COMP6211I: Trustworthy Machine Learning

Lecture 1

Minhao Cheng

Math Basics

Linear Algebra

- Linear dependence, span
- Orthogonal, orthonormal,
- Eigendecomposition, quadratic form
  - $f(x) = x^T Ax, s.t \ \|x\|_2 = 1$
- Positive definite: all eigenvalues are positive, positive semidefinite are all positive or zero
  - $\forall x, \ x^T Ax \geq 0$
- Singular Value Decomposition (SVD)
  - $A = UDV^T$, where A is $m \times n$ matrix, U is $m \times m$ matrix, V is $n \times n$ vector

Math Basics

Matrix calculus

- $f = \|Xw - y\|^2$, solve $\dfrac{\partial f}{\partial w}$, where y is $m \times 1$ vector, X is $m \times n$ matrix, w is $n \times 1$ vector
- $$\begin{aligned} df = d(\|Xw - y\|^2) &= d((Xw - y)^T(Xw - y)) = d((Xw - y)^T)(Xw - y) + (Xw - y)^T d(Xw - y) \\ &= (Xdw)^T(Xw - y) + (Xw - y)^T(Xdw) = 2(Xw - y)^T Xdw \end{aligned}$$
- So $\dfrac{\partial f}{\partial w} = 2X^T(Xw - y)$

Regression

Linear regression

- Classification:
  - Customer record $\longrightarrow$ Yes/No
- Regression: predicting credit limit
  - Customer record $\longrightarrow$ dollar amount
- Linear Regression:
  - $h(x) = \sum_{i=0}^{d} w_i x_i = w^T x$

Linear Regression

The data set

- Training data:
  - $(x1,y1), (x2,y2), \ldots, (x_N, y_N)$
  - $x_n \in \mathbb{R}^d$: feature vector for a sample
  - $y_n \in \mathbb{R}$: observed output (real number)

Linear Regression

The data set

- Training data:
  - $(x1, y1), (x2, y2), \ldots, (x_N, y_N)$
  - $x_n \in \mathbb{R}^d$: feature vector for a sample
  - $y_n \in \mathbb{R}$: observed output (real number)
- Linear regression: find a function $h(x) = w^T x$ to approximate y

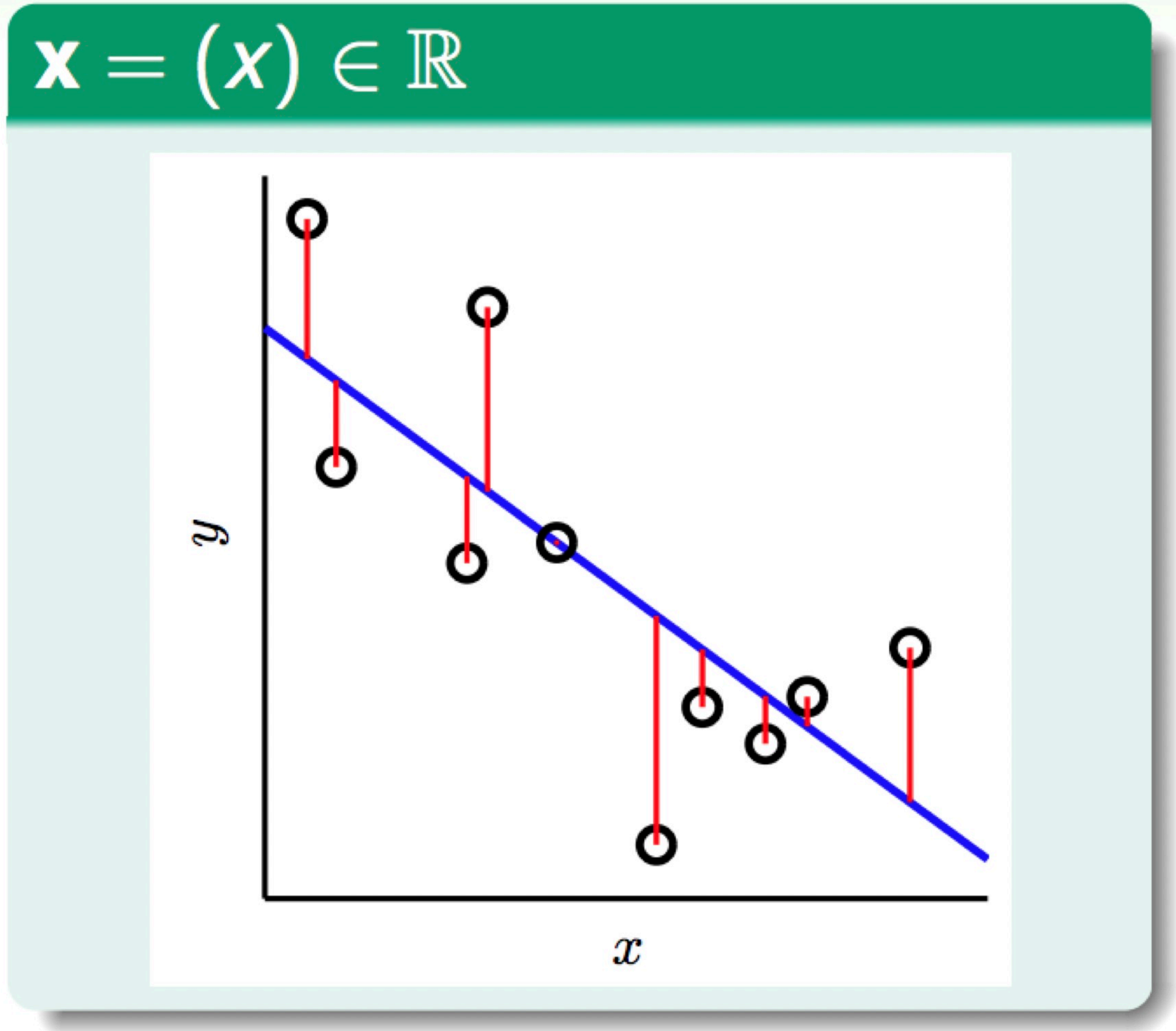
Linear Regression

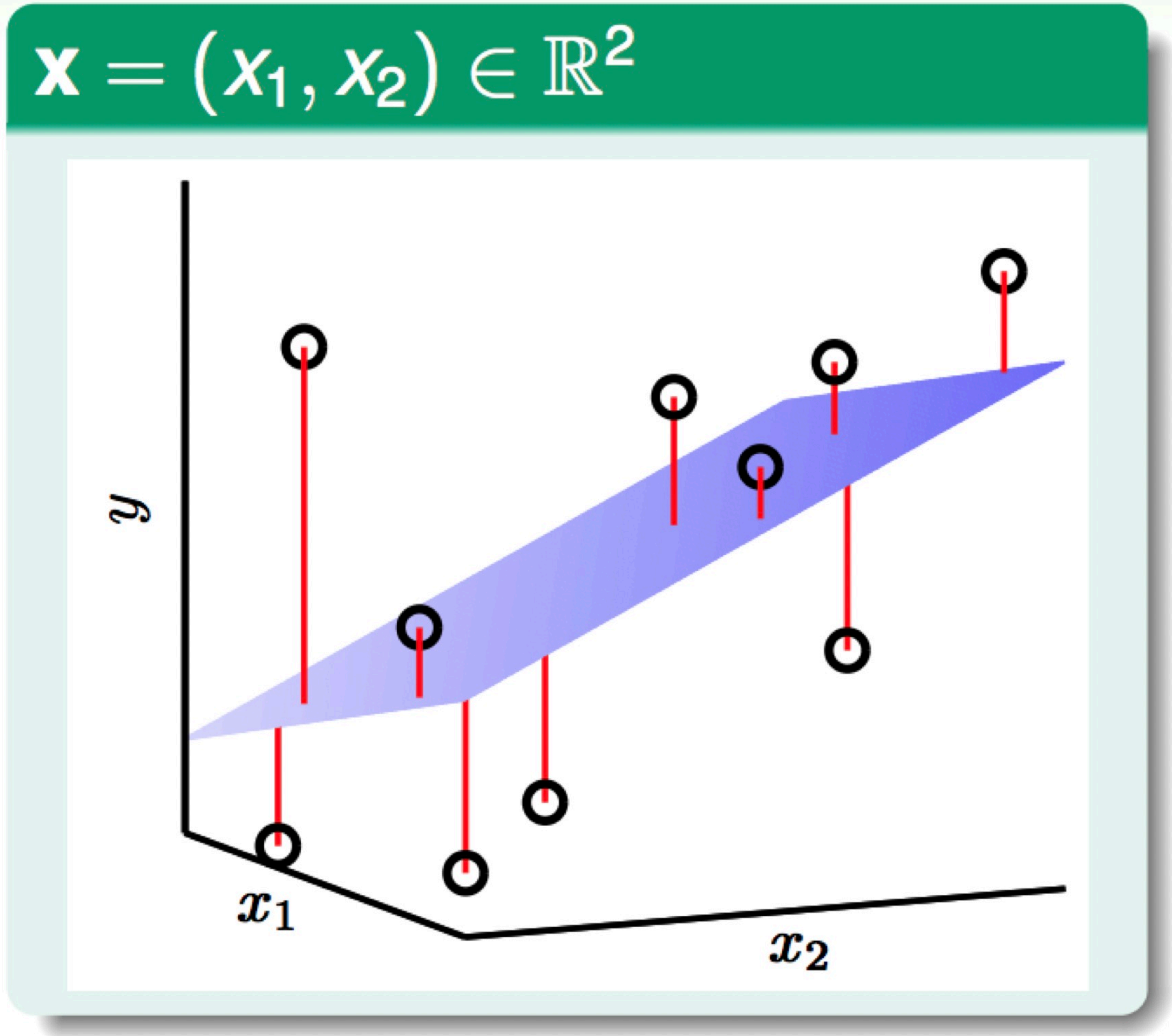
The data set

- Training data:
  - $(x1, y1), (x2, y2), \ldots, (x_N, y_N)$
  - $x_n \in \mathbb{R}^d$: feature vector for a sample
  - $y_n \in \mathbb{R}$: observed output (real number)
- Linear regression: find a function $h(x) = w^T x$ to approximate y
- Measure the error by $(h(x) - y)^2$ (square error)
  - Training error: $E_{\text{train}}(h) = \dfrac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2$

Linear Regression

Illustration





Linear Regression

Matrix form

$$E_{\text{train}}(w) = \frac{1}{N}\sum_{n=1}^{N}(x_n^T w - y_n)^2 = \frac{1}{N}\| \begin{bmatrix} x_1^T w - y_1 \\ x_2^T w - y_2 \\ \vdots \\ x_N^T w - y_N \end{bmatrix} \|^2$$

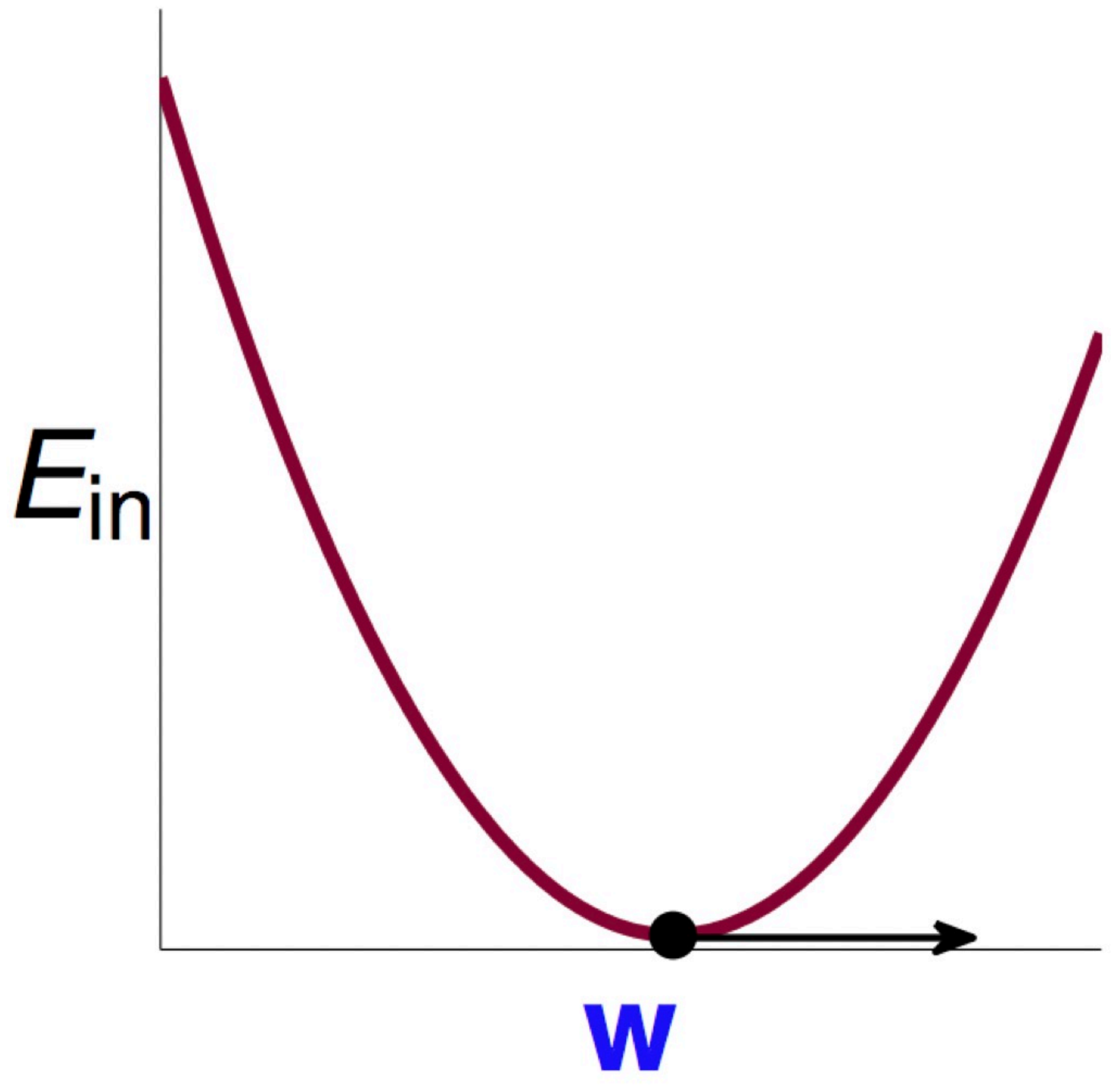
- $$= \frac{1}{N}\| \begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \vdots \\ -x_N^T- \end{bmatrix} w - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \|$$

$$= \frac{1}{N}\| \underbrace{X}_{N\times d} w - \underbrace{y}_{N\times 1} \|^2$$

Linear Regression

Minimize E_{train}

- $min_w f(w) = \|Xw - y\|^2$
  - E_{train}: continuous, differentiable, convex
  - Necessary condition of optimal w:
  - $$\nabla f(w^*) = \begin{bmatrix} \frac{\partial f}{\partial w_0}(w^*) \\ \vdots \\ \frac{\partial f}{\partial w_d}(w^*) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$



Linear Regression

Minimizing f

$$f(w) = \|Xw - y\|^2 = w^T X^T X w - 2 w^T X^T y + y^T y$$

$$\nabla f(w) = 2(X^T X w - X^T y)$$

- $$\nabla f(w^*) = 0 \Rightarrow \underbrace{X^T X w^* = X^T y}_{\text{normal equation}}$$

Linear Regression

Minimizing f

$$f(w) = \|Xw - y\|^2 = w^T X^T X w - 2w^T X^T y + y^T y$$

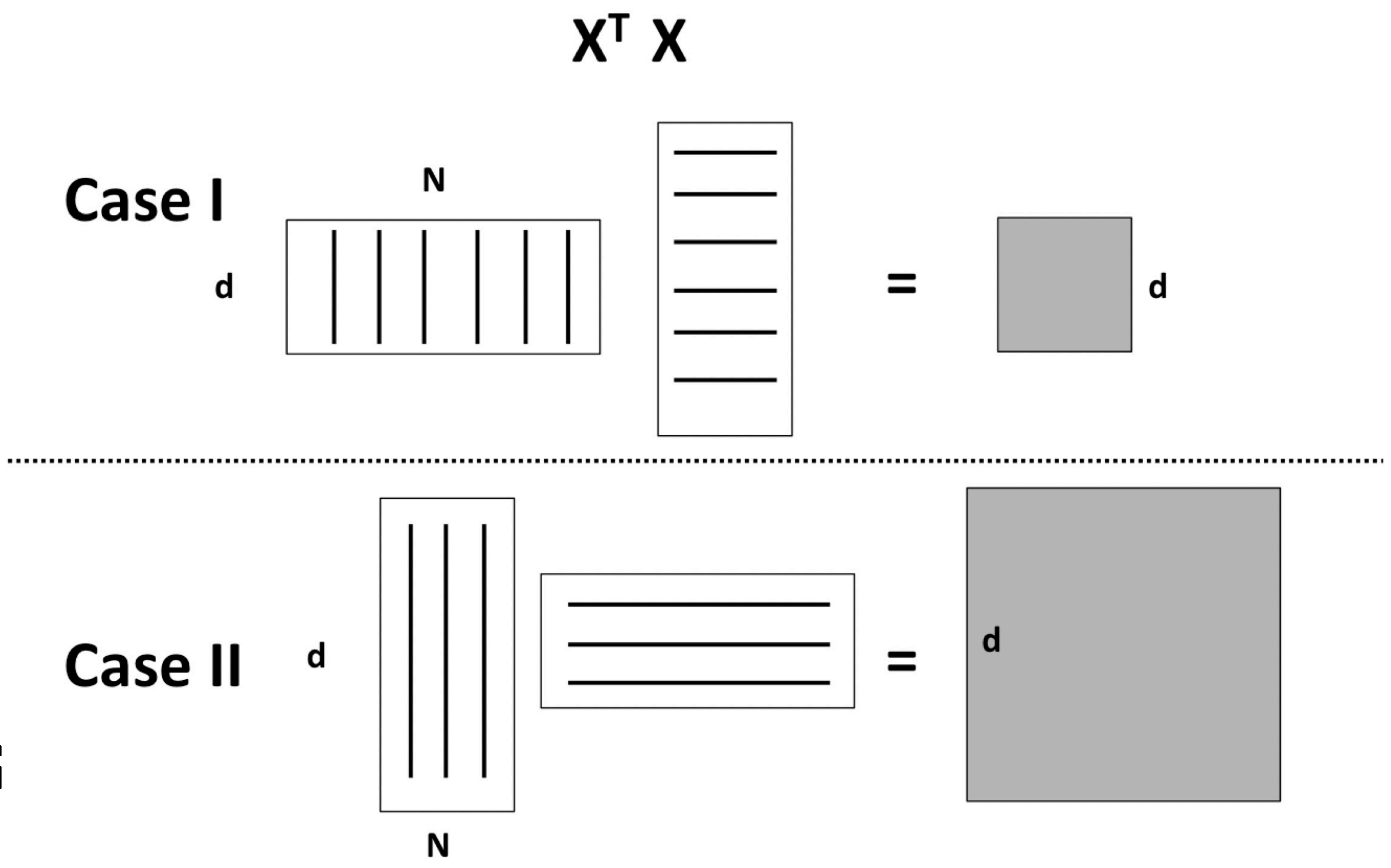
$$\nabla f(w) = 2(X^T X w - X^T y)$$

- $$\nabla f(w^*) = 0 \Rightarrow \underbrace{X^T X w^* = X^T y}_{\text{normal equation}}$$
- $\Rightarrow w^* = (X^T X)^{-1} X^T y$ How?

Linear Regression

Solutions

- Case I: $X^T X$ is invertible $\Rightarrow$ Unique solution
  - Often when $N > d$
  - Yes, $w^* = (X^T X)^{-1} X^T y$
- Case II: $X^T X$ is non-invertible $\Rightarrow$ Many solutions
  - Often when $d > N$



Logistic Regression

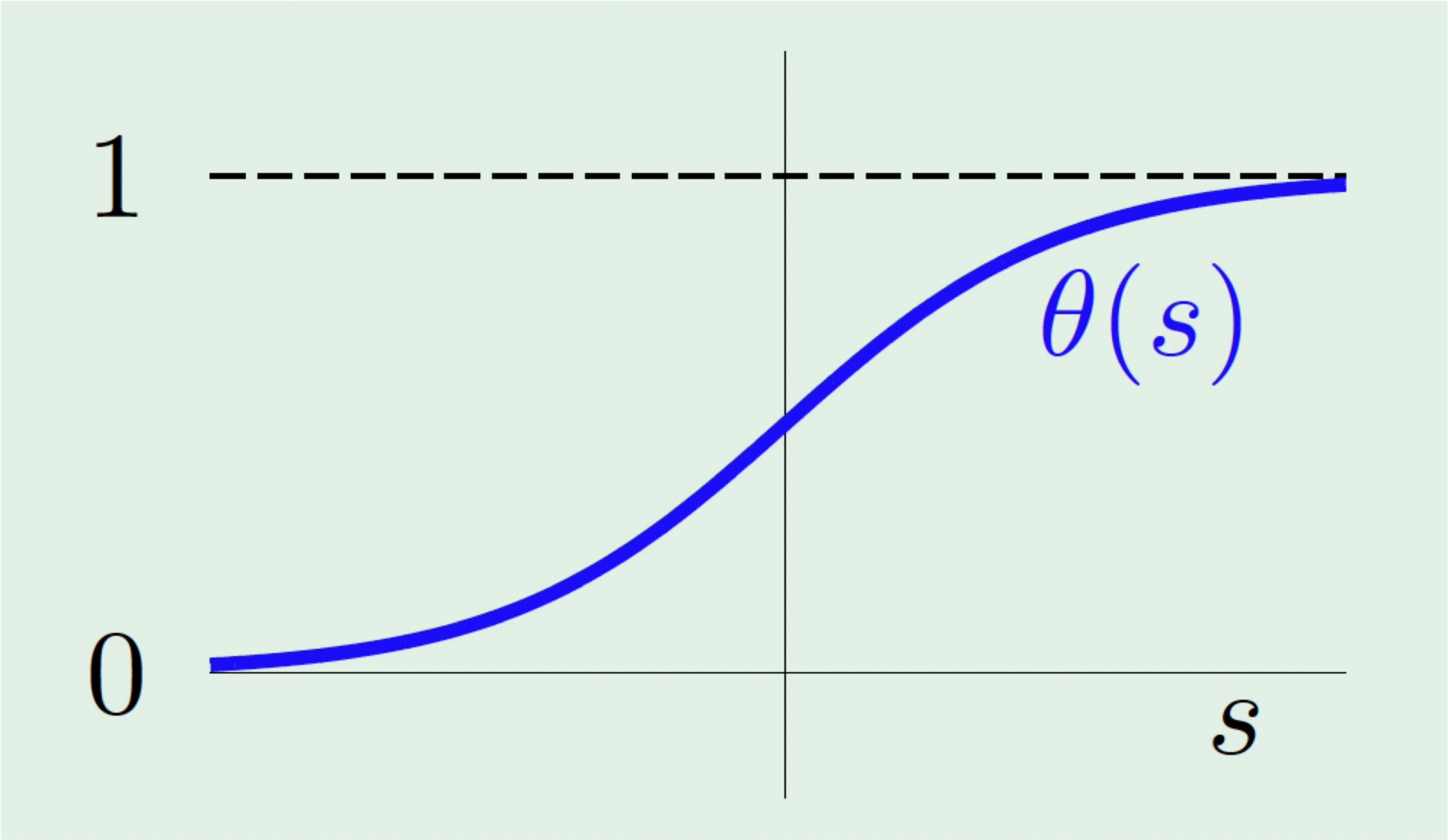
Binary Classification

- Input: training data $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$ and corresponding outputs $y_1, y_2, \ldots, y_n \in \{+1, -1\}$
- Training: compute a function f such that $\text{sign}(f(x_i)) \approx y_i$ for all i
- Prediction: given a testing sample $\tilde{x}$, predict the output as $\text{sign}(f(x_i))$

Logistic Regression

Binary Classification

- Assume linear scoring function: $s = f(x) = w^T x$
- Logistic hypothesis:
  - $P(y = 1 \,|\, x) = \theta(w^T x)$,
  - Where $\theta(s) = \dfrac{e^s}{1 + e^s} = \dfrac{1}{1 + e^{-s}}$



- How about $P(y = -1 \,|\, x)$?
  - $P(y = -1 \,|\, x) = 1 - \dfrac{1}{1 + e^{-w^T x}} = \dfrac{1}{1 + e^{w^T x}} = \theta(-w^T x)$
- Therefore, $P(y \,|\, x) = \theta(y w^T x)$

Logistic Regression

Maximizing the likelihood

- Likelihood of $\mathscr{D} = (x_1, y_1), \ldots, (x_N, y_N)$:
- $\prod_{n=1}^{N} P(y_n \mid x_n) = \prod_{n=1}^{N} \theta(y_n w^T x_n)$

Logistic Regression

Maximizing the likelihood

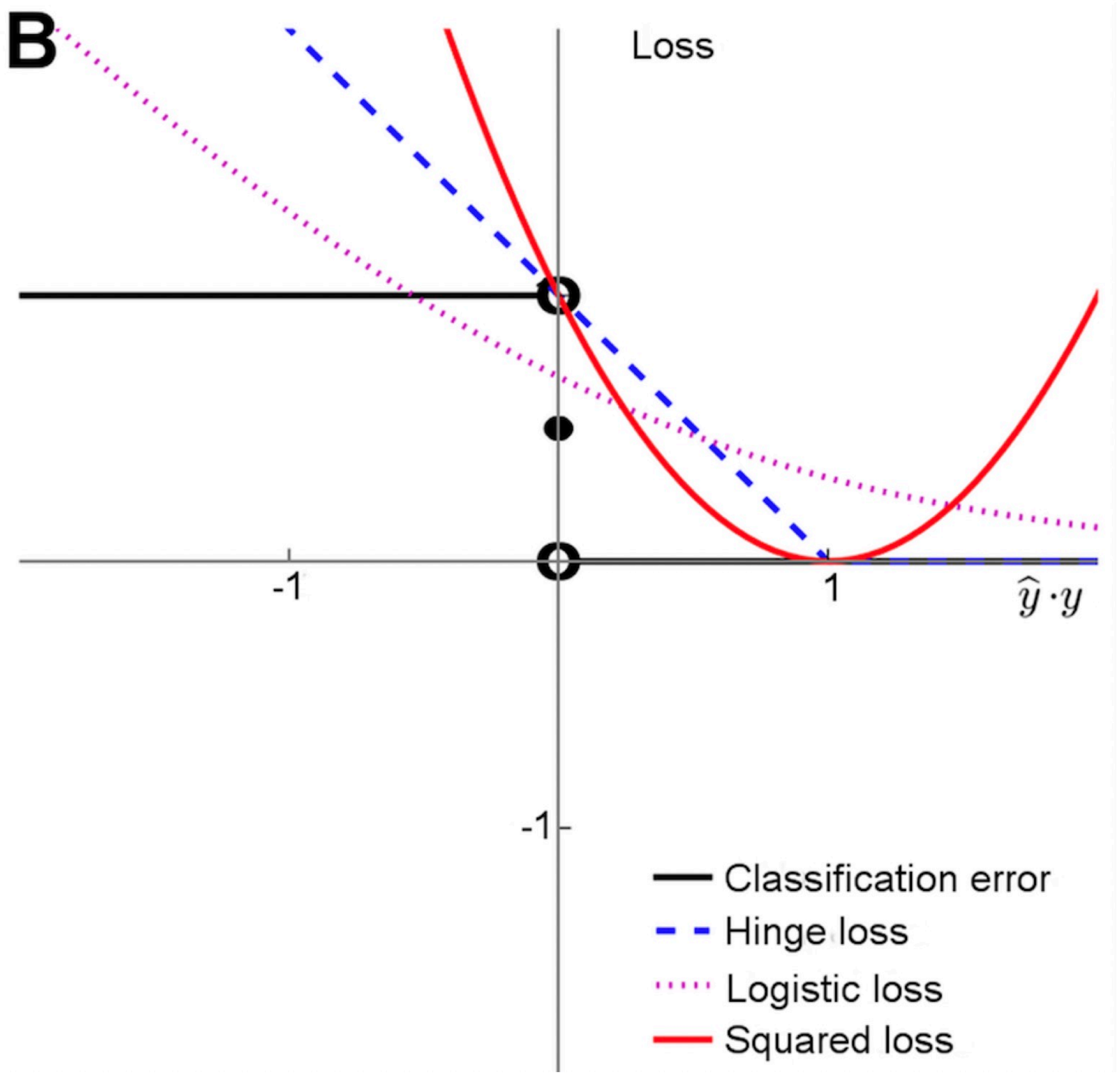
- Likelihood of $\mathscr{D} = (x_1, y_1), \ldots, (x_N, y_N)$:
  - $$\prod_{n=1}^{N} P(y_n \mid x_n) = \prod_{n=1}^{N} \theta(y_n w^T x_n)$$

- Find w to maximize the likelihood!
  - $$\max_w \prod_{n=1}^{N} \theta(y_n w^T x_n)$$
    $$\leftrightharpoons \max_w log(\prod_{n=1}^{N} \theta(y_n w^T x_n))$$
    $$\leftrightharpoons \min_w - \sum_{n=1}^{N} log(\theta(y_n w^T x_n))$$
    $$\leftrightharpoons \min_w \sum_{n=1}^{N} log(1 + e^{-y_n w^T x_n})$$

Logistic Regression

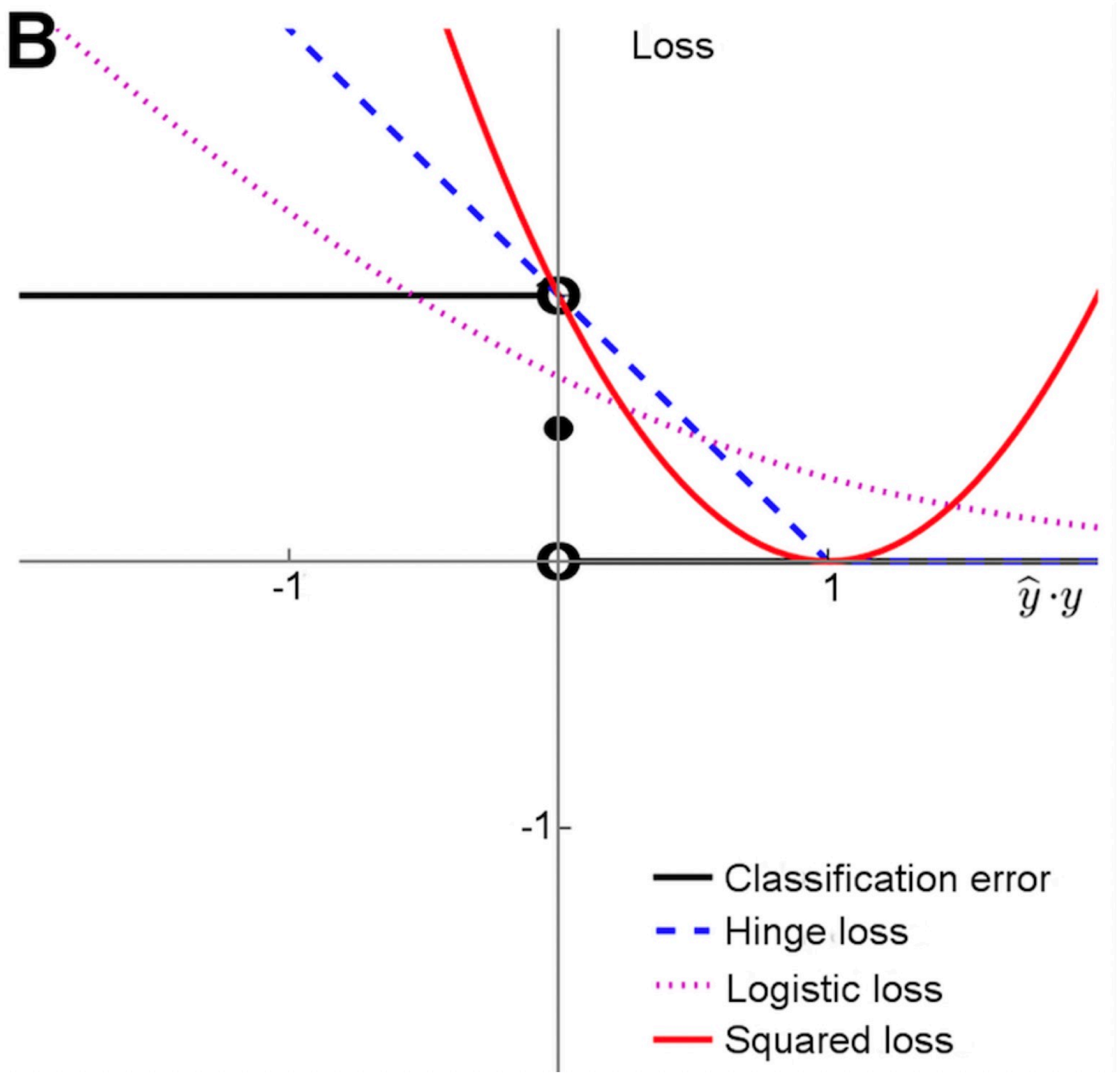
Empirical Risk Minimization (linear)

- Linear classification/regression:
- $$min_w \frac{1}{N}\sum_{n=1}^{N} \text{loss}(\underbrace{w^T x_n}_{\hat{y}_n\text{:the predicted score}}, y_n)$$
  - Linear regression:
    loss$(h(x_n), y_n) = (w^T x_n - y_n)^2$
  - Logistic regression:
    loss$(h(x_n), y_n) = \log(1 + e^{-y_n w^T x_n})$



Support Vector Machines
Hinge loss

- Replace the logistic loss by hinge loss:
- $$\min_{w} \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - y_n w^T x_n)$$



Logistic Regression

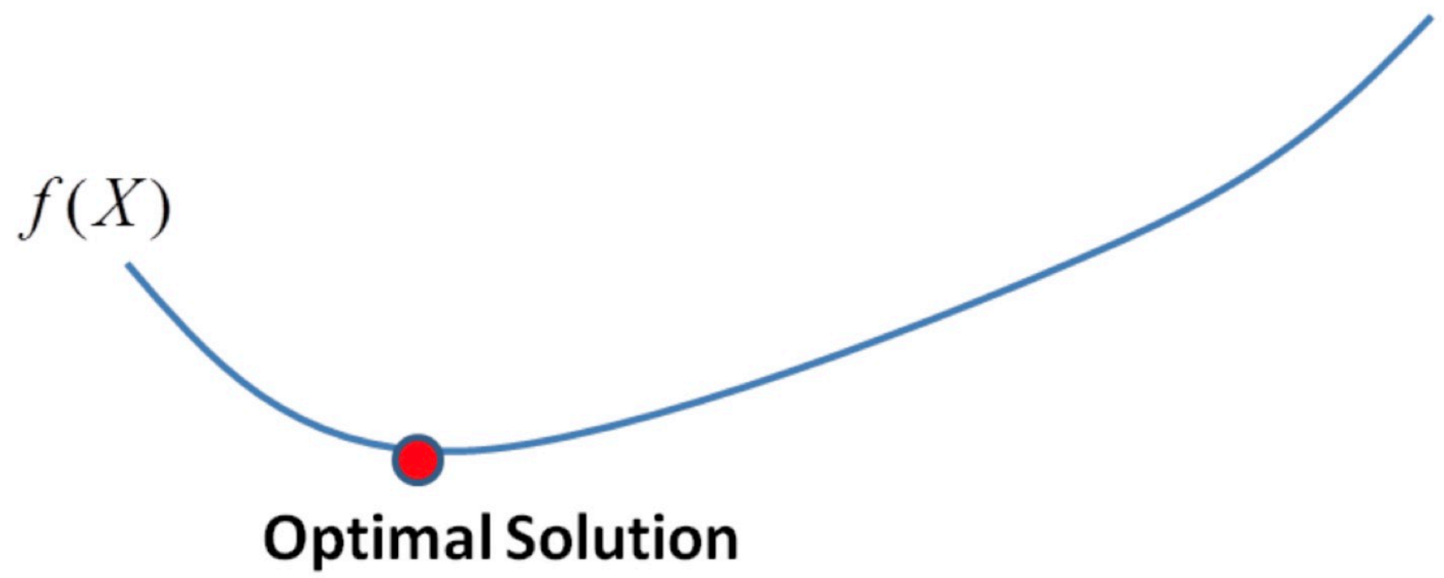
Empirical Risk Minimization (general)

- Assume $f_W(x)$ is the decision function to be learned
  - (W is the parameters of the function)
- General empirical risk minimization
  - $min_W \frac{1}{N} \sum_{n=1}^{N} \text{loss}(f_W(x_n), y_n)$
- Example: Neural network ($f_W(\cdot)$ is the network)

Optimization

Goal

- Goal: find the minimizer of a function
  - $min_w f(w)$
- For now we assume f is twice differentiable



Optimization

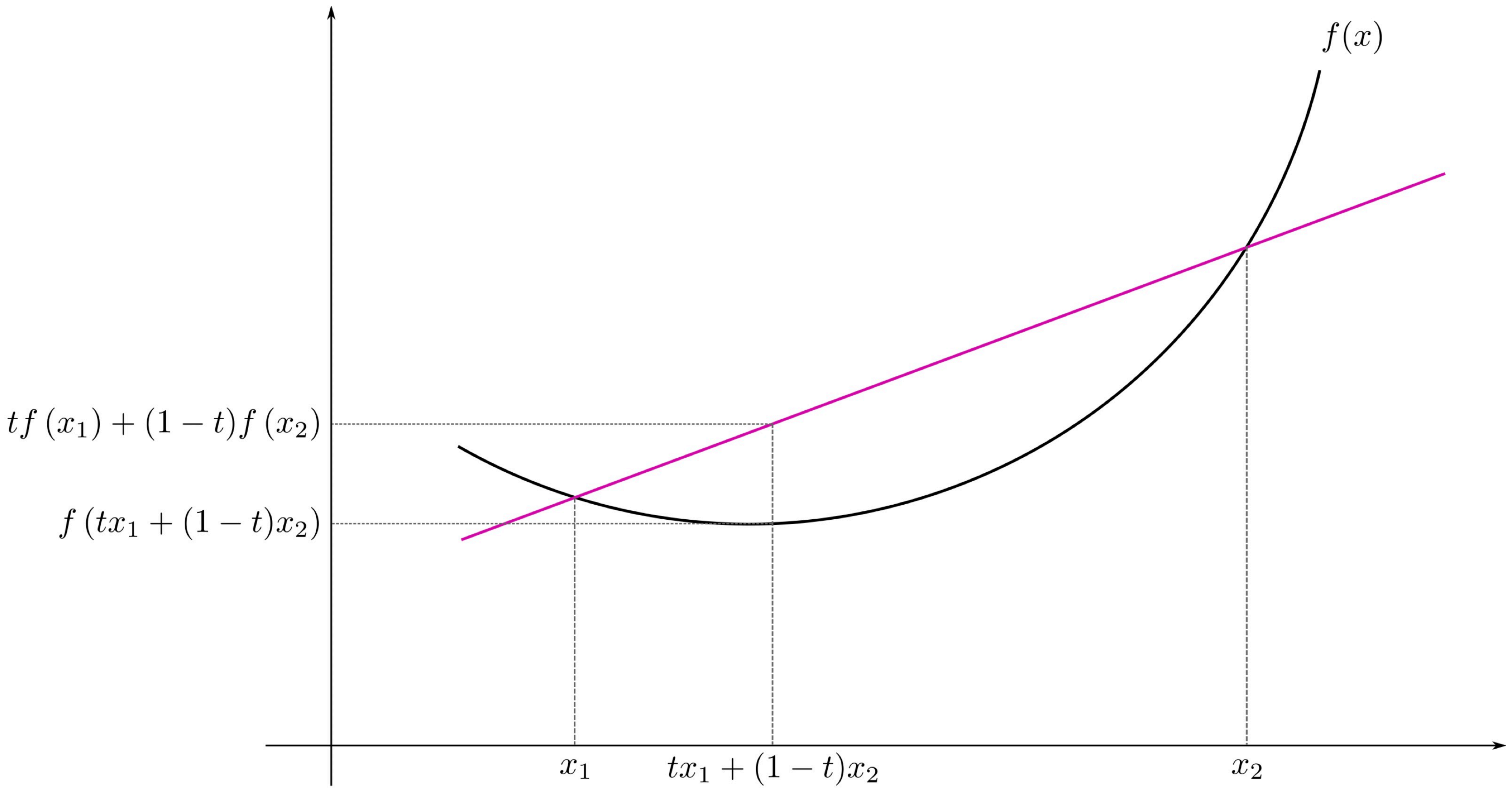
Convex function

- A function $f: \mathbb{R}^n \to \mathbb{R}$ is a convex function

- $\Leftrightarrow$ the function f is below any line segment between two points on f:

  - $\forall x_1, x_2, \forall t \in [0,1],$

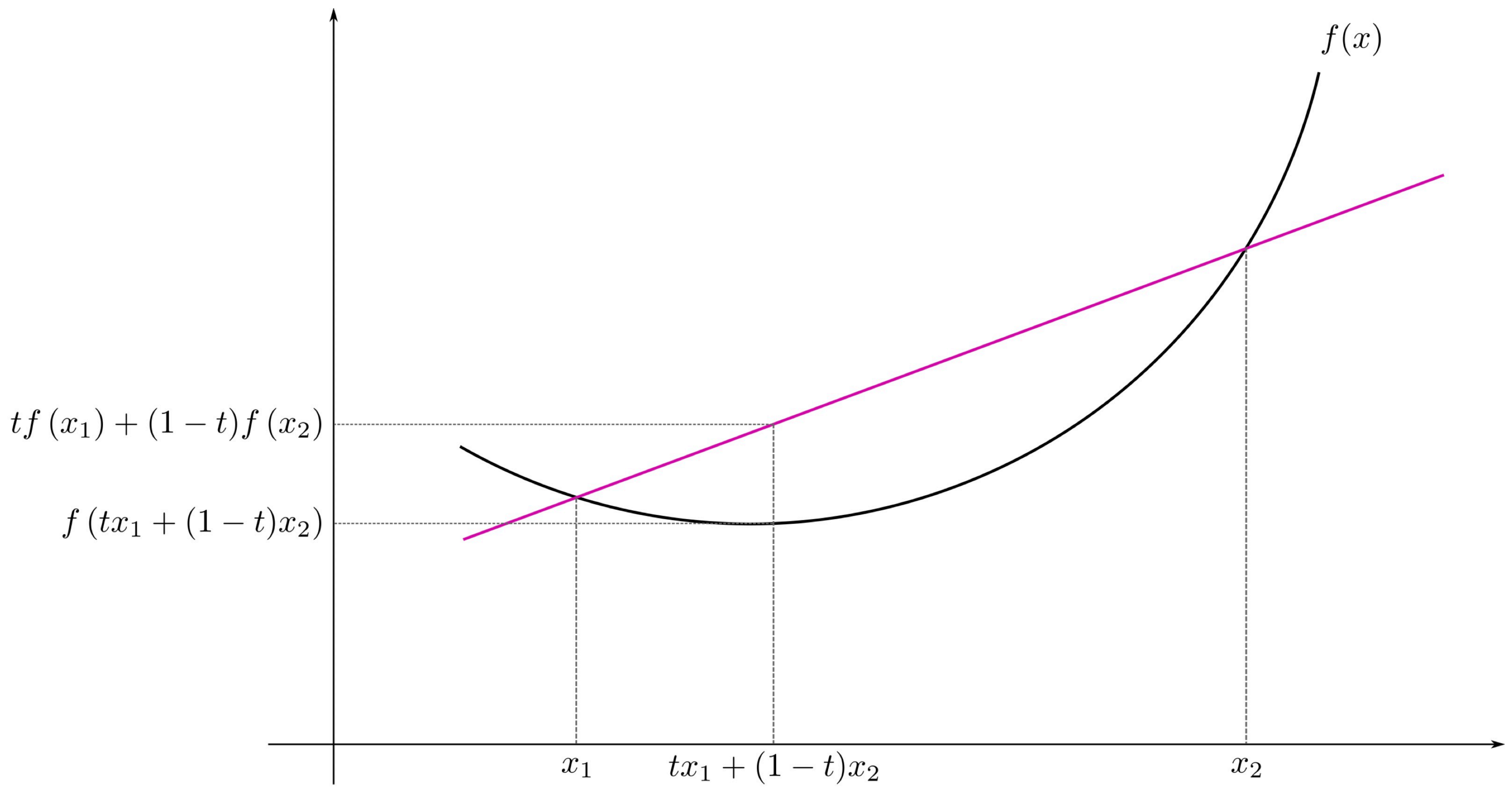
  - $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$



Optimization

Convex function

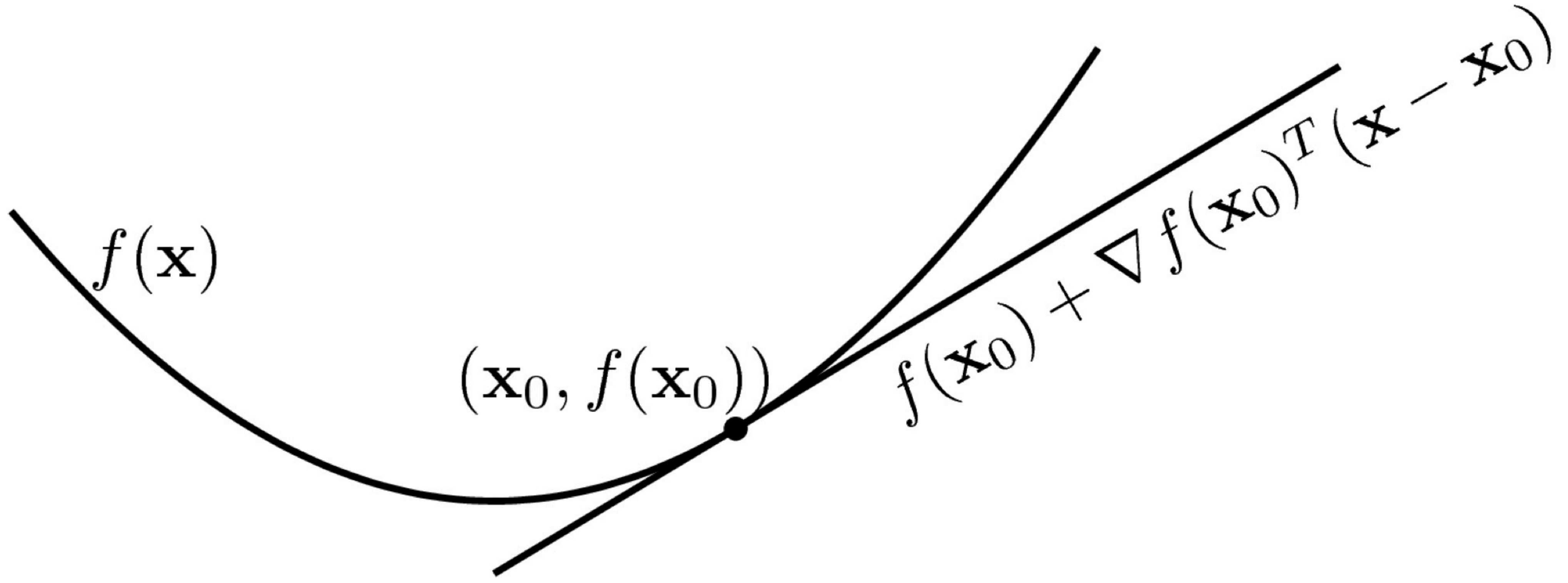
- A function $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function
- $\Leftrightarrow$ the function f is below any line segment between two points on f:
  - $\forall x_1, x_2, \forall t \in [0,1]$,
  - $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$
- Strictly convex:
  $f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$



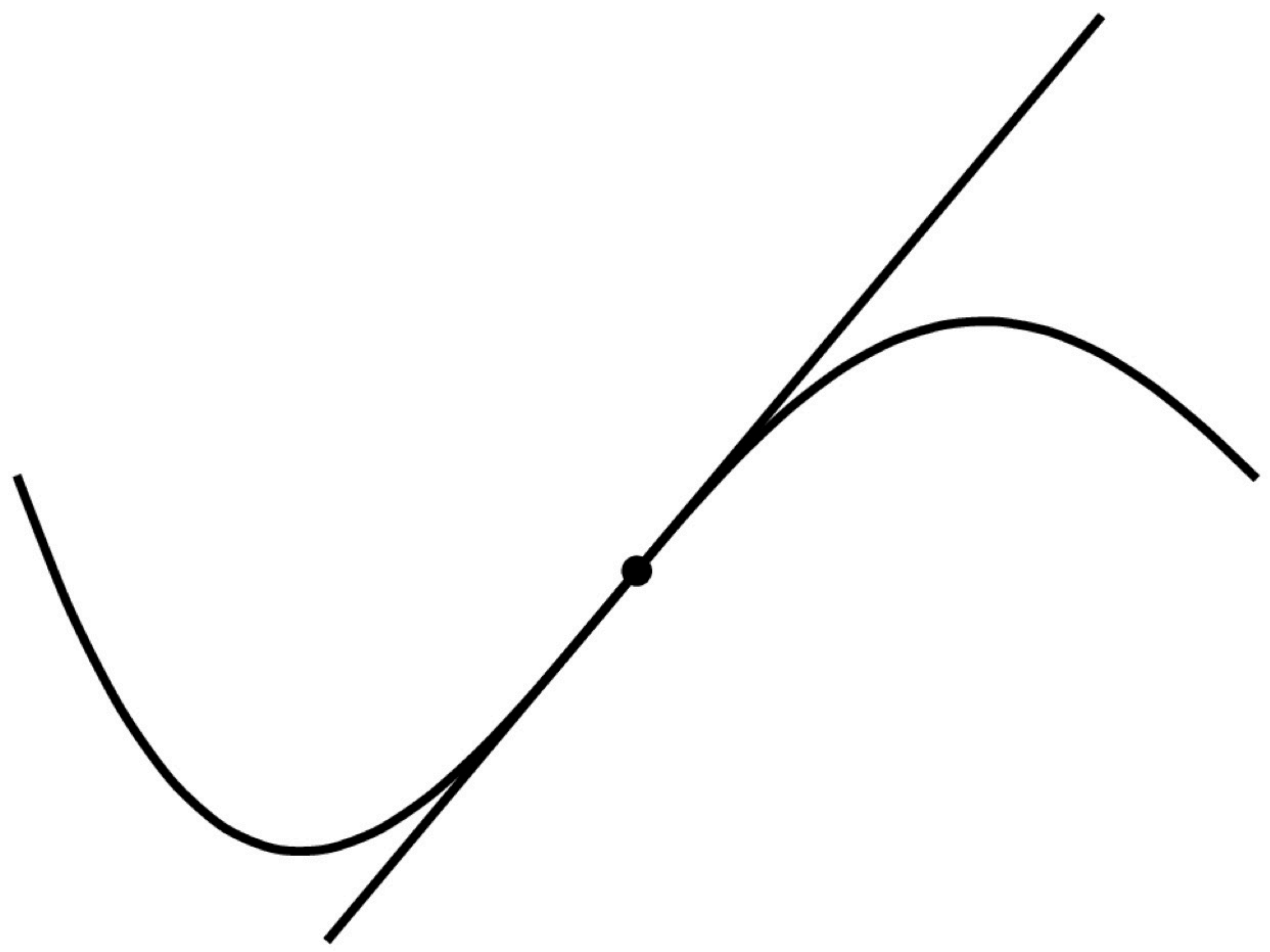
Optimization

Convex function

- Another equivalent definition for differentiable function:
  - f is convex if and only if $f(x) \geq f(x_0) + \nabla f(x_0)^T(x - x_0), \forall x, x_0$



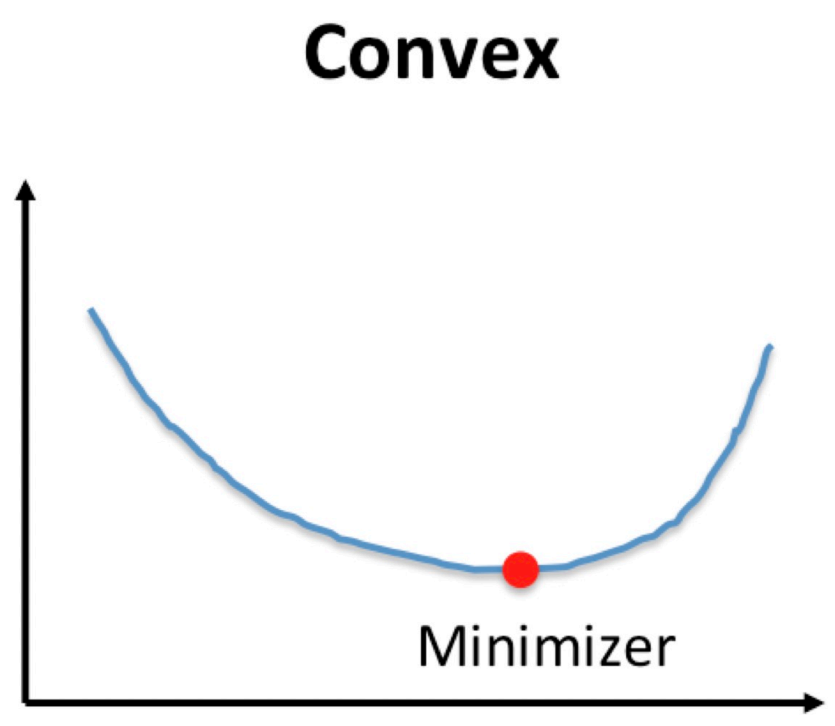
convex function



nonconvex function

Optimization

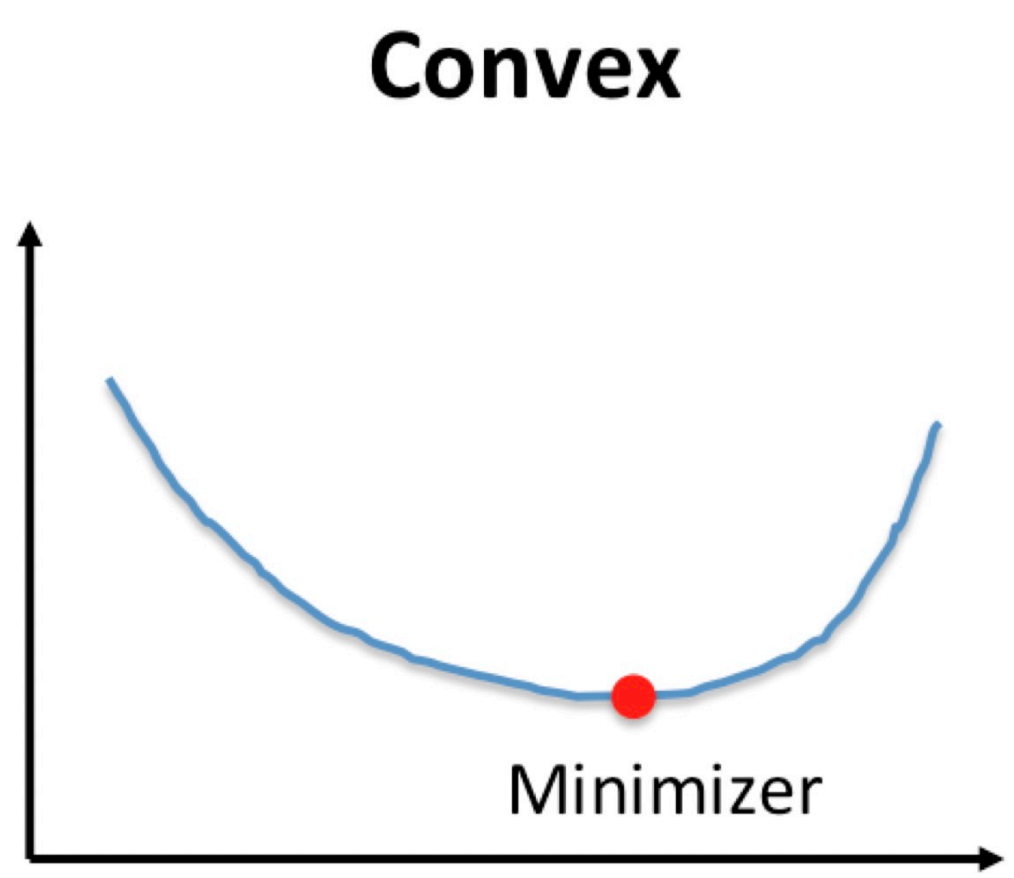
Convex function



- Convex function:
  - (For differentiable function) $\nabla f(w^*) = 0 \Leftrightarrow w^*$ is a global minimum
  - If f is twice differentiable $\Rightarrow$
    - F is convex if and only if $\nabla^2 f(w)$ is **positive semi-definite**
    - Example: linear regression, logistic regression, …

Optimization

Convex function

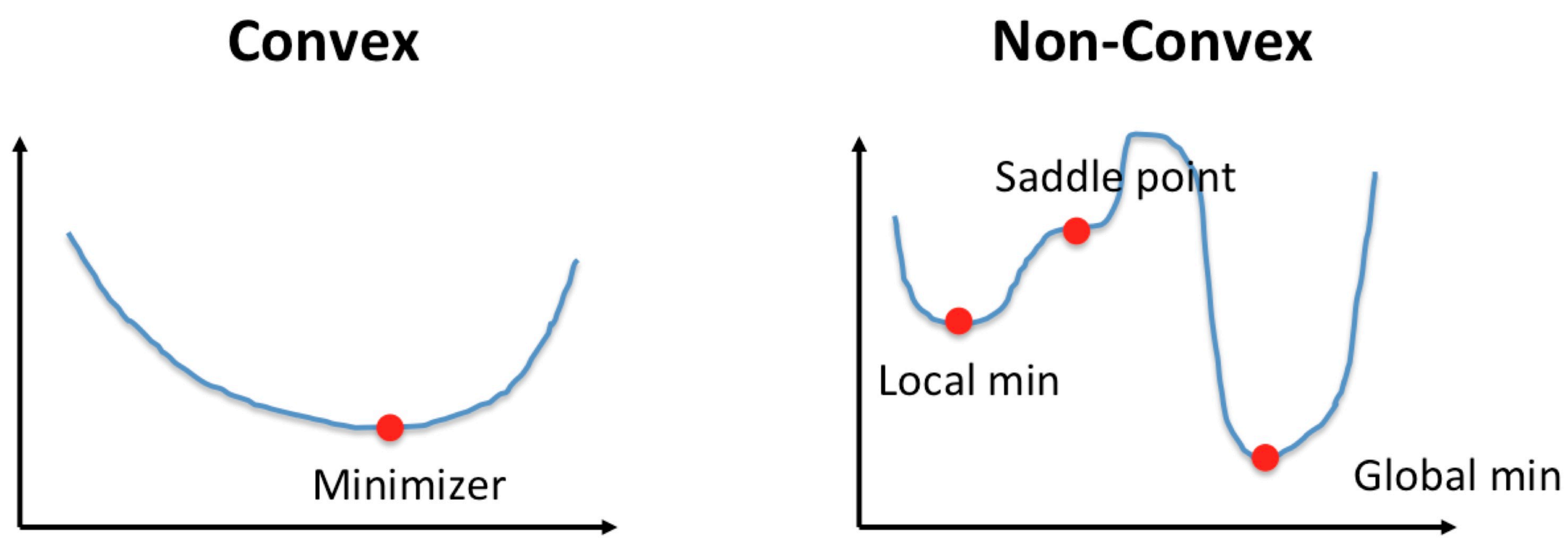


- Strict convex function:
  - $\nabla f(w^*) = 0 \Leftrightarrow w^*$ is the unique global minimum
    - Most algorithms only converge to gradient=0
    - Example: Linear regression when $X^T X$ is invertible

Optimization

Convex vs Nonconvex

- Convex function:
  - $\nabla f(x) = 0 \longleftrightarrow$ Global minimum
  - A function is convex if $\nabla^2 f(x)$ is positive definite
  - Example: linear regression, logistic rgression, …
- Non-convex function:
  - $\nabla f(x) = 0 \longleftrightarrow$ Global min, local min, or saddle point
    - Most algorithms only converge to gradient =0
    - Example: neural network, …



Optimization

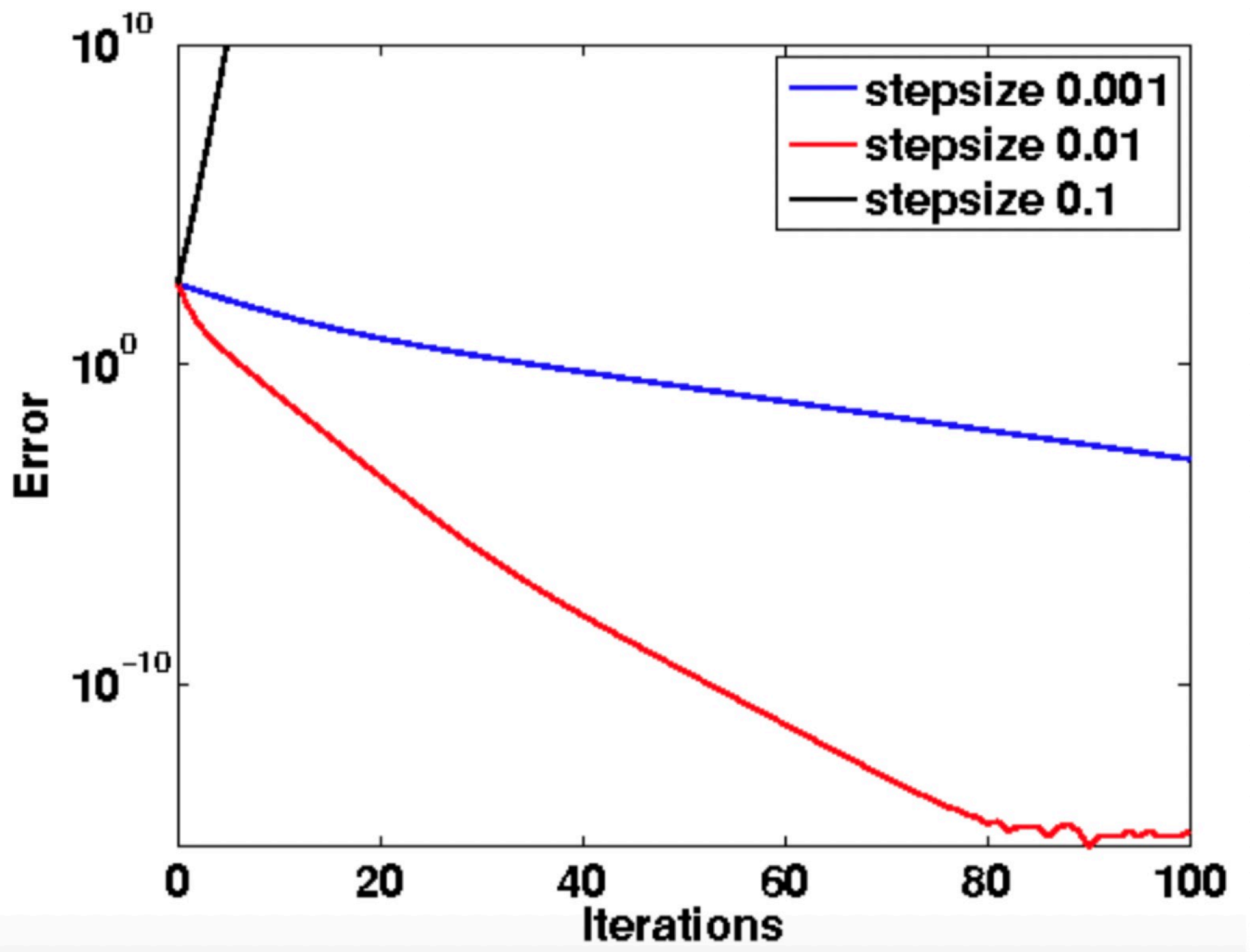
Gradient descent

- Gradient descent: repeatedly do
  - $w^{t+1} \leftarrow w^t - \alpha \nabla f(w^t)$
  - $\alpha > 0$ is the step size
- Generate the sequence $w^1, w^2, \ldots$
  - Converge to stationary points ($\lim_{t \to \infty} ||\nabla f(w^t)|| = 0$)

Optimization

Gradient descent

- Gradient descent: repeatedly do
  - $w^{t+1} \leftarrow w^t - \alpha \nabla f(w^t)$
  - $\alpha > 0$ is the step size
- Generate the sequence $w^1, w^2, \dots$
  - Converge to stationary points ($\lim_{t \to \infty} \|\nabla f(w^t)\| = 0$)
  - Step size too large $\Rightarrow$ diverge;
  - too small $\Rightarrow$ slow convergence



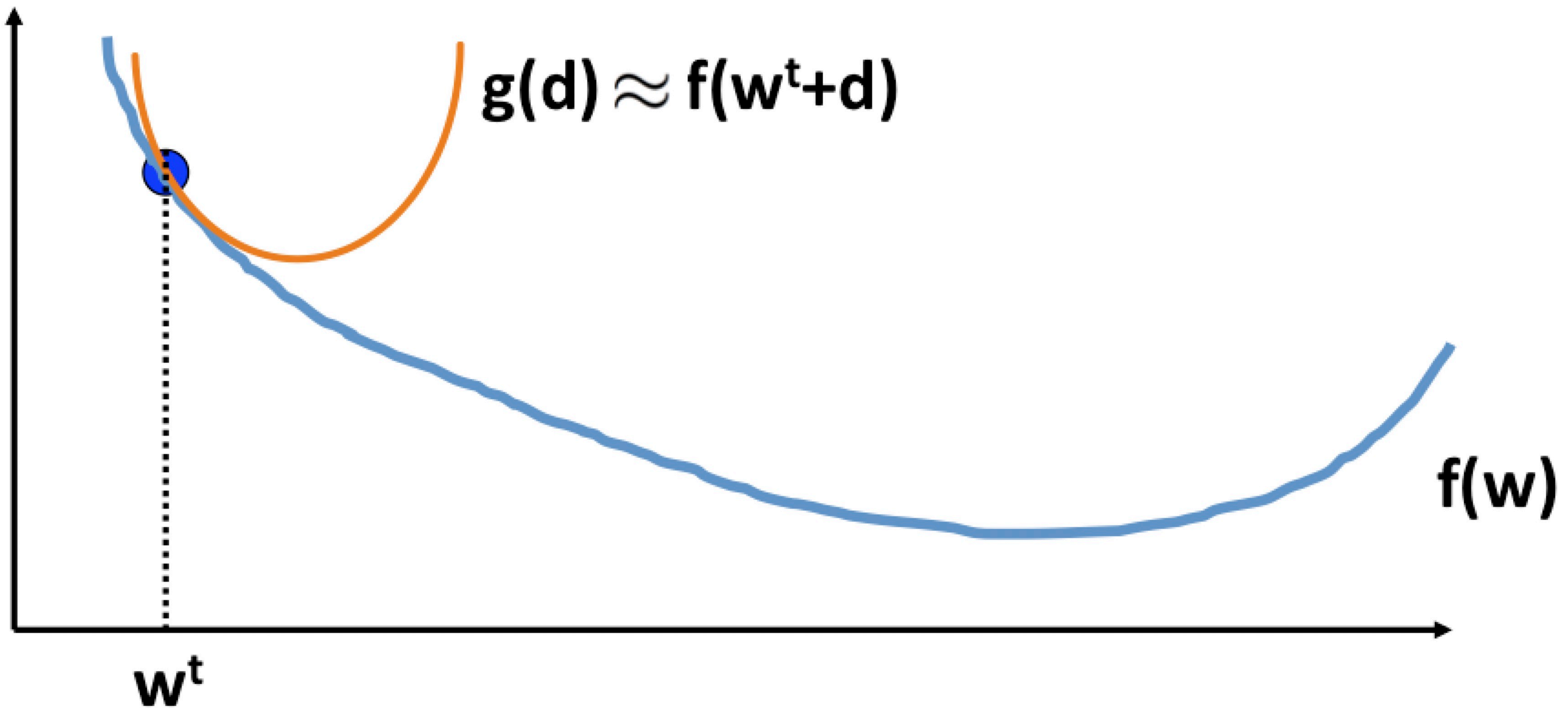
Optimization

Why gradient descent

- At each iteration, form a approximation function of $f(\cdot)$:
  - $f(w+d) \approx g(d) := f(w^t) + \nabla f(w^t) d + \frac{1}{2\alpha} \|d\|^2$
- Update solution by $w^{t+1} \leftarrow w^t + d*$
- $d* = \arg\min_d g(d)$
  - $\nabla g(d*) = 0 \Rightarrow \nabla f(w^t) + \frac{1}{\alpha} d* = 0 \Rightarrow d* = -\alpha \nabla f(w^t)$
- $d*$ will decrease $f(\cdot)$ if α (step size) is sufficiently small

Optimization

Illustration of gradient descent

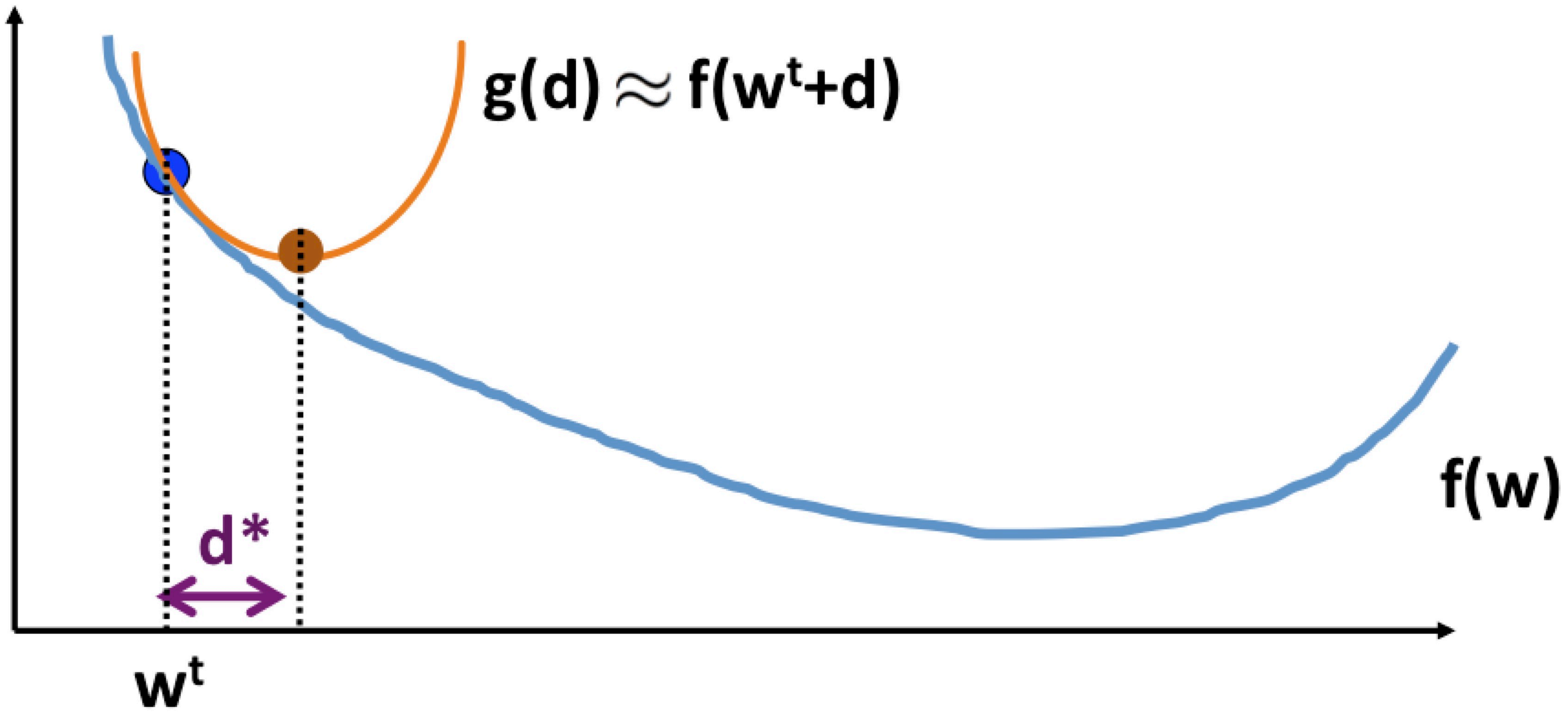


- Form a quadratic approximation

  - $f(w+d) \approx g(d) := f(w^t) + \nabla f(w^t)d + \frac{1}{2\alpha}\|d\|^2$

Optimization

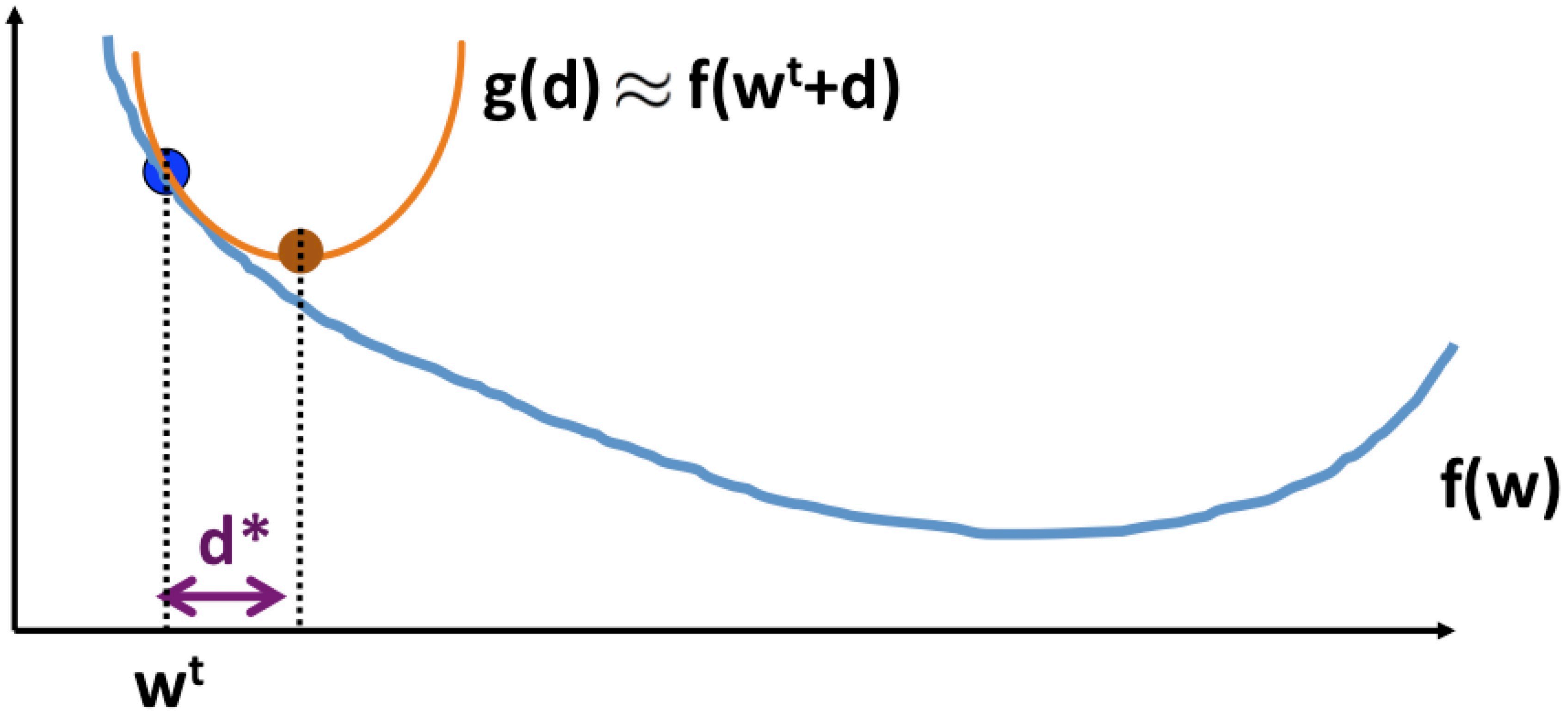
Illustration of gradient descent



- Minimize $g(d)$
  - $\nabla g(d^*) = 0 \Rightarrow \nabla f(w^t) + \frac{1}{\alpha} d^* = 0 \Rightarrow d^* = -\alpha \nabla f(w^t)$

Optimization

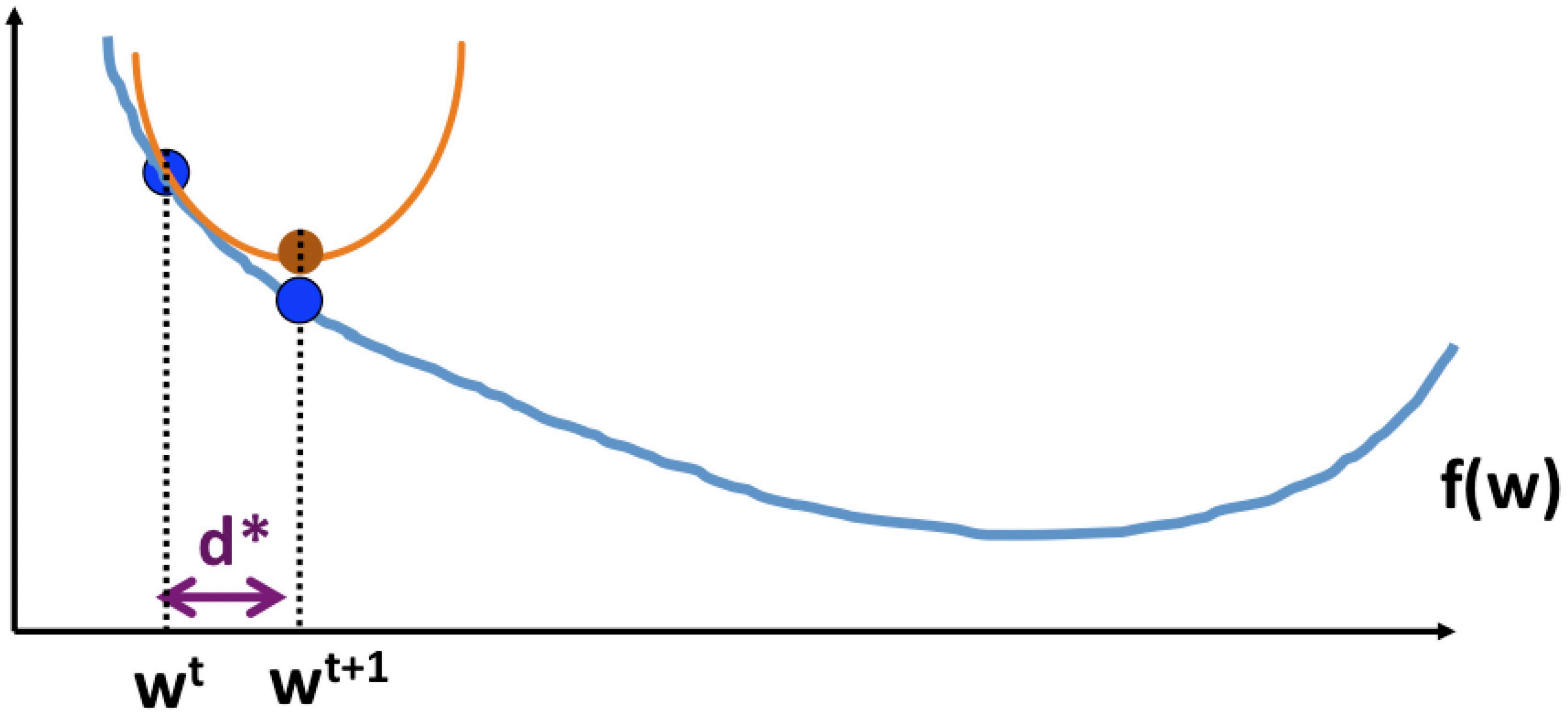
Illustration of gradient descent



- Update w
  - $w^{t+1} = w^t + d^* = w^t - \alpha \nabla f(w^t)$

Optimization

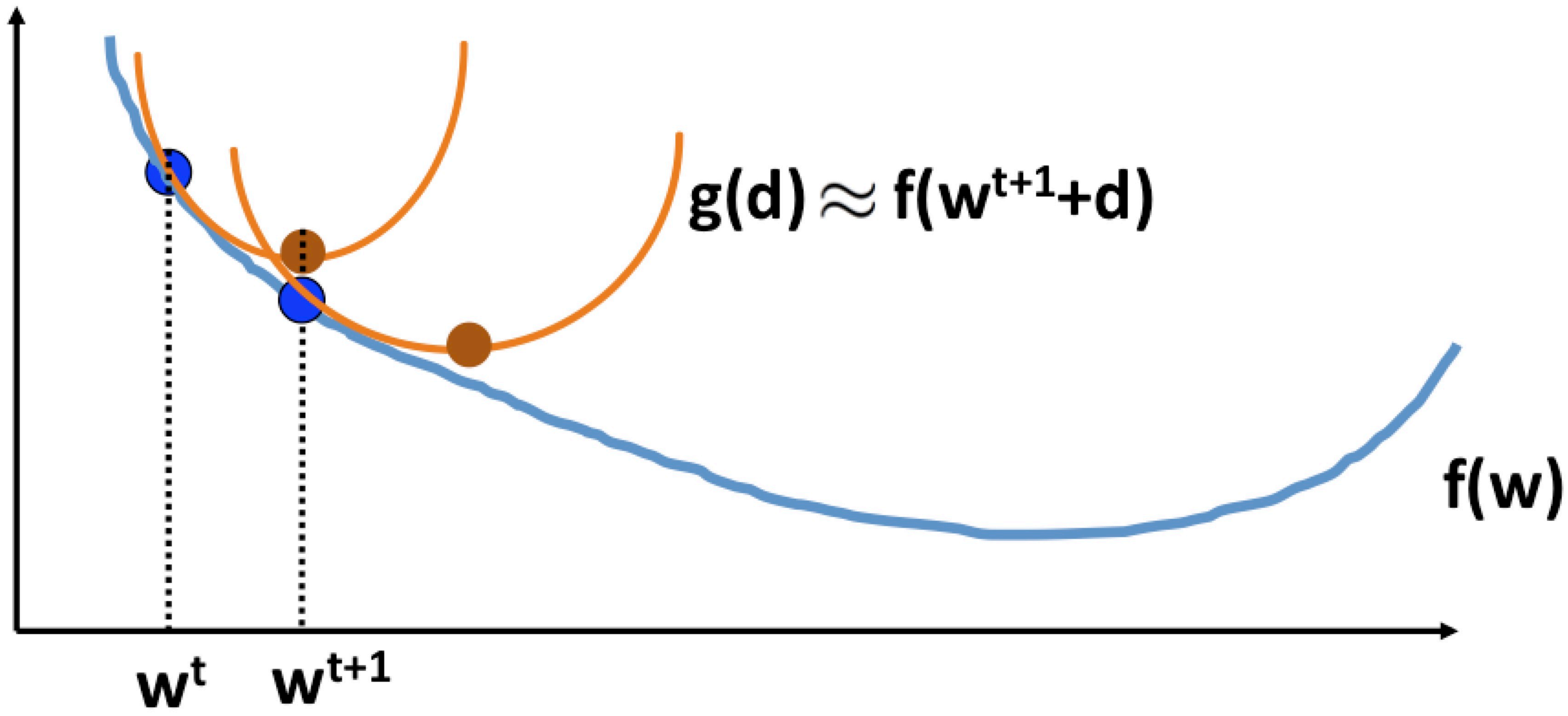
Illustration of gradient descent



- Update w
  - $w^{t+1} = w^t + d* = w^t - \alpha \nabla f(w^t)$

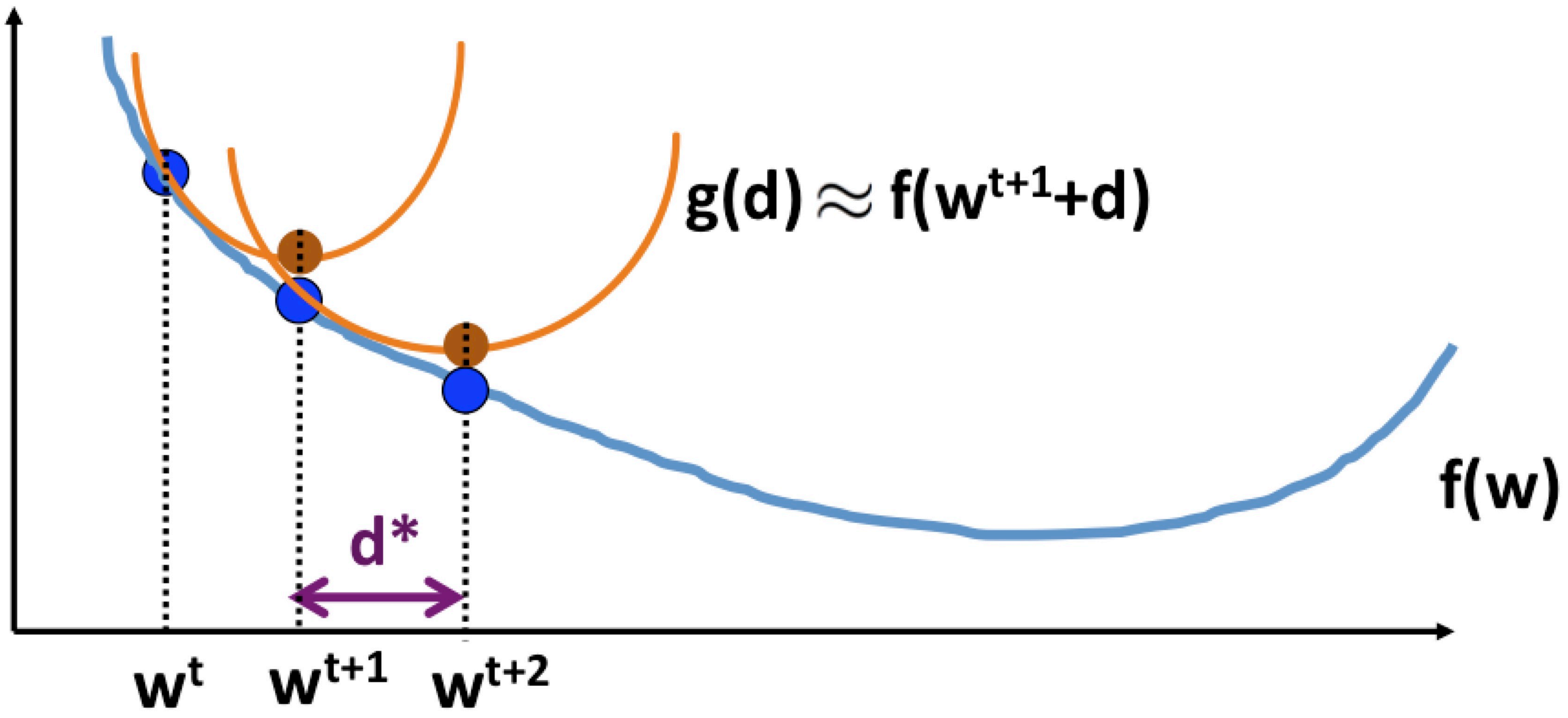
Optimization

Illustration of gradient descent



Optimization

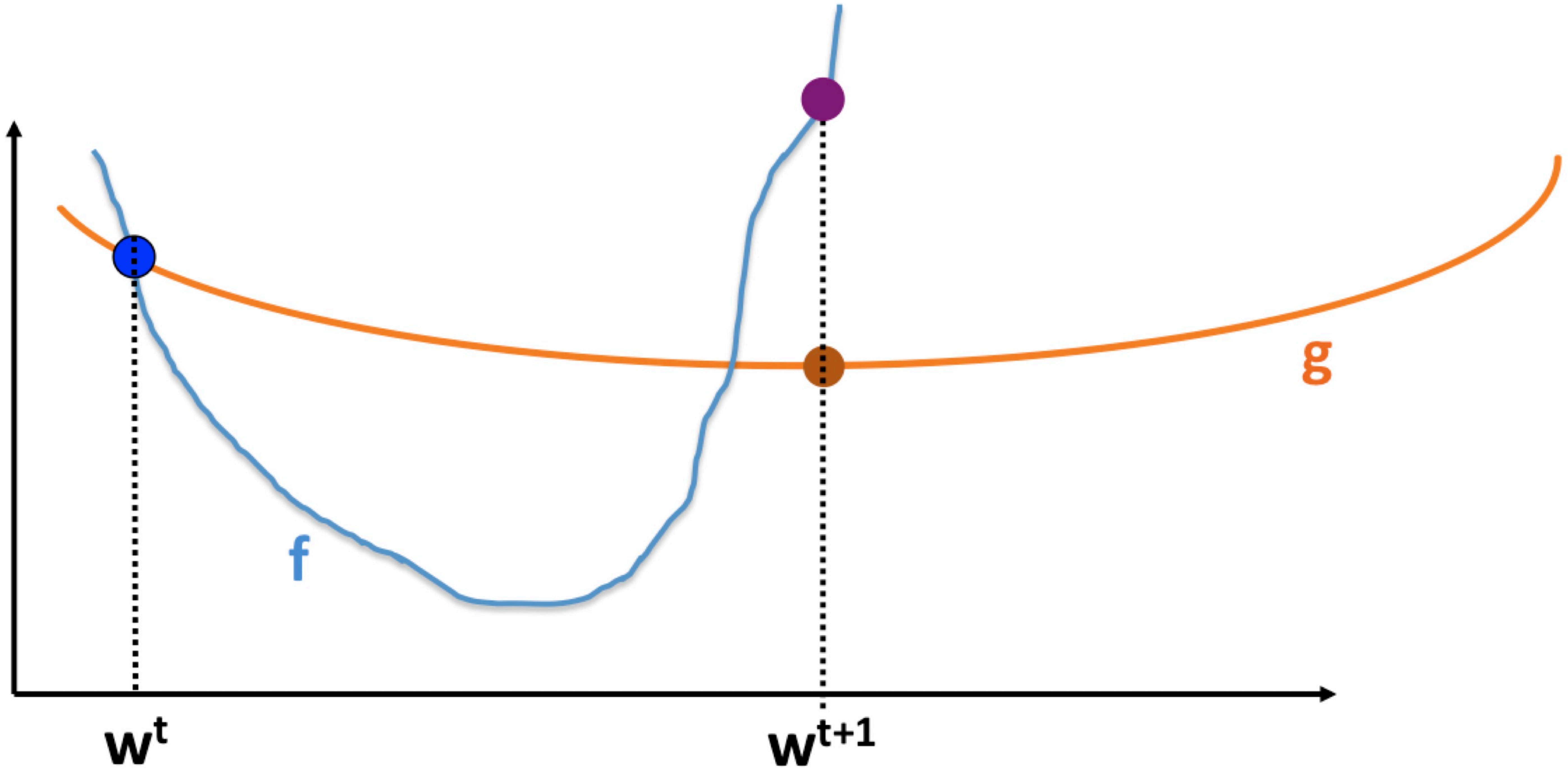
Illustration of gradient descent



Optimization

When will it diverge

Can diverge ($f(w^t) < f(w^{t+1})$) if g is not an upper bound of f

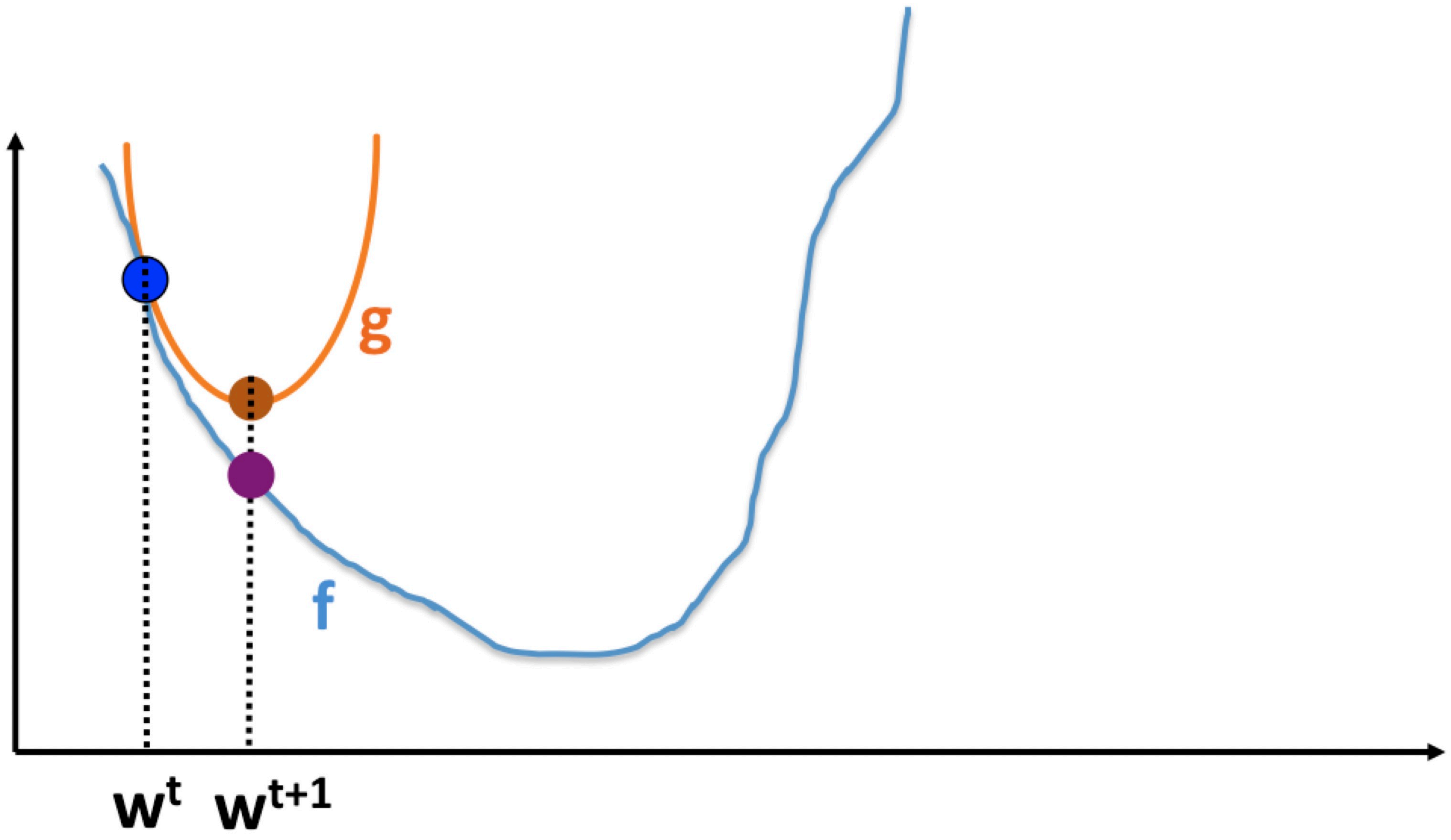


f(w^t) < f(w^{t+1}), diverge because g's curvature is too small

Optimization

When will it converge

Always converge ($f(w^t) > f(w^{t+1})$) if g is an upper bound of f



f(w^t) > f(w^{t+1}), converge when g's curvature is large enough

Optimization

Convergence

- A differential function f is said to be L-Lipschitz continuous:
  - $\|f(x_1) - f(x_2)\|_2 \leq L\|x_1 - x_2\|_2$
- A differential function f is said to be L-smooth: its gradient are Lipschitz continuous:
  - $\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq L\|x_1 - x_2\|_2$
  - And we could get
    - $\nabla^2 f(x) \preceq LI$
    - $f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}L\|y - x\|^2$

Optimization

Convergence

- Let L be a Lipchitz constant ($\nabla^2 f(x) \preceq LI$ for all x)
- Theorem: gradient descent converges if $\alpha < \frac{1}{L}$
- In practice, we do not know L …
  - Need to tune step size when running gradient descent

Optimization

Applying to logistic regression

gradient descent for logistic regression

- Initialize the weights $\boldsymbol{w}_0$
- For $t = 1, 2, \cdots$
  - Compute the gradient

$$\nabla f(\boldsymbol{w}) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \boldsymbol{x}_n}{1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n}}$$

  - Update the weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f(\boldsymbol{w})$
- Return the final weights $\boldsymbol{w}$

Optimization

Applying to logistic regression

- When to stop?
  - Fixed number of iterations, or
  - Stop when $||\nabla f(w)|| < \epsilon$

gradient descent for logistic regression

- Initialize the weights $\boldsymbol{w}_0$
- For $t = 1, 2, \cdots$
  - Compute the gradient

$$\nabla f(\boldsymbol{w}) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \boldsymbol{x}_n}{1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n}}$$

  - Update the weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f(\boldsymbol{w})$
- Return the final weights $\boldsymbol{w}$

Optimization

Line search

- In practice, we do not know L …
  - Need to tune step size when running gradient descent
- Line Search: Select step size automatically (for gradient descent)

Optimization

Line search

- The back-tracking line search:
  - Start from some large α_0
  - Try $\alpha = \alpha_0, \alpha_0/2, \alpha_0/4, \ldots$
    - Stop when α satisfies some sufficient decrease condition

Optimization

Line search

- The back-tracking line search:
  - Start from some large α_0
  - Try $\alpha = \alpha_0, \alpha_0/2, \alpha_0/4, \ldots$
    - Stop when α satisfies some sufficient decrease condition
  - A simple condition: $f(w + \alpha d) < f(w)$

Optimization

Line search

- The back-tracking line search:
  - Start from some large α_0
  - Try $\alpha = \alpha_0, \alpha_0/2, \alpha_0/4, \ldots$
    - Stop when α satisfies some sufficient decrease condition
  - A simple condition: $f(w + \alpha d) < f(w)$
    - Often works in practice but doesn't work in theory

Optimization

Large-scale problem

- Machine learning: usually minimizing the training loss:
  - $\min_{w}\{\frac{1}{N}\sum_{n=1}^{N}\ell(w^T x_n, y_n)\} := f(w)$ (linear model)
  - $\min_{w}\{\frac{1}{N}\sum_{n=1}^{N}\ell(f_W(x_n), y_n)\} := f(w)$ (general hypothesis)
  - ℓ: loss function (e.g., $\ell(a, b) = (a - b)^2$)
- Gradient descent:
  - $w \leftarrow w - \eta \underbrace{\nabla f(w)}_{\text{Main computation}}$

Optimization

Large-scale problem

- Machine learning: usually minimizing the training loss:
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(w^T x_n, y_n)\} := f(w)$ (linear model)
  - $\min_w \{\frac{1}{N}\sum_{n=1}^{N} \ell(f_W(x_n), y_n)\} := f(w)$ (general hypothesis)
  - ℓ: loss function (e.g., $\ell(a, b) = (a - b)^2$)
- Gradient descent:
  - $w \leftarrow w - \eta \underbrace{\nabla f(w)}_{\text{Main computation}}$
- In general, $f(w) = \frac{1}{N}\sum_{n=1}^{N} f_n(w)$,
  - Each $f_n(w)$ only depends on (x_n, y_n)

Optimization

Stochastic gradient

- Gradient: $\nabla f(w) = \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(w),$
- Each gradient computation needs to go through all training samples
  - Slow when millions of samples
- Faster way to compare "approximate gradient"?

Optimization

Stochastic gradient

- Gradient: $\nabla f(w) = \frac{1}{N} \sum_{n=1}^{N} \nabla f_n(w),$
- Each gradient computation needs to go through all training samples
  - Slow when millions of samples
- Faster way to compare "approximate gradient"?
- Use stochastic sampling:
  - Sample a small subset $B \subseteq \{1, \ldots, N\}$
  - Estimated gradient
    - $\nabla f(w) \approx \frac{1}{B} \sum_{n \in B} \nabla f_n(w)$
    - $|B|$: batch size

Optimization

Stochastic gradient descent

Stochastic Gradient Descent (SGD)

- Input: training data $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$
- Initialize $\boldsymbol{w}$ (zero or random)
- For $t = 1, 2, \cdots$
  - Sample a small batch $B \subseteq \{1, \cdots, N\}$
  - Update parameter

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta^t \frac{1}{|B|} \sum_{n \in B} \nabla f_n(\boldsymbol{w})$$

- Extreme case: $|B| = 1 \Rightarrow$ Sample one training data at a time

Optimization

Logistic Regression by SGD

- Logistic regression
  - $$\min_{w} \frac{1}{N} \sum_{n=1}^{N} \underbrace{\log(1 + e^{-y_n w^T x_n})}_{f_n(w)}$$

SGD for Logistic Regression

- Input: training data $\{\boldsymbol{x}_n, y_n\}_{n=1}^{N}$
- Initialize $\boldsymbol{w}$ (zero or random)
- For $t = 1, 2, \cdots$
  - Sample a batch $B \subseteq \{1, \cdots, N\}$
  - Update parameter

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta^t \frac{1}{|B|} \sum_{i \in B} \underbrace{\frac{-y_n \boldsymbol{x}_n}{1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n}}}_{\nabla f_n(\boldsymbol{w})}$$

Optimization

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:
- $\mathbb{E}[\frac{1}{|B|}\sum_{n \in B} \nabla f_n(w)] = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(w) = \nabla f(w)$

Optimization

Why SGD works?

- Stochastic gradient is an unbiased estimator of full gradient:
  - $\mathbb{E}[\frac{1}{|B|}\sum_{n\in B}\nabla f_n(w)] = \frac{1}{N}\sum_{n=1}^{N}\nabla f_n(w) = \nabla f(w)$
- Each iteration updated by
  - Gradient + zero-mean noise

Optimization

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?

Optimization

Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers

Optimization

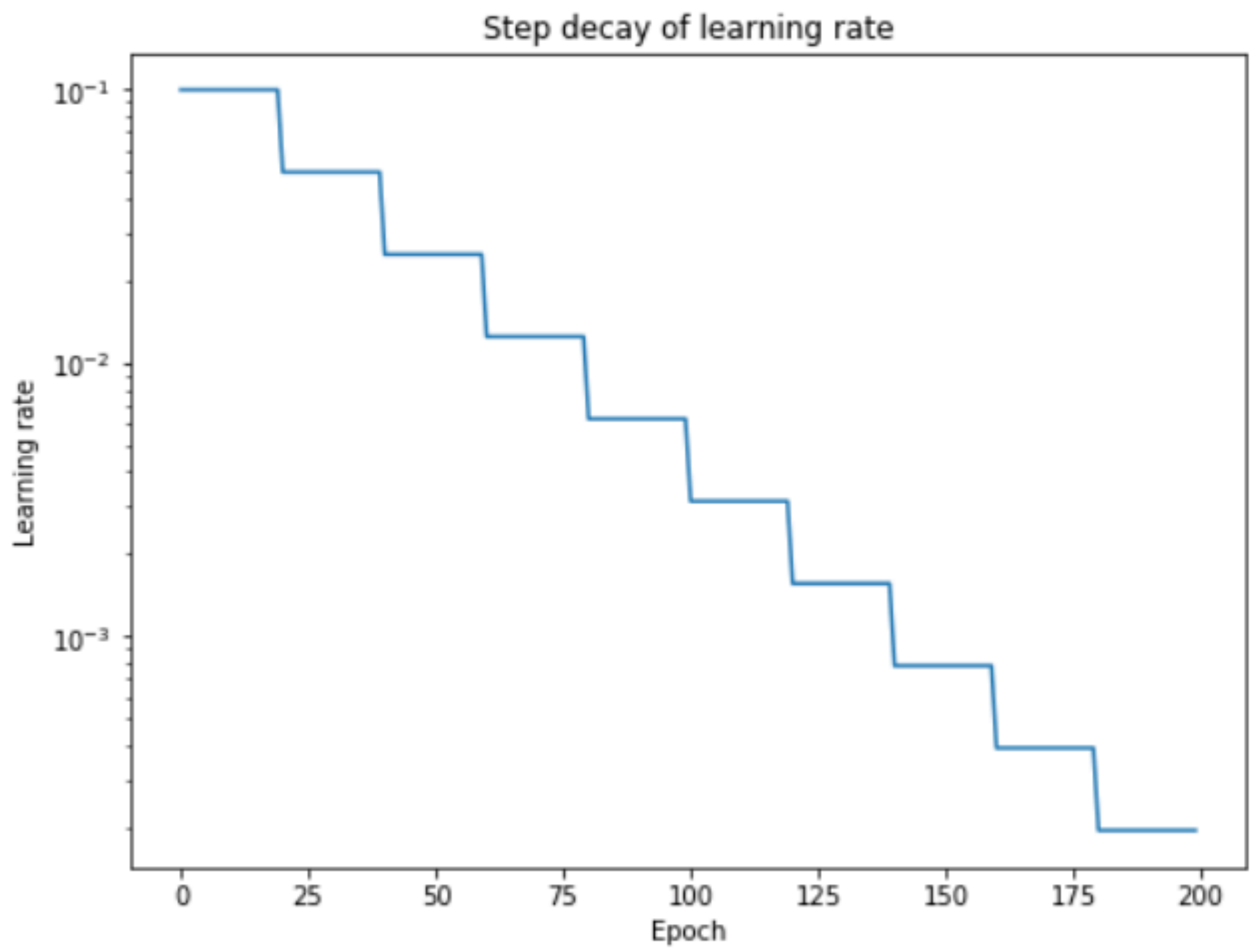
Stochastic gradient descent

- In gradient descent, η (step size) is a fixed constant
- Can we use fixed step size for SGD?
- SGD with fixed step size cannot converge to global/local minimizers
- If w^* is the minimizer, $\nabla f(w^*) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(w^*) = 0,$
  - But $\frac{1}{|B|}\sum_{n \in B} \nabla f_n(w) \neq 0$ if B is a subset
  - (Even if we got minimizer, SGD will move away from it)

Optimization

Stochastic gradient descent: step size

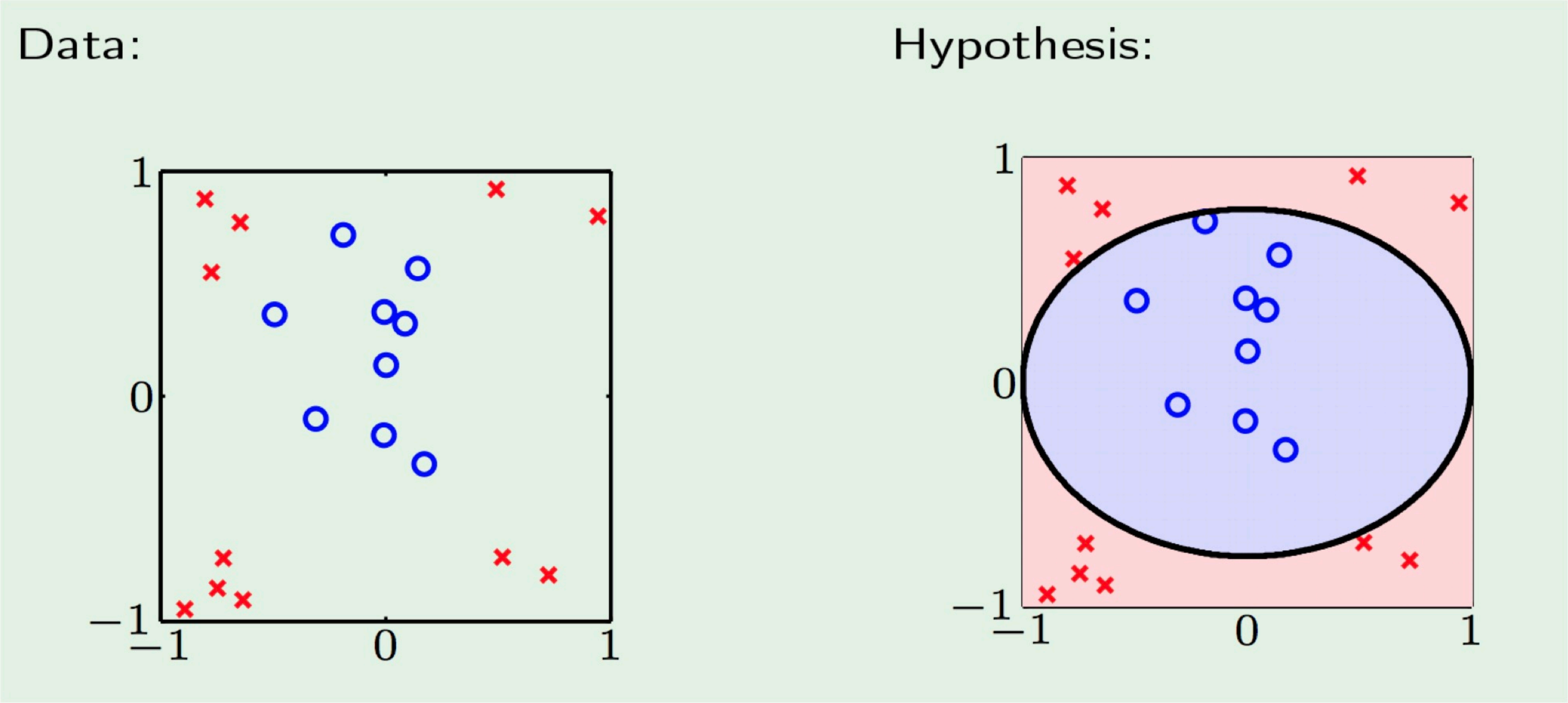
- To make SGD converge:
  - Step size should decrease to 0
  - $\eta^t \to 0$
  - Usually with polynomial rate $\eta^t \approx t^{-a}$ with constant a
- Step decay of learning rate



Nonlinear transformation

Linear hypotheses

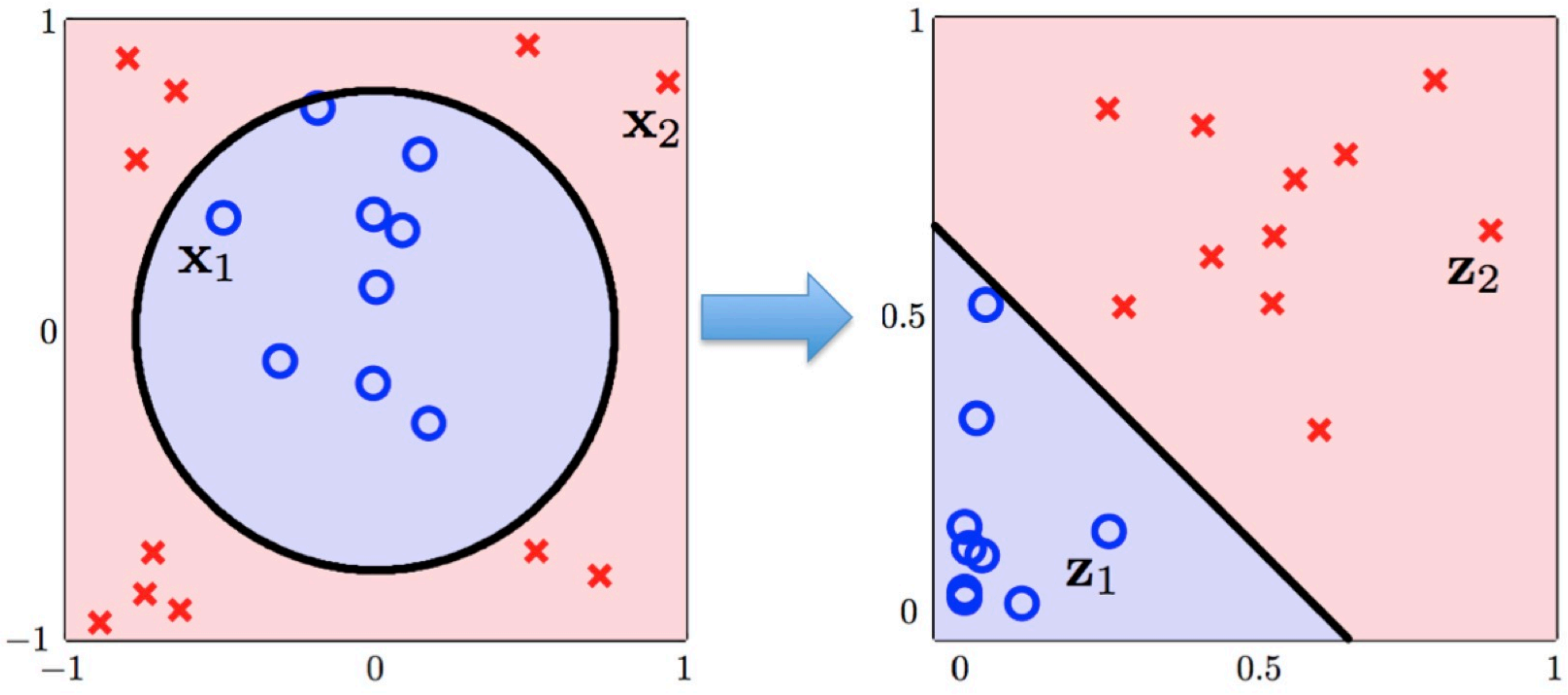
- Up to now: linear hypotheses
    - Perception, Linear regression, Logistic regression, …
- Many problems are not linearly separable



Nonlinear transformation

Circular Separable and Linear Separable

- $$h(x) = \text{sign}(\underbrace{0.6}_{\tilde{w}_0} \cdot \underbrace{1}_{\tilde{z}_0} + \underbrace{(-1)}_{\tilde{w}_1} \cdot \underbrace{x_1^2}_{\tilde{z}_1} + \underbrace{(-1)}_{\tilde{w}_2} \cdot \underbrace{x_2^2}_{\tilde{z}_2})$$
  $$= \text{sign}(\tilde{w}^T z)$$
- $\{(x_n, y_n)\}$ circular separable $\Rightarrow$ $\{(z_n, y_n)\}$ linear separable
- $x \in \mathcal{X} \rightarrow x \in \mathcal{Z}$ (using a nonlinear transformation ϕ)



Nonlinear Transformation

Definition

- Define nonlinear transformation
  - $\phi(\mathbf{x}) = (1, x_1^2, x_2^2) = (z_0, z_1, z_2) = \mathbf{z}$
- Linear hypotheses in $\mathcal{Z}$-space:
  - $\text{sign}(\tilde{h}(\mathbf{z})) = \text{sign}(\tilde{h}(\phi(\mathbf{x}))) = \text{sign}(w^T \phi(\mathbf{x}))$
- Line in $\mathcal{Z}$-space ⇔ some quadratic curves in $\mathcal{X}$-space

Nonlinear Transformation

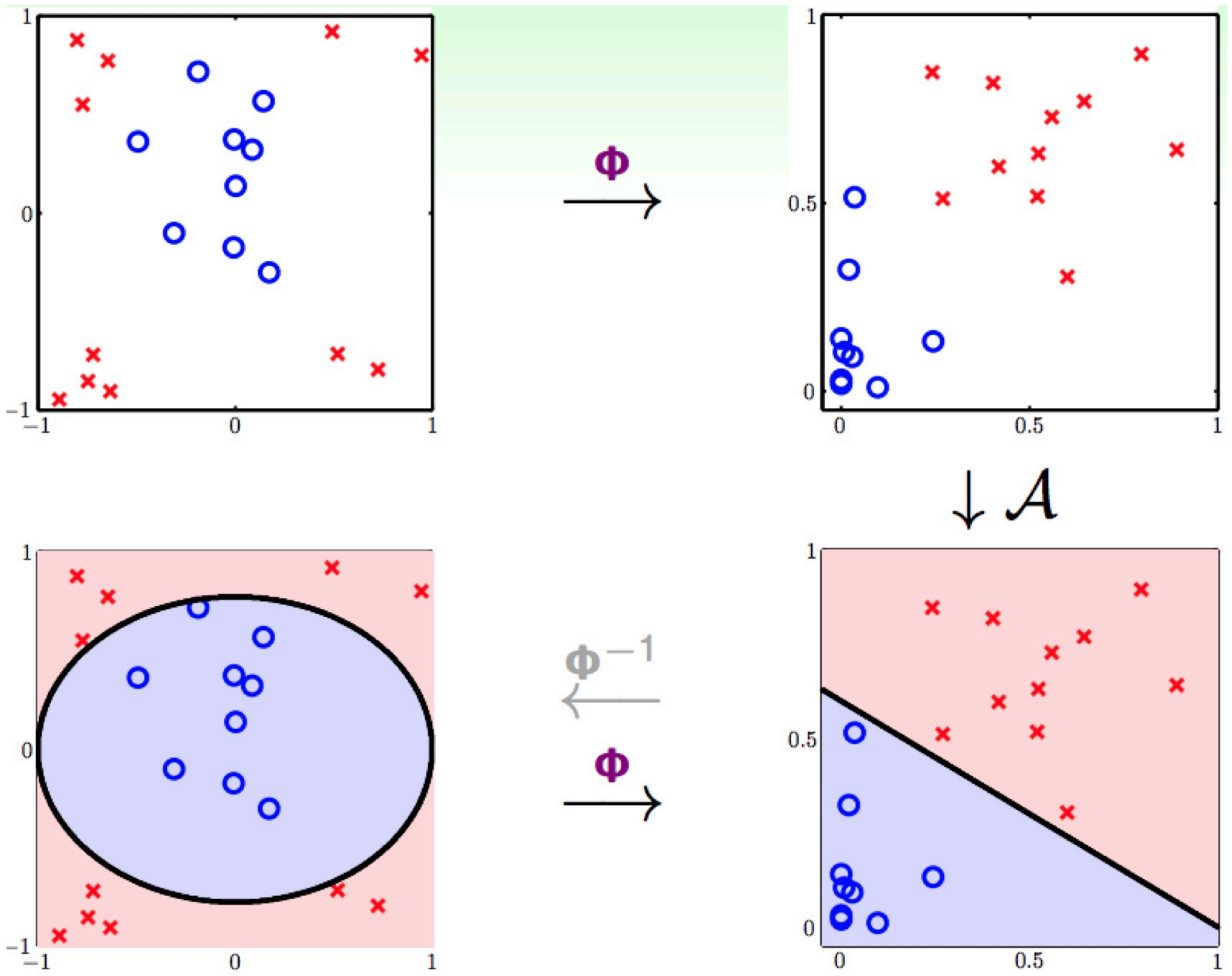
General Quadratic Hypothesis Set

- A "bigger " $\mathscr{Z}$-space:
  - $\phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$
- Linear in $\mathscr{Z}$-space ⇔ quadratic hypotheses in $\mathscr{X}$-space
- The hypotheses space:
  - $\mathscr{H}_{\phi_2} = \{h(x) : h(x) = \tilde{w}^T \phi_2(x)$ for some $\tilde{w}\}$ (quadratic hypotheses)
- Also include linear model as a degenerate case

Nonlinear transformation

Learning a good quadratic function

- Transform original data $\{x_n, y_n\}$ to $\{z_n = \phi(x_n), y_n\}$
- Solve a linear problem on $\{z_n, y_n\}$ using your favorite algorithm $\mathscr{A}$ to get a good model $\tilde{w}$
- Return the model $h(x) = \text{sign}(\tilde{w}^T \phi(x))$



Nonlinear transformation

Polynomial mappings

- Can now freely do quadratic classification, quadratic regression
- Can easily extend to any degree of polynomial mappings
  - E.g.,
    $\phi(x) = (x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2^2, x_1x_3^2, x_1x_2^2, x_2^2x_3, x_2^2x_3, x_1^3, x_2^3, x_3^3)$

Nonlinear Transformation

The price we pay: computational complexity

- Q-th oder polynomial transform:

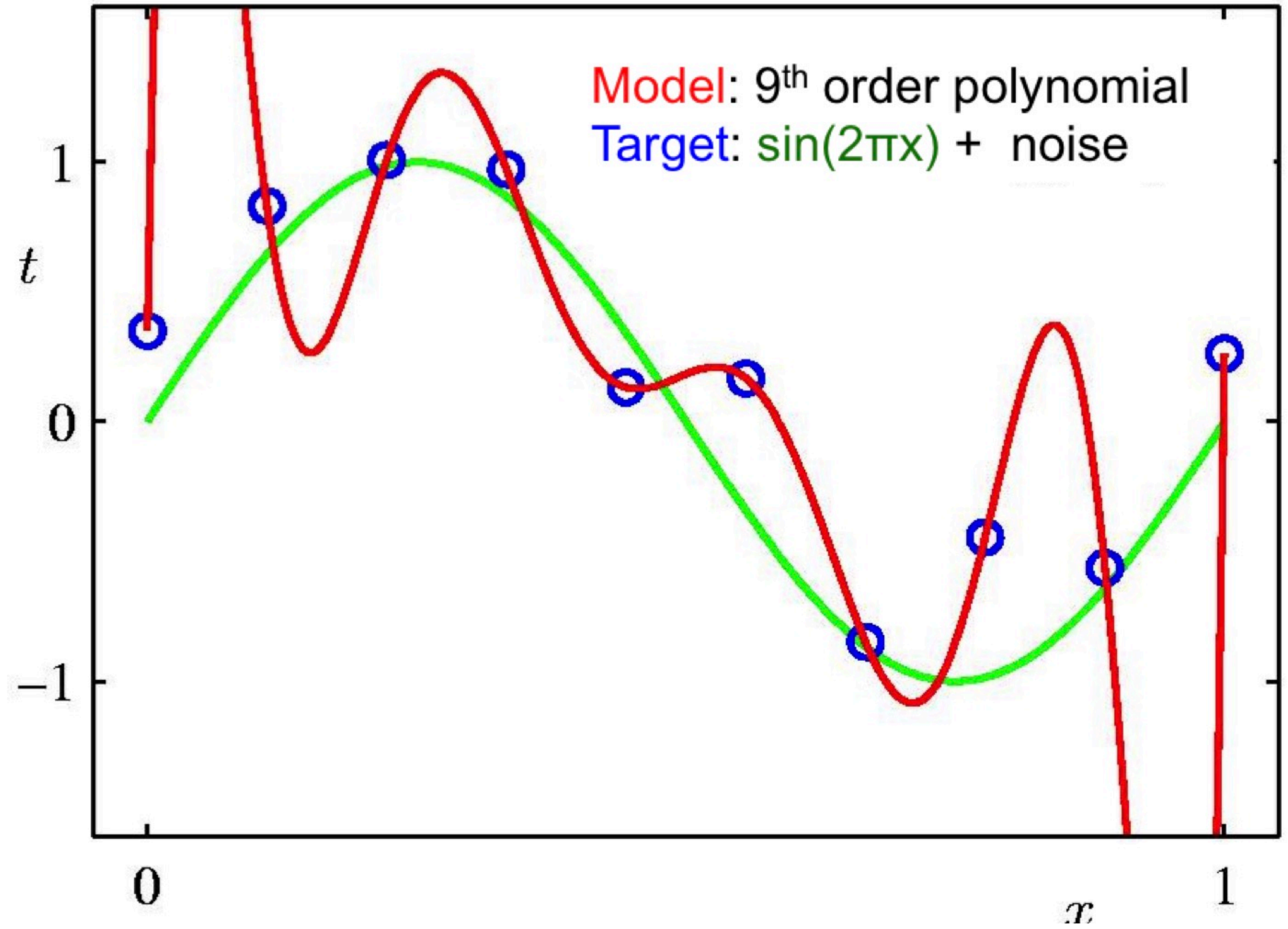
$$\phi(x) = (1, x_1, x_2, \ldots, x_d,$$
$$x_1^2, x_1 x_2, \ldots, x_d^2, \ldots, x_d^2,$$
$$\cdots$$
$$x_1^Q, x_1^{Q-1} x_2, \ldots, x_d^Q)$$

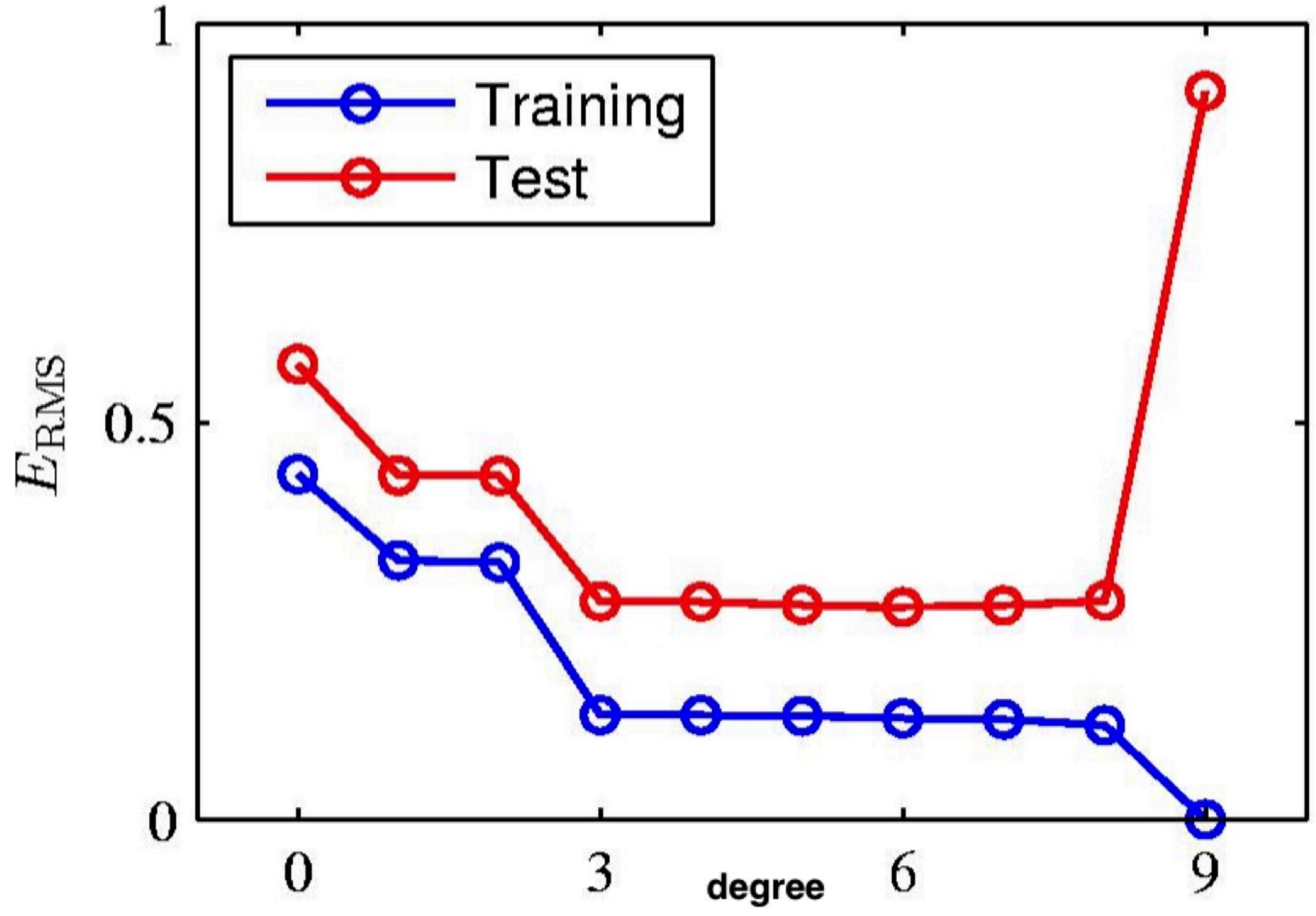
- $O(d^Q)$ dimensional vector ⇒ High computational cost
  - Kernel method

Nonlinear Transformation

The price we pay: overfitting

- Overfitting: the model has low training error but high prediction error





Theory of Generalization

Training versus testing

- Machine learning pipeline:
  - Training phase:
    - Obtain the best model h by minimizing training error
  - Test (inference) phase:
    - For any incoming test data x"
      - Make prediction by $h(x)$
    - Measure the performance of h: test error

Theory of Generalization

Training versus testing

- Does low training error imply low test error?
  - They can be totally different if
    - train distribution $\neq$ test distribution

Theory of Generalization

Training versus testing

- Does low training error imply low test error?
  - They can be totally different if
    - train distribution $\neq$ test distribution
  - Even under the same distribution, they can be very different:
    - Because h is picked to minimize training error, not test error

Theory of Generalization

Formal definition

- Assume training and test data are both sampled from D
- The ideal function (for generating labels) is $f: f(x) \rightarrow y$
- Training error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{tr}(h) = \frac{1}{N}\sum_{n=1}^{N} e(h(x_n), f(x_n))$
  - h is determined by $x_1, \ldots, x_n$
- Test error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{te}(h) = \frac{1}{M}\sum_{m=1}^{M} e(h(x_m), f(x_m))$
  - h is independent to $x_1, \ldots, x_n$

Theory of Generalization
Formal definition

- Assume training and test data are both sampled from D
- The ideal function (for generating labels) is $f: f(x) \to y$
- Training error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{tr}(h) = \frac{1}{N} \sum_{n=1}^{N} e(h(x_n), f(x_n))$
  - h is determined by $x_1, \ldots, x_n$
- Test error: Sample $x_1, \ldots, x_N$ from D and
  - $E_{te}(h) = \frac{1}{M} \sum_{m=1}^{M} e(h(x_m), f(x_m))$
  - h is independent to $x_1, \ldots, x_n$
- Generalization error = Test error = Expected performance on D:
  - $E(h) = \mathbb{E}_{x \sim D}[e(h(x), f(x))] = E_{te}(h)$

Theory of Generalization

The 2 questions of learning

- $E(h) \approx 0$ is achieved through:
  - $E(h) \approx E_{tr}(h)$ and $E_{tr}(h) \approx 0$

Theory of Generalization

The 2 questions of learning

- $E(h) \approx 0$ is achieved through:
  - $E(h) \approx E_{tr}(h)$ and $E_{tr}(h) \approx 0$
- Learning is split into 2 questions:
  - Can we make sure that $E(h) \approx E_{tr}(h)$?
    - Generalization
  - Can we make $E_{tr}(h)$ small?
    - Optimization

Theory of Generalization

Connection to Learning

- Given a function h
- If we randomly draw $x_1, \ldots, x_n$ (independent to h):
  - $E(h) = \mathbb{E}_{x \sim D}[h(x) \neq f(x)] \Leftrightarrow \mu$ (generalization error, unknown)
  - $\frac{1}{N} \sum_{n=1}^{N} [h(x_n) \neq y_n] \Leftrightarrow \nu$ (error on sampled data, known)
- Based on Hoeffding's inequality:
  - $p[\,|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$
- "$\mu = \nu$" Is probably approximately correct (PAC)
- However, this can only "verify" the error of a hypothesis:
  - h and $x_1, \ldots, x_N$ must be independent

Theory of Generalization

A simple solution

- For each particular h,
  - $P[\,|E_{tr}(h) - E(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}$
- If we have a hypothesis set $\mathscr{H}$,we want to derive the bound for $P[\text{sup}_{h\in\mathscr{H}}|E_{tr}(h) - E(h)| > \epsilon]$
  - $P[\,|E_{tr}(h_1) - E(h_1)| > \epsilon]$ or … or $P[\,|E_{tr}(h_{|\mathscr{H}|}) - E(h_{|\mathscr{H}|})| > \epsilon]$
  - $\leq \sum_{m=1}^{\mathscr{H}} P[\,|E_{tr}(h_m) - E(h_m)|\,] \leq 2|\mathscr{H}|e^{-2\epsilon^2 N}$
    - Because of union bound inequality $P(\bigcup_{i=1}^{\infty} A_i) \leq \sum_{i=1}^{\infty} P(A_i)$

Theory of generalization

When is learning successful?

- When our learning algorithm $\mathscr{A}$ picks the hypothesis g:
  - $P[\text{SUP}_{h \in \mathscr{H}} | E_{tr}(h) - E(h) | > \epsilon] \leq 2 | \mathscr{H} | e^{-2\epsilon^2 N}$
- If $|\mathscr{H}|$ is small and N is large enough:
  - If $\mathscr{A}$ finds $E_{tr}(g) \approx 0 \Rightarrow E(g) \approx 0$ (Learning is successful!)

Theory of Generalization

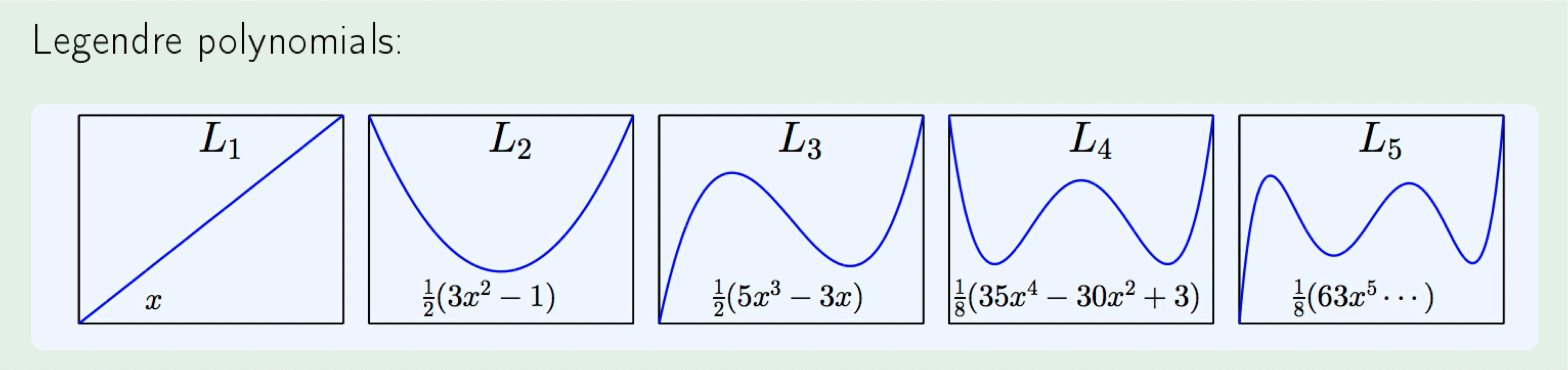
Feasibility of Learning

- $P[\,|E_{tr}(g) - E(g)| > \epsilon] \leq 2\,|\mathscr{H}|\,e^{-2\epsilon^2 N}$
  - Two questions:
    - 1. Can we make sure $E(g) \approx E_{tr}(g)$?
    - 2. Can we make sure $E_{tr}(g) \approx 0$?
- $|\mathscr{H}|$: complexity of model
  - Small $|\mathscr{H}|$: 1 holds, but 2 may not hold (too few choices) (under-fitting)
  - Large $|\mathscr{H}|$: 1 doesn't hold, but 2 may hold (over-fitting)

Regularization

The polynomial model

- $\mathscr{H}_Q$: polynomials of order Q
  - $\mathscr{H}_Q = \{ \sum_{q=0}^{Q} w_q L_q(x) \}$
- Linear regression in the $\mathscr{Z}$ space with
  - $z = [1, L_1(x), \ldots, L_Q(x)]$



Regularization

Unconstrained solution

- Input $(x_1, y_1), \ldots, (x_N, y_N) \to (z_1, y_1), \ldots, (z_N, y_N)$
- Linear regression:
  - Minimize: $E_{\text{tr}}(w) = \frac{1}{N} \sum_{n=1}^{N} (w^T z_n - y_n)^2$
  - Minimize: $\frac{1}{N}(Zw - y)^T(Zw - y)$
- Solution $w_{\text{tr}} = (Z^T Z)^{-1} Z^T y$

Regularization
Constraining the weights

- Hard constraint: $\mathscr{H}_2$ is constrained version of $\mathscr{H}_{10}$ (with $w_q = 0$ for $q > 2$)

Regularization

Constraining the weights

- Hard constraint: $\mathscr{H}_2$ is constrained version of $\mathscr{H}_{10}$ (with $w_q = 0$ for $q > 2$)
- Soft-order constraint: $\sum_{q=0}^{Q} w_q^2 \leq C$

Regularization

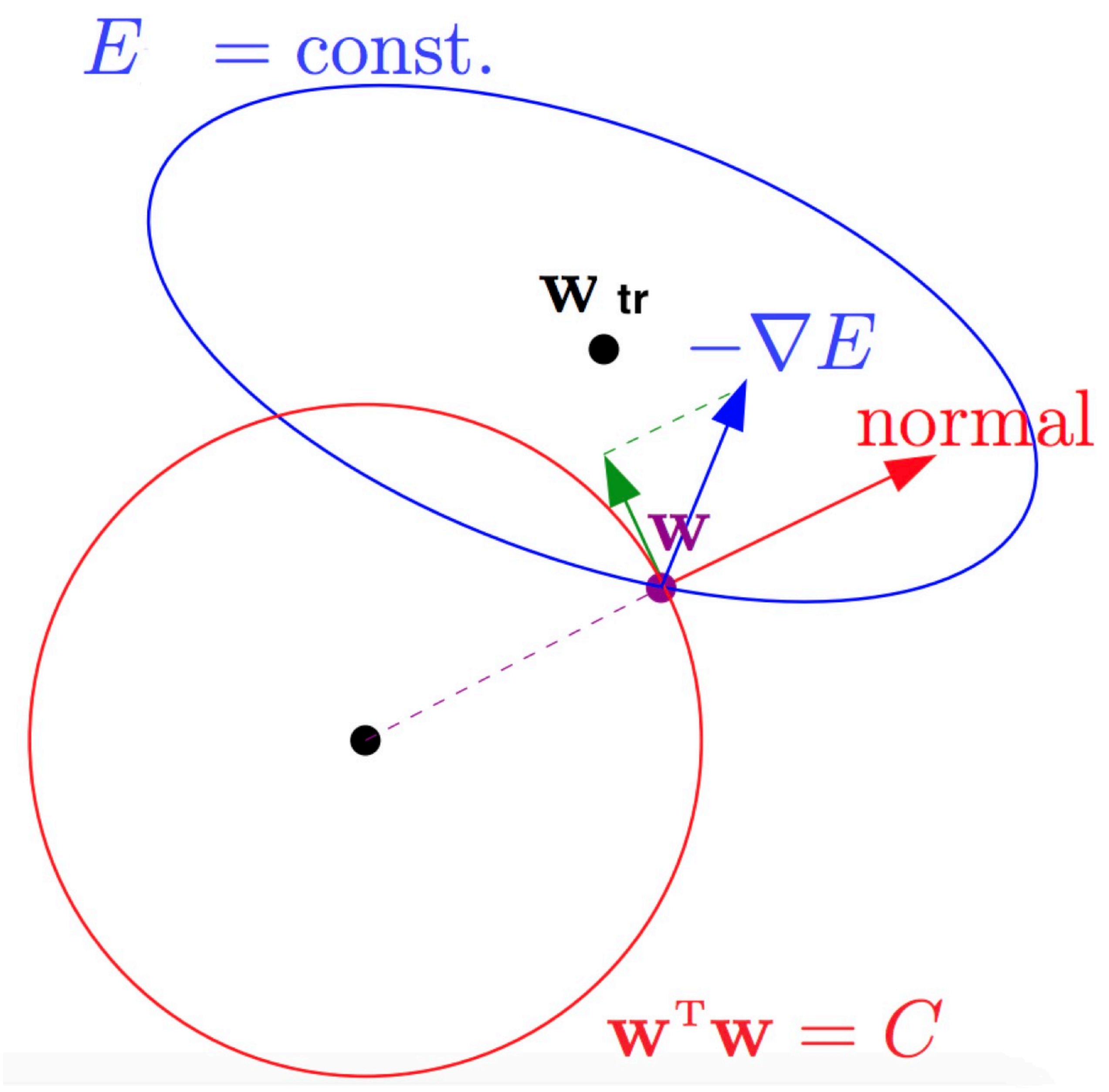
Constraining the weights

- Hard constraint: $\mathscr{H}_2$ is constrained version of $\mathscr{H}_{10}$ (with $w_q = 0$ for $q > 2$)
- Soft-order constraint: $\sum_{q=0}^{Q} w_q^2 \leq C$
- The problem given soft-order constraint:
  - Minimize $\frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $\underbrace{w^T w \leq C}_{\text{smaller hypothesis space}}$
- Solution w_{reg} instead of w_{tr}

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$
    - s.t. $w^T w \leq C$



- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto - w_{\text{reg}}$
  - Why? If $-\nabla E_{\text{tr}}(w_{\text{reg}})$ and w are not parallel, can decrease $E_{\text{tr}}(w)$ without violating the constraint

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \leq C$
- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N}w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N}w_{\text{reg}} = 0$

Regularization

Equivalent to the unconstrained version

- Constrained version:
    - $\min_w E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \leq C$
- Optimal when
    - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N}w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N}w_{\text{reg}} = 0$
- w_{reg} is also the solution of unconstrained problem
    - $\min_w E_{\text{tr}}(w) + \frac{\lambda}{N}w^T w$ (Ridge regression!)

Regularization

Equivalent to the unconstrained version

- Constrained version:
  - $\min_{w} E_{\text{tr}}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$ s.t. $w^T w \leq C$
- Optimal when
  - $\nabla E_{\text{tr}}(w_{\text{reg}}) \propto -w_{\text{reg}}$
- Assume $\nabla E_{\text{tr}}(w_{\text{reg}}) = -2\frac{\lambda}{N} w_{\text{reg}} \Rightarrow \nabla E_{\text{tr}}(w_{\text{reg}}) + 2\frac{\lambda}{N} w_{\text{reg}} = 0$
- w_{reg} is also the solution of unconstrained problem
  - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$ (Ridge regression!) $C \uparrow \quad \lambda \downarrow$

Regularization

Ridge regression solution

- $\min_w E_{\text{reg}}(w) = \frac{1}{N}\left((Zw - y)^T(Zw - y) + \lambda w^T w \right)$
- $\nabla E_{\text{reg}}(w) = 0 \Rightarrow Z^T Z(w - y) + \lambda w = 0$

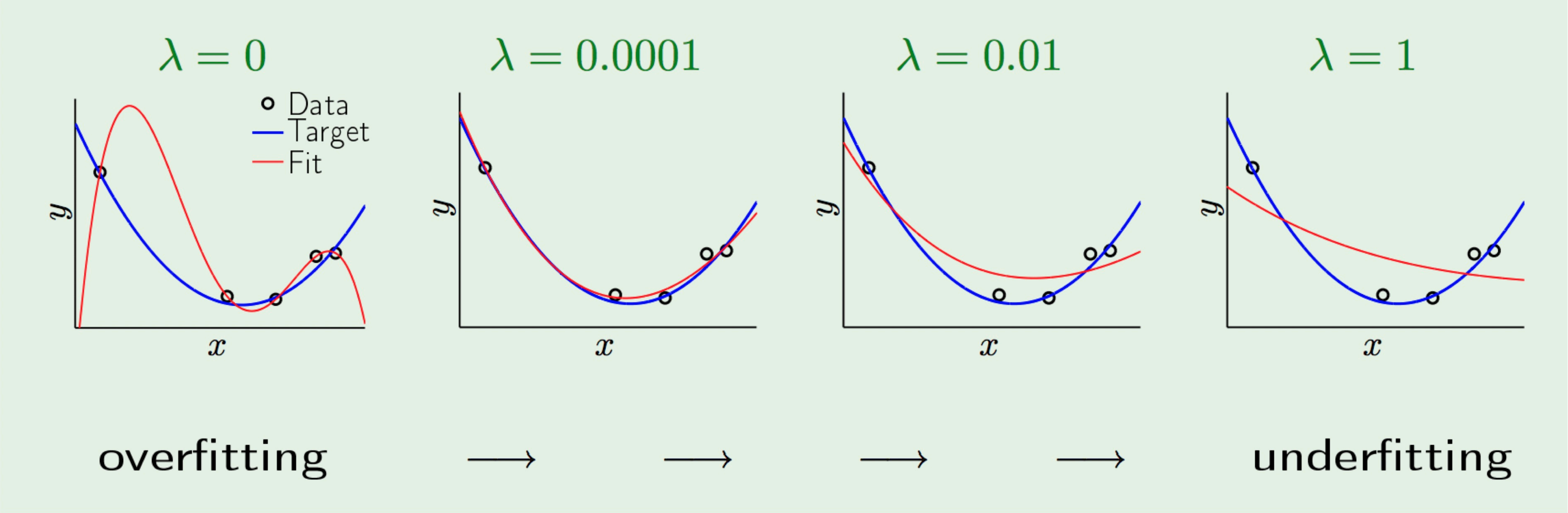
Regularization

Ridge regression solution

- $\min_{w} E_{\text{reg}}(w) = \frac{1}{N}\left((Zw - y)^T(Zw - y) + \lambda w^T w\right)$
- $\nabla E_{\text{reg}}(w) = 0 \Rightarrow Z^T Z(w - y) + \lambda w = 0$
- So, $w_{\text{reg}} = (Z^T Z + \lambda I)^{-1} Z^T y$ (with regularization) as opposed to $w_{\text{tr}} = (Z^T Z)^{-1} Z^T y$ (without regularization)

Regularization
The result

- $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$



Regularization

Equivalent to “weight decay”

- Consider the general case
  - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$

Regularization

Equivalent to "weight decay"

- Consider the general case
  - $\min_{w} E_{\text{tr}}(w) + \frac{\lambda}{N} w^T w$
- Gradient descent:

$$w_{t+1} = w_t - \eta(\nabla E_{\text{tr}}(w_t) + 2\frac{\lambda}{N} w_t)$$

- $$= w_t \underbrace{(1 - 2\eta\frac{\lambda}{N})}_{\text{weight decay}} - \eta \nabla E_{\text{tr}}(w_t)$$

Regularization

Variations of weight decay

- Calling the regularizer $\Omega = \Omega(h)$, we minimize
  - $E_{\text{reg}}(h) = E_{\text{tr}}(h) + \frac{\lambda}{N}\Omega(h)$
- In general, $\Omega(h)$ can be any measurement for the "size" of h

Regularization

L2 vs L1 regularizer

- L1-regularizer: $\Omega(w) = \|w\|_1 = \sum_q |w_q|$
- Usually leads to a sparse solution (only few w_q will be nonzero)