Machine Learning

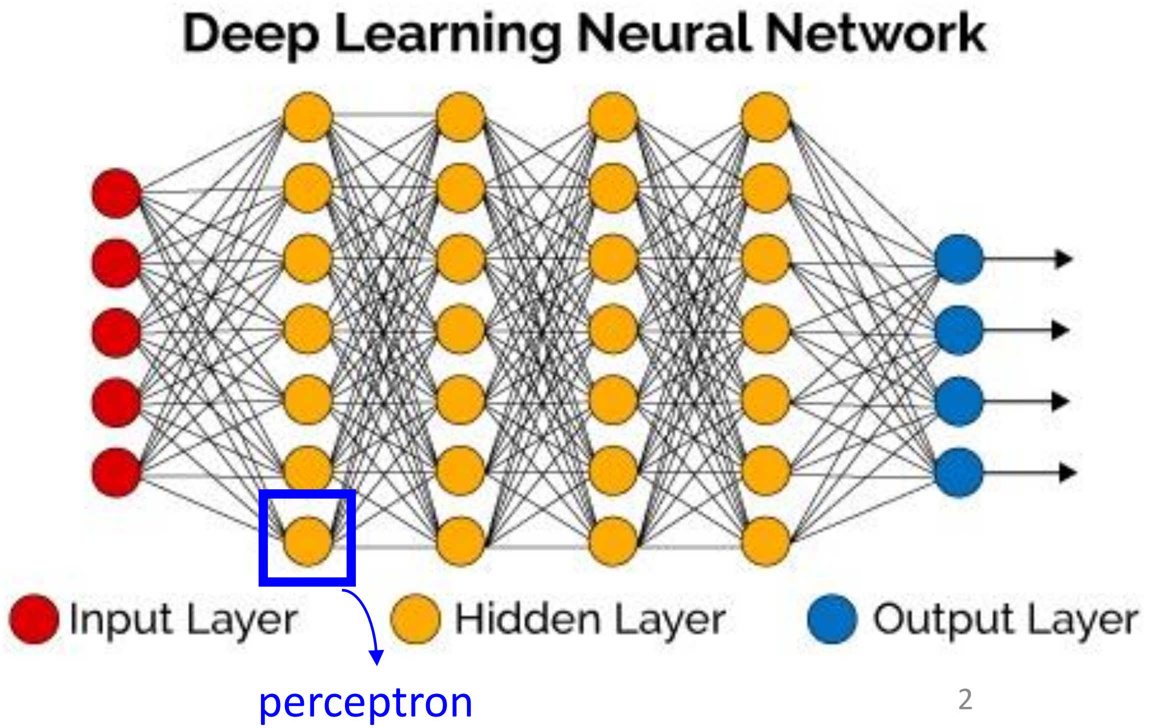
Lecture 5 – Perceptron

Chen-Kuo Chiang (江 振 國)
ckchiang@cs.ccu.edu.tw

中正大學 資訊工程學系

深度學習架構最基礎的模型 – Perceptron

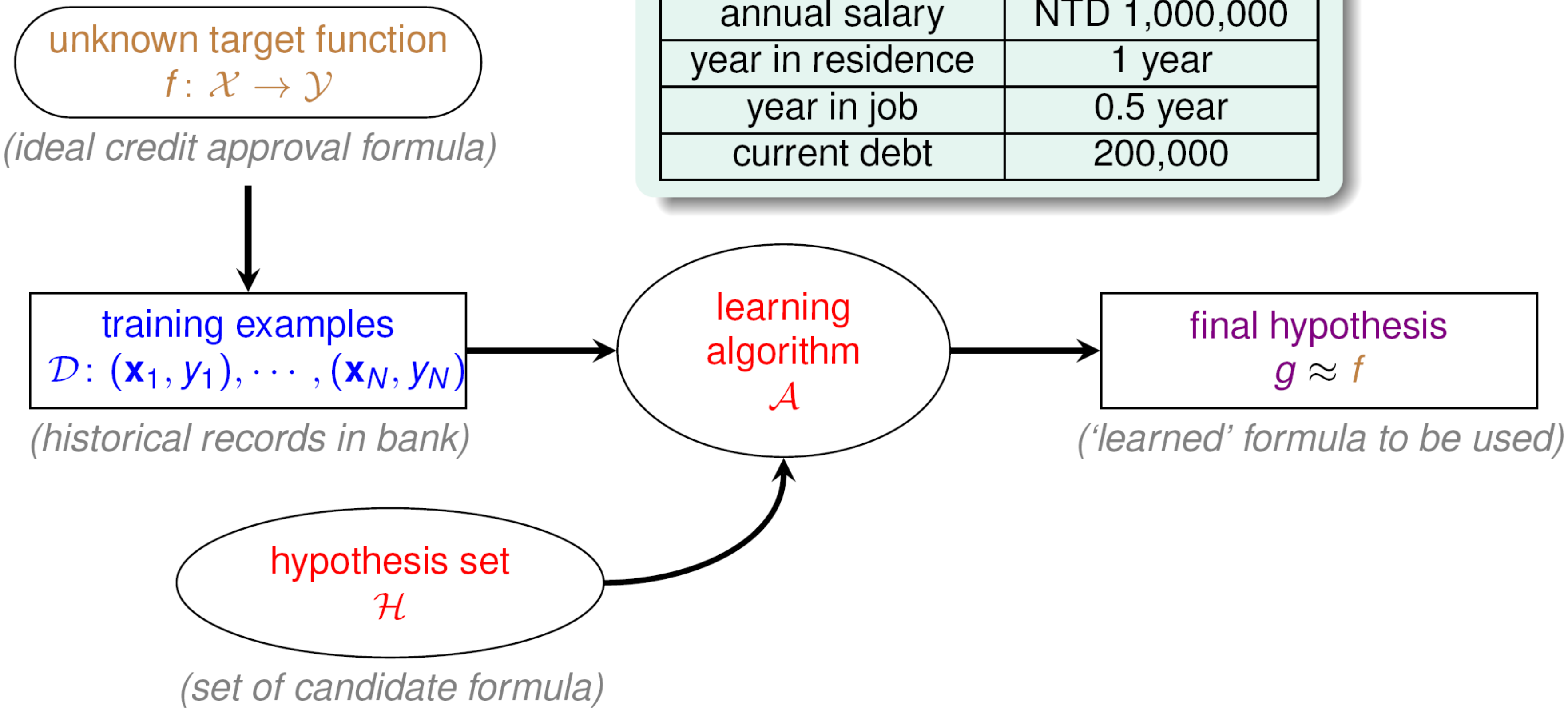
- Perceptron (深度學習模型的單一節點)
  - Perceptron Hypothesis Set
  - Perceptron Learning Algorithm (PLA)
  - Guarantee of PLA
  - Dealing with Non-separable data



Credit Approval Problem Revisited

Applicant Information

age	23 years
gender	female
annual salary	NTD 1,000,000
year in residence	1 year
year in job	0.5 year
current debt	200,000

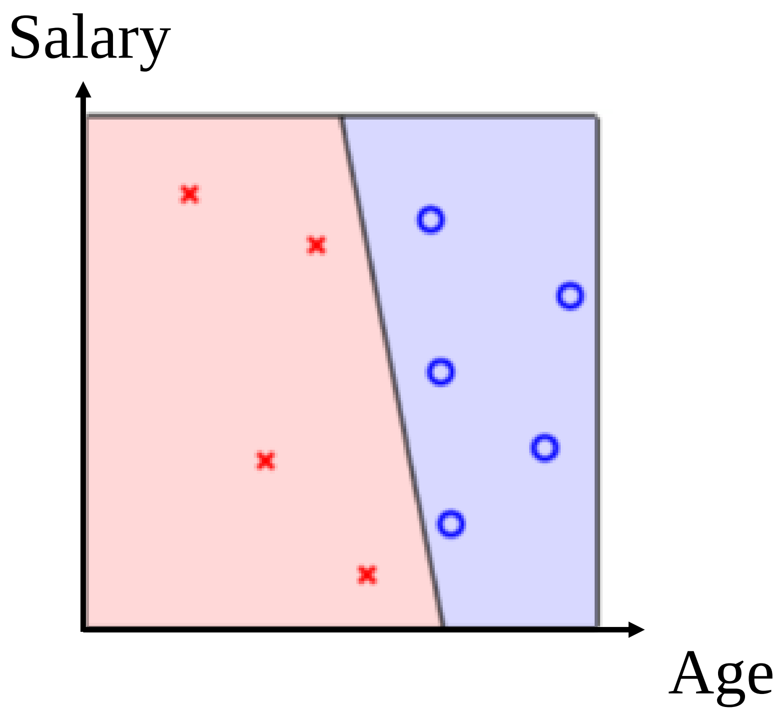


what hypothesis set can we use?

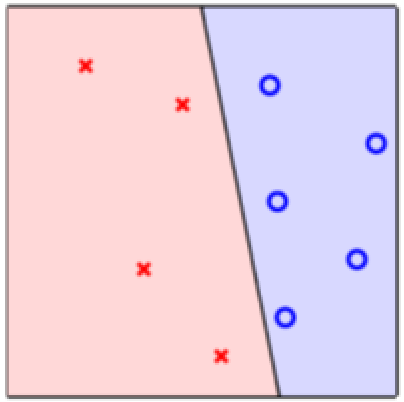
Let’s simplify our data to 2-dimension…

- If we only consider Credit Approval Problem by (age, salary)…

	Age	**Salary**	**Approval**
Customer 1	23	22,000	N
Customer 2	45	75,000	Y
Customer 3	31	60,000	Y
:	:	:	:
Customer n	26	25,000	N



Perceptron (感知器)



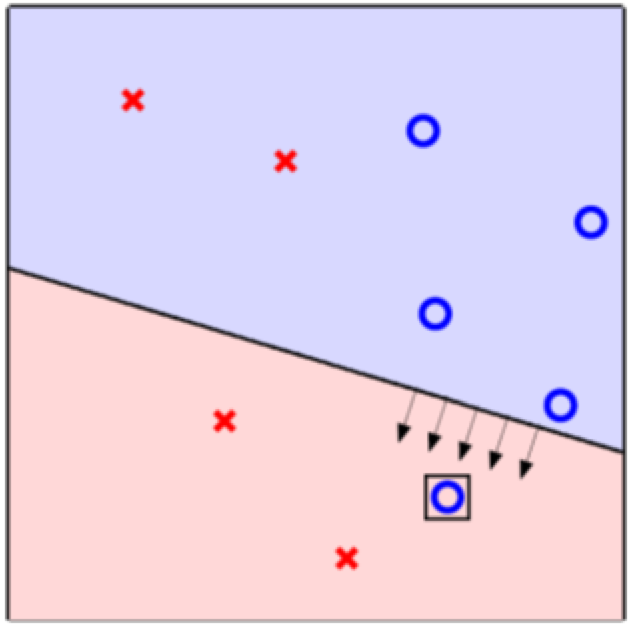
- customer features $\mathbf{x}$: points on the plane (or points in $\mathbb{R}^d$)
- labels y: $\circ$ (+1), $\times$ (-1)
- hypothesis h: **lines** (or hyperplanes in $\mathbb{R}^d$)
  —positive on one side of a line, negative on the other side
- different line classifies customers differently

perceptrons $\Leftrightarrow$ **linear (binary) classifiers**

如何選出正確的Perceptron?

$\mathcal{H}$ = all possible perceptrons, $g = ?$

- want: $g \approx f$ (hard when f unknown)
- almost necessary: $g \approx f$ on $\mathcal{D}$, ideally $g(\mathbf{x}_n) = f(\mathbf{x}_n) = y_n$
- difficult: $\mathcal{H}$ is of **infinite** size
- idea: start from some g_0, and 'correct' its mistakes on $\mathcal{D}$

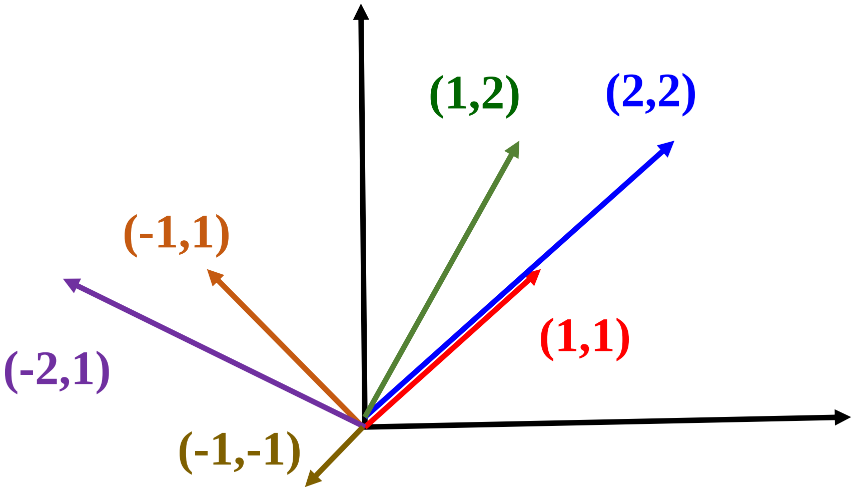


will represent g_0 by its weight vector $\mathbf{w}_0$

Recall line function and some properties…

- The line function in 2d space: $ax + by + c = 0$
  - (a, b) 為直線的法向量

- Property of Inner Product
  - 兩向量方向完全相同，向量的cos角度為1
  - 兩向量方向完全相反，向量的cos角度為-1
  - 兩向量方向垂直，向量的cos角度為0
  - 兩向量夾角差異小於90度時為正、大於90度為負、等於九十度為0



設 $\vec{v}_1 = (x_1, y_1), \vec{v}_2 = (x_2, y_2)$，且 $\vec{v}_1, \vec{v}_2$ 的夾角為 θ，

則 $\cos\theta = \dfrac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1||\vec{v}_2|} = \dfrac{x_1x_2 + y_1y_2}{\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2}}$ 。

符號定義

- 原本的二維資料 (x, y)，重新寫成 (x_1, x_2)
- 原本的直線方程式 $ax+by+c=0$，重新寫成 $w_0+w_1x_1+w_2x_2=0$

- 直線方程式可以視為(w_0, w_1, w_2) 與 $(1, x_1, x_2)$ 的內積=0。
  - $w_0+w_1x_1+w_2x_2 = (w_0, w_1, w_2) \cdot (1, x_1, x_2) = w \cdot x = 0$

- 平面上的點落在直線右邊，$w \cdot x > 0$；否則 $w \cdot x < 0$
  - 分類器可以用 **sign($w \cdot x$)** 表示
  - 資料以 x 表示、資料的label以 y 表示

Perceptron Learning Algorithm

start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

1. find a mistake of $\mathbf{w}_t$ called $\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$

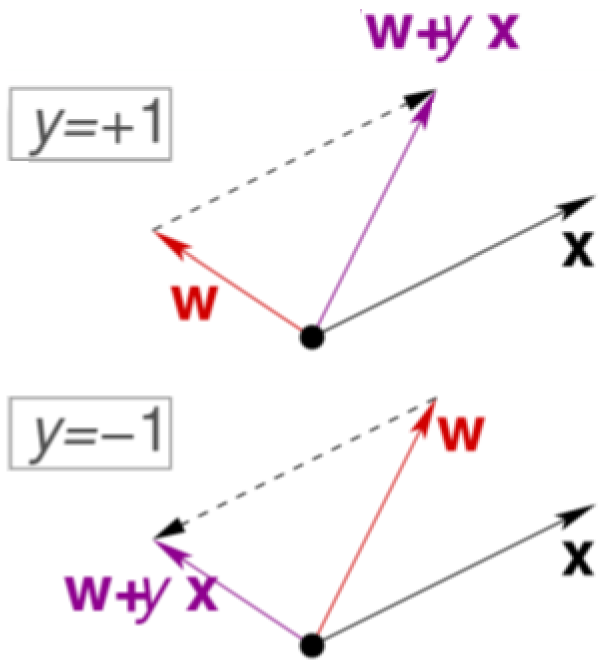
$$\mathrm{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

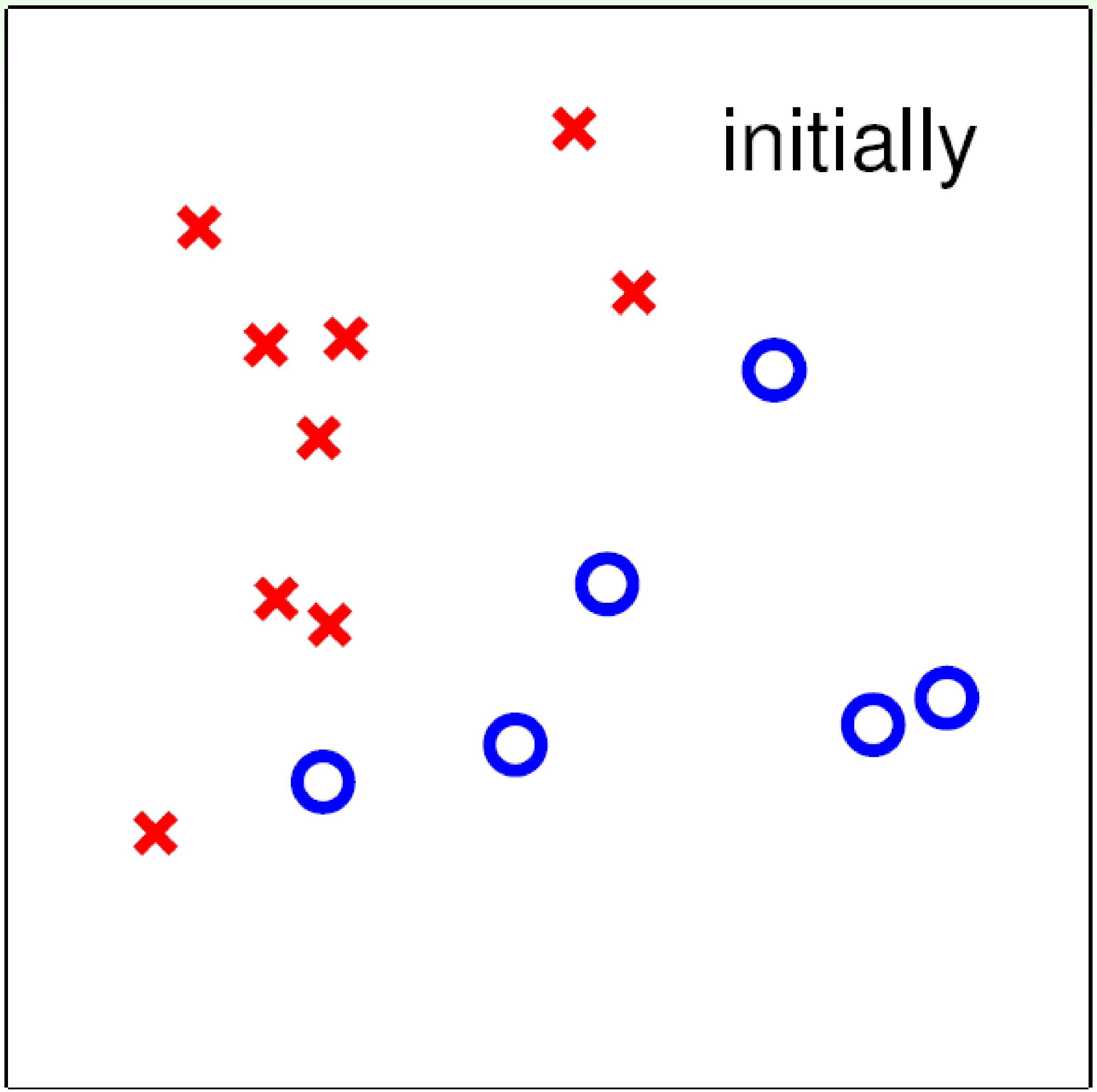
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}$$

. . . until no more mistakes

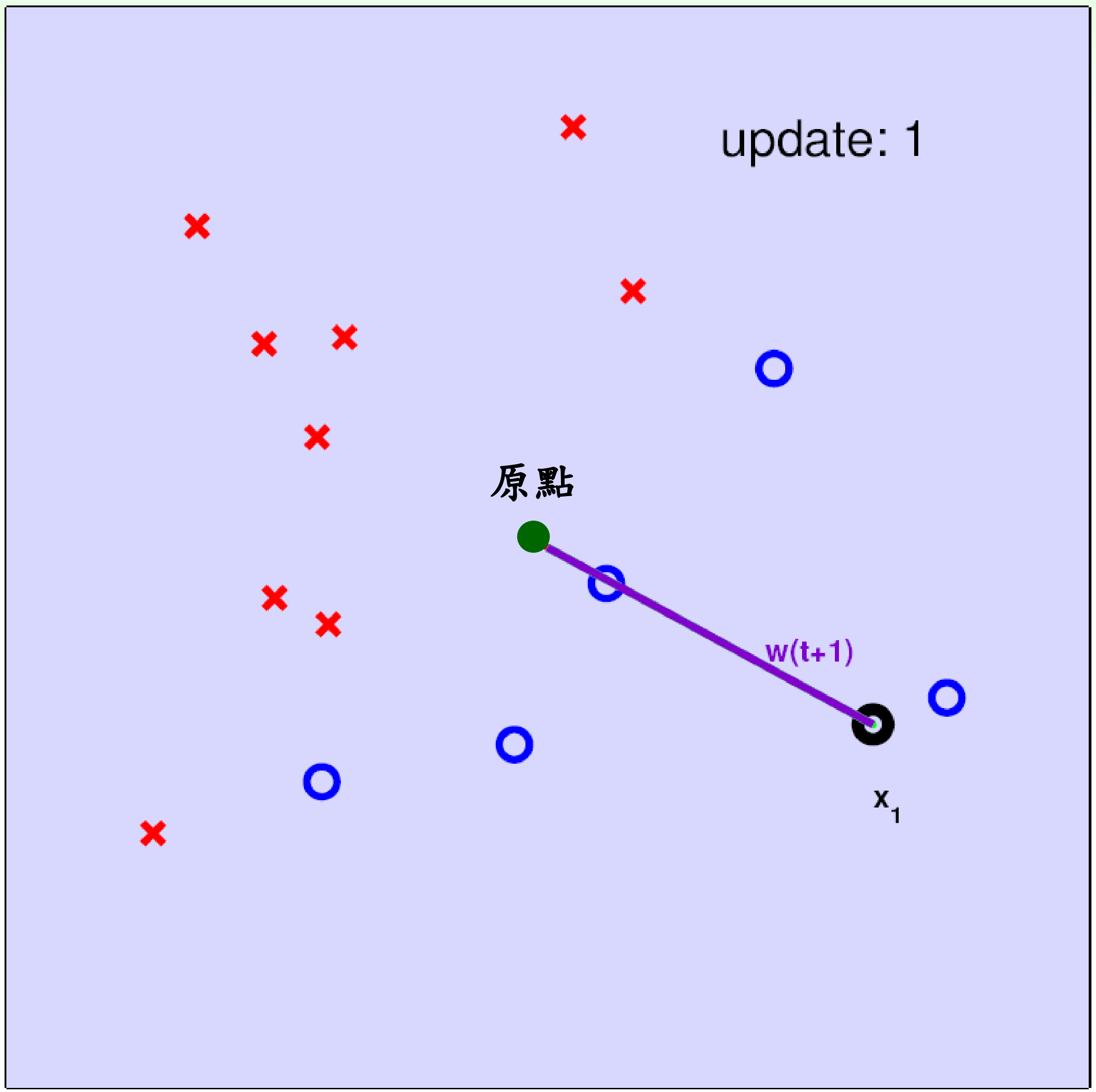
return last $\mathbf{w}$ (called $\mathbf{w}_{\mathrm{PLA}}$) as g



Seeing is Believing

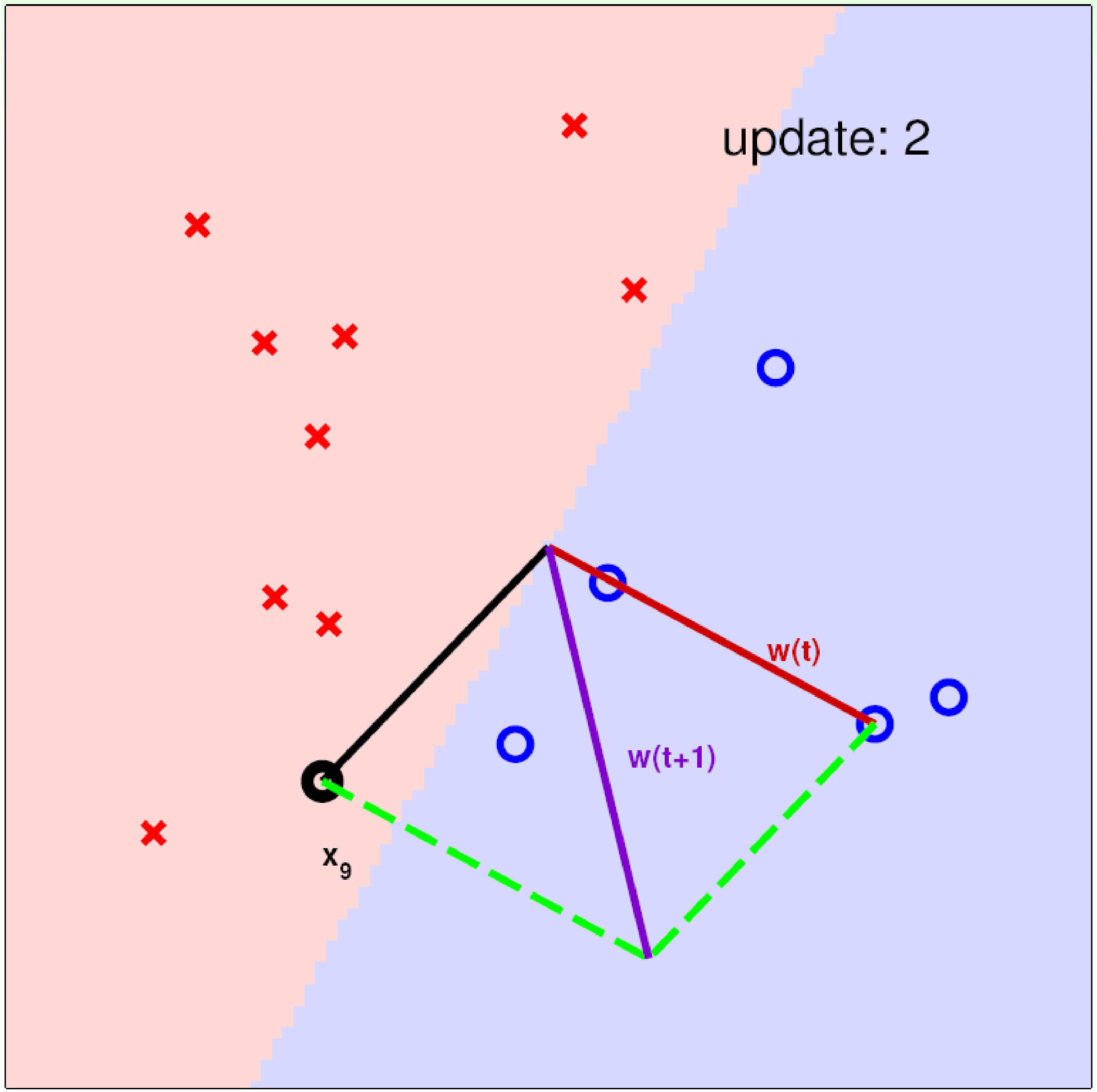


Seeing is Believing

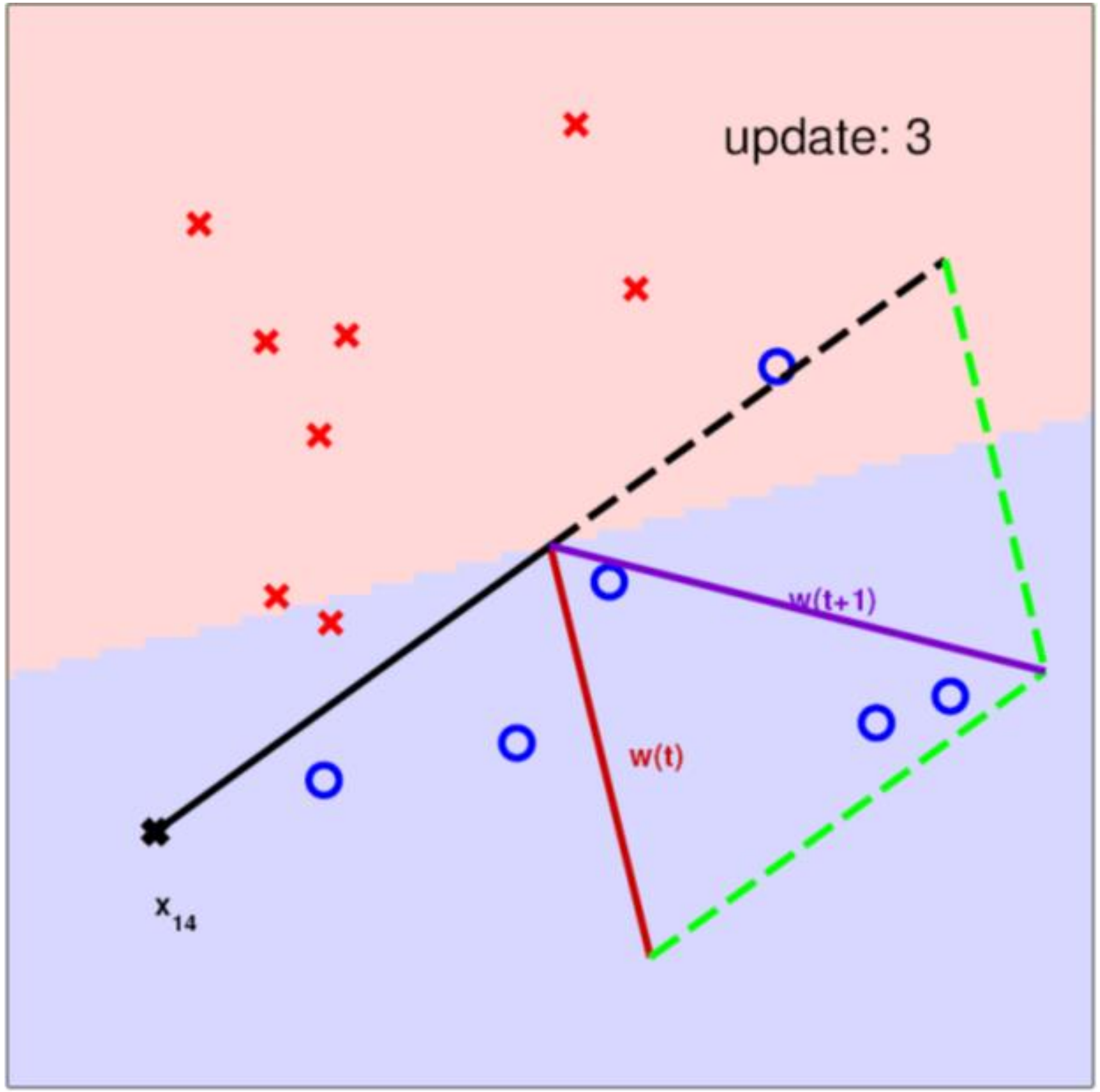


一開始沒有任何線，任取一點當錯誤點

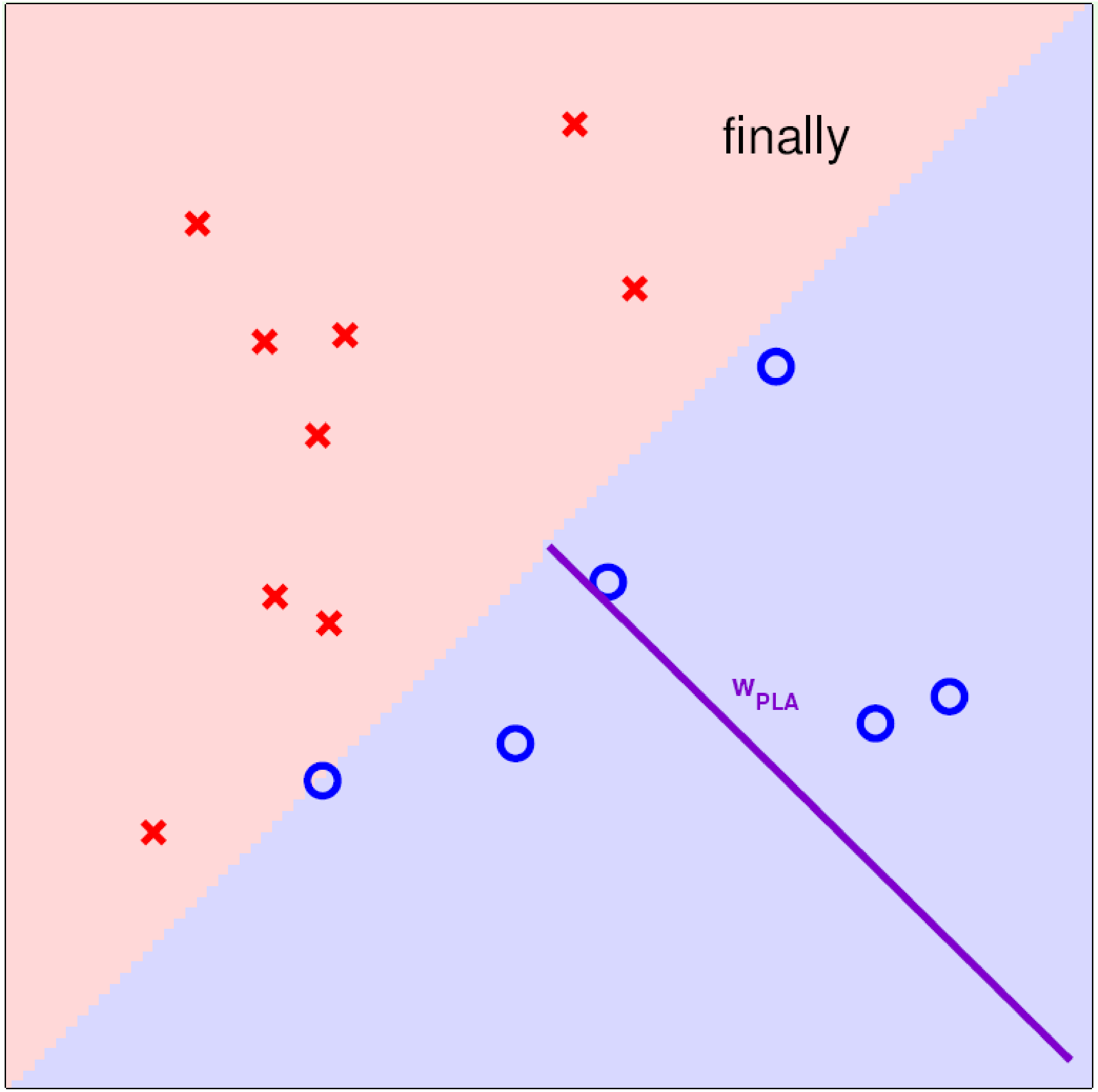
Seeing is Believing



Seeing is Believing



Seeing is Believing



Practical Implementation of PLA

start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

Cyclic PLA

For $t = 0, 1, \ldots$

1. find **the next** mistake of $\mathbf{w}_t$ called $\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$

$$\operatorname{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}$$

$\ldots$ until **a full cycle of not encountering mistakes**

next can follow naïve cycle $(1, \cdots, N)$
or precomputed random cycle

Fun Time

Let's try to think about why PLA may work.

Let $n = n(t)$, according to the rule of PLA below, which formula is true?

$$\operatorname{sign}\left(\mathbf{w}_t^T \mathbf{x}_n\right) \neq y_n, \quad \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_n \mathbf{x}_n$$

1. $\mathbf{w}_{t+1}^T \mathbf{x}_n = y_n$
2. $\operatorname{sign}(\mathbf{w}_{t+1}^T \mathbf{x}_n) = y_n$
3. $y_n \mathbf{w}_{t+1}^T \mathbf{x}_n \geq y_n \mathbf{w}_t^T \mathbf{x}_n$
4. $y_n \mathbf{w}_{t+1}^T \mathbf{x}_n < y_n \mathbf{w}_t^T \mathbf{x}_n$

Reference Answer: 3

Simply multiply the second part of the rule by $y_n \mathbf{x}_n$. The result shows that **the rule somewhat 'tries to correct the mistake.'**

How About High-dimensional Data?

age	23 years
annual salary	NTD 1,000,000
year in job	0.5 year
current debt	200,000

- For $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ '**features of customer**', compute a weighted 'score' and

$$\text{approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold}$$

$$\text{deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}$$

- $\mathcal{Y}$: $\{+1(\textbf{good}), -1(\textbf{bad})\}$, 0 ignored—linear formula $h \in \mathcal{H}$ are

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

Vector Form of Perceptron Hypothesis

$$\begin{aligned}
h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\
&= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\
&= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) \\
&= \text{sign}\left(\mathbf{w}^T \mathbf{x}\right)
\end{aligned}$$

- each ‘tall’ **w** represents a hypothesis *h* & is multiplied with ‘tall’ **x** —**will use tall versions to simplify notation**

Fun Time

- Consider using a perceptron to detect spam messages.
- Assume that each email is represented by the frequency of keyword occurrence, and output +1 indicates a spam. Which keywords below shall have large positive weights in a **good perceptron** for the task?

1. coffee, tea, hamburger, steak
2. free, drug, fantastic, deal
3. machine, learning, statistics, textbook
4. national, Taiwan, university, courser

Reference Answer: 2

The occurrence of keywords with positive weights increase the 'spam score', and hence those keywords should often appear in spams.

Some Remaining Issues of PLA

'correct' mistakes on $\mathcal{D}$ **until no mistakes**

Algorithmic: halt (with no mistake)?

- naïve cyclic: ??
- random cyclic: ??
- other variant: ??

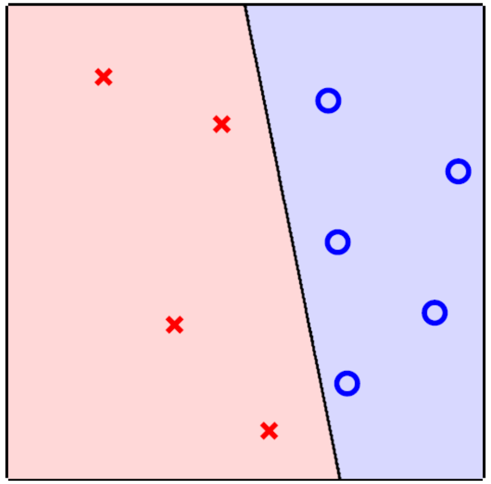
Learning: $g \approx f$?

- on $\mathcal{D}$, if halt, yes (no mistake)
- outside $\mathcal{D}$: ??
- if not halting: ??

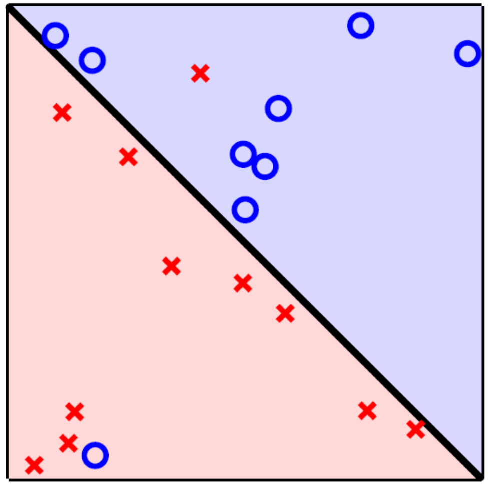
[to be shown] if (...), after 'enough' corrections,
any PLA variant halts

Linear Separability

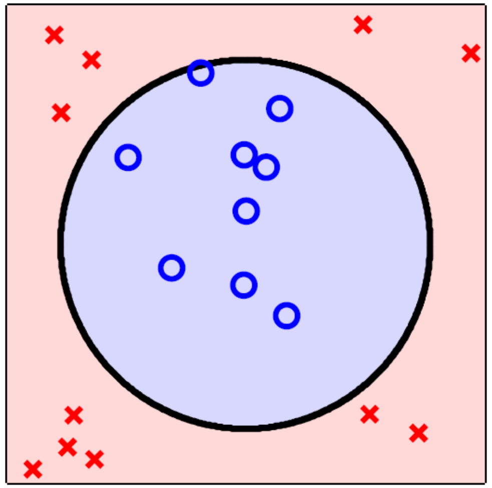
- **if** PLA halts (i.e. no more mistakes),
  (necessary condition) $\mathcal{D}$ allows some $\mathbf{w}$ to make no mistake
- call such $\mathcal{D}$ **linear separable**



(linear separable)



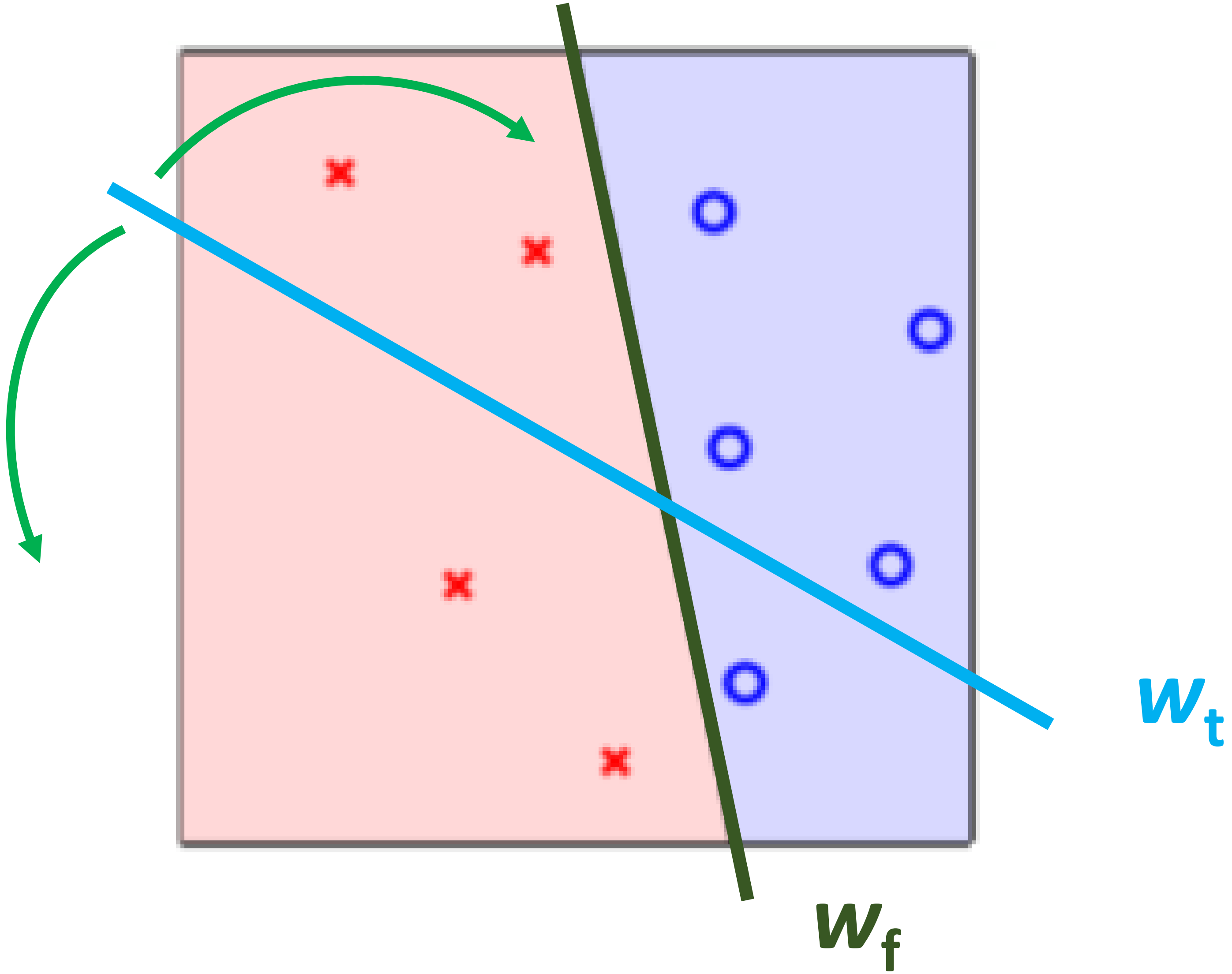
(not linear separable)



(not linear separable)

assume linear separable $\mathcal{D}$,
does PLA always **halt**?

PLA找出的解，真的越來越好嗎?



PLA Fact: $\mathbf{w}_t$ Gets More Aligned with $\mathbf{w}_f$

linear separable $\mathcal{D} \Leftrightarrow$ **exists perfect $\mathbf{w}_f$ such that** $y_n = \text{sign}(\mathbf{w}_f^T \mathbf{x}_n)$

- $\mathbf{w}_f$ perfect hence every $\mathbf{x}_n$ correctly away from line:

$$y_{n(t)} \mathbf{w}_f^T \mathbf{x}_{n(t)} \geq \min_n y_n \mathbf{w}_f^T \mathbf{x}_n > 0$$

- $\mathbf{w}_f^T \mathbf{w}_t \uparrow$ by updating with any $(\mathbf{x}_{n(t)}, y_{n(t)})$

$$\begin{aligned} \mathbf{w}_f^T \mathbf{w}_{t+1} &= \mathbf{w}_f^T \left(\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}\right) \\ &\geq \mathbf{w}_f^T \mathbf{w}_t + \min_n y_n \mathbf{w}_f^T \mathbf{x}_n \\ &> \mathbf{w}_f^T \mathbf{w}_t + 0. \end{aligned}$$

$\mathbf{w}_t$ appears more aligned with $\mathbf{w}_f$ after update
(really?)

$$\vec{v}_1 \cdot \vec{v}_2 = |\vec{v}_1| |\vec{v}_2| \cos\theta$$

More about PLA

Guarantee

as long as linear separable and correct by mistake

- inner product of $\mathbf{w}_f$ and $\mathbf{w}_t$ grows fast; length of $\mathbf{w}_t$ grows slowly
- PLA 'lines' are more and more aligned with $\mathbf{w}_f \Rightarrow$ halts

Pros

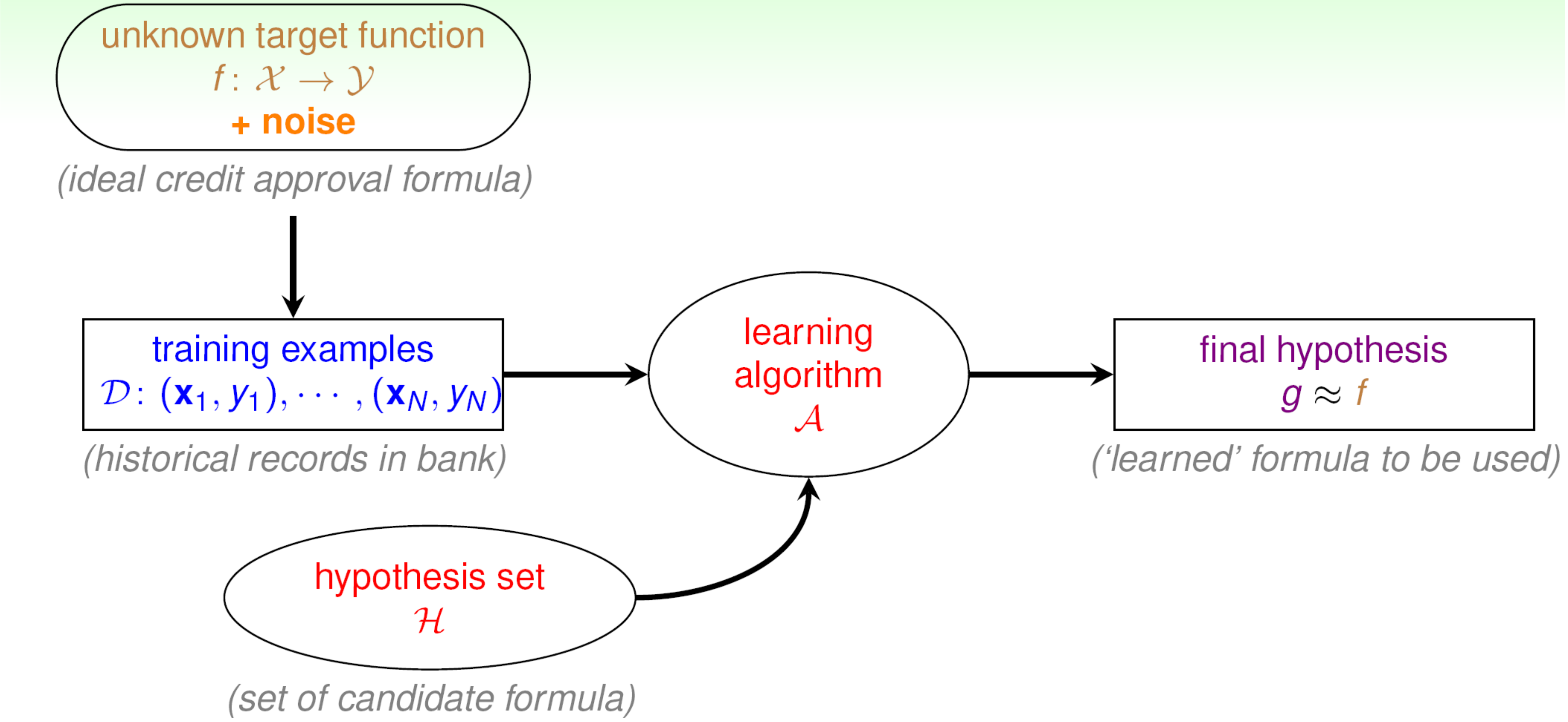
simple to implement, fast, works in any dimension d

Cons

- **'assumes' linear separable** $\mathcal{D}$ to halt
  —property unknown in advance (no need for PLA if we know $\mathbf{w}_f$)
- not fully sure **how long halting takes**
  —though practically fast

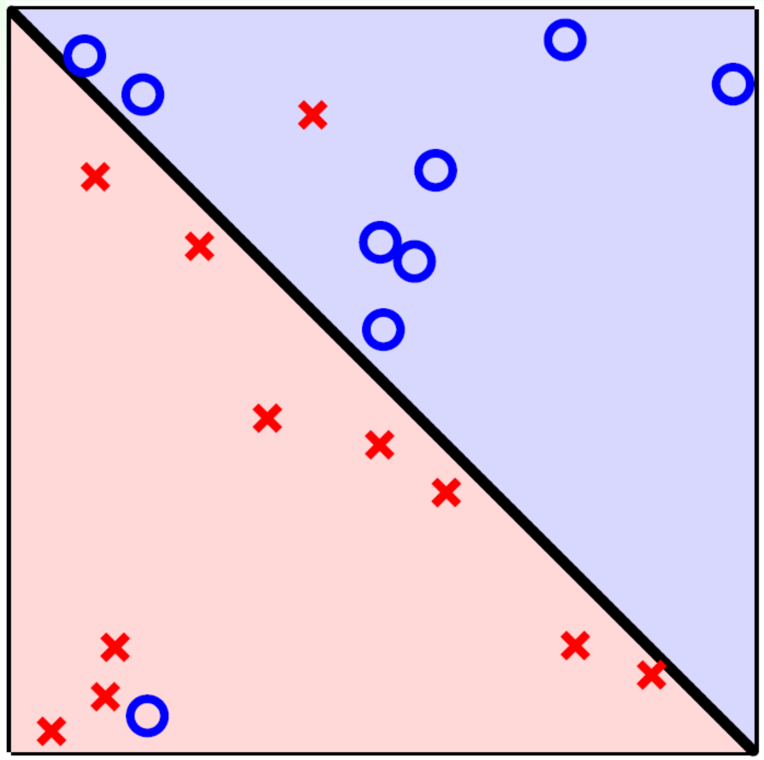
what if $\mathcal{D}$ not linear separable?

Learning with **Noisy Data**



how to at least get $g \approx f$ on **noisy** $\mathcal{D}$?

Line with Noise Tolerance



- assume 'little' noise: $y_n = f(\mathbf{x}_n)$ **usually**
- if so, $g \approx f$ on $\mathcal{D} \Leftrightarrow y_n = g(\mathbf{x}_n)$ **usually**
- how about

$$\mathbf{w}_g \leftarrow \underset{\mathbf{w}}{\text{argmin}} \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^T \mathbf{x}_n)]\!]$$

—**NP-hard to solve, unfortunately**

can we modify PLA to get
an 'approximately good' g?

Pocket Algorithm

modify PLA algorithm (black lines) by **keeping best weights in pocket**

initialize pocket weights $\hat{\mathbf{w}}$

For $t = 0, 1, \cdots$

1. find a (random) mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$
2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

3. **if $\mathbf{w}_{t+1}$ makes fewer mistakes than $\hat{\mathbf{w}}$, replace $\hat{\mathbf{w}}$ by $\mathbf{w}_{t+1}$**

...until **enough iterations**

return **$\hat{\mathbf{w}}$ (called $\mathbf{w}_{\text{POCKET}}$) as g**

a simple modification of PLA to find (somewhat) 'best' weights

Fun Time

Should we use pocket or PLA?

Since we do not know whether $\mathcal{D}$ is linear separable in advance, we may decide to just go with pocket instead of PLA. If $\mathcal{D}$ is actually linear separable, what's the difference between the two?

1. pocket on $\mathcal{D}$ is slower than PLA
2. pocket on $\mathcal{D}$ is faster than PLA
3. pocket on $\mathcal{D}$ returns a better g in approximating f than PLA
4. pocket on $\mathcal{D}$ returns a worse g in approximating f than PLA

Reference Answer: 1

Because pocket need to check whether $\mathbf{w}_{t+1}$ is better than $\hat{\mathbf{w}}$ in each iteration, it is slower than PLA. On linear separable $\mathcal{D}$, $\mathbf{w}_{\text{POCKET}}$ is the same as $\mathbf{w}_{\text{PLA}}$, both making no mistakes.

Summary

- Perceptron Hypothesis Set
  hyperplanes/linear classifiers in $\mathbb{R}^d$
- Perceptron Learning Algorithm (PLA)
  correct mistakes and improve iteratively
- Guarantee of PLA
  no mistake eventually if linear separable
- Non-Separable Data
  hold somewhat 'best' weights in pocket

[助教飛飛的重點提示]
感知器到底是什麼?
①數學中二維平面上的一條直線、②高維空間的超平面、③線性二元分類器、④ML世界中最基礎的模型