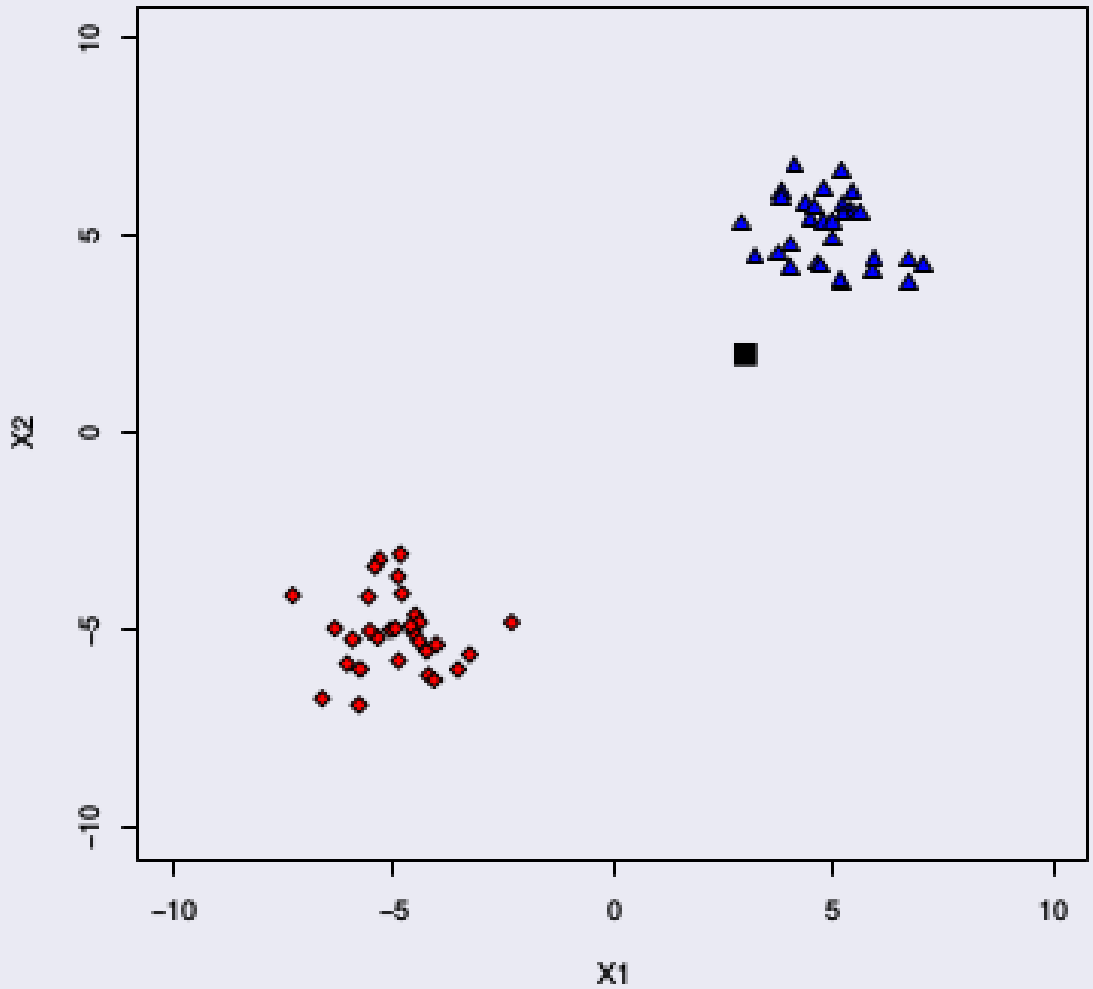
Apprentissage Automatique : K Plus Proches Voisins, Perceptron

Rym Guibadj

LISIC, ULCO, EILCO

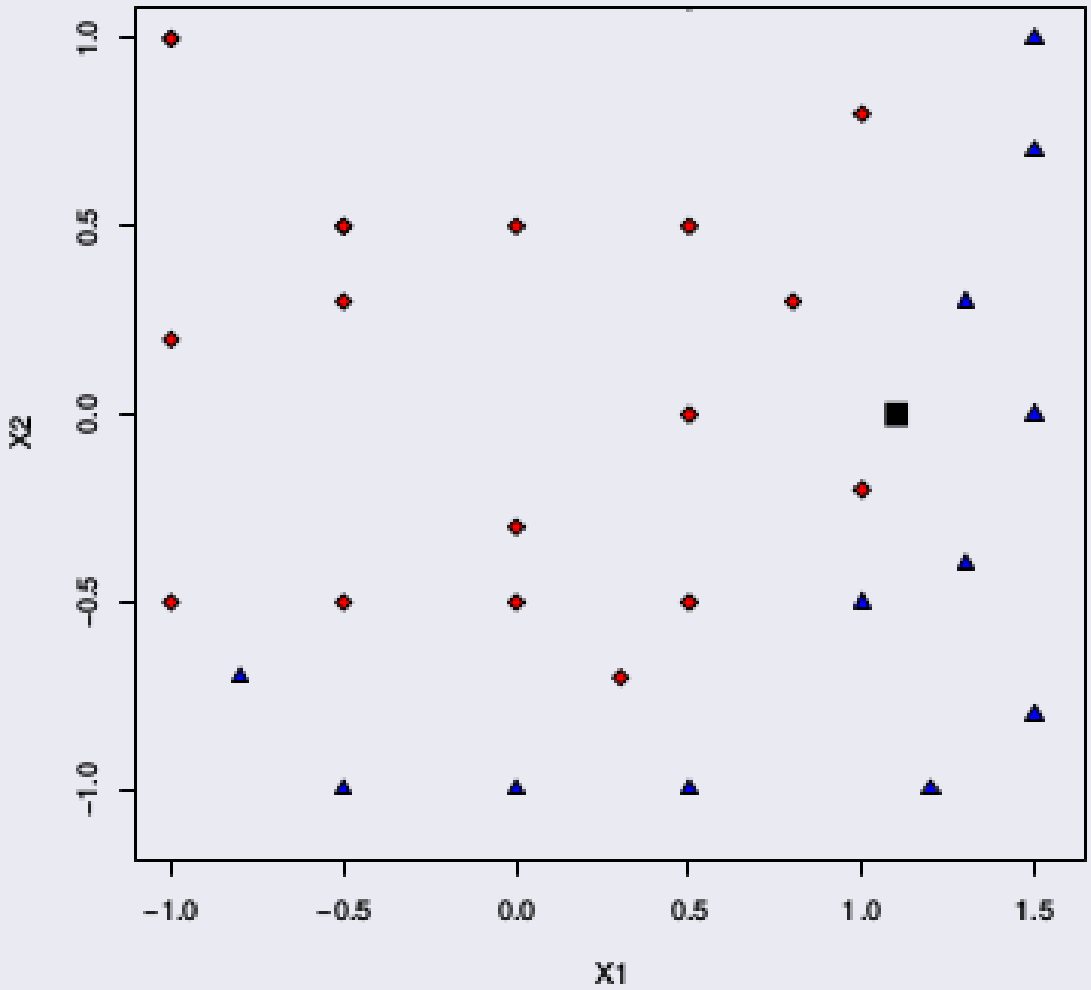
Algorithme des K plus proches voisins

Comment classer le carré noir ?



Algorithme des K plus proches voisins

Comment classer le carré noir ?



Algorithme des K plus proches voisins

Comment classer le carré noir ?

- On cherche le plus proche voisin.
- On tag le nouveau point avec la classe correspondante.

Cas des points aberrants

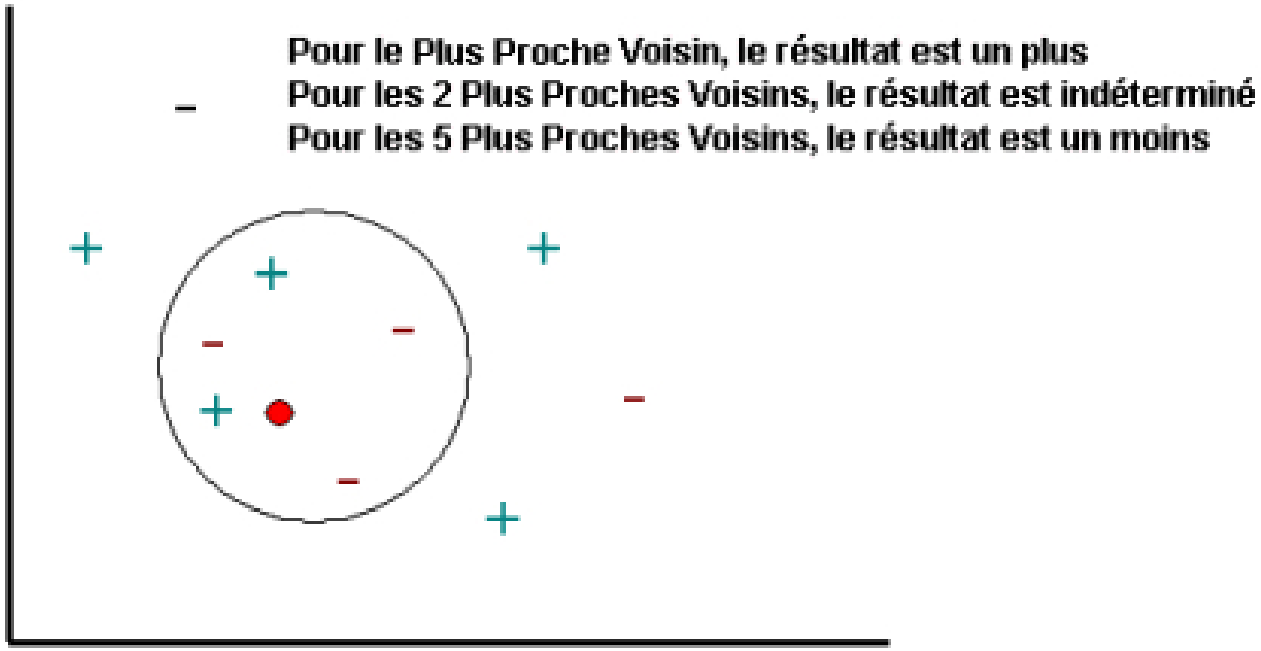
- Un point aberrant change fortement la frontière apprise.
- Comportement non souhaité.

Une solution

- Utiliser plus de 1 voisin.
- Compter les K plus proches voisins et prendre la classe dominante.

Algorithme des K plus proches voisins

Choix de K ?



Algorithme des K plus proches voisins

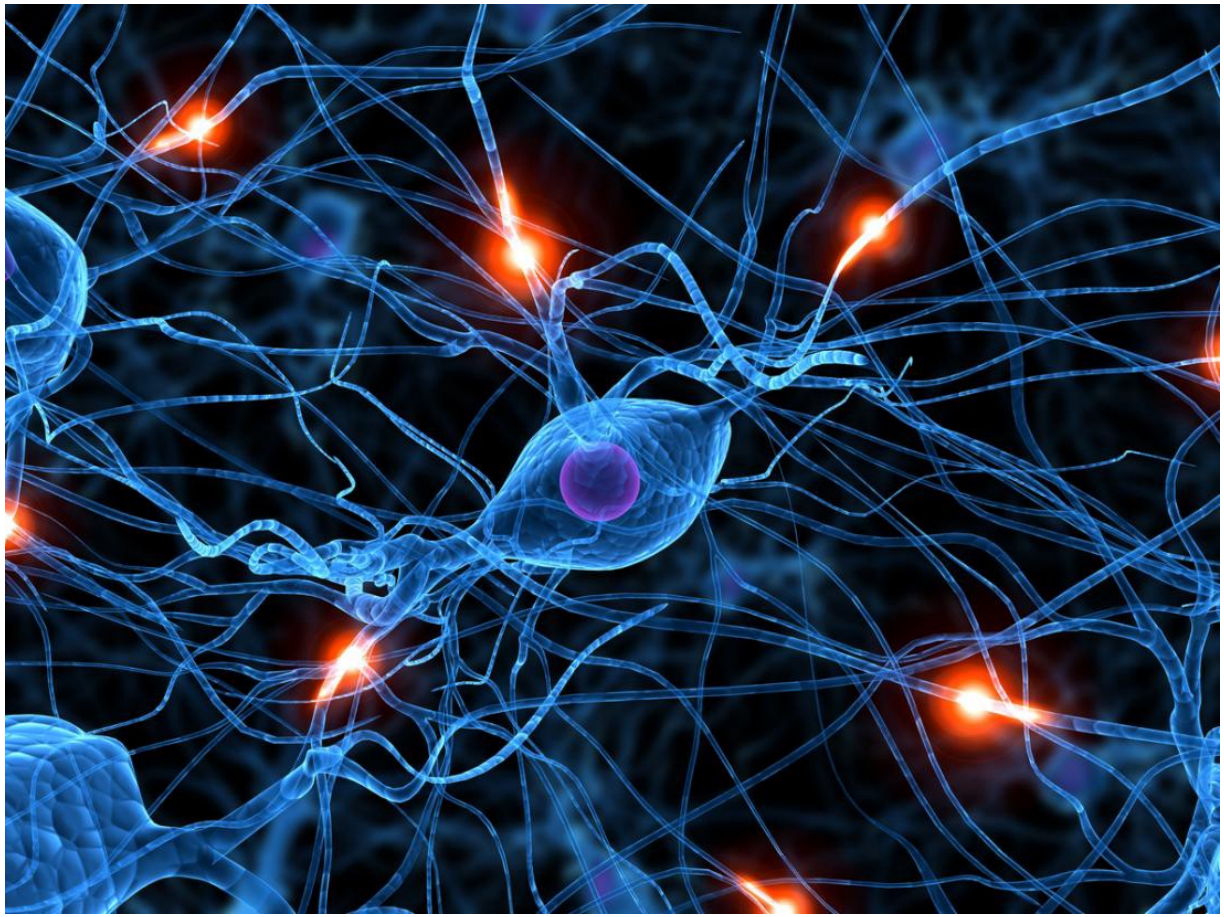
- Données :
  - L'ensemble des points d'apprentissage (x_i, y_i).
  - Le nouveau point à classer x.
- Algorithme :
  - Toutes les distances $D(x_i, x)$.
  - Sélection des K plus proches exemples $x_1 \ldots x_k$.
  - Choix de la classe la plus présente parmi les $y_1 \ldots y_k$ correspondants.

Algorithme des *K* plus proches voisins

- **KNN avec de données catégorielle :** utiliser des mesures de dissimilarité adaptées aux données catégorielles (exemple la distance de *Hamming*)
- **KNN pour la régression :** au lieu de voter pour la classe majoritaire comme dans la classification, on prend la moyenne (ou une autre mesure de centralité) des valeurs cibles des k voisins les plus proches.

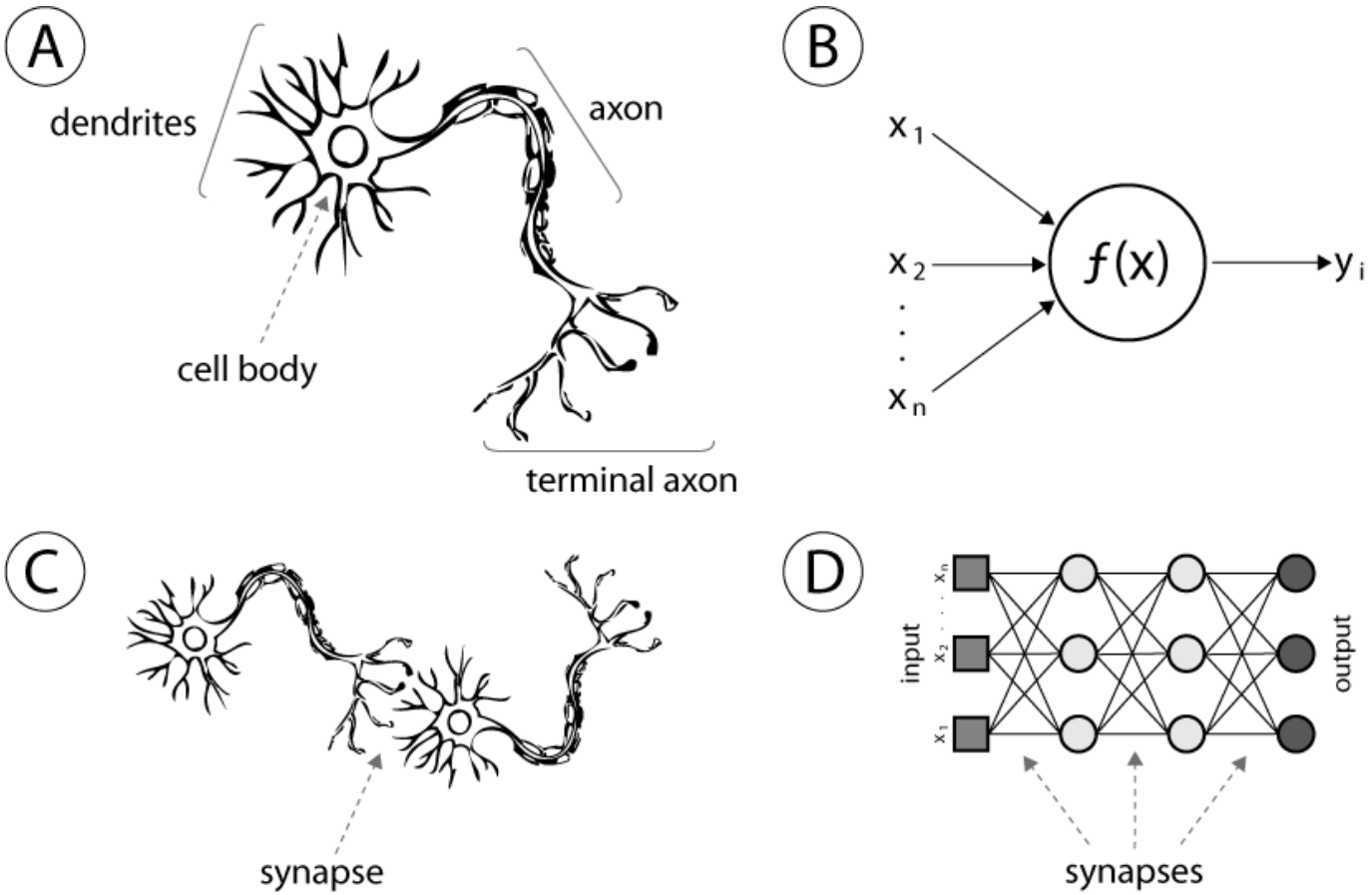
Perceptron : introduction

Le **cerveau** : réseau très complexe ayant un grand nombre de cellules de base interconnectées. Il y a environs 100 milliards de neurones et 10^{15} connexions.

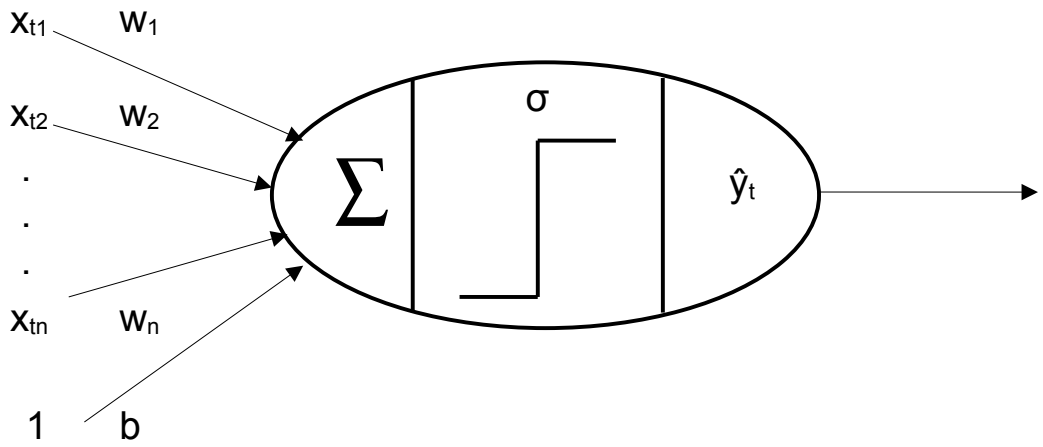


Perceptron : introduction

Les **réseaux de neurones** : modèle très simple du cerveau où les unités de calcul élémentaires sont interconnectées.



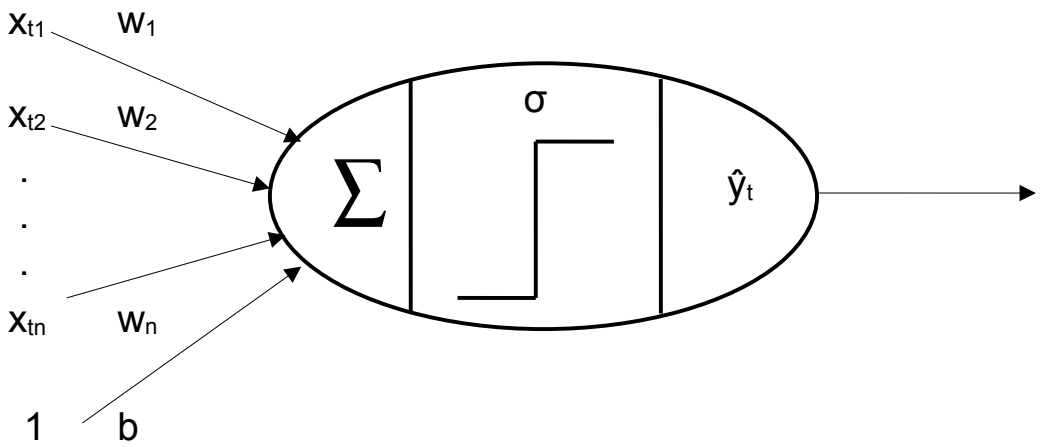
Perceptron (Rosenblatt, 1957)



Définition

- Le perceptron est un classifieur linéaire.
- Il a une seule sortie à laquelle toutes les entrées sont connectées.

Perceptron (Rosenblatt, 1957)

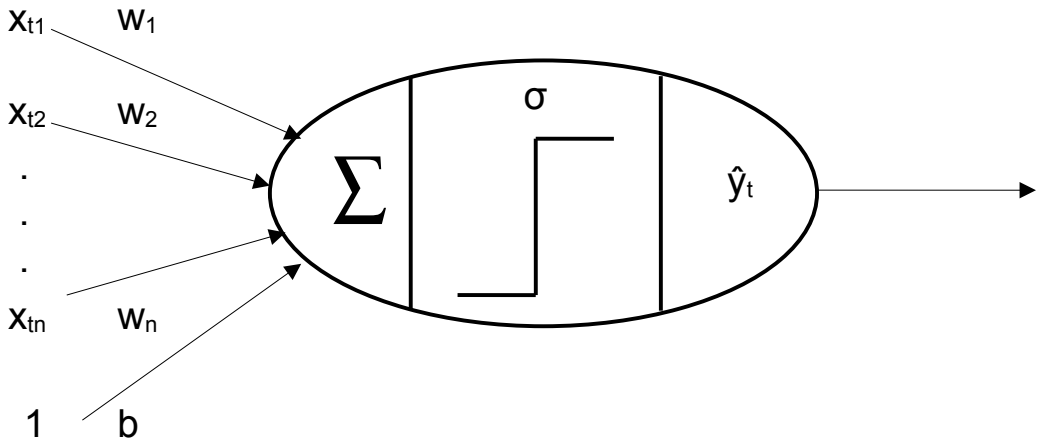


Principe

- Le **vecteur de poids** W correspond aux **paramètres** du modèle
- On ajoute également un biais b, qui équivaut à ajouter une entrée $x_{n+1} = 1$
- A partir de l'échantillon d'apprentissage, il faut trouver le vecteur de poids W et le biais b, tel que :

$$W.X + biais \begin{cases} \geq & 0 \\ < & 0 \end{cases} \Rightarrow X \in \begin{cases} c_1 \\ c_2 \end{cases}$$

Perceptron (Rosenblatt, 1957)



Principe

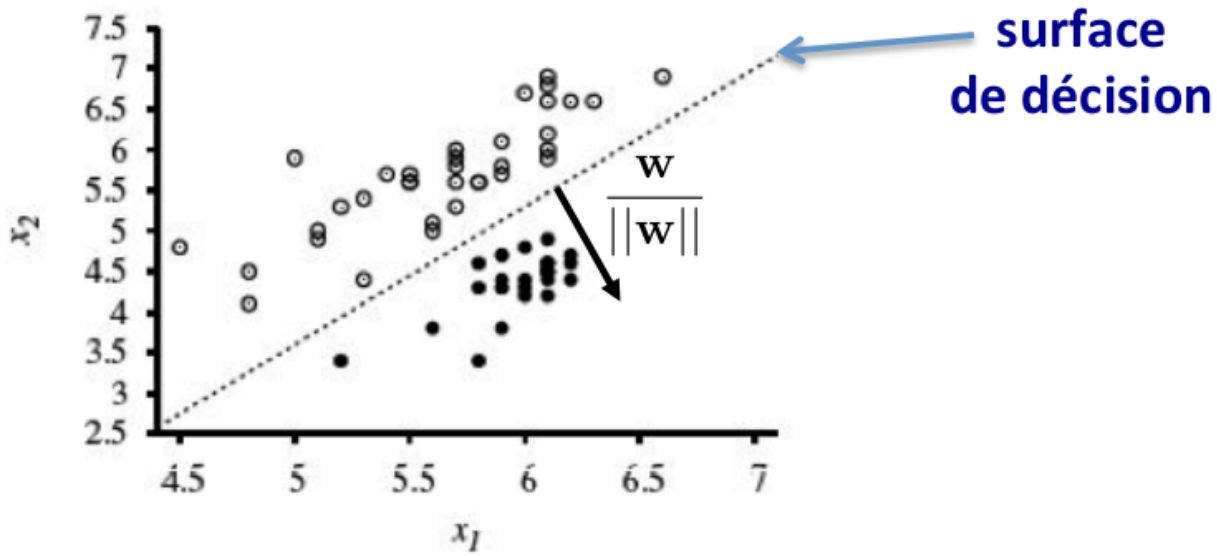
- **Idée :** modéliser la décision à l'aide d'une fonction linéaire, suivi d'une fonction d'activation :

$$\hat{y}_t = \sigma(\sum_{i=1}^{i=n} w_i x_{t,i} + b)$$

où $\sigma(z) = 1$ si $z \geq 0$, sinon $\sigma(z) = 0$

Surface de séparation

- Le perceptron cherche un **séparateur linéaire** entre les deux classes



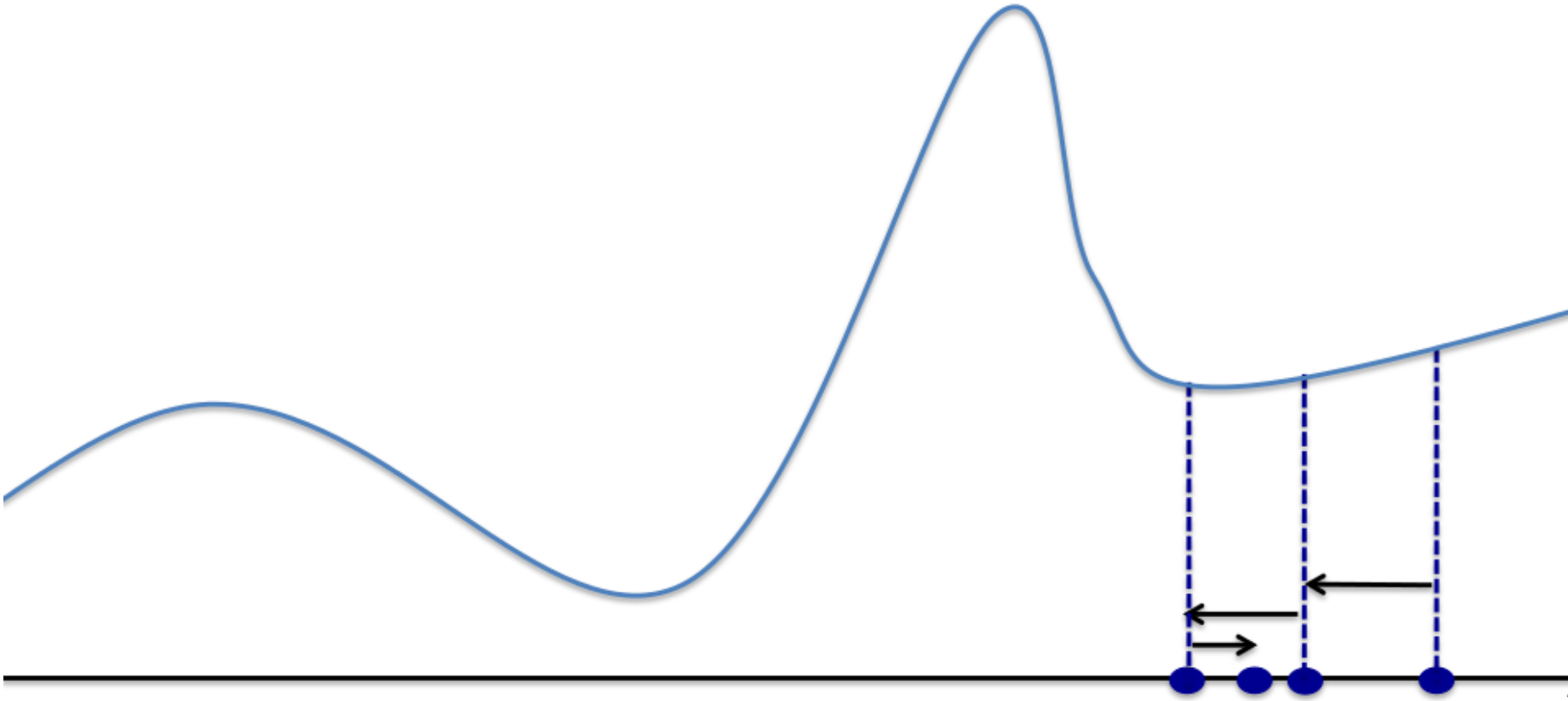
.

- La **surface de décision** d'un classifieur est la surface qui sépare les deux régions classifiées dans les deux classes différentes

Apprentissage vue comme la minimisation d'une perte

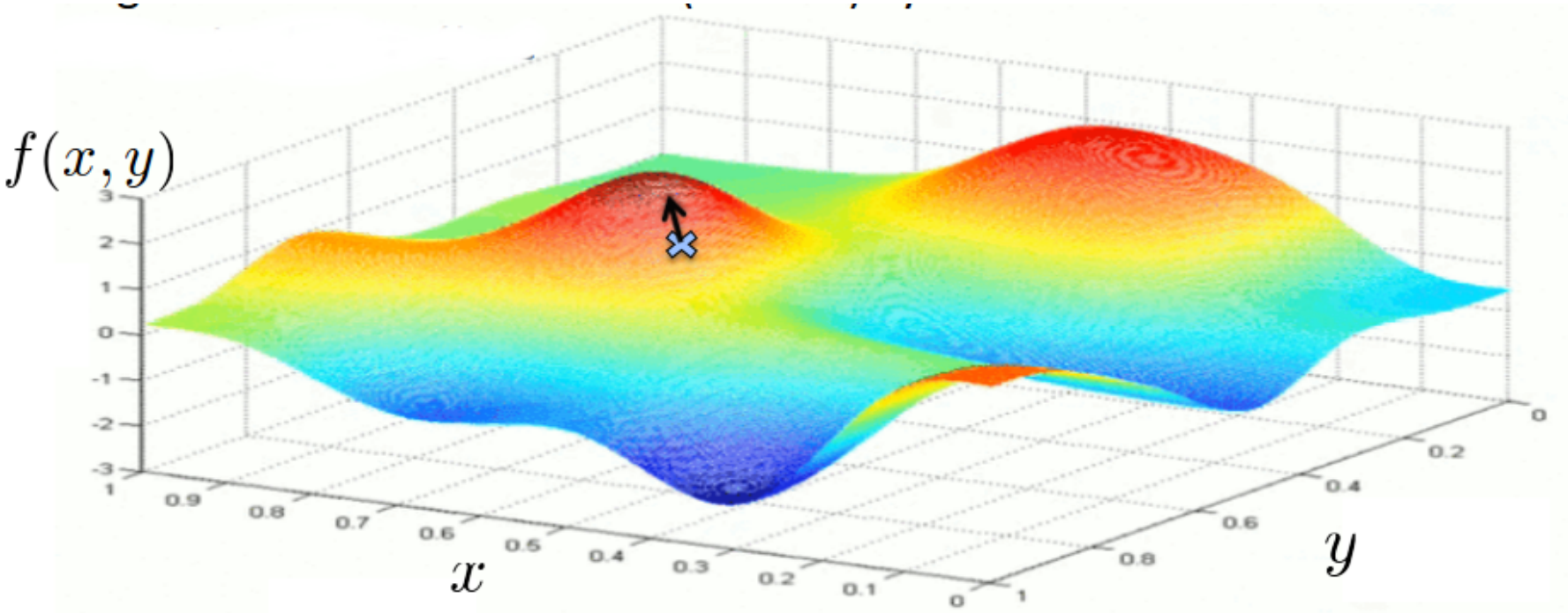
- Le problème de l'apprentissage peut être formulé comme un problème d'optimisation
  - Pour chaque exemple d'entraînement, on souhaite minimiser une certaine distance $Loss(y_t, \hat{y}_t)$ entre la cible y_t et la prédiction $\hat{y}_t$
  - On appelle cette distance une **perte** : $Loss(y_t, \hat{y}_t) = (y_t - \hat{y}_t)^2$

Algorithme de descente de gradient



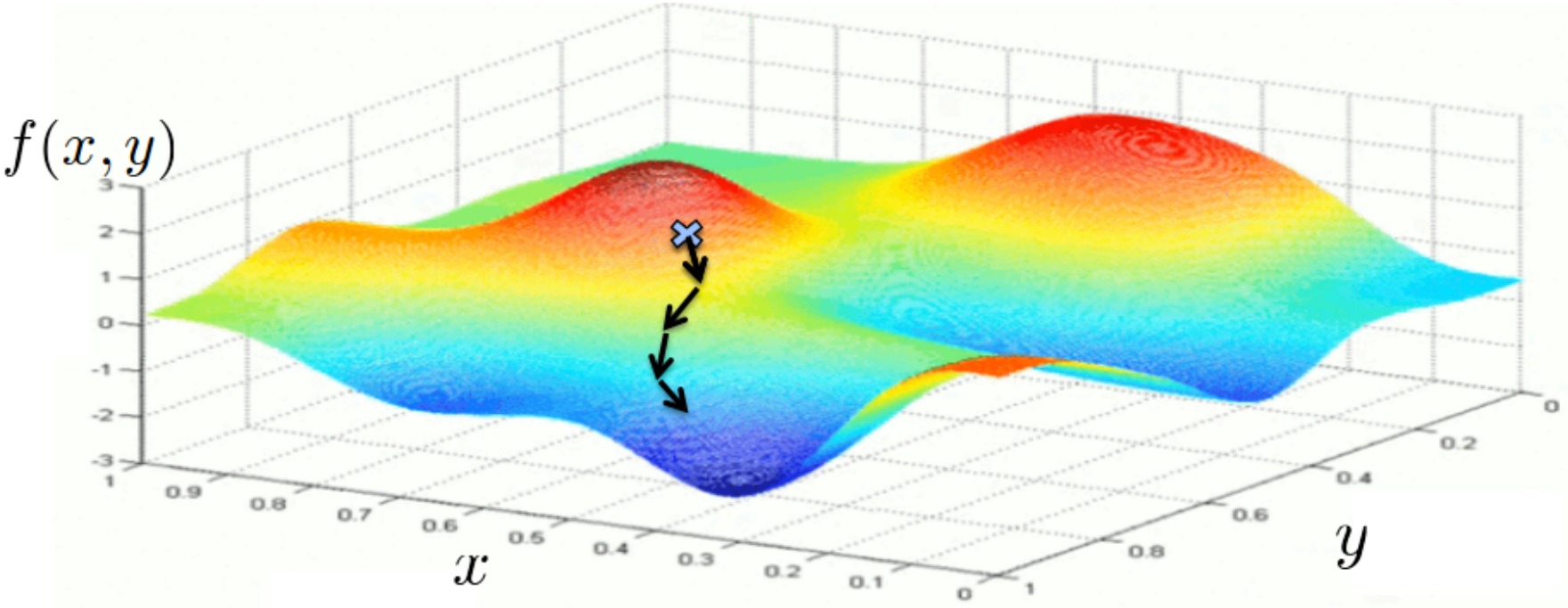
Algorithme de descente de gradient

- Le gradient donne la direction (vecteur) ayant le taux d'accroissement de la fonction le plus élevé



Algorithme de descente de gradient

- Le gradient donne la direction (vecteur) ayant le taux d'accroissement de la fonction le plus élevé



Algorithme de descente de gradient

- En apprentissage automatique, on souhaite optimiser :

$$\frac{1}{|D|} \sum_{(x_t, y_t) \in D} Loss(y_t, \hat{y}_t)$$

  - D : les données d'apprentissage
  - $\hat{y}_t$: la prédiction du modèle $\hat{y}_t = \sigma(\sum_{i=1}^{i=n} w_i x_i + b)$
  - y_t : la valeur réelle
- Le gradient par rapport à la perte moyenne contient les dérivées partielles :

$$\frac{1}{|D|} \sum_{(x_t, y_t) \in D} \frac{\partial}{\partial w_i} Loss(y_t, \hat{y}_t) \quad \forall i$$

- Calculer la moyenne des dérivées sur tous les exemples d'entraînement avant de faire une mise à jour des paramètres !

Descente de gradient stochastique

- Descente de gradient stochastique : mettre à jour les paramètres à partir du gradient d'un seul exemple, choisi aléatoirement :

Initialiser w aléatoirement
Pour T itérations
Pour chaque exemple d'entraînement (x_t, y_t)

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, \hat{y}_t) \quad \forall i$$

- Cette procédure est plus efficace lorsque l'ensemble d'entraînement est grand

Retour sur le perceptron

- On utilise le gradient pour déterminer une direction de mise à jour des paramètres :

$$\frac{\partial}{\partial w_i} Loss(y_t, \hat{y}_t) = \frac{\partial}{\partial w_i}(y_t - \hat{y}_t)^2 = 2(y_t - \hat{y}_t) * \frac{\partial}{\partial w_i}(y_t - \hat{y}_t)$$

$$= 2(y_t - \hat{y}_t) * \frac{\partial}{\partial w_i}(y_t - (w_1 x_{t,1} + w_2 x_{t,2} + \ldots + w_i x_{t,i} + \ldots + w_n x_{t,n} + b))$$

$$\simeq -2(y_t - \hat{y}_t) x_{t,i}$$

- Pour mettre à jour les paramètres, on va dans la direction opposée de ce gradient :

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, \hat{y}_t) \quad \forall i$$

- on obtient la règle d'apprentissage du perceptron

$$w_i \leftarrow w_i + \alpha(y_t - \hat{y}_t) x_{t,i} \quad \forall i$$

Algorithme du perceptron

1. Pour chaque paire de l'ensemble d'apprentissage $(x_t, y_t) \in D$
   - Calculer la réponse produite par le perceptron $\hat{y}_t$
   - Si $y_t = \hat{y}_t$ alors l'exemple est bien classé : ne rien faire
   - Sinon ($y_t \neq \hat{y}_t$) mettre à jour les poids et le biais selon la règle suivante :

$$w_i \leftarrow w_i + \alpha(y_t - \hat{y}_t) * x_{t,i} \quad \forall i$$

2. Retourner à 1 jusqu'à l'atteinte d'un critère d'arrêt : nombre maximal d'itérations atteint ou tous les exemples sont bien classés

La mise à jour des poids est appelée la **règle d'apprentissage** du perceptron. Le multiplicateur α est appelé le **taux d'apprentissage**

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

- On initialise les poids et le biais aléatoirement : $w \leftarrow [0, 0]$, $b = 0.5$
- On choisit le pas égal à 0.1 : $\alpha = 0.1$
- On lit le prochain exemple.
- On calcule la sortie.
- On met à jour les poids si nécessaire $w_i \leftarrow w_i + \alpha(y_t - \hat{y}_t) * x_{t,i}$
- On ré-itère tant qu'on n'a pas convergé et qu'on a du temps

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1

- Puisque $\hat{y}_1 = y_1$, on ne fait pas de mise à jour w et b

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0

- Puisque $\hat{y}_2 \neq y_2$, on met à jour w et b :
- $w \leftarrow w + \alpha * (y_2 - \hat{y}_2) * x_2 = [0, 0] + 0.1 * (0 - 1)[0, 3] = [0, -0.3]$
- $b \leftarrow b + \alpha * (y_2 - \hat{y}_2) = 0.5 + 0.1(0 - 1) = 0.4$

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0

Calcul de la séparatrice

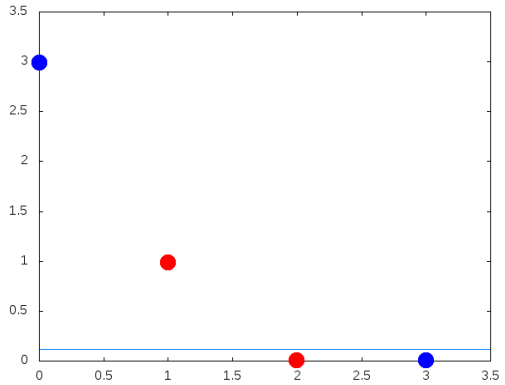
$$
\begin{aligned}
b + w_1 x_1 + w_2 x_2 &= 0 \\
x_2 &= -\frac{w_1}{w_2} * x_1 - \frac{b}{w_2} \\
x_2 &= -\frac{0}{-0.3} * x_1 - \frac{0.4}{-0.3} \\
x_2 &= 0.12
\end{aligned}
$$

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0



Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0

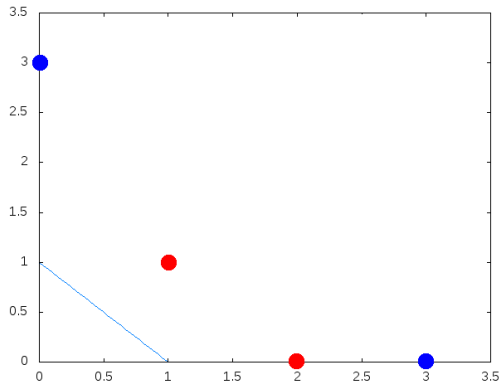
- Puisque $\hat{y}_3 \neq y_3$, on met à jour w et b :
- $w \leftarrow w + \alpha * (y_3 - \hat{y}_3) * x_2 = [0, -0.3] + 0.1 * (0 - 1)[3, 0] = [-0.3, -0.3]$
- $b \leftarrow b + \alpha * (y_3 - \hat{y}_3) = 0.4 + 0.1(0 - 1) = 0.3$

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0
4	$[1, 1]$	0.3	-0.3	-0.3	-0.3	0	1



Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0
4	$[1, 1]$	0.3	-0.3	-0.3	-0.3	0	1

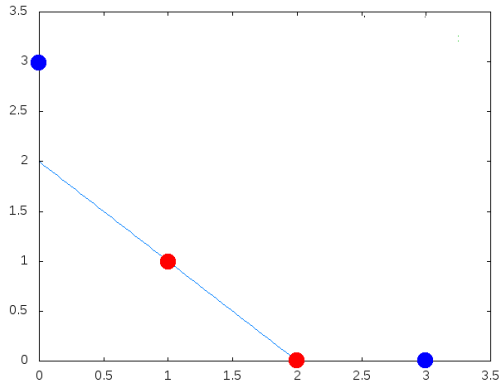
- Puisque $\hat{y}_4 \neq y_4$, on met à jour w et b :
- $w \leftarrow w + \alpha * (y_4 - \hat{y}_4) * x_2 = [-0.3, -0.3] + 0.1 * (1 - 0)[1, 1] = [-0.2, -0.2]$
- $b \leftarrow b + \alpha * (y_4 - \hat{y}_4) = 0.3 + 0.1(1 - 0) = 0.4$

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0
4	$[1, 1]$	0.3	-0.3	-0.3	-0.3	0	1
5	$[2, 0]$	0.4	-0.2	-0.2	0	1	1



Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0
4	$[1, 1]$	0.3	-0.3	-0.3	-0.3	0	1
5	$[2, 0]$	0.4	-0.2	-0.2	0	1	1

- Puisque $\hat{y}_1 = y_1$, on ne met pas à jour w et b
- Et ainsi de suite, jusqu'à l'atteinte d'un critère d'arrêt...

Exemple de déroulement

- Ensemble d'apprentissage D

x_t	y_t
$[2, 0]$	1
$[0, 3]$	0
$[3, 0]$	0
$[1, 1]$	1

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[2, 0]$	0.5	0	0	0.5	1	1
2	$[0, 3]$	0.5	0	0	0.5	1	0
3	$[3, 0]$	0.4	0	-0.3	0.4	1	0
4	$[1, 1]$	0.3	-0.3	-0.3	-0.3	0	1
5	$[2, 0]$	0.4	-0.2	-0.2	0	1	1
6	$[0, 3]$	0.4	-0.2	-0.2	-0.2	0	0
7	$[3, 0]$	0.4	-0.2	-0.2	-0.2	0	0
8	$[1, 1]$	0.4	-0.2	-0.2	0	1	1

- Arrêt de l'algorithme car tous les exemples sont bien classés.

Exemple 2 de déroulement

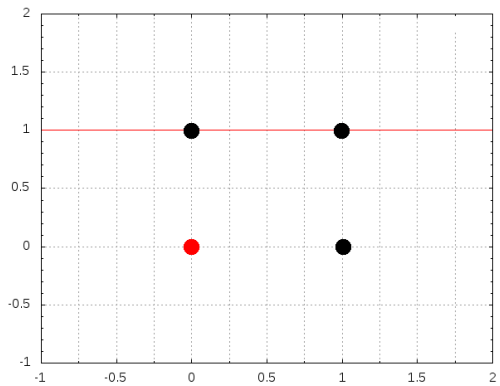
- Apprendre la fonction **OU logique**
- Ensemble d'apprentissage D

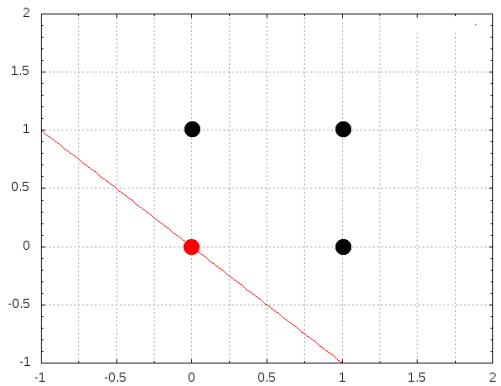
x_t	y_t
$[0, 0]$	0
$[0, 1]$	1
$[1, 0]$	1
$[1, 1]$	1

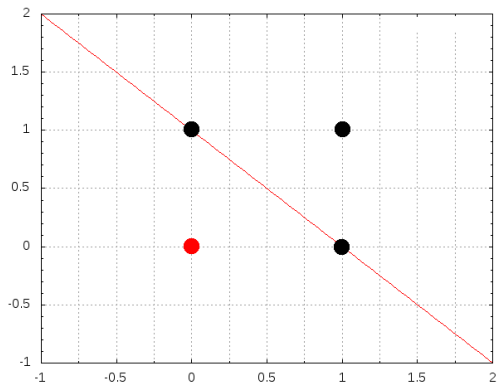
- On initialise les poids et le biais aléatoirement : $w \leftarrow [0, 1]$, $b = -1$
- On choisit le pas égal à 1 : $\alpha = 1$

Exemple 2 de déroulement

Iteration	Exemple	b	w_1	w_2	$\sum x_{t,i} * w_i + b$	$\hat{y}_t$	y_t
1	$[0, 0]$	-1	0	1	-1	0	0
2	$[0, 1]$	-1	0	1	0	1	1
3	$[1, 0]$	-1	0	1	-1	0	1
4	$[1, 1]$	0	1	1	2	1	1
5	$[0, 0]$	0	1	1	0	1	0
6	$[0, 1]$	-1	1	1	0	1	1
7	$[1, 0]$	-1	1	1	0	1	1
8	$[1, 1]$	-1	1	1	1	1	1
9	$[0, 0]$	-1	1	1	-1	0	0

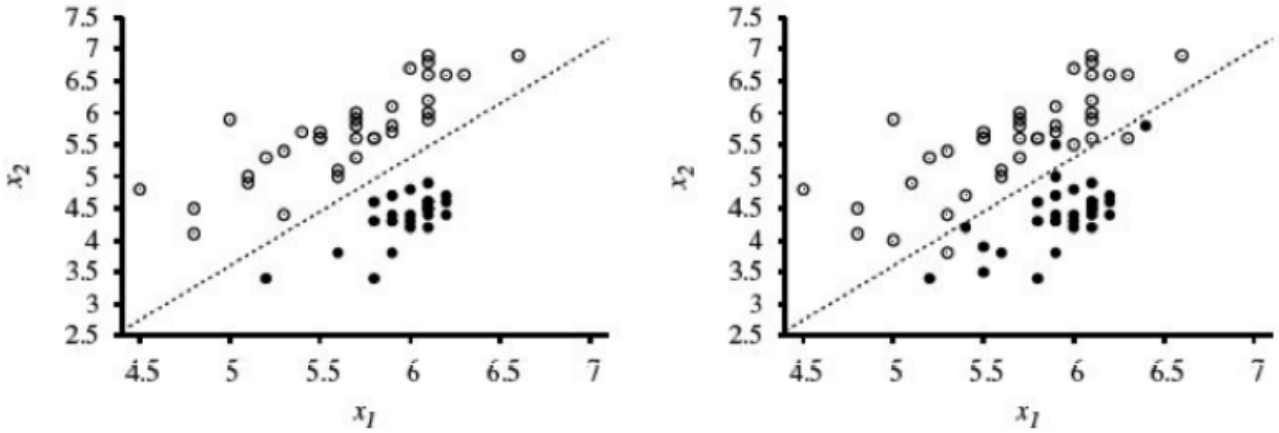






Convergence et séparabilité

- Si les exemples sont **linéairement séparables**, le perceptron garanti de converger à **une solution avec une erreur nulle** sur l'ensemble d'entraînement, pour tout α

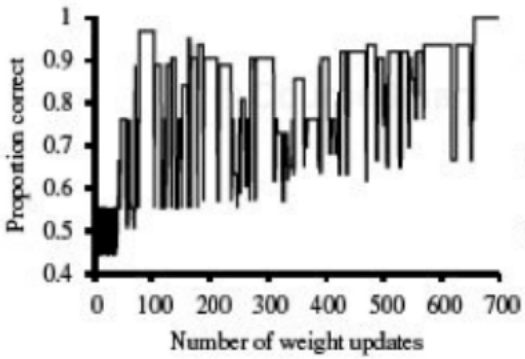


.

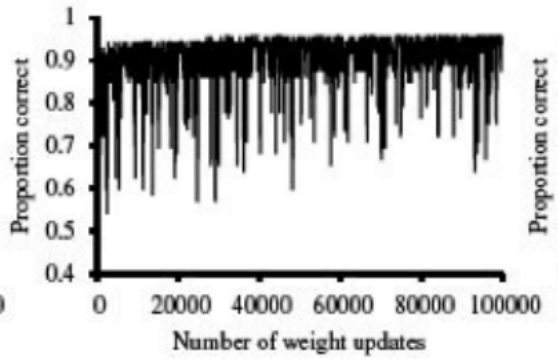
- Sinon, pour garantir la convergence à une **solution ayant la plus petite erreur possible en entraînement**, on doit décroître le taux d'apprentissage, par ex. selon $\alpha_k = \frac{\alpha}{1+\beta k}$

Courbe d'apprentissage

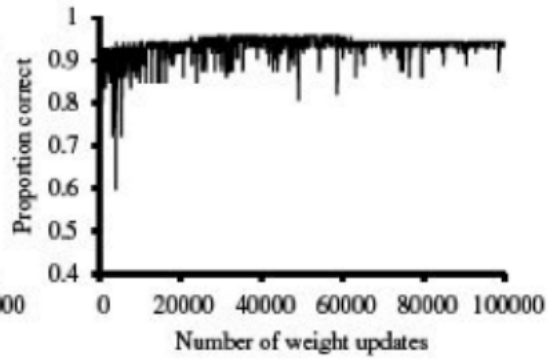
- La courbe d'apprentissage est la courbe du taux d'erreur (ou de succès) en fonction du nombre de mises à jour des paramètres
- Utile pour visualiser la progression de l'apprentissage



linéairement séparable



pas linéairement séparable



taux d'apprentissage décroissant