

# SYS843

## A.3 Réseaux de neurones multicouches sans rétroaction

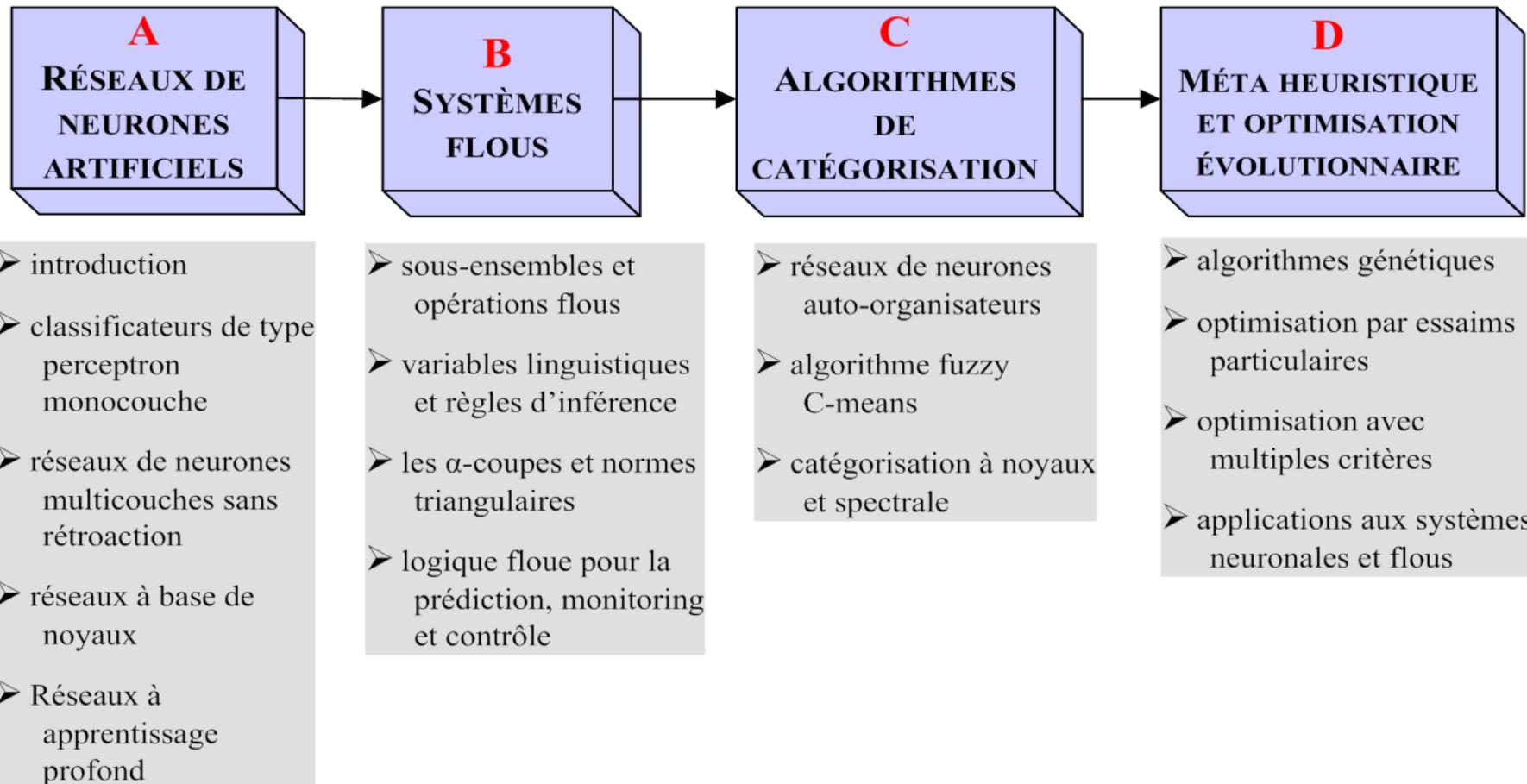
Eric Granger

Ismail Ben Ayed

Hiver 2017

---

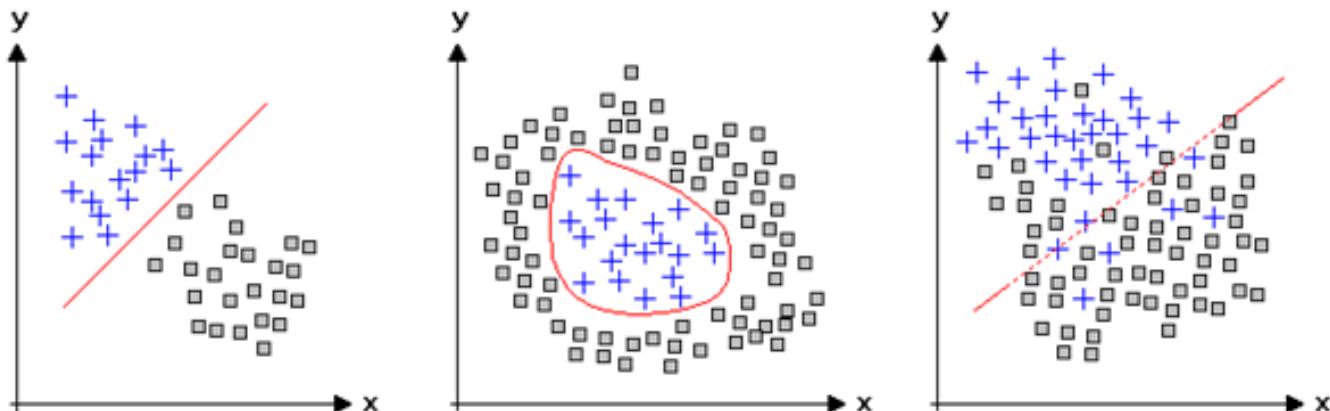
# CONTENU DU COURS



# CONTENU DU COURS

## Réseaux de neurones multicouches

- ▶ Les RNA multicouches sans rétroaction permettent l'estimation de surfaces de décision non-linéaires
- ▶ La principale caractéristique des RNA multicouches est leur capacité d'apprendre ces surfaces à l'aide d'exemples (données) des fonctions de transfert (mapping) arbitrairement complexe



# CONTENU DU COURS

## A.3 Réseaux de neurones multicouches sans rétroaction (MLP)

- 1) Classification de données linéairement non-séparables
- 2) Règle d'apprentissage delta pour une couche de perceptrons
- 3) Règle d'apprentissage delta généralisée
- 4) Entraînement par la rétro-propagation des erreurs
- 5) Facteurs d'apprentissage

# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

### ► Condition de non-séparabilité linéaire:

- Supposons les deux ensembles d'entraînement Y1 (classe 1) et Y2 (classe 2) constitués de vecteurs augmentés  $y$ .
- Les ensembles de données Y1 et Y2 sont linéairement non-séparables s'il n'existe aucun vecteur  $w$  tel que:

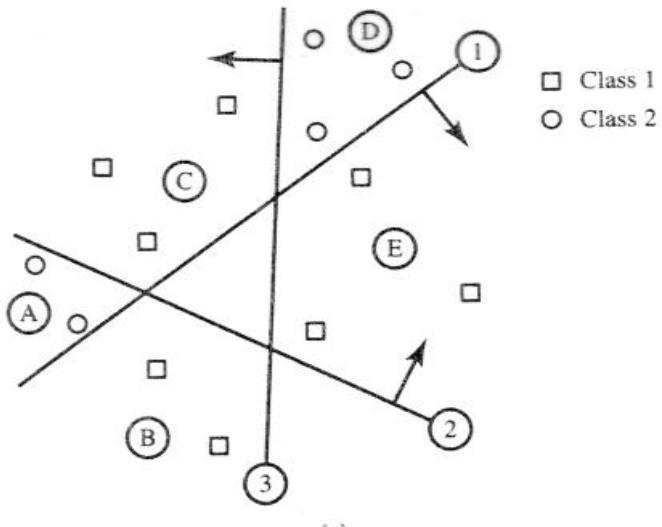
$$y^t w > 0 \text{ pour chaque } y \in Y_1, \text{ et}$$

$$y^t w < 0 \text{ pour chaque } y \in Y_2$$

# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

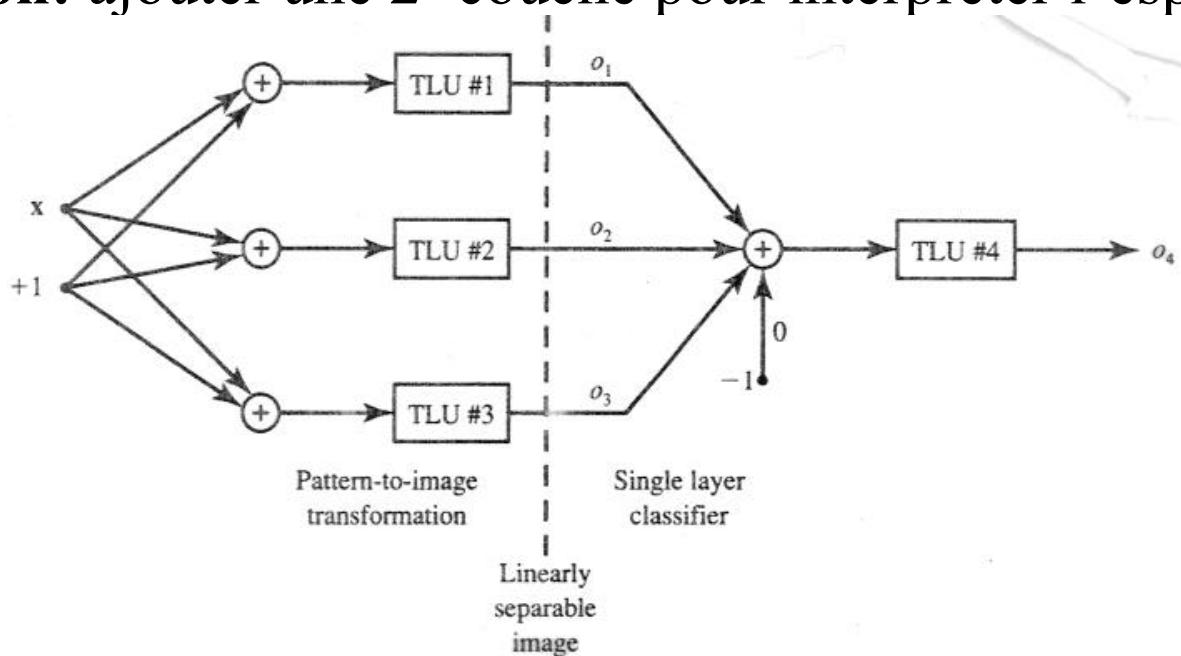
- **Exemple – un ensemble d'entraînement linéairement non-séparable avec  $R = 2$  classes et  $n = 2$  dimensions**
  - 3 plans linéaires, et donc 3 perceptrons discrets, sont utilisés pour séparer les données distribuées des 2 classes



# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

- **Exemple** – Il est évident que les patrons des classes 1 et 2 peuvent être classés correctement à l'aide de *fonctions de décision linéaires par parties*
  - **Solution:** ajouter une 2<sup>e</sup> couche pour interpréter l'espace image



# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

### ► Exemple – suite

- **Couche 1 (plans):** 3 hyperplans de décision peuvent être implantés dans l'espace des données à l'aide de 3 perceptrons discrets  
espace données  $y \in R^n \rightarrow$  espace image  $o \in \{-1,1\}^3$
- **Couche 2 (voteur):** Le dernier perceptron permet de rendre la décision finale dans l'espace image

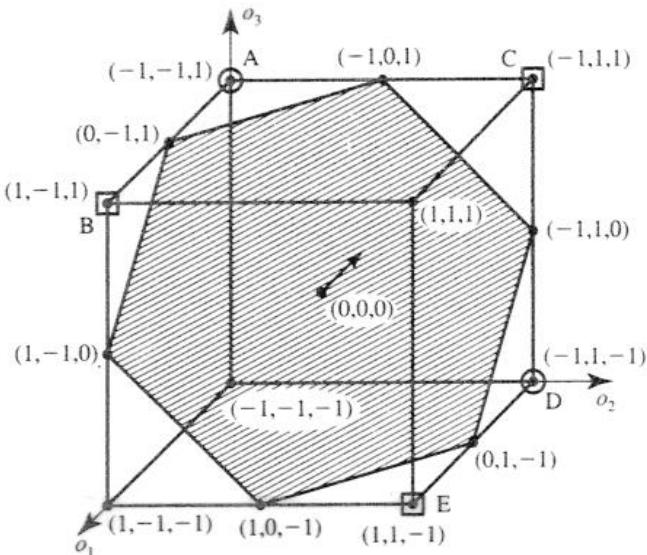
$$o_4 = \begin{cases} sgn(o_1 + o_2 + o_3) > 0 : \text{classe1} \\ sgn(o_1 + o_2 + o_3) < 0 : \text{classe2} \end{cases}$$

espace image  $o \in \{-1,1\}^3 \rightarrow$  espace sortie  $o_4 \in \{-1,1\}$

# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

- ▶ Exemple – Illustration de l'hyperplan de décision du RNA dans l'espace image  $o$  du réseau  $\in \{-1,1\}^3$ :
  - L'espace image  $o$  d'un **RNA multicouche** est le résultat de la transformation du vecteur  $y$  à la sortie du premier étage de neurones



# 1) Classification lin. non-séparables

## Impact de l'ajout d'une couche

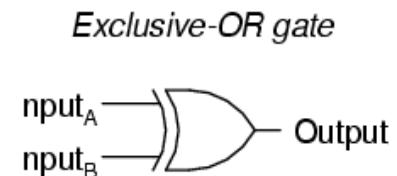
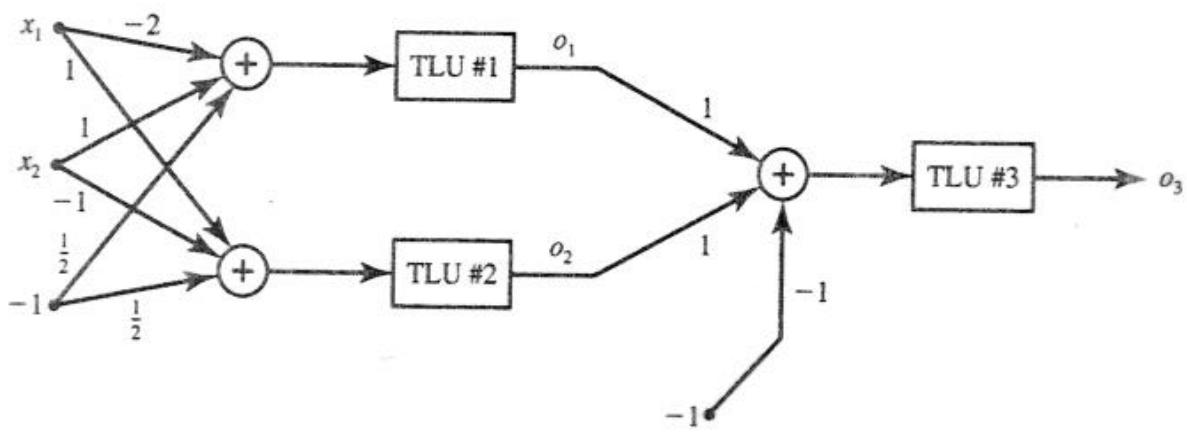
- ▶ **Exemple** – L'acquisition des connaissances (estimation des poids) est effectuée graduellement par le RNA lors de la ‘procédure d’entraînement’
  - Ici, les frontières de décision ont été produites **par inspection** des exemples – des entrées entrées  $y$  et les sorties  $d$  correspondantes de l'ensemble d'apprentissage
  - La procédure d'entraînement d'un réseau multicouche doit produire des images linéairement séparables dans l'espace image  $o$ .

# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

### ► Fonction XOR:

- On ne peut pas résoudre avec une couche de perceptrons
- Les RNA multicouche permettent la résolution du problème de mise en œuvre de cette fonction.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

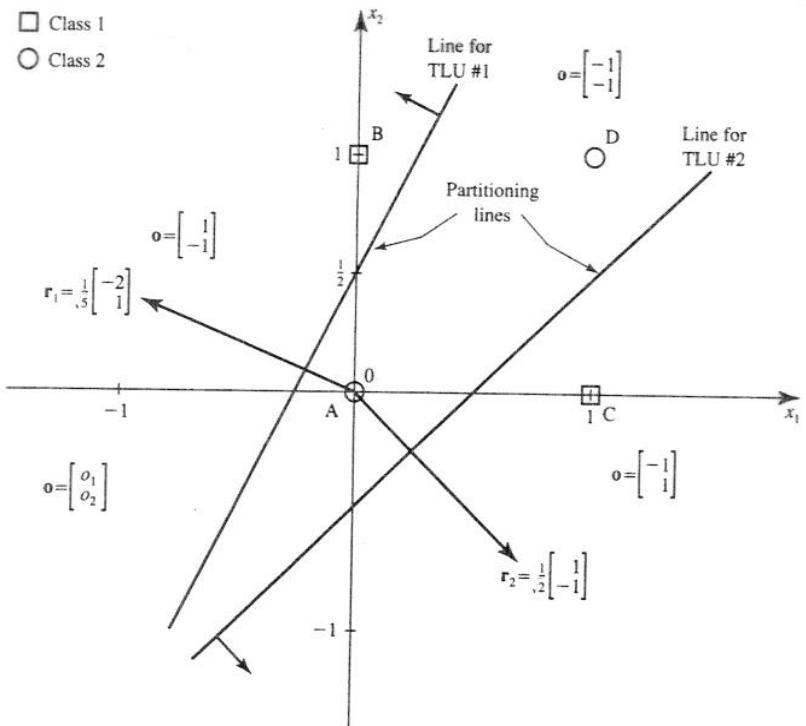
# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

- ▶ Illustration de l'espace données  $x$  du RNA XOR
  - Couche 1: transforme  $y = [x_1 \ x_2 \ -1] \rightarrow o = [o_1 \ o_2]$

### Espace des données $x$ :

On exploite 2 perceptrons bipolaires pour fonctions



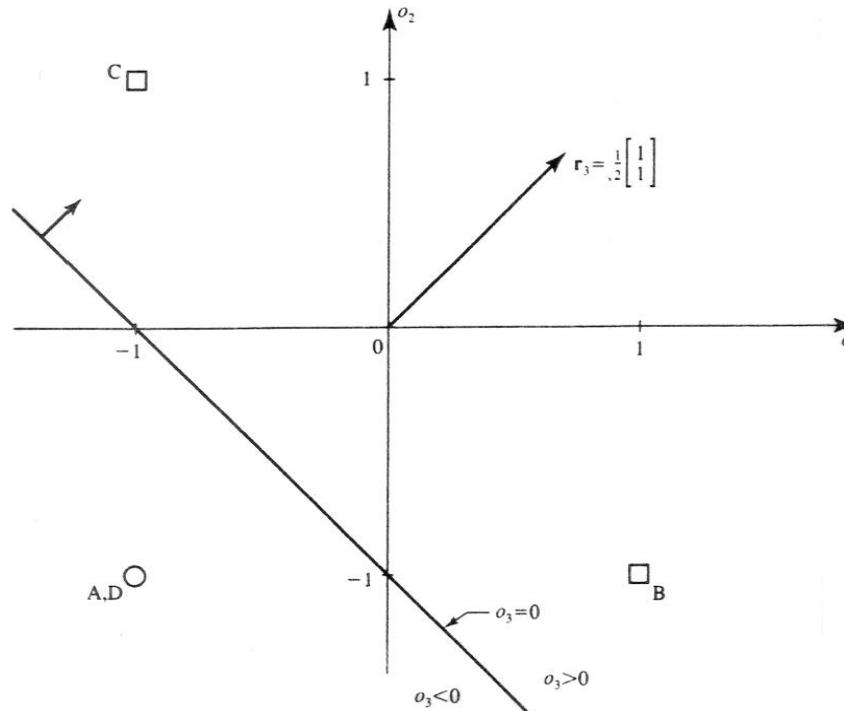
# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

- ▶ Illustration de l'espace image  $\mathbf{o}$  du RNA XOR
  - Couche 2: transform  $\mathbf{o} = [o_1 \ o_2] \rightarrow o_3 = \text{sgn}(o_1 + o_2 + 1)$

### Espace image $\mathbf{o}$ :

Un plan linéaire qui permet de séparer les 2 populations (les 2 classes sont lin séparables dans cet espace)



# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

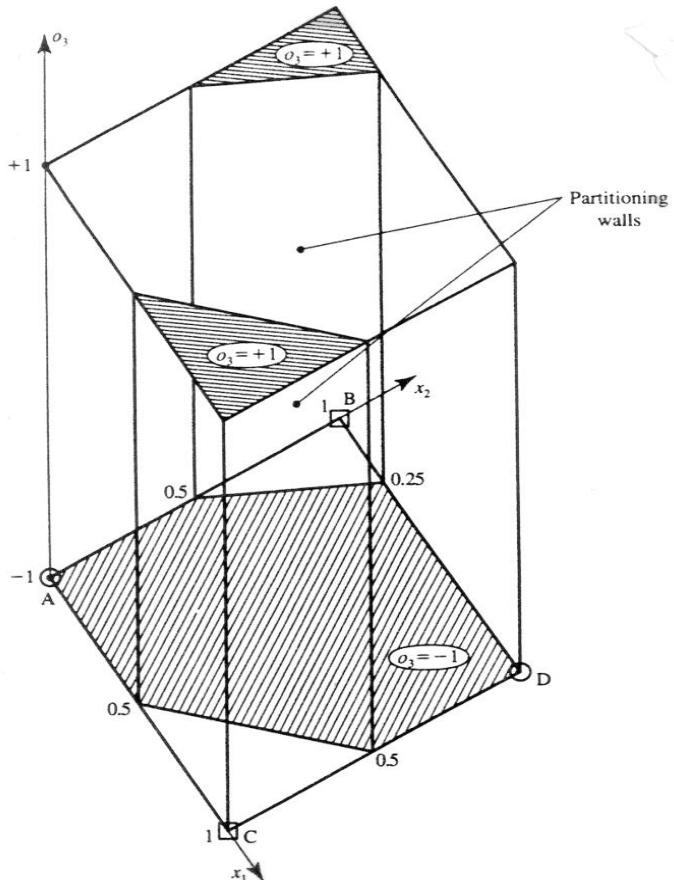
- ▶ **Tableau de classification pour les 2 couches du RNA XOR**
  - Transform  $\mathbf{x} = [x_1 \ x_2] \rightarrow o_3 = \text{sgn}(o_1 + o_2 + 1)$

Symbol	Pattern Space		Image Space		TLU #3 Input $o_1 + o_2 + 1$	Output Space $o_3$	Class Number
	$x_1$	$x_2$	$o_1$	$o_2$			
A	0	0	-1	-1	-	-1	2
B	0	1	1	-1	+	+1	1
C	1	0	-1	1	+	+1	1
D	1	1	-1	-1	-	-1	2

# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

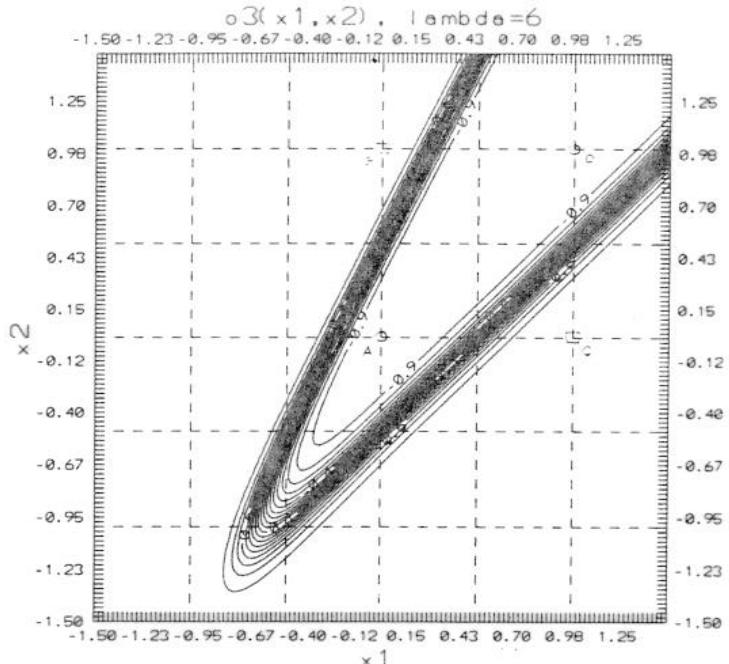
- ▶ Visualisation de la fonction de transfert  $x = [x_1 \ x_2] \rightarrow o_3$  implantée



# 1) Classification lin. non-séparables

## Exemple classique – réseau XOR

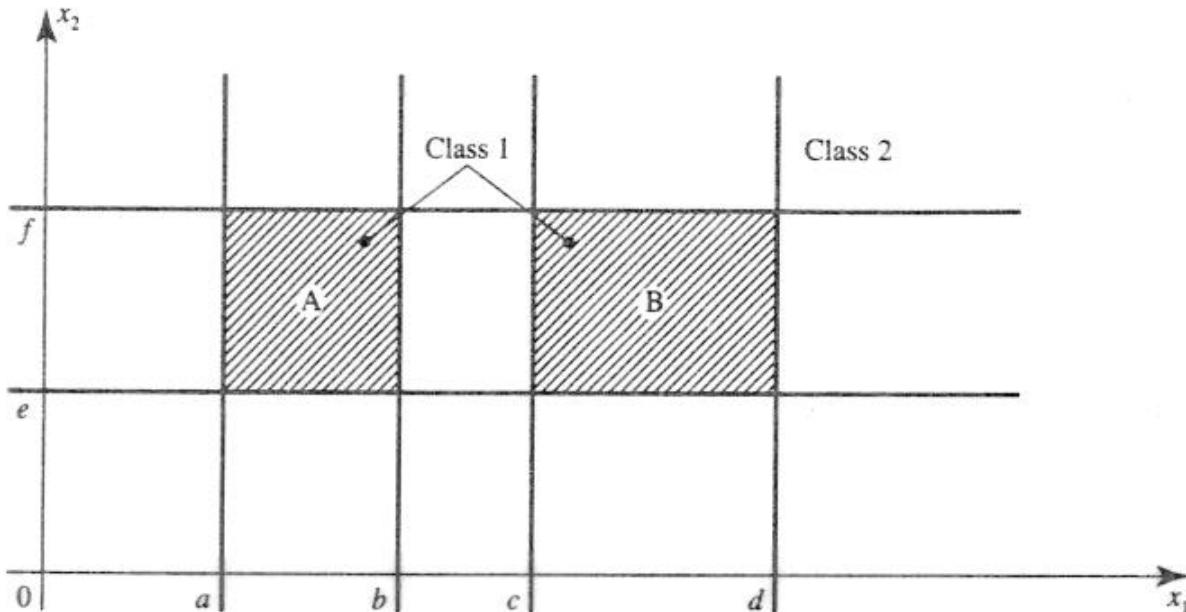
- ▶ **Diagramme de contour:** valeurs de la fonction  $o_3(x_1, x_2)$  estimée par le RNA en remplaçant les 3 perceptrons **discrets** par des perceptrons **continus** bipolaires, avec  $\lambda = 6$ .
  - les poids ont les mêmes valeurs que celles du réseau de perceptrons discrets
  - on remplace seulement la fonction d'activation



# 1) Classification lin. non-séparables

## Architecture de réseau de type *comité*

- ▶ Un application de reconnaissance en 2D plus complexe
  - classe 1: régions A et B (hachurées)
  - classe 2: autres régions (non hachurées)



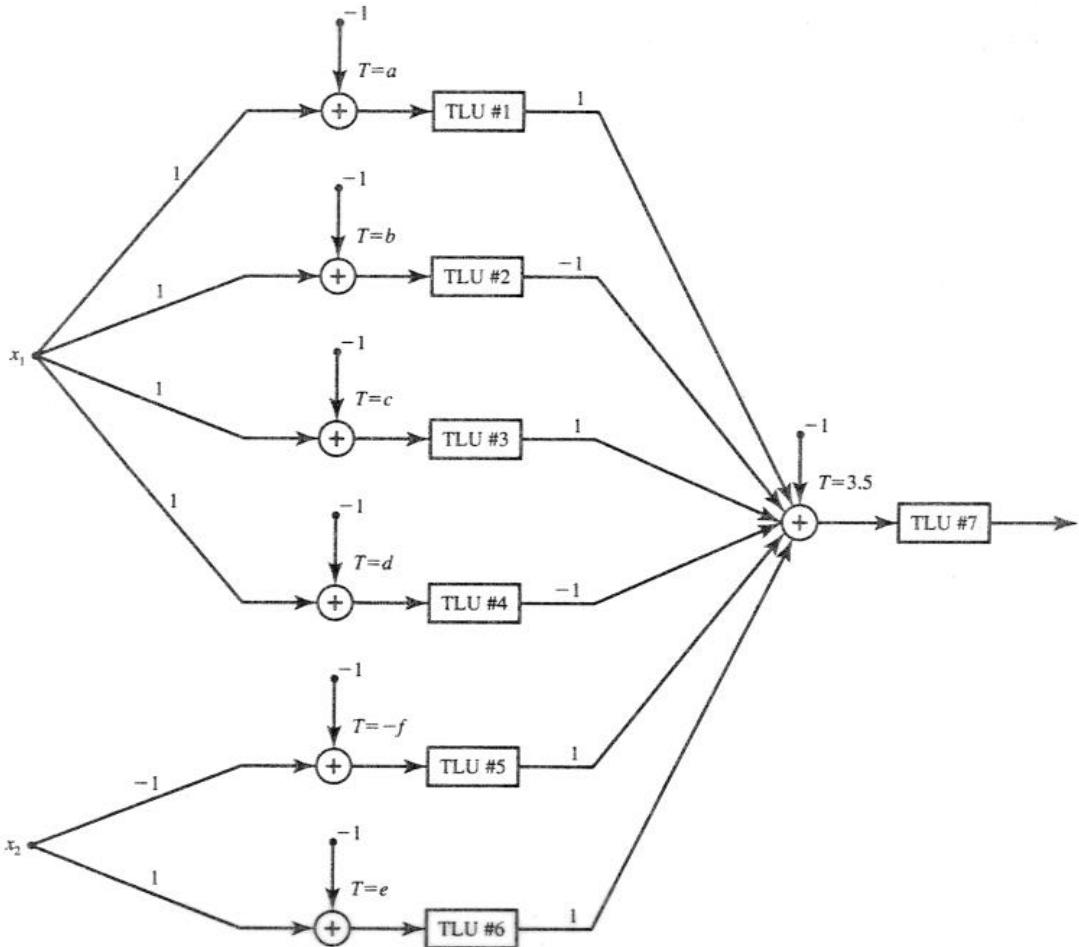
# 1) Classification lin. non-séparables

## Architecture de réseau de type *comité*

- ▶ La mise-en-œuvre du classificateur est réalisée à l'aide de plusieurs perceptrons qui sont appelés à coopérer.
  - **couche 1:** 6 perceptrons (fonctions discriminantes) pour les surfaces de décisions linéaires dans l'espace des données  $y$
  - **couche 2 (voteur):** 1 perceptron (fonction discriminante) pour les surfaces de décisions dans l'espace image  $o$
  - Chaque perceptron (couche 1) prononce une décision et le verdict final va à la classe qui remporte la majorité des votes (à la couche 2)

# 1) Classification lin. non-séparables

## Architecture de réseau de type *comité*



Les connexions avec un poids égal à zéro ne sont pas montrées

Les autres poids sont tous soit  $\pm 1$

Les seuils définissent les frontières de décision

# CONTENU DU COURS

## A.3 Réseaux de neurones multicouches sans rétroaction (MLP)

- 1) Classification de données linéairement non-séparables
- 2) Règle d'apprentissage delta pour une couche de perceptrons
- 3) Règle d'apprentissage delta généralisée
- 4) Entraînement par la rétro-propagation des erreurs
- 5) Facteurs d'apprentissage

## 2) Règle delta – couche de perceptrons

### Entraînement par rétro propagation des erreurs

- ▶ **Règle DELTA pour une couche de perceptron:** connue sous le nom *d'algorithme d'entraînement par la rétro-propagation des erreurs*
  - *Error Back-Propagation Training Algorithm*
  - Règle de base pour d'apprentissage des poids liés à une **couche de sortie** de perceptrons continues
  - L'algorithme d'entraînement par la rétro-propagation des erreurs permet au réseau multicouche d'acquérir par expérience la connaissance de la fonction de transfert (mapping) entre les patrons entrées  $y$  et sorties  $\mathbf{o}$ .

## 2) Règle delta – couche de perceptrons

### Entraînement par rétro propagation des erreurs

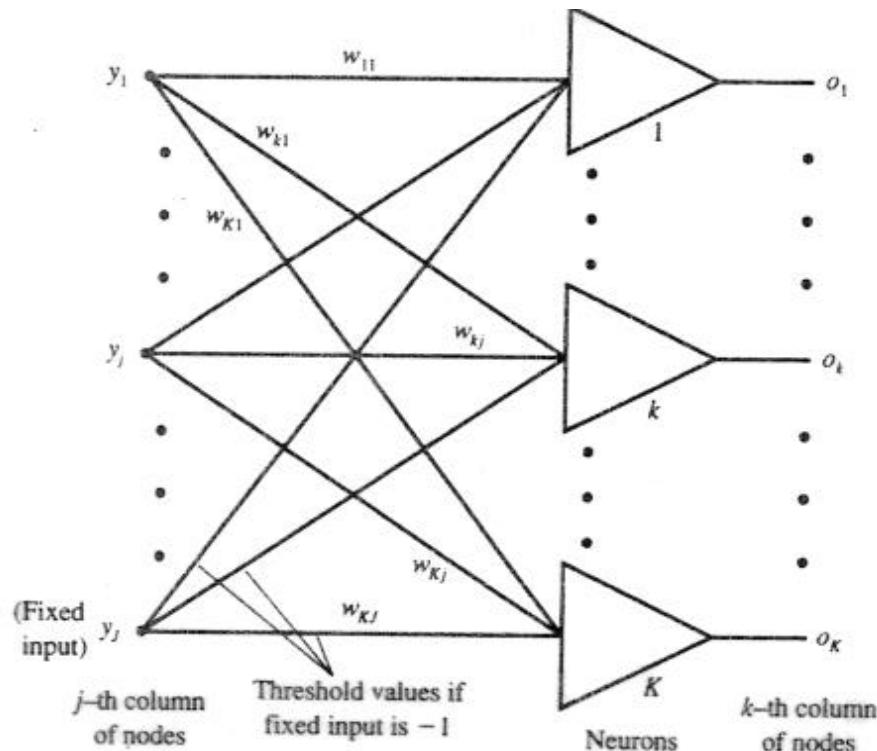
#### ► Apprentissage basée sur l'erreur:

- Si la classification (rappel) d'un patron présenté au RNA est erronée, alors les poids synaptiques et les seuils sont ajustés de telle sorte que **l'erreur quadratique moyen de classification** évaluée pour ce patron soit diminuée:
- Fonction d'erreur à minimiser:  $E = \sum_i(o_i - d_i)^2$
- Le processus d'apprentissage continue pour tous les patrons de l'ensemble d'entraînement, et termine lorsque la valeur de l'erreur totale évaluée sur l'ensemble est satisfaisante

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- Architecture d'un réseau monocouche composé de  $K$  de perceptrons continus:



## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

► Ajustement des poids synaptiques d'un RNA monocouche de perceptrons continus:

- Le vecteur de poids lié au neurone  $k$ :  $\mathbf{w}_k = [w_{k1}, w_{k2}, \dots, w_{kJ}]^t$

où le poids  $w_{kj}$  est attribué à la connexion qui relie la sortie du  $j^{\text{ième}}$  noeud de la couche d'entrée à l'entrée du  $k^{\text{ième}}$  neurone de la couche de sortie

- la matrice des poids synaptiques:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1J} \\ w_{21} & w_{22} & \dots & w_{2J} \\ \vdots & \vdots & & \ddots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & & \ddots \\ w_{K1} & w_{K2} & \dots & w_{KJ} \end{bmatrix}$$

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- ▶ Ajustement des poids synaptiques d'un RNA monocouche de perceptrons continus:
  - Phase de rappel: le vecteur de sortie  $\mathbf{o}$  du réseau est obtenu par la relation (notation vectorielle):
$$\mathbf{o} = \Gamma[\mathbf{W}\mathbf{y}]^t = \Gamma[\mathbf{net}]^t$$
  - Pour chaque neurone de la couche de sortie  $k = 1, 2, \dots, K$ , l'activation  $net_k$  peut être exprimé par:

$$net_k = \mathbf{w}_k^t \mathbf{y} = \sum_{j=1}^J w_{kj} y_j$$

et la sortie du neurone  $k$  est égale à:  $o_k = f(net_k)$

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- Ajustement des poids synaptiques d'un RNA monocouche de perceptrons continus:
- les vecteurs d'entrées  $y$ , de sorties  $\mathbf{o}$ , l'opérateur nonlinéaire  $\Gamma$ , et le vecteur des sorties désirées  $d$  sont définis par:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_J \end{bmatrix} \quad \Gamma[\cdot] = \begin{bmatrix} f(\cdot) & 0 & \dots & 0 \\ 0 & f(\cdot) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f(\cdot) \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ \vdots \\ o_K \end{bmatrix} \quad d \doteq \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_K \end{bmatrix}$$

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- ▶ Une **descente de gradient** est effectuée pour diminuer l'erreur  $E_p$  par l'ajustement des poids synaptiques:
  - L'erreur de classification engendrée par la présentation du patron  $y_p$  de l'ensemble d'entraînement à l'entrée du réseau est évaluée en tenant compte des  $K$  neurones de sortie, alors:

$$E_p = \frac{1}{2} \sum_{k=1}^K (d_{pk} - o_{pk})^2 = \frac{1}{2} \|\mathbf{o}_p - \mathbf{d}_p\|^2$$

- La valeur des seuils  $T_k$ , pour  $k = 1, 2, \dots, K$ , sont ajustables au même titre que tous les autres poids, ce qui implique que:

$$w_{kJ} = T_k \text{ pour } k = 1, 2, \dots, K$$

et que la valeur  $y_J$  est fixée à la valeur  $y_J = -1$  pour les phases d'entraînement et de rappel.

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- ▶ Avec la règle delta, l'ajustement individuel des poids  $w_{kj}$  a lieu séquentiellement, lors de la présentation d'un patron  $y_p$ :
  - Stratégie: afin de minimiser la fonction d'erreur  $E_p$ , les poids sont ajustés dans la direction négative du gradient d'erreur:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

(Pour simplifier la notation, on remplace  $E_p$  par  $E$  dans les dérivations futurs.)

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- Le signal d'erreur produit par le neurone  $k$ , appelée  $\delta_{ok}$ , peut être défini pour cette couche de neurones:
- La variation de  $E_p$  en fonction de  $net_k$ :  $\delta_{ok} \doteq - \frac{\partial E}{\partial (net_k)}$
  - **Règle de delta pour un RNA monocouche:** l'ajustement des poids synaptiques peut être reformulé en terme de ce signal d'erreur  $\delta_{ok}$  :

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} = \eta \delta_{ok} y_j$$

pour  $k = 1, 2, \dots, K$  et pour  $j = 1, 2, \dots, J$

## 2) Règle delta – couche de perceptrons

### Formulation générale de la règle delta

- ▶ En effet, la composante  $\partial E / \partial w_{kj}$  dépend seulement du terme  $net_k$  d'un seul neurone, car l'erreur à la sortie du  $k^{\text{ième}}$  neurone est uniquement fonction des poids synaptiques  $w_{kj}$ , pour  $j = 1, 2, \dots, J$  du neurone  $k$ 
  - Selon la règle d'enchainement pour les dérivés partielles, on peut décomposer  $\partial E / \partial w_{kj}$  comme:

$$\partial E / \partial w_{kj} = \partial E / \partial (net_k) \cdot \partial (net_k) / \partial w_{kj}$$

- Étant donné que  $\partial (net_k) / \partial w_{kj} = \partial (\mathbf{w}_k^T \mathbf{y}) / \partial w_{kj} = y_j$ , alors

$$\partial E / \partial w_{kj} = -\delta_{ok} y_j$$

## 2) Règle delta – couche de perceptrons

### Formulation selon la fonction d'activation

- Par contre, la valeur de  $\delta_{ok}$  doit être formulée en termes de chaque type de fonction d'activation.
- Étant donné que  $E = E[o_k(net_k)]$  pour chaque neurone  $k$ , et:

$$\delta_{ok} = - \frac{\partial E}{\partial (net_k)} = - \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial (net_k)}$$

- **1<sup>e</sup> terme:** représente l'erreur local,  $\frac{\partial E}{\partial o_k} = -(d_k - o_k)$
- **2<sup>er</sup> terme:** représente un facteur de gain ajustable – la dérivée (pente) de la fonction d'activation du neurone  $k$

$$\frac{\partial o_k}{\partial (net_k)} = f'_k(net_k)$$

## 2) Règle delta – couche de perceptrons

### Formulation selon la fonction d'activation

► On retrouve l'équation générale pour le signal d'erreur  $\delta_{ok}$ :

$$\delta_{ok} = (d_k - o_k) f'_k(\text{net}_k)$$

- Cette équation indique que le signal d'erreur  $\delta_{ok}$  reflète l'erreur locale ( $d_k - o_k$ ) évaluée à la sortie du  $k^{\text{ième}}$  neurone, pondérée par le facteur multiplicatif  $f'_k(\text{net}_k)$  qui représente la pente de la fonction d'activation évaluée pour la valeur d'activation  $\text{net}_k$
- **Règle delta:** La formulation finale pour l'ajustement des poids d'un réseau de neurones monocouche se résume à:

$$w'_{kj} = w_{kj} + \Delta w_{kj} = w_{kj} + \eta (d_k - o_k) f'_k(\text{net}_k) y_j$$

pour  $k = 1, 2, \dots, K$ , et  $j = 1, 2, \dots, J$

## 2) Règle delta – couche de perceptrons

### Cas 1: Fonction d'activation unipolaire

► Soit la fonction d'activation unipolaire continue:

$$f(\text{net}) \doteq \frac{1}{1 + \exp(-\lambda \text{net})}, \quad \text{avec } \lambda > 0$$

– Si  $\lambda = 1$ , la dérivée de la fonction d'activation correspond à

$$f'(\text{net}) = \frac{\exp(-\text{net})}{[1 + \exp(-\text{net})]} = \frac{1}{1 + \exp(-\text{net})} \cdot \frac{1 + \exp(-\text{net}) - 1}{1 + \exp(-\text{net})}$$

– Puisque est équivalent à  $f(\text{net}) = o$  et  $f'(\text{net}) = o(1-o)$ , la valeur de delta devient égale à:

$$\delta_{ok} = (d_k - o_k)o_k(1 - o_k)$$

## 2) Règle delta – couche de perceptrons

### Cas 1: Fonction d'activation unipolaire

- La mise à jour des poids pour ce qui est de la règle d'apprentissage delta se résume à:
- Cas A: fonction d'activation unipolaire continue

$$w'_{kj} = w_{kj} + \eta (d_k - o_k) o_k (1 - o_k) y_j$$

$$\text{pour } o_k = \frac{1}{1 + \exp(\text{net}_k)}$$

avec  $k = 1, 2, \dots, K$ , et  $j = 1, 2, \dots, J$

## 2) Règle delta – couche de perceptrons

### Cas 2: Fonction d'activation bipolaire

► Soit la fonction d'activation bipolaire continue:

$$f(net) \doteq \frac{2}{1 + \exp(-\lambda net)} - 1 \quad \text{avec} \quad \lambda > 0$$

- Si  $\lambda = 1$ , la dérivée de la fonction d'activation correspond à

$$f'(net) = \frac{2 \exp(-net)}{[1 + \exp(-net)]^2} = \frac{2 \exp(-net)}{[1 + \exp(-net)]^2}$$

- Puisque est équivalent à  $f(net) = o$  et  $f'(net) = \frac{1}{2}(1-o^2)$ , la valeur de delta devient égale à:

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)(1 - o_k^2)$$

## 2) Règle delta – couche de perceptrons

### Cas 2: Fonction d'activation bipolaire

- La mise à jour des poids pour ce qui est de la règle d'apprentissage delta se résume à:
- Cas B: fonction d'activation bipolaire continue

$$w'_{kj} = w_{kj} + \frac{1}{2} \eta (d_k - o_k)(1 - o_k^2)y_j$$

$$\text{pour } o_k = 2 \left[ \frac{1}{1 + \exp(-net_k)} - \frac{1}{2} \right]$$

avec  $k = 1, 2, \dots, K$ , et  $j = 1, 2, \dots, J$

## 2) Règle delta – couche de perceptrons

### Forme vectorielle compact

- La mise à jour des poids pour un réseau monocouche de type perceptron continu peut être exprimée par:

$$\mathbf{W}' = \mathbf{W} + \eta \delta_o \mathbf{y}^t$$

- Le signal d'erreur est un vecteur  $\delta_o$ , avec une composante  $\delta_{ok}$  par neurone  $k = 1, 2, \dots, K$

$$\delta_o \doteq \begin{bmatrix} \delta_{o1} \\ \delta_{o2} \\ \vdots \\ \vdots \\ \delta_{oK} \end{bmatrix}$$

## 2) Règle delta – couche de perceptrons

### Algorithme d'apprentissage pour les réseaux monocouches de perceptrons continus

(*Multiclass Continuous Perceptron Training Algorithm: MCPTA*)

#### ► On suppose:

- l'ensemble d'entraînement constitué de  $P$  vecteurs d'apprentissage  $\mathbf{y}_p$  et de leur classe d'appartenance  $\mathbf{d}_p$ , pour  $p = 1, 2, \dots, P$  :  
$$\{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \dots, (\mathbf{x}_P, \mathbf{d}_P)\},$$
où  $\mathbf{y}_p$  est de dimension ( $J \times 1$ ) et  $\mathbf{d}_i$  de dimension ( $K \times 1$ )
- Espace augmenté: chaque  $J^{\text{ième}}$  composante des vecteurs  $\mathbf{y}_p$  a pour valeur -1.
- $q$  : le numéro de l'itération d'apprentissage (au total)
- $p$  : compteur de patron à l'intérieur d'un cycle ou epoch d'entraînement

## 2) Règle delta – couche de perceptrons

### Algorithme d'apprentissage pour les réseaux monocouches de perceptrons continus

#### ► Initialisation:

- **Étape 1:** Choix du taux d'apprentissage  $\eta > 0$  et du seuil d'erreur  $E_{\max} > 0$
- **Étape 2:** La matrice de poids  $\mathbf{W} = [\mathbf{w}_{kj}]$  de dimension  $(K \times J)$  est initialisée aléatoirement avec des petites valeurs, et  $k \leftarrow 1$ ,  $p \leftarrow 1$  et  $E \leftarrow 0$

#### ► Cycle d'apprentissage

- **Étape 3:** Un vecteur  $\mathbf{y}$  est présenté au réseau et les sorties sont calculées.

$$\mathbf{y} \leftarrow \mathbf{y}_p, \mathbf{d} \leftarrow \mathbf{d}_p \text{ et } o_k \leftarrow f(\mathbf{w}_k^T \mathbf{y}) \text{ pour } k = 1, 2, \dots, K$$

$\mathbf{w}_k$  est la  $k^{\text{ième}}$  rangée de  $\mathbf{W}$  et  $f(\text{net})$  est la fonction d'activation continue bipolaire

- **Étape 4:** Les poids sont ajustés

$$\mathbf{w}_k \leftarrow \mathbf{w}_k + \frac{1}{2} \eta (d_k - o_k)(1 - o_k^2) \mathbf{y}, \text{ pour } k = 1, 2, \dots, K$$

## 2) Règle delta – couche de perceptrons

### Algorithme d'apprentissage pour les réseaux monocouches de perceptrons continus

#### ► Cycle d'apprentissage (suite)

- **Étape 5:** Mise à jour de l'erreur accumulée du cycle d'apprentissage (la fonction de coût à minimiser):

$$E \leftarrow \frac{1}{2} (d_k - o_k)^2 + E, \quad \text{pour } k = 1, 2, \dots, K$$

- **Étape 6:** Si  $p < P$ , alors  $q \leftarrow q + 1, p \leftarrow p + 1$  et aller à l'étape 3, sinon autrement aller à l'étape 7.
- **Étape 7:** Le cycle d'apprentissage est complété
  - Si  $E < E_{max}$ : l'entraînement est terminé, et conserver les  $W, q$  et  $E$ .
  - Si  $E > E_{max}$ : alors  $E \leftarrow 0, q \leftarrow q + 1, p \leftarrow 1$  et commencer un nouveau cycle d'apprentissage à l'étape 3.

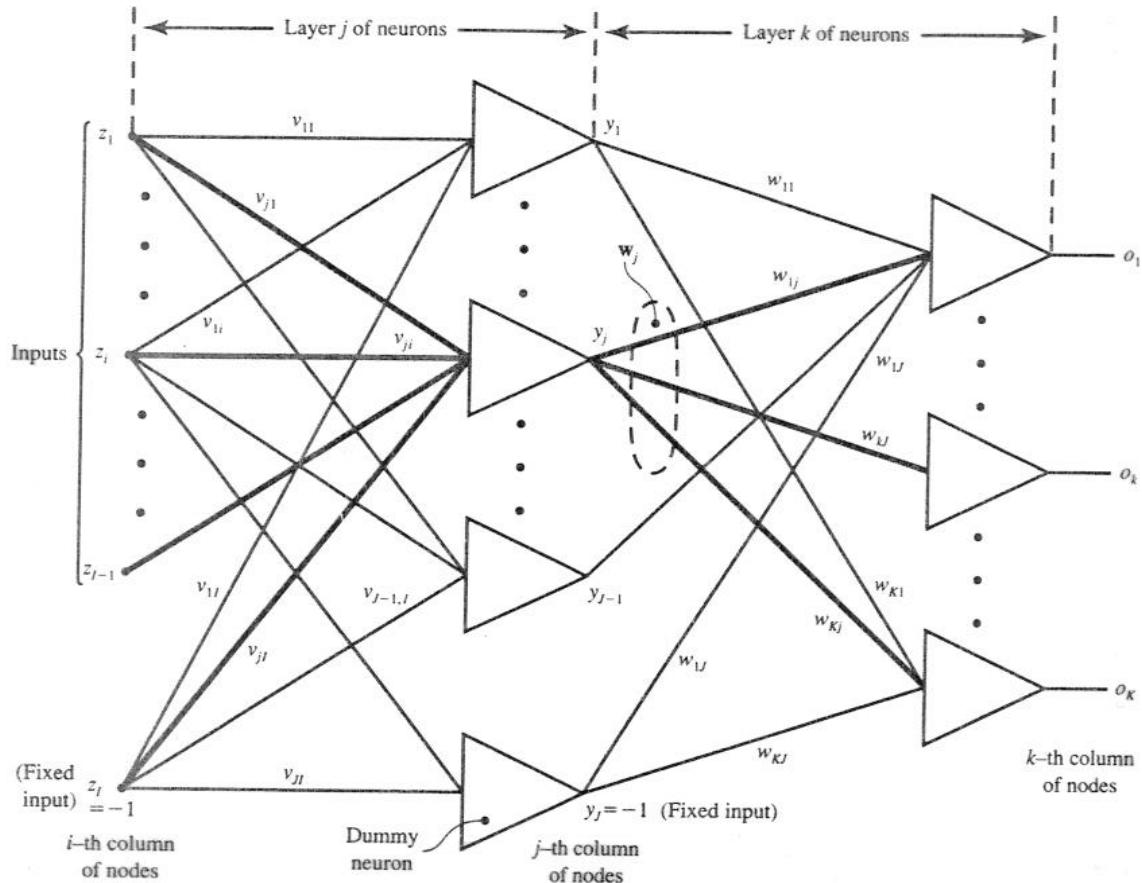
# CONTENU DU COURS

## A.3 Réseaux de neurones multicouches sans rétroaction (MLP)

- 1) Classification de données linéairement non-séparables
- 2) Règle d'apprentissage delta pour une couche de perceptrons
- 3) Règle d'apprentissage delta généralisée**
- 4) Entrainement par rétro-propagation des erreurs
- 5) Facteurs d'apprentissage

### 3) Règle delta généralisée

## Rétropropagation dans un RNA avec couche cachée



### 3) Règle delta généralisée

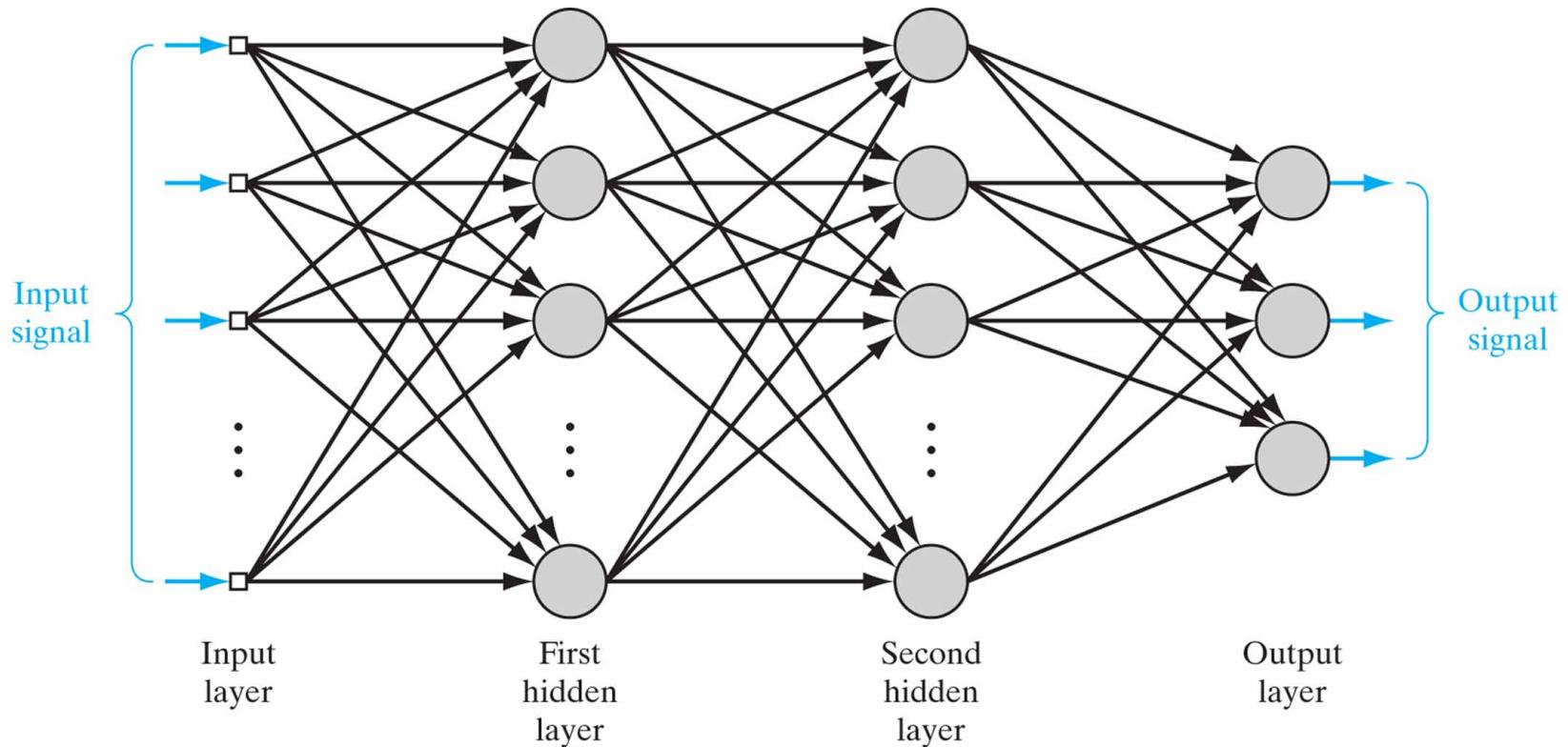
#### Rétropropagation dans un RNA avec couche cachée

- ▶ **Nombre de couches d'un RNA**  $\equiv$  le nombre de couches composées de neurones actifs, excluant la couche d'entrée qui sert seulement à distribuer les entrées aux neurones de la couche suivante
- ▶ **La règle *delta généralisée* pour l'apprentissage supervisé pour les poids synaptique V**
  - diffère dans la façon d'évaluer le signal d'erreur  $\delta$
  - cette règle propage l'erreur vers l'arrière d'une couche, et la même procédure d'évaluation est répétée pour la prochaine couche cachée

### 3) Règle delta généralisée

#### Rétropropagation dans un RNA avec couche cachée

► Un réseau MLP avec 2 couches cachées:



### 3) Règle delta généralisée

#### Rétropropagation dans un RNA avec couche cachée

##### ► Phases de rappel:

- En général, la sortie du réseau est le résultat de l'implication (*mapping*) du patron d'entrées  $z$  vers la sortie  $o$ :  $o = N[z]$
- Pour un RNA composé de deux couches, l'implication  $z \rightarrow o$  peut être représentée par la relation:

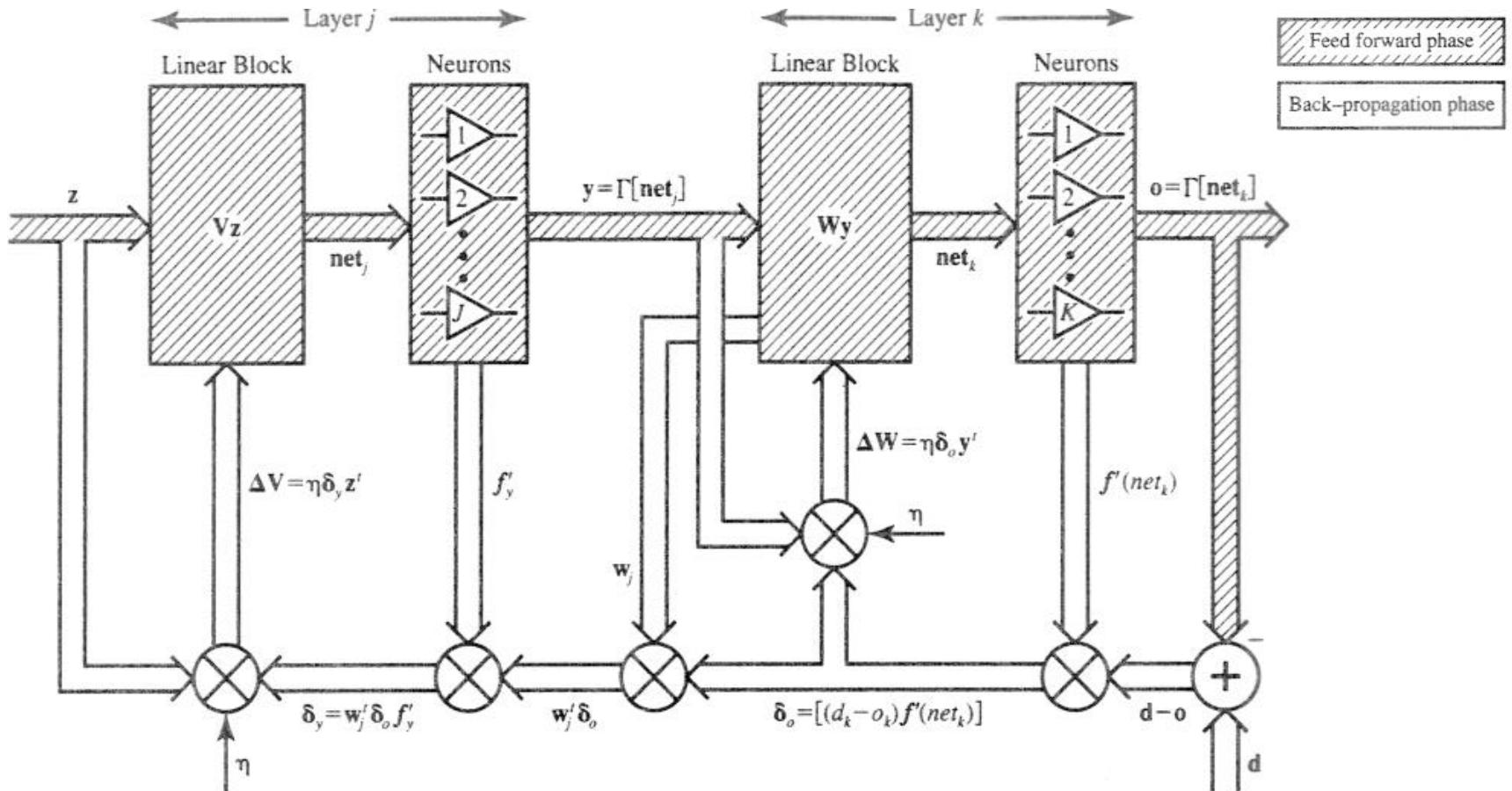
$$o = \Gamma[W \Gamma[V z]] = \Gamma[W y]$$

car l'implication imbriquée qui est liée à la couche cachée  $z \rightarrow y$  est représentée par la relation:

$$y = \Gamma[V z]$$

### 3) Règle delta généralisée

- Illustration de la rétro-propagation des erreurs pour un réseau de neurones à deux couches:



### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ Expression générale de l'ajustement  $\Delta v_{ji}$  des poids de neurones localisés sur une couche cachée.
  - Formule de la descente négative du gradient:

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}},$$

pour  $j = 1, 2, \dots, J$  et  $i = 1, 2, \dots, I$

avec  $\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial(\text{net}_j)} \cdot \frac{\partial(\text{net}_j)}{\partial v_{ji}} = -\delta_{yj} z_i$

avec le patron d'entrée du réseau  $\mathbf{z} = (z_1, z_2, \dots, z_I)$ .

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ **Expression générale de l'ajustement  $\Delta v_{ji}$  des poids:**
  - Après remplacement, l'ajustement des poids synaptiques de la couche cachée se résume à:
$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}} = \eta \delta_{yj} z_i$$
pour  $i = 1, 2, \dots, I$  et pour  $j = 1, 2, \dots, J$ ,
  - Le terme  $\delta_{yj}$  représente le signal d'erreur de la couche cachée ayant comme sortie  $y$ . Pour le  $j^{\text{ième}}$  neurone de la couche cachée, et il est égale à :
$$\delta_{yj} \doteq - \frac{\partial E}{\partial (\text{net}_j)}$$

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ **Expression générale de l'ajustement  $\Delta v_{ji}$  des poids:**
- Contrairement à l'excitation  $net_k$  qui affecte seulement la sortie du  $k^{\text{ième}}$  neurone de sortie,  $net_j$  contribue maintenant à chacune des composantes dans la sommation des erreurs évaluées à la sortie du réseau par l'équation: 
$$E = \frac{1}{2} \sum_{k=1}^K (d_k - o_k)^2 = \frac{1}{2} \| d - o \|^2$$
- Conséquemment, le signal d'erreur au noeud  $j$  peut être évalué par : 
$$\delta_{yj} = - \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (net_j)} \quad \text{avec}$$
 
$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} \left[ \frac{1}{2} \sum_{k=1}^K \{d_k - f[net_k(y)]\}^2 \right]$$
 
$$\frac{\partial y_j}{\partial (net_j)} = f'_j(net_j)$$

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ Expression générale de l'ajustement  $\Delta v_{ji}$  des poids:
  - Après manipulation des termes, la variation de  $E$  par rapport à  $y_j$ :

$$\begin{aligned}
 \frac{\partial E}{\partial y_j} &= - \sum_{k=1}^K (d_k - o_k) \frac{\partial}{\partial y_j} \{ f[net_k(y)] \} \\
 &= - \sum_{k=1}^K (d_k - o_k) f'(net_k) \frac{\partial (net_k)}{\partial y_j} \\
 &= - \sum_{k=1}^K \delta_{ok} w_{kj}
 \end{aligned}$$

- Alors, la formulation du signal d'erreur  $\delta_{yj}$  au nœud  $j$  devient, après remplacement, égale à :

$$\delta_{yj} = - \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial (net_j)} = f'_j(net_j) \sum_{k=1}^K \delta_{ok} w_{kj}, \quad \text{pour } j = 1, 2, \dots, J$$

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- **Règle d'apprentissage delta généralisée :**
  - L'ajustement des poids dans la couche cachée se résume à:

$$\Delta v_{ji} = \eta f'(net_j) z_i \sum_{k=1}^K \delta_{ok} w_{kj},$$

pour  $j = 1, 2, \dots, J, \text{ et } i = 1, 2, \dots, I$

où  $f'(net_j)$  est évaluée selon le choix de la fonction d'activation

- **Interprétation:** L'ajustement des poids du neurone  $j$  de la couche cachée est proportionnel à la somme pondérée de toutes les valeurs  $\delta_{ok}$  de la couche de adjacente (où les nœuds sont connectés à la sortie du neurone  $j$ ).
- Toutes les erreurs évaluées à la sortie du réseau, soit  $\delta_{ok} w_{kj}$  contribuent à l'ajustement des poids  $v_{ji}$  de la couche cachée

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ Les poids synaptiques de la couche cachée sont modifiés par la relation suivante

$$\Delta v_{ji} = \eta f'_j(\text{net}_j) z_i \sum_{k=1}^J \delta_{ok} w_{kj},$$

pour  $j = 1, 2, \dots, J$ , et  $i = 1, 2, \dots, I$

ou sous la forme compacte vectorielle :

$$V' = V + \eta \delta_y z^t$$

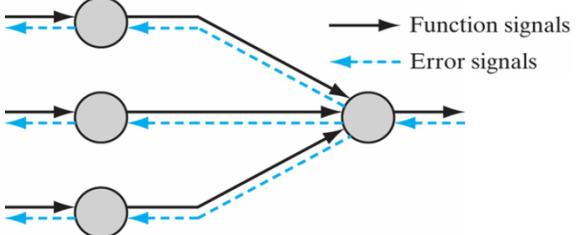
- avec le patron d'entrée  $z^t = (z_1, z_2, \dots, z_I)$ ,
- le signal d'erreur  $\delta_y^t = (\delta_{y1}, \delta_{y2}, \dots, \delta_{yJ})$ ,
- et la matrice de poids:

$$V = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1J} \\ v_{21} & v_{22} & \dots & v_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ v_{K1} & v_{K2} & \dots & v_{KJ} \end{bmatrix}$$

### 3) Règle delta généralisée

#### L'ajustement des poids couche cachée

- ▶ La règle d'apprentissage delta généralisée permet à la couche cachée de propager l'erreur vers l'arrière par une couche



- ▶ La  $j^{\text{ième}}$  colonne de la matrice  $W$  est  $w_j$ , alors le vecteur  $\delta_y$  peut être exprimé par:  $\delta_y = w_j^t \delta_o f'_y$

où  $f'_y$  est le vecteur colonne composé des éléments  $f'_{yj}$  exprimé pour chaque neurone de la couche cachée, pour une fonction d'activation continue unipolaire:

$$f'_{yj} = y_j(1-y_j)$$

ou continue bipolaire:

$$f'_{yj} = \frac{1}{2} (1-y_j^2)$$

### 3) Règle delta généralisée

#### Synthèse

- ▶ La principale différence entre la règle d'apprentissage *delta* et la règle d'apprentissage *delta généralisée* est dans la façon dont le signal d'erreur  $\delta$  est évalué:
  - **Poids de la couche de sortie:** Pour ce qui est de la *règle d'apprentissage delta*, le vecteur  $\delta_o$  est simplement la différence entre la sortie actuelle et la sortie désirée, multipliée par la dérivée de la fonction d'activation.
  - **Poids de la couche cachée:** Pour ce qui est de la *règle d'apprentissage delta généralisée*, le vecteur  $\delta_y$  est composé des éléments  $w_j^t \delta_o f'_{yj}$  qui représentent la somme pondérée des signaux d'erreur  $\delta_o$  produit par la couche de neurones suivante

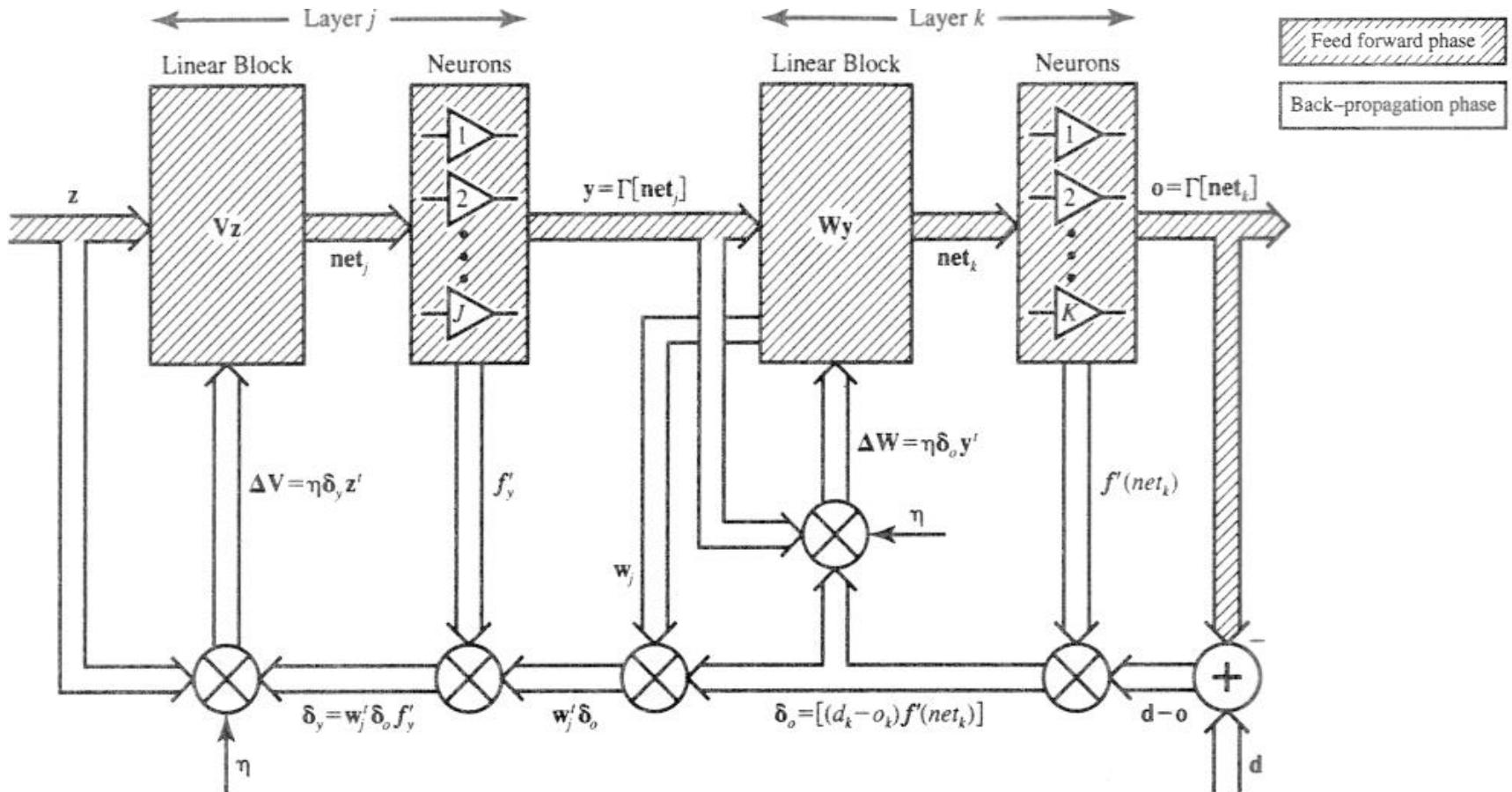
# CONTENU DU COURS

## A.3 Réseaux de neurones multicouches sans rétroaction (MLP)

- 1) Classification de données linéairement non-séparables
- 2) Règle d'apprentissage delta pour une couche de perceptrons
- 3) Règle d'apprentissage delta généralisée
- 4) Entrainement par rétro-propagation des erreurs**
- 5) Facteurs d'apprentissage

## 4) Rétro-propagation des erreurs

- Illustration de la rétro-propagation des erreurs pour un réseau de neurones à deux couches:



# 4) Rétro-propagation des erreurs

## Algorithme d'entraînement par la rétro-propagation des erreurs

*(Error Back-Propagation Training Algorithm : EBPTA)*

► **On suppose:**

- l'ensemble d'entraînement constitué de  $P$  vecteurs d'apprentissage  $\mathbf{y}_p$  et de leur classe d'appartenance  $\mathbf{d}_p$ , pour  $p = 1, 2, \dots, P$  :

$$\{(z_1, \mathbf{d}_1), (z_2, \mathbf{d}_2), \dots, (z_P, \mathbf{d}_P)\},$$

où  $z_p$  est de dimension ( $I \times 1$ ) et  $\mathbf{d}_i$  de dimension ( $K \times 1$ )

- Espace augmenté: chaque  $I^{\text{ième}}$  composante des vecteurs  $z_p$  a pour valeur  $z_I = -1$ . Chaque  $J^{\text{ième}}$  composante des vecteurs  $\mathbf{y}$  (représentant les sorties de la couche cachée) a pour valeur  $y_J = -1$
- $\mathbf{y}$  est de dimension ( $J \times 1$ ) et  $\mathbf{o}$  est de dimension ( $K \times 1$ )
- $q$  : le numéro de l'itération d'apprentissage (au total)
- $p$  : compteur à l'intérieur d'un cycle ou epoch d'entraînement

# 4) Rétro-propagation des erreurs

## Algorithme d'entraînement par la rétro-propagation des erreurs

### ► Étape 1 – initialisation:

- Choix du taux d'apprentissage  $\eta > 0$  et du seuil d'erreur  $E_{\max} > 0$
- Les matrices de poids  $W = [\mathbf{w}_{kj}]$  de dimension  $(K \times J)$  et  $V = [\mathbf{v}_{ji}]$  de dimension  $(J \times I)$  sont initialisées aléatoirement avec des petites valeurs
- $k \leftarrow 1, p \leftarrow 1$  et  $E \leftarrow 0$
- Une sélection aléatoire des vecteurs de l'ensemble d'entraînement donne habituellement les meilleurs résultats.

### ► Cycle d'apprentissage

- **Étape 2:** Le vecteur  $\mathbf{z}_p$  est présenté à l'entrée du réseau et les sorties sont des couches sont évaluées:

$$\mathbf{z} \leftarrow \mathbf{z}_p, \mathbf{d} \leftarrow \mathbf{d}_p$$

$y_j \leftarrow f(\mathbf{v}_j^T \mathbf{z})$  pour  $j = 1, 2, \dots, J$ , où  $\mathbf{v}_j$  est la  $j^{\text{ième}}$  rangée de  $V$

$o_k \leftarrow f(\mathbf{w}_k^T \mathbf{y})$  pour  $k = 1, 2, \dots, K$  où  $\mathbf{w}_k$  est la  $k^{\text{ième}}$  rangée de  $W$

## 4) Rétro-propagation des erreurs

### Algorithme d'entraînement par la rétro-propagation des erreurs

#### ► Cycle d'apprentissage (suite)

- **Étape 3:** L'erreur effectuée par le réseau est évaluée et accumulée

$$E \leftarrow \frac{1}{2} (d_k - o_k)^2 + E, \quad \text{pour } k = 1, 2, \dots, K$$

- **Étape 4:** Les vecteurs  $\delta_o$  et  $\delta_y$  associés aux signaux d'erreur de la couche de sortie et de la couche cachée sont évalués (en ordre inversé des calculs):

- lorsque la fonction d'activation continue bipolaire est utilisée:

$$\delta_{ok} = \frac{1}{2} (d_k - o_k)(1 - o_k^2), \quad \text{pour } k = 1, 2, \dots, K$$

$$\delta_{yj} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^K \delta_{ok} w_{kj}, \quad \text{pour } j = 1, 2, \dots, J$$

- lorsque la fonction d'activation continue unipolaire est utilisée:

$$\delta_{ok} = (d_k - o_k)(1 - o_k)o_k, \quad \text{pour } k = 1, 2, \dots, K$$

$$\delta_{yj} = y_j(1 - y_j) \sum_{k=1}^K \delta_{ok} w_{kj}, \quad \text{pour } j = 1, 2, \dots, J$$

# 4) Rétro-propagation des erreurs

## Algorithme d'entraînement par la rétro-propagation des erreurs

### ► Cycle d'apprentissage (suite)

- **Étape 5:** Les poids synaptiques de la couche de sortie sont ajustés

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{ok} y_j, \text{ pour } k = 1, 2, \dots, K \text{ et } j = 1, 2, \dots, J$$

- **Étape 6:** Les poids synaptiques de la couche cachée sont ajustés

$$\nu_{ji} \leftarrow \nu_{ji} + \eta \delta_{yj} z_j, \text{ pour } j = 1, 2, \dots, J \text{ et } i = 1, 2, \dots, I$$

- **Étape 7:** Si  $p < P$ , alors  $q \leftarrow q + 1, p \leftarrow p + 1$  et aller à l'étape 2, sinon autrement aller à l'étape 8.

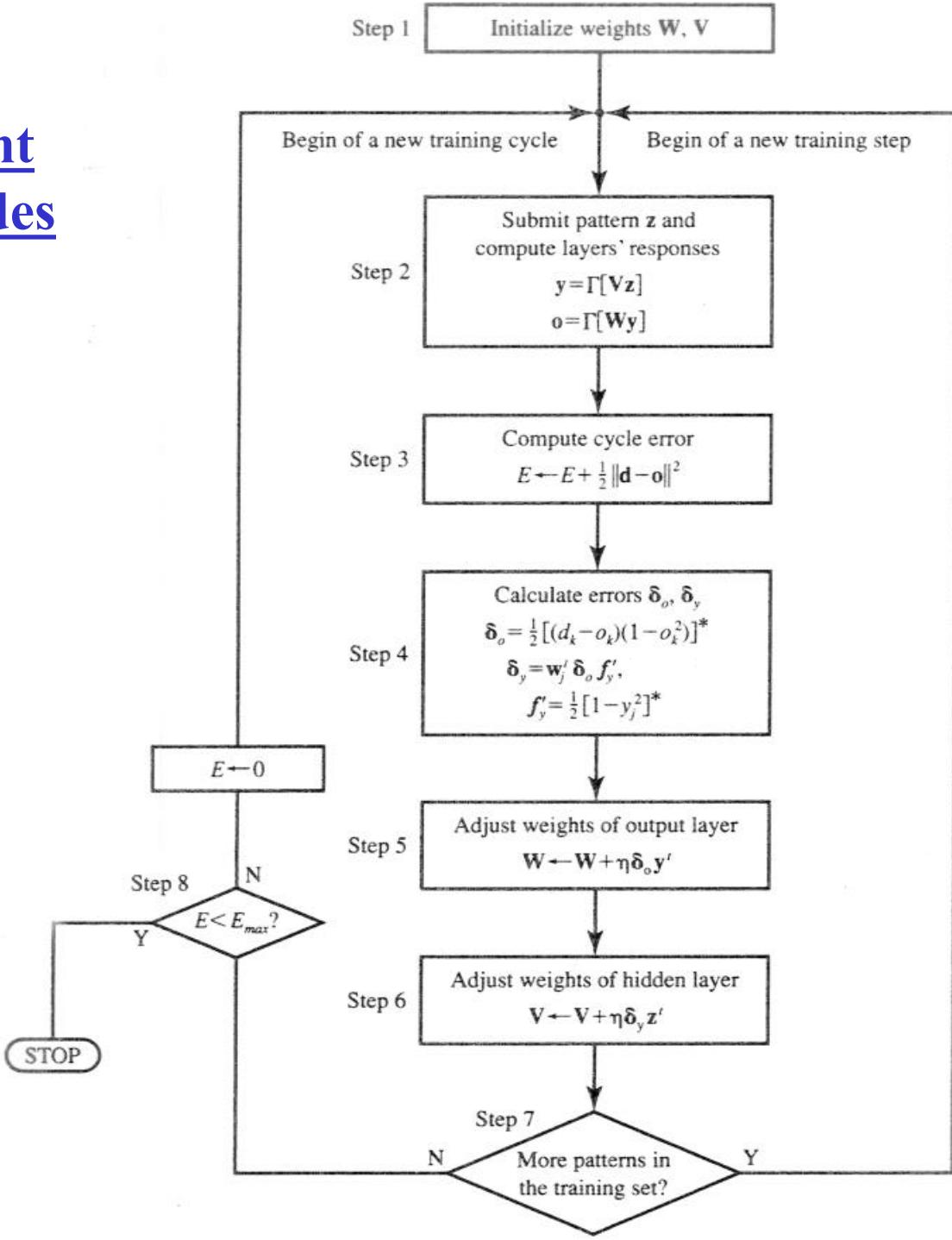
- **Étape 8:** Le cycle d'apprentissage est complété

**Si  $E < E_{max}$ :** l'entraînement est terminé, et  $W, V, q$  et  $E$  sont conservés

**Si  $E > E_{max}$ :** alors  $E \leftarrow 0, q \leftarrow q + 1, p \leftarrow 1$  et commencer un nouveau cycle d'apprentissage à l'étape 2.

**FIN. (EBPTA)**

# Algorithme d'entraînement par la rétro-propagation des erreurs



\*If  $f(\text{net})$  given by (2.4a) is used in Step 2, then in Step 4 use  
 $\delta_o = [(d_k - o_k)(1 - o_k)o_k]$ ,  $f_y' = [(1 - y_j)y_j]$

## 4) Rétro-propagation des erreurs

### Évaluation de l'erreur d'entraînement

- ▶ Ces algorithmes d'apprentissage reposent sur la minimisation d'une *mesure de l'erreur* effectuée par le réseau de neurones
  - À une itération donnée de la **phase d'entraînement**, l'erreur  $E$  à minimiser est habituellement celle produite lors de l'application d'un patron d'entraînement à l'entrée du RNA
  - Pour ce qui est de l'évaluation de la **qualité de l'entraînement du réseau**, l'erreur cumulée par epoch, pour tous les patrons de l'ensemble d'entraînement, est évaluée  
(Plusieurs définitions permettent d'exprimer des points de vues différents.)

## 4) Rétro-propagation des erreurs

### Évaluation de l'erreur d'entraînement

- **L'erreur cumulée** est évaluée pour tous les cycles de la phase d'entraînement (Étape 3, EBPTA) et est définie à partir de l'erreur quadratique:

$$E = \frac{1}{2} \sum_{p=1}^P \sum_{k=1}^K (d_{pk} - o_{pk})^2$$

- permet de mesurer la précision obtenue après une epoch ( $P$  itérations d'entraînement)
- cette mesure est cependant proportionnelle au nombre de neurones de sorties  $K$  et au nombre de vecteurs d'entraînement  $P$
- elle ne permet donc pas la comparaison directe de différents types de réseaux, ou la comparaison de la qualité d'entraînement d'un réseau pour différentes

## 4) Rétro-propagation des erreurs

### Évaluation de l'erreur d'entraînement

- L'erreur RMS ( $E_{rms}$ ) est une mesure de l'erreur qui est normalisée:

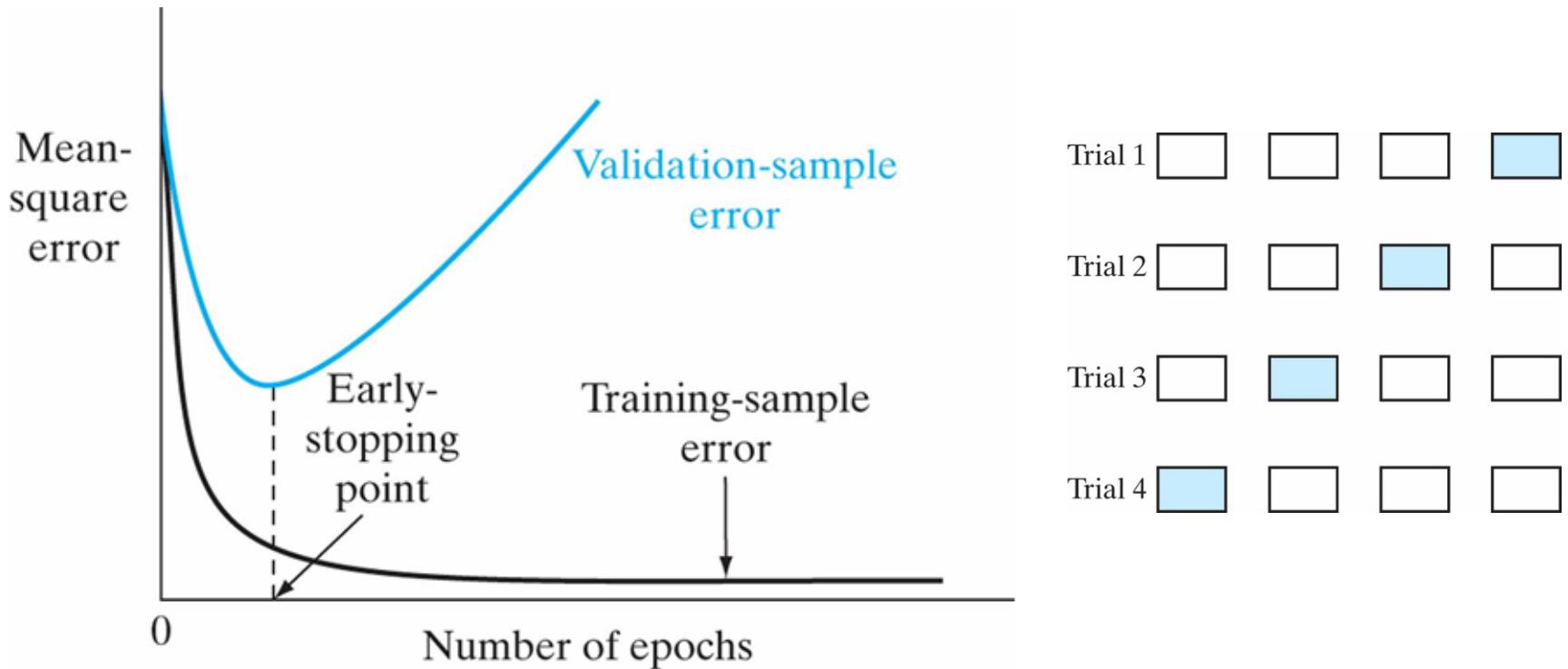
$$E_{rms} = \sqrt{\frac{1}{PK} \sum_{p=1}^P \sum_{k=1}^K (d_{pk} - o_{pk})^2}$$

- cette mesure a l'avantage d'être invariante par rapport au nombre de neurones de sorties et au nombre de patrons d'entraînement.
- $E$  et  $E_{rms}$  précédentes permettent d'évaluer la précision de la fonction d'implication estimée par le RNA:
  - en mode apprentissage, pour guider l'adaptation des poids
  - mesure seulement la capacité d'assimiler la base d'entraînement

# 4) Rétro-propagation des erreurs

## Évaluation de l'erreur d'entraînement

- ▶ Illustration du processus de validation croisée pour arrêter les epochs d'entraînement:



## 4) Rétro-propagation des erreurs

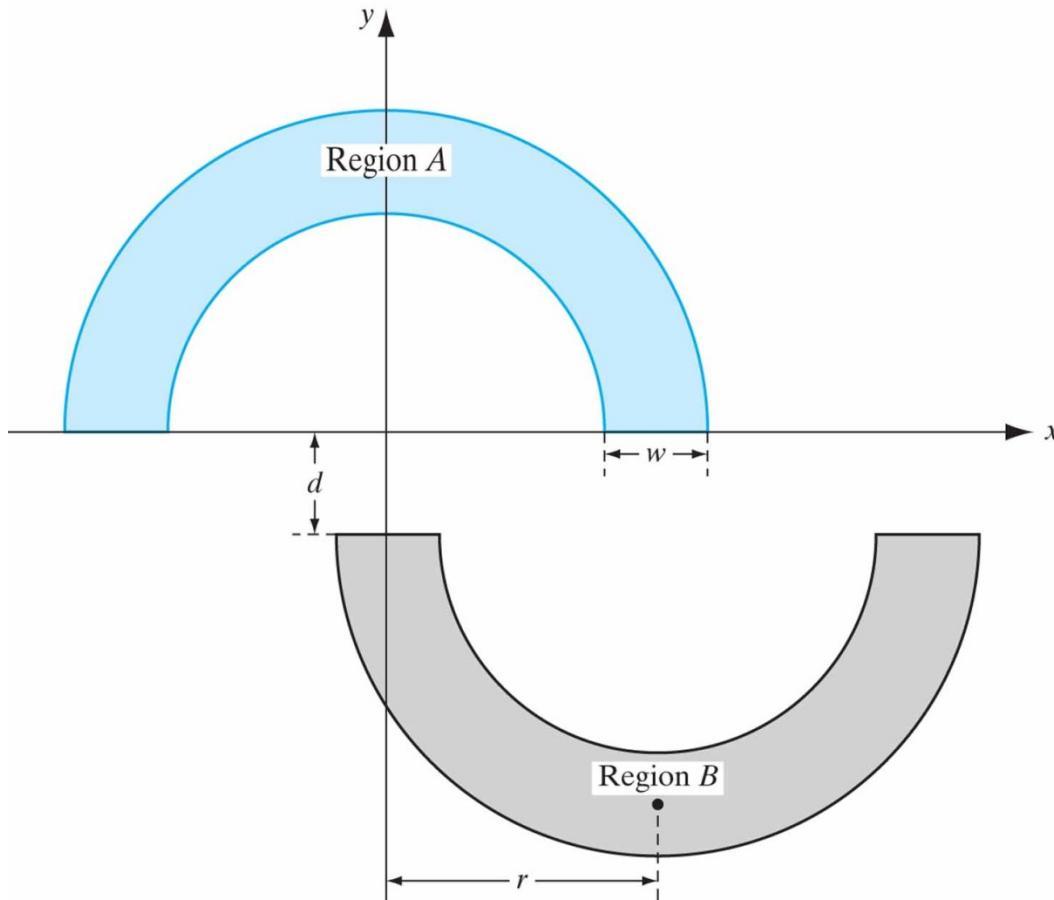
### Évaluation de l'erreur en test

► **Taux d'erreurs sur un ensemble de patrons de test  $D$ :**

- en classification, les neurones de sorties sont seuillés: e.g., des valeurs plus petites que 0.1 ou plus grandes que 0.9 sont considérées comme 0 et 1 respectivement
- L'erreur de **classification**:  $E_c = | \text{patrons erronés} | / | D |$
- L'erreur de **décision**:  $E_d = N_{\text{err}} / (K | D |)$   
où  $N_{\text{err}}$  représente le nombre de bits en erreur, évalué sur  $D$
- Les RNA utilisés comme classificateur peuvent performer parfaitement en reconnaissance, avec  $E_c = 0\%$  et  $E_d = 0\%$ , montrer une erreur cumulée (ou rms) non négligeable

## 4) Rétro-propagation des erreurs

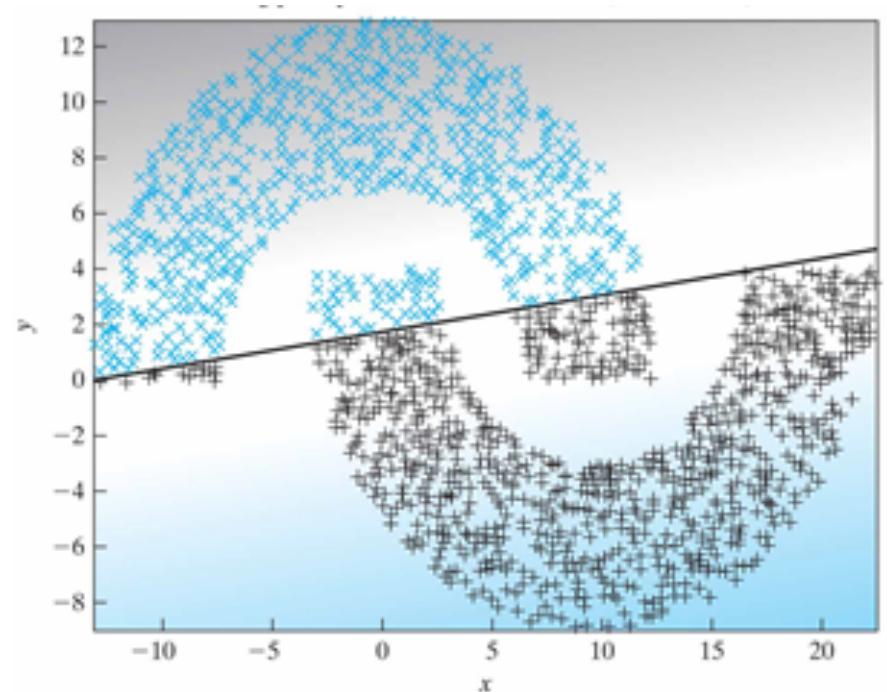
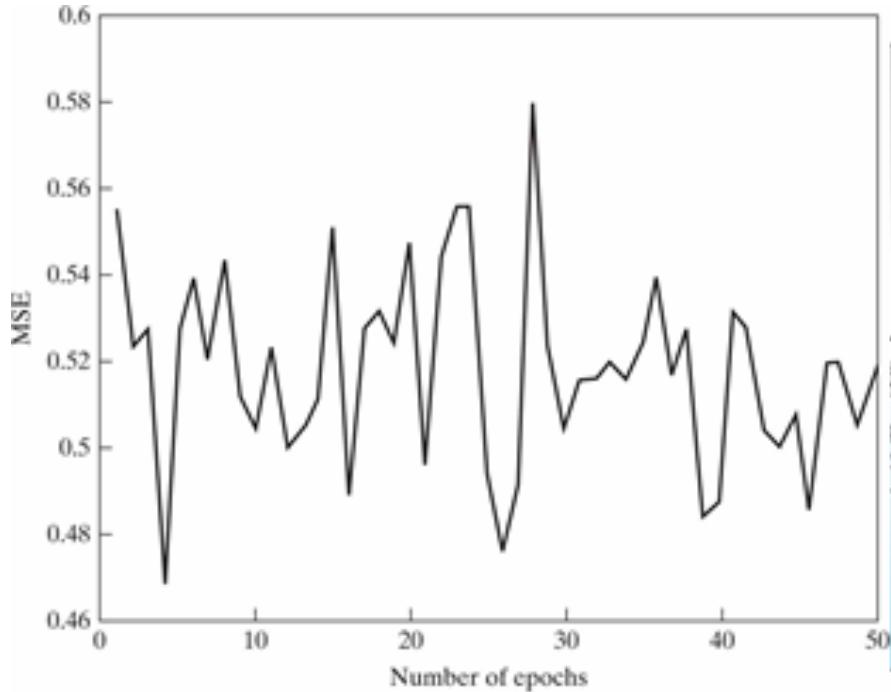
### Problème de classification



## 4) Rétro-propagation des erreurs

### Problème de classification

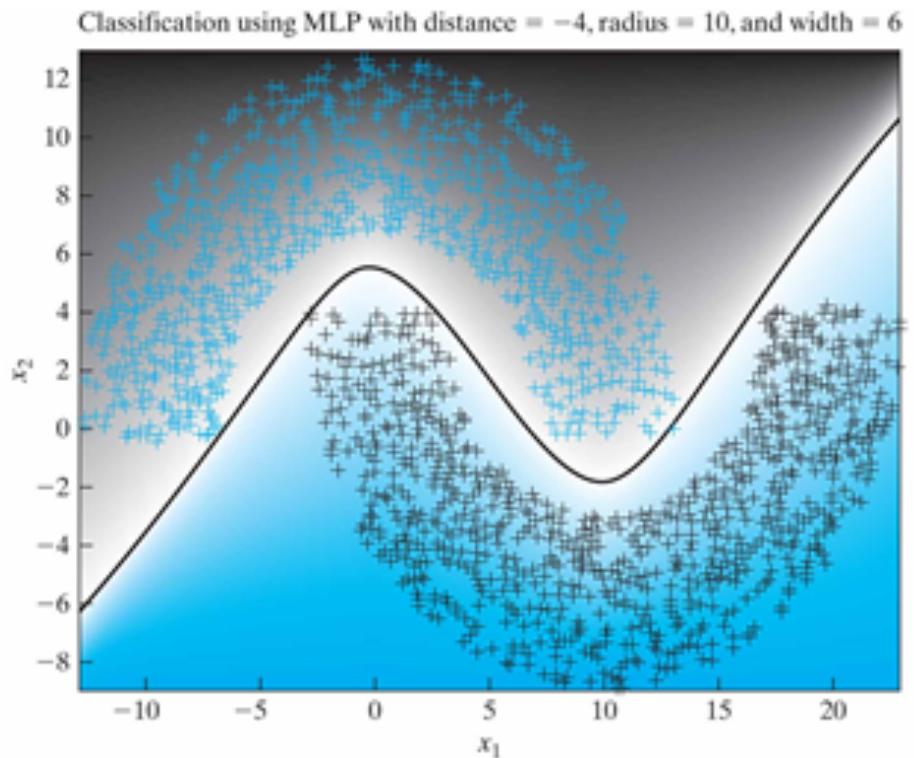
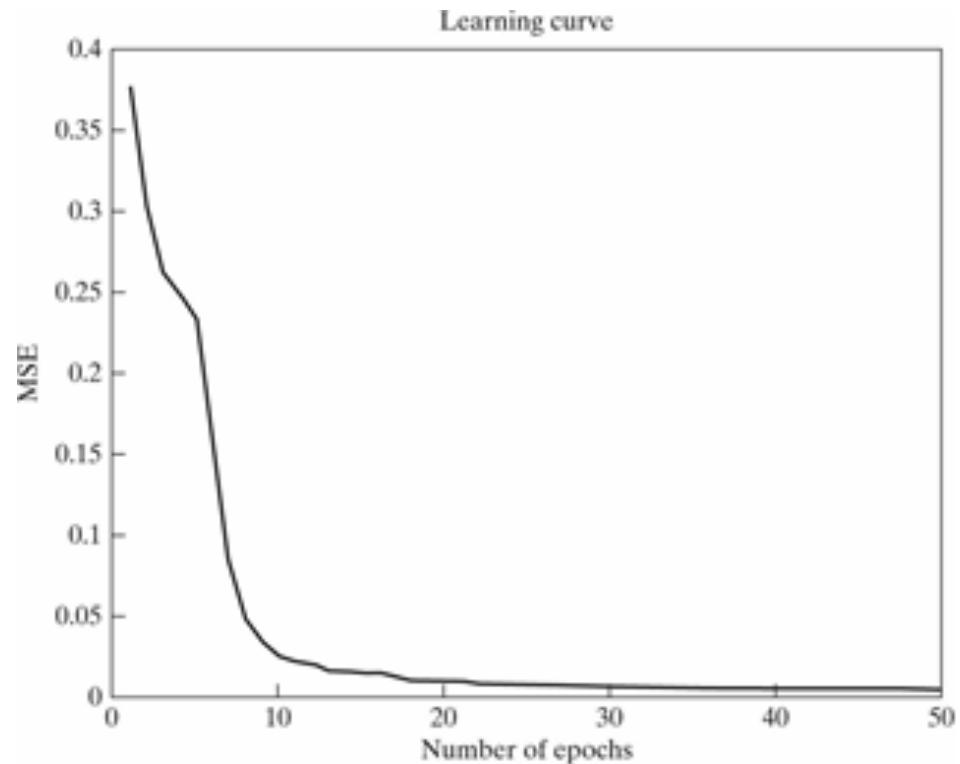
- Résultat du perceptron avec  $d = -4$ ,  $r = 10$  et  $w = 6$



## 4) Rétro-propagation des erreurs

### Problème de classification

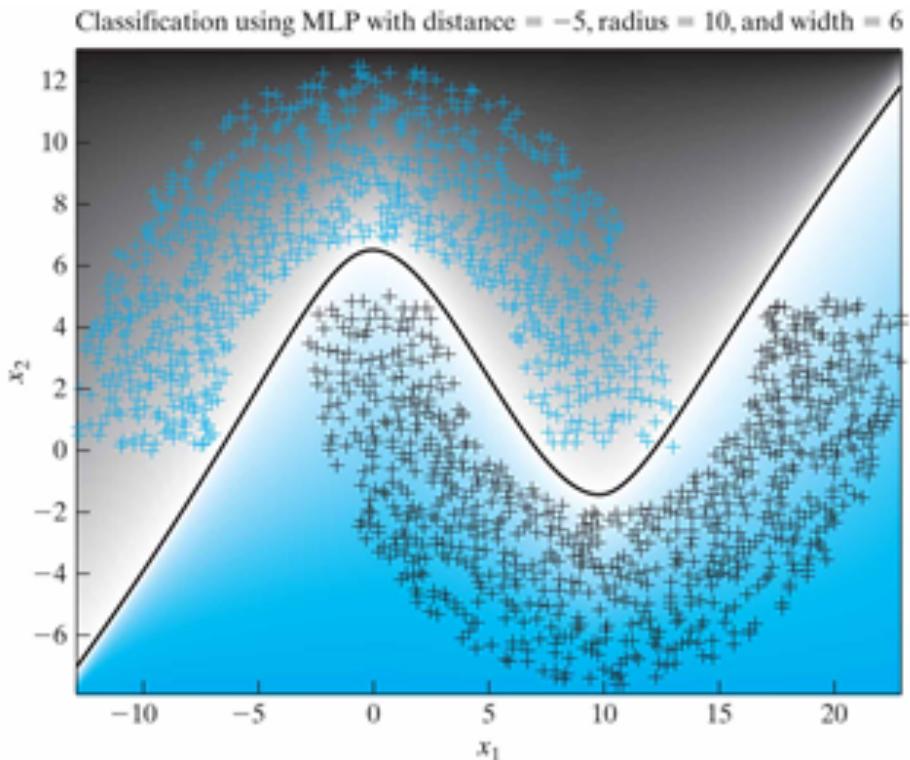
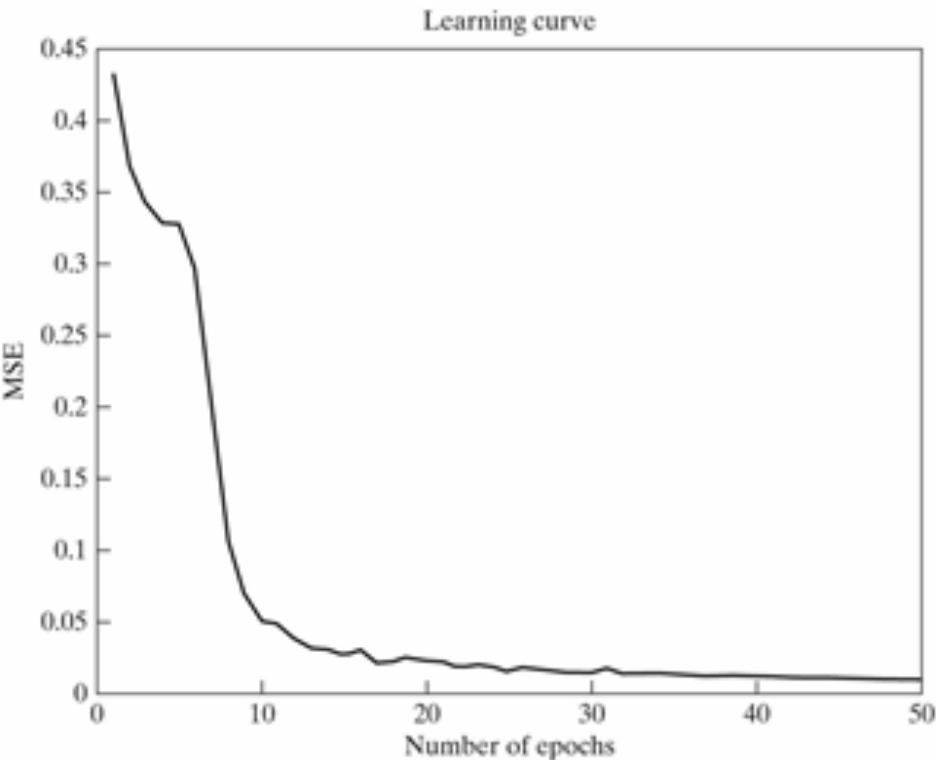
- Résultat du MLP-BP avec  $d = -4$ ,  $r = 10$  et  $w = 6$



## 4) Rétro-propagation des erreurs

### Problème de classification

- Résultat du MLP-BP avec  $d = -5$ ,  $r = 10$  et  $w = 6$



# CONTENU DU COURS

## A.3 Réseaux de neurones multicouches sans rétroaction (MLP)

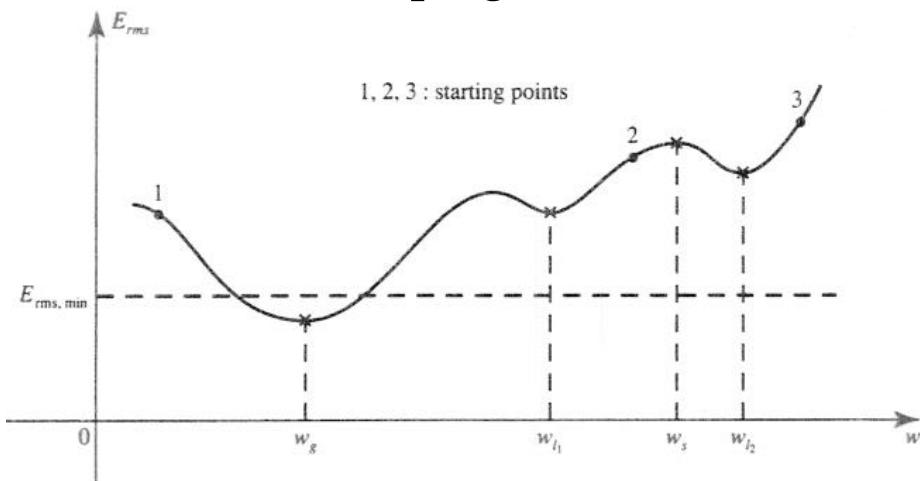
- 1) Classification de données linéairement non-séparables
- 2) Règle d'apprentissage delta pour une couche de perceptrons
- 3) Règle d'apprentissage delta généralisée
- 4) Entrainement par rétro-propagation des erreurs
- 5) Facteurs d'apprentissage

## 5) Facteurs d'apprentissage

### Convergence de RNA

► En pratique, la mise-en-œuvre d'un MLP-BP n'est pas sans rencontrer des problèmes de convergence

- commun aux techniques d'optimisation dans un espace de grande dimension
- un problème particulier lié aux procédures de minimisation de fonction d'erreur est le bloquage dans un *minimum local*



## 5) Facteurs d'apprentissage

### Convergence de RNA

- ▶ **Solution simple si les minimums locaux ne sont pas très prononcés**
  - randomiser l'ordre de présentation des patrons lors de l'entraînement
  - superposition d'un bruit de faible amplitude et de moyenne nulle aux données de l'ensemble d'entraînement
    - (si les données d'entraînement ne sont pas le résultat d'un processus aléatoire, mais la réalisation d'un processus purement déterministe)
- ▶ **Plusieurs facteurs peuvent influencer la convergence de la procédure d'entraînement**

## 5) Facteurs d'apprentissage

### a) Initialisation des poids

- ▶ **L'initialisation des poids a un effet très important sur la qualité de l'entraînement du réseau**
  - Il s'agit de l'état initial pour l'optimisation des paramètres
  - Par exemple, des poids égaux peuvent empêcher le réseau de converger vers une solution acceptable
  - Les poids sont en général initialisés aléatoirement à des petites valeurs selon une distribution uniforme sur la plage [-0.1, 0.1].
  - **Une solution commune:** effectuer plusieurs réplications d'entraînement, et chaque fois réinitialiser les poids

## 5) Facteurs d'apprentissage

### b) Ajustement cumulatif vs progressif des poids

- ▶ **Apprentissage séquentiel après chaque cycle:** L'algorithme d'apprentissage par la rétro-propagation des erreurs est basé sur la réduction d'erreur en ajustant les poids **après chaque itération** (à chaque présentation d'un patron d'entraînement)
  - **Avantages:** Permet d'implanter une vraie descente de gradient. De plus, l'ajustement des poids n'a pas à être conservé intégralement en mémoire.
  - **Désavantages:** Le RNA entraîné de cette façon peut être grandement influencé par l'ordre de présentation, et par quelques patrons de l'ensemble d'entraînement.

## 5) Facteurs d'apprentissage

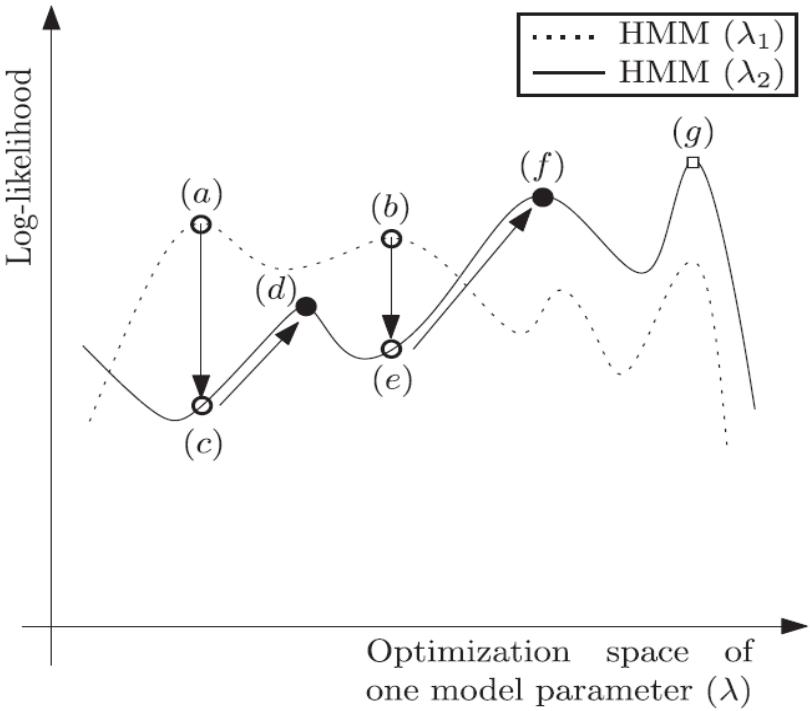
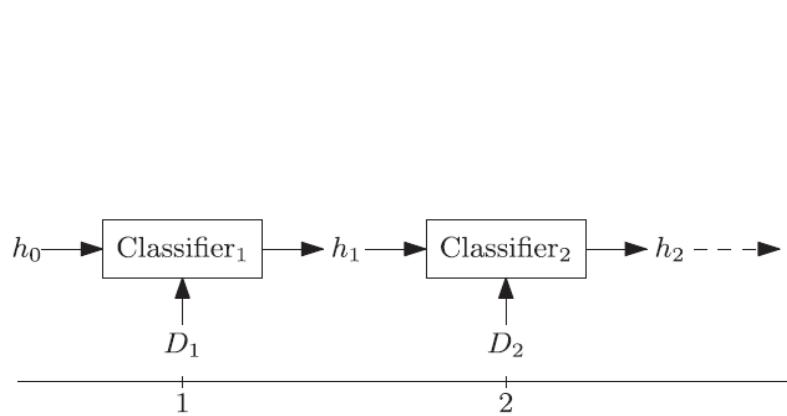
### b) Ajustement cumulatif vs progressif des poids

- ▶ **Apprentissage par lot:** Il est généralement préférable d'ajuster les poids après chaque epoch (présentation de tous les patrons d'entraînement), donc:
  - utiliser de la valeur cumulée des changements de poids par epoch (sur  $P$  cycles consécutifs):  $\Delta w = \sum_{p=1}^P \Delta w_p$
  - randomiser l'ordre de présentation des patrons d'une epoch
  - utiliser une petite valeur de  $\eta$  (constante d'apprentissage)
  - **Désavantage:** L'apprentissage est plus long (plus d'itérations au total), et plus de consommation mémoire

## 5) Facteurs d'apprentissage

### b) Ajustement cumulatif vs progressif des poids

- **Défis de l'apprentissage incrémental:** La corruption de connaissances est un problème commun lors de l'apprentissage en-ligne ou incrémental de nouvelles données d'entraînement



## 5) Facteurs d'apprentissage

### c) Gain de la fonction d'activation

► **Le choix et la forme de la fonction d'activation influencent grandement la vitesse d'apprentissage des réseaux**

- La dérivée de la fonction d'activation  $f'(net)$  est utilisée comme un facteur multiplicatif dans l'évaluation des signaux d'erreur  $\delta_o$  et  $\delta_y$
- Pour une constante  $\eta$  fixe, l'ajustement des poids est effectué en proportion de la valeur de  $\lambda$
- L'utilisation d'une  $f(net)$  avec un gain  $\lambda$  élevé est équivalent à choisir une constante d'apprentissage  $\eta$  élevée.

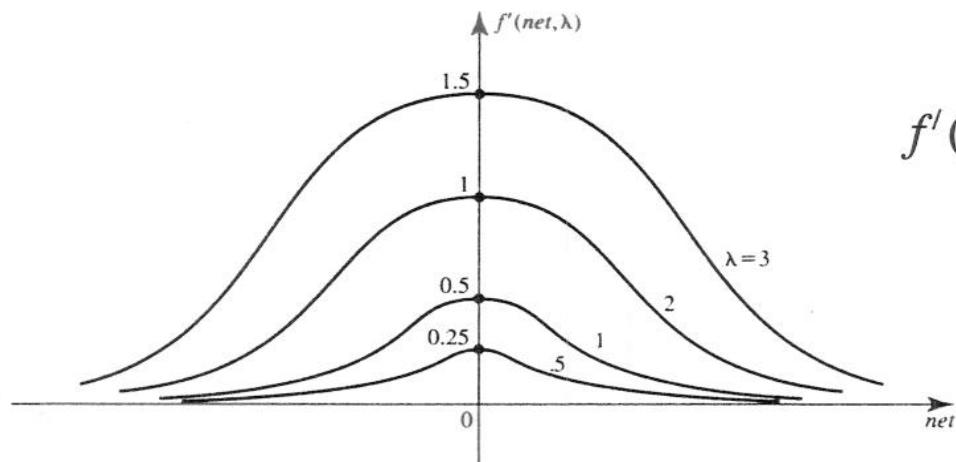
**En pratique, il est commun d'ajuster le gain à  $\lambda=1$ , et de contrôler la vitesse d'entraînement simplement en ajustant la valeur de la constante d'apprentissage  $\eta$ .**

## 5) Facteurs d'apprentissage

### c) Gain de la fonction d'activation

► Exemple – cas bipolaire:

- Cette fonction atteint une valeur maximum de  $\frac{1}{2}\lambda$  lorsque  $net = 0$
- Puisque les poids sont ajustés en proportion de la valeur de  $f'(net)$ , alors les poids associés aux neurones qui répondent près de la zone médiane ( $net = 0$ ) de la plage dynamique de  $f(net)$  seront modifiés de façon très importante



$$f'(net) = \frac{2\lambda \exp(-\lambda net)}{[1 + \exp(-\lambda net)]^2}$$

## 5) Facteurs d'apprentissage

### d) Constante d'apprentissage $\eta$

- **La précision et la vitesse de convergence de l'algorithme d'apprentissage dépendent en grande partie de la constante d'apprentissage  $\eta \in [0; 1]$**
- Comme avec plusieurs algorithmes d'optimisation basés sur la descente de gradient, la valeur optimale de  $\eta$  dépend du problème particulier à résoudre, et il n'y a pas de valeurs privilégiées pour différents cas
  - La forme de la surface d'erreur commande une valeur de  $\eta$  adaptée
    - $\eta$  grande si la pente près du minimum est très petite
    - $\eta$  petite si la pente près du minimum est très étroite

**La constante d'apprentissage  $\eta$  doit être choisie expérimentalement pour chaque différent problème**

## 5) Facteurs d'apprentissage

### e) Momentum

- **Le momentum est une solution typique d'accélérer la vitesse de convergence de l'algorithme d'entraînement**

- La méthode consiste à ajouter à la valeur courante d'ajustement des poids  $\Delta w(t)$  une fraction de l'ajustement précédente  $\Delta w(t-1)$

$$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1)$$

où deuxième terme représente le momentum et  $\alpha \in [0; 1]$

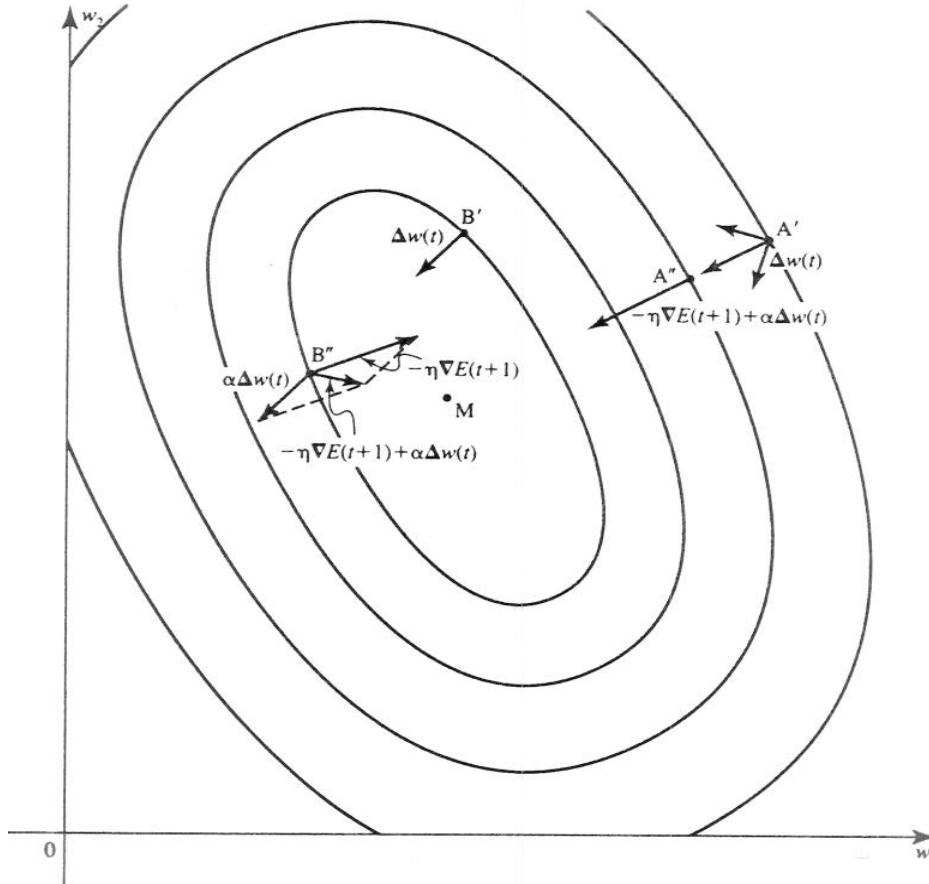
- Lorsque l'ajustement des poids est effectué à tous les  $N$  d'itérations, alors le changement de poids peut être formulé par

$$\Delta w(t) = -\eta \sum_{n=0}^N (\alpha^n \nabla E(t-n))$$

## 5) Facteurs d'apprentissage

### e) Momentum

► Illustration de l'effet de la méthode du momentum:



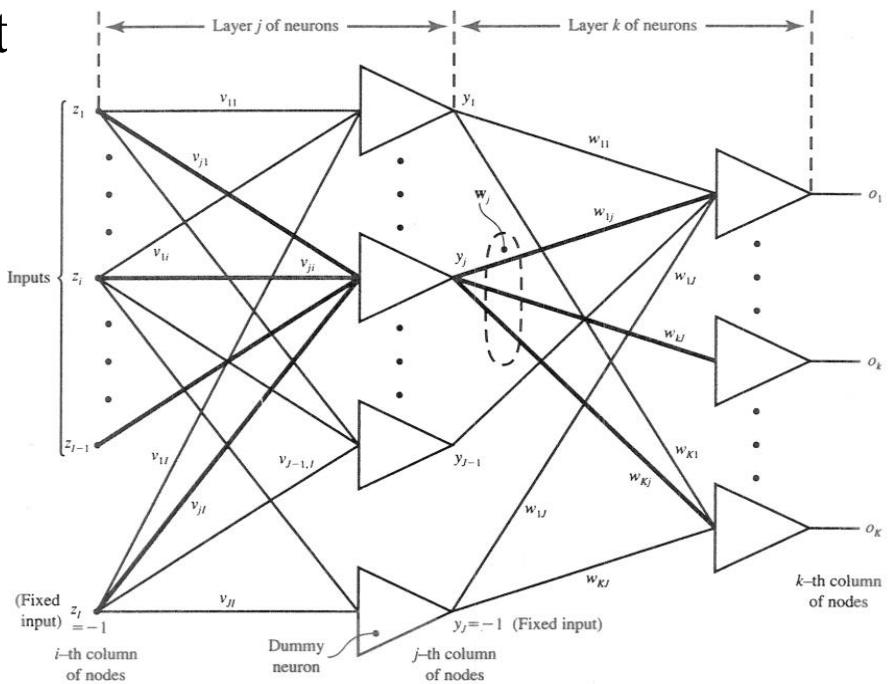
## 5) Facteurs d'apprentissage

### f) Architecture de RNA

- Un des paramètres qui a le plus d'influence sur la performance des RNA est le choix de l'architecture

- nombre de neurones  $I$ ,  $J$  et  $K$  de la couche d'entrée, cachée et de sortie, respectivement

Architecture typique d'un RNA avec une couche cachée



## 5) Facteurs d'apprentissage

### f) Architecture de RNA

- ▶ **Le nombre de neurones  $I$  de la couche d'entrée est déterminé par la dimension des patrons d'entrées**
  - **techniques d'extraction et de sélection:** permettent de représenter les objets en entrées avec des patrons compacts et discriminants
  - on veut typiquement minimiser  $I$  sans affecter de façon significative la discrimination
  - par contre, si les objets sont difficiles à classer dans un espace de faible dimension, l'impact sur l'architecture du RNA sera important, et le nombre de patrons d'entraînement requis pour l'apprentissage doit également être augmenté

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

- ▶ **Représentation *locale des données* de sorties:**
  - Pour un RNA de type classificateur, le nombre  $K$  de neurones de sorties correspond au nombre de classes
  - Le RNA agit comme un décodeur, où la classe associée au patron d'entrée correspond au neurone de sortie activé
- ▶ **Habituellement ce type de représentation permet la mise en œuvre d'un classificateur plus robuste. Par contre:**
  - temps d'apprentissage plus long, et
  - un nombre de patrons requis pour l'entraînement plus grand étant donné le nombre plus élevé de poids à ajuster

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Représentation distribuée des données de sorties:

- Le nombre  $K$  de neurones de sorties peut être réduit si le décodage des classes n'est pas requis
- Par exemple,  $K = 5$  neurones seulement sont requis pour implanter un classificateur capable de reconnaître les 26 lettres de l'alphabet
- Le **nombre de neurones de sorties** requis pour un classificateur varie entre  $\log_2 K$  (représentation distribuée) et  $K$  (représentation locale)

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Nombre de neurones $J$ sur la couche cachée:

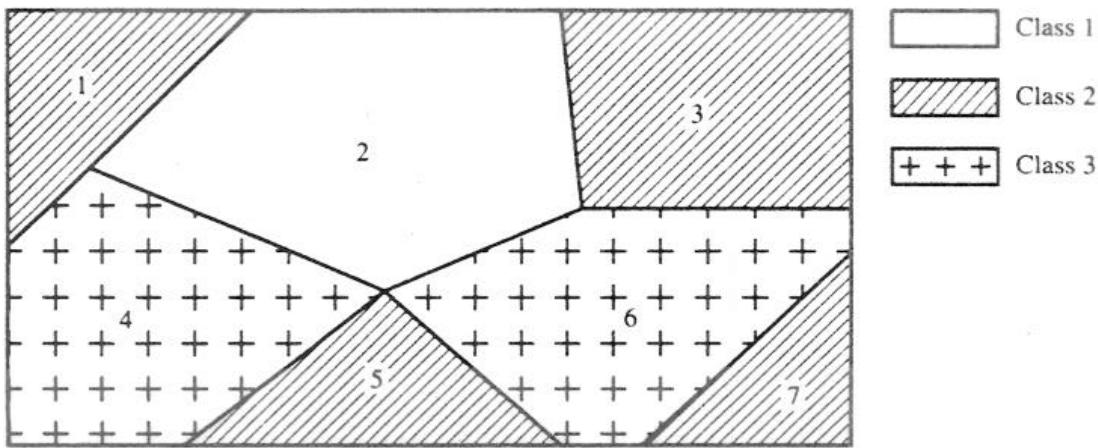
- Le choix de  $J$  est parmi les facteurs les plus importants à considérer lors de la réalisation d'un RNA
- Lié à la capacité de représentation du RNA pour un problème
- Il n'y a aucunes méthodes exactes dans la littérature pour déterminer  $J$  pour une application particulière. Cependant, certaines **règles empiriques** peuvent être appliquées.
- Le nombre  $J$  dépend de  $n$  (la dimension du patron d'entrée) et de  $M$  (le nombre de régions de décision requis dans l'espace des données d'entrées)

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Estimation du nombre de neurones $J$ cachées:

- Supposons que l'espace d'entrée est constitué de  $M$  régions disjointes et linéairement séparables. Chaque région peut être étiquetée selon une des  $R$  classes, avec  $R \leq M$
- **Exemple:**  $n = 2$ ,  $M = 7$  régions et  $R = 3$  classes.



## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Estimation du nombre de neurones $J$ cachées:

- Intuitivement, le nombre  $M$  de régions linéairement séparables dicte le nombre minimum d'exemples requis pour l'entraînement du réseau, soit  $P = M$ .
- Un nombre élevé d'exemples d'entraînement ( $P \gg M$ ) permet au réseau une définition plus fine des frontières entre les régions.
- Il existe une relation entre  $M$ ,  $J$  et  $n$  qui permet d'évaluer un paramètre si les deux autres sont connus.

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Estimation du nombre de neurones $J$ cachées:

- Cas général ( $J \geq n$ ): Le *nombre maximum de régions linéairement séparables* avec  $J$  neurones cachés et un espace d'entrée de dimension  $n$  est donné par:

$$M(J,n) = \sum_{k=0}^n \binom{J}{k}, \quad \text{où} \quad \binom{J}{k} = 0 \quad \text{pour } J < k$$

- Alors, connaissant  $n$  et  $M$ , la valeur de  $J$  peut être estimée. Par exemple le nombre de neurones cachés requis pour solutionner le problème de classification de la figure précédente avec  $n = 2$  et  $M = 7$  est de  $J = 3$ .

## 5) Facteurs d'apprentissage

### f) Architecture de RNA

#### ► Exemple – Conception du réseau XOR: ( $n = 2$ )

- Posons  $J \geq n$ , alors nous obtenons  $J = 2$  pour  $M = 4$  régions
- Architecture de base pour réaliser de la fonction XOR avec  $J = 2$  neurones cachés

