1]$, $\lambda \boldsymbol{u} + (1-\lambda)\boldsymbol{v} \in \mathcal{W}$



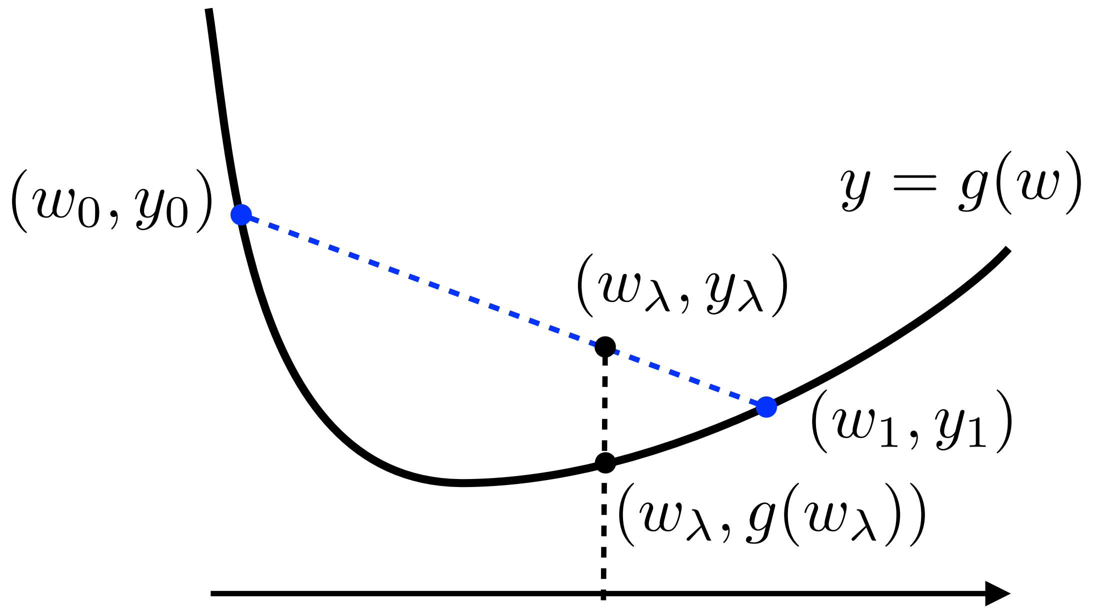
convex non-convex

Convexity – Convex Function

- Convex set: a set $\mathcal{W} \subseteq \mathbb{R}^d$ is convex if $\forall\, \boldsymbol{u}, \boldsymbol{v} \in \mathcal{W}$ and $\forall\, \lambda \in [0,1]$, $\lambda \boldsymbol{u} + (1-\lambda)\boldsymbol{v} \in \mathcal{W}$

- Convex function: a function $g : \mathcal{W} \to \mathbb{R}$ is convex if its domain is a convex set and $\forall\, \boldsymbol{w}_0, \boldsymbol{w}_1 \in \mathcal{W} \;\; \forall\, \lambda \in [0,1]$ ($\boldsymbol{w}_\lambda \triangleq (1-\lambda)\boldsymbol{w}_0 + \lambda \boldsymbol{w}_1$, $y_i \triangleq g(\boldsymbol{w}_i)$, $y_\lambda \triangleq (1-\lambda)y_0 + \lambda y_1$)

$$y_\lambda - g(\boldsymbol{w}_\lambda) \geq 0$$

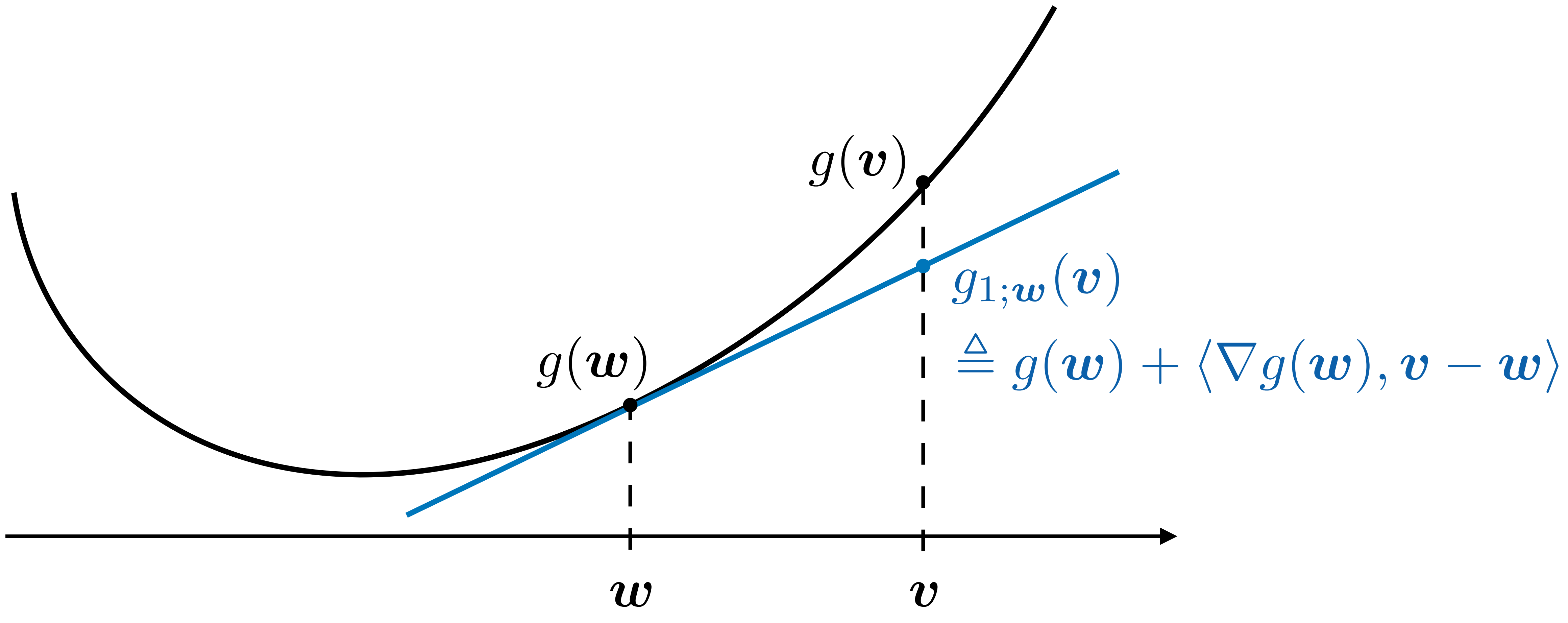


Conditions for Convexity

- For a differentiable function $g : \mathcal{W} \to \mathbb{R}$, it is convex iff

$$g(\boldsymbol{v}) - g_{1;\boldsymbol{w}}(\boldsymbol{v}) \geq 0 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W}$$ (first-order condition)

$$g_{1;\boldsymbol{w}}(\boldsymbol{v}) \triangleq g(\boldsymbol{w}) + \langle \nabla g(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle$$ (first-order approximation)


$g(\boldsymbol{v})$
$g_{1;\boldsymbol{w}}(\boldsymbol{v})$
$\triangleq g(\boldsymbol{w}) + \langle \nabla g(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle$
$g(\boldsymbol{w})$
$\boldsymbol{w}$
$\boldsymbol{v}$

Conditions for Convexity

- For a differentiable function $g : \mathcal{W} \to \mathbb{R}$, it is convex iff

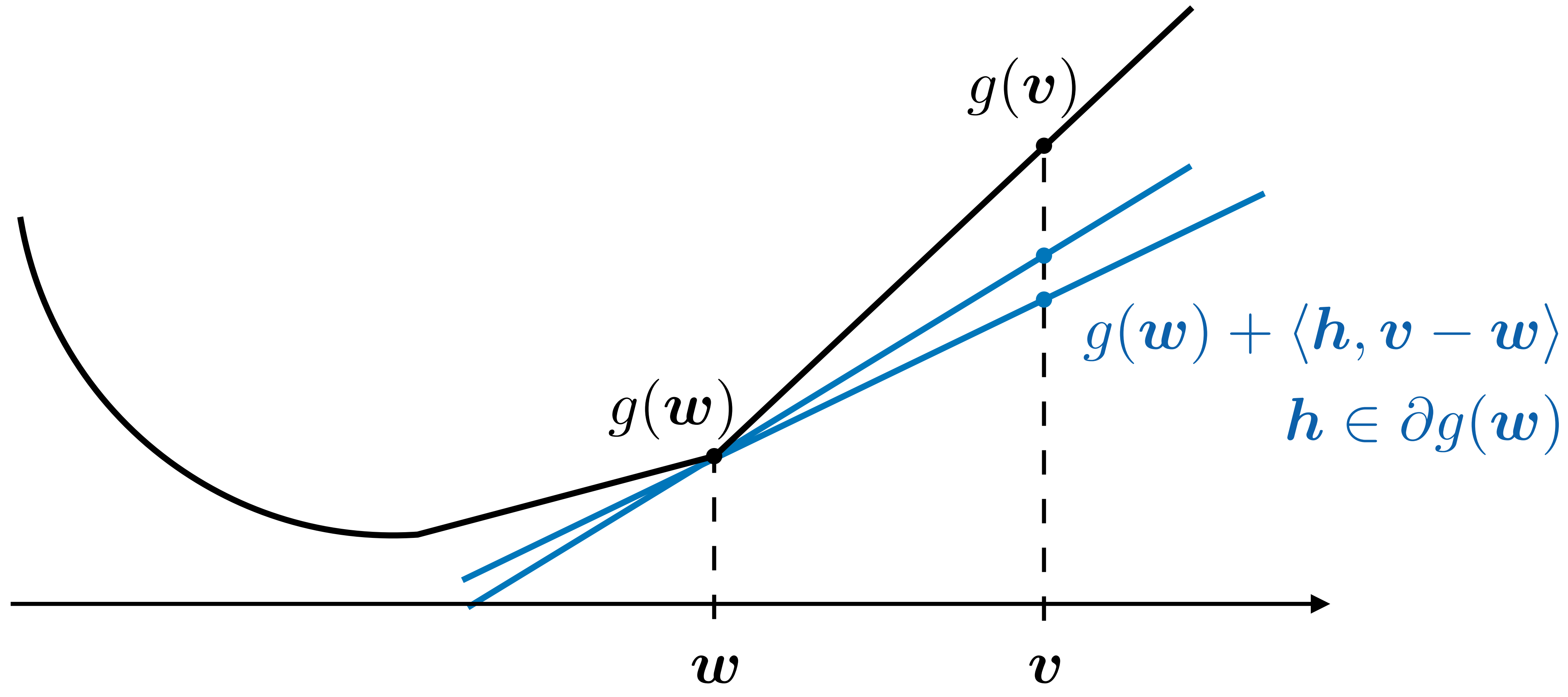
$$g(\boldsymbol{v}) - g_{1;\boldsymbol{w}}(\boldsymbol{v}) \geq 0 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W} \quad \text{(first-order condition)}$$

$$g_{1;\boldsymbol{w}}(\boldsymbol{v}) \triangleq g(\boldsymbol{w}) + \langle \nabla g(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle \quad \text{(first-order approximation)}$$

- For a general function $g : \mathcal{W} \to \mathbb{R}$, define its subdifferential at point $\boldsymbol{w}$ as

$$\partial g(\boldsymbol{w}) \triangleq \{\boldsymbol{h} : g(\boldsymbol{v}) \geq g(\boldsymbol{w}) + \langle \boldsymbol{h}, \boldsymbol{v} - \boldsymbol{w} \rangle, \ \forall\, \boldsymbol{v} \in \mathcal{W}\}$$

Function $g : \mathcal{W} \to \mathbb{R}$ is convex iff $\partial g(\boldsymbol{w}) \neq \emptyset \ \forall\, \boldsymbol{w} \in \mathcal{W}$


$g(\boldsymbol{v})$
$g(\boldsymbol{w})$
$g(\boldsymbol{w}) + \langle \boldsymbol{h}, \boldsymbol{v} - \boldsymbol{w} \rangle$
$\boldsymbol{h} \in \partial g(\boldsymbol{w})$
$\boldsymbol{w}$
$\boldsymbol{v}$

Conditions for Convexity

- For a differentiable function $g : \mathcal{W} \to \mathbb{R}$, it is convex iff

$$g(\boldsymbol{v}) - g_{1;\boldsymbol{w}}(\boldsymbol{v}) \geq 0 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W}$$ (first-order condition)

$$g_{1;\boldsymbol{w}}(\boldsymbol{v}) \triangleq g(\boldsymbol{w}) + \langle \nabla g(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w} \rangle$$ (first-order approximation)

- For a general function $g : \mathcal{W} \to \mathbb{R}$, define its subdifferential at point $\boldsymbol{w}$ as

$$\partial g(\boldsymbol{w}) \triangleq \{\boldsymbol{h} : g(\boldsymbol{v}) \geq g(\boldsymbol{w}) + \langle \boldsymbol{h}, \boldsymbol{v} - \boldsymbol{w} \rangle,\ \forall\, \boldsymbol{v} \in \mathcal{W}\}$$

Function $g : \mathcal{W} \to \mathbb{R}$ is convex iff $\partial g(\boldsymbol{w}) \neq \emptyset\ \forall\, \boldsymbol{w} \in \mathcal{W}$

- For a twice-differentiable function $g : \mathcal{W} \to \mathbb{R}$, it is convex iff its Hessian is *positive semi-definite*:

$$\nabla^2 g(\boldsymbol{w}) \succeq \mathbf{0},\ \forall\, \boldsymbol{w} \in \mathcal{W}$$ (second-order condition)

Minimizer of a Convex Function

- **Convex program** (convex optimization problem):

$$\min_{\boldsymbol{w} \in \mathcal{W}} g(\boldsymbol{w})$$

  - Objective function $g : \mathcal{W} \to \mathbb{R}$ is a convex function
  - Feasible set $\mathcal{W}$ is a convex set.

- For a differentiable convex objective function $g : \mathcal{W} \to \mathbb{R}$, the minimizer of the above can be characterized by the following first-order optimality condition:

$$\boldsymbol{w}^* \in \arg\min_{\boldsymbol{w} \in \mathcal{W}} g(\boldsymbol{w}) \iff \langle \nabla g(\boldsymbol{w}^*), \boldsymbol{w} - \boldsymbol{w}^* \rangle \geq 0, \ \forall \, \boldsymbol{w} \in \mathcal{W}$$

- Local optimum = global optimum for convex programs

Strong Convexity

Sometimes we need something stronger than convexity.

Get faster convergence, gain stability for the minimizer, etc.

- A function $g : \mathcal{W} \to \mathbb{R}$ is called α-strongly convex if $\forall\, \boldsymbol{w}_0, \boldsymbol{w}_1 \in \mathcal{W} \quad \forall\, \lambda \in [0,1]$,

$$y_\lambda - g(\boldsymbol{w}_\lambda) \geq \frac{\alpha}{2}\lambda(1-\lambda)\left\|\boldsymbol{w}_0 - \boldsymbol{w}_1\right\|^2$$

- First-order condition:

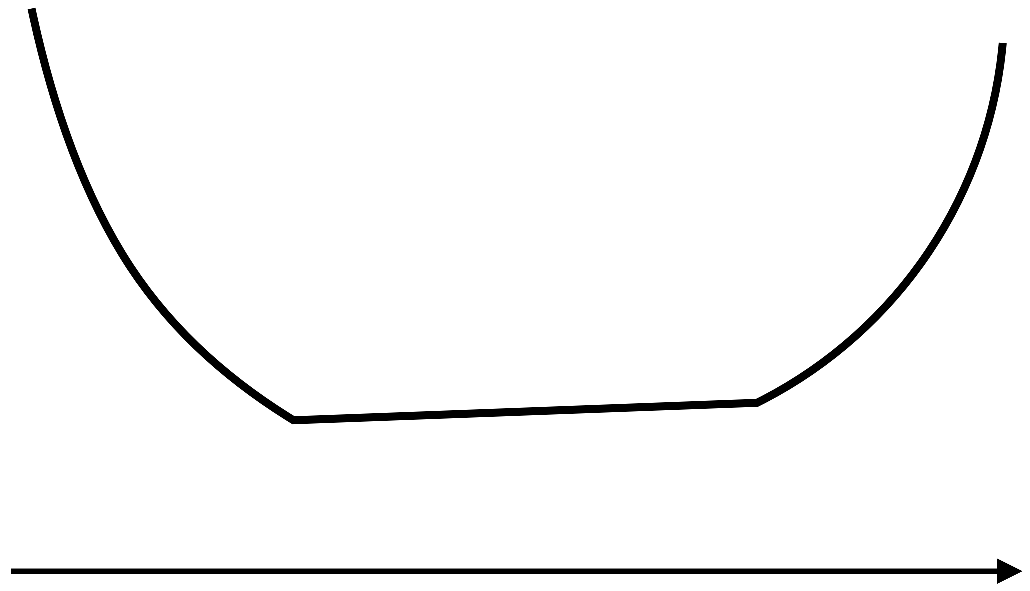
$$g(\boldsymbol{v}) - g_{1;\boldsymbol{w}}(\boldsymbol{v}) \geq \frac{\alpha}{2}\left\|\boldsymbol{v} - \boldsymbol{w}\right\|^2 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W}$$

- Second-order condition: $\nabla^2 g(\boldsymbol{w}) \succeq \alpha \mathbf{I}_d, \ \forall\, \boldsymbol{w} \in \mathcal{W}$

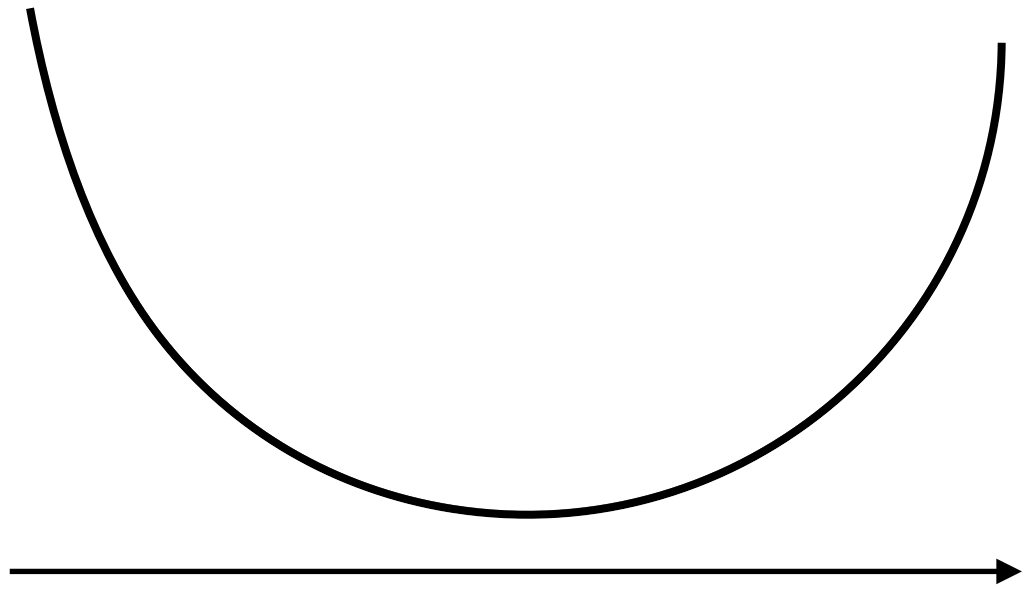
- Minimizer of α-strongly convex function (say, $\boldsymbol{w}^*$) satisfies

$$g(\boldsymbol{w}) - g(\boldsymbol{w}^*) \geq \frac{\alpha}{2}\left\|\boldsymbol{w} - \boldsymbol{w}^*\right\|^2, \ \forall\, \boldsymbol{w} \in \mathcal{W}$$

Convex



Strongly Convex



Smoothness

- If a function has a Lipschitz continuous gradient, it must be fairly smooth. The degree of smoothness is governed by the Lipschitz constant of the gradient.

- A differentiable function $g : \mathcal{W} \to \mathbb{R}$ is β-smooth if

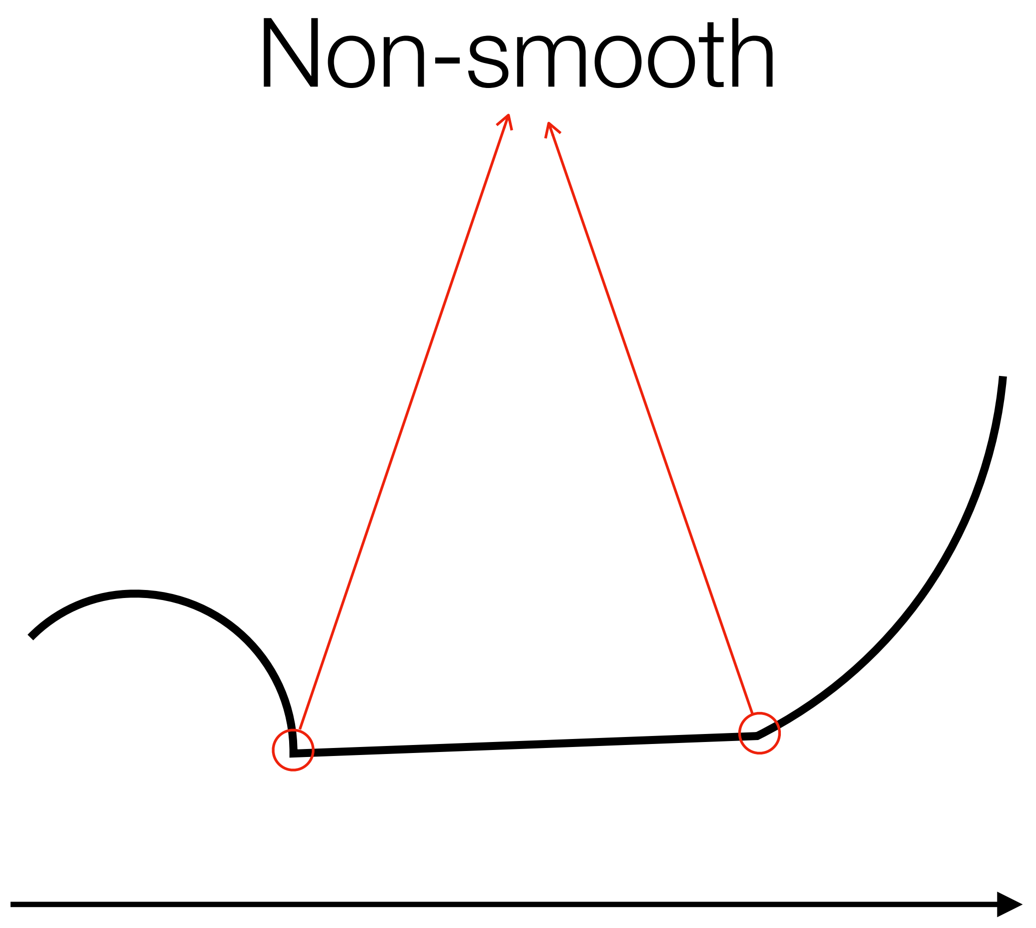
$$\langle \nabla g(\boldsymbol{w}) - \nabla g(\boldsymbol{v}), \boldsymbol{w} - \boldsymbol{v} \rangle \leq \beta \left\| \boldsymbol{w} - \boldsymbol{v} \right\|^2 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W}$$

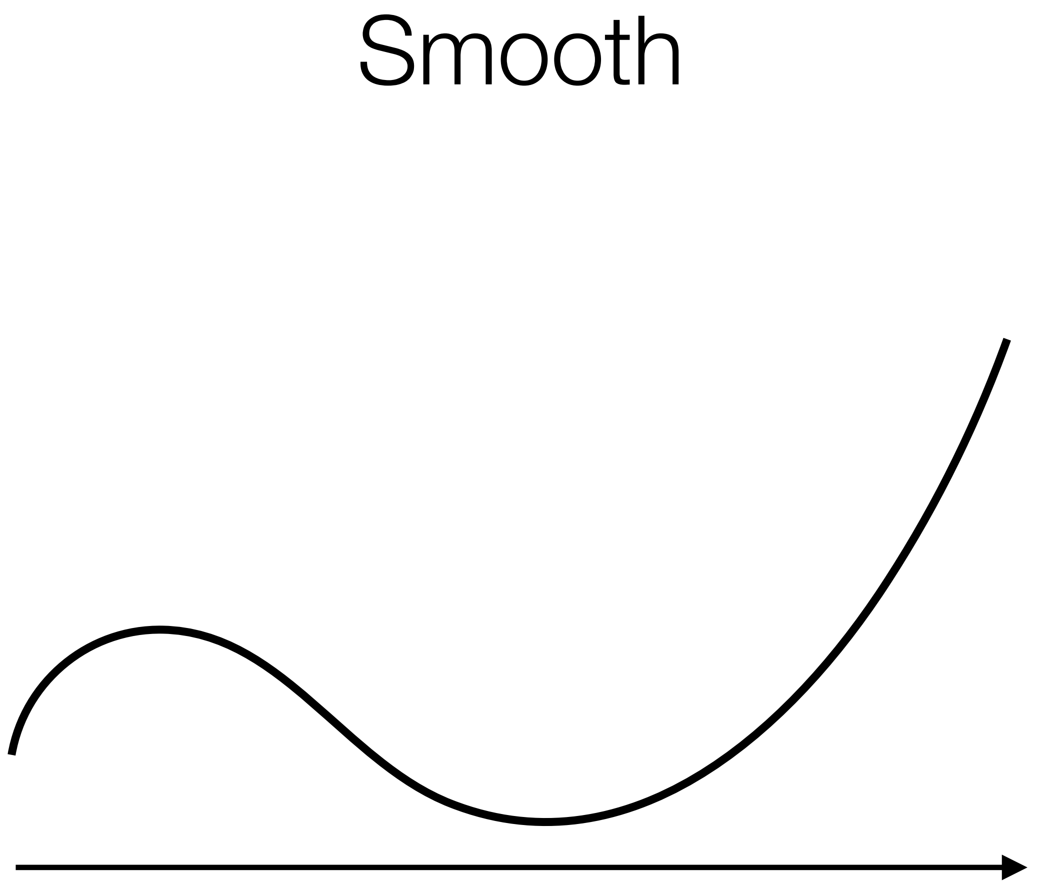
- First-order condition:

$$g(\boldsymbol{v}) - g_{1;\boldsymbol{w}}(\boldsymbol{v}) \leq \frac{\beta}{2} \left\| \boldsymbol{v} - \boldsymbol{w} \right\|^2 \quad \forall\, \boldsymbol{w}, \boldsymbol{v} \in \mathcal{W}$$

- Second-order condition: $\nabla^2 g(\boldsymbol{w}) \preceq \beta \mathbf{I}_d, \ \forall\, \boldsymbol{w} \in \mathcal{W}$

- Note: it is dual of strong convexity.


Non-smooth


Smooth

Gradient Method

- Consider unconstrained smooth optimization:

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}) \qquad f: \ \beta\text{-smooth}$$

- A general descent iterative method:
    - Update: call the oracle and decide an increment $\boldsymbol{\delta}^{[t]}$ to "move" the point from $\boldsymbol{x}^{[t]}$ to $\boldsymbol{x}^{[t+1]}$: $\boldsymbol{x}^{[t+1]} = \boldsymbol{x}^{[t]} + \boldsymbol{\delta}^{[t]}$
    - The direction $\frac{\boldsymbol{\delta}^{[t]}}{\|\boldsymbol{\delta}^{[t]}\|_2}$ is called the *descent* direction.

- In unconstrained smooth problems with first-order oracle, the descent direction is usually determined by gradients.
- Gradient descent: choose the descent direction as the *anti-gradient*:

$$\boldsymbol{\delta}^{[t]} = -\eta_t \nabla f(\boldsymbol{x}^{[t]}), \ \eta_t > 0$$

Constrained Problems

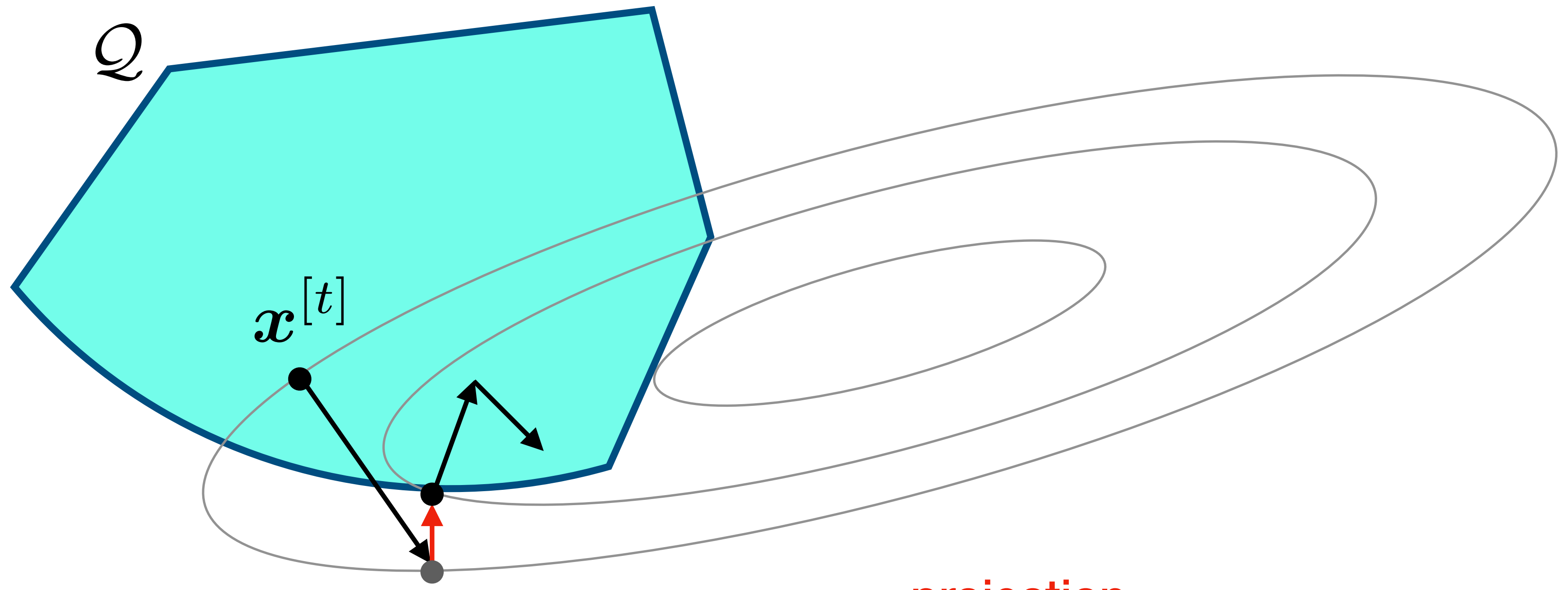
$$\min_{\boldsymbol{x} \in Q} f(\boldsymbol{x})$$

- The feasible set tells *hard* constraints on the solution.

- In general the feasible set $Q \subsetneq \mathbb{R}^d$. GD might go out of bound. Some additional steps are needed.

- First idea: after the descent, find the “closest” point in Q and use it as the new iterate.

Projected Gradient Descent



$$\boldsymbol{x}^{[t]} - \eta_t \nabla f(\boldsymbol{x}^{[t]}) \xrightarrow{\text{projection}} \mathsf{Proj}_{\mathcal{Q}}\left\{\boldsymbol{x}^{[t]} - \eta_t \nabla f(\boldsymbol{x}^{[t]})\right\} =: \boldsymbol{x}^{[t+1]}$$

- Projection step (Euclidean Projection):

$$\mathsf{Proj}_{\mathcal{Q}}\{\boldsymbol{x}\} \triangleq \underset{\boldsymbol{y}\in\mathcal{Q}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{x}\|_2$$

- Caveat: projection might be computationally expensive.

Convergence of Gradient Descent

Set-up:

- Criterion: $f(\boldsymbol{x}^{\mathrm{o}}) - f(\boldsymbol{x}^*)$ (gap in objective function values)
- # of update iterations: τ
- Initial distance to the optimum: $D := \|\boldsymbol{x}^{[1]} - \boldsymbol{x}^*\|^2$

Convex, β-smooth objective function:

- Convexity only: $O(D\beta\tau^{-1})$
- α-strongly convex: $O(D\beta(1 - \frac{\alpha}{\beta})^{\tau})$
- Achieved with fixed step size $\eta_t = \beta^{-1}$

Non-Differentiable Problems

What if it is not differentiable at certain points?

Idea: replace the gradient by a *subgradient*.

- Call it “subgradient descent” (sGD).

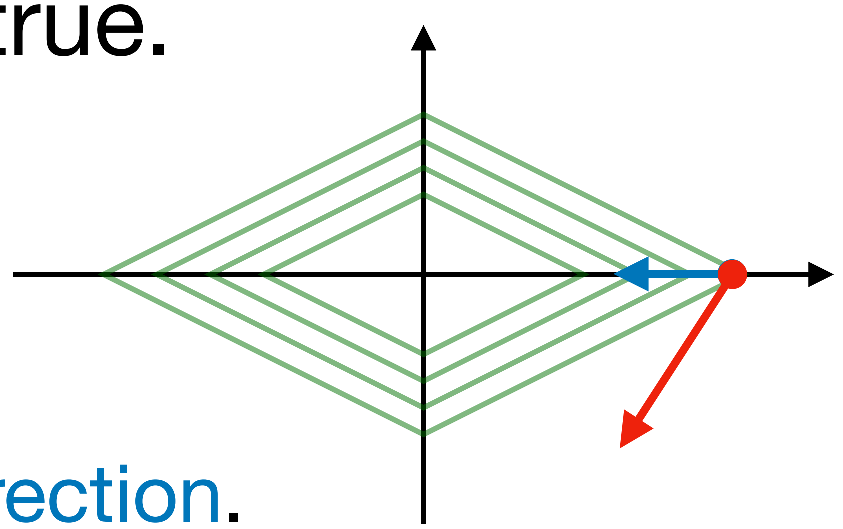
How to guarantee convergence without smoothness?

- Subgradient may no longer be a descent direction.
- Convergence becomes slower.

Anti-subgradient ≠ Descent Direction

- With smoothness, anti-gradient direction is a descent direction as long as step size is small enough.

- Without smoothness, this is no longer true.



- Example: $f(x_1, x_2) = |x_1| + 2|x_2|$

  At $(x_1, x_2) = (1, 0)$:
  - $\boldsymbol{g}_1 = (1, 0) \in \partial f(x_1, x_2)$, $-\boldsymbol{g}_1$ is a descent direction.
  - $\boldsymbol{g}_2 = (1, 2) \in \partial f(x_1, x_2)$, $-\boldsymbol{g}_2$ is not a descent direction.
- Why? Without smoothness, one can change directions significantly but still satisfy the subgradient condition.
- Since $\{f(\boldsymbol{x}^{[t]}), t = 1, 2, ...\}$ is no longer monotone, the final output of the algorithm is no longer the last one.

Lipschitz Continuity

- A function $g : \mathcal{W} \to \mathbb{R}$ is ρ-Lipschitz if $\forall \boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathcal{W}$,

$$|g(\boldsymbol{w}_1) - g(\boldsymbol{w}_2)| \leq \rho \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|$$

- Intuitively, Lipschitz function cannot change too fast.

- For a convex function g defined on an open set $\mathcal{W} \subseteq \mathbb{R}^d$, being ρ-Lipschitz is equivalent to having *bounded (sub-)gradients* (upper bounded by ρ).

Convergence of Gradient Descent

Set-up:

- Criterion: $f(\boldsymbol{x}^{\mathrm{o}}) - f(\boldsymbol{x}^*)$ (gap in objective function values)
- # of update iterations: τ
- Initial distance to the optimum: $D := \|\boldsymbol{x}^{[1]} - \boldsymbol{x}^*\|^2$

Convex, ρ-Lipschitz continuous objective function:

- Convexity only: $O(\sqrt{D\rho^2}\tau^{-\frac{1}{2}})$
- α-strongly convex: $O(\rho^2\alpha^{-1}\tau^{-1})$
- Taking average of the iterates
- Or selecting the best among all iterates

Summary: Iteration Complexity

Criterion: $f(\boldsymbol{x}^{\mathrm{o}}) - f(\boldsymbol{x}^*) \leq \epsilon$

Convex, β-smooth objective function:

- Convexity only: $O(D\beta\epsilon^{-1})$
- α-strongly convex: $O(\frac{\beta}{\alpha}\log(\epsilon^{-1}))$

condition number
$\kappa \triangleq \beta/\alpha$

Convex, ρ-Lipschitz continuous objective function:

- Convexity only: $O(D\rho^2\epsilon^{-2})$
- α-strongly convex: $O(\rho^2\alpha^{-1}\epsilon^{-1})$

Work Complexity of Full GD in Training

$$\min_{\boldsymbol{w}} \; F(\boldsymbol{w}) \equiv \frac{1}{n}\sum_{i=1}^{n} \underbrace{\ell(\boldsymbol{w}, z_i)}_{f_i(\boldsymbol{w})}, \qquad \boldsymbol{w} \in \mathbb{R}^d$$

Criterion: $F(\boldsymbol{w}^{\text{o}}) - F(\boldsymbol{w}^{*}) \leq \epsilon$. Per iteration cost: $\Theta(nd)$

Convex, β-smooth loss function:

- Convexity only: $O(D\beta\epsilon^{-1}) \times \Theta(nd)$
- α-strongly convex: $O(\kappa \log(\epsilon^{-1})) \times \Theta(nd)$

Convex, ρ-Lipschitz continuous loss function:

- Convexity only: $O(D\rho^2\epsilon^{-2}) \times \Theta(nd)$
- α-strongly convex: $O(\rho^2\alpha^{-1}\epsilon^{-1}) \times \Theta(nd)$

Stochastic Convex Optimization

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} F(\boldsymbol{w}) = \mathsf{E}_Z \left[f(\boldsymbol{w}; Z) \right]$$

Questions:

- What if the (sub)gradient becomes noisy?
- How does the noise impact the convergence rate?

Convergence *in expectation* (can be strengthened)

- Convergence for strongly convex smooth optimization slows down significantly. Fixed step sizes fail.
- Convergence for non-smooth problems is not affected.

Stochastic (sub)Gradient Method

Stochastic (sub)Gradient Method

<u>Initialize</u>: start with $\boldsymbol{w}^{[1]}$

<u>Update</u>: in each iteration $t \geq 1$:

1) Generate a realization of a random ξ_t
2) Compute a stochastic vector $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$
3) Choose a step size $\eta_t > 0$
4) Update $\boldsymbol{w}^{[t+1]} \leftarrow \boldsymbol{w}^{[t]} - \eta_t \boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$

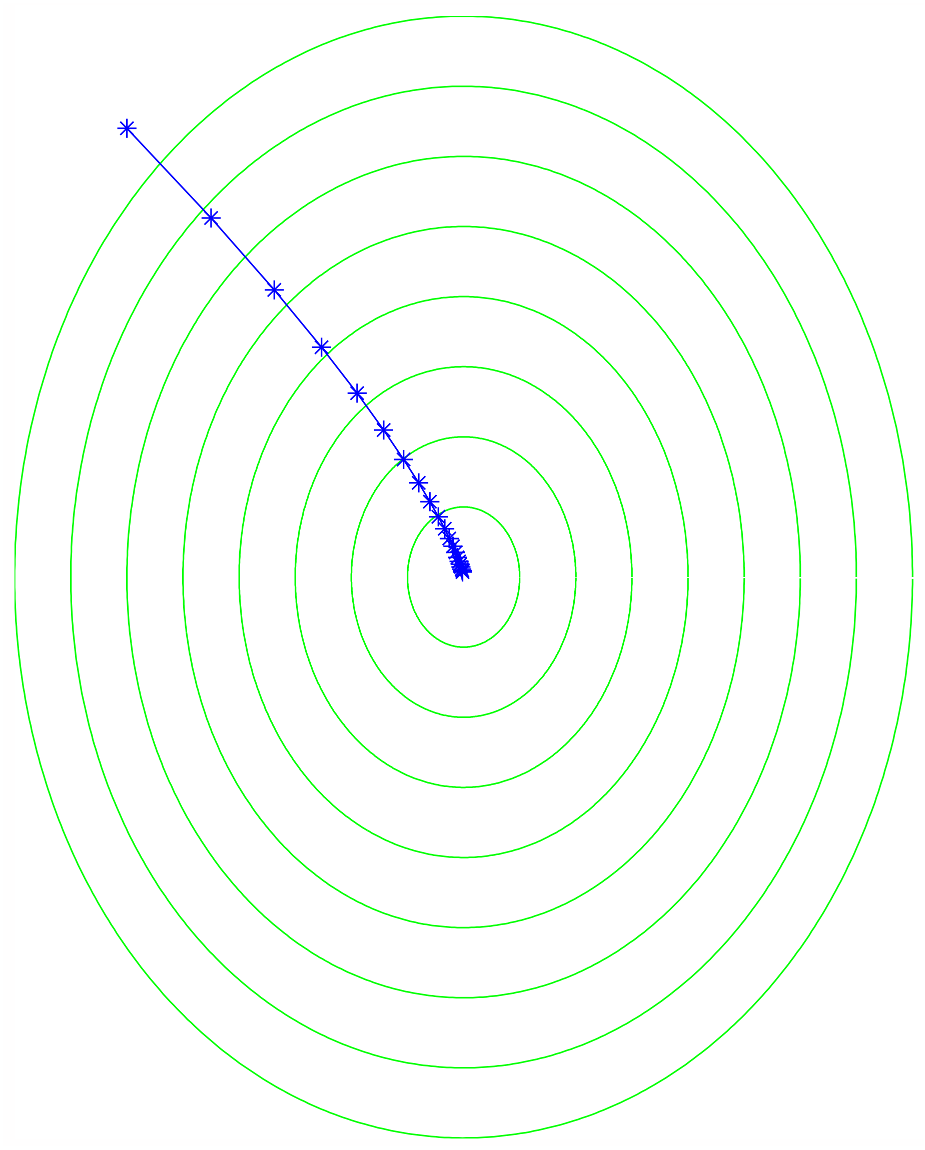
<u>Terminate</u>: after τ iterations, output

$$\boldsymbol{w}^{\mathrm{o}} \overset{\mathrm{f}}{=} \{\boldsymbol{w}^{[1]}, \ldots, \boldsymbol{w}^{[\tau+1]}\}$$

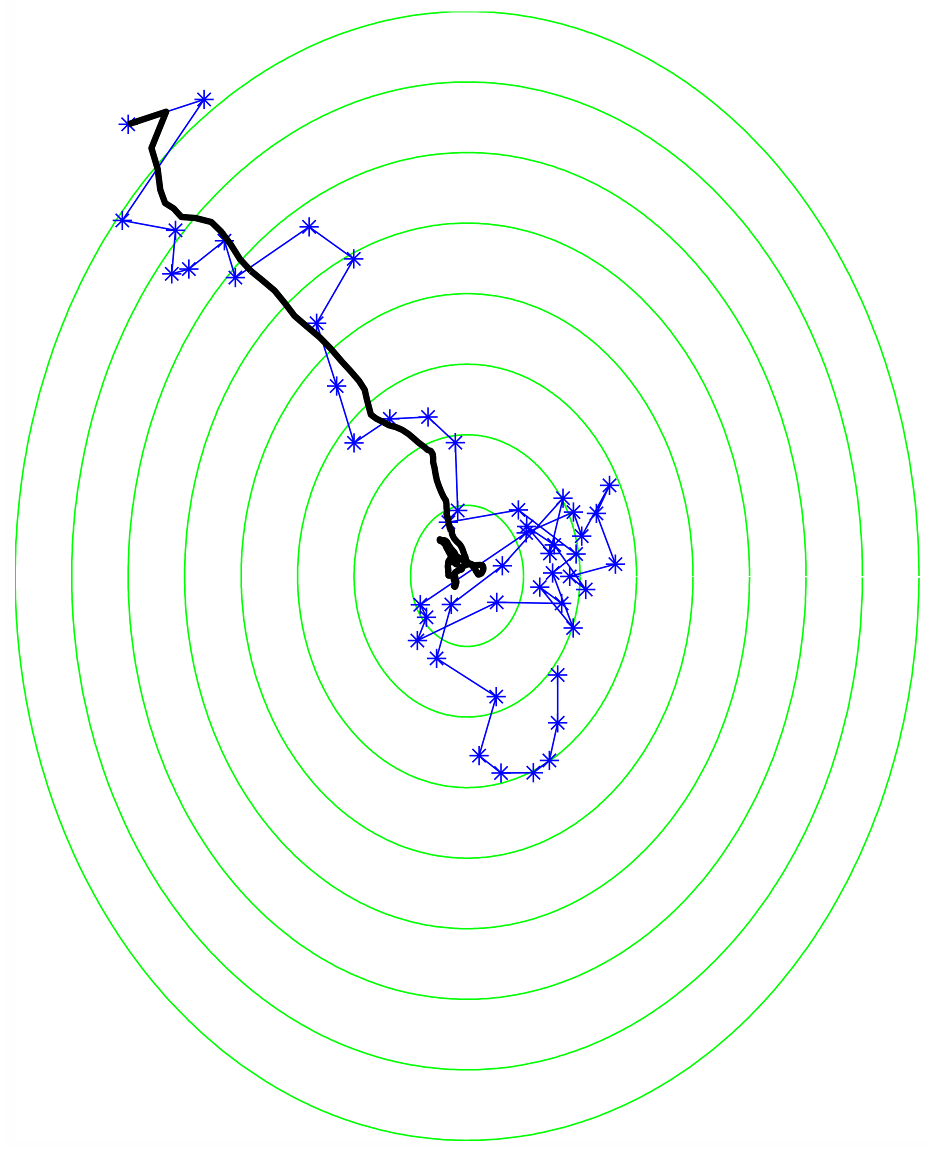
- Here $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$ is an unbiased estimate of subgradient:

$$\mathsf{E}_{\xi_t}\left[\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \,\middle|\, \boldsymbol{w}^{[t]}\right] \in \partial F(\boldsymbol{w}^{[t]})$$

Full GD



SGD



Strongly Convex Smooth Problems

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} F(\boldsymbol{w}) = \mathsf{E}_Z\left[f(\boldsymbol{w};Z)\right]$$

Assumptions:

- $F:\mathbb{R}^d \to \mathbb{R}$ is α-strongly convex and β-smooth
- Variance of $\boldsymbol{g}(\boldsymbol{w}^{[t]},\xi_t)$ is controlled:

$$\mathsf{E}_\xi\left[\|\boldsymbol{g}(\boldsymbol{w},\xi)\|^2\right] \le \sigma_{\mathrm{g}}^2 + c_{\mathrm{g}}\|\nabla F(\boldsymbol{w})\|^2, \quad \forall\, \boldsymbol{w}$$

Convergence: for fixed step size $\eta_t = \eta \le \frac{1}{\beta c_{\mathrm{g}}}$,

$$\mathsf{E}\left[F(\boldsymbol{w}^{[\tau+1]}) - F(\boldsymbol{w}^*)\right] \le (1-\eta\alpha)^\tau (F(\boldsymbol{w}^{[1]}) - F(\boldsymbol{w}^*))$$
$$+\frac{\beta}{\alpha}\frac{\eta}{2}\sigma_{\mathrm{g}}^2$$

Impact of Noise and Step Sizes

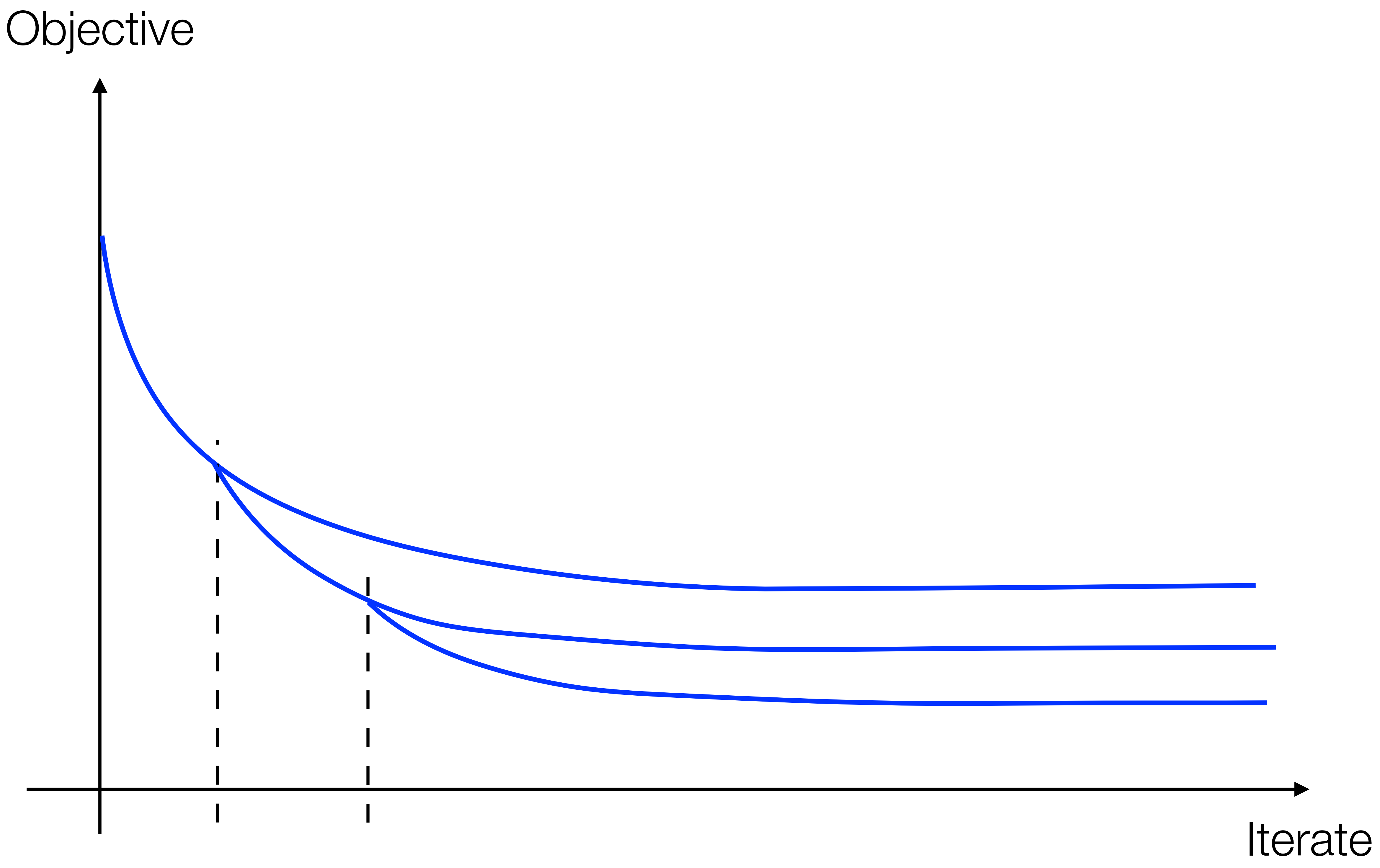
$$\mathrm{E}\left[F(\boldsymbol{w}^{[\tau+1]}) - F(\boldsymbol{w}^*)\right] \leq (1-\eta\alpha)^{\tau}(F(\boldsymbol{w}^{[1]}) - F(\boldsymbol{w}^*)) + \frac{\beta}{\alpha}\frac{\eta}{2}\sigma_{\mathrm{g}}^2$$

- Fast progress in the beginning.
- Once getting close to the optimum, variation of SG prevents further progress
- Smaller step sizes give more accurate final answers, but SGD converges much more slowly
- Go for diminishing step sizes

Diminishing Step Sizes

- Practical idea: whenever progress slows down, reduce the step size.

- Balancing the two terms gives an $O(\tau^{-1})$ convergence:



Whenever progress stalls, we half step sizes and repeat.

Diminishing Step Sizes

- Practical idea: whenever progress slows down, reduce the step size.

- Balancing the two terms gives an $O(\tau^{-1})$ convergence:

Convergence: with diminishing step sizes $\eta_t = \Omega(\frac{1}{\alpha t})$,

$$\mathsf{E}\left[F(\boldsymbol{w}^{[\tau+1]}) - F(\boldsymbol{w}^*)\right] = O(\max\{\sigma_{\mathrm{g}}^2, D\}\alpha^{-1}\tau^{-1})$$

- A similar τ^{-1} order holds without smoothness.
- Lower bound: for first-order stochastic optimization problems, the order τ^{-1} cannot be improved anymore.

Non-Smooth Problems

- Without smoothness: stochastic subgradient method.

- Convergence guarantee is identical to the deterministic version (in expectation). Analysis is similar too.

Convergence: suppose the stochastic subgradient is bounded almost surely ($\leq \rho$), with constant step size η ,

$$\mathrm{E}\left[F(\overline{\boldsymbol{w}}^{[\tau]}) - F(\boldsymbol{w}^*)\right] \leq \frac{D}{2\eta\tau} + \frac{\eta\rho^2}{2} \quad \overline{\boldsymbol{w}}^{[\tau]} \equiv \frac{1}{\tau}\sum_{t=1}^{\tau} \boldsymbol{w}^{[t]}$$

$$= O(\sqrt{D\rho^2}\tau^{-\frac{1}{2}})$$

Strong convexity improves the convergence to $O(\frac{\rho^2}{\alpha}\tau^{-1})$

Remark: smoothness does not help improve the order!

Training Cost Comparison

$$\min_{\boldsymbol{w}} \ F(\boldsymbol{w}) \equiv \frac{1}{n}\sum_{i=1}^{n} \underbrace{\ell(\boldsymbol{w}, z_i)}_{f_i(\boldsymbol{w})}, \quad \boldsymbol{w} \in \mathbb{R}^d$$

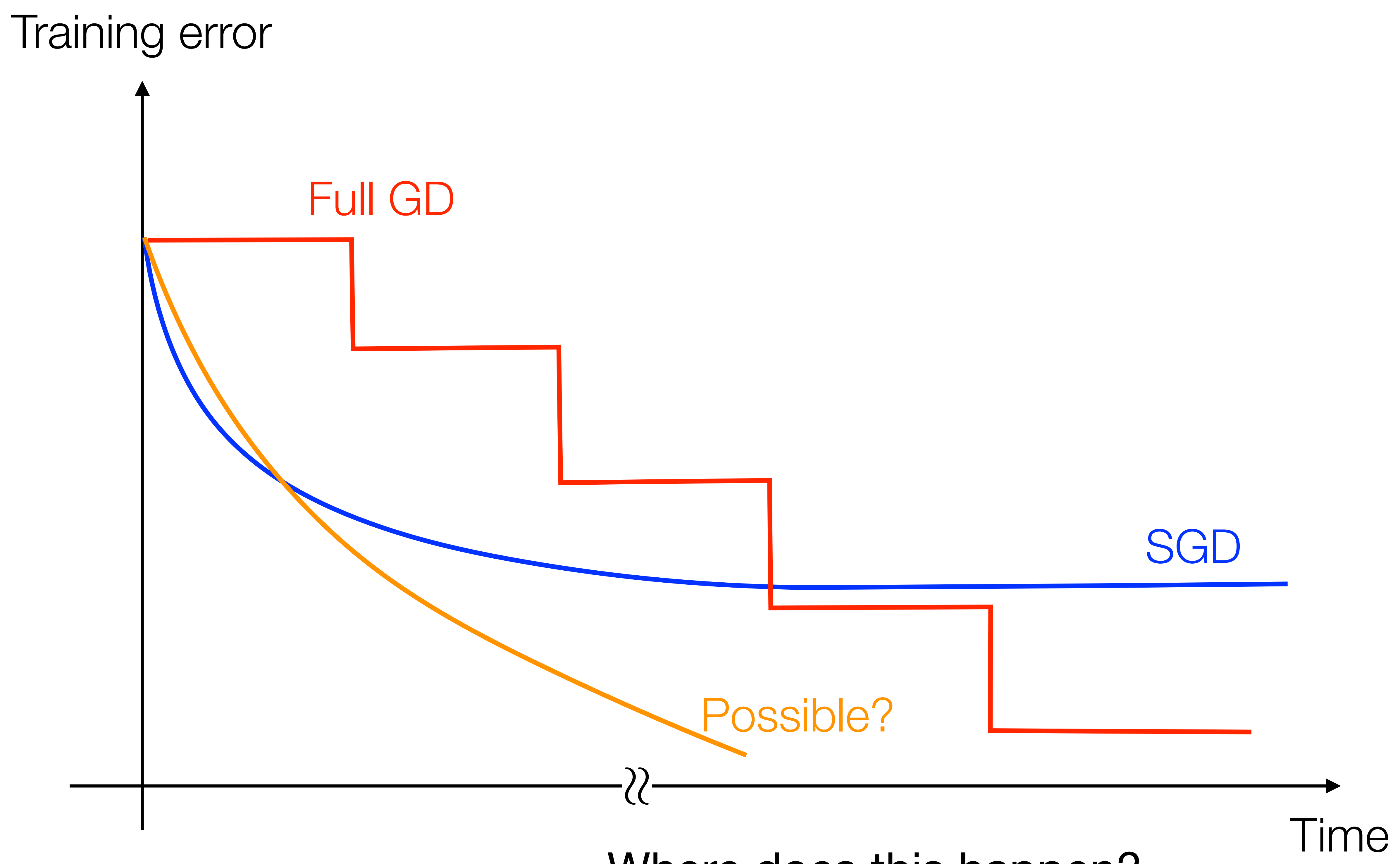
Per iteration cost: $\Theta(nd)$ vs. $\Theta(d)$

Convex, β-smooth loss function:

- Convexity only: $O(D\beta\epsilon^{-1}) \times \Theta(nd)$ Full GD
  $O(D\rho^2\epsilon^{-2}) \times \Theta(d)$ SGD
- α-strongly convex: $O(\kappa \log(\epsilon^{-1})) \times \Theta(nd)$ Full GD
  $O(\alpha^{-1}\epsilon^{-1}) \times \Theta(d)$ SGD

Convex, ρ-Lipschitz continuous loss function:

- Not need to compare. SGD is clear winner.



Where does this happen?
Depends on size of data set

Variability of Stochastic Gradients

$$\mathsf{E}\left[F(\boldsymbol{w}^{[\tau+1]}) - F(\boldsymbol{w}^*)\right] \leq (1-\eta\alpha)^\tau (F(\boldsymbol{w}^{[1]}) - F(\boldsymbol{w}^*)) + \frac{\beta}{\alpha}\frac{\eta}{2}\sigma_{\mathrm{g}}^2$$

- SGD with constant step sizes tends to oscillate around global minimum, since large step sizes cannot suppress variability in stochastic gradients.

- Key: variability of $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t)$, σ_{g}^2.

- For vanilla SGD in training, $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) = \nabla f_{\xi_t}(\boldsymbol{w}^{[t]})$
  Not negligible even when $\boldsymbol{w}^{[t]} = \boldsymbol{w}^*$

Variance Reduction

- Instead of vanilla SGD $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \leftarrow \nabla f_{\xi_t}(\boldsymbol{w}^{[t]})$, use other kinds of stochastic gradient with smaller variability.

- Minibatch: $\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \leftarrow \frac{1}{|\xi_t|} \sum_{i \in \xi_t} \nabla f_i(\boldsymbol{w}^{[t]})$
  Minibatch size $|\xi_t| = n_{\mathrm{mb}}$, variance reduced by $1/n_{\mathrm{mb}}$.
  Batch size can be *dynamic*.

- Gradient aggregation: take some zero-mean $\boldsymbol{v}^{[t]}$ and set
  $$\boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) \leftarrow \nabla f_{\xi_t}(\boldsymbol{w}^{[t]}) - \boldsymbol{v}^{[t]}$$
  Still an unbiased estimate of full gradient.
  $\boldsymbol{v}^{[t]}$ highly correlated with $\nabla f_{\xi_t}(\boldsymbol{w}^{[t]})$ to reduce variance.
  Use past gradient info if current iterate close to past ones

Stochastic Variance Reduced Gradient

Idea: if we can access an previous iterate $\boldsymbol{w}^{\text{old}}$, and it is not so far away from the current iterate $\boldsymbol{w}^{[t]}$, then pick

$$\boldsymbol{v}^{[t]} = \nabla f_{\xi_t}(\boldsymbol{w}^{\text{old}}) - \nabla F(\boldsymbol{w}^{\text{old}})$$

- Roughly equal to the "bias" of $\nabla f_{\xi_t}(\boldsymbol{w}^{[t]})$ if $\boldsymbol{w}^{[t]} \approx \boldsymbol{w}^{\text{old}}$
- Correcting $\nabla f_{\xi_t}(\boldsymbol{w}^{[t]})$ towards $\nabla F(\boldsymbol{w}^{[t]})$:

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{w}^{[t]}, \xi_t) &\leftarrow \nabla f_{\xi_t}(\boldsymbol{w}^{[t]}) - \left(\nabla f_{\xi_t}(\boldsymbol{w}^{\text{old}}) - \nabla F(\boldsymbol{w}^{\text{old}})\right) \\ &\approx \nabla f_{\xi_t}(\boldsymbol{w}^{[t]}) - \Big(\nabla f_{\xi_t}(\boldsymbol{w}^{[t]}) - \nabla F(\boldsymbol{w}^{[t]})\Big) \\ &= \nabla F(\boldsymbol{w}^{[t]}) \end{aligned}$$

- It is also zero-mean.

SVRG Algorithm [Johnson and Zhang, 2013]

Run in epochs.

In the s-th epoch:

- Take a snapshot $\boldsymbol{w}_s^{\text{old}}$, compute the full gradient $\nabla F(\boldsymbol{w}_s^{\text{old}})$
- Inner loop: use $\boldsymbol{w}_s^{\text{old}}$ to reduce variance: for $t = 1, ..., m$

$$\boldsymbol{w}_s^{[t+1]} \leftarrow \boldsymbol{w}_s^{[t]} - \eta \left\{ \nabla f_{\xi_t}(\boldsymbol{w}_s^{[t]}) - \left(\nabla f_{\xi_t}(\boldsymbol{w}_s^{\text{old}}) - \nabla F(\boldsymbol{w}_s^{\text{old}}) \right) \right\}$$

Note: this is a hybrid approach – full gradient + SG

- Full gradient is computed once every epoch.
- Each epoch contains $2m + n$ gradient computations

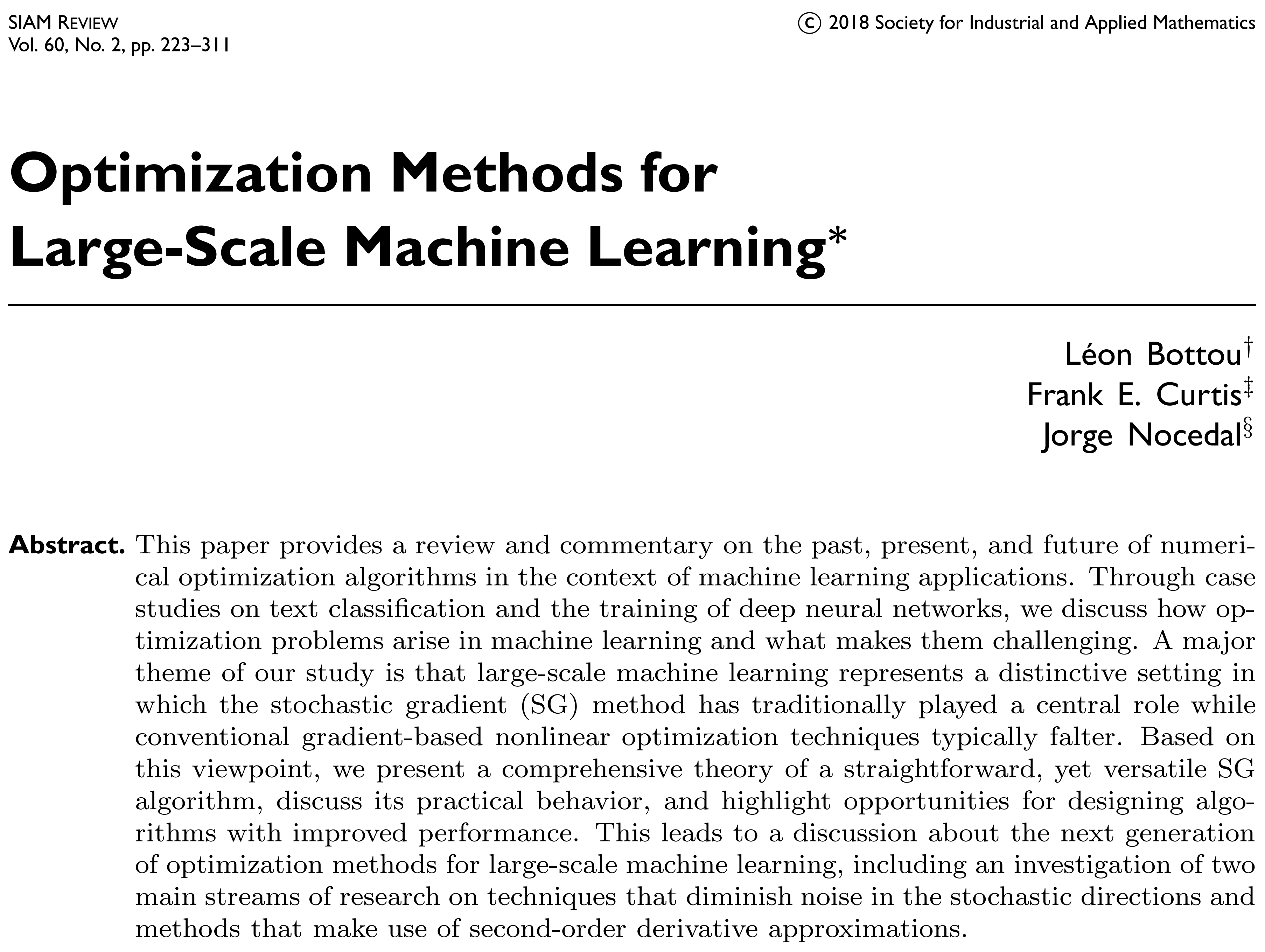
Summary

- For large-scale ML problems, stochastic methods usually have advantage over batch methods

- Typical anecdote: suppress variability of stochastic gradients by diminishing step sizes
- How to diminish step size depends on problem structures

- Not covered:
    - More methods for noise reduction
    - Non-convex problems

Main Reference



SIAM REVIEW
Vol. 60, No. 2, pp. 223–311

Optimization Methods for Large-Scale Machine Learning*

Léon Bottou[†]
Frank E. Curtis[‡]
Jorge Nocedal[§]

Abstract. This paper provides a review and commentary on the past, present, and future of numerical optimization algorithms in the context of machine learning applications. Through case studies on text classification and the training of deep neural networks, we discuss how optimization problems arise in machine learning and what makes them challenging. A major theme of our study is that large-scale machine learning represents a distinctive setting in which the stochastic gradient (SG) method has traditionally played a central role while conventional gradient-based nonlinear optimization techniques typically falter. Based on this viewpoint, we present a comprehensive theory of a straightforward, yet versatile SG algorithm, discuss its practical behavior, and highlight opportunities for designing algorithms with improved performance. This leads to a discussion about the next generation of optimization methods for large-scale machine learning, including an investigation of two main streams of research on techniques that diminish noise in the stochastic directions and methods that make use of second-order derivative approximations.

References

[1] H. Robbins and S. Monro, “A Stochastic Approximation Method,” The Annals of Mathematical Statistics, vol. 22, no. 3, pp. 400-407, 1951.

[2] L. Bottour, F. E. Curtis, and J. Nocedal, “Optimization Methods for Large-Scale Machine Learning,” SIAM Review, vol. 60, no. 2, pp. 223-311, 2018.

[3] R. Johnson and T. Zhang, “Accelerating stochastic gradient descent using predictive variance reduction,” NIPS, 2013.