50.007 Machine Learning

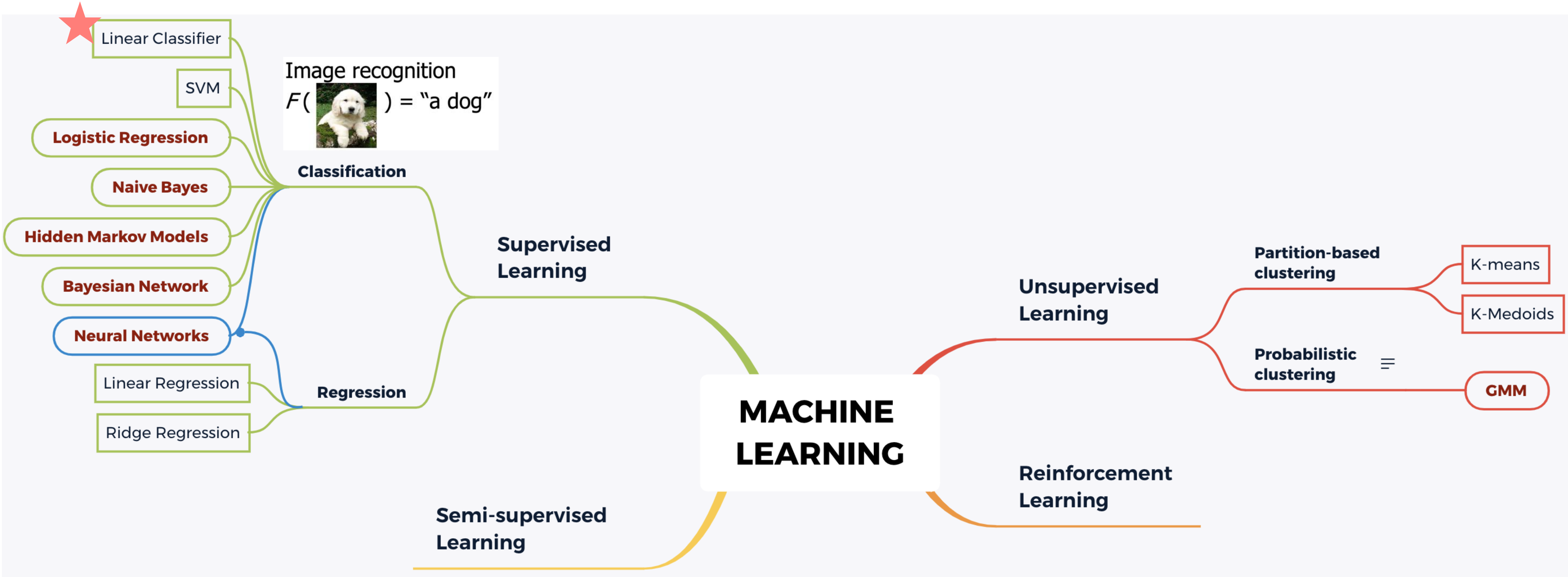
2026 Spring

3. Hinge Loss

Na Zhao
Assistant Professor, ISTD

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Recap



Linear Classifier
SVM
Logistic Regression
Naive Bayes
Hidden Markov Models
Bayesian Network
Neural Networks
Linear Regression
Ridge Regression
Image recognition
F( ) = "a dog"
Classification
Regression
Supervised Learning
Semi-supervised Learning
MACHINE LEARNING
Unsupervised Learning
Reinforcement Learning
Partition-based clustering
Probabilistic clustering
K-means
K-Medoids
GMM

Recap - Linear Classifier (separable case)

1. Training Set (Linearly Separable)

$$\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$$

2. Model (Linear Classifier)

$$h(x;\theta) = \text{sign}(\theta_1 x_1 + \cdots + \theta_d x_d)$$

3. Training Error (Zero-one Loss)

$$\boldsymbol{\varepsilon}_n(\theta) = \frac{1}{n}\sum_{(x,y)\in\mathcal{S}_n} [\![\, y(\theta^\top x) \leq 0 \,]\!]$$

4. Algorithm (The Perceptron Algorithm)

$$\text{if } y^{(t)} \neq h(x^{(t)}; \theta^{(k)}) \text{ then}$$

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$$

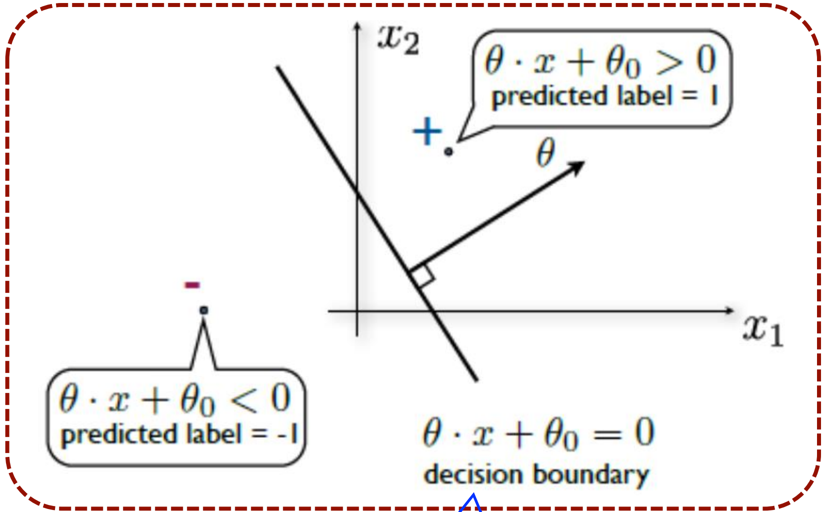
Recap - Linear Classifier (separable case)

- **Zero-one loss**:

$$\mathcal{E}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n}[\![y^{(t)} \neq h(x^{(t)};\theta)]\!] = \frac{1}{n}\sum_{t=1}^{n}[\![y^{(t)}(\theta \cdot x^{(t)}) \leq 0]\!]$$

- **Realizable case**: *zero* training error
- **Linear classifier with offset**:

$$h(x;\theta,\theta_0) = \text{sign}(\theta \cdot x + \theta_0) = \begin{cases} +1, & \theta \cdot x + \theta_0 \geq 0 \\ -1, & \theta \cdot x + \theta_0 < 0 \end{cases}$$



θ is orthogonal to the decision boundary.
θ points in direction of region labelled +1.

- **Perceptron Update Rule**:

$$\text{if } y^{(t)} \neq h(x^{(t)};\theta^k,\theta_0^k) \text{ then}$$
$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$$
$$\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)}$$

+ Perceptron convergence theorem

If training examples are linearly separable, then the perceptron algorithm converges after a finite number of mistakes.

Learning Objectives

You should be able to know:

1. What is a convex function and how to check if a function is convex or not?
2. What is hinge loss?
3. How is hinge loss different from zero-one loss?
4. What is stochastic gradient descent?
5. How to implement the stochastic (sub-)gradient descent?

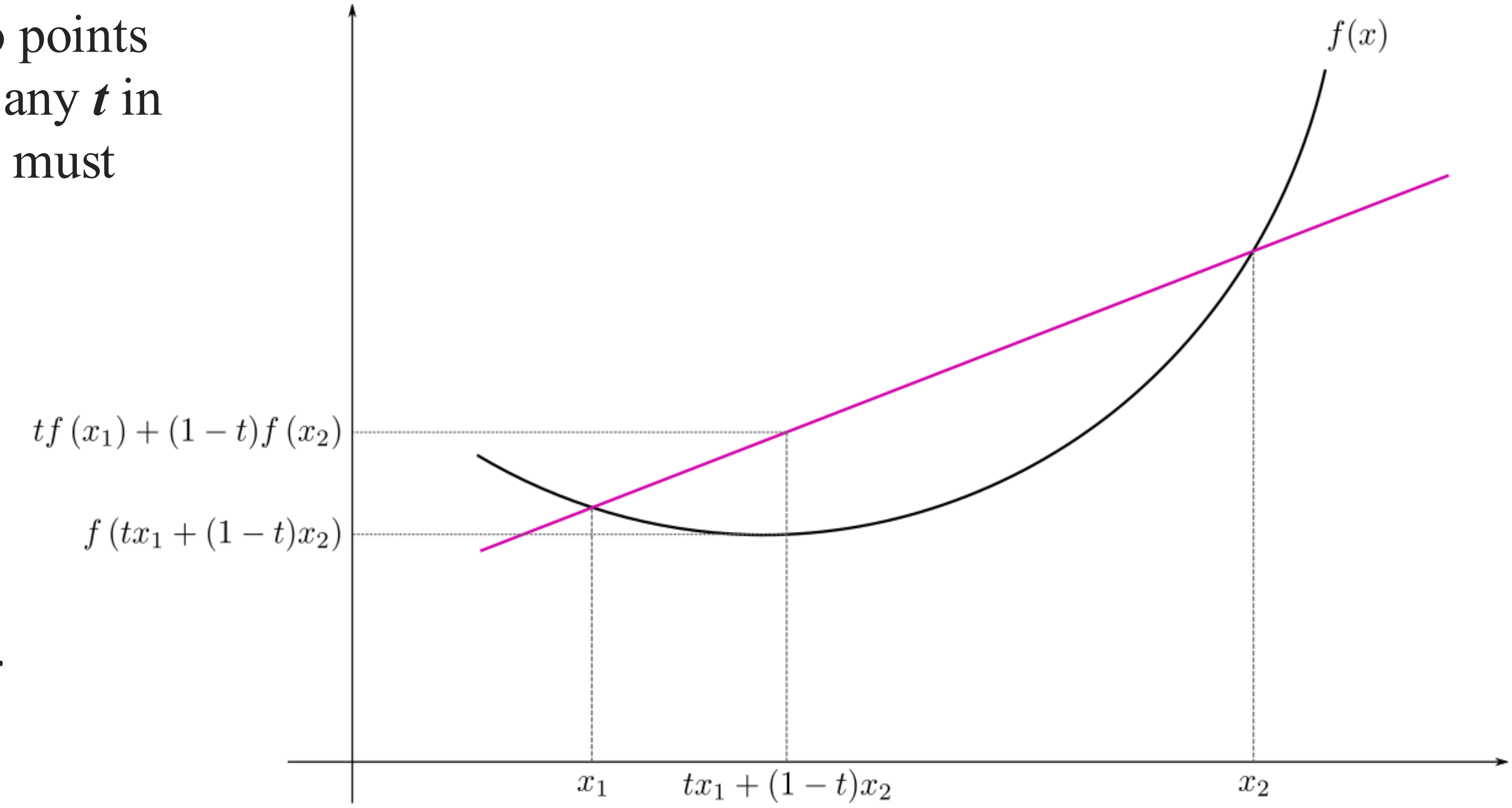
global min pt - lowest possible point anywhere
local min pt- lowest pon nearby

Convex Function

A function ***f(x)*** is **convex** if for any two points x_1 and x_2 in the domain of ***f(x)***, and for any ***t*** in the range **[0,1],** the following condition must holds true:

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

- This means: line segment between any pair of points on the curve of f to be above or just meets the graph.
- Intuitively: the function is **bowl-shaped**.
- Convex function ensures that there is only one global minimum, and no local minimum, make optimization easier and more reliable.

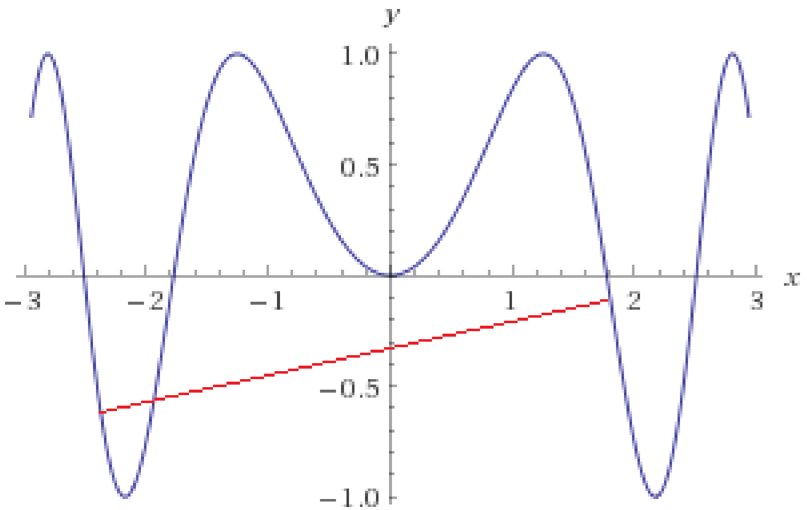


Graph of a convex function : https://en.wikipedia.org/wiki/Convex_function

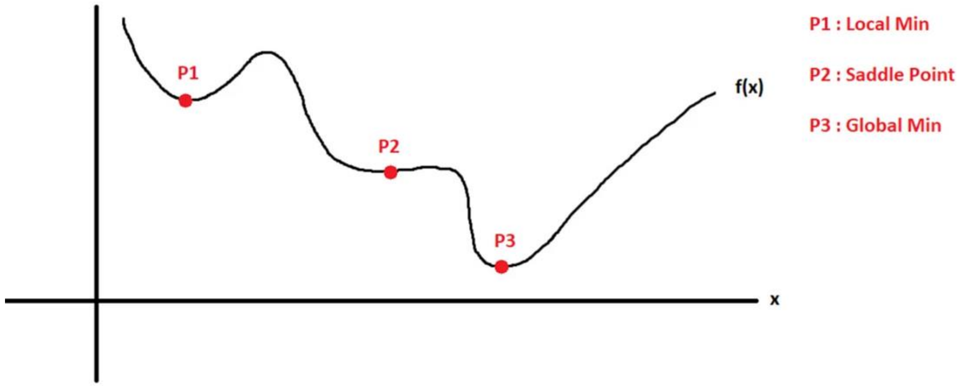
Non-convex Function

- A function is said to be non-convex if it is not convex.
- Geometrically, a non-convex function is one that curves downwards or has multiple peaks and valleys.
- The **challenge** with optimizing non-convex functions: may get stuck in a local minimum (or saddle point) and miss the global minimum, which is the optimal solution we are looking for.

saddle point - gradient = 0, not min or max

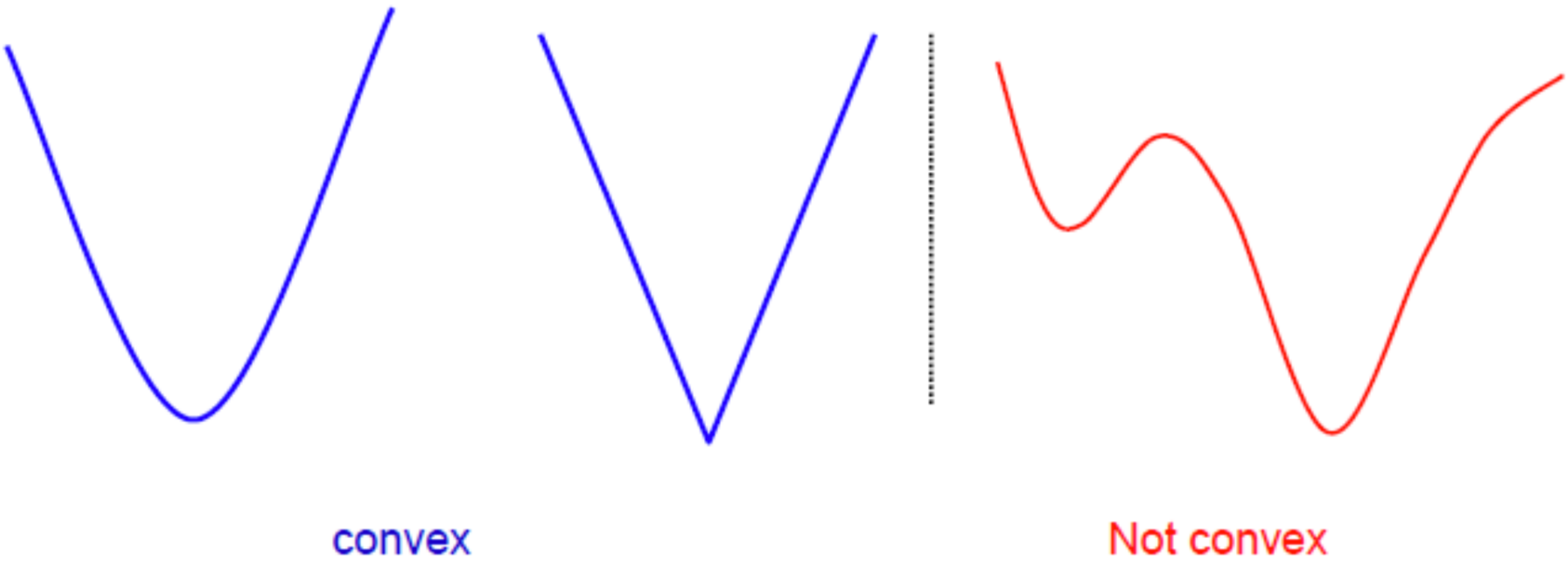


Graph of a non-convex function (source)



Graph of a non-convex function with Local minima, Global minima, and a Saddle point (source)

Convex Function Examples

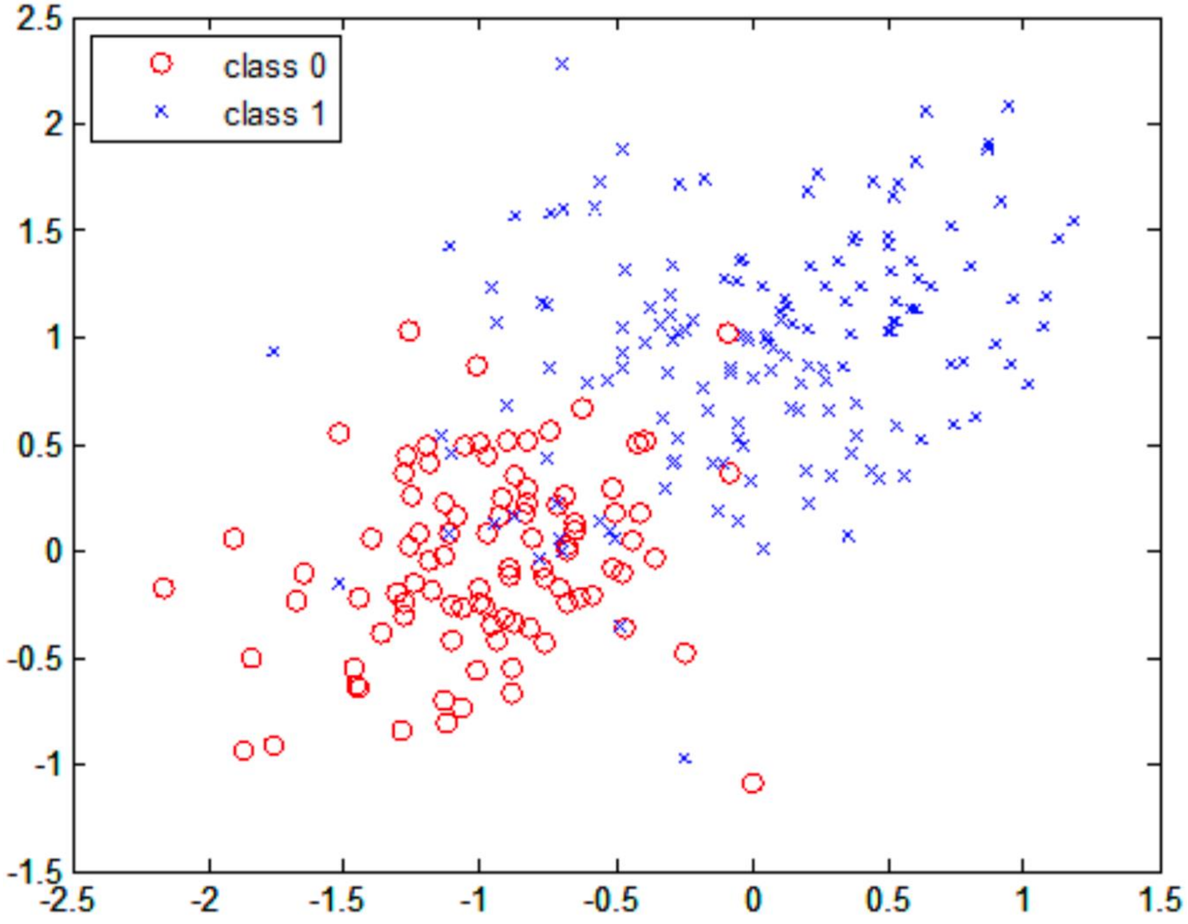


A non-negative sum of convex functions is convex

Linear Classifier – Non-Separable Case

- If training examples are **not linearly separable**, the perceptron algorithm will not converge nor will it find the classifier with the smallest error.

- **Challenge**: The 0-1 risk/loss is a non-convex objective, and hence is hard to optimize (NP-hard) in this **non-realizable case**.



Loss h =

$Loss_h(z) = \{ 1 - y(\theta^T$

Hinge Loss

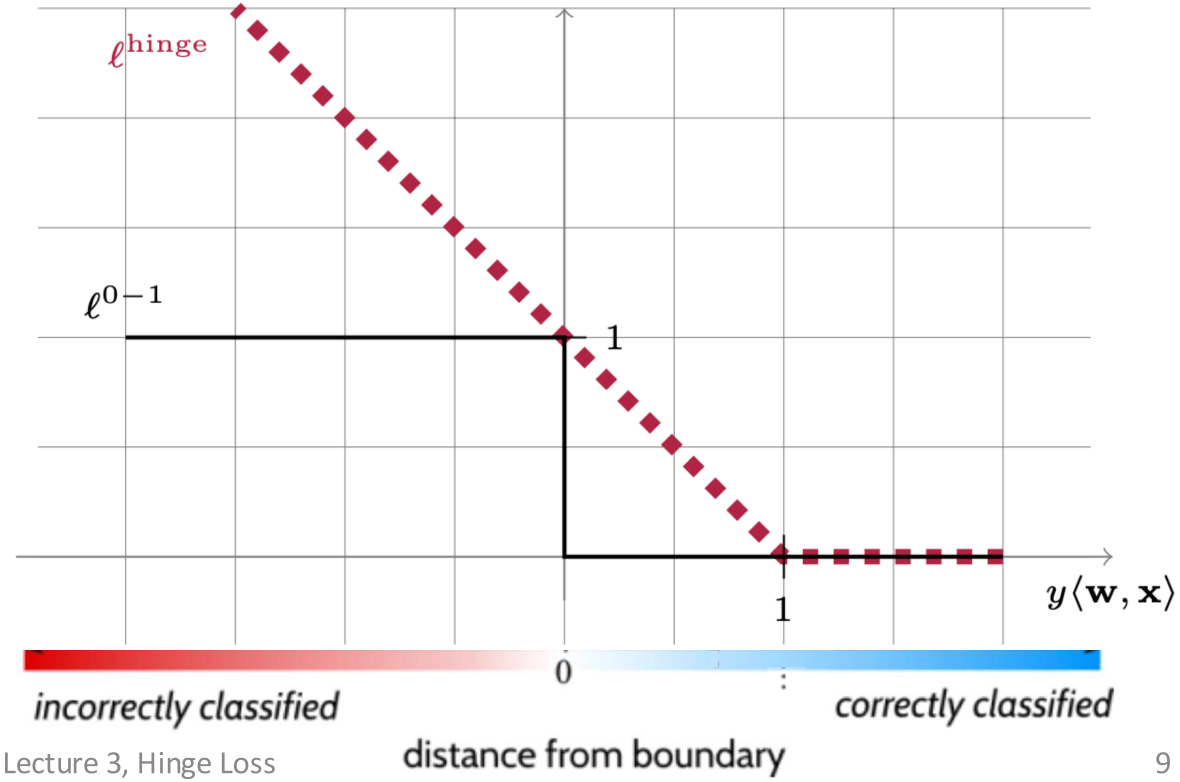
- As convex problems can be learned efficiently, we can **upper bound** the non-convex loss function by a **convex** surrogate loss function.
- **Hinge loss** is a convex surrogate for the zero-one loss!
  - **Zero-one loss**

    $$\mathrm{Loss}_{0|1}(z) = [\![y(\theta^\top x) \leq 0]\!]$$

  - **Hinge loss**

    $$\mathrm{Loss}_{\mathrm{h}}(z) = \max\{1 - y(\theta^\top x), 0\}$$

    CONVEX!
    Penalize larger mistakes more.
    Penalize near-mistakes, i.e. $0 \leq y(\theta^\top x) \leq 1$.



Zero-one Loss *vs.* Hinge Loss - Examples

- **Zero-one loss**

  $\mathrm{Loss}_{0|1}(z) = [\![y(\theta^{\top}x) \le 0]\!]$

- **Hinge loss**

  $\mathrm{Loss}_{\mathrm{h}}(z) = \max\{1 - y(\theta^{\top}x), 0\}$

to check $\theta \cdot x > 0$?

predicted label $=1$

Example 1

$z = y(\theta^{T}x) = -1(0.4) = -0.4 \Rightarrow$ means zero-loss fxn true

- original label = -1 and prediction score = 0.4 (*this means the model predicted class as 1*)
- penalty = max(0, 1+1(0.4)) = 1.4 which is higher than penalty=1 if we use 0-1 loss.

Example 2 true label

$z = y(\theta^{T}x) = 1(-0.9) = -0.9 < 0$, indicator return 1

- original label = 1 and prediction score =(-0.9) (*this means the model predicted class as -1*)
- penalty = max(0, 1-1(-0.9)) = 1.9 which is a very high penalty since the prediction was inaccurate

  ↳ hinge loss

Example 3

- original label = 1 and prediction score = 0.7 (*this means the model predicted class as 1*)
- penalty = max(0, 1-1(0.7)) = 0.3 where loss is low but not 0, since the prediction is correct yet the prediction score is not very high (less than 1)

eg 4: $y = 1$, pred score $= 1.4$

$z = 1(1.4) = 1.4 > 0$

Empirical Risk Minimization

- **Empirical risk** is the training error of all training samples:

$$R_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} \text{Loss}(y^{(t)}(\theta \cdot x^{(t)}))$$

↳ can be any loss fuction (hinge / zero-one)

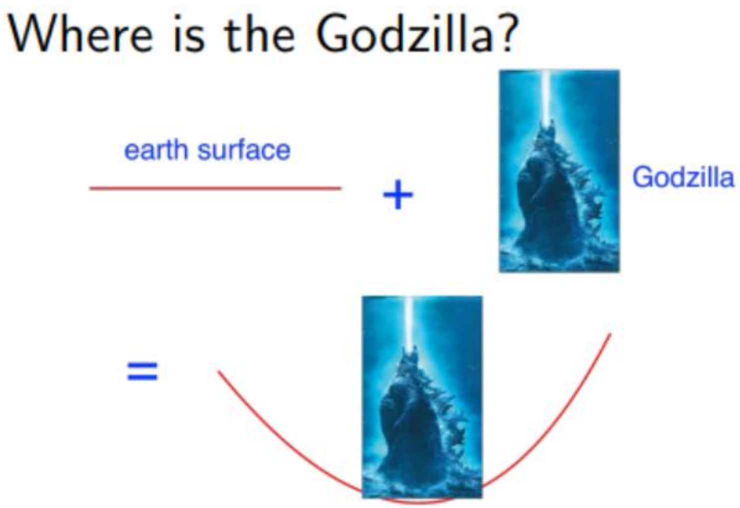
- If we adopt **hinge loss**, it becomes:

$$R_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} \text{Loss}_h(y^{(t)}(\theta \cdot x^{(t)})) = \frac{1}{n}\sum_{t=1}^{n} \max\{1 - y^{(t)}(\theta \cdot x^{(t)}), 0\}$$

- Remember that the classifier training **goal** is to minimize the test or generalization error.
- The **idea** of using hinge loss is to give the classifier a little more feedback in terms of how close its predictions are to the training labels, thus it will generalize better.
- Moreover, now $R_n(\theta)$ is a convex function, and convexity of empirical risk allows us to find the minimum even in non-realizable case (the examples are *not linearly separable*).

If you want to learn more about (non)convex optimization:

- Theory: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf
- Funny examples: http://www.stat.cmu.edu/~ryantibs/convexopt/lectures/nonconvex.pdf



"convex"

Each Godzilla defines a local minima and the "heaviest" Godzilla: The global minima

"non-convex"

Gradient Descent

- **Gradient descent** is a first-order **iterative** algorithm for finding a **local minimum** (global minimum if the function is convex) of a **differentiable** multivariate function.

- When use **gradient descent** to minimize $R_n(\theta)$, it simply iteratively updates the parameters:

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla_\theta R_n(\theta)_{\theta=\theta^{(k)}}$$

  - η_k is known as the *step-size* or *learning rate*.
  - The gradient points in the **direction** where $R_n(\theta)$ **increases**.

$$\nabla_\theta R_n(\theta) = \left[\frac{\partial R_n(\theta)}{\partial \theta_1}, \ldots, \frac{\partial R_n(\theta)}{\partial \theta_d}\right]^T$$

  - Thus, the weight is updated in the **opposite direction** to minimize error.

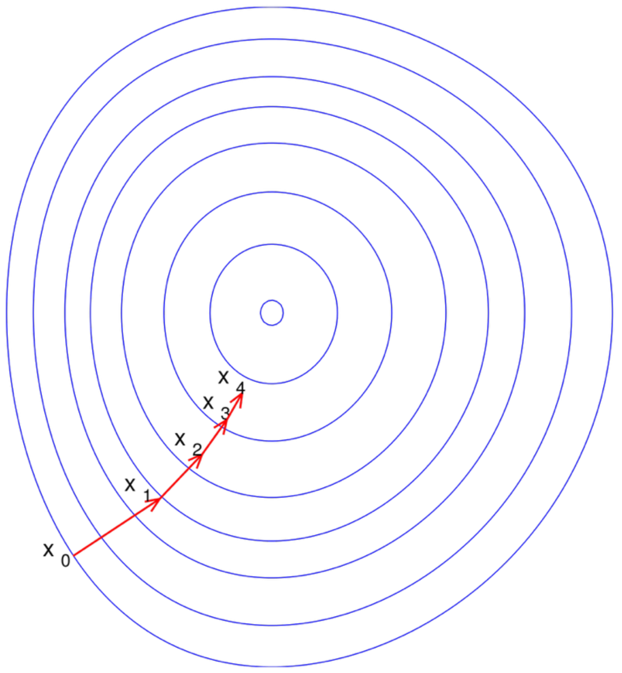


Illustration of gradient descent on a series of level sets

Stochastic Gradient Descent

- In **Gradient Descent**, the entire training dataset is used to compute the gradient.
- **Stochastic Gradient Descent (SGD)** is a stochastic approximation of gradient descent, as it replaces the actual gradient by an estimate calculated from a randomly selected sample.
  - Reduce computational cost
  - Converge faster (but with more erratic updates)

update may be in wrong direction

- The procedure of SGD algorithm:

$\theta^{(0)} = 0$ (vector) *# Random initialization*

select $t \in \{1, \ldots, n\}$ at random,

learning rate

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla_\theta \operatorname{Loss}_h(y^{(t)}\theta \cdot x^{(t)})_{|\theta=\theta^{(k)}}$$ *# Update*

Repeat the update step until stopping criterion is met.

Are we done here?

NO!!!

- **Hinge loss**

$$\text{Loss}_\text{h}(z) = \max\{1 - y(\theta^\top x), 0\}$$

- The hinge loss function is defined as **piecewise linear function**, which is **not everywhere differentiable**!🙁

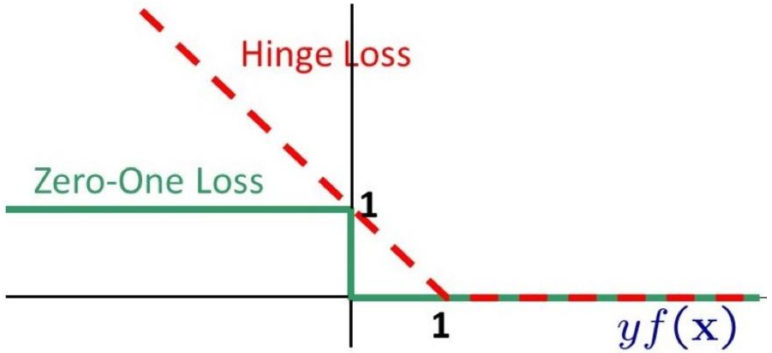
Sub-differential Problem

- $R_n(\theta)$ is also not differentiable everywhere as it is a summation of hinge loss functions.

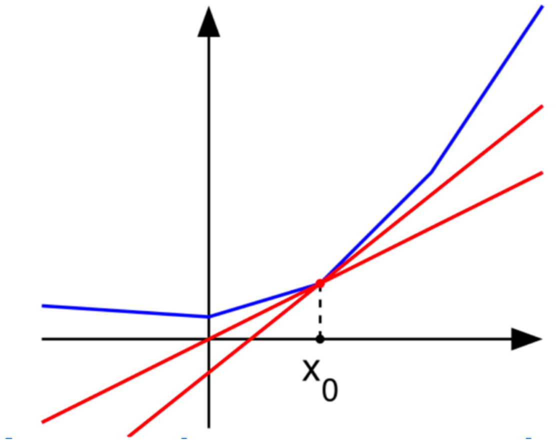
$$R_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} \underline{\text{Loss}_h(y^{(t)}(\theta \cdot x^{(t)}))} = \frac{1}{n}\sum_{t=1}^{n} \underline{\max\{1 - y^{(t)}(\theta \cdot x^{(t)}), 0\}}$$

- There are several possible gradients at the **kinks** (*points where the function is not differentiable*) which are collectively defined as **sub-differential**.

- In our case, to minimize $R_n(\theta)$, we need to select one possible gradient at each kink *regardless of* how many choices there are.



❖ Recall and observe the piece-wise linear shape of hinge loss function



A sample representation of sub-differential

Stochastic Sub-Gradient Descent

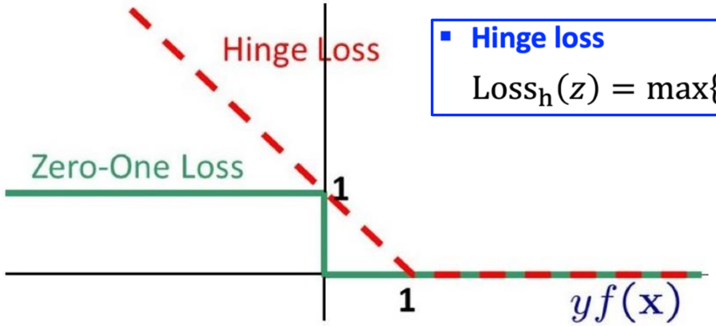
- Stochastic Gradient Descent

$\theta^{(0)} = 0$ (vector) *# Random initialization*

select $t \in \{1, \ldots, n\}$ at random,

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla_\theta \operatorname{Loss}_h(y^{(t)}\theta \cdot x^{(t)})_{|\theta=\theta^{(k)}}$$ *# Update*

Repeat the update step until stopping criterion is met.



- **Hinge loss**

$\operatorname{Loss}_h(z) = \max\{1 - y(\theta^\top x), 0\}$

If the agreement $y^{(t)}(\theta^{(k)}x^{(t)}) > 1$ then the loss is identically zero and so is the gradient. $\Rightarrow$ **No update**

Else: $\nabla_\theta \operatorname{Loss}_h(\underbrace{y^{(t)}\theta \cdot x^{(t)}}_{z})_{|\theta=\theta^{(k)}} = \underbrace{\nabla_\theta(1 - y^{(t)}\theta \cdot x^{(t)})_{|\theta=\theta^{(k)}}}_{\text{gradient}} = -y^{(t)}x^{(t)}$

- Stochastic Sub-Gradient Descent

$\theta^{(0)} = 0$ (vector) *# Random initialization*

select $t \in \{1, \ldots, n\}$ at random,

if $y^{(t)}(\theta^{(k)} \cdot x^{(t)}) \leq 1$, then

update only if this applicable

$$\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$$ *# Update*

Repeat the update step until stopping criterion is met.

perceptron algorithm

Does it look familiar?

Stochastic Sub-Gradient Descent VS Perceptron Algorithm

$\theta^{(0)} = 0$ (vector) *# Random initialization* 1

select $t \in \{1, \ldots, n\}$ at random, #random select training data from sample

if $y^{(t)}(\theta^{(k)} \cdot x^{(t)}) \leq 1$ 2, then

$$\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$$ *# Update*

Repeat the update step until stopping criterion is met.

$\max[1 - y(\theta \cdot x), 0]$

```
Random initialize θ = 0          (1)
while TRUE do
    m = 0
    for (x^(t), y^(t)) ∈ D do    (2)
        if y^(t)(θ · x^(t)) ≤ 0 then
            θ ← θ + y^(t) x^(t)
            m ← m + 1
        end if
    end for
    if m = 0 then
        break
    end if
end while
```

$\max[-y(\theta \cdot x, 0]$

- Differences
  1. SSGD also penalizes (update parameters) *near* mistakes, while the perceptron only penalizes mistakes. (due to zero-one loss)
  2. SSGD use a decreasing learning rate η_k (later updates will be smaller), while the perceptron sets it to one.
  3. SSGD randomly selects the training example, while the perceptron cycles through all examples in order.

How to select learning rate?

- Any positive **learning rate** that satisfies the following condition would permit the algorithm to converge *though the speed may vary*.

$$\sum_{k=1}^{\infty} \eta_k^2 < \infty, \quad \sum_{k=1}^{\infty} \eta_k = \infty$$

- In our context, $\eta_k = 1/(k+1)$ would ensure that our stochastic sub-gradient descent algorithm converges to the minimum of $R_n(\theta)$.
- In practice, one can choose other learning rates based on **empirical observations**.
  - Simply setting $\eta_k = 0.1$ turns out to be a popular choice in many problems.

How to get the best parameters?

- Due to the **erratic updates** in SGD, the $\theta^{(k)}$ may not be the best solution obtained so far (till k-th step).
- Hence, it is often quite beneficial to keep track of the best solution $\theta^{(i_k)}$, where

$$i_k = \operatorname{argmin}_{i=1,\ldots,k} R_n(\theta^{(i)})$$

$\theta^{(0)} = 0$ (vector) *# Random initialization*

select $t \in \{1, \ldots, n\}$ at random,

if $y^{(t)}(\theta^{(k)} \cdot x^{(t)}) \leq 1,$ then

$\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$ *# Update*

Repeat the update step until stopping criterion is met.

- When increasing k, the empirical risk of $\theta^{(k)}$, *i.e.*, $R_n(\theta^{(k)})$ goes down in a noisy fashion or oscillates, while $R_n(\theta^{(i_k)})$ is **monotonically decreasing** (*guaranteed by the definition!*)
- Note: $\lim_{k\to\infty} R_n(\theta^{(i_k)})$ is not necessarily zero as the points may be not linearly separable.

Linear Classifier (Non-separable case)

1. Training Set (Not Necessarily Linearly Separable)

$$\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$$

2. Model (Linear Classifier)

$$h(x;\theta) = \text{sign}(\theta_1 x_1 + \cdots + \theta_d x_d)$$

3. Training Error (Hinge Loss)

$$R_n(\theta) = \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_n} \max\{1 - y(\theta^\top x), 0\}$$

4. Algorithm (Stochastic Sub-Gradient Descent)

$$\text{select } t \in \{1, \ldots, n\} \text{ at random,}$$

$$\text{if } y^{(t)}(\theta^{(k)} \cdot x^{(t)}) \le 1, \text{ then}$$

$$\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$$ update θ

Summary

1. What is a convex function and how to check if a function is convex or not?

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

   Convex: Line segment between any pair of points on the curve of f to be above or just meets the graph.

2. What is hinge loss?

   Hinge loss is a convex surrogate for the zero-one loss, defined as $\boxed{\mathrm{Loss_h}(z) = \max\{1 - y(\theta^\top x), 0\}}$

3. How is hinge loss different from zero-one loss?

   Hinge loss function is convex, it penalizes near mistake and penalizes larger mistakes more than 0-1 loss.

4. What is stochastic gradient descent?

   SGD is a stochastic approximation of gradient descent by estimating the gradient from a randomly selected sample.

5. How to implement the stochastic (sub-)gradient descent?

   If the agreement $y^{(t)}(\theta^{(k)}x^{(t)}) > 1$ then the loss is identically zero and so is the gradient. ⇒ **No update**

   Else: $\nabla_\theta \operatorname{Loss}_h(y^{(t)}\theta \cdot x^{(t)})_{|\theta=\theta^{(k)}} = \nabla_\theta(1 - y^{(t)}\theta \cdot x^{(t)})_{|\theta=\theta^{(k)}} = -y^{(t)}x^{(t)}$

Acknowledgements

- Some slides and content of this lecture are adopted from:
  - McGill COMP-652 Machine Learning
  - SUTD 50.007 Machine Learning, Spring 2023 (Asst Prof. Malika Meghjani)
  - MIT 6.036 Introduction to Machine Learning
  - Shalev-Shwartz, Shai, and Shai Ben-David. ”Understanding machine learning: From theory to algorithms”. Cambridge university press, 2014. (Chapter 8.2 & 12.3)