Week02 - Part 01

Review: Binary Linear Classification Learning Model

- **Training Set:** $\mathcal{D} = \{(\underline{x}_1, y_1), \cdots, (\underline{x}_N, y_N)\}$

  $\underline{x}_n \in \mathcal{X}$, $\underline{x}_n = (x_{n1}, x_{n2}, \cdots, x_{nd})$, $y_n \in \{-1, +1\} = \mathcal{Y}$

- **Task:** Given any $\underline{x} \in \mathcal{X}$, output $\hat{y} \in \{-1, +1\} = \mathcal{Y}$

- **Hypothesis (Decision Rule):** $h(\underline{x}) = \text{Sign}\left(\sum_{i=1}^{d} w_i x_i + b\right)$

- **Training:**

$$E_{in}(\underline{w}, b) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left(f(\underline{x}_n) \neq h(\underline{x}_n)\right) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left(y_n \neq \text{Sign}\left(\sum_{i=1}^{d} w_i x_{ni} + b\right)\right)$$

Simple Learning Model Diagram

Unknown Target function $y = f(x)$

$\downarrow$

Training Example
$D = \{(\underline{x}_n, y_n)\}_{n=1}^{N}$

$\downarrow$

Hypothesis set $\mathcal{H}$ $\rightarrow$ Learning Algorithm $\mathcal{A}$ $\rightarrow$ final hypothesis $g \approx f$

Last lecture:

- We saw that finding a linear classifier that minimizes E_{in} is NP-hard
- However, if the dataset is linearly separable, we have an algorithm that can find the perfect linear classifier efficiently.
- That algorithm is Perceptron Learning Algorithm (PLA)

Today:

- Perceptron Learning Algorithm

Perceptron Learning Algorithm

- Efficiently finds a <u>perfect</u> discriminator for linearly separable data set.
- To have cleaner math, we change our notation a bit

Old formulation of the decision rule: $h(\underline{x}) = \text{Sign}\left(b + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \right)$

Augment $\underline{x} = (x_0 = 1, x_1, \dots, x_d)$ $\quad \underline{w} = (w_0 = b, w_1, \dots, w_d)$

$b + w_1 x_1 + \dots + w_d x_d = \underline{w}^T \underline{x}$

New formulation: $\underset{\sim}{h}(\underline{x}) = \text{Sign}(\underline{w}^T \underline{x})$ ⟵ it is called "perceptron"

Observe that $\underline{x} \in \{1\} \times \mathcal{X}$

New formulation of Binary Linear Classification

- **Training Set:** $\mathcal{D} = \{(\underline{x}_1, y_1), \cdots, (\underline{x}_N, y_N)\}$

$\underline{x}_n \in \{1\} \times \mathcal{X}$, $\quad \underline{x}_n = (x_{n0} = 1, x_{n1}, x_{n2}, \cdots, x_{nd})$, $\quad y_n \in \{-1, +1\} = \mathcal{Y}$

- **Hypothesis set:** $h_{\underline{w}} \in \mathcal{H}$, where $h_{\underline{w}}(\underline{x}) = \text{sign}(\underline{w}^T \underline{x})$

weight vector: $\underline{w} = (w_0, w_1, \cdots, w_d) \in \mathbb{R}^{d+1}$

- **Training:** Minimize $E_{in}(\underline{w}) = \dfrac{1}{N} \sum_{n=1}^{N} \mathbb{1}(y_n \neq h_{\underline{w}}(\underline{x}))$

$$= \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left(y_n \neq \text{Sign}(\underline{w}^T \underline{x})\right)$$

Perceptron Learning Algorithm (PLA)

Input: training set D that is linearly separable

Output: $\underline{w} \in \mathbb{R}^{d+1}$ that achieves $E_{in}(\underline{w}) = 0$

Initialization: choose arbitrary $\underline{w}$, e.g., $\underline{w} = \underline{0}$

Step 1: check if $E_{in}(\underline{w}) = 0$. If yes, stop and return $\underline{w}$.

Step 2: Let $(\underline{x}_n, y_n)$ be a miss-classified point, i.e., $y_n \neq \hat{y}_n$ (including the points on the boundary)

$$\left.\begin{array}{l} \text{If } y_n = +1, \quad \underline{w} \leftarrow \underline{w} + \underline{x}_n \\ \text{If } y_n = -1, \quad \underline{w} \leftarrow \underline{w} - \underline{x}_n \end{array}\right\} \quad \underline{w} \leftarrow \underline{w} + y_n \underline{x}_n$$

Go to Step 1.

<demo: vinizinho's PLA visualization>

Why Does PLA Work? (Intuitive explanation)

Let's have a closer look at datapoint $(\underline{x}_n, y_n)$

y_n	$\underline{w}^T \underline{x}_n$	correct classification?	$y_n \underline{w}^T \underline{x}_n$
$+1$	> 0	✓	> 0
$+1$	< 0	✗	≤ 0
-1	> 0	✗	≤ 0
-1	< 0	✓	> 0
± 1	$= 0$	✗	≤ 0

Let's have a closer look at the updating rule.

- Suppose $(\underline{x}_n, y_n)$ is missclassified.

$$\underline{w}_{new} = \underline{w} + y_n \underline{x}_n$$

- Now let's see what is the impact of updating $\underline{w}$ on how we classify $\underline{x}_n$

$$y_n \underline{w}_{new}^T \underline{x}_n = y_n (\underline{w} + y_n \underline{x}_n)^T \underline{x}_n = y_n (\underline{w}^T + y_n \underline{x}_n^T) \underline{x}_n$$

$$= y_n \underline{w}^T \underline{x}_n + (y_n)^2 \underline{x}_n^T \underline{x}_n = \underbrace{y_n \underline{w}^T \underline{x}_n}_{\text{we know that it is non-positive}} + \underbrace{y_n^2}_{=1} \underbrace{\|\underline{x}_n\|^2}_{>0}$$

- Observe that $y_n \underline{w}_{new}^T \underline{x}_n > y_n \underline{w}^T \underline{x}_n$ because $x_{n_0} = 1$

Note: What we saw was an <u>intuitive</u> explanation. Although PLA update rule give us a better classifier for the missclassified point $\underline{x}_n$, it may cause new missclassification for other points?

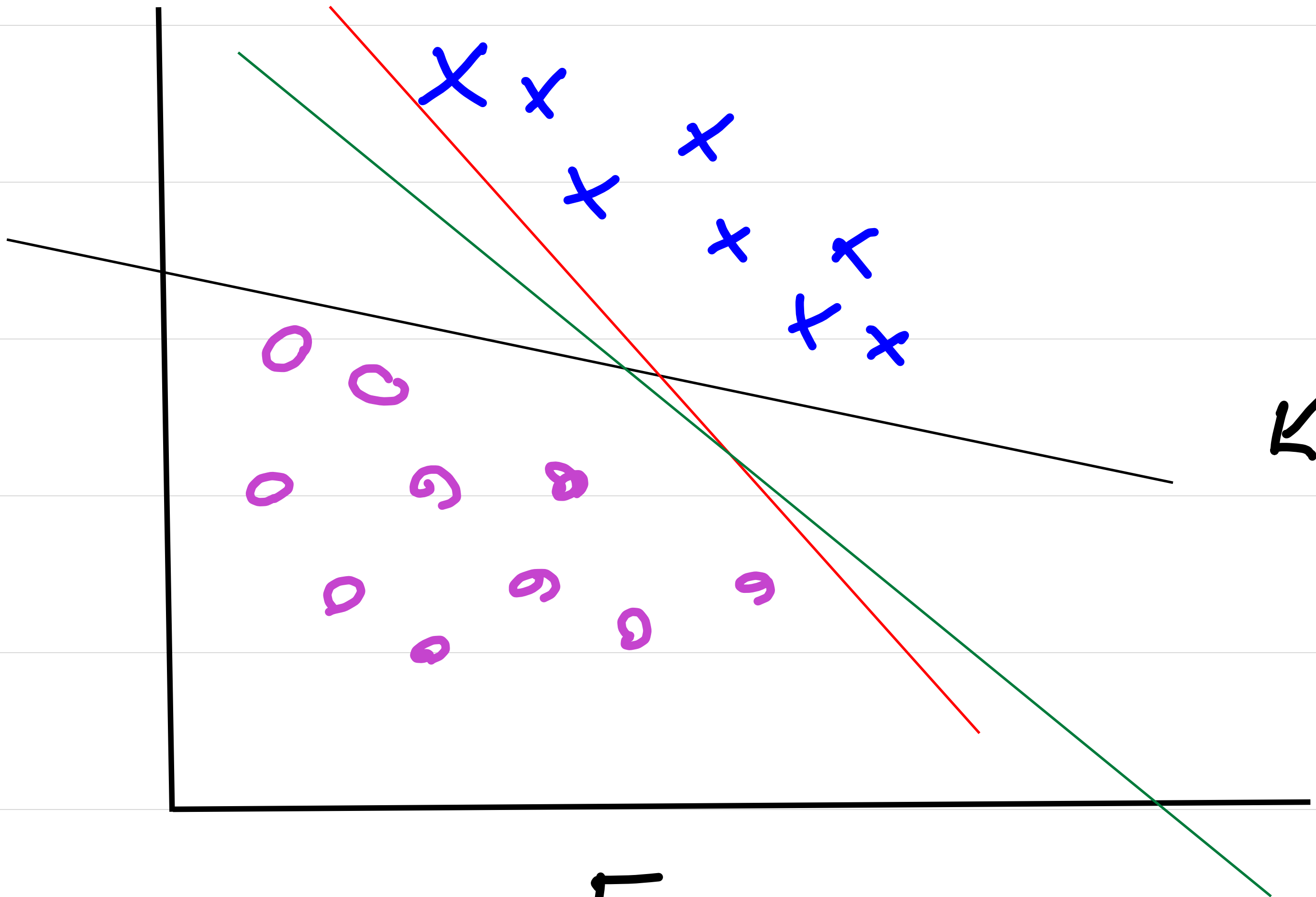
- We need more than intuitive explanation to show that PLA indeed works.

 (This was proved by Rosenblatt, 1957)

Rosenblatt Theorem: Given a linearly separable dataset, PLA terminates in a finite number of steps yielding $E_{in}(\underline{w}) = 0$

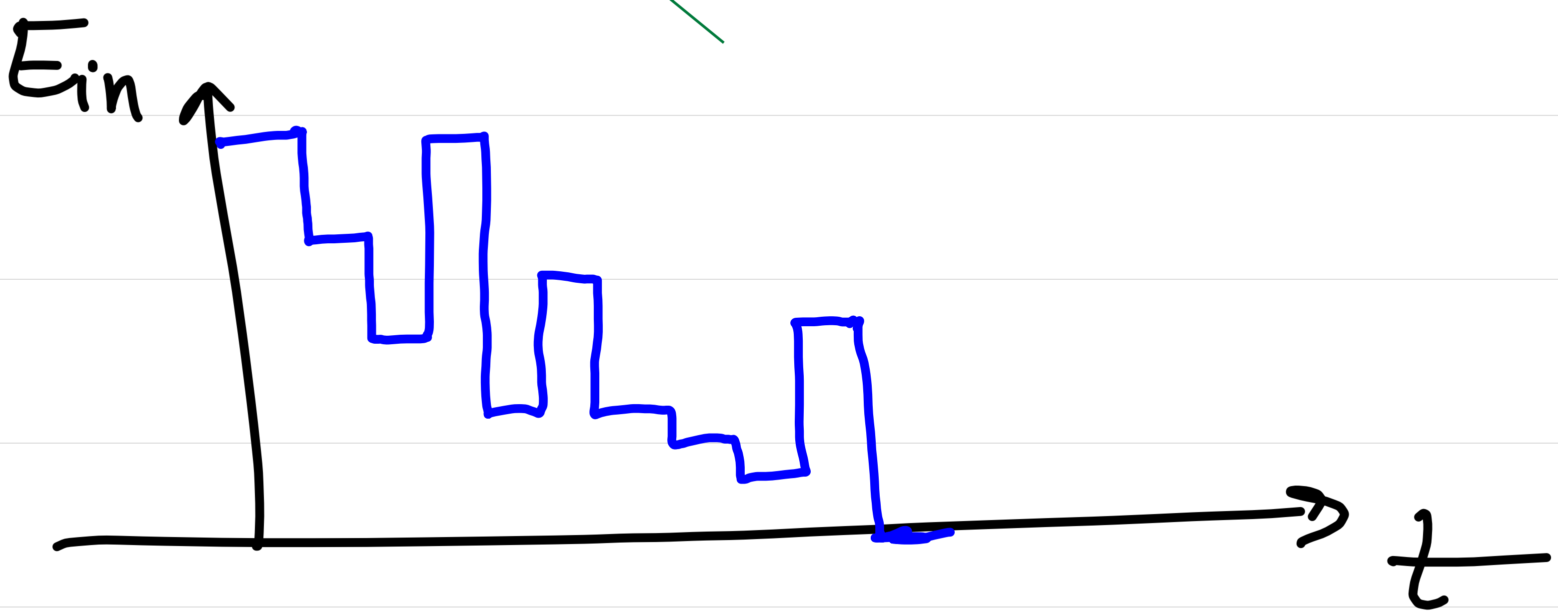
(If you are interested, the proof is in Problem 1.3 of LFD)

Remark: The output of PLA is not unique.



which line is better?

Remark:



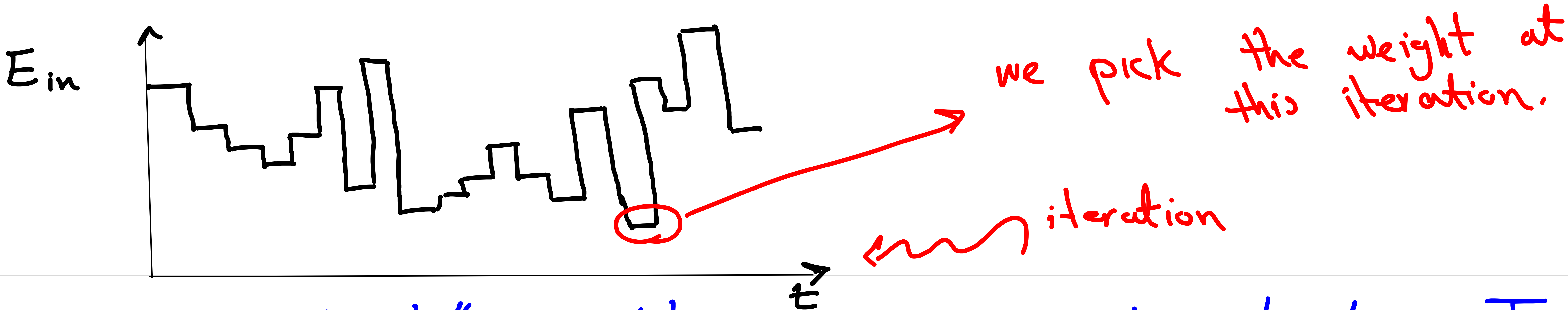
- So far we have only considered linearly Separable dataset and Saw that PLA works for such dataset.

- What if the dataset is NOT linearly separable?
    - What would happen if we use PLA for such datasets? Never stops.

- How can we modify PLA to work with non-separable dataset?

Pocket Algorithm

- Pocket Algorithm extends PLA for dataset that are not linearly separable.



Keep the "best" weight vector $\underline{w}$ upto iteration T in the Pocket.

Pocket Algorithm:

Pick time horizon T

Set Pocketed weight vector $\underline{w}^*$ to $\underline{w}(0)$ in PLA.

for $t = 1, 2, \dots, T$:

    Run PLA for one update to obtain $\underline{w}(t)$

    Evaluate $E_{in}(\underline{w}(t))$

    if $E_{in}(\underline{w}(t)) < E_{in}(\underline{w}^*)$ **then**

        Set $\underline{w}^* = \underline{w}(t)$

    End if

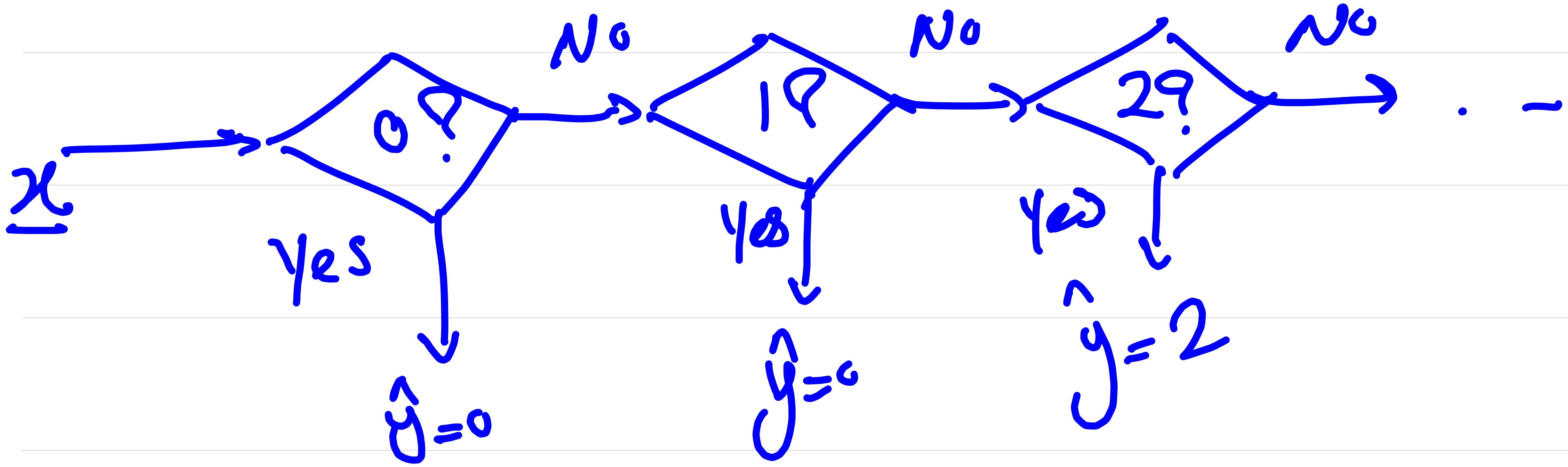
End for

Return $\underline{w}^*$

- So far, we saw binary classification.

- Can we use Perceptron idea to do classification with more than two classes (i.e. multiary classification)?

- Multiary classification



Not efficient/effective.