01.112/50.007 Machine Learning

Lecture 3

Hinge Loss

Recap

Training data

$$\mathcal{S}_n = \left\{ \left(x^{(i)}, y^{(i)}\right) \middle| \, i = 1, \dots, n \right\}$$

- Features/Inputs $x^{(i)} = \left(x_1^{(i)}, \dots, x_d^{(i)}\right)^\top \in \mathbb{R}^d$
- Labels/Output $y^{(i)} \in \{-1, +1\}$

Model

Set of *linear* classifiers $h: \mathbb{R}^d \rightarrow \{-1, +1\}$

$$h(x; \theta, \theta_0) = \text{sign}(\theta_d x_d + \cdots + \theta_1 x_1 + \theta_0)$$

$$= \text{sign}(\theta^\top x + \theta_0) = \begin{cases} +1 & \text{if } \theta^\top x + \theta_0 \geq 0, \\ -1 & \text{if } \theta^\top x + \theta_0 < 0. \end{cases}$$

Model Parameters

$$\theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}$$

Also called the *offset*

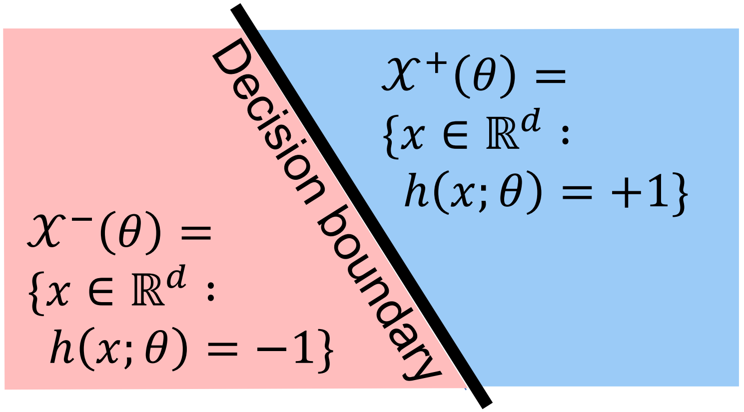
$[\![\cdot]\!]$ is the *indicator* function that returns a 1 if its argument is true, and 0 otherwise.

Training Error

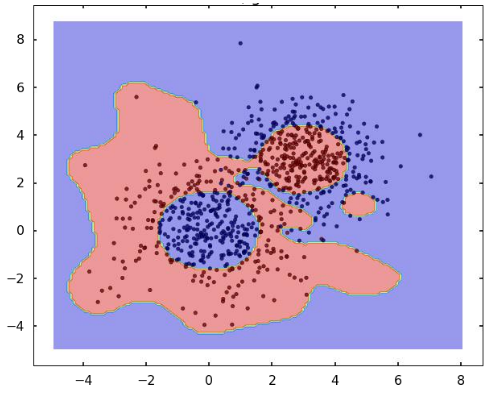
$$Loss(\theta, \theta_0; x, y) = [\![y \neq h(x; \theta, \theta_0)]\!]$$

$$\mathcal{R}(\theta, \theta_0; \mathcal{S}_*) = \frac{1}{n} \sum_{(x,y) \in \mathcal{S}_*} Loss(\theta, \theta_0; x, y)$$

Decision Regions



linear classifier



non-linear classifier

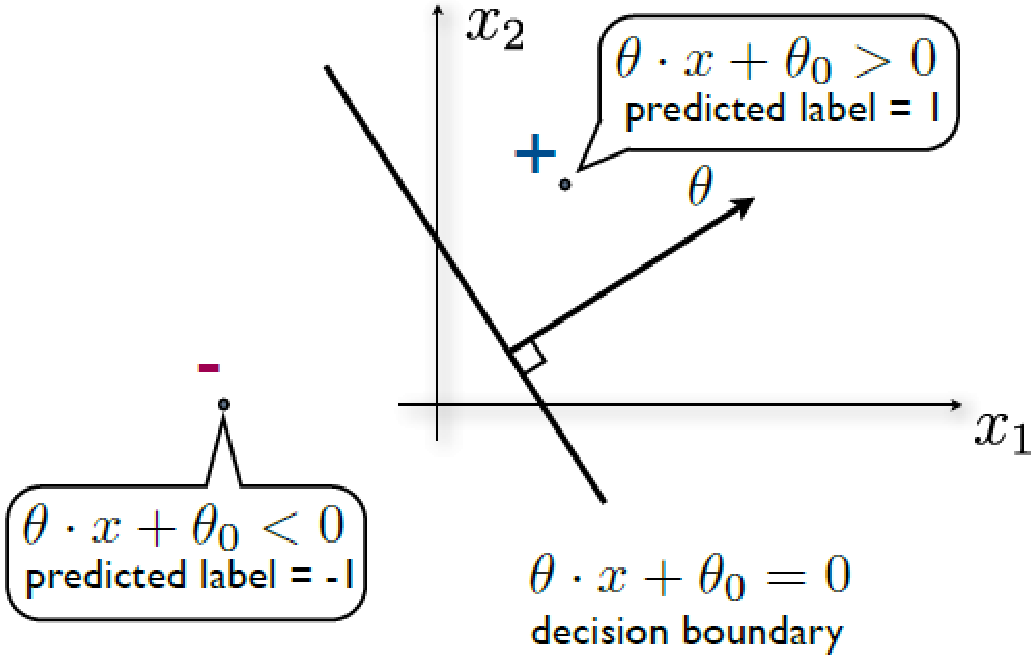
A classifier h partitions the space into decision regions that are separated by decision boundaries. In each region, all the points map to the same label. Many regions could have the same label.

For linear classifiers, these regions are half spaces.

Decision Boundaries

Vector θ is orthogonal to the decision boundary.

Vector θ points in direction of region labelled $+1$.



Linear Classifier

- **Linear classifier (with offset):**

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

- **Training error:**

$$\mathcal{E}_n(\hat{\theta},\hat{\theta}_0) = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)}(\hat{\theta}\cdot x^{(t)} + \hat{\theta}_0) \leq 0]\!]$$

$[\![\cdot]\!]$ is the *indicator* function that returns a 1 if its argument is true, and 0 otherwise.

- Linear classifier that achieves zero training error is called **realizable**.

Zero-One Loss

$$\mathcal{E}_n(\hat{\theta}, \hat{\theta}_0) = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)}(\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) \leq 0]\!]$$

Let $\mathcal{E}_n(\hat{\theta}, \hat{\theta}_0)$ = 1 (0 otherwise) if

- $y \neq h(x; \theta)$, or [misclassified]
- (x, y) is on decision boundary [boundary]

Note that $y(\theta^\top x) \leq 0$ if

- $\theta^\top x$ and y differ in sign, or [misclassified]
- $\theta^\top x$ is zero [boundary]

Perceptron Algorithm

- Initialize the **weight** ($\theta = 0$).
- For each training example 't' in S_n, classify the instance
  - if correct, continue
  - else, $\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$

    $\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)}$
- Terminate if the **training error** is zero (realizable) or a pre-determined number of iterations are completed (non-realizable).

Perceptron update rule

Linearly separable case

Theorem 2.1 *The perceptron update rule converges after a finite number of mistakes when the training examples are linearly separable through origin.*

- The above theorem implies, **zero training error** can be achieved using **perceptron update rule** for **linearly separable** training examples.

$$\mathcal{E}_n(\hat{\theta}, \hat{\theta}_0) = \frac{1}{n} \sum_{t=1}^{n} [\![y^{(t)}(\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) \leq 0]\!] = 0$$

Perceptron Algorithm

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 (Set of Perceptrons)

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

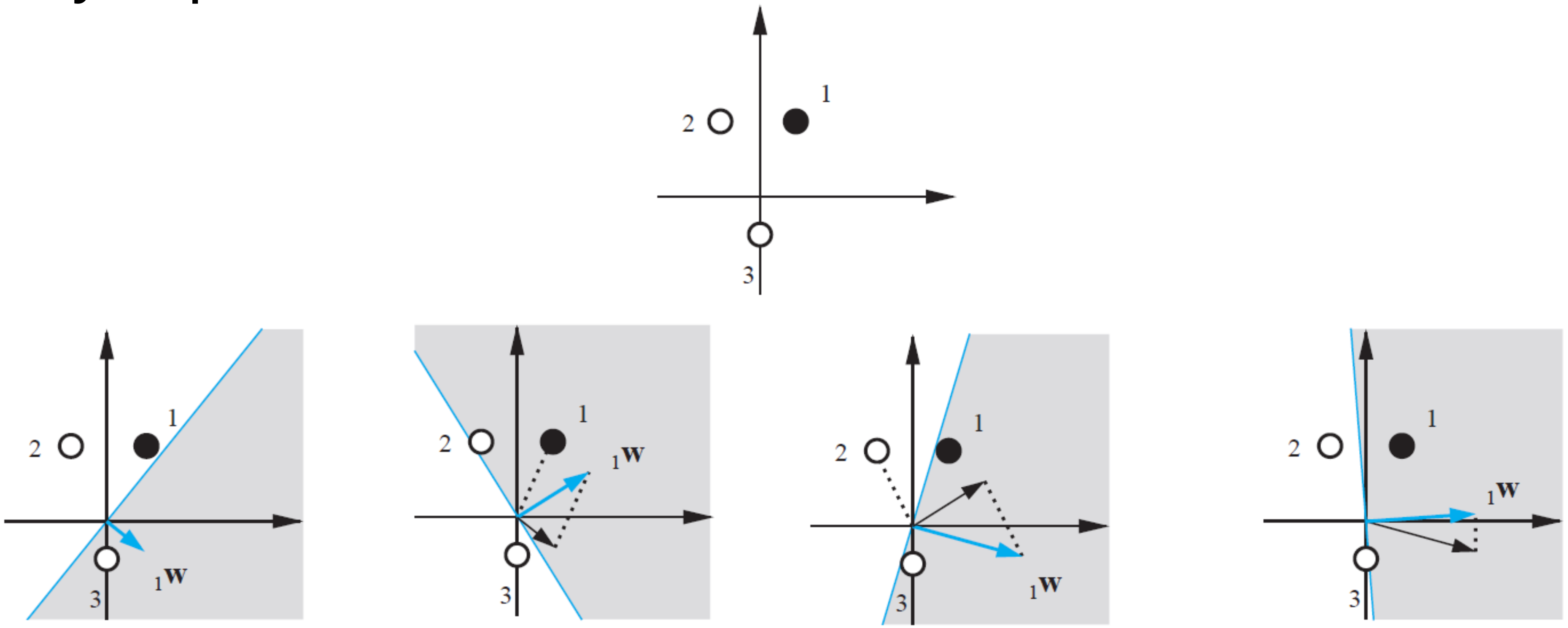
3. Training Loss (Fraction of Misclassified/Boundary Points)

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

3. Algorithm (Mistake-Driven Algorithm)

Example (Linearly Separable)

- Perceptron algorithm oscillates and terminates with zero error in linearly separable case.



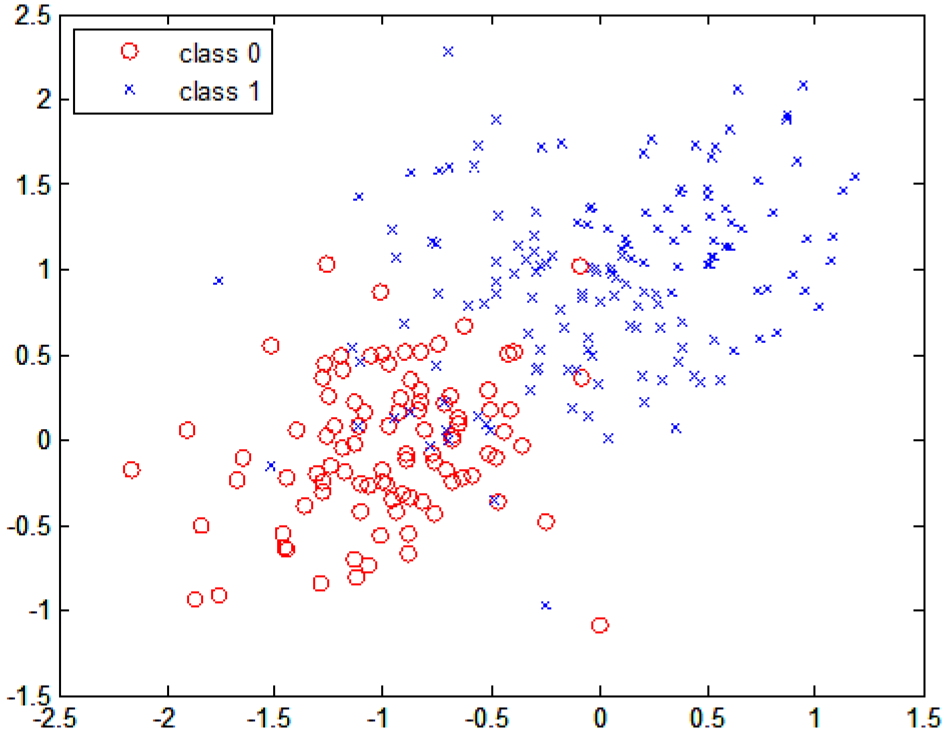
Linear Classifier

Non-Separable Case

Non-Separable case

- **Perceptron algorithm** will **not converge nor** will it find the classifier with the **smallest error**, if training example are **linearly not separable**.

- **Goal:** Find a **classifier that minimizes the training error** in the **non-realizable case.**

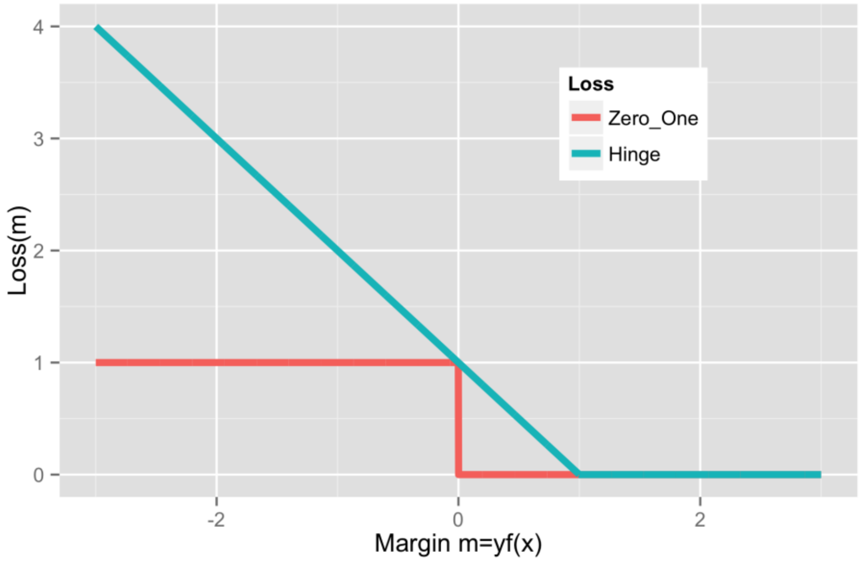


Loss Functions

- **Training Loss / Empirical risk:**

$$R_n(\theta) = \frac{1}{n} \sum_{\text{data } (x,y)} \text{Loss}\big(y(\theta^\top x)\big)$$

- **Zero-one loss:** $\text{Loss}_{0|1}(z) = [\![\, z \le 0 \,]\!]$



- **Hinge loss:** $\text{Loss}_{\text{h}}(z) = \max\{1 - z, 0\}$

CONVEX!
Penalize larger mistakes more.
Penalize near-mistakes, i.e. $0 \le z \le 1$.

Hinge Loss (Examples)

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

Example 1

- 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 a very high penalty since the prediction was inaccurate

Hinge Loss (Examples)

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

Example 1

- 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 a very high penalty since the prediction was inaccurate

Example 2

- 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 (Examples)

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

Example 1

- 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 a very high penalty since the prediction was inaccurate

Example 2

- 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

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 (loss is very less but not 0, since the prediction is accurate and has high confidence but not 100%)

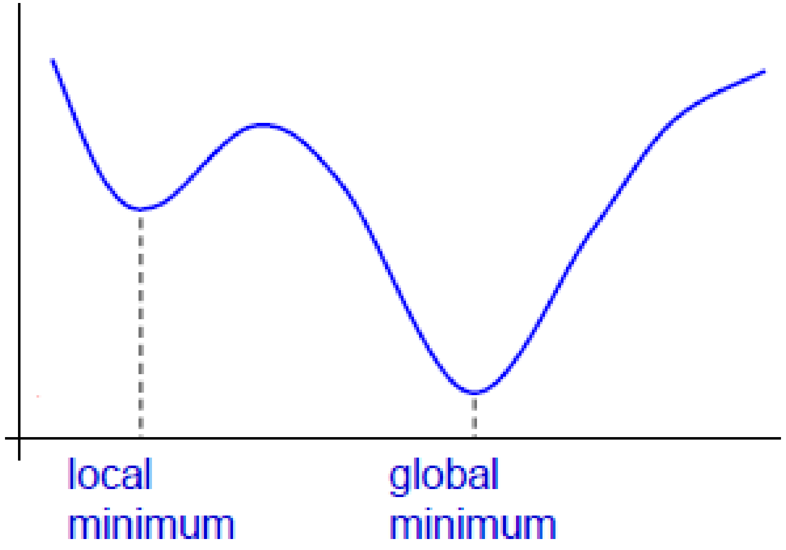
Hinge Loss

- **Empirical risk using hinge loss:**

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

- **Convexity** of empirical risk allows us to find the **minimum** even in **non-realizable** case.

Optimization



- Does this loss function have a unique solution?
- If the loss function is convex, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case)

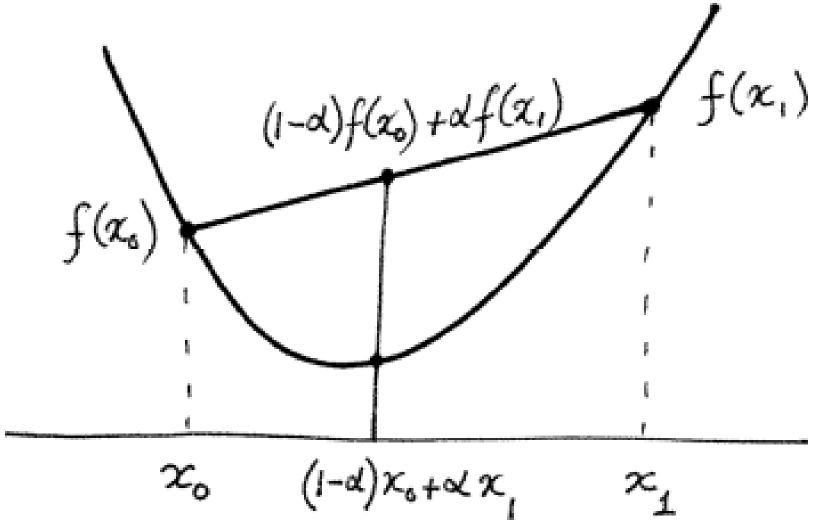
Convex Functions

D – a domain in $\mathbb{R}^n$.

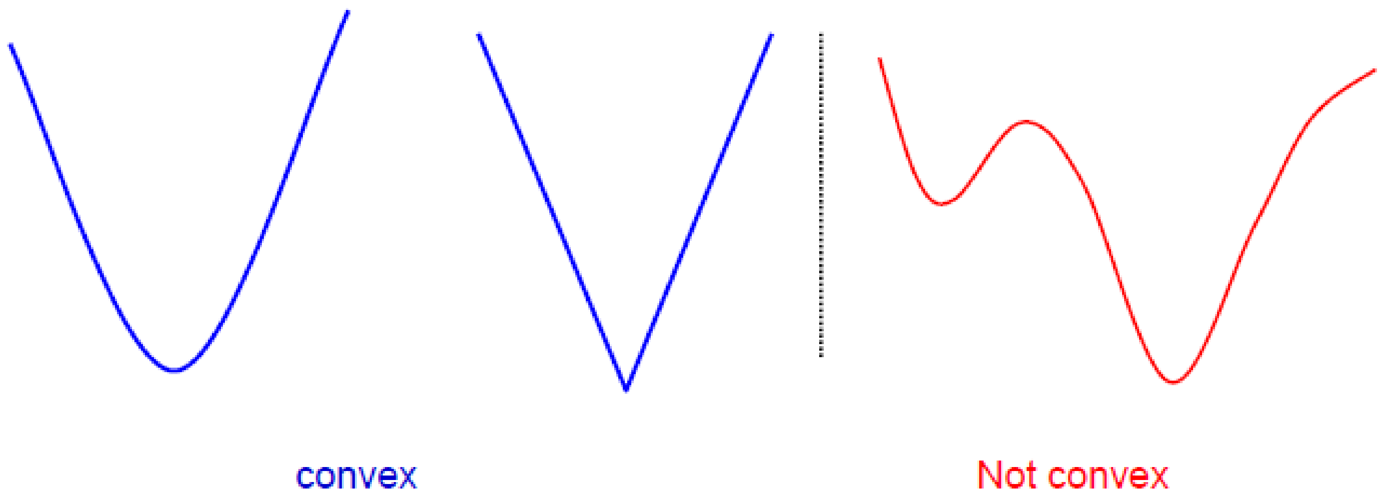
A convex function $f : D \to \mathbb{R}$ is one that satisfies, for any $\mathbf{x}_0$ and $\mathbf{x}_1$ in D:

$$f((1-\alpha)\mathbf{x}_0 + \alpha\mathbf{x}_1) \leq (1-\alpha)f(\mathbf{x}_0) + \alpha f(\mathbf{x}_1) \quad .$$

Line joining $(\mathbf{x}_0, f(\mathbf{x}_0))$ and $(\mathbf{x}_1, f(\mathbf{x}_1))$ lies above the function graph.



Convex Function Examples



A non-negative sum of convex functions is convex

Gradient Descent

- Use **gradient descent** to minimize $R_n(\theta)$

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

- Gradient points in the direction where $R_n(\theta)$ **increases**.
- Need to update the weight in the **opposite direction**.

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

Gradient Descent

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

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

- Need to update the weight in the **opposite direction**.

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

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

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

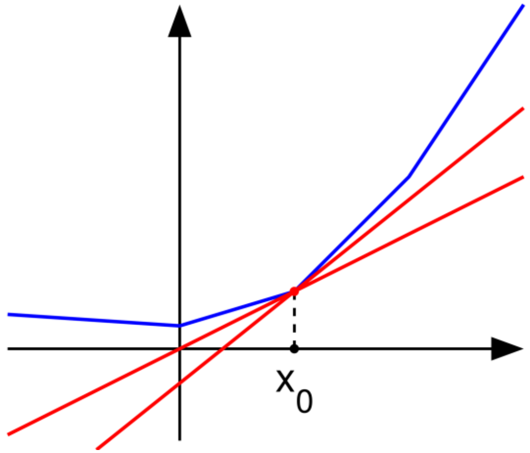
$$\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$$

Gradient Descent

- $R_n(\theta)$ is **not differentiable** everywhere as **hinge loss functions** are **piece-wise linear**.

- There are several **possible gradients** at the kinks which are collectively defined as **sub-differential**.

- To minimize $R_n(\theta)$, we need to select one possible gradient at any point.



Stochastic Gradient Descent

1. Initialize the **weight** ($\theta^{(0)} = 0$).
2. Select $t \in \{1, \ldots, n\}$ at random

- If $y^{(t)}(\theta^{(k)} \cdot x^{(t)}) \leq 1$, then update the weight

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

3. Repeat Step (2) until stopping criterion is met.
(e.g. when improvement in $R_n(\theta)$ is small enough)

Stochastic Gradient Descent

Differences from Perceptron algorithm:

- Near mistakes are also penalized

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

- Learning rate is decreasing (later updates will be smaller)

$$\eta_k = 1/(k+1)$$

- Random sample avoid oscillations

Stochastic Gradient Descent

- Keep track of best solution (weight), $\theta^{(i_k)}$, where,
  $i_k = \text{argmin}_{i=1,\ldots,k} R_n(\theta^{(i)})$

- **Note: asymptotically empirical risk does not become zero as the points may not be linearly separable.**

Hinge Loss Algorithm

1. Training Set (Not Necessarily Linearly Separable)

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

2. Model (Set of Perceptrons)

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

3. Training Loss (Hinge Loss)

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

3. Algorithm (Gradient Descent)

$$\theta \longleftarrow \theta + \frac{\eta_k}{n} \sum_{(x,y) \in \mathcal{S}_n} yx$$

Summary

- **Linear Classification**
  - Decision Region
  - Decision Boundary
  - Linearly Separable
- **Perceptron Algorithm**
  - Zero-One Loss
  - Mistake-Driven
  - Only for Linearly Separable Data
- **Hinge Loss**
  - Gradient Descent
  - Hinge Loss SGD Algorithm
  - Differences with Perceptron Algorithm
  - OK for Non Linearly Separable Data

Intended Learning

Hinge Loss

- Write down the hinge loss, and plot its graph. Write down the training loss and its gradient.
- List differences between the hinge loss SGD algorithm and the perceptron algorithm.
- Explain why the hinge loss SGD applies to non linearly separable data while the perceptron algorithm does not.