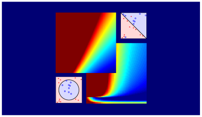
Machine Learning Foundations
(機器學習基石)



Lecture 2: Learning to Answer Yes/No

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. **When** Can Machines Learn?

Lecture 1: The Learning Problem

$\mathcal{A}$ takes $\mathcal{D}$ and $\mathcal{H}$ to get g

Lecture 2: Learning to Answer Yes/No

- Perceptron Hypothesis Set
- Perceptron Learning Algorithm (PLA)
- Guarantee of PLA
- Non-Separable Data

2. Why Can Machines Learn?
3. How Can Machines Learn?
4. How Can Machines Learn Better?

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

unknown target function
$f: \mathcal{X} \rightarrow \mathcal{Y}$

(ideal credit approval formula)

training examples
$\mathcal{D}: (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$

(historical records in bank)

learning algorithm
$\mathcal{A}$

final hypothesis
$g \approx f$

('learned' formula to be used)

hypothesis set
$\mathcal{H}$

(set of candidate formula)

what hypothesis set can we use?

A Simple Hypothesis Set: the 'Perceptron'

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

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

called 'perceptron' hypothesis historically

Vector Form of Perceptron Hypothesis

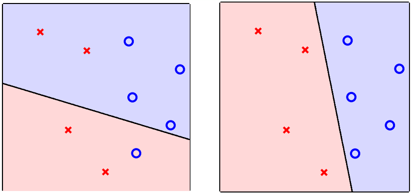
$$
\begin{aligned}
h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\
&= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\
&= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right)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

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

what do perceptrons h 'look like'?

Perceptrons in $\mathbb{R}^2$

$$h(\mathbf{x}) = \text{sign}\left(w_0 + w_1 x_1 + w_2 x_2\right)$$



- 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**

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, coursera

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, coursera

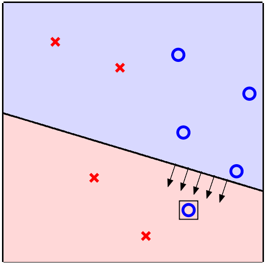
Reference Answer: 2

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

Select g from $\mathcal{H}$

$\mathcal{H}$ = 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 $\mathbf{w}_0$

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 $(\mathbf{x}_{n(t)}, y_{n(t)})$

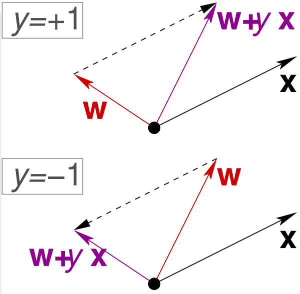
$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

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 $\mathbf{w}_{\text{PLA}}$) as g



That's it!

—*A fault confessed is half redressed.* :-)

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 $\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$

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

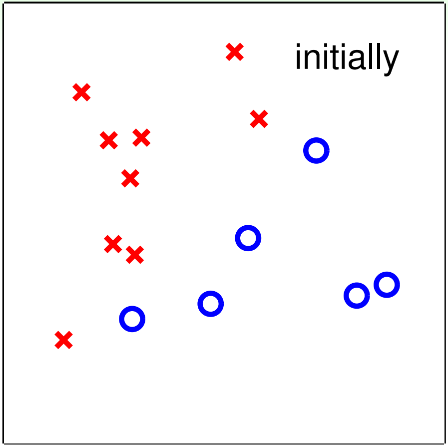
2. correct the mistake by

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

. . . until **a full cycle of not encountering mistakes**

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

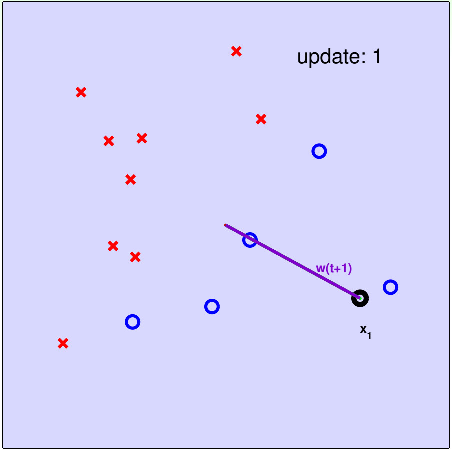
Seeing is Believing



worked like a charm with < 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

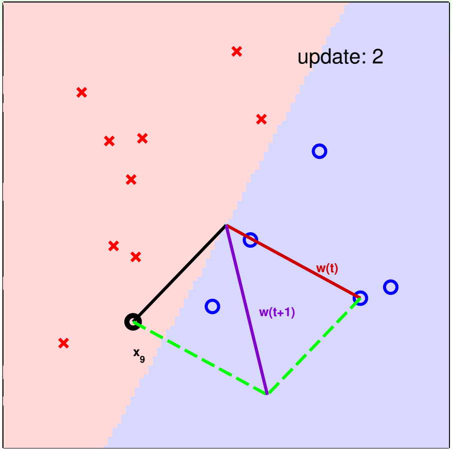
Seeing is Believing



worked like a charm with < 20 lines!!

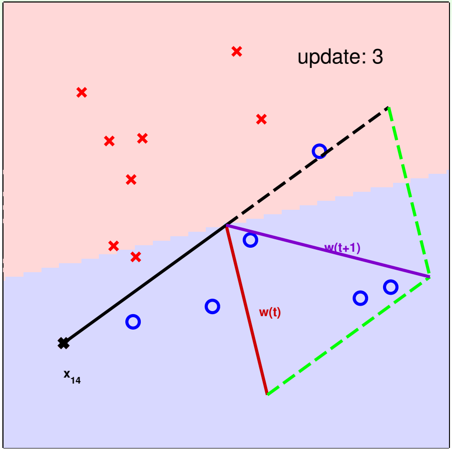
(note: made $x_i \gg x_0 = 1$ for visual purpose)

Seeing is Believing



worked like a charm with $<$ 20 lines!!
(note: made $x_i \gg x_0 = 1$ for visual purpose)

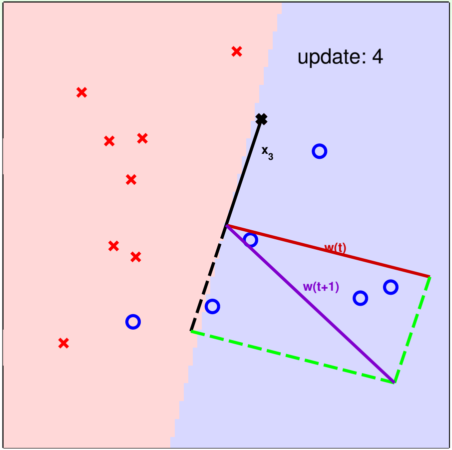
Seeing is Believing



worked like a charm with $<$ 20 lines!!

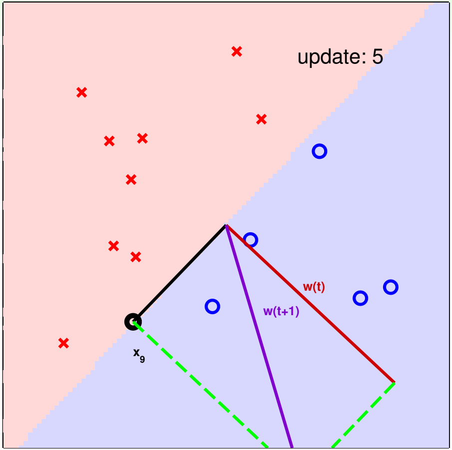
(note: made $x_i \gg x_0 = 1$ for visual purpose)

Seeing is Believing



worked like a charm with <20 lines!!
(note: made $x_i \gg x_0 = 1$ for visual purpose)

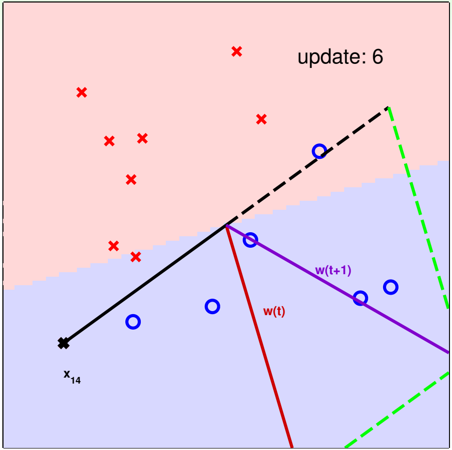
Seeing is Believing



worked like a charm with $<$ 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

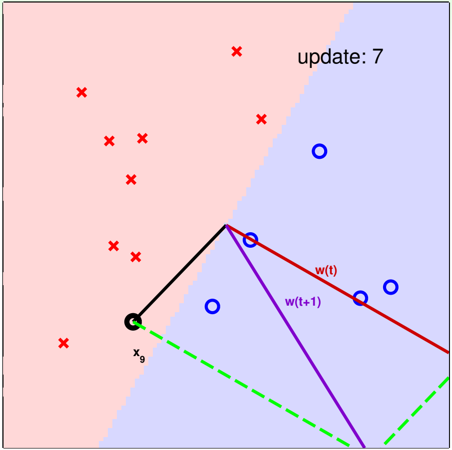
Seeing is Believing



worked like a charm with $<$ 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

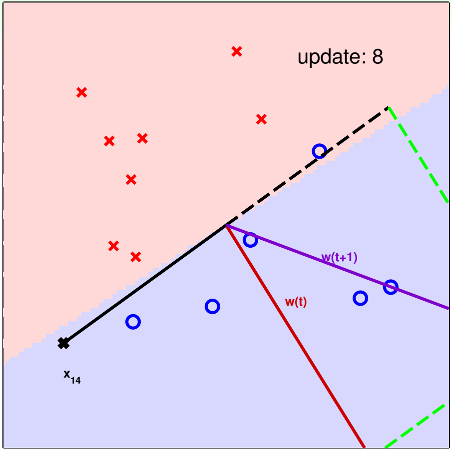
Seeing is Believing



worked like a charm with $<$ 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

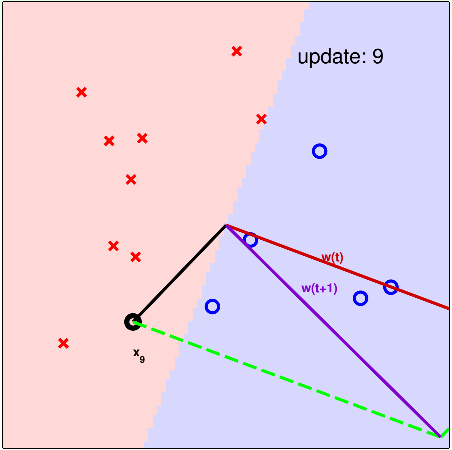
Seeing is Believing



worked like a charm with < 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

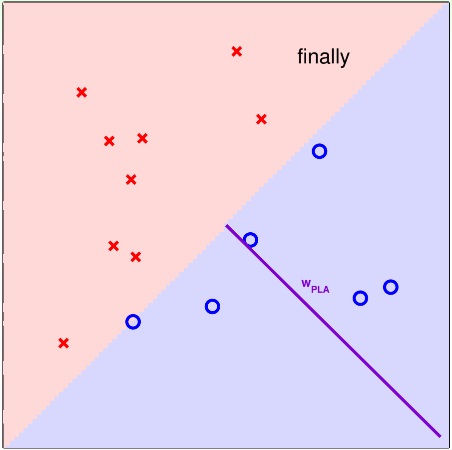
Seeing is Believing



worked like a charm with $<$ 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

Seeing is Believing



worked like a charm with $<$ 20 lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)

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

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?

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

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

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?

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

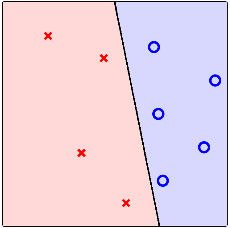
1. $\mathbf{w}_{t+1}^T \mathbf{x}_n = y_n$
2. $\text{sign}(\mathbf{w}_{t+1}^T \mathbf{x}_n) = y_n$
3. $y_n \mathbf{w}_{t+1}^T \mathbf{x}_n \geq y_n \mathbf{w}_t^T \mathbf{x}_n$
4. $y_n \mathbf{w}_{t+1}^T \mathbf{x}_n < y_n \mathbf{w}_t^T \mathbf{x}_n$

Reference Answer: 3

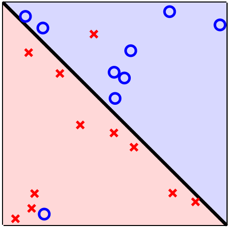
Simply multiply the second part of the rule by $y_n \mathbf{x}_n$. The result shows that **the rule somewhat 'tries to correct the mistake.'**

Linear Separability

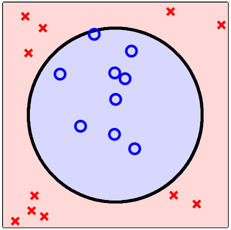
- **if** PLA halts (i.e. no more mistakes),
  (necessary condition) $\mathcal{D}$ allows some $\mathbf{w}$ to make no mistake
- call such $\mathcal{D}$ **linear separable**



(linear separable)



(not linear separable)



(not linear separable)

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

PLA 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 $\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$

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

$\mathbf{w}_t$ appears more aligned with $\mathbf{w}_f$ after update

(really?)

PLA Fact: $\mathbf{w}_t$ Does Not Grow Too Fast

$\mathbf{w}_t$ changed only when mistake

$$\Leftrightarrow \text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)} \Leftrightarrow y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} \leq 0$$

- mistake 'limits' $\|\mathbf{w}_t\|^2$ growth, even when updating with 'longest' $\mathbf{x}_n$

$$\begin{aligned}
\|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}\|^2 \\
&= \|\mathbf{w}_t\|^2 + 2y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)} + \|y_{n(t)}\mathbf{x}_{n(t)}\|^2 \\
&\leq \|\mathbf{w}_t\|^2 + 0 + \|y_{n(t)}\mathbf{x}_{n(t)}\|^2 \\
&\leq \|\mathbf{w}_t\|^2 + \max_n \|y_n\mathbf{x}_n\|^2
\end{aligned}$$

start from $\mathbf{w}_0 = \mathbf{0}$, after T mistake corrections,

$$\frac{\mathbf{w}_f^T}{\|\mathbf{w}_f\|} \frac{\mathbf{w}_T}{\|\mathbf{w}_T\|} \geq \sqrt{T} \cdot \text{constant}$$

Fun Time

Let's upper-bound T, the number of mistakes that PLA 'corrects'.

$$\text{Define } R^2 = \max_n \|\mathbf{x}_n\|^2 \quad \rho = \min_n y_n \frac{\mathbf{w}_f^T}{\|\mathbf{w}_f\|}\mathbf{x}_n$$

We want to show that $T \leq \square$. Express the upper bound $\square$ by the two terms above.

1. R/ρ
2. R^2/ρ^2
3. R/ρ^2
4. ρ^2/R^2

Fun Time

Let's upper-bound T, the number of mistakes that PLA 'corrects'.

$$\text{Define } R^2 = \max_n \|\mathbf{x}_n\|^2 \quad \rho = \min_n y_n \frac{\mathbf{w}_f^T}{\|\mathbf{w}_f\|}\mathbf{x}_n$$

We want to show that $T \leq \square$. Express the upper bound $\square$ by the two terms above.

1. R/ρ
2. R^2/ρ^2
3. R/ρ^2
4. ρ^2/R^2

Reference Answer: 2

The maximum value of $\frac{\mathbf{w}_f^T}{\|\mathbf{w}_f\|}\frac{\mathbf{w}_t}{\|\mathbf{w}_t\|}$ is 1. Since T mistake corrections **increase the inner product by $\sqrt{T} \cdot$ constant**, the maximum number of corrected mistakes is $1/\text{constant}^2$.

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

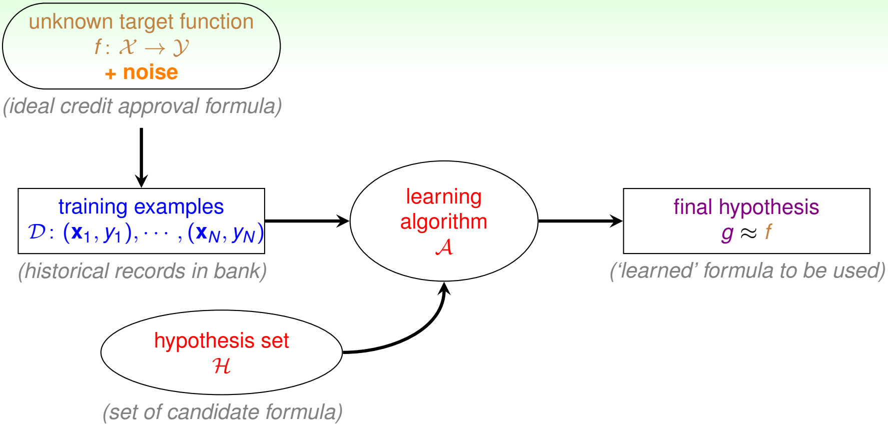
simple to implement, fast, works in any dimension d

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** (ρ depends on $\mathbf{w}_f$)
  —though practically fast

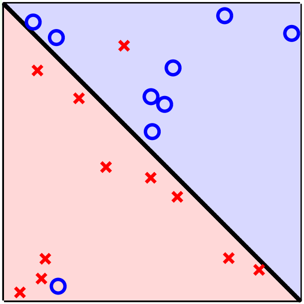
what if $\mathcal{D}$ not linear separable?

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

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

—**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

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

1. **When** Can Machines Learn?

Lecture 1: The Learning Problem

Lecture 2: Learning to Answer Yes/No

- 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

- **next: the zoo of learning problems**

2. Why Can Machines Learn?
3. How Can Machines Learn?
4. How Can Machines Learn Better?