# Perceptron and Neural Networks

Huang Yanzhen

June 2, 2025

# 1 Perceptron Algorithm

## 1.1 Problem Setup

**Given:**

A set of $l$-dimensional data samples:

$$\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^l \tag{1}$$

Two classes:

$$\Omega = \{\omega_1, \omega_2\} \tag{2}$$

A ground-truth classification relation (i.e., which sample belongs to which class):

$$R : \mathcal{X} \mapsto \Omega \tag{3}$$

**Do:**

Our goal is to decide a hyperplane $g(\mathbf{x})$:

$$g(\mathbf{x}) : \mathbf{w}^\top \mathbf{x} + w_0 = 0 \tag{4}$$

where $\mathbf{w}$ is the $l$-dimensional column weight vector and $w_0$ is the threshold.

Assume that the classes are linearly seperable. Thus, we are able to find a weight vector $\mathbf{w}$ that separates the two types.

$$\exists \mathbf{w}, w_0,\ \begin{cases} \mathbf{w}^\top \mathbf{x} + w_0 > 0,\ R(\mathbf{x}) = \omega_1 \\ \mathbf{w}^\top \mathbf{x} + w_0 < 0,\ R(\mathbf{x}) = \omega_2 \end{cases} \tag{5}$$

## 1.2 Notations

More specifically, we can write the hyperplane as:

$$g(\mathbf{x}) : \begin{bmatrix} w_1 & w_2 & \cdots & w_l \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix} + w_0 = 0 \tag{6}$$

This is equivalent to:

$$g(\mathbf{x}) : \begin{bmatrix} w_1 & w_2 & \cdots & w_l & w_0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \\ 1 \end{bmatrix} = 0 \tag{7}$$

To unify everything into matrix calculation, we omit the explicit expression of the threshold $w_0$ by:

$$\mathbf{w}_{\text{new}} = \begin{bmatrix} \mathbf{w}_{\text{old}} \\ w_0 \end{bmatrix} \tag{8}$$

$$\mathbf{x}_{\text{new}} = \begin{bmatrix} \mathbf{x}_{\text{old}} \\ 1 \end{bmatrix} \tag{9}$$

Namely,

$$\mathbf{w}_{\text{old}}\mathbf{x}_{\text{old}} + w_0 \equiv \mathbf{w}_{\text{new}}\mathbf{x}_{\text{new}} \tag{10}$$

The property of the hyperplane mentioned in 1.1 would be written as follows:

$$\exists \mathbf{w} \in \mathbb{R}^{l+1}, \begin{cases} \mathbf{w}^\top \mathbf{x} > 0 & R(\mathbf{x}) = \omega_1 \\ \mathbf{w}^\top \mathbf{x} < 0 & R(\mathbf{x}) = \omega_2 \end{cases} \tag{11}$$

In the following part of the document, this notation is implicitly used.

### 1.3 Perceptron Cost Function

A **cost function** $\mathcal{J}(\mathbf{w})_{\delta_\mathbf{x}}$ is defined by the following:

$$\mathcal{J}(\mathbf{w})_{\delta_\mathbf{x}} = \sum_{\mathbf{x} \in \mathcal{X}_\mathrm{e}} \delta_\mathbf{x} \mathbf{w}^\top \mathbf{x} \tag{12}$$

where:

- $\mathcal{X}_\mathrm{e} \subset \mathcal{X}$ is a set of all wrongly-classified data samples. Surely, $\mathbf{X}_\mathrm{e}$ is a subset of all the data samples $\mathbf{X}$.
- $\delta_\mathbf{x}$ is the cost, i.e., the punishment for this data sample to be different as the ground truth. In this section, we define it as follows:
  $\delta_\mathbf{x} = \begin{cases} -1 & \text{if } \mathbf{x} \in \omega_1 \\ +1 & \text{if } \mathbf{x} \in \omega_2 \end{cases}$

Q: Why is the cost function only accepting the parameter of $\mathbf{w}$?

A: Because $\mathbf{x}$ are ground-truths that can't be changed. We are finding a hyperplane, i.e., a $\mathbf{w}$ that fits the unchanging $\mathbf{x}$.

The gradient of the cost function:

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{w})_{\delta_\mathbf{x}}}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \sum_{x \in \mathcal{X}_\mathrm{e}} \delta_\mathbf{x} \mathbf{w}^\top \mathbf{x} & (13) \\ &= \sum_{\mathbf{x} \in \mathcal{X}_\mathrm{e}} \delta_\mathbf{x} \mathbf{x} \in \mathbb{R}^{l+1} & (14) \end{aligned}$$

The gradient is a vector with the same shape as the weight vector $\mathbf{w}$, i.e., $l+1$. The gradient descent algorithm would be:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \gamma_t \left. \frac{\partial \mathcal{J}(\mathbf{w})_{\delta_\mathbf{x}}}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t} & (15) \\ &= \mathbf{w}_t - \gamma_t \sum_{\mathbf{x} \in \mathcal{X}_\mathrm{e}} \delta_\mathbf{x} \mathbf{x} & (16) \end{aligned}$$

where $\gamma_t$ is the learning rate at step $t$[1].

[1]The learning rate belongs to an explicit field of optimizers, which will not be discussed in this chapter. Just know that it is a scalar that changes every step.