

# Empirical risk minimization

CPT\_S 434/534 Neural network design and application

# Empirical risk minimization

$$\begin{aligned}
 & h_S^{\text{ERM}} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \hat{R}_S(h) \quad \longrightarrow \text{Empirical risk} \\
 & \text{Plug in} \\
 & R(h) - R^* = \underbrace{\left( R(h) - \inf_{h \in \mathcal{H}} R(h) \right)}_{\text{estimation}} + \underbrace{\left( \inf_{h \in \mathcal{H}} R(h) - R^* \right)}_{\text{approximation}} \quad \longrightarrow \text{Controlled by constraints/regularization}
 \end{aligned}$$

Q: How to control?

**Proposition 4.1** *For any sample  $S$ , the following inequality holds for the hypothesis returned by ERM:*

$$\mathbb{P} \left[ R(h_S^{\text{ERM}}) - \inf_{h \in \mathcal{H}} R(h) > \epsilon \right] \leq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |R(h) - \hat{R}_S(h)| > \frac{\epsilon}{2} \right]. \quad (4.3)$$

**Corollary 3.19 (VC-dimension generalization bounds)** *Let  $\mathcal{H}$  be a family of functions taking values in  $\{-1, +1\}$  with VC-dimension  $d$ . Then, for any  $\delta > 0$ , with probability at least  $1 - \delta$ , the following holds for all  $h \in \mathcal{H}$ :*

$$R(h) \leq \hat{R}_S(h) + \sqrt{\frac{2d \log \frac{em}{d}}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} = O(\sqrt{1/m}) \quad (3.29)$$

# Empirical risk minimization

**Definition 2.2 (Empirical error)** *Given a hypothesis  $h \in \mathcal{H}$ , a target concept  $c \in \mathcal{C}$ , and a sample  $S = (x_1, \dots, x_m)$ , the empirical error or empirical risk of  $h$  is defined by*

$$\min_h \hat{R}_S(h) = \frac{1}{m} \sum_{i=1}^m 1_{h(x_i) \neq c(x_i)}. \quad (2.2)$$

Too hard: need a surrogate

# Maximum likelihood principle

Bernoulli distribution:

If  $X$  is a random variable with this distribution, then:

$$\Pr(X = 1) = p = 1 - \Pr(X = 0) = 1 - q.$$

The **probability mass function**  $f$  of this distribution, over possible outcomes  $k$ , is

$$f(k; p) = \begin{cases} p & \text{if } k = 1, [2] \\ q = 1 - p & \text{if } k = 0. \end{cases}$$

This can also be expressed as

$$f(k; p) = p^k (1 - p)^{1-k} \quad \text{for } k \in \{0, 1\}$$

$(x_i, y_i) \rightarrow p(x_i)$  is the **underlying** probability of  $x_i$  belonging to class 1 ( $y_i = 1$ )

Observe  $n$  samples (simultaneously)

$$\prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i} \quad \text{Likelihood function}$$

# Maximum likelihood estimation (MLE)

- Likelihood function

$$L(w) = P_w(X_1 = x_1, \dots, X_n = x_n) = f(w; x_1) \times \dots \times f(w; x_n) = \prod_{i=1}^n f(w; x_i)$$

approximate

$$\prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$

# Maximum likelihood estimation (MLE)

- Maximizing likelihood function

$$\max_w L(w) = \prod_{i=1}^n f(w; x_i)$$

Difficult to optimize



# Maximum likelihood estimation (MLE)

- Maximizing likelihood function

$$\max_w L(w) = \prod_{i=1}^n f(w; x_i)$$

Difficult to optimize

- Maximizing **log**-likelihood function

$$\max_w \log L(w) = \log \prod_{i=1}^n f(w; x_i) = \sum_{i=1}^n \log(f(w; x_i))$$

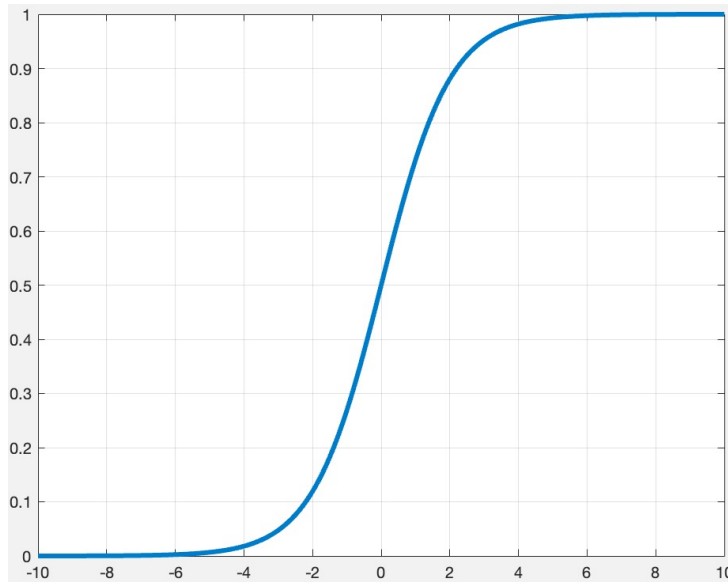
Q: how to approximate the probability?

Easy to optimize

# Logistic function

- Logistic regression

$$\min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) = - \max_w \frac{1}{n} \sum_{i=1}^n \log\left(\frac{1}{1 + e^{-(2y_i-1)f(w;x_i)}}\right)$$



Probability=1

Probability=0

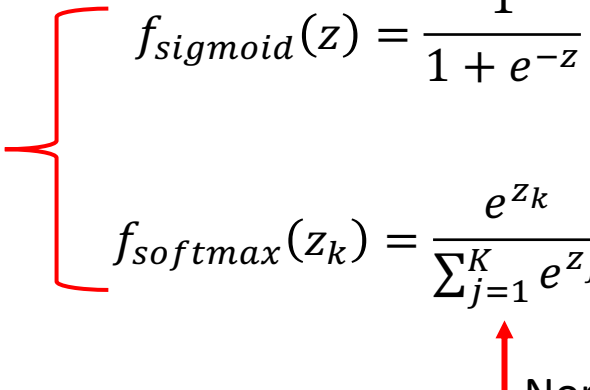
$$f_{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$



# Softmax function

- Softmax function: generalization of logistic function to multiclass

Prediction: probability-like output


$$\begin{aligned} f_{sigmoid}(z) &= \frac{1}{1 + e^{-z}} & z \in \mathbb{R}: \text{prediction to class 1} \\ f_{softmax}(z_k) &= \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}} & z \in \mathbb{R}^K: \text{prediction to K classes} \end{aligned}$$

Normalization: summation of all elements=1

# Softmax function

- Logistic model (binary classification)

$$\min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) = \min_w - \frac{1}{n} \sum_{i=1}^n \log(f_{sigmoid}((2y_i-1)f(w;x_i)))$$

# Softmax function

- Logistic model (binary classification)

$$\min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) = \min_w - \frac{1}{n} \sum_{i=1}^n \log(f_{sigmoid}((2y_i-1)f(w;x_i)))$$

- Softmax classifier (multiclass classification)

$$\min_w - \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{i,k} \log(f_{softmax}(f(w;x_{i,k})))$$

Cross-entropy loss

One-hot
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

8 bits

# Softmax function

- Logistic model (binary classification)

$$\min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) = \min_w - \frac{1}{n} \sum_{i=1}^n \log(f_{sigmoid}((2y_i-1)f(w;x_i)))$$

- Softmax classifier (multiclass classification)

$$\min_w - \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{i,k} \log(f_{softmax}(f(w;x_{i,k}))) \quad \leftarrow \text{Cross-entropy loss}$$

$$\sum_{k=1}^K y_{i,k} \log \left( \frac{e^{f(w;x_{i,k})}}{\sum_{j=1}^K e^{f(w;x_{i,j})}} \right) = \sum_{k=1}^K y_{i,k} f(w;x_{i,k}) - \sum_{k=1}^K y_{i,k} \log \left( \sum_{j=1}^K e^{f(w;x_{i,j})} \right)$$

$$= \sum_{k=1}^K y_{i,k} f(w;x_{i,k}) - \log \left( \sum_{j=1}^K e^{f(w;x_{i,j})} \right)$$

(Usually used in deep learning)

One-hot
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

8 bits

# Why is ERM general?

- Including many objective functions used in machine learning
- MLE: a special case of ERM
  - E.g., logistic regression

$$\min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)})$$

# Why is ERM general?

- Including many objective functions used in machine learning
- MLE: a special case of ERM
  - E.g., logistic regression

$$\begin{aligned} & \min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) \\ &= -\max_w -\frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) \end{aligned}$$

# Why is ERM general?

- Including many objective functions used in machine learning
- MLE: a special case of ERM
  - E.g., logistic regression


$$\begin{aligned} & \min_w \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) \\ &= -\max_w -\frac{1}{n} \sum_{i=1}^n \log(1 + e^{-(2y_i-1)f(w;x_i)}) \\ &= -\max_w \frac{1}{n} \sum_{i=1}^n \log\left(\frac{1}{1 + e^{-(2y_i-1)f(w;x_i)}}\right) \end{aligned}$$

# Determining model parameters

- An optimization problem (on training set)

Objective function

n training data


$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^n l(f(w; x_i), y_i) = \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

- Analytical solution?




# Determining model parameters

- An optimization problem (on training set)

Objective function

n training data


$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^n l(f(w; x_i), y_i) = \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

- Analytical solution?

$$\min_w \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

$$X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times n}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$$


$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0$$

# Determining model parameters

- An optimization problem (on training set)

Objective function

n training data


$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^n l(f(w; x_i), y_i) = \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

- Analytical solution?

$$\min_w \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

$$X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times n}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$$

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX' w^* - XY = 0 \Rightarrow w^* = (XX')^{-1} XY$$

# Determining model parameters

- An optimization problem (on training set)

Objective function

n training data

$$\min_w f(w) = \frac{1}{n} \sum_{i=1}^n l(f(w; x_i), y_i) = \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

- Analytical solution?

$$\min_w \frac{1}{2n} \sum_{i=1}^n (x_i' w - y_i)^2$$

$$X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times n}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$$

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow$$

$$XX'w^* - XY = 0$$

$$w^* = (XX')^{-1}XY$$

Q: is this closed form solution a good way in practice? Why?

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Inverse of a scalar?

$$x x^{-1} = 1 \rightarrow x^{-1} = 1/x$$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Inverse of a scalar?

$$x x^{-1} = 1 \rightarrow x^{-1} = 1/x$$

- Inverse of a matrix?

$$XX^{-1} = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

Matrix multiplication	One $n \times m$ matrix & one $m \times p$ matrix	One $n \times p$ matrix	Schoolbook matrix multiplication	$O(nmp)$
Matrix inversion*	One $n \times n$ matrix	One $n \times n$ matrix	Gauss–Jordan elimination	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms	$O(n^{2.373})$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

Matrix multiplication	One $n \times m$ matrix & one $m \times p$ matrix	One $n \times p$ matrix	Schoolbook matrix multiplication	$O(nmp)$
-----------------------	--	-------------------------	----------------------------------	----------

Not every matrix has inversion

Matrix inversion\*

One $n \times n$ matrix	One $n \times n$ matrix	Gauss–Jordan elimination	$O(n^3)$
		Strassen algorithm	$O(n^{2.807})$
		Coppersmith–Winograd algorithm	$O(n^{2.376})$
		Optimized CW-like algorithms	$O(n^{2.373})$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Matrix multiplication:



# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Matrix multiplication:

$$XX': d \times n \times d \quad XY: d \times n \quad (XX')^{-1}XY: d \times d \times n \rightarrow O(d^2n)$$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Matrix multiplication:

$$XX': d \times n \times d \quad XY: d \times n \quad (XX')^{-1}XY: d \times d \times n \rightarrow O(d^2n)$$

- Inverse of a matrix:

$$(XX')^{-1}: O(d^{2.373})$$

# Determining model parameters

- Computational complexity for the analytical solution?

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow 0 \Rightarrow XX'w^* - XY = 0 \Rightarrow w^* = (XX')^{-1}XY$$

- Matrix multiplication:

$$XX': d \times n \times d \quad XY: d \times n \quad (XX')^{-1}XY: d \times d \times n \rightarrow O(d^2n)$$

- Inverse of a matrix:

$$(XX')^{-1}: O(d^{2.373})$$

- Total complexity

$$O(d^2n + d^{2.373})$$

# Determining model parameters

- Gradient descent (GD)

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow O(dn)$$

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d) \longrightarrow \text{An iterative algorithm}$$

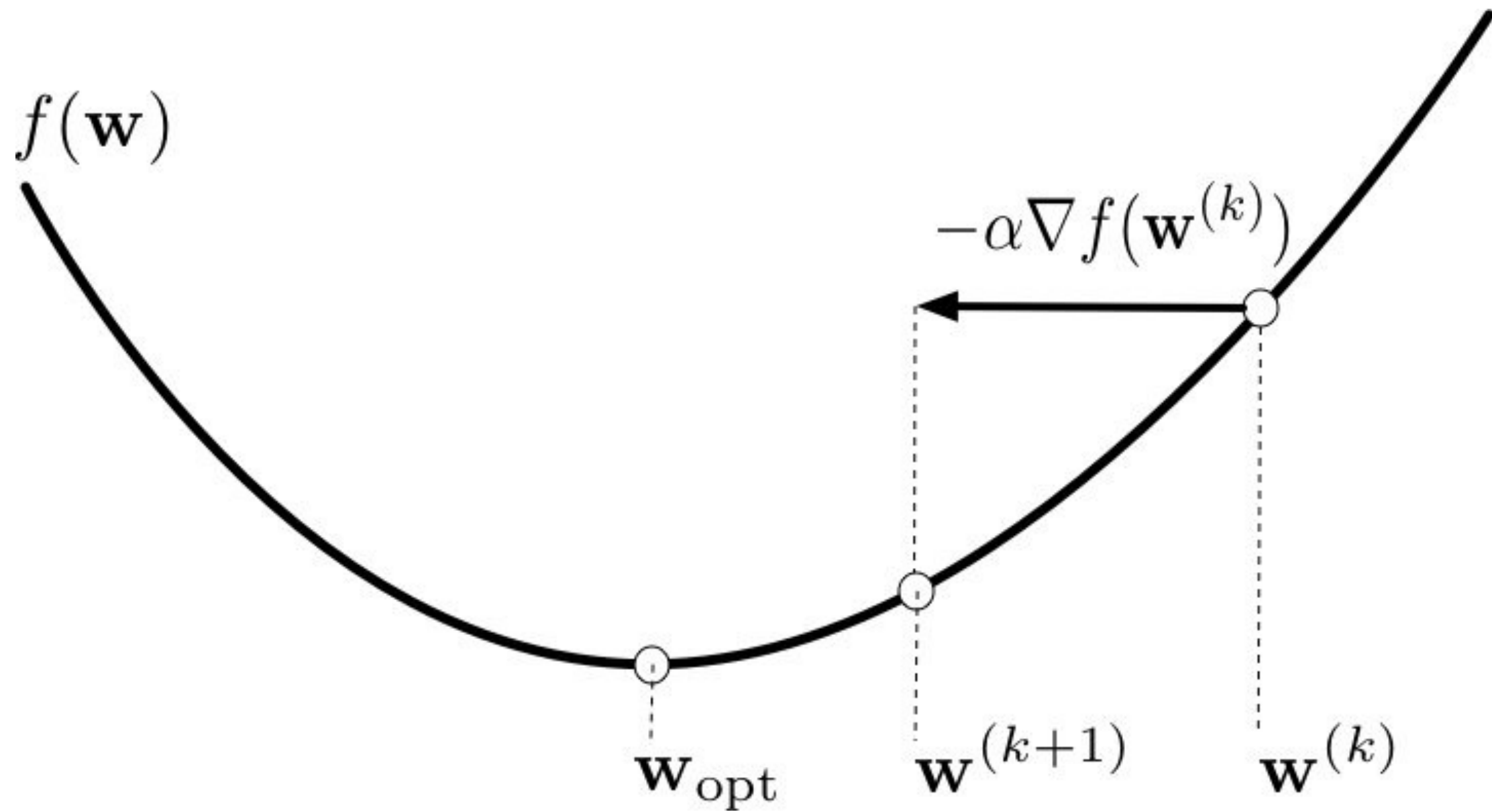
# Determining model parameters

- Gradient descent (GD)

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow O(dn)$$

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d) \longrightarrow \text{An iterative algorithm}$$

Step size (learning rate):  
usually pre-defined



# Determining model parameters

- Gradient descent (GD)

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow O(dn)$$

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d) \longrightarrow \text{An iterative algorithm}$$

- Suppose run GD for T iterations
- Total complexity

$$O(dnT)$$

# Determining model parameters

- Gradient descent (GD)

$$\nabla_w f(w) = \frac{1}{n} \sum_{i=1}^n x_i' w x_i - y_i x_i \rightarrow O(dn)$$

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d) \longrightarrow \text{An iterative algorithm}$$

- Suppose run GD for T iterations
- Total complexity

$O(dnT)$  vs.  $O(d^2n + d^{2.373})$  for the closed form solution



# Determining model parameters

- When to terminate GD (determining T)?
  - Convergence rate for GD?

**Theorem 2.1.14** *If  $f \in \mathcal{S}_{\mu,L}^{1,1}(R^n)$  and  $0 < h \leq \frac{2}{\mu+L}$  then the gradient method generates a sequence  $\{x_k\}$  such that*

$$\|x_k - x^*\|^2 \leq \left(1 - \frac{2h\mu L}{\mu + L}\right)^k \|x_0 - x^*\|^2.$$

*If  $h = \frac{2}{\mu+L}$  then*

$$\|x_k - x^*\| \leq \left(\frac{Q_f - 1}{Q_f + 1}\right)^k \|x_0 - x^*\|,$$

$$f(x_k) - f^* \leq \frac{L}{2} \left(\frac{Q_f - 1}{Q_f + 1}\right)^{2k} \|x_0 - x^*\|^2,$$

*where  $Q_f = L/\mu$ .*

# Determining model parameters

- When to terminate GD (determining T)?
  - Convergence rate for GD?

**Theorem 2.1.14** *If  $f \in \mathcal{S}_{\mu,L}^{1,1}(R^n)$  and  $0 < h \leq \frac{2}{\mu+L}$  then the gradient method generates a sequence  $\{x_k\}$  such that*

$$\|x_k - x^*\|^2 \leq \left(1 - \frac{2h\mu L}{\mu + L}\right)^k \|x_0 - x^*\|^2.$$

*If  $h = \frac{2}{\mu+L}$  then*

$$\|x_k - x^*\| \leq \left(\frac{Q_f - 1}{Q_f + 1}\right)^k \|x_0 - x^*\|,$$

$$f(x_k) - f^* \leq \frac{L}{2} \left(\frac{Q_f - 1}{Q_f + 1}\right)^{2k} \|x_0 - x^*\|^2,$$

Approximated solution (not exact)

$$= \epsilon(k) = O(a^k)$$

*where  $Q_f = L/\mu$ .*

# Determining model parameters

- When to terminate GD (determining T)?
  - Convergence rate for GD?

**Theorem 2.1.14** *If  $f \in \mathcal{S}_{\mu,L}^{1,1}(\mathbb{R}^n)$  and  $0 < h \leq \frac{2}{\mu+L}$  then the gradient method generates a sequence  $\{x_k\}$  such that*

$$\|x_k - x^*\|^2 \leq \left(1 - \frac{2h\mu L}{\mu + L}\right)^k \|x_0 - x^*\|^2.$$

*If  $h = \frac{2}{\mu+L}$  then*

$$\|x_k - x^*\| \leq \left(\frac{Q_f - 1}{Q_f + 1}\right)^k \|x_0 - x^*\|,$$

$$f(x_k) - f^* \leq \frac{L}{2} \left(\frac{Q_f - 1}{Q_f + 1}\right)^{2k} \|x_0 - x^*\|^2,$$

*where  $Q_f = L/\mu$ .*

Approximated solution (not exact)

$$= \epsilon(k) = O(a^k)$$

$$0 < a < 1$$

$$k = O(\log_{1/a}(1/\epsilon))$$

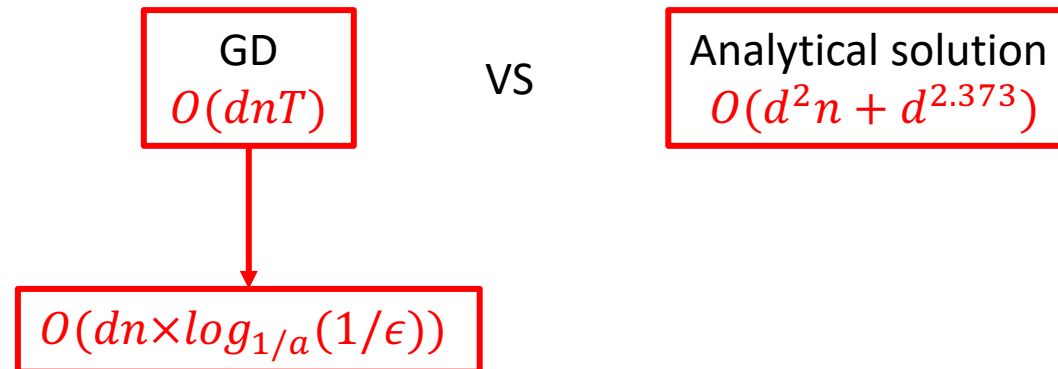
# Determining model parameters

- Now we can answer the question:  
Can Gradient Descent (GD) do better?

GD $O(dnT)$	vs	Analytical solution $O(d^2n + d^{2.373})$
----------------	----	--

# Determining model parameters

- Now we can answer the question:  
Can Gradient Descent (GD) do better?

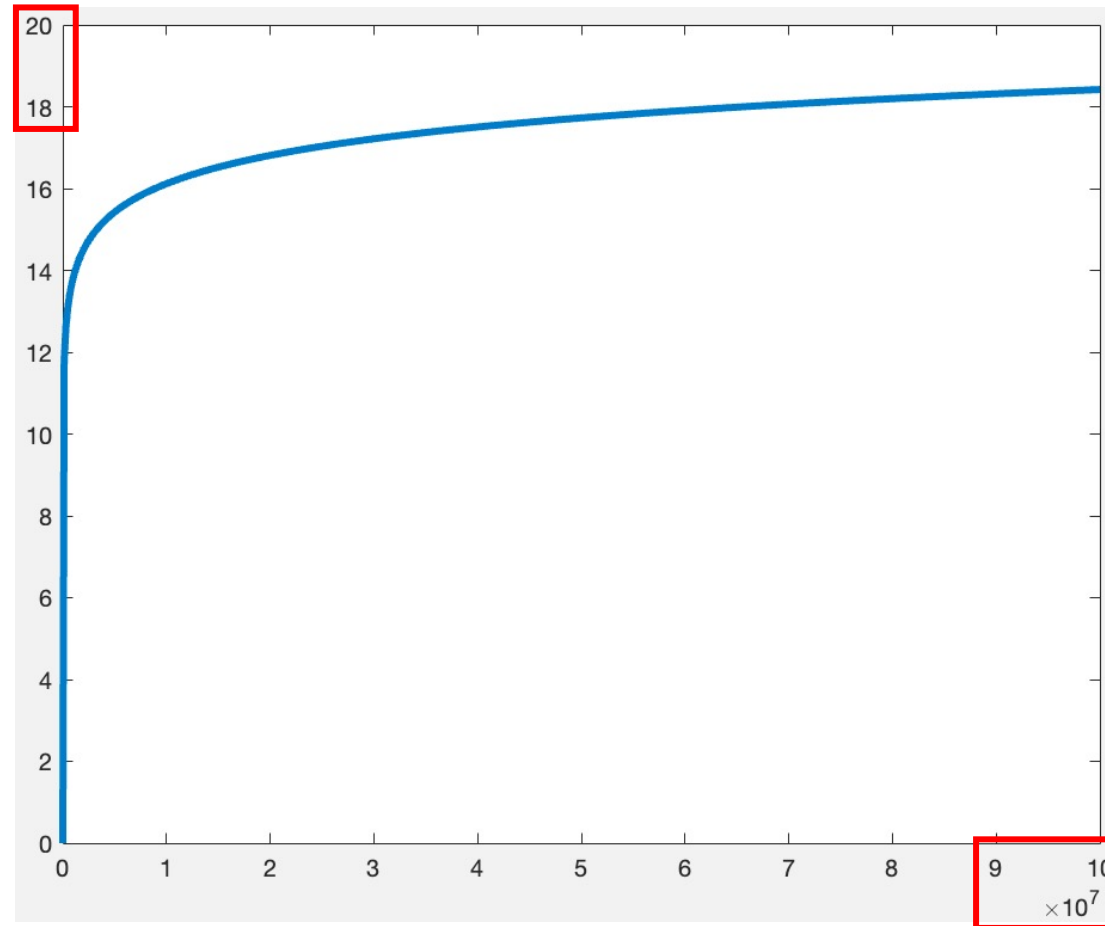


# Determining model parameters

- Is log term  $\log_{1/a}(1/\epsilon)$  large?

# Determining model parameters

- Is log term  $\log_{1/a}(1/\epsilon)$  large?



# Determining model parameters

- Is  $d$  large?

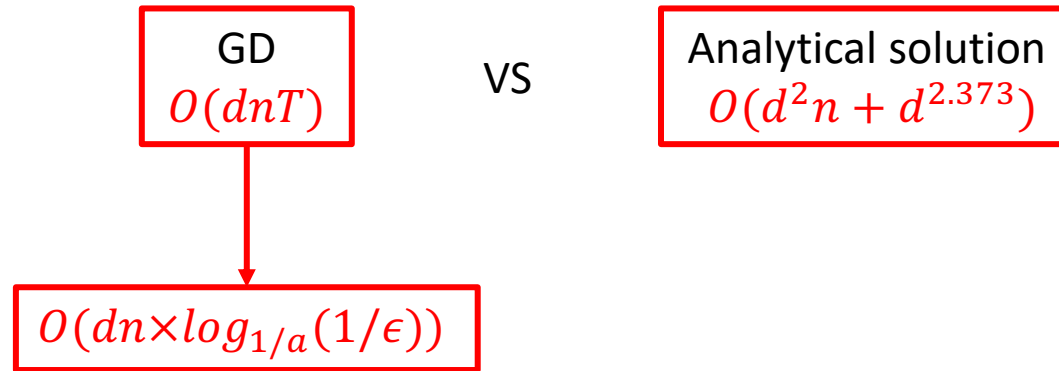


name	source	type	class	training size	testing size	feature
<a href="#">a1a</a>	<a href="#">UCI</a>	classification	2	1,605	30,956	123
<a href="#">a2a</a>	<a href="#">UCI</a>	classification	2	2,265	30,296	123
<a href="#">a3a</a>	<a href="#">UCI</a>	classification	2	3,185	29,376	123
<a href="#">a4a</a>	<a href="#">UCI</a>	classification	2	4,781	27,780	123
<a href="#">a5a</a>	<a href="#">UCI</a>	classification	2	6,414	26,147	123
<a href="#">a6a</a>	<a href="#">UCI</a>	classification	2	11,220	21,341	123
<a href="#">a7a</a>	<a href="#">UCI</a>	classification	2	16,100	16,461	123
<a href="#">a8a</a>	<a href="#">UCI</a>	classification	2	22,696	9,865	123
<a href="#">a9a</a>	<a href="#">UCI</a>	classification	2	32,561	16,281	123
<a href="#">australian</a>	<a href="#">Statlog</a>	classification	2	690		14
<a href="#">avazu</a>	<a href="#">Avazu's Click-through Prediction</a>	classification	2	40,428,967	4,577,464	1,000,000
<a href="#">breast-cancer</a>	<a href="#">UCI</a>	classification	2	683		10
<a href="#">cod-rna</a>	<a href="#">[AVU06a]</a>	classification	2	59,535		8
<a href="#">colon-cancer</a>	<a href="#">[AU99a]</a>	classification	2	62		2,000
<a href="#">covtype.binary</a>	<a href="#">UCI</a>	classification	2	581,012		54
<a href="#">criteo</a>	<a href="#">Criteo's Display Advertising Challenge</a>	classification	2	45,840,617	6,042,135	1,000,000
<a href="#">criteo_tb</a>	<a href="#">Criteo's Terabyte Click Logs</a>	classification	2	4,195,197,692	178,274,637	1,000,000
<a href="#">diabetes</a>	<a href="#">UCI</a>	classification	2	768		8
<a href="#">duke breast-cancer</a>	<a href="#">[MW01a]</a>	classification	2	44		7,129
<a href="#">epsilon</a>	<a href="#">PASCAL Challenge 2008</a>	classification	2	400,000	100,000	2,000
<a href="#">fourclass</a>	<a href="#">[TKH96a]</a>	classification	2	862		2
<a href="#">german.numer</a>	<a href="#">Statlog</a>	classification	2	1,000		24
<a href="#">gisette</a>	<a href="#">NIPS 2003 Feature Selection Challenge [IG05a]</a>	classification	2	6,000	1,000	5,000
<a href="#">heart</a>	<a href="#">Statlog</a>	classification	2	270		13
<a href="#">HIGGS</a>	<a href="#">UCI</a>	classification	2	11,000,000		28
<a href="#">ijcnn1</a>	<a href="#">[DP01a]</a>	classification	2	49,990	91,701	22
<a href="#">ionosphere</a>	<a href="#">UCI</a>	classification	2	351		34
<a href="#">kdd2010 (algebra)</a>	<a href="#">KDD CUP 2010</a>	classification	2	8,407,752	510,302	20,216,830
<a href="#">kdd2010 (bridge to algebra)</a>	<a href="#">KDD CUP 2010</a>	classification	2	19,264,097	748,401	29,890,095
<a href="#">kdd2010 raw version (bridge to algebra)</a>	<a href="#">KDD CUP 2010</a>	classification	2	19,264,097	748,401	1,163,024
<a href="#">kdd2012</a>	<a href="#">KDD CUP 2012</a>	classification	2	149,639,105		54,686,452

# Determining model parameters

- Is  $d$  large?

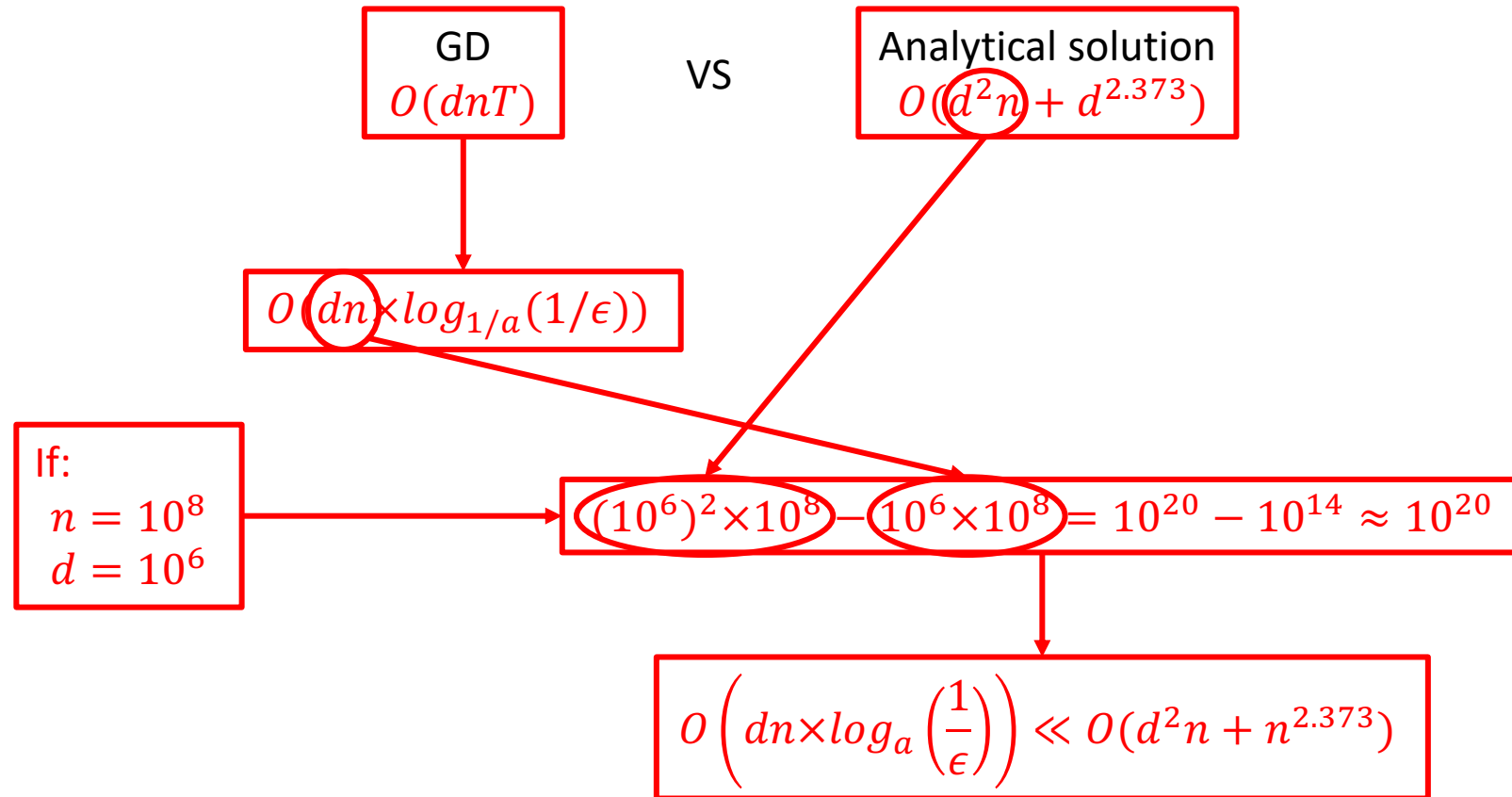
Yes!



# Determining model parameters

- Is  $d$  large?

Yes!



# Determining model parameters

- Stochastic gradient descent (SGD)

Randomly sample  $b$  data  $\longrightarrow$   $\nabla_w f(w) = \frac{1}{b} \sum_{i=1}^b x_i' w x_i - y_i x_i \rightarrow O(d **b**)$   $b \geq 1$

$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d)$   $\longrightarrow$  An iterative algorithm

# Determining model parameters

- Stochastic gradient descent (SGD)

Randomly sample  $b$  data  $\longrightarrow \nabla_w f(w) = \frac{1}{b} \sum_{i=1}^b x_i' w x_i - y_i x_i \rightarrow O(db)$   $b \geq 1$

$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) \rightarrow O(d)$   $\longrightarrow$  An iterative algorithm

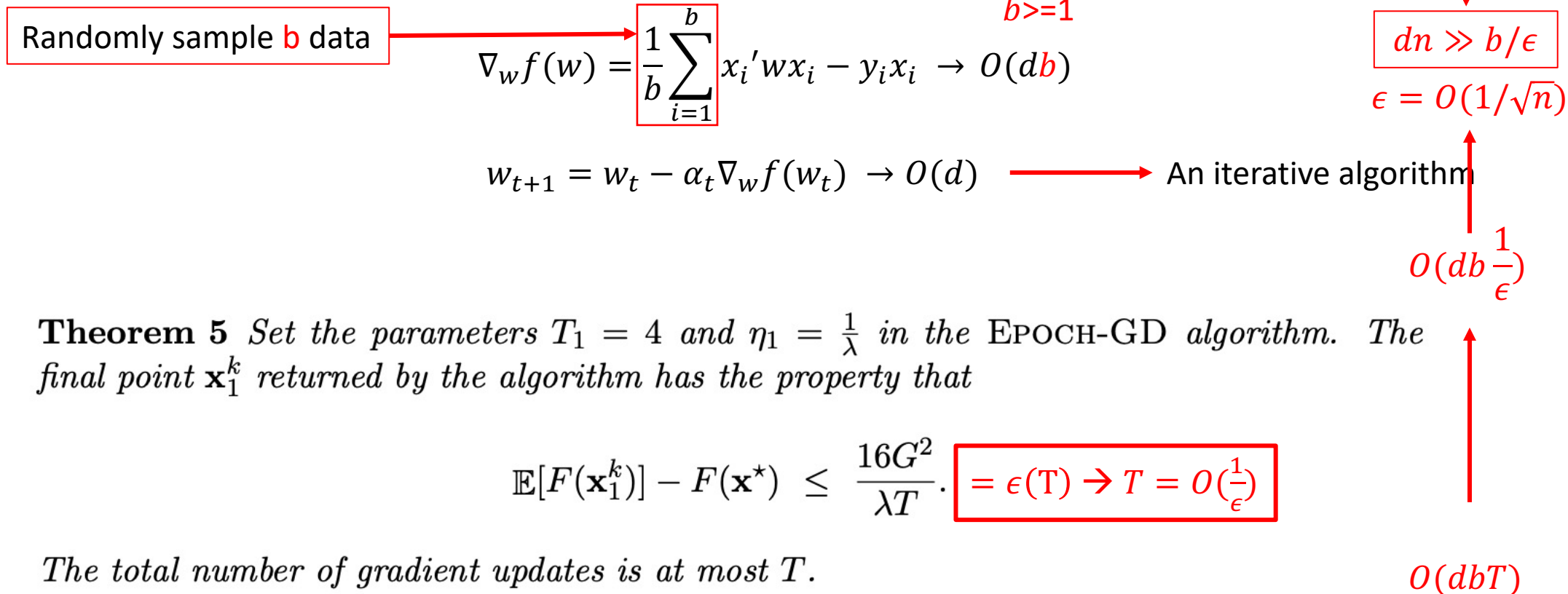
**Theorem 5** Set the parameters  $T_1 = 4$  and  $\eta_1 = \frac{1}{\lambda}$  in the EPOCH-GD algorithm. The final point  $\mathbf{x}_1^k$  returned by the algorithm has the property that

$$\mathbb{E}[F(\mathbf{x}_1^k)] - F(\mathbf{x}^*) \leq \frac{16G^2}{\lambda T}. \quad = \epsilon(T) \rightarrow T = O\left(\frac{1}{\epsilon}\right)$$

The total number of gradient updates is at most  $T$ .

# Determining model parameters

- Stochastic gradient descent (SGD)



# References

- Jung, Alexander. (2018). Machine Learning: Basic Principles.
- Nesterov, Yurii. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media, 2003.