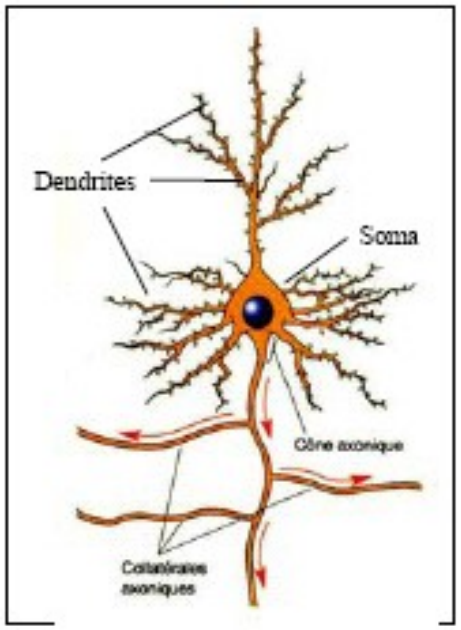
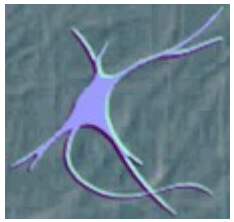
# Intelligence Artificielle
# Apprentissage
# Partie 2

# Réseaux de Neurones

## Boites à outil sur le web

**1. Stuttgart Neural Network Simulator (SNNS)**
http://www-ra.informatik.uni-tuebingen.de/SNNS/
**2. Netlab neural network software (NetLab)**
http://www.ncrg.aston.ac.uk/netlab/
**3. Artificial Neural Network (ANN)**
http://www.scilab.org/contrib/index_contrib.php?page=download.php&category=MODELING%20AND%20CONTROL%20TOOLS



Diamètre de la cellule: 0.01 - 0.05 mm,

**Soma (corps cellulaire)**

formes variables (gén. sphériques),
20 µm de diamètre,
contient le noyau,
entourée d 'une membrane de 5 nm d'ép.

**Axone**

unique aux cellules nerveuses,
diamètre: 1-25 µm (humain),
1 mm (poulpe),
longueur: 1 mm to 1 m. (!!!),
connexion vers les autres neurones (synapses),
permet la transmission d'information,

**Dendrites**

reçoivent les signaux des autres neurones,
chacune couverte de centaines de synapses.

# Quelques chiffres ...

- nombre de neurones dans le cerveau: ~ $10^{11}$
- nombre de connexions par neurone: ~ $10^{4}$ - $10^{5}$
- temps de cycle (*switching time*): ~ $10^{-3}$ seconde
- temps moyen d'une activité cognitive: ~ 0.1 seconde
(ex. reconnaissance de visages)

*Il n'y a donc de la place que pour 100 cycles de traitement, ce qui est insuffisant pour une activité complexe !!!*

*Le cerveau doit donc effectuer des opérations en parallèle !!!*

# Modélisation

• **Intuition**

– les capacités (intelligence?) résident dans les connexions entre les milliards de neurones du cerveau humain.

• Propriétés d'une modélisation

– neuro-biologiquement plausible (unité ≈ neurone),
– traitement distribué et parallélisme massif
  • • beaucoup d'interconnexions entre unités,
  • • ajustement automatique des poids des connexions,
– traitement non-symbolique (i.e., pas de règles explicites d'inférence),
– intelligence comme propriété émergente,
– connaissance représentée par les poids des connexions entre unités,
– pas de "maître de cérémonie" (*executive supervisor*).

Le terme **"réseaux de neurones"** peut prendre des significations différentes:

– réseaux de neurones artificiels,
– modèles connexionnistes,
– neurosciences calculatoires,
– modélisation neuronale.

# Quelques repères ...

**1943: *J. McCulloch & W. Pitts***
– proposent un modèle simple de neurones capable de produire une machine de Turing,
– démontrent qu'un assemblage synchrone de tels neurones est une machine universelle de calcul (càd. n'importe quelle fonction logique peut être représentée par des unités à seuil),

**1948: *D. Hebb***
– propose une règle d'apprentissage pour des réseaux de neurones

**1958: *F. Rosenblatt***
– propose le modèle du *Perceptron* et démontre son théorème de convergence,

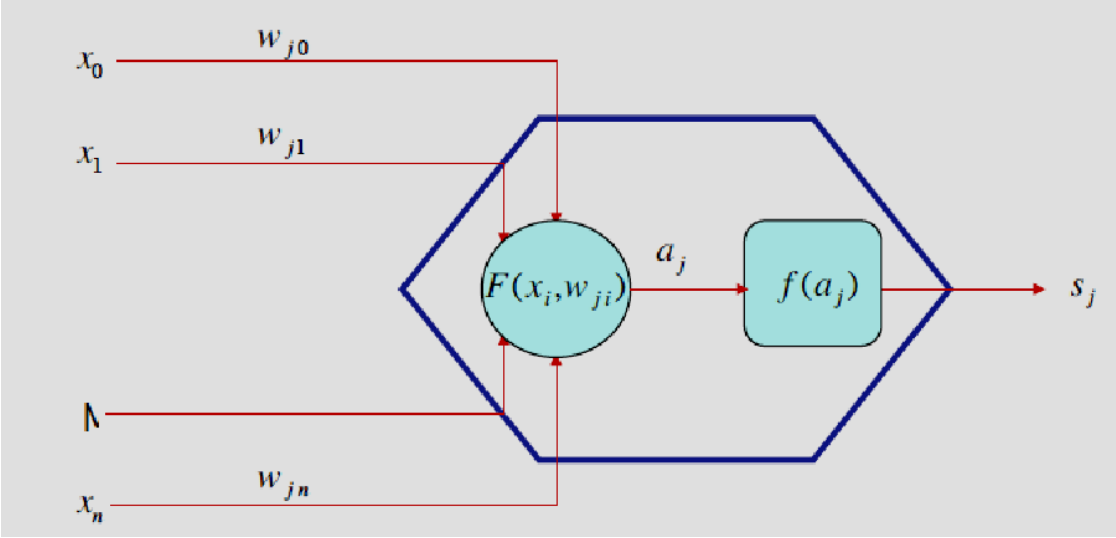
**1969: M. Minsky & S. Papert**
– démontrent les limitations du modèle du *Perceptron*,

**1985:** apprentissage par rétro-propagation pour les réseaux multi-couches.

# Le Neurone Formel

## Définition :

Un neurone formel (artificiel) est une unité de traitement qui reçoit des données en entrée, sous la forme d'un vecteur, et produit une sortie réelle.
Cette sortie est une fonction des entrées et des poids des connexions.

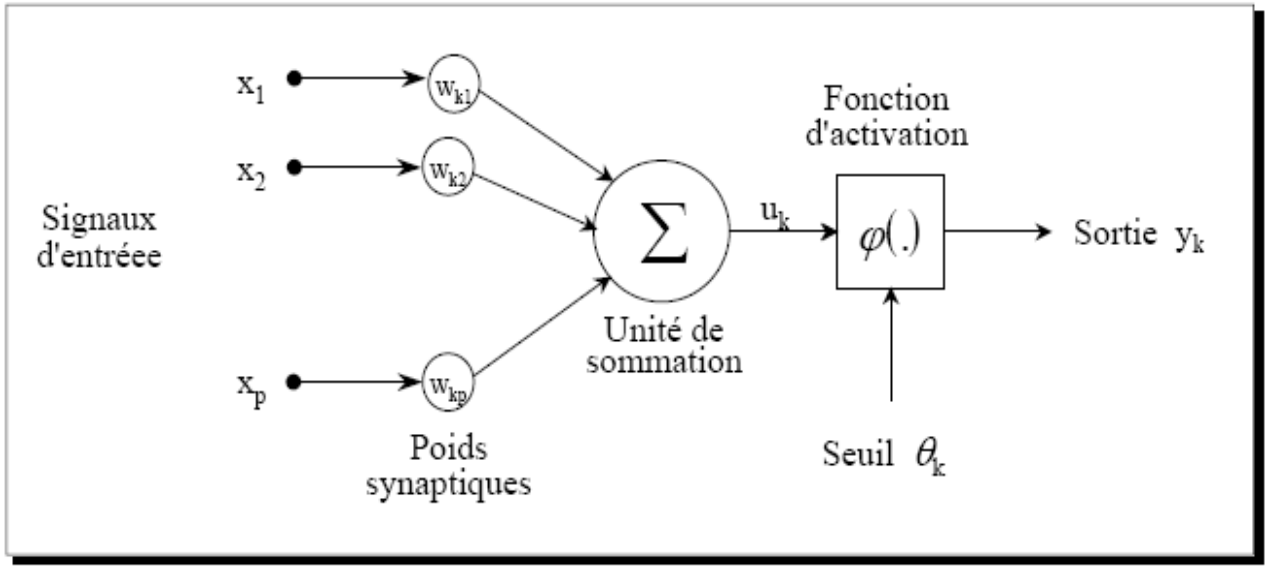



## Le Poids de Connexion

**Définition :** Une connexion entre deux unités i et j indique la possibilité d'une relation physique entre ces deux unités.

La valeur numérique du poids associé à une connexion entre deux unités reflète la force de la relation entre ces deux unités. **Si cette valeur est positive**, la connexion est dite **excitatrice**, sinon elle est dite **inhibitrice**. La convention usuelle est de noter le poids de la connexion reliant le neurone i au neurone j : $\mathbf{w_{ji}}$

# Modèle de McCulloch & Pitts




Ce modèle est mathématiquement décrit par 2 équations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \qquad \text{et} \qquad y_k = \varphi(u_k - \theta_k)$$

où:

$x_1, x_2, \ldots, x_p$ sont les *entrées*,

$w_{k1}, w_{k2}, \ldots, w_{kp}$ sont les *poids synaptiques* du neurone k,

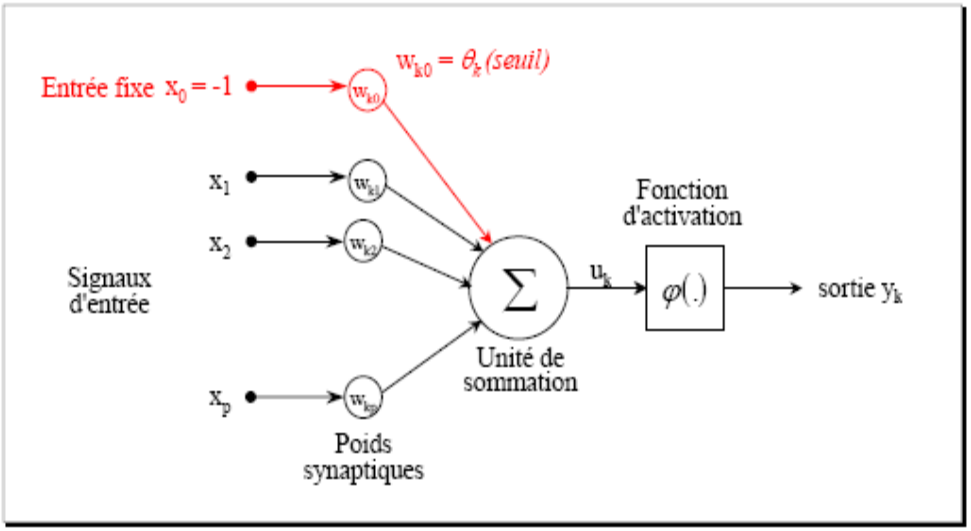
$u_k$ est la *sortie de l'unité de sommation*,

$\Theta_k$ est le *seuil*,

$\phi(.)$ est la *fonction d'activation*,

$y_k$ est le signal de *sortie* du neurone k.

# Modèle étendu

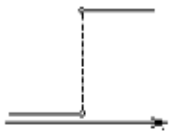



La fonction d'activation définit la valeur de sortie d'un neurone en termes des niveaux d'activité de ses entrées.

3 types de fonctions d'activation :

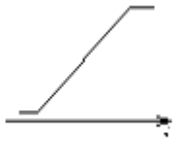
- *fonction à seuil*

$$y_k = \varphi(v_k) = \begin{cases} 1 & \text{si } v_k \geq 0 \\ 0 & \text{si } v_k < 0 \end{cases}$$



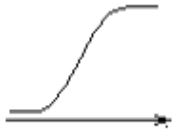

- *fonction linéaire par parties*

$$\varphi(v) = \begin{cases} 1 & \text{si } v \geq \alpha \\ v & \text{si } \alpha > v > \beta \\ 0 & \text{si } v \leq \beta \end{cases}$$



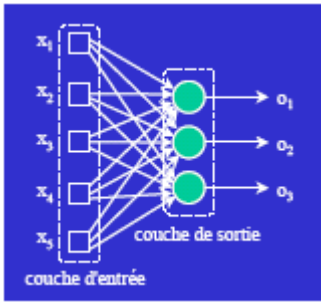

- *fonction sigmoïde*

$$\varphi(v) = \frac{1}{1+e^{-av}}$$

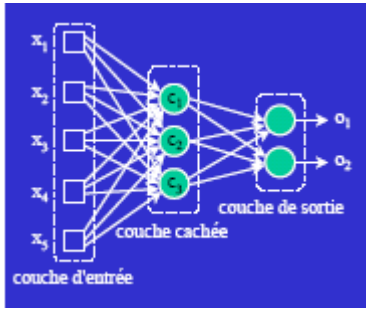


# Topologies de réseaux

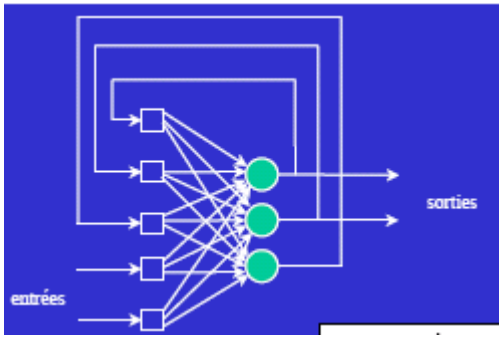
• Réseau acyclique
(*"feedforward"*) à une seule couche



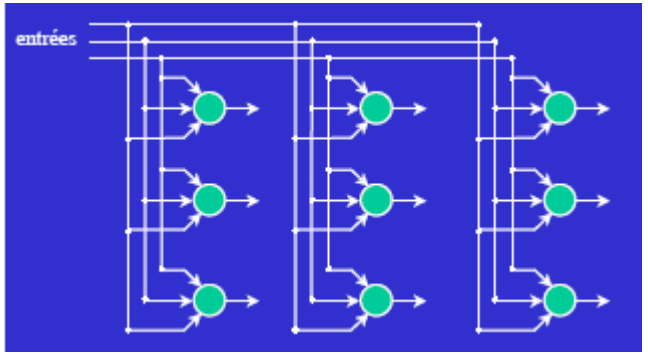

• Réseau acyclique multi-couches



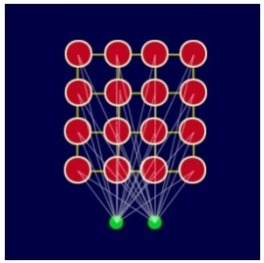

• Réseau récursif (réseau de Hopfield)




• Réseau *"en treillis"*



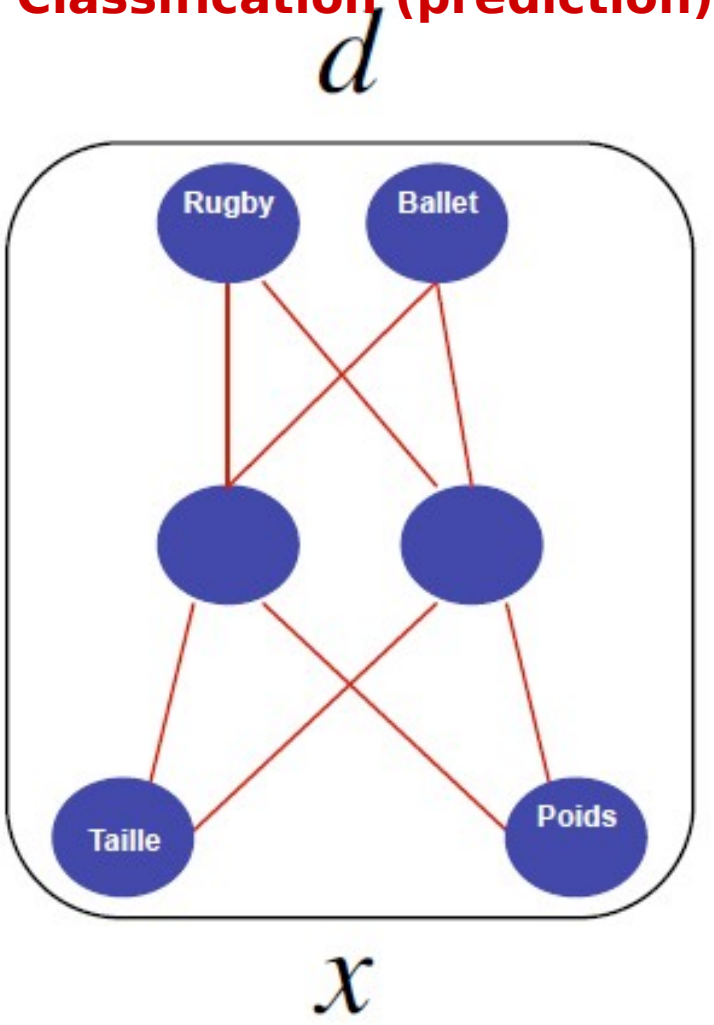

• Autres: cartes de Kohonen,…

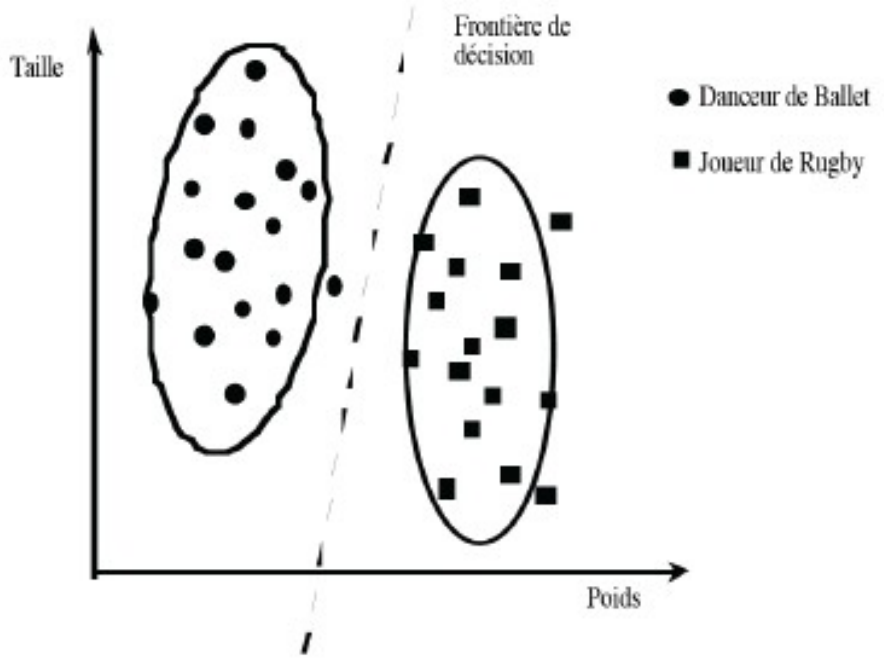


# tilisation des modèles connexionnistes

**Grand nombre de mesures + Loi sous-jacente inconnue**

**Classification (prédiction)**

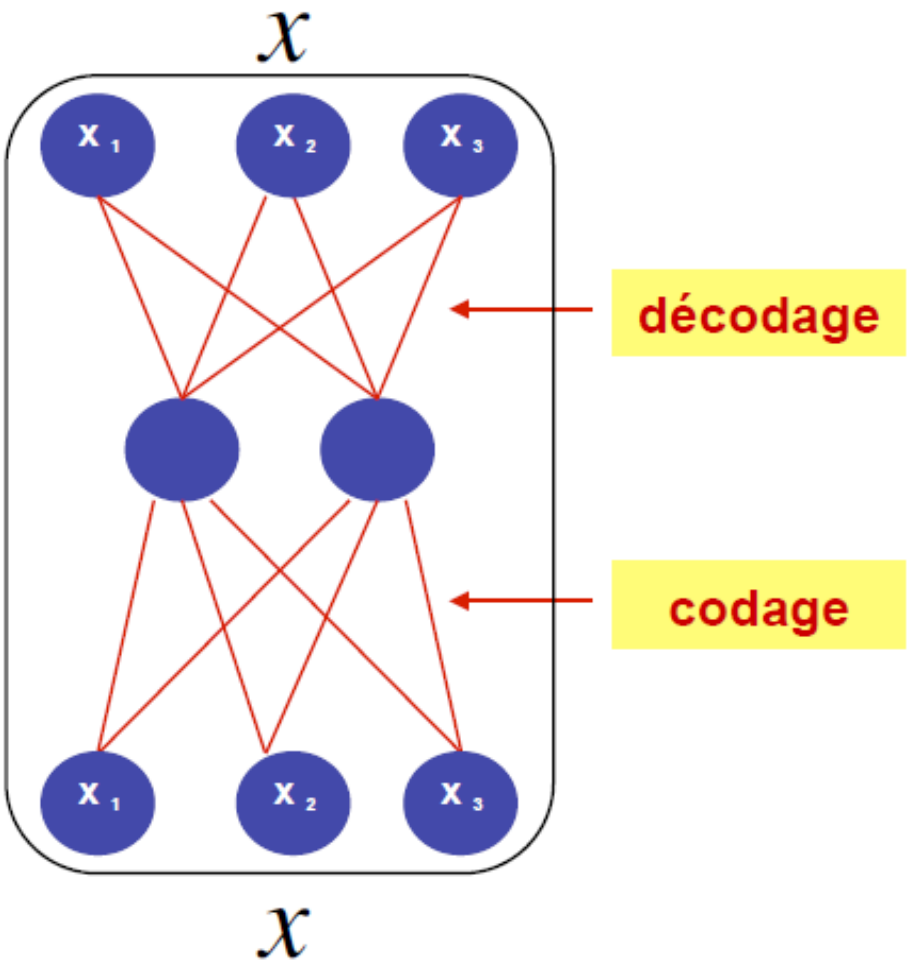


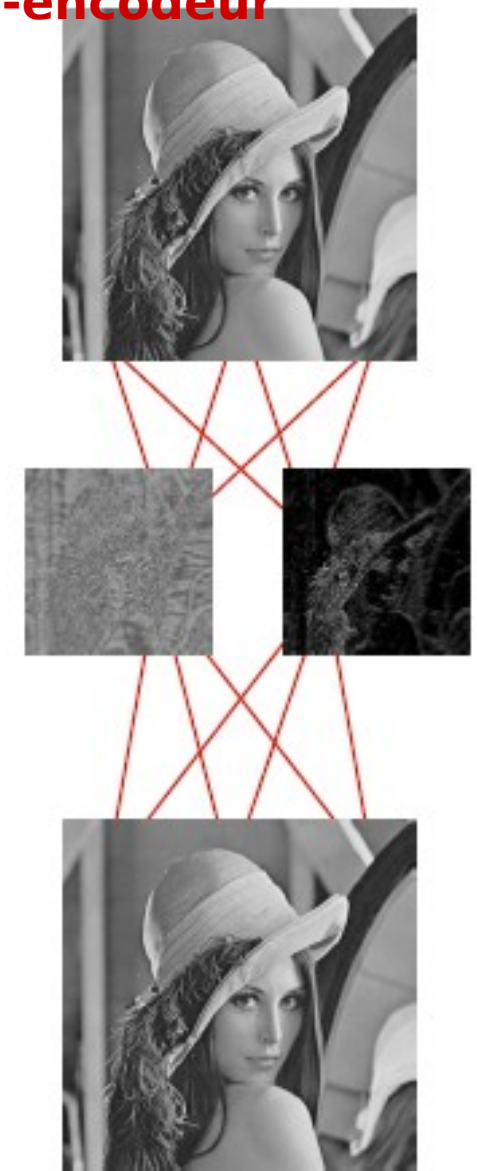



# tilisation des modèles connexionnistes

**Grand nombre de mesures + Loi sous-jacente inconnue**

**Classification/clustering  (prédiction) auto-encodeur**

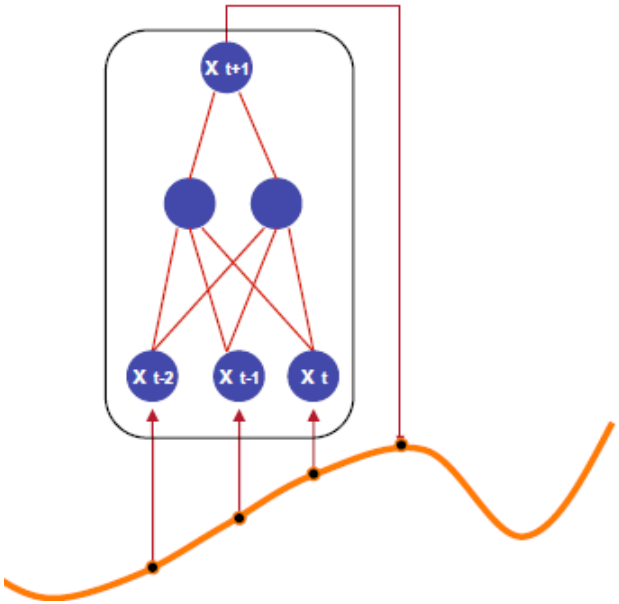


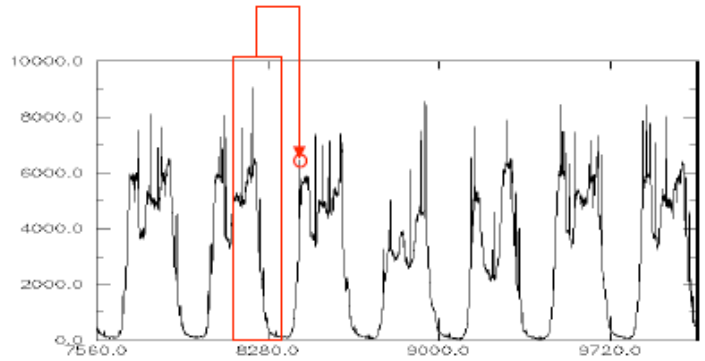



# tilisation des modèles connexionnistes

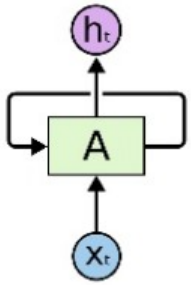
**Grand nombre de mesures + Loi sous-jacente inconnue**

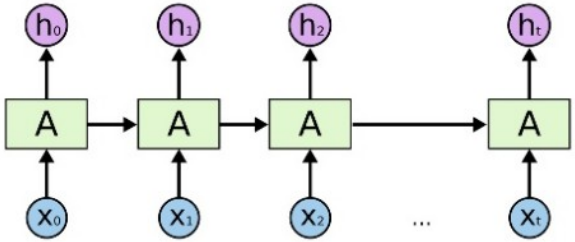
**Regression / Series**



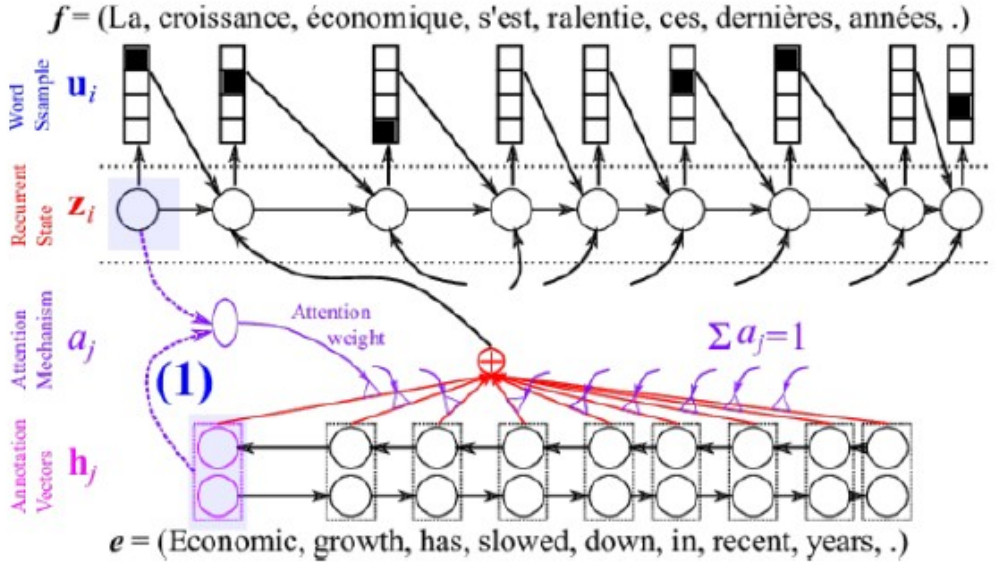




**RNN/LSTM**

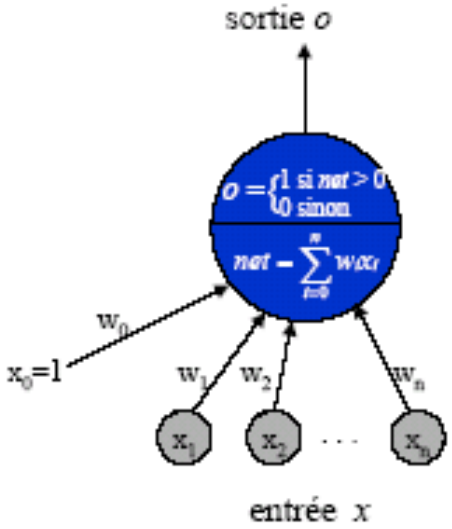



**RNN/LSTM: Traduction**

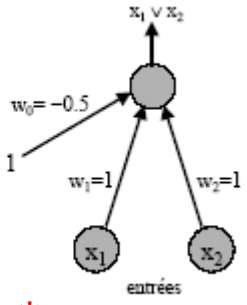


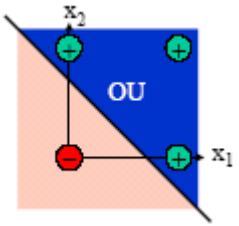
# Le modèle "Perceptron"




Fonctions Logiques, exemple: "ou" logique

| entrée x1 | entrée x2 | sortie |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




$w_0 + w_1 * x_1 + w_2 * x_2$

$-0.5 + x_1 + x_2$

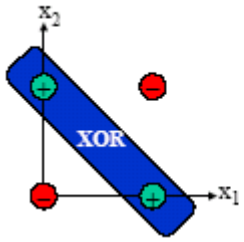
$w_0 = -0.5$

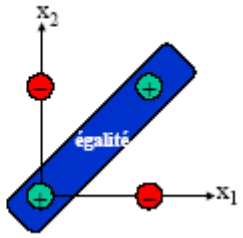
$w_1 = 1$

$w_2 = 1$

Séparabilité linéaire

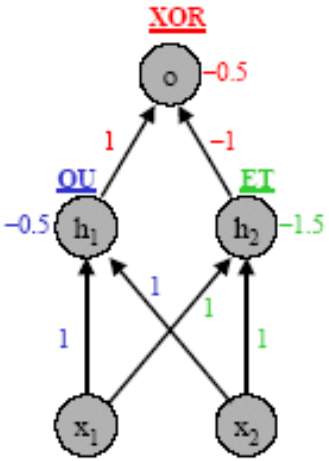
Problème: Non-séparabilité linéaire (XOR, EGALITE)




Solution : XOR

| entrée x1 | entrée x2 | sortie |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# Théorème d'apprentissage (F. Rosenblatt)

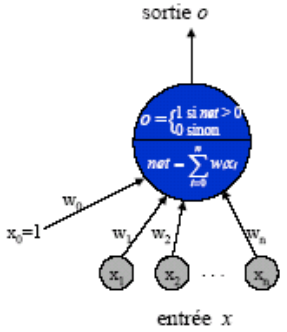
***Étant donné suffisamment d'exemples d'apprentissage, il existe un algorithme qui apprendra n'importe quelle fonction linéairement séparable.***

***Càd : trouver les poids $w_k$ /***

$$w_0 + w_1 * x_1 + w_2 * x_2 + \ldots + w_n * x_n = 0$$

Equation d'un hyperplan

## Algorithme d'apprentissage du Perceptron (correction d'erreur)




**Entrées**: ensemble d'apprentissage $\{(x_1,x_2,\ldots,x_n,t)\}$

**Méthode**

initialiser aléatoirement les poids $w_i$, $0 \leq i \leq n$

répéter jusqu'à convergence:

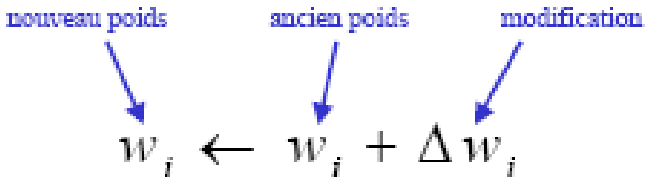
pour chaque exemple

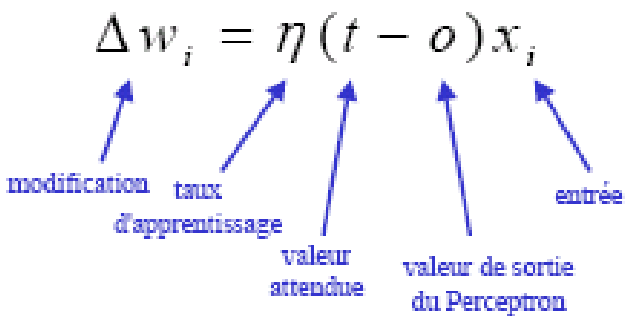
calculer la valeur de sortie $o$ du réseau.

ajuster les poids:

$\Delta w_i = \eta(t - o)\, x_i$

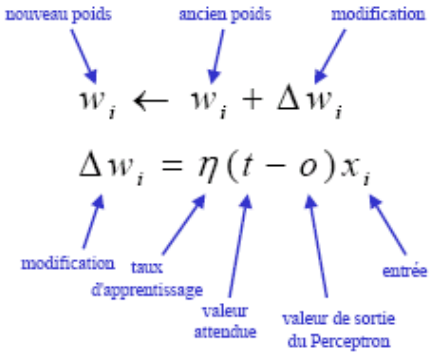
$w_i \leftarrow w_i + \Delta w_i$

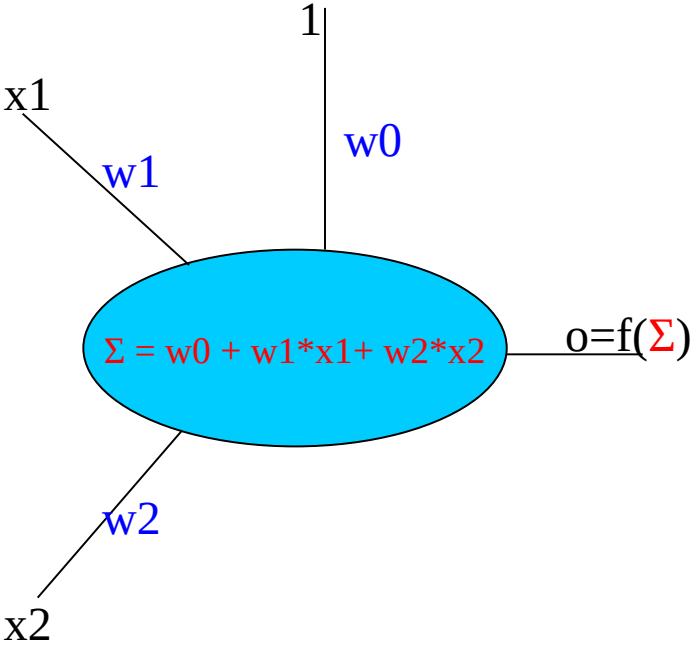



*Règle d'apprentissage du Perceptron ( **WIDROW-HOFF**)*

## EXEMPLE




f(x)= 1 si x > 0

f(x)=0 sinon

η = 1

**Exemple d'apprentissage par correction d'erreur**

| | $w_0$ | $w_1$ | $w_2$ | Entrée | Σ | o | t | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 0 0 | 0 | 0 | 0 | 0 | 0 | 0 |

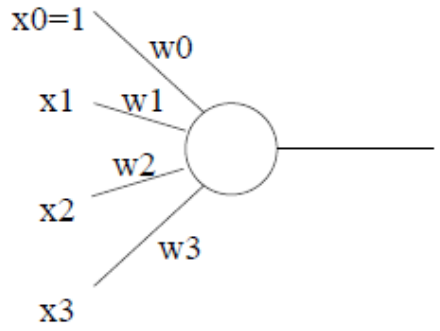
**Fin d'apprentissage:**

**Réseau: (w0,w1,w2) = (0,1,1)**

**La droite: x1+x2 =0 sépare deux classes**

**EXERCICE 1**

Considerez le perceptron et l'echantillon suivant :




| | | | | | classe |
|---|---|---|---|---|---|
| $\vec{y_1}$ | : | 0 | 1 | 0 | 1 |
| $\vec{y_2}$ | : | 1 | 0 | 1 | 0 |
| $\vec{y_3}$ | : | 1 | 0 | 0 | 0 |
| $\vec{y_4}$ | : | 0 | 1 | 1 | 1 |

**Question 1**: Donnez des valeurs aux poids w0,w1,w2 et w3 de sorte que le perceptron classifie correctement l'échantillon. (w0 = ?, w1 = ?, w2 =?, w3 =?)

**Question 2** : Donnez des valeurs aux poids w0,w1,w2 et w3 de sorte que le perceptron classifie correctement l'échantillon inverse (vecteur $\vec{y_1}$ classe 0, $\vec{y_2}$ classe 1, $\vec{y_3}$ classe 1, $\vec{y_4}$ classe 0)
(w0 = ?, w1 = ?, w2 =?, w3 =?)

Considérons l'échantillon suivant:

| | | | | | classe |
|---|---|---|---|---|---|
| $\vec{y_1}$ | : | 1 | 0 | 0 | 1 |
| $\vec{y_2}$ | : | 1 | 0 | 1 | 1 |
| $\vec{y_3}$ | : | 0 | 1 | 1 | 0 |
| $\vec{y_4}$ | : | 0 | 1 | 0 | 1 |

**Question 3** Si on applique l'algorithme d'apprentissage par correction d'erreur en commençant par le vecteur w = (0, 0, 0, 0) on obtient le vecteur de poids suivant : ?

On considère un perceptron spécial avec trois entrées $x_1$ $x_2$, et $x_3$, trois poids $w_1$, $w_2$ et $w_3$ et deux seuils $\theta_1$ et $\theta_2$. La sortie de ce perceptron spécial est 1 si $\theta_1 < w_1x_1+w_2x_2+w_3x_3 <\theta_2$ et 0 sinon.

**Question 4** : Donnez $w_1$, $w_2$, $w_3$, $\theta_1$ et $\theta_2$ de sorte que le perceptron classifie correctement l'échantillon suivant:

| $x_1$ | $x_2$ | $x_3$ | $classe$ |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

SOLUTIONS:

Question 1 Donnez des valeurs aux poids w0,w1,w2 et w3 de sorte que le perceptron classifie correctement l'échantillon. **w0 = 0, w1 = -1, w2 = 1, w3 = 0**

Question 2 Donnez des valeurs aux poids w0,w1,w2 et w3 de sorte que le perceptron classifie correctement l'échantillon inverse (vecteur **~ y1 classe 0, ~ y2 classe 1, ~ y3 classe 1, ~ y4 classe 0**)
**w0 = 0, w1 = 1, w2 = -1, w3 = 0**

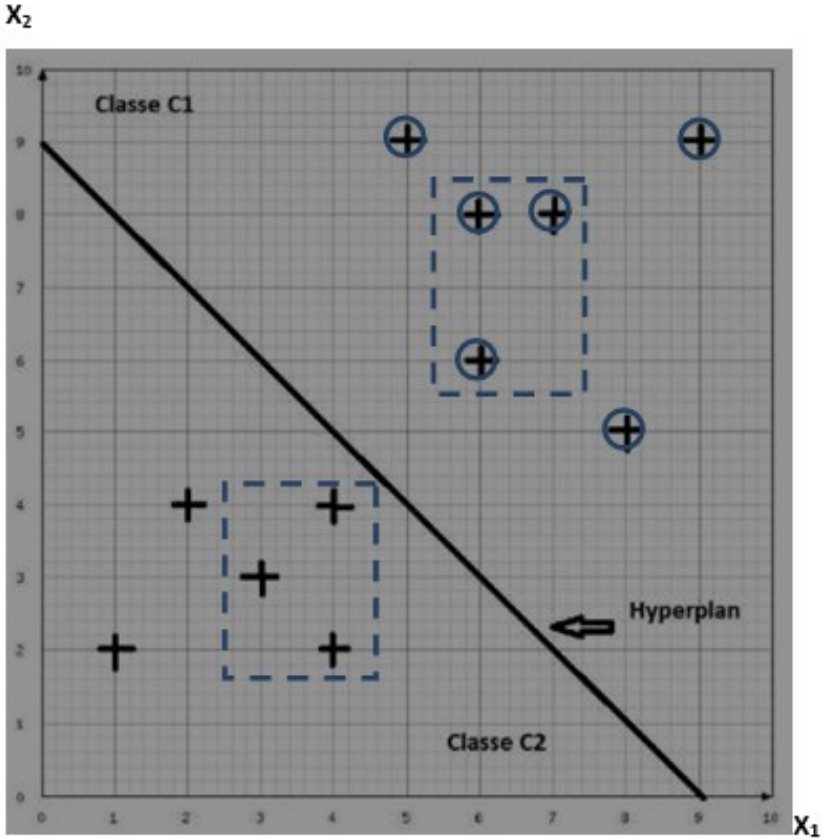
Question 3 Si on applique l'algorithme d'apprentissage par correction d'erreur en commençant par le vecteur ~w = (0; 0; 0; 0) on obtient le vecteur de poids (a quatre dimensions, le premier pour le seuil) suivant : **(1; 1; 0;-1)**

Question 4 Donnez w1, w2, w3, θ 1 et θ 2 de sorte que le perceptron classifie correctement l'échantillon suivant:

**w1 = 0 , w2 = 1 , w3 = 1 , θ1 = 0.5 et θ2 = 1.5**

## Exercice : Perceptron

Considérons le plan ($x_1$,$x_2$) (figure suivante) de points avec deux classes (C1 :⊕et C2 : +)




1) Donnez l'équation de l'hyperplan (droite) séparateur de deux classes de la figure.

2) Considérons les 6 points encadrés (pointillés) de C1 (1) et C2 (0) comme exemples d'entrainement pour un perceptron :

- Donner le schéma de ce perceptron
- Déterminer à l'aide de l'algorithme d'apprentissage l'hyperplan résultant

Indication : à chaque itération on utilise un exemple d'une manière alternative (tantôt C1 tantôt C2) en commençant par C1 et on s'arrête au dernier exemple introduit.

- Comparer les 2 hyperplans et conclure.

# Réseaux multi-couches

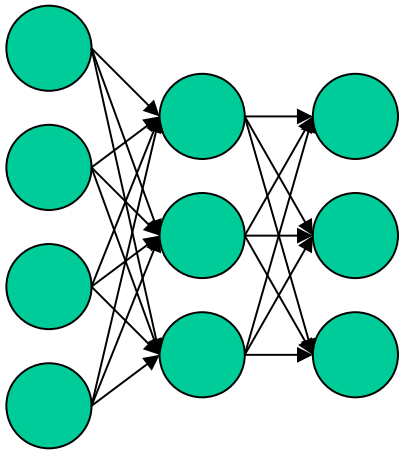
• Un réseau à 2 couches (une couche cachée) avec des unités à seuil peut représenter la fonction logique "ou exclusif" (xor).

• Les réseaux multi-couches "feedforward" peuvent être entraînés par rétro-propagation pour autant que la fonction de transition des unités soit différentiable (les unités à seuil ne conviennent donc pas).

• Il faut apprendre les valeurs des poids $w_i$ qui minimisent l'erreur quadratique

$$E[\vec{w}] = \frac{1}{2} \sum_e (t_e - o_e)^2$$

- Un réseau de neurones à couches cachées est définie par une architecture vérifiant les propriétés:
  - Les cellules sont réparties dans des couches $C_0, C_1, \ldots, C_q$.
  - La première couche $C_0$ (rétine) est composée des cellules d'entrée (correspondent aux $n$ variables d'entrée).
  - Les couches $C_1, \ldots, C_{q-1}$ sont les couches cachées.
  - La couche $C_q$ est composée des cellules de décision.
  - Les entrées d'une cellule d'une couche $C_i$ sont toutes les cellules de la couche $C_{i-1}$ seulement.
- La dynamique du réseau est synchrone par couche.
- Perceptron multi-couches (PMC) linéaires à seuil.
- Chaque fonction booléenne peut être calculée par un PMC linéaire à seuil avec une seule couche cachée.



## L'algorithme de retropropagation du gradient

- On choisit une fonction d'activation dérivable qui s'approche de la fonction de Heaviside:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

- Une cellule élémentaire à $n$ entrée réelles $\vec{x} = (x_1, \ldots, x_n)$ est définie par les poids synaptiques réels $\vec{w} = (w_1, \ldots, w_n)$ et la sortie est calculée par:

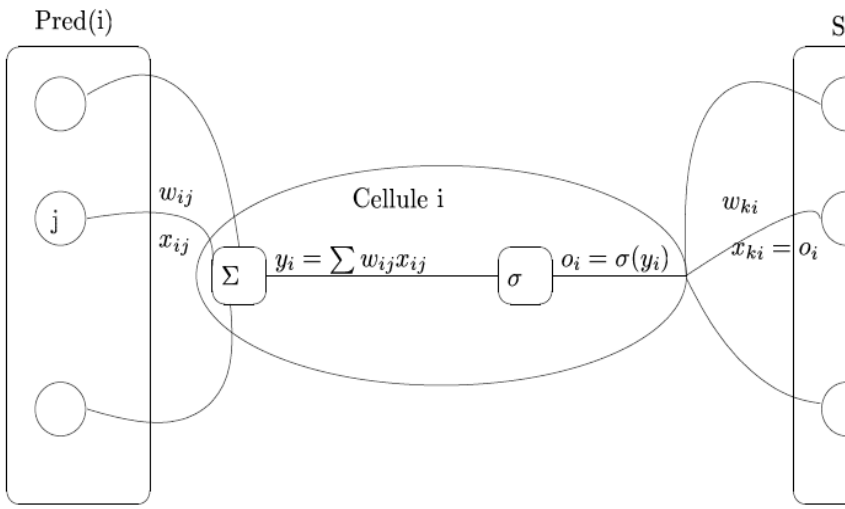
$$o(\vec{x}) = \frac{1}{1+e^{-y}} \text{ avec } y = \vec{x}\vec{w} = \sum_{i=1}^{n} w_i x_i$$

- L'erreur du PMC sur un échantillon $S$ est définie par:

$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x}^s, \vec{c}^s) \in S} \sum_{k=1}^{p} (c_k^s - o_k^s)^2$$

où $o_k^s$ est la $k$-ième composante du vecteur de sortie $\vec{o}$ calculée par le PMC sur l'entrée $\vec{x}$.

## Algorithme de retropropagation du gradient (Notation)




## Calcul de $\frac{\partial E(\vec{w})}{\partial w_{ij}}$

On minimise l'erreur du réseau sur un exemple en appliquant la méthode de descente du gradient. Pour cela, il faut calculer

$$\frac{\partial E(\vec{w})}{\partial w_{ij}} = \frac{\partial E}{\partial w_{ij}}$$

- $w_{ij}$ ne peut influencer la sortie du réseau qu'à travers $y_i$:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} x_{ij}$$

- Il suffit donc de calculer

$$\frac{\partial E}{\partial y_i}$$

- Il y a deux cas: la cellule $i$ est
  - une cellule de sortie
  - une cellule interne

## $i$ est une cellule de sortie

- $y_i$ n'influence la sortie du réseau qu'à travers de $o_i$:

$$\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial y_i}$$

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i}\frac{1}{2}\sum_{k=1}^{p}(c_k - o_k)^2 = \frac{\partial}{\partial o_i}\frac{1}{2}(c_i - o_i)^2 = -(c_i - o_i)$$

$$\frac{\partial o_i}{\partial y_i} = \frac{\partial \sigma(y_i)}{\partial y_i} = \sigma(y_i)(1 - \sigma(y_i)) = o_i(1 - o_i)$$

- On obtient donc:

$$\frac{\partial E}{\partial y_i} = -(c_i - o_i)o_i(1 - o_i)$$

**$O_i$** = sortie calculée

**$C_i$** = sortie désirée

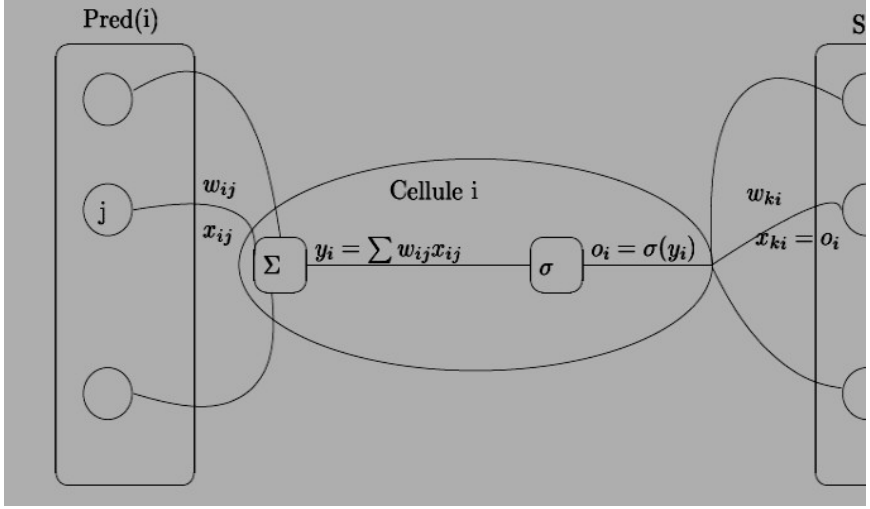
**Couche de sortie, Erreur est:**

**$\delta_i = O_i(1-O_i)(C_i-O_i) = \sigma'(y_i)(C_i-O_i)$**

**Couche Cachée, Erreur est:**

**$\delta_i = O_i(1-O_i)\sum \delta_k W_{ki}$**

## Algorithme de retropropagation du gradient (Notation)




## $i$ est une cellule interne

- $y_i$ influence le réseau par tous les calculs des cellules de $Succ(i)$

$$\frac{\partial E}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial o_i}\frac{\partial o_i}{\partial y_i} = \sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k} w_{ki} o_i(1-o_i)$$

$$= o_i(1 - o_i)\sum_{k \in Succ(i)} \frac{\partial E}{\partial y_k} w_{ki}$$

- On choisit $\Delta w_{ij} = -\epsilon \frac{\partial E(\vec{w})}{\partial w_{ij}}$
- On définit $\delta_i = -\frac{\partial E}{\partial y_i}$
- Cellule de sortie: $\delta_i = o_i(1 - o_i)(c_i - o_i)$
- Cellule interne: $\delta_i = o_i(1 - o_i)\sum_{k \in Succ(i)} \delta_k w_{ki}$

$O_i$ = sortie calculée

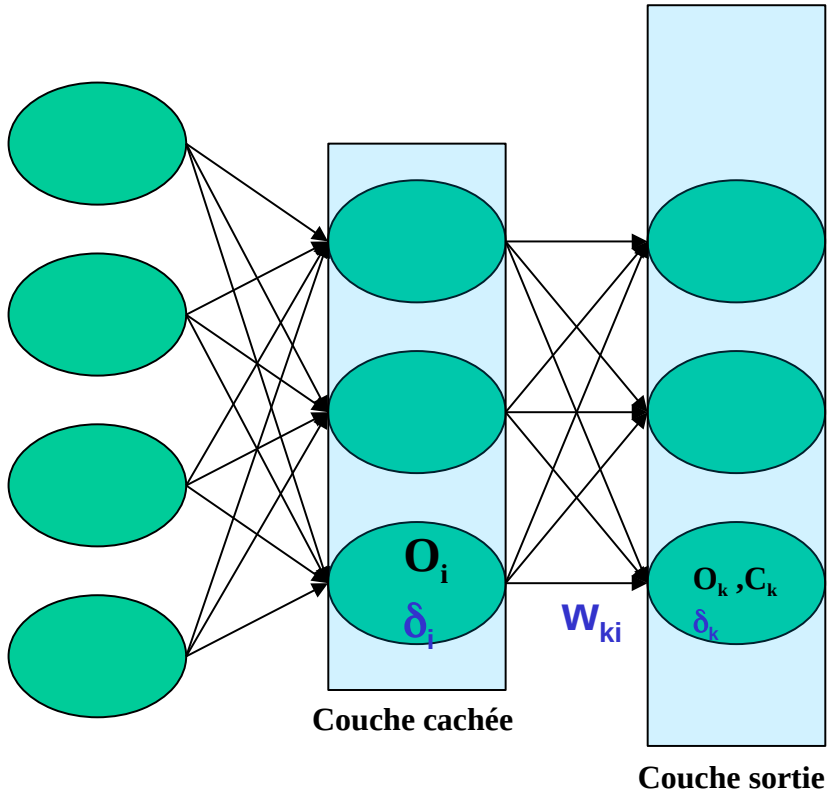
$C_i$ = sortie désirée

**Couche de sortie, Erreur est:**

$$\delta_i = O_i(1-O_i)(C_i-O_i) = \sigma'(y_i)(C_i-O_i)$$

**Couche Cachée, Erreur est:**

$$\delta_i = O_i(1-O_i) \sum_{k \in Succ(i)} \delta_k W_{ki}$$



## Algorithme de retropropagation du gradient

**Entrée:** un PMC avec une couche d'entrée $C_0$, $q-1$ couches cachées $C_1, \ldots,$ une couche de sortie $C_q$, $n$ cellules.

Initialisation aléatoire des poids $w_i$ dans $[-0.5, 0.5]$ pour $i$ de 1 à $n$

**Répéter**

Prendre un exemple $(\vec{x}, c)$ dans $S$ et calculer $\vec{o}$

**Pour** toute cellule de sortie $i$ $\quad \delta_i \leftarrow o_i(1-o_i)(c_i - o_i)$ **FinPour**

**Pour** chaque couche de $q-1$ à 1

**Pour** chaque cellule $i$ de la couche courante

$$\delta_i \leftarrow o_i(1-o_i)\sum_{k \in Succ(i)} \delta_k w_{ki}$$

**fin Pour**

**fin Pour**

**Pour** tout poids $w_{ij} \leftarrow w_{ij} + \epsilon \delta_i x_{ij}$ **fin Pour**

**fin Répéter**

**Sortie:** Un PMC défini par la structure initiale choisie et les $w_{ij}$

## Remarques

- C'est une extension de l'algorithme de Widrow-Hoff
- Peu de problèmes avec les minima locaux
- On peut utiliser un *moment* (momentum)
  - On fixe une constante $\alpha \in [0, 1[$
  - La règle de modification des poids devient

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}(t)$$

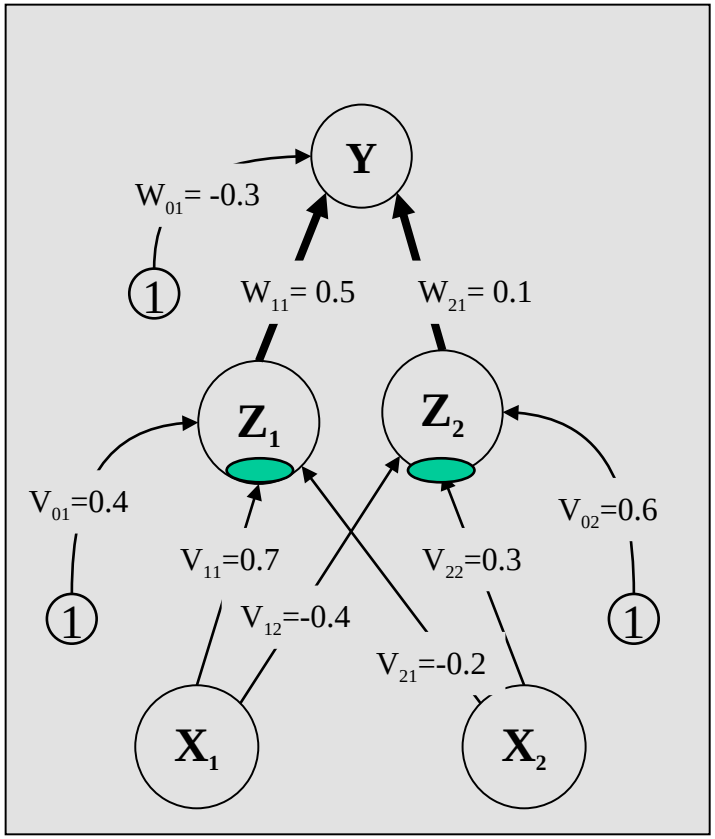
$$\Delta w_{ij}(t) = \epsilon \delta_i x_{ij} + \alpha \Delta w_{ij}(t-1)$$

    où $t$ est un compteur des itérations.
- Quel critère d'arrêt ?
- Ordre de présentation des exemples ?
- Comment choisir les paramètres ?

# Exemple

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1) (réf. Fausett, Prentice Hall, 1994)

**Propagation avant**




**1. Net** de la couche cachée

$\mathbf{z\_in_1 = 1.V_{01} + X_1.V_{11} + X_2.V_{21}}$

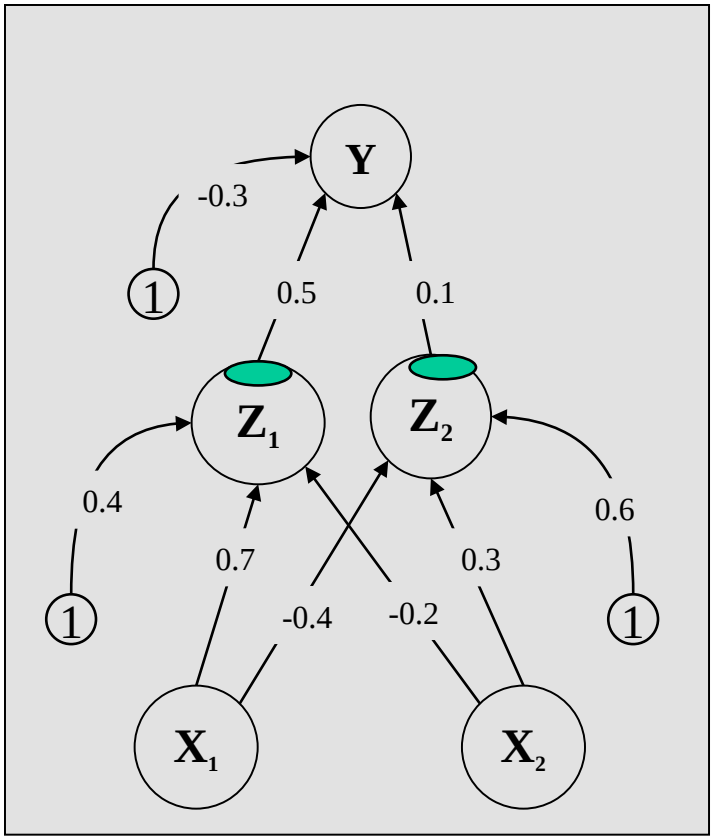
$z\_in_1 = 0.4 + (0.0)\ (0.7) + (1.0)\ (-0.2) = 0.2$

$\mathbf{z\_in_2 = 1.V_{02} + X_1.V_{12} + X_2.V_{22}}$

$z\_in_2 = 0.6 + (0.0)\ (-0.4) + (1.0)\ (0.3) = 0.9$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et $\eta$=0,25.

**Propagation avant**



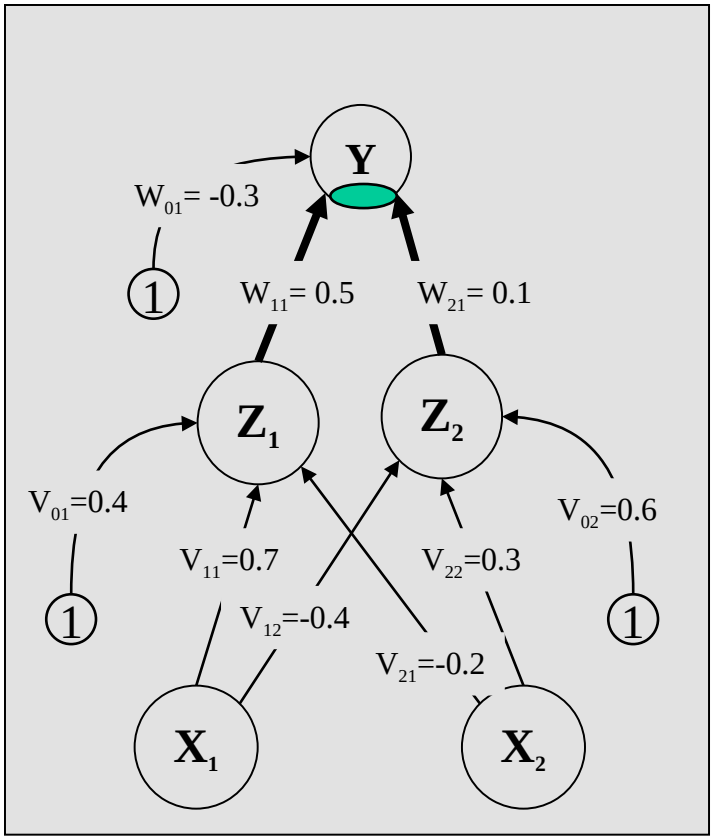

**2. Out** de la couche cachée

$z_1 = 1 / (1+ \exp (- z\_in_1)) = 0.550$

$z_2 = 1 / (1+ \exp (- z\_in_2)) = 0.711$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et $\eta$=0,25.

**Propagation avant**

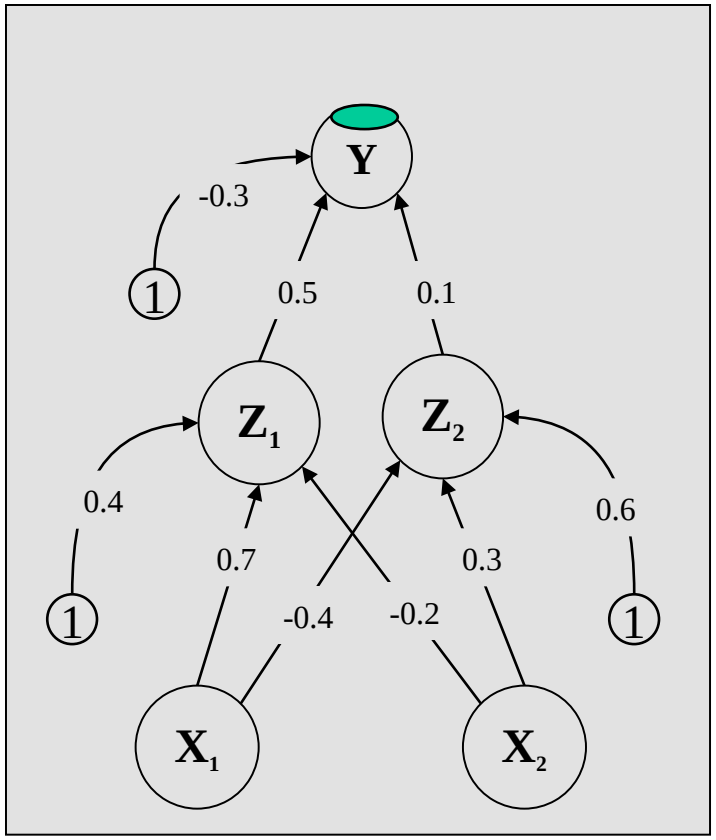



**3. Net** de la couche de sortie

$y\_in = -0.3 + (z_1)\,(0.5) + (z_2)\,(0.1) = 0.046$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et $\eta$=0,25.

**Propagation avant**




**1. Net** de la couche cachée

$z\_in_1 = 0.4 + (0.0)\ (0.7) + (1.0)\ (-0.2) = 0.2$

$z\_in_2 = 0.6 + (0.0)\ (-0.4) + (1.0)\ (0.3) = 0.9$

**2. Out** de la couche cachée

$z_1 = 1 / (1+ \exp (- z\_in_1)) = 0.550$

$z_2 = 1 / (1+ \exp (- z\_in_2)) = 0.711$

**3. Net** de la couche de sortie

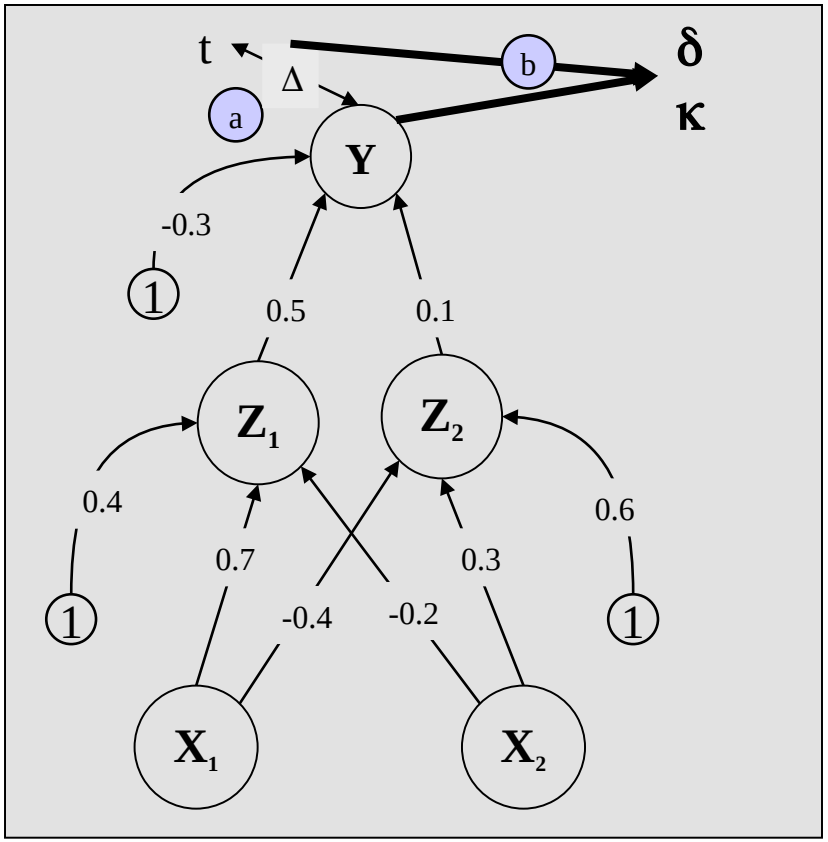
$y\_in = -0.3 + (z_1)\ (0.5) + (z_2)\ (0.1) = 0.046$

**4. Out** de la couche de sortie

$y = 1 / (1+ \exp (- y\_in)) = 0.511$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et η=0,25. Avec valeur désirée t=1.

**Rétro-propagation**




**5. Erreur**

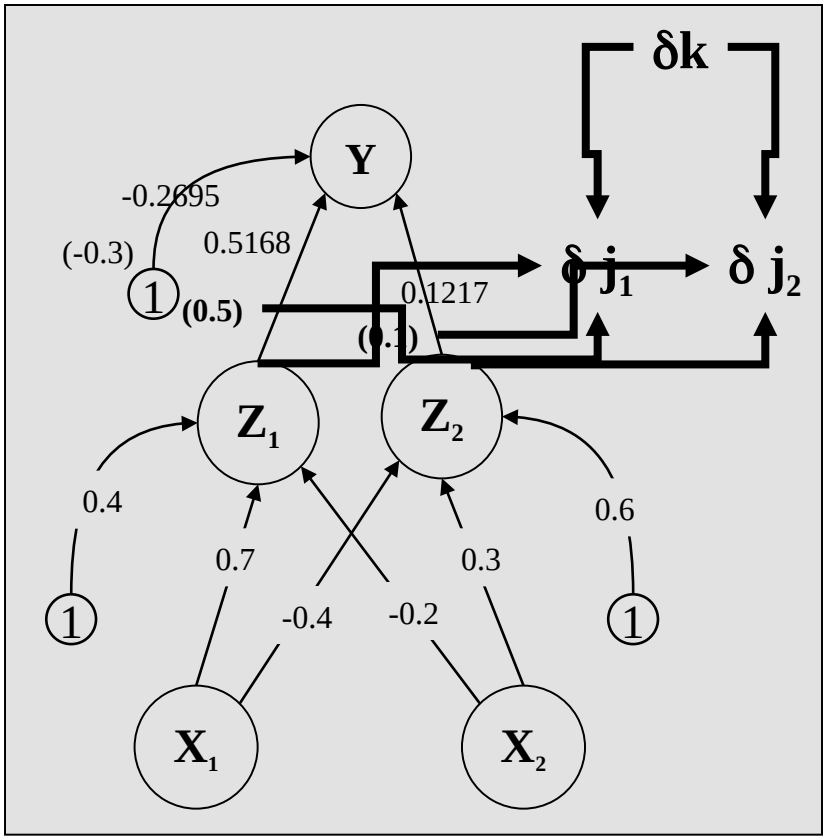
$t - y = 1 - 0.511 = 0.489$

**6. δκ**

$\delta\kappa = (t - y)\,(y)\,(1 - y) = 0.122$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et η=0,25.

**Rétro-propagation**




**Algorithme de retropropagation du gradient**

**Entrée:** un PMC avec une couche d'entrée $C_0$, $q-1$ couches cachées $C_1, \ldots,$ une couche de sortie $C_q$, $n$ cellules.
Initialisation aléatoire des poids $w_i$ dans $[-0.5, 0.5]$ pour $i$ de 1 à $n$
**Répéter**
Prendre un exemple $(\vec{x}, c)$ dans $S$ et calculer $\vec{o}$
**Pour** toute cellule de sortie $i$ $\delta_i \leftarrow o_i(1-o_i)(c_i-o_i)$ **FinPour**
**Pour** chaque couche de $q-1$ à 1
**Pour** chaque cellule $i$ de la couche courante
$\delta_i \leftarrow o_i(1-o_i)\sum_{k \in Succ(i)} \delta_k w_{ki}$
**fin Pour**
**fin Pour**
**Pour** tout poids $w_{ij} \leftarrow w_{ij} + \epsilon \delta_i x_{ij}$ **fin Pour**
**fin Répéter**
**Sortie:** Un PMC défini par la structure initiale choisie et les $w_{ij}$

Dans le cas général :

$$\sum_{k=1}^{m} d_k w_{jk}$$

**Dérivée de f(z_in$_j$)**

**8. δ $j_1$**

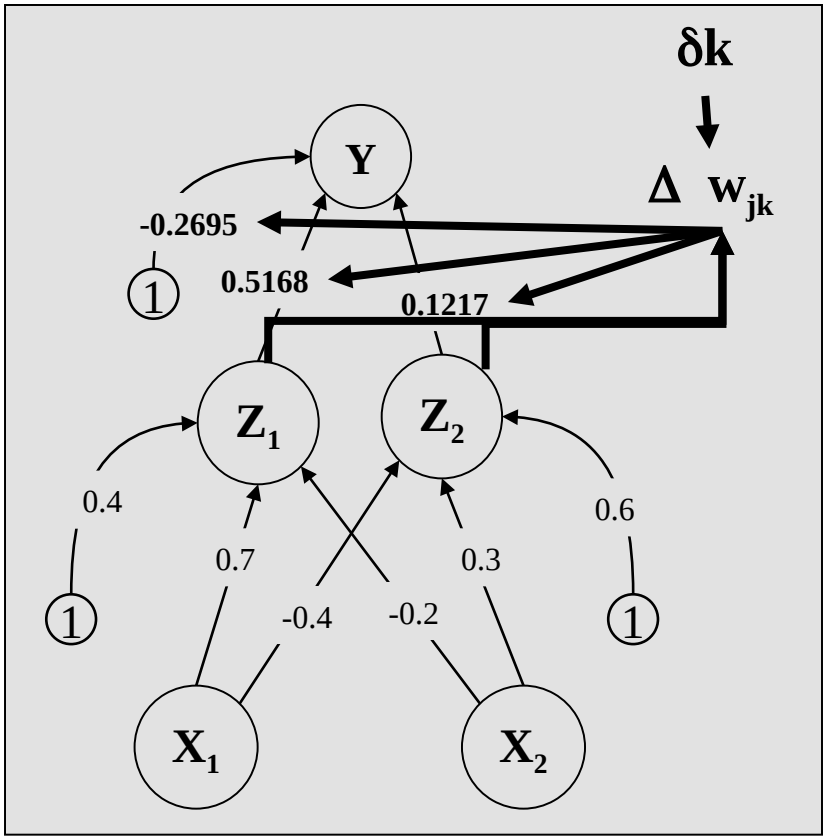
$$\delta j_1 = (\delta\ k)\,(w_{11})\,\mathbf{(z_1)\,(1 - z_1)} = 0.015$$

**9. δ $j_2$**

$$\delta j_2 = (\delta\ k)\,(w_{21})\,(z_2)\,(1 - z_2) = 0.0025$$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et $\eta$=0,25.

**Rétro-propagation**




**7.** $\Delta\ w_{jk}$

$\Delta\ w_{01} = (\eta)\ (\delta k) = 0.0305$

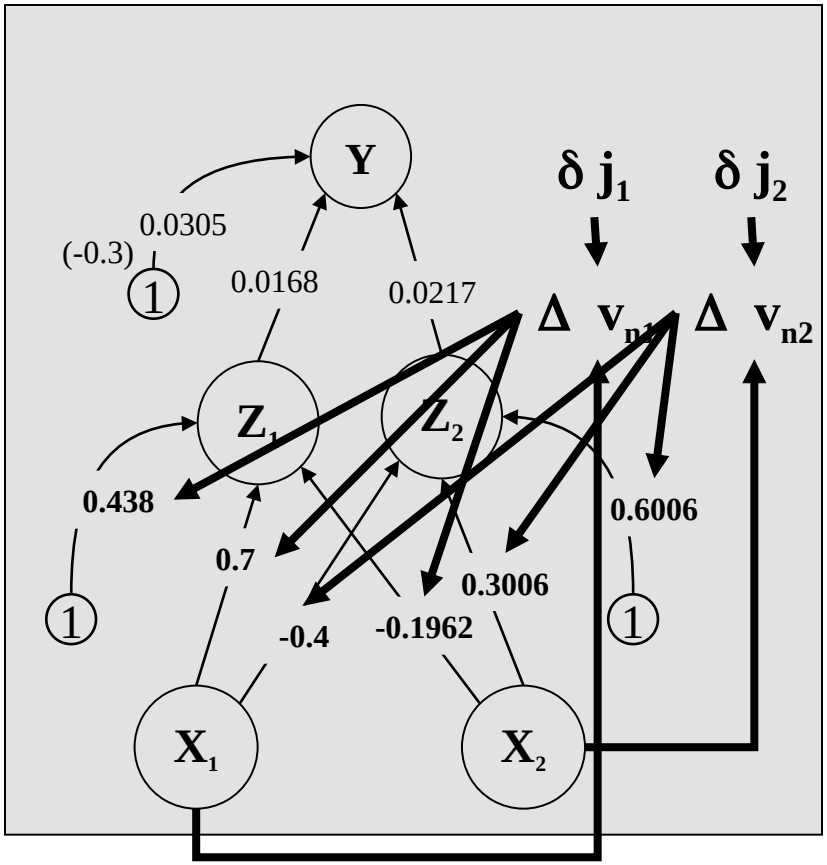
$\Delta\ w_{11} = (\eta)\ (\delta k)\ (z_1) = 0.0168$

$\Delta\ w_{21} = (\eta)\ (\delta k)\ (z_2) = 0.0217$

$$w_{jk} = w_{jk} + \Delta\ w_{jk}$$

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (0,1). t=1 et $\eta$=0,25.

**Rétro-propagation**




**10.**

**Δ** $v_{np}$

$\Delta v_{01} = (\eta)(\delta j_1) = 0.038$

$\Delta v_{11} = (\eta)(\delta j_1)(x_1) = 0.0$

$\Delta v_{21} = (\eta)(\delta j_1)(x_2) = 0.038$

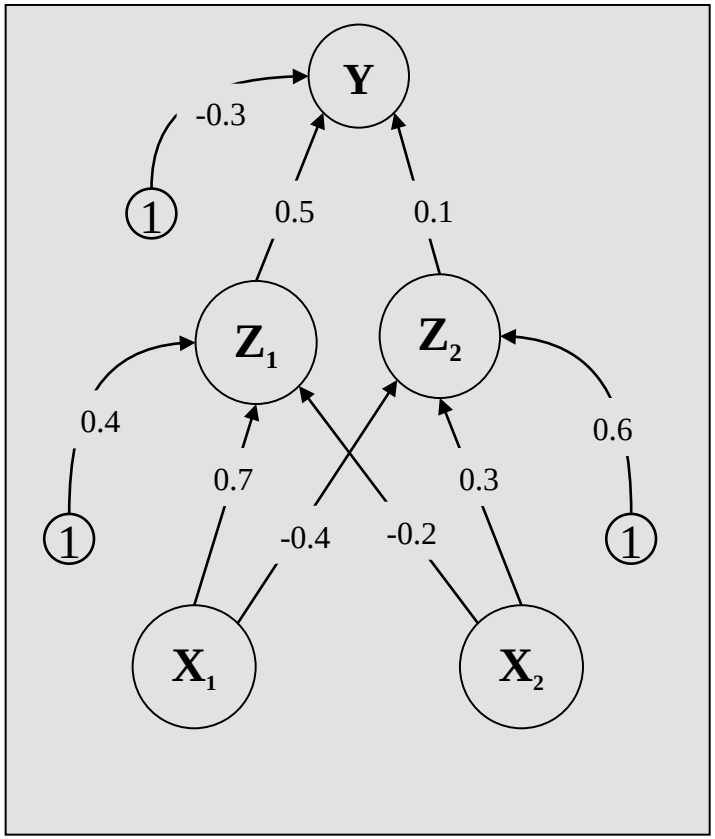
$\Delta v_{02} = (\eta)(\delta j_2) = 0.0006$

$\Delta v_{12} = (\eta)(\delta j_2)(x_1) = 0.0$

$\Delta v_{22} = (\eta)(\delta j_2)(x_2) = 0.0006$

# Exercice à faire:

Trouver les nouveaux poids du réseau de la figure ci-dessous si on présente le vecteur d'apprentissage (-1,1) et on utilise une sigmoïde bipolaire comme fonction d'activation



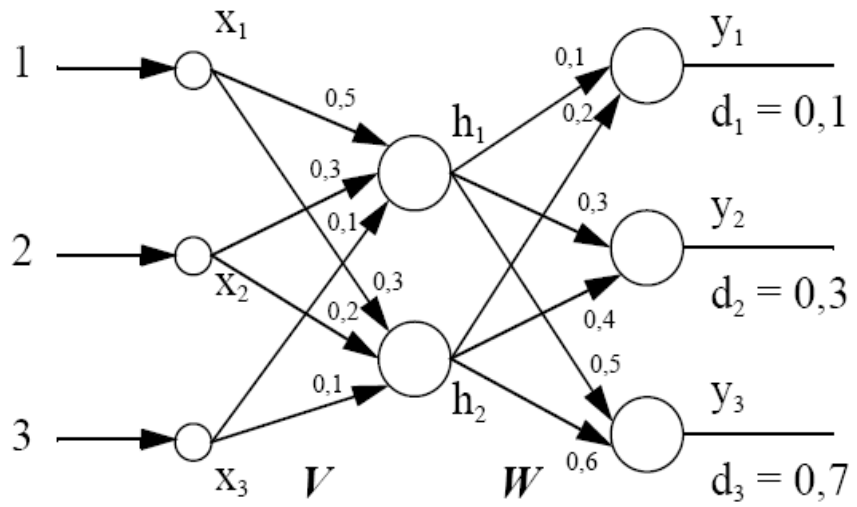

**Seuls changent la dérivée de la fonction d'activation bipolaire et la mise à jour des poids entre l'entrée et la couche cachée.**

$$\textit{sigmoïde bipolaire}: y = f(a) = \tanh\left(\frac{a}{2}\right)$$

$$f'(a) = 1/2*(1+y)(1-y)$$

**Exercice2**

Soit le perceptron multicouche suivant (**V, W matrices poids et $d_i$ sortie désirée**):




Dans l'unique but de simplifier les calculs, les neurones ne sont pas munis de l'habituel paramètre de polarisation (seuil). Les poids de connexion affichés directement sur la connexion sont résumés dans les deux matrices de connexion :

$$V = \begin{bmatrix} 0,5 & 0,3 & 0,1 \\ 0,3 & 0,2 & 0,1 \end{bmatrix} \qquad W = \begin{bmatrix} 0,1 & 0,2 \\ 0,3 & 0,4 \\ 0,5 & 0,6 \end{bmatrix}$$

Pour le vecteur d'entrée $\mathbf{(1,2,3)^t}$ :

- Donner les erreurs à la sortie (calcul intermédiaire) : $\delta_1$, $\delta_2$ et $\delta_3$
- **Calculez les nouvelles valeurs de poids des $V_{11}$ et $W_{11}$ par rétropropagation du gradient.**

**Les paramètres du réseau sont :**

$$\eta = 1 \qquad f(net) = \frac{1}{1 + e^{-net}}$$

**net = somme pondérée au niveau d'une cellule.**
**$\eta$ : coefficient d'apprentissage**

solution

le stimulus $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ doit donner la réponse $\mathbf{t} = \begin{bmatrix} .1 \\ .3 \\ .7 \end{bmatrix}$

Cette activation est ensuite convertie en réponse. En utilisant la fonction logistique, on obtient:

$$\mathbf{h} = f(\mathbf{b}) = \begin{bmatrix} 0.8022 \\ 0.7311 \end{bmatrix} . \qquad \textbf{(VI.15)}$$

Cette activation est ensuite transmise aux cellules de la couche de sortie. Elles calculent leur activation:

$$\mathbf{a} = \mathbf{Z}\mathbf{h} = \begin{bmatrix} 0.2264 \\ 0.5331 \\ 0.8397 \end{bmatrix} , \qquad \textbf{(VI.16)}$$

elles la transforment en réponse en utilisant la fonction logistique:

$$\mathbf{o} = f(\mathbf{a}) = \begin{bmatrix} 0.5564 \\ 0.6302 \\ 0.6984 \end{bmatrix} . \qquad \textbf{(VI.17)}$$

Les cellules de sortie peuvent évaluer leur *signal d'erreur*:

$$\boldsymbol{\delta}_{\text{sortie}} = f'(\mathbf{a}) \circledast \mathbf{e} = \mathbf{o} \circledast (\mathbf{1} - \mathbf{o}) \circledast (\mathbf{t} - \mathbf{o})$$
$$= \begin{bmatrix} -0.1126 \\ -0.0770 \\ 0.0003 \end{bmatrix} .$$

Réponse :

$$V = \begin{bmatrix} 0,4946 & 0,2892 & 0,0837 \\ 0,2896 & 0,1791 & 0,0687 \end{bmatrix} \qquad W = \begin{bmatrix} 0,0096 & 0,1177 \\ 0,2383 & 0,3437 \\ 0,5003 & 0,6002 \end{bmatrix}$$

## Exercice 3

On considère un "perceptron" avec entrées $x_1, x_2, \ldots, x_n$ et poids $w_1, w_2, \ldots, w_n$. La sortie $o$ est donnée par

$$o = w_1(x_1 + x_1^3) + w_2(x_2 + x_2^3) \cdots + w_n(x_n + x_n^3)$$

- Donnez un algorithme d'apprentissage basé sur la descente du gradient pour un tel "perceptron". Indication : Définissez d'abord une fonction d'erreur standard. Expliquez votre façon d'obtenir l'algorithme.

On considère un tel "perceptron" avec 2 entrées $x_1$ et $x_2$, les poids $w_1 = 1$ et $w_2 = -1$. L'échantillon d'apprentissage consiste en un vecteur $x_1 = 1, x_2 = 1$ et la sortie désirée est 1.

- Détaillez le premier pas de votre algorithme.

solution

$$o = w_1(x_1 + x_1^3) + w_2(x_2 + x_2^3) \cdots + w_n(x_n + x_n^3)$$

*Il suffit de définir l'erreur sur un échantillon comme*

$$E(\vec{w}) = \frac{1}{2} \sum_{(\vec{x}^s, c^s)} (c^s - o^s)^2$$

*Ensuite on prend les dérivées partielles par rapport aux $w_i$. Ça donne:*

$$\frac{\partial(\vec{w})}{\partial w_i} E = \sum_S (c^s - o^s)(-x_i^s - (x_i^s)^3)$$

*Et après on obtient facilement un algorithme d'apprentissage en remplaçant dans l'algorithme d'apprentissage par descente du gradient le calcul du $\Delta w_i$ par $\Delta w_i \leftarrow \Delta w_i + \epsilon(c^s - o^s)(x_i^s + (x_i^s)^3)$.*

# Question ?

*Quand utiliser des réseaux de neurones artificiels pour résoudre des problèmes d'apprentissage?*

• lorsque les données sont représentées par des paires attribut-valeur,

• lorsque les exemples d'apprentissage sont bruités,

• lorsque des temps d'apprentissage (très) longs sont acceptables,

• lorsqu'une évaluation rapide de la fonction apprise est nécessaire,

.

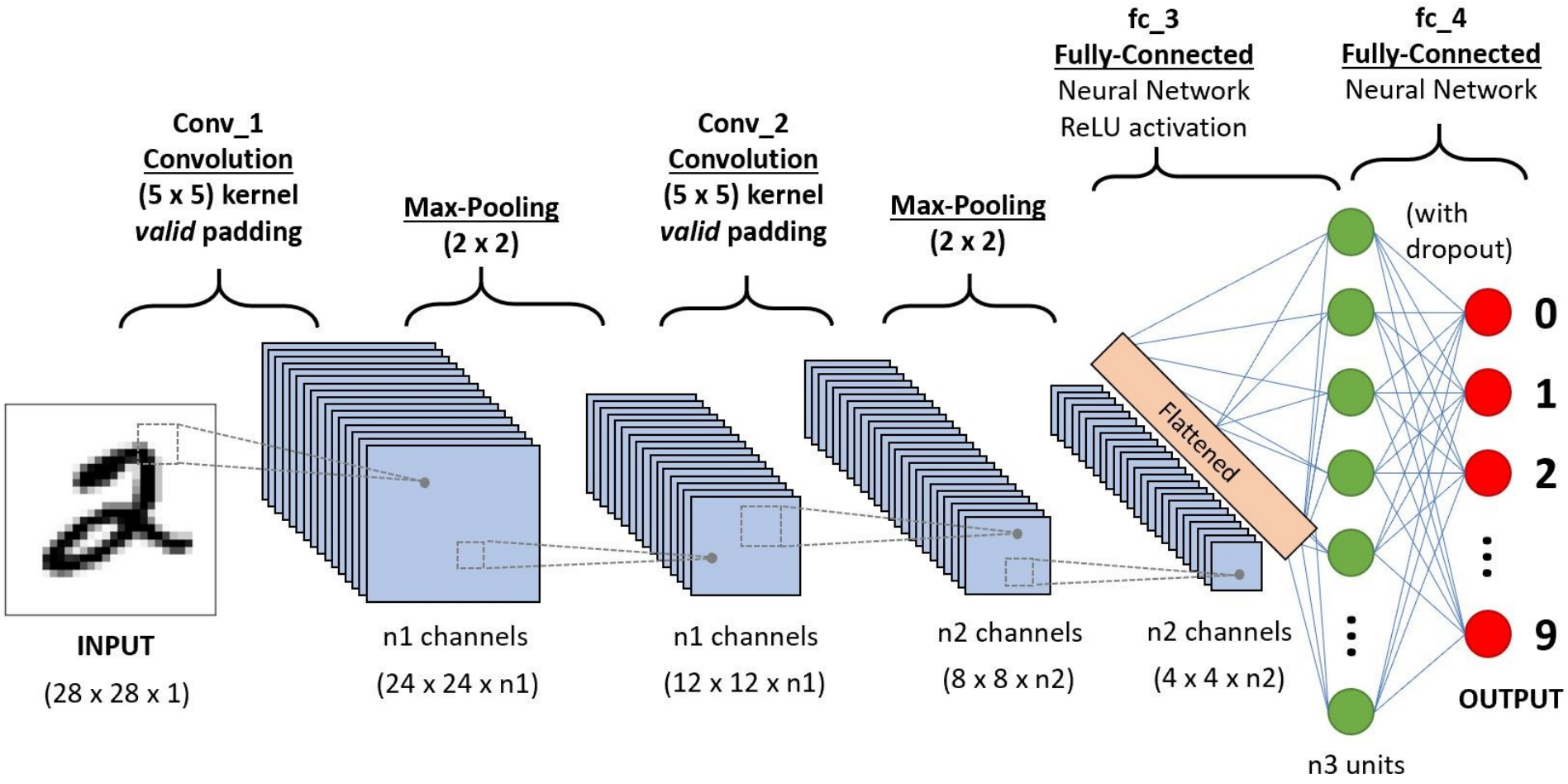
# Deep Learning

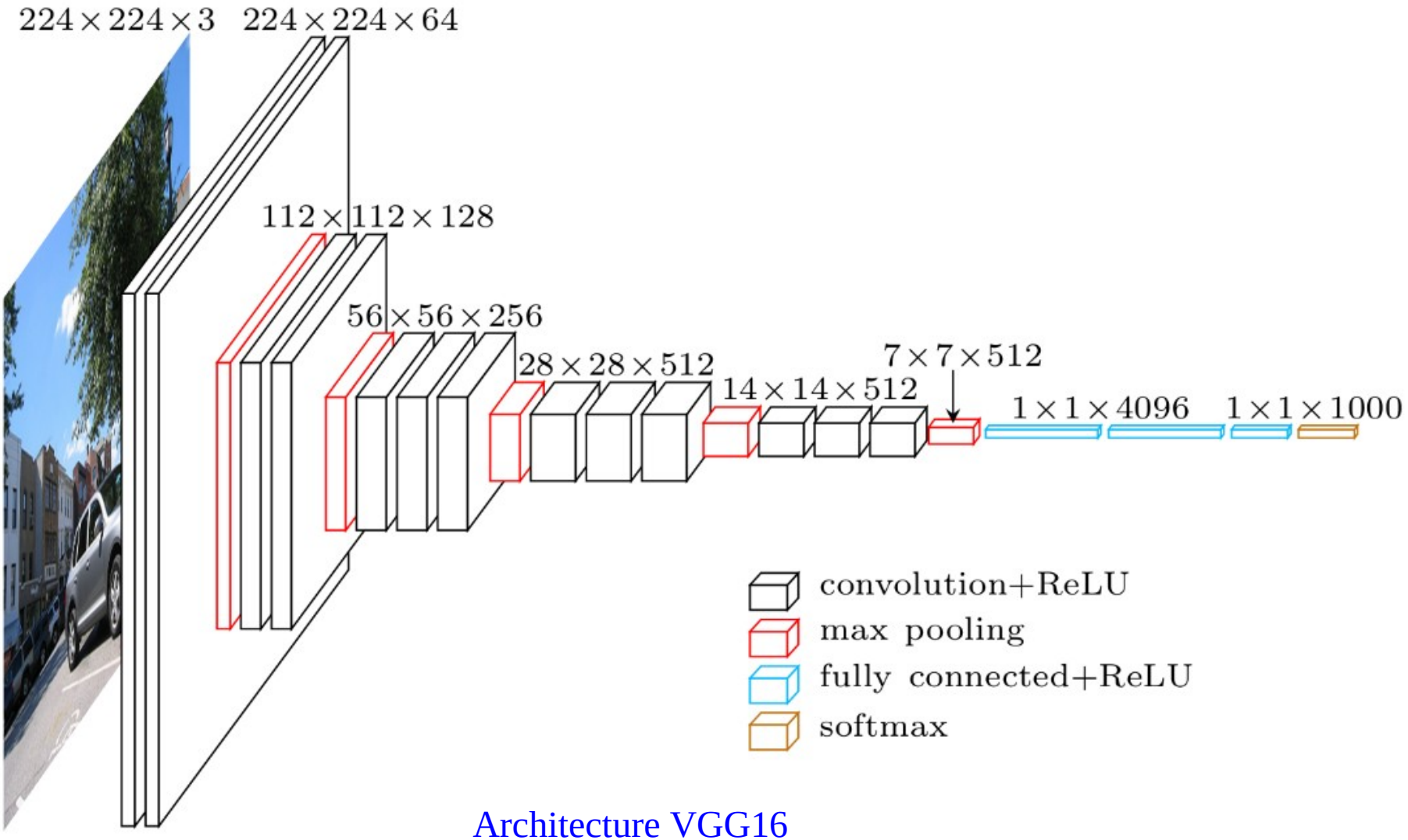
# Deep Learning (Travaux pionniers)

- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks
- Many layered **MLP with backpropagation**
  - Tried early but without much success
    - Lent
    - Diffusion du gradient

- Présentation du deep networks avec apprentissage **non supervisé** ***(Auto-encoder, machine de boltzmann,….)***

# Convolutional Neural Networks (CNN)  bien adapté à l'imagerie





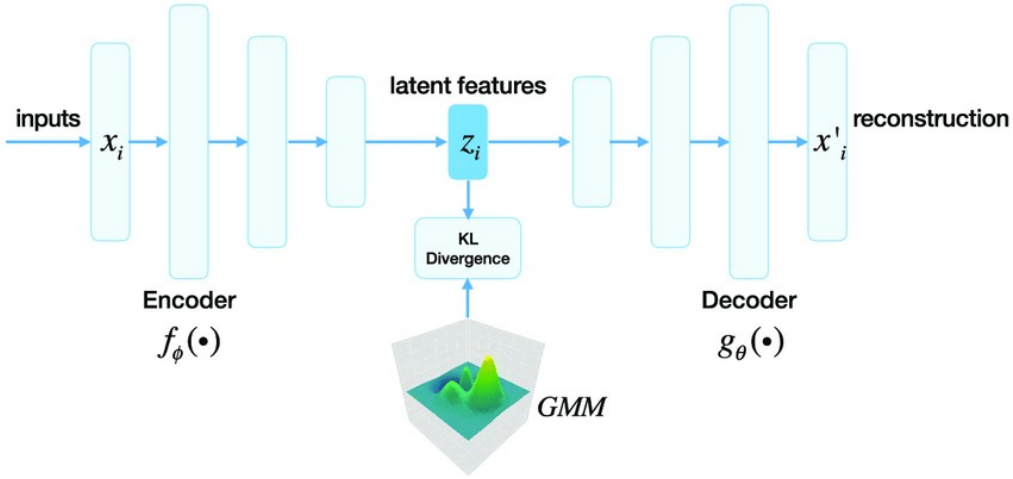

Architecture VGG16

Framework : TensorFlow – Keras (python)

# Apprentissage des Deep Networks

- Construire un espace de représentation (feature space)
- Notez que c'est ce que nous faisons avec les noyaux SVM, ou les couches cachées dans MLP, etc, mais maintenant, nous allons construire l'espace de représentation en utilisant les architectures profondes.
- Apprentissage non supervisé entre les couches peut décomposer le problème en sous-problèmes distribués (avec des niveaux d'abstraction plus élevés) à être encore décomposé à des couches successives

**Stacked Auto-Encoders (CNN)**



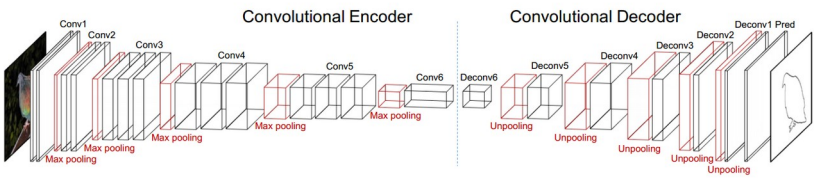




Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

# Applications des RNA

- Contrôle
  - conduite automatique de véhicules:
    - Alvinn: réseau de pilotage d'un véhicule à partir d'images vidéo,
  - synthèse vocale:
    - NETtalk: un réseau qui apprend à prononcer un texte en anglais
  - processus de fabrication/production,

- Reconnaissance/classification/analyse
  - textes imprimés, caractères manuscrits (codes postaux), parole,
  - images fixes (ex. visages), animées ou clips vidéo,
  - risques (financiers, naturels, etc.)

- Prédiction
  - économie, finances, analyse de marchés, médecine,
  - …

# Bibliographie


• Hopfield, J. J. (1982). *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy of Sciences 79:2554-2558.

• Hertz, J., A. Krogh, and R. G. Palmer. (1991). *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley.

• Rumelhart, D. E., J. McClelland, and the PDP Research Group. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

• Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. New York: Oxford University Press.

• Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. New York: Macmillan College Publishing.

• Churchland, P. S., and T. J. Sejnowski. (1992). *The Computational Brain*. Cambridge, MA: MIT Press.

• Arbib, A. M. (1995). *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press.

• Sejnowski, T. (1997). *Computational neuroscience. Encyclopedia of Neuroscience*. Amsterdam: Elsevier Science Publishers.

# Démos

• Animations Java
http://www.cim.mcgill.ca/~jer/courses/java.html

• Approximation de fonctions
http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.html

• Apprentissage Perceptron
http://diwww.epfl.ch/mantra/tutorial/english/perceptron/html/

• Reconnaissance de caractères
http://sund.de/netze/applets/BPN/bpn2/ochre.html

• Réseaux auto-organisés
http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html

• Réseaux de Hopfield
http://www.physics.syr.edu/courses/modules/MM/sim/hopfield.html