Machine Learning
（机器学习）

Shuyuan Yang, Zhixi Feng

E-mail： syyang@xidianedu.cn

zxfeng@xidian.edu.cn

Review

- Lecture 1: Introduction of Machine Learning
  - Course Introduction
  - What is Machine Learning
  - Applications of Machine Learning
  - Components of Machine Learning
  - Machine Learning and Other Fields
- Lecture 2: Learning to prediction/classification
  - Perceptron Hypothesis Set
  - Hyperplanes /linear classifiers

Daily Needs: Food, Clothing, Housing, Transportation

data → ML → skill

1. Food (Sadilek et al., 2013)
   - data: Twitter data (words + location)
   - skill: tell food poisoning likeliness of restaurant properly
2. Clothing (Abu-Mostafa, 2012)
   - data: sales figures + client surveys
   - skill: give good fashion recommendations to clients
3. Housing (Tsanas and Xifara, 2012)
   - data: characteristics of buildings and their energy load
   - skill: predict energy load of other buildings closely
4. Transportation (Stallkamp et al., 2012)
   - data: some traffic sign images and meanings
   - skill: recognize traffic signs accurately

ML is everywhere!

Formalize the Learning Problem

Basic Notations

- input: $\mathbf{x} \in \mathcal{X}$ (customer application)
- output: $y \in \mathcal{Y}$ (good/bad after approving credit card)
- unknown pattern to be learned $\Leftrightarrow$ target function:
  $f\colon \mathcal{X} \to \mathcal{Y}$ (ideal credit approval formula)
- data $\Leftrightarrow$ training examples: $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$
  (historical records in bank)
- hypothesis $\Leftrightarrow$ skill with hopefully good performance:
  $g\colon \mathcal{X} \to \mathcal{Y}$ ('learned' formula to be used)

$$\{(\mathbf{x}_n, y_n)\} \text{ from } f \longrightarrow \boxed{\text{ML}} \longrightarrow g$$

Machine Learning and Data Mining

Machine Learning	Data Mining
use data to compute hypothesis g that approximates target f	use **(huge)** data to **find property** that is interesting

- if 'interesting property' **same as** 'hypothesis that approximate target'
  —**ML = DM** (usually what KDDCup does)
- if 'interesting property' **related to** 'hypothesis that approximate target'
  —**DM can help ML, and vice versa** (often, but not always)
- traditional DM also focuses on **efficient computation in large database**

difficult to distinguish ML and DM in reality

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

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

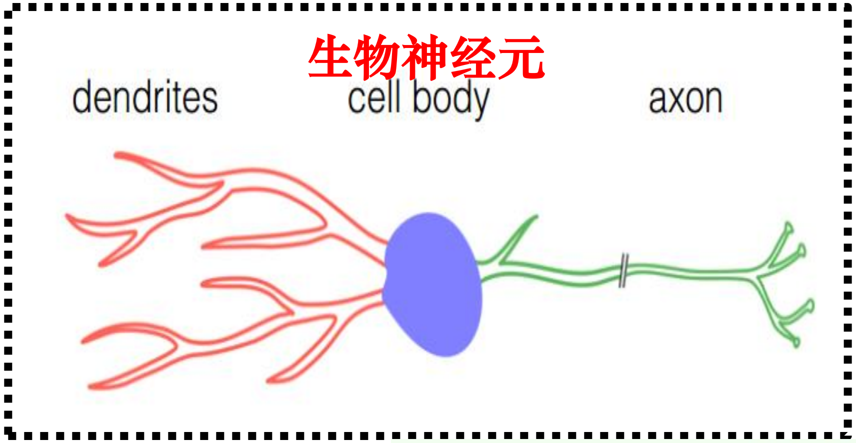
$$\text{deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}$$

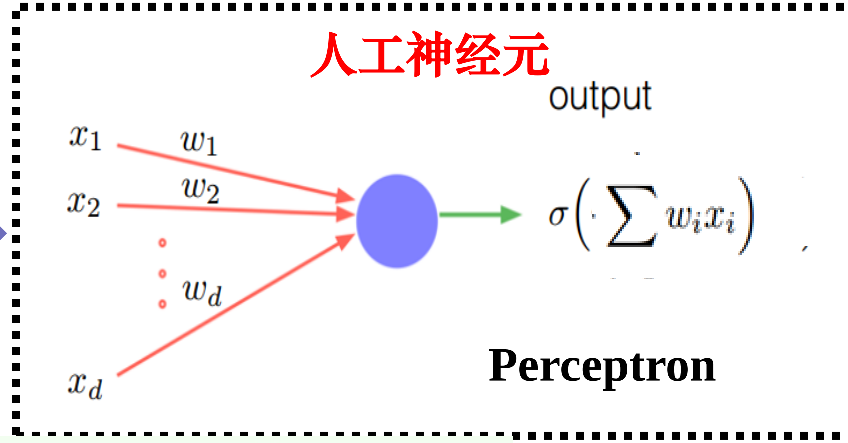
- $\mathcal{Y}$: $\{+1(\textbf{good}), -1(\textbf{bad})\}$, 0 ignored—linear formula $h \in \mathcal{H}$ are

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

called 'perceptron' hypothesis historically

Biologically Inspired Perceptron



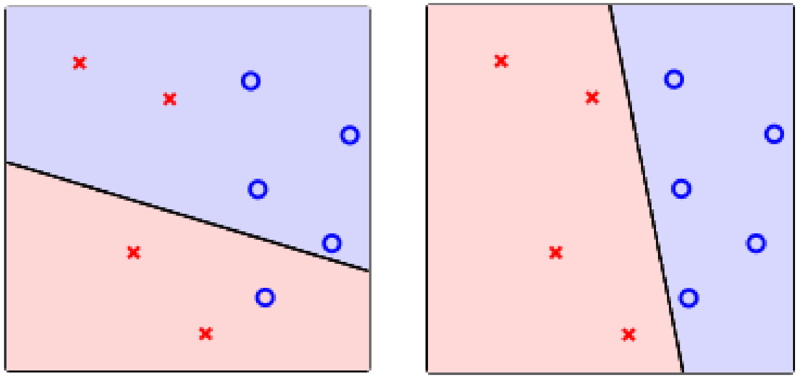


$$
\begin{aligned}
h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\
&= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\
&= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right)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

感知器假设

Perceptron: Binary Classifier

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



- 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**

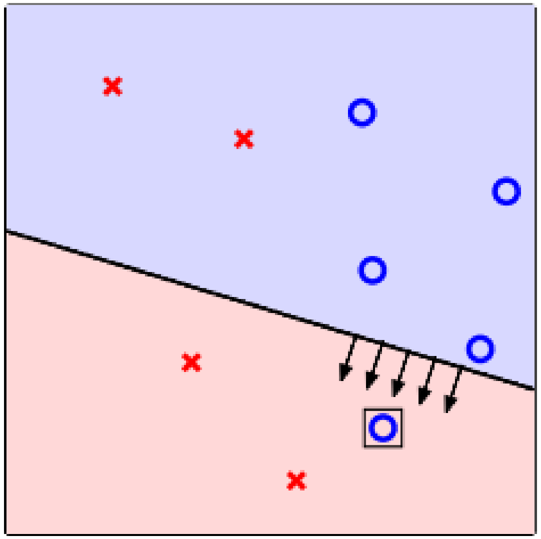
Content

- Lecture 2: Learning to prediction/classification
  - Perceptron Hypothesis Set
  - Hyperplanes /linear classifiers
  - Perceptron Learning Algorithm (PLA)
  - Some Remaining Issues of PLA
  - Discussion: Perceptron vs Bayesian Classifier
  - Guarantee of PLA
  - Non-Separable Data
  - Hold somewhat ‘best’ weights in pocket

Select g from 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: Loss Function

Training samples:

$$D=\{(\mathbf{x}_i, y_i); i=1,\cdots,N\}$$

Empirical risk (error):

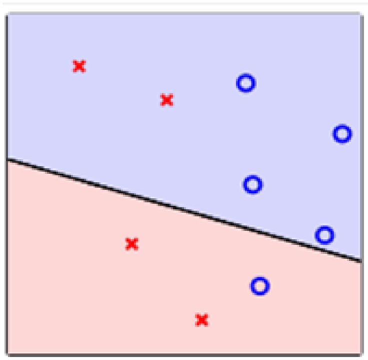
$$\hat{\varepsilon}(h)=\boxed{\frac{1}{N}\sum_{i=1}^{N}1\{h(\mathbf{x}_i)\neq y_i\}}$$

- 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

$$h(\mathbf{x}) = \mathrm{sign}(w_0 + w_1x_1 + w_2x_2)$$

$$\boxed{\min_h \hat{\varepsilon}(h)}$$

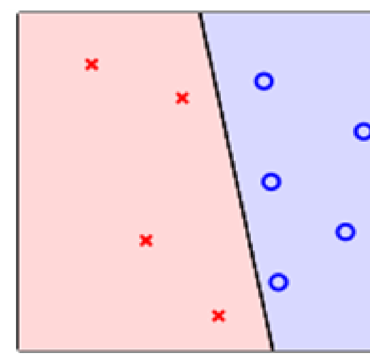
Non differentiable



?

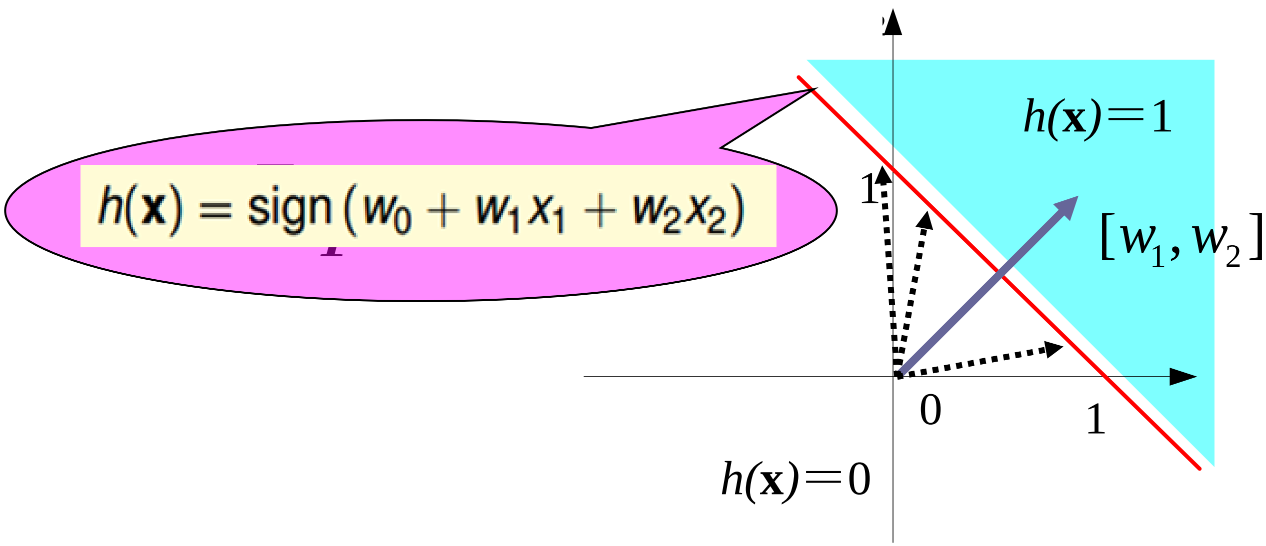
....

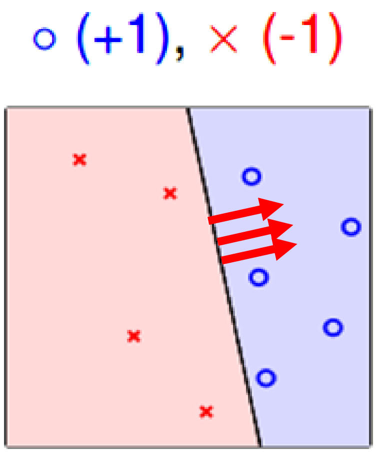
learning algorithm



How to Find the Hypothesis $g \approx f$ on D

*one can **guess** g using the properties of perceptron:*





correct *g*

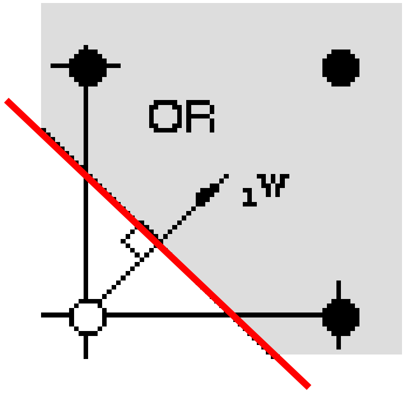
1. Each perceptron has a decision **boundary** $w_0 + w_1 x_1 + w_2 x_2 = 0$.
2. A perceptron can only implement a **binary** classification.
3. **Multiple** perceptron (S) can classify the data into 2^S clusters.

An Example: guess a correct g

*one can **guess** g using the properties of perceptron:*

$$\left\{\mathbf{x}_1 = \begin{bmatrix}0\\0\end{bmatrix}, y_1 = -1\right\} \quad \left\{\mathbf{x}_2 = \begin{bmatrix}0\\1\end{bmatrix}, y_2 = 1\right\} \quad \left\{\mathbf{x}_3 = \begin{bmatrix}1\\0\end{bmatrix}, y_3 = 1\right\} \quad \left\{\mathbf{x}_4 = \begin{bmatrix}1\\1\end{bmatrix}, y_4 = 1\right\}$$

$$\mathbf{w} = \begin{bmatrix}w_1\\w_2\end{bmatrix} = \begin{bmatrix}0.5\\0.5\end{bmatrix} \xleftarrow{\textit{observe}}$$



$$\boxed{w_0 + w_1x_1 + w_2x_2 = 0}$$

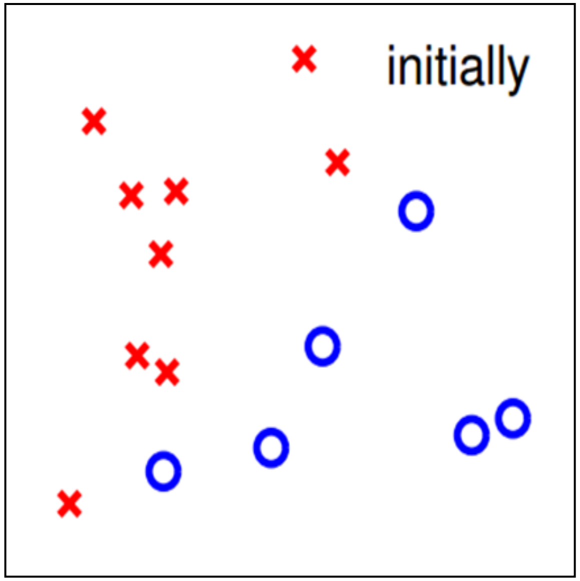
$$\boxed{\mathbf{w}^T\mathbf{x} + w_0 = \begin{bmatrix}0.5 & 0.5\end{bmatrix}\begin{bmatrix}0\\0.5\end{bmatrix} + w_0 = 0.25 + w_0 = 0}$$

$$\Rightarrow w_0 = -0.25$$

Perceptron Learning Algorithm

$\mathcal{H}$ = 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 $\mathcal{D}$

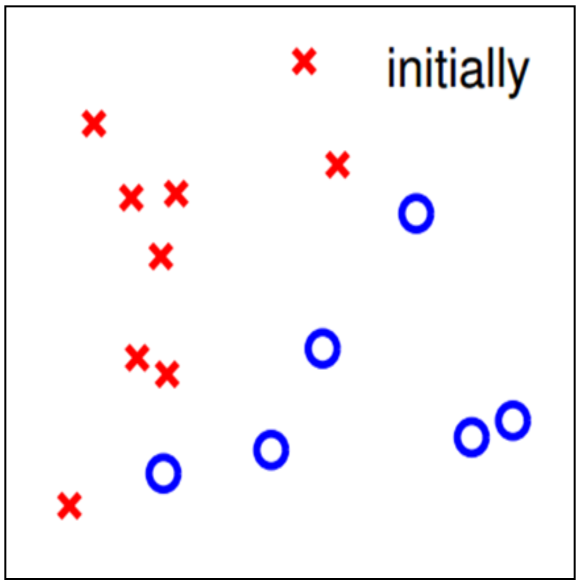


○ (+1), × (-1)

What is the rule ?

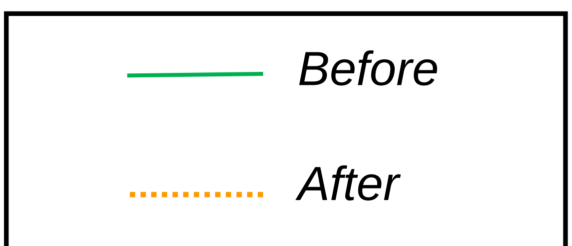
An Extreme Rule

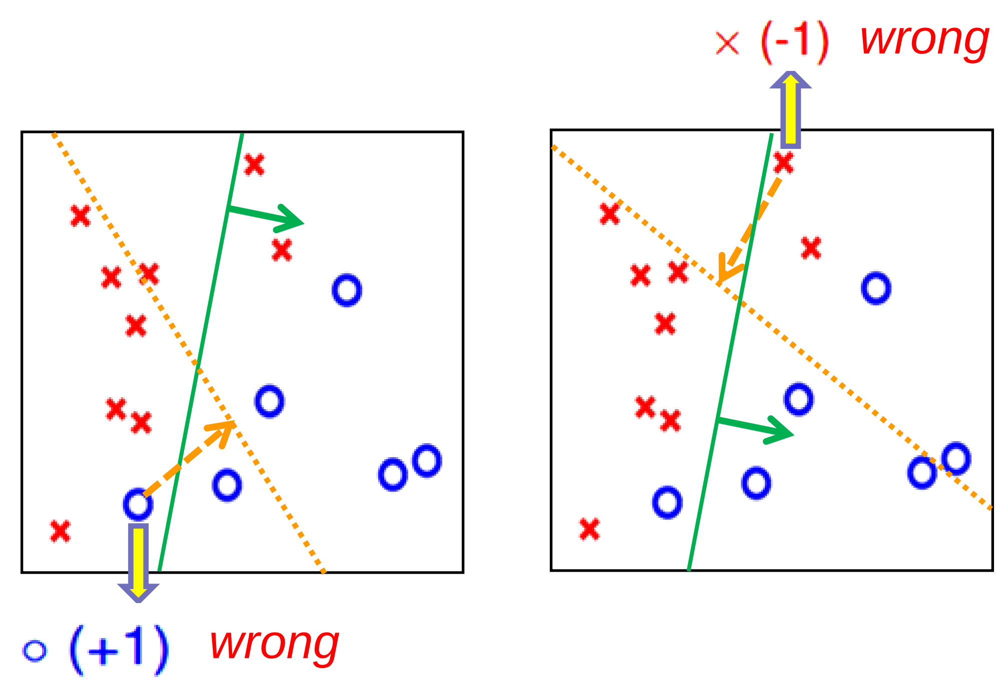
○ (+1), × (-1)



If +1 *has a wrong label-----**w=x***

If -1 *has a wrong label-----**w=-x***





A Compromised Rule: PLA

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

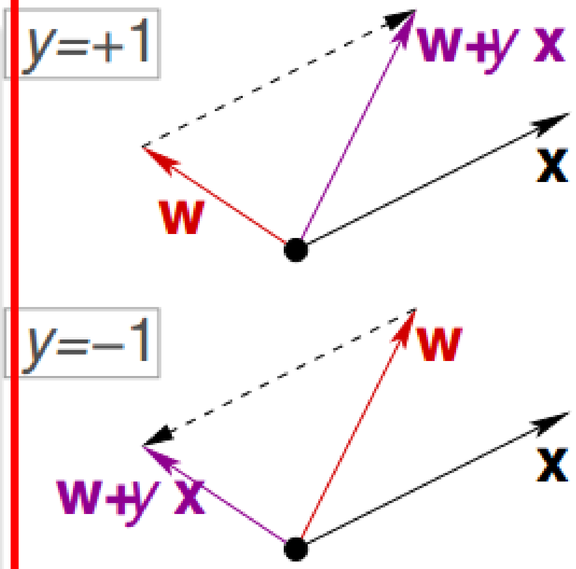
$$\operatorname{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

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

$\ldots$ until no more mistakes

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

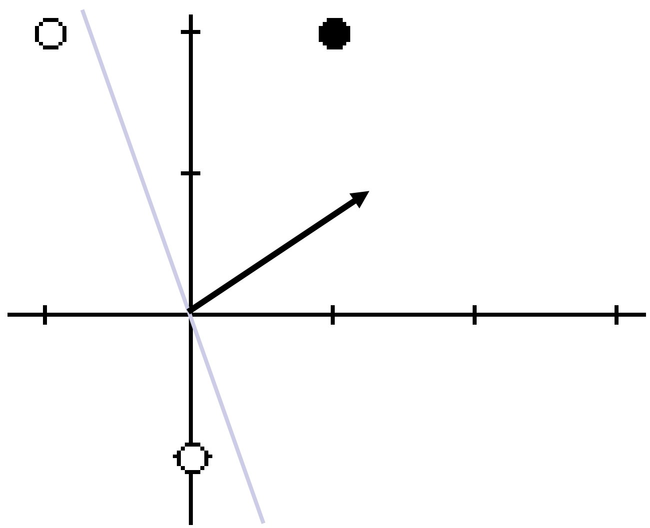


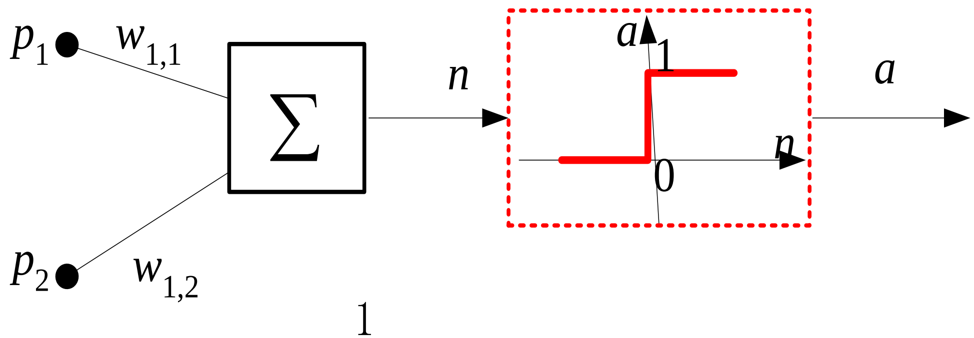
Perceptron Learning Algorithm (PLA)

An Example of PLA

$$\{\mathbf{p}_1, \mathbf{t}_1\}\ ,\ \{\mathbf{p}_2, \mathbf{t}_2\}\ ,\ \ldots,\ \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$



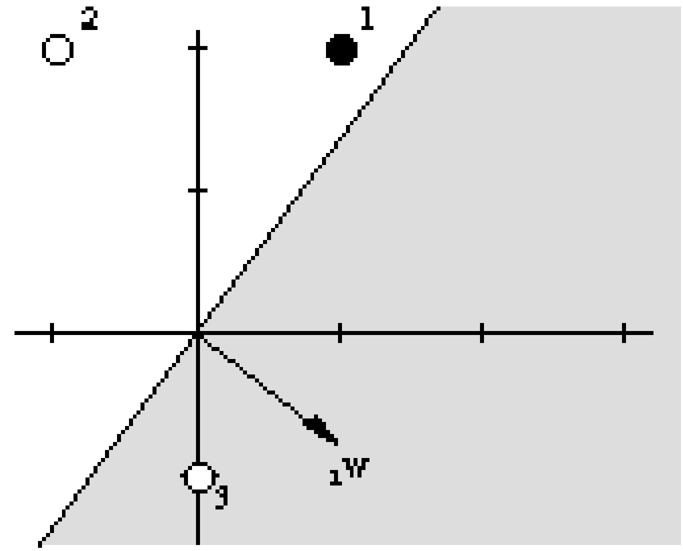


$a = hardlim(n) = hardlim(\mathbf{W}\boldsymbol{p})$

An Example of PLA

Initialization:

$$ {}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} $$



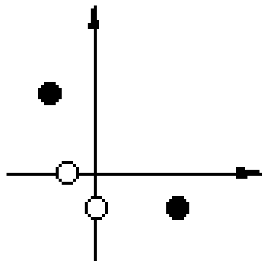
The first input sample:

$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) $$

$$ a = hardlim(-0.6) = 0 $$

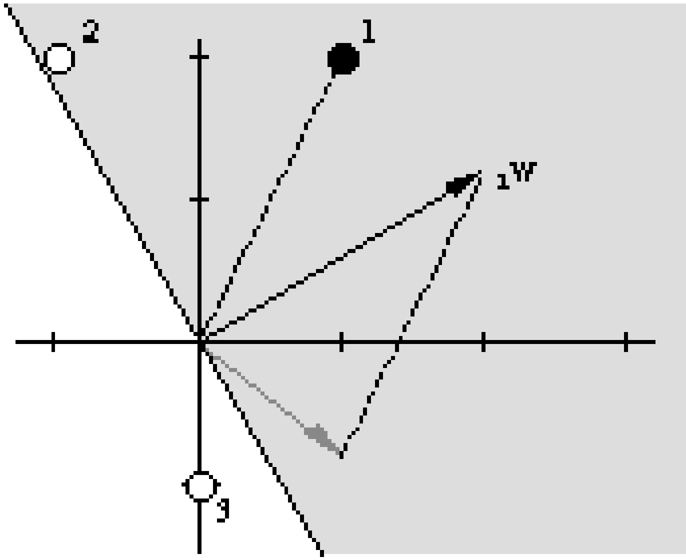
An Example of PLA

- ${}_1\mathbf{w} = \mathbf{p}_1$– , unstable
- Insteadly, add $\mathbf{p}_1$ to ${}_1\mathbf{w}$



Learning rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$



An Example of PLA

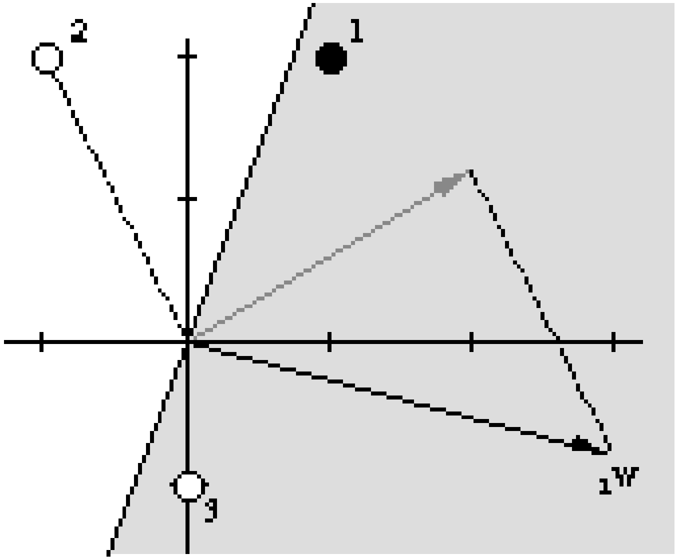
The second input sample:

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2) = hardlim\left(\begin{bmatrix}2.0 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1$$ **(incorrect)**

Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix}2.0\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3.0\\-0.8\end{bmatrix}$$



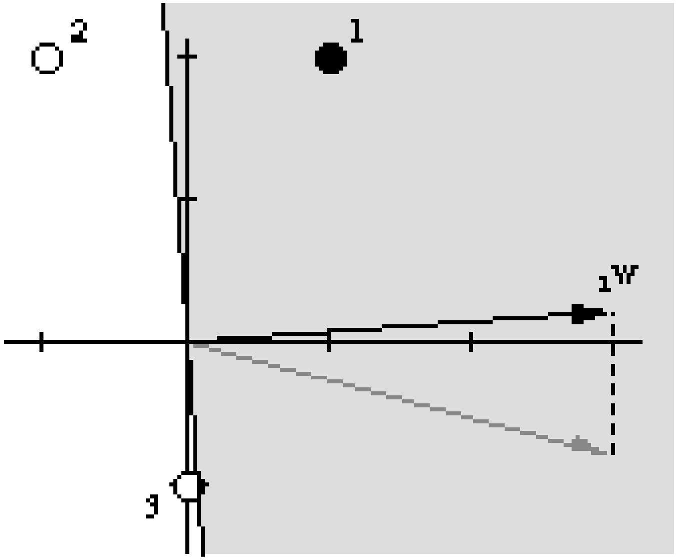
An Example of PLA

The third input sample:

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_3) = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right)$$

$$\mathrm{a} = \mathrm{hardlim}(0.8) = 1$$ **(incorrect)**

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$



Correct

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

An Example of PLA

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\}$$

$$\left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\}$$

$$\left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$

$$W = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$

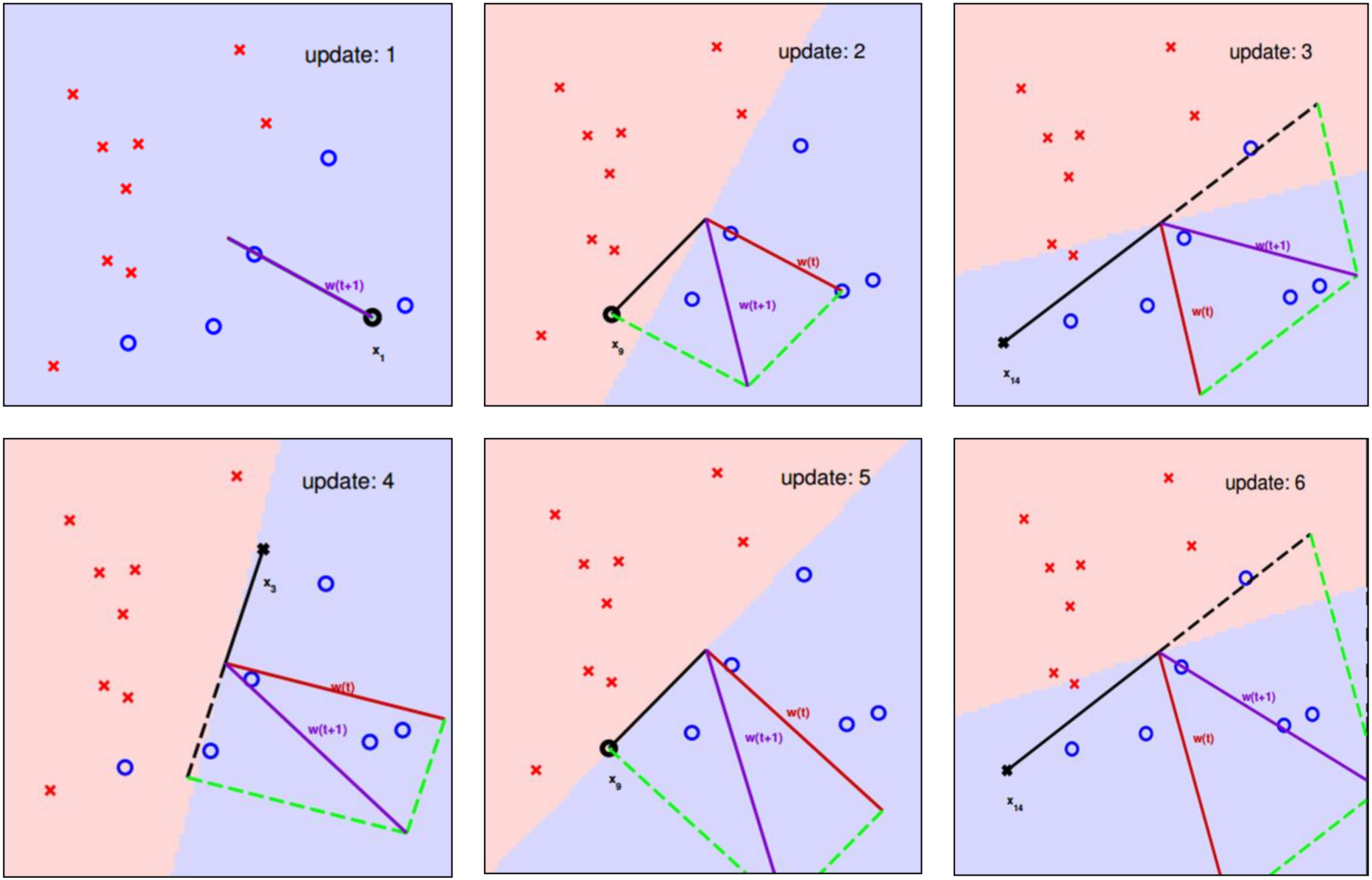
$a = hardlim(\mathbf{W}^{\mathbf{T}}\boldsymbol{p} + b)$

$a = hardlim(\mathbf{3.4}) = \mathbf{1}$

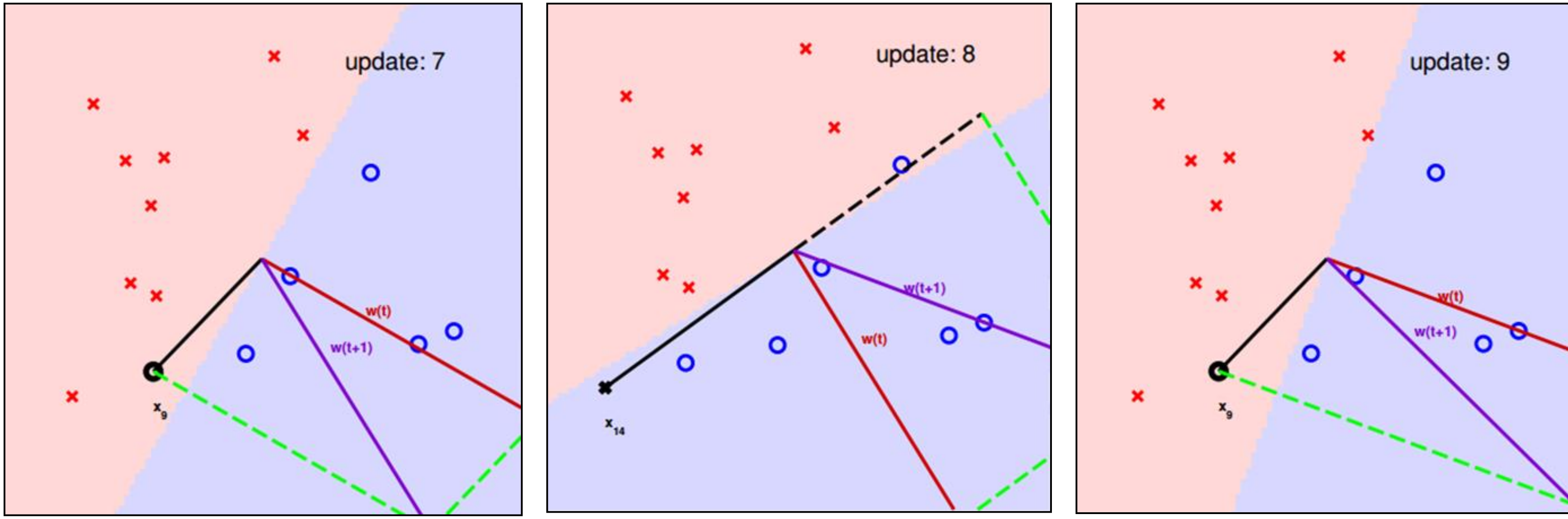
$a = hardlim(\mathbf{-2.6}) = \mathbf{0}$

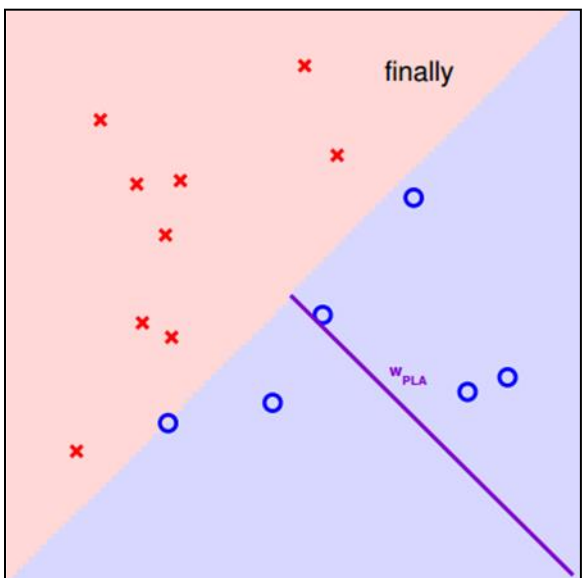
$a = hardlim(\mathbf{-0.2}) = \mathbf{0}$

○ (+1), × (-1)


update: 1
w(t+1)
x_1
update: 2
w(t)
w(t+1)
x_9
update: 3
w(t+1)
w(t)
x_{14}
update: 4
x_3
w(t)
w(t+1)
update: 5
w(t)
w(t+1)
x_9
update: 6
w(t+1)
w(t)
x_{14}

○ (+1), × (-1)





Some Remaining Issues of PLA:

① ***What is the practical implementation of PLA? What is the order of x (naïve, random or others) in PLA?***

② ***Will PLA halt after "enough" number of corrections for all D?***

③ ***Does PLA has guarantee of convergence? (How many number of corrections are need for PLA?***

Q1: 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 $(\mathbf{x}_{n(t)}, y_{n(t)})$

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

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

Q2: Will PLA "halt"?

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

Algorithmic: halt (with no mistake)?

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

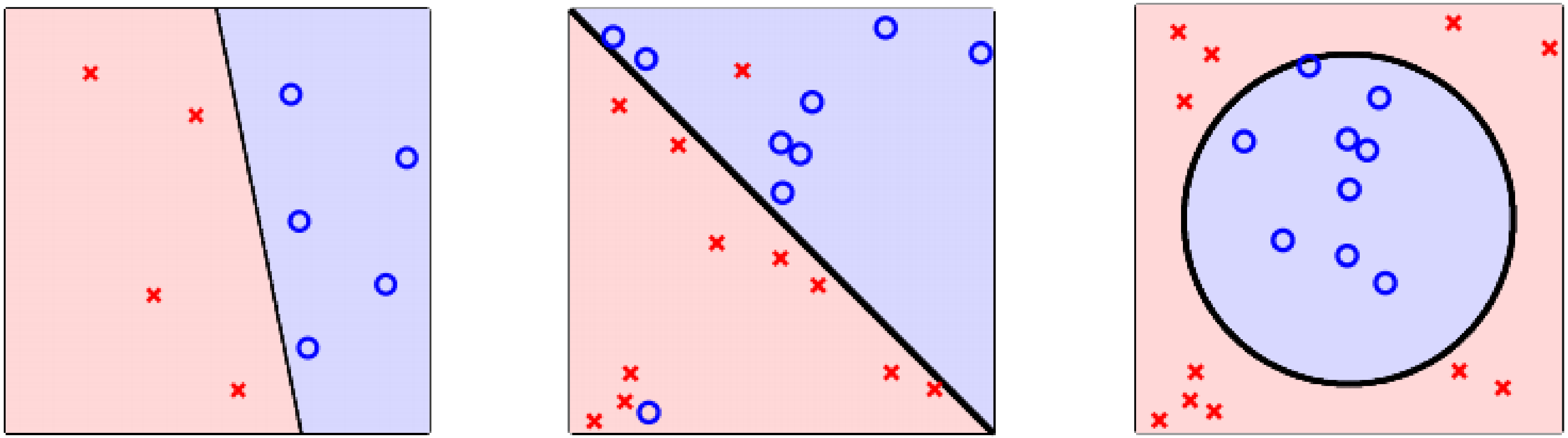
Learning: $g \approx f$?

- 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**

Q2: Will PLA "halt"?

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

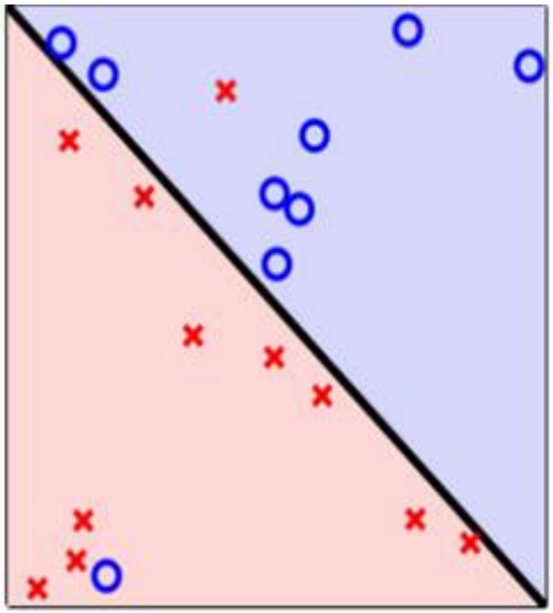


(linear separable) (not linear separable) (not linear separable)

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

Yes

Q2: Will PLA "halt"?



- 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?

Q2: Will PLA "halt"?

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.

Q3: Guarantee of PLA

Give $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots\ldots\ldots, (\mathbf{x}_Q, y_Q)\}$

Output $a = hard\lim({}_1\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$

Let

$$\mathbf{w} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}, \qquad \mathbf{x}_q = \begin{bmatrix} \mathbf{x}_q \\ 1 \end{bmatrix}$$

$$n = {}_1\mathbf{w}^T\mathbf{p} + b = \mathbf{w}^T\mathbf{x}$$

$$\mathbf{w}^{new} = \mathbf{w}^{old} + e\mathbf{x}$$

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{x}'(k-1)$$

$$\mathbf{x}'(k-1) : \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_Q, -\mathbf{x}_1, -\mathbf{x}_2, \ldots, -\mathbf{x}_Q\}$$

$$\textbf{If } y_q = 1, \qquad \mathbf{w}^{*T}\mathbf{x}_q > \delta > 0$$

$$\textbf{If } y_q = 0, \qquad \mathbf{w}^{*T}\mathbf{x}_q < -\delta < 0$$

After k iterations,

$$\mathbf{w}(k) = \mathbf{x}^{'}(0) + \mathbf{x}^{'}(1) + ... + \mathbf{x}^{'}(k-1)$$

$$\Downarrow$$

Then

$$\mathbf{w}^{*\mathbf{T}}\mathbf{w}(k) = \mathbf{w}^{*\mathbf{T}}\mathbf{x}^{'}(0) + \mathbf{w}^{*\mathbf{T}}\mathbf{x}^{'}(1) + ... + \mathbf{w}^{*\mathbf{T}}\mathbf{x}^{'}(k-1)$$

$$\mathbf{w}^{*T}\mathbf{x}^{'}(i) > \delta,$$

$$\mathbf{w}^{*T}\mathbf{w}(k) > k\delta$$

$$(\mathbf{w}^{*T}\mathbf{w}(k))^2 \le \left\|\mathbf{w}^*\right\|^2 \left\|\mathbf{w}(k)\right\|^2$$

❖**The lower boundary**

$$\left\|\mathbf{w}(k)\right\|^2 \ge \frac{(\mathbf{w}^{*T}\mathbf{w}(k))^2}{\left\|\mathbf{w}^*\right\|^2} > \frac{(k\delta)^2}{\left\|\mathbf{w}^*\right\|^2}$$

❖**The upper boundary**

$$\left\|\mathbf{w}(k)\right\|^2 = \mathbf{w}^{\mathbf{T}}(k)\mathbf{w}(k)$$

$$= \left[\mathbf{w}(k-1) + \mathbf{x}^{'}(k-1)\right]^T \left[\mathbf{w}(k-1) + \mathbf{x}^{'}(k-1)\right]$$

$$= \mathbf{w}^{\mathbf{T}}(k-1)\mathbf{w}(k-1) + 2\mathbf{w}^{\mathbf{T}}(k-1)\mathbf{x}^{'}(k-1) + \mathbf{x}^{'\mathbf{T}}(k-1)\mathbf{x}^{'}(k-1)$$

❖**For**

$$\mathbf{w}^{\mathbf{T}}(k-1)\mathbf{x}^{'}(k-1) \le 0$$

$$\Downarrow$$

$$\left\|\mathbf{w}(k)\right\|^2 \le \left\|\mathbf{w}(k-1)\right\|^2 + \left\|\mathbf{x}^{'}(k-1)\right\|^2$$

$$\|\mathbf{w}(k)\|^2 \le \|\mathbf{x}'(0)\|^2 + \ldots\ldots + \|\mathbf{x}'(k-1)\|^2$$

❖**Let** $II = \max\left\{\|\mathbf{x}'(i)\|^2\right\}$

$$\|\mathbf{w}(k)\|^2 \le kII$$

$$\Downarrow$$

$$kII \ge \|\mathbf{w}(k)\|^2 > \frac{(k\delta)^2}{\|\mathbf{w}^*\|^2}$$

$$\Downarrow$$

$$\boxed{k < \frac{II\,\|\mathbf{w}^*\|^2}{\delta^2}}$$

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:

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

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

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

$\mathbf{w}_t$ 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,

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

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

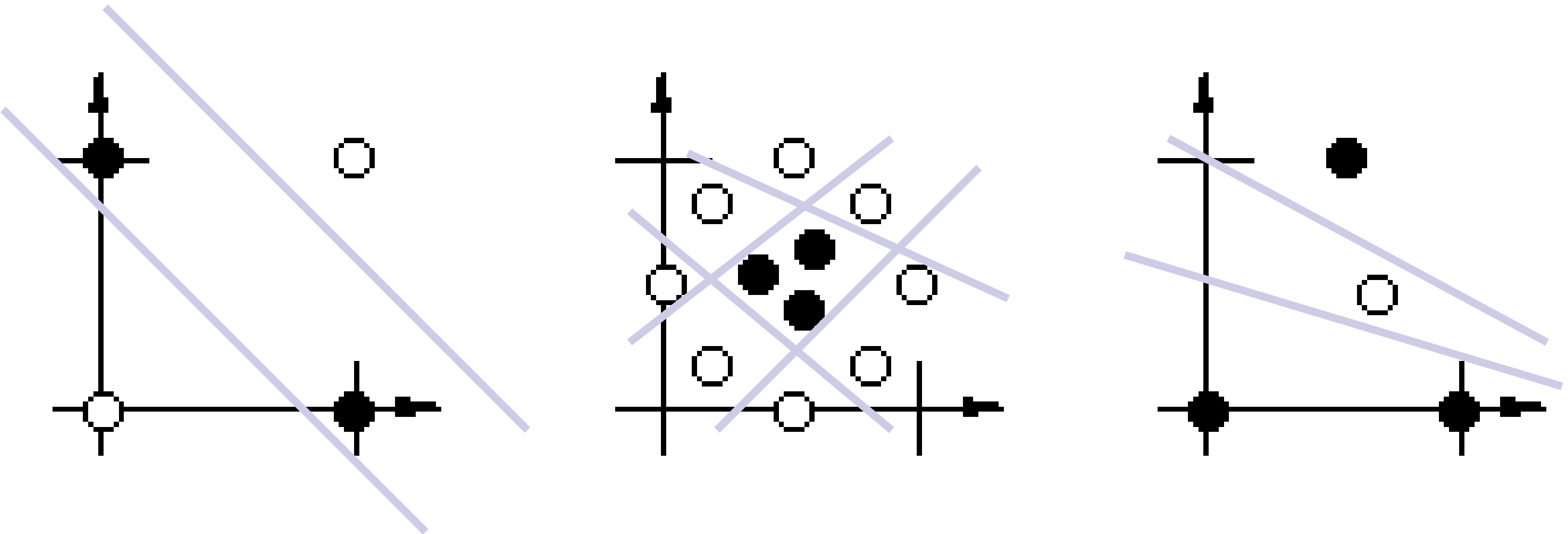
- **'assumes' linear separable $\mathcal{D}$** to halt
  —property unknown in advance (no need for PLA if we know $\mathbf{w}_f$)
- not fully sure **how long halting takes** (ρ depends on $\mathbf{w}_f$)
  —though practically fast

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

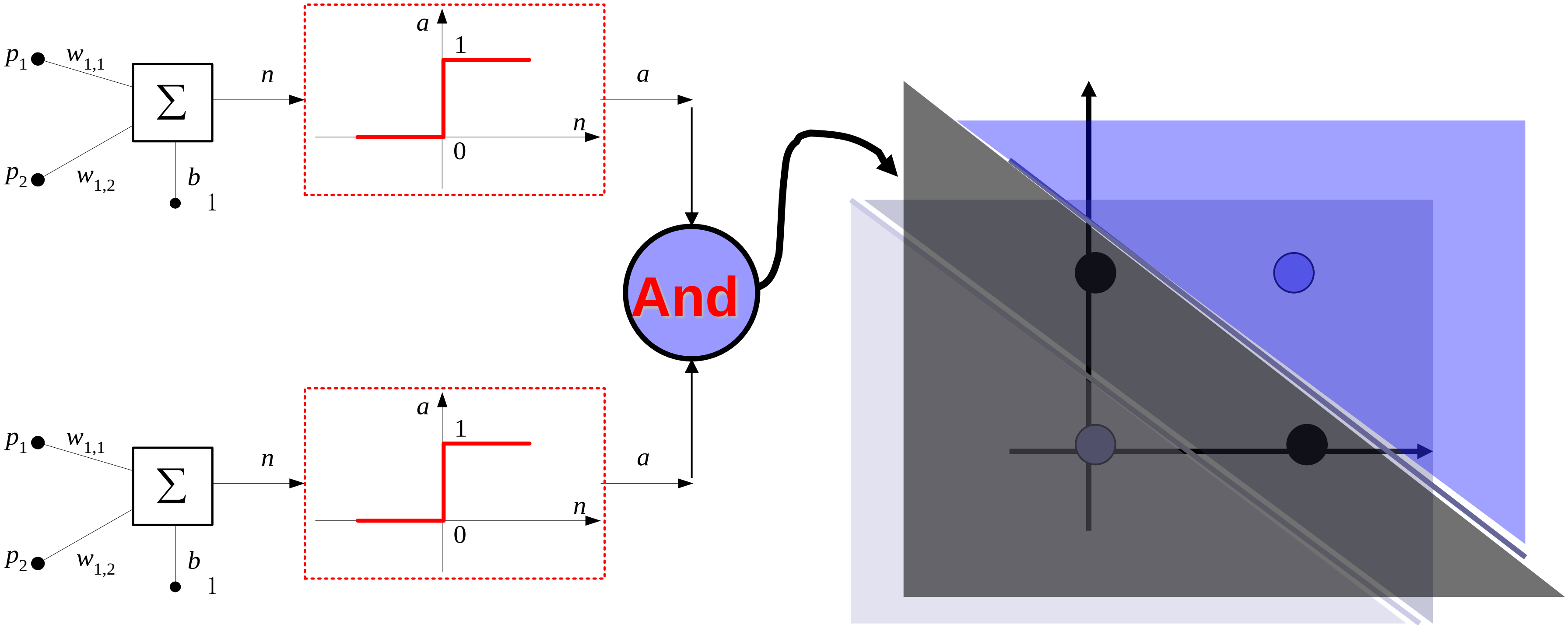
Content

- Lecture 2: Learning to prediction/classification
  - Correct mistakes and improve iteratively
  - Guarantee of PLA
  - Non-Separable Data
  - Multiple Layer Perceptron Network (MLPN)

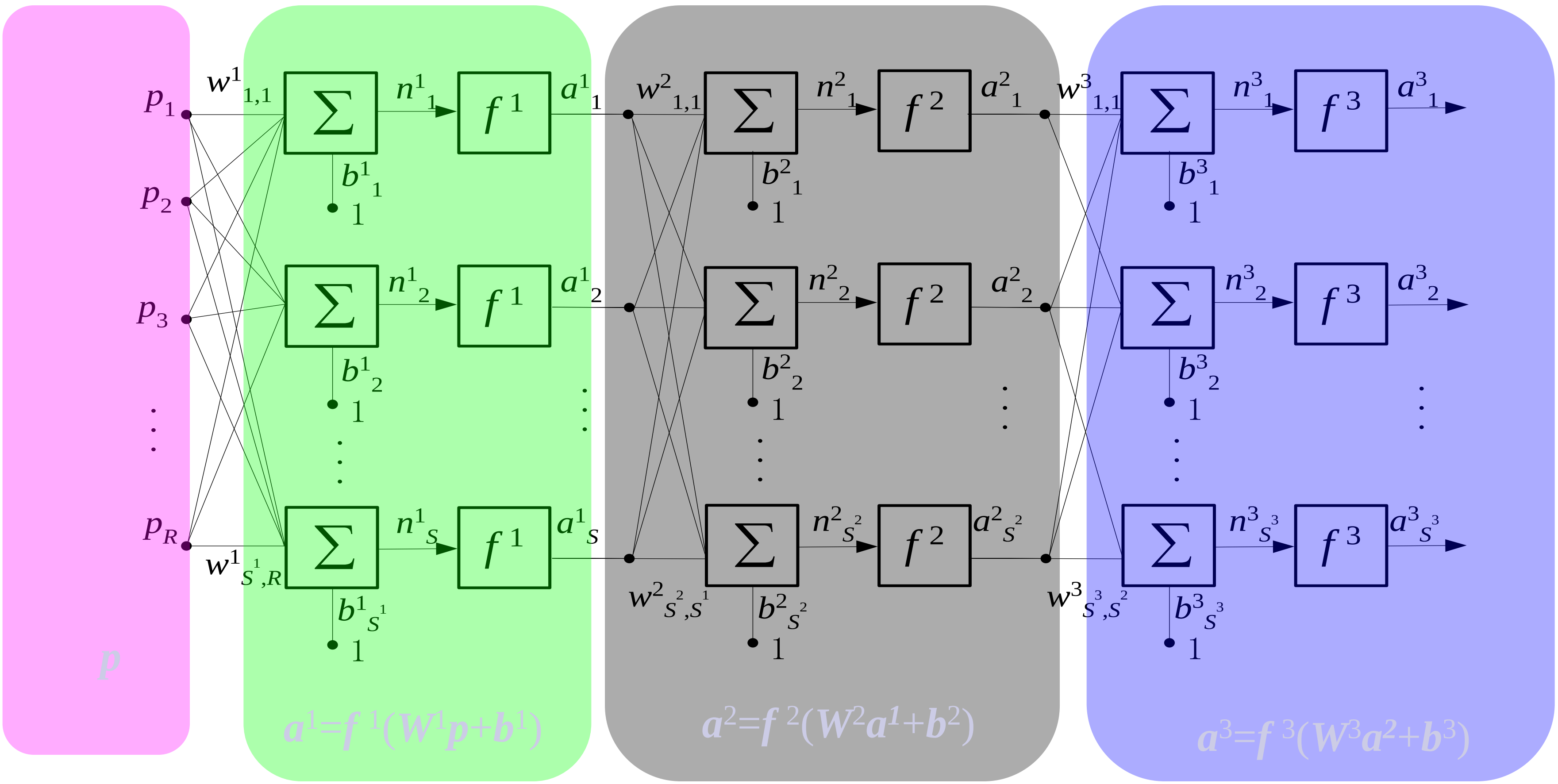
Non-Separable Data



Non-Separable Data

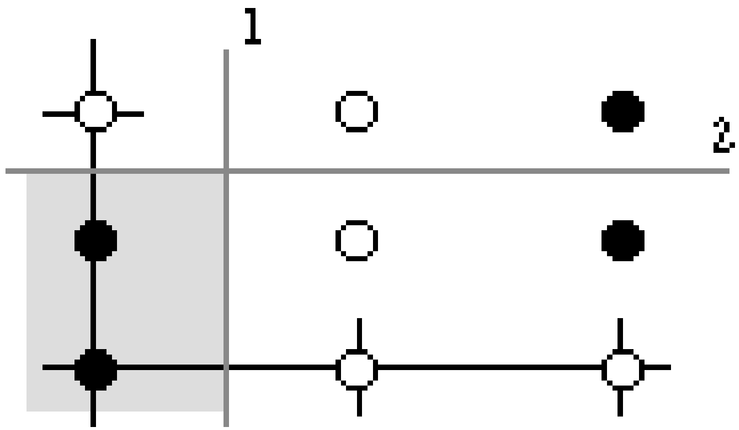


Multiple Layer Perceptron Network



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

An example



Boundary 1:

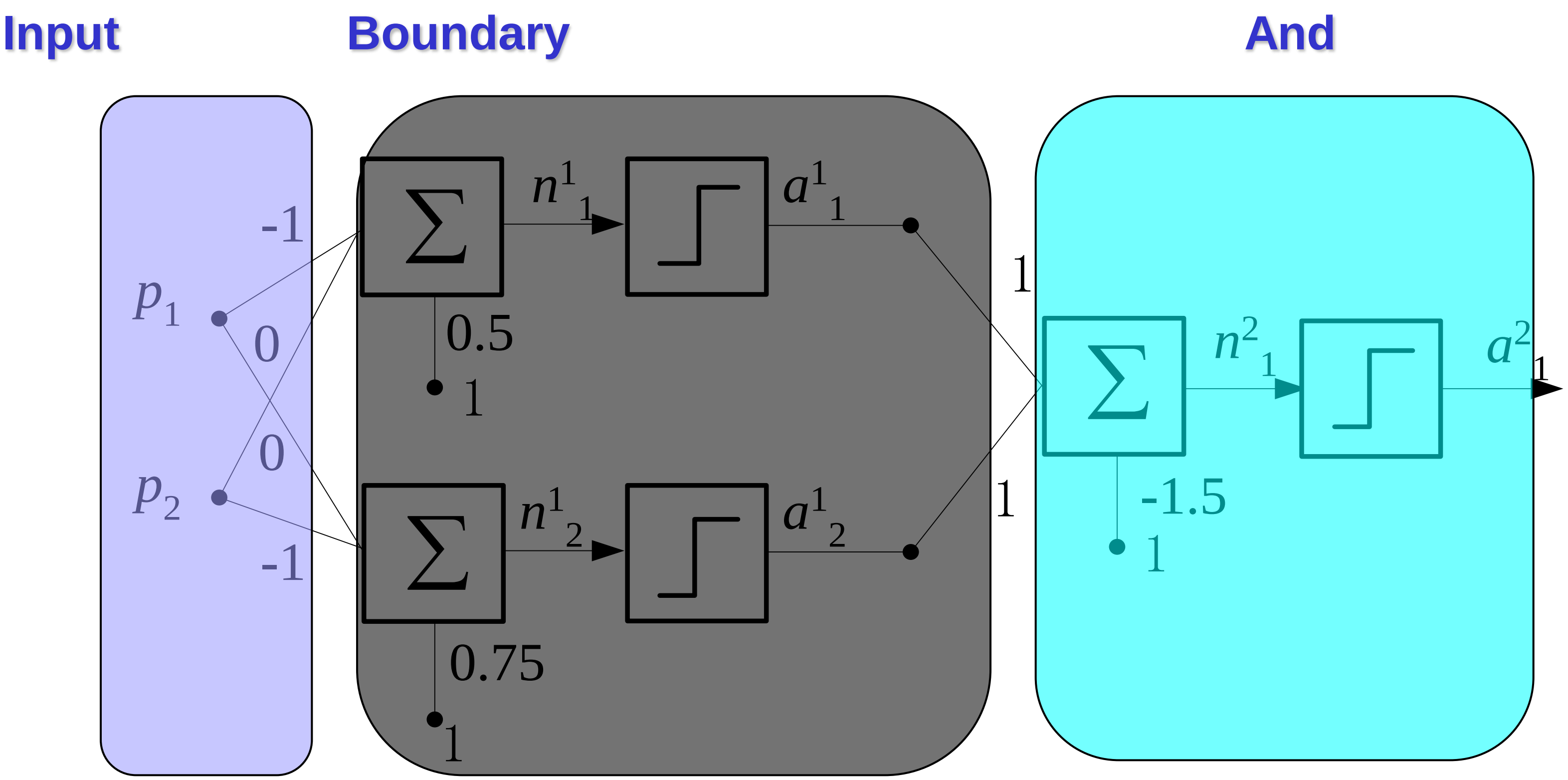
$$a_1^1 = hardlim(\begin{bmatrix} -1 & 0 \end{bmatrix}\mathbf{p} + 0.5)$$

Boundary 2:

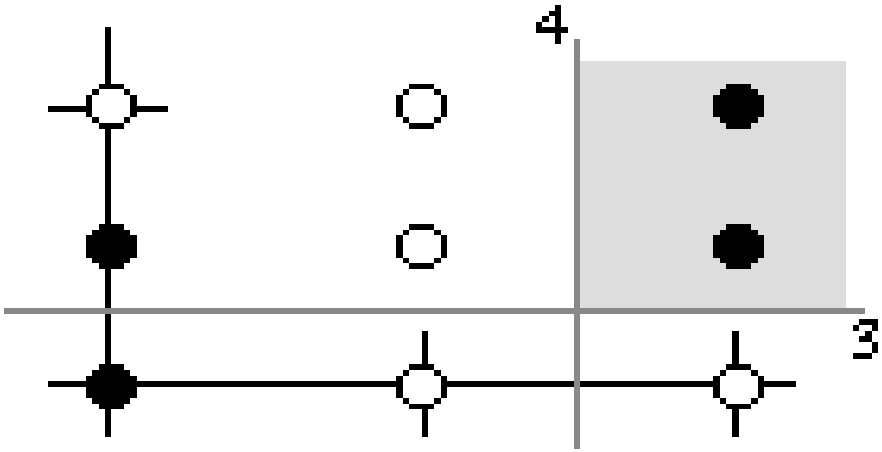
$$a_2^1 = hardlim(\begin{bmatrix} 0 & -1 \end{bmatrix}\mathbf{p} + 0.75)$$

An example

Sub-network 1



An example

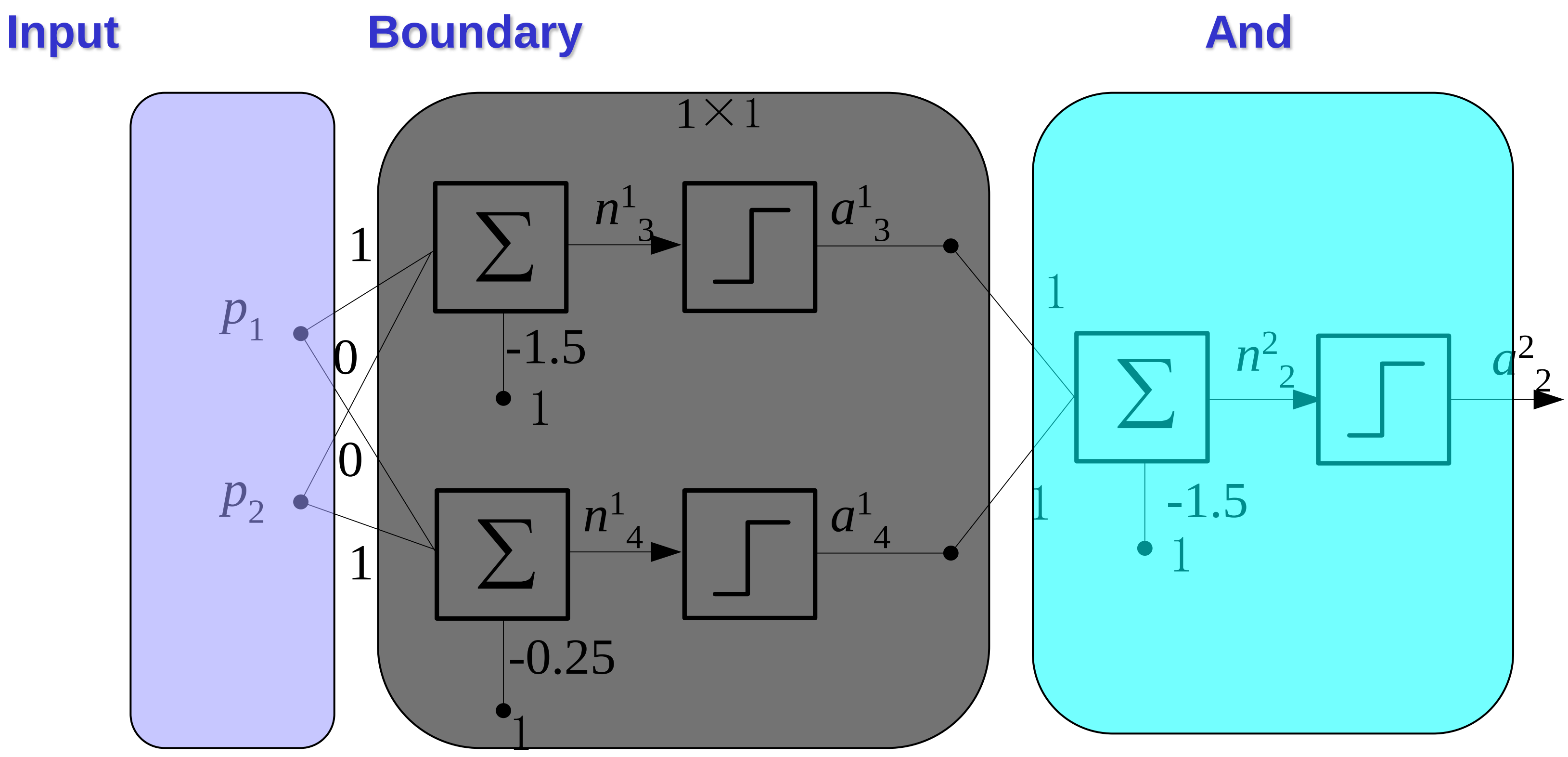


Boundary 3: $a_3^1 = hardlim(\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{p} - 1.5)$

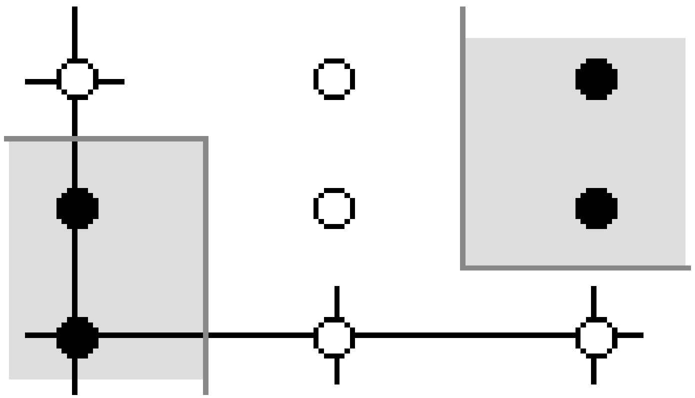
Boundary 4: $a_4^1 = hardlim(\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{p} - 0.25)$

An example

Sub-network 2



An example



$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{b}^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}$$

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$

Input **Boundary** **And** **Or**