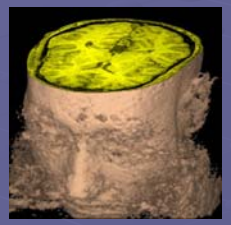
# Les réseaux de neurones artificiels

Andrei Doncescu

# Human Beings Think v.s. Intelligent Machines Think

## How Human Beings Think ?



Deduction
Abduction
Induction

Limited by
Computational
and
Memory Capacity

The genius people are able to mix these three fundamental modes of reasoning

## How Intelligent Machines Think ?

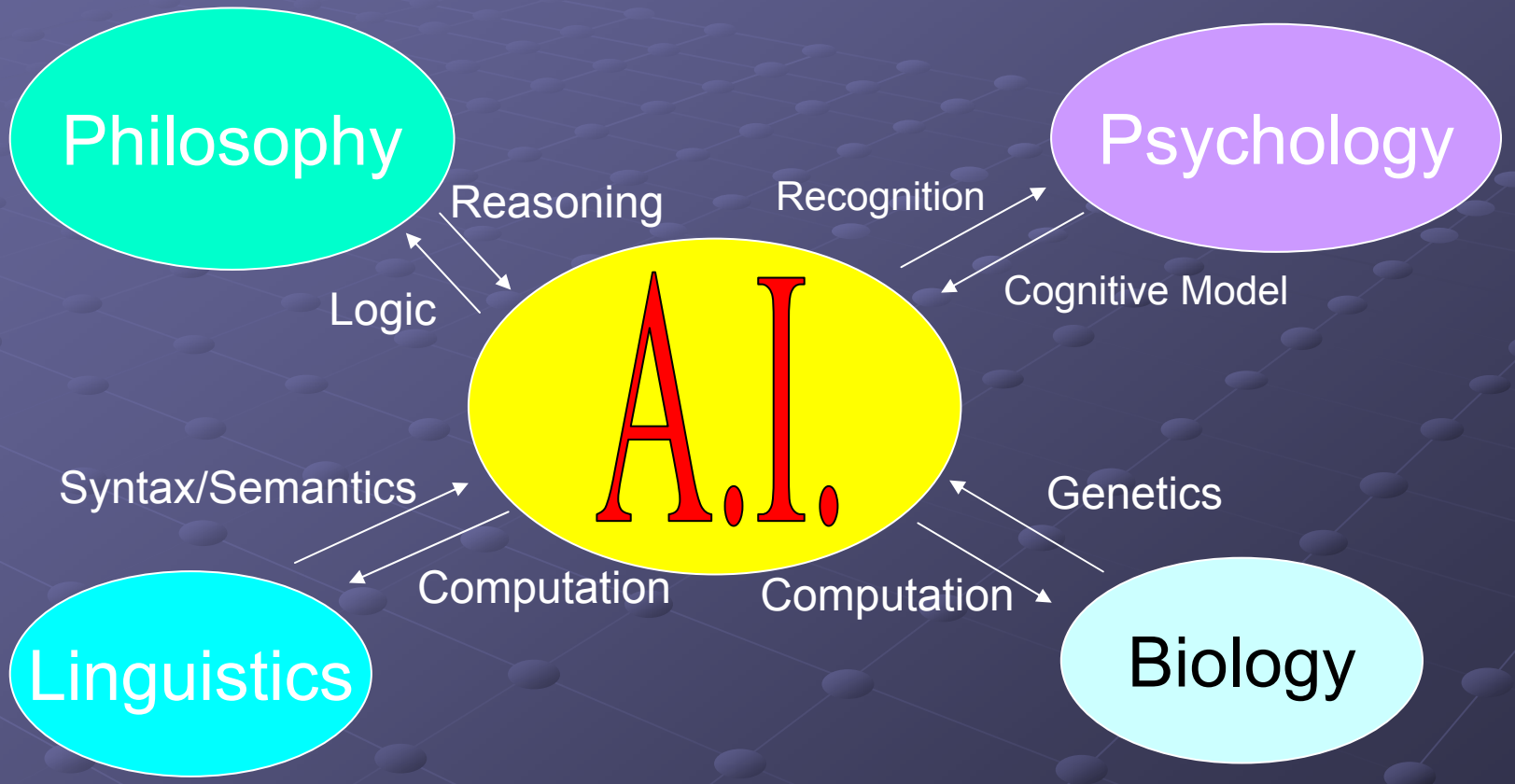
Induction
Abduction
Deduction

- Diagnosis
- Design
- Characterization
- Discovery
- Verification

**Discovery in Huge Data**

Combination of Induction & Abduction → One of the most powerful theoretical answers for the next generation of Intelligent Machine (Inoue 2001,2004)

# Position of AI in Science




**Goal of A.I. :**

1. **Scientific goal : elucidation of human intelligence**
2. **Engineering goal : automated computation**

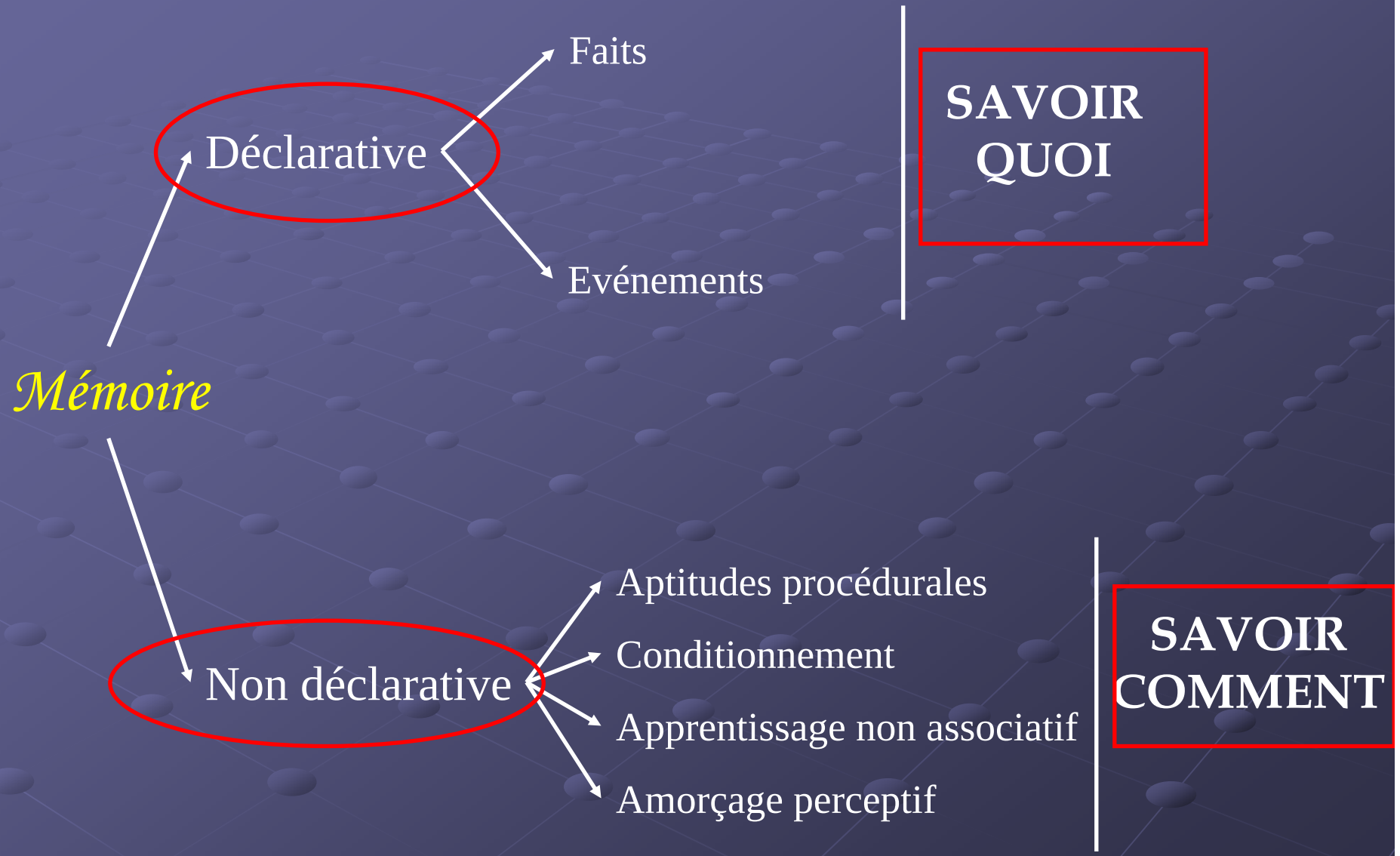
# Modèles de connaissances

*Modéliser : remplacer du visible compliquer par de l'invisible simple*

- Boîte noire : il peut avoir une valeur prédictive, dans un certain domaine de validité mail il n'a aucune valeur explicative. Il requiert seulement des mesures. On peut parler de prévision.

- Boîte blanche : prédictif et explicatif

# Apports des neurosciences (Squire, 1992)



# Lois fondamentales de l'apprentissage

- *La loi de de l'exercice, l'usage ou la fréquence pose que plus une situation est fréquemment associé à une réponse, plus cette association tend à se renforcer*

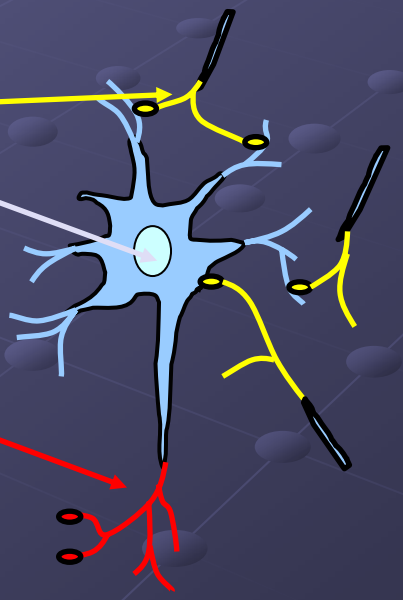
- *La loi de l'effet pose que l'association stimulus/réponse tend à renforcer quand les conséquences en sont positives et s'affaiblir quand les conséquences sont négatives*

# Fondements Biologiques
# Structure des neurones

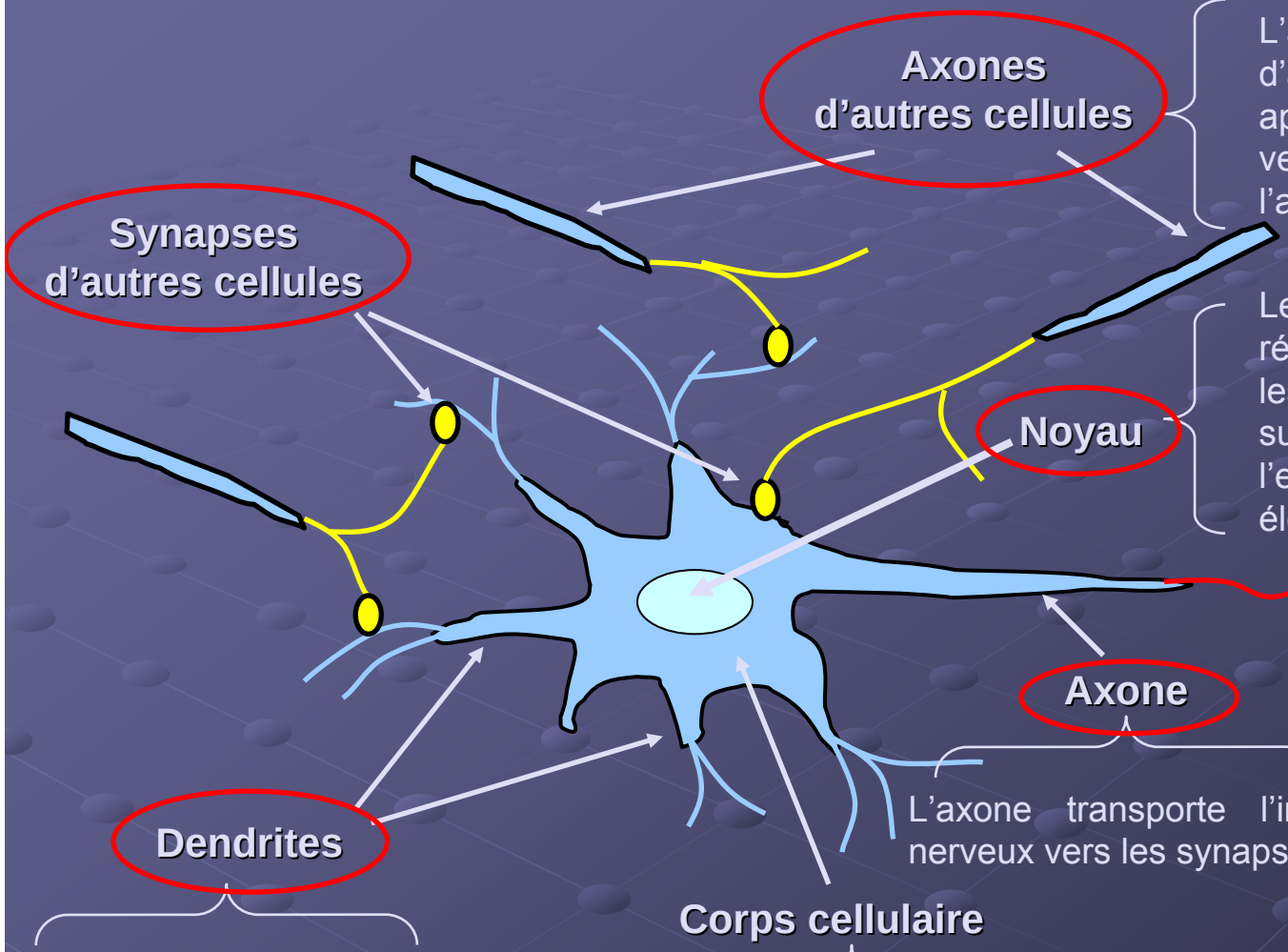
Le système nerveux est composé de $10^{12}$ neurones interconnectés. Bien qu'il existe une grande diversité de neurones, ils fonctionnent tous sur le même schéma.

Ils se décomposent en trois régions principales :

- Le corps cellulaire
- Les dendrites
- L 'axone



# Qu'est-ce qu'un neurone biologique?




L'axone et les synapses d'autres cellules nerveuses apportent des influx nerveux venant exciter ou inhiber l'activité du neurone.

Le noyau est le centre des réactions électrochimiques. Si les stimulations externes sont suffisantes, le noyau provoque l'envoi d'un influx nerveux électrique à travers l'axone.

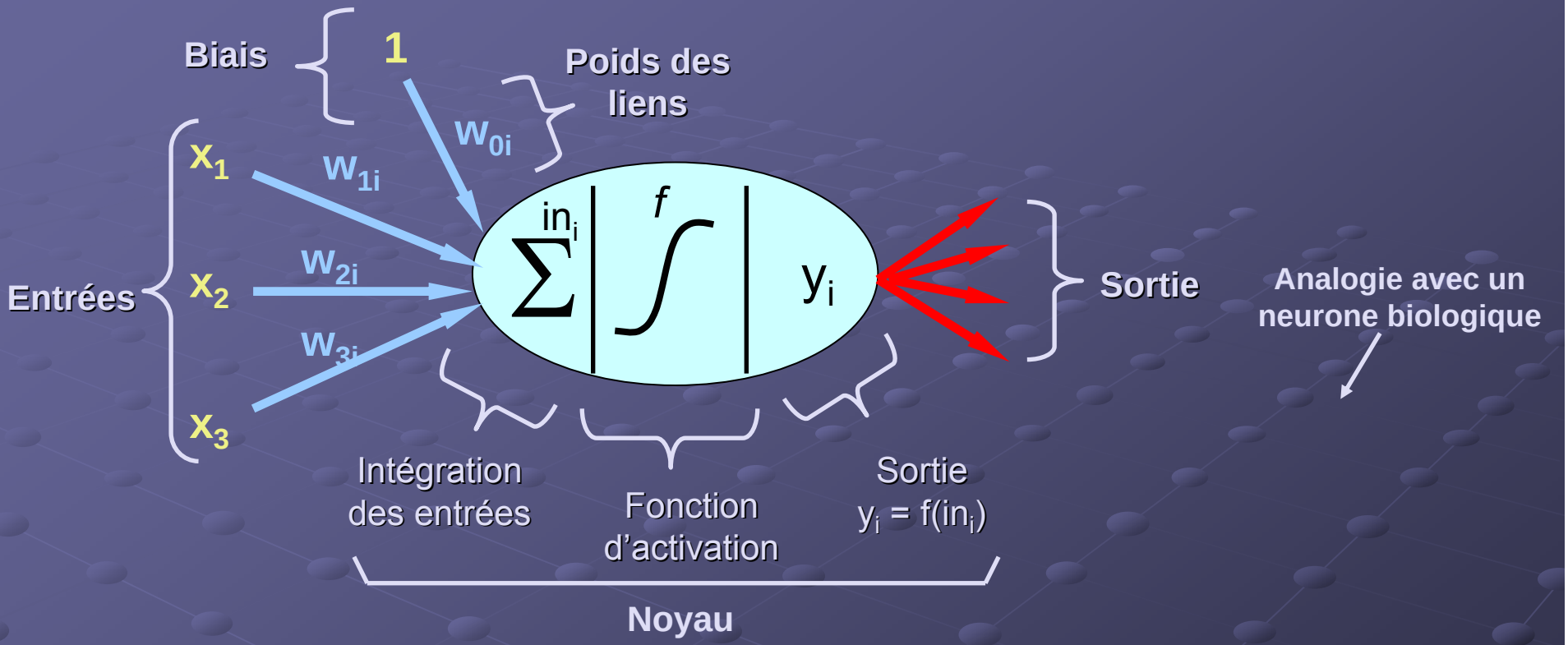
L'axone transporte l'influx nerveux vers les synapses.

Les dendrites, selon leur longueur et leur perméabilité, affectent la quantité d'influx nerveux qui se rend au noyau.

Le corps cellulaire participe aux réactions électrochimiques avec le noyau.

Les synapses transmettent l'influx nerveux provenant de l'axone vers d'autres cellules à partir de neurotransmetteurs inhibiteurs ou excitateurs.

# Qu'est-ce qu'un neurone artificiel ?




**Entrées :** Directement les entrées du système ou peuvent provenir de d'autres neurones.

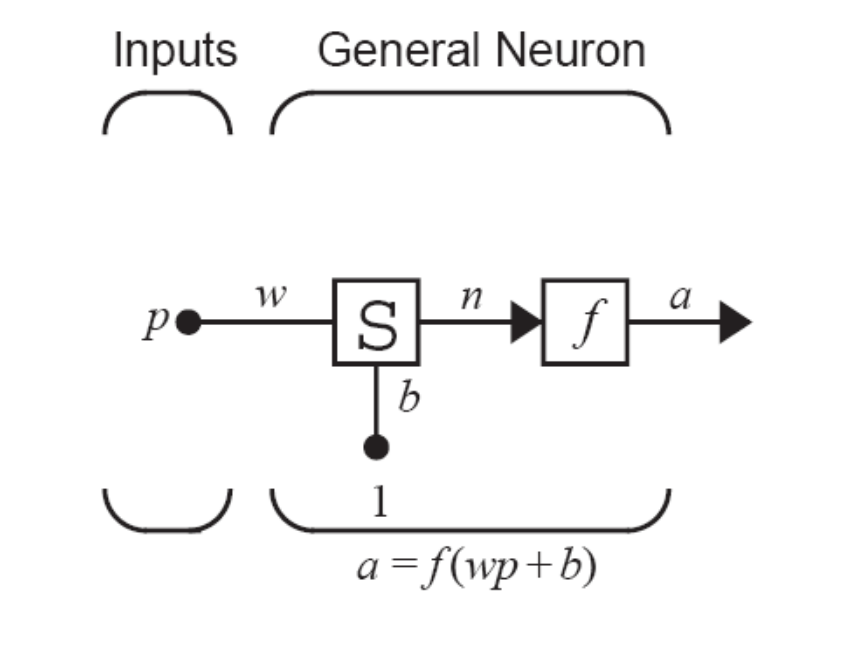
**Biais :** Entrée toujours à 1 qui permet d'ajouter de la flexibilité au réseau en permettant de varier le seuil de déclenchement du neurone par l'ajustement du poids du biais lors de l'apprentissage.

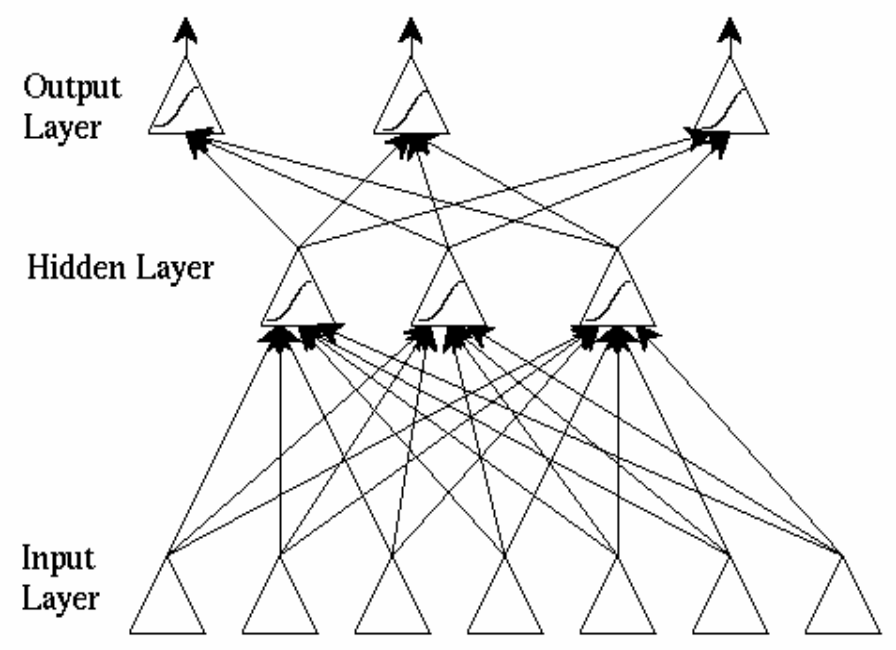
**Poids :** Facteurs multiplicateurs qui affectent l'influence de chaque entrée sur la sortie du neurone.

**Noyau :** Intègre toutes les entrées et le biais et calcule la sortie du neurone selon une fonction d'activation qui est souvent non-linéaire pour donner une plus grande flexibilité d'apprentissage.

**Sortie :** Directement une des sorties du système ou peut être distribuée vers d'autres neurones.

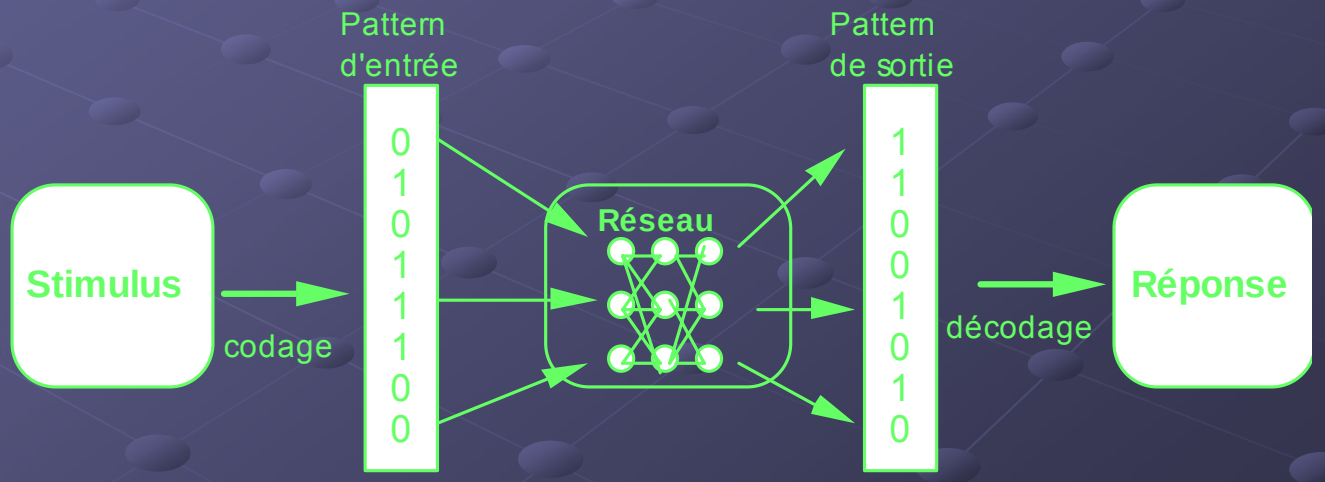
# Qu'est-ce qu'un réseau de neurones ?



# Définition :

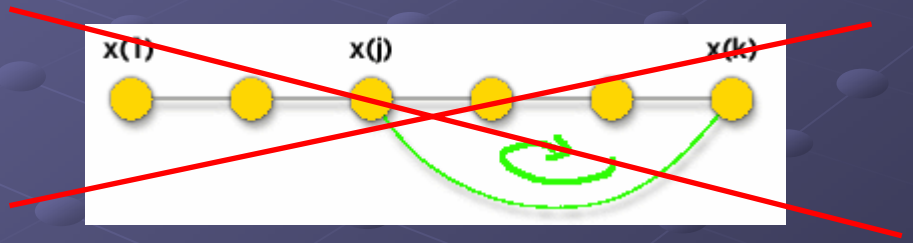
- *Les réseaux de neurones artificiels sont des réseaux fortement connectés de **processeurs élémentaires fonctionnant en parallèle.***
- *Chaque processeur élémentaire calcule **une sortie unique** sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau.*



# Topologie des Réseaux de Neurones

- Nous pouvons considérer une RNA comme un graphe acyclique, donc il n y a pas de chemin qui commence et fini dans le même nœud , mais …

- Un cycle est un chemin simple rebouclant sur lui-même. Un graphe dans lequel il n'existe aucun cycle est dit *acyclique*.

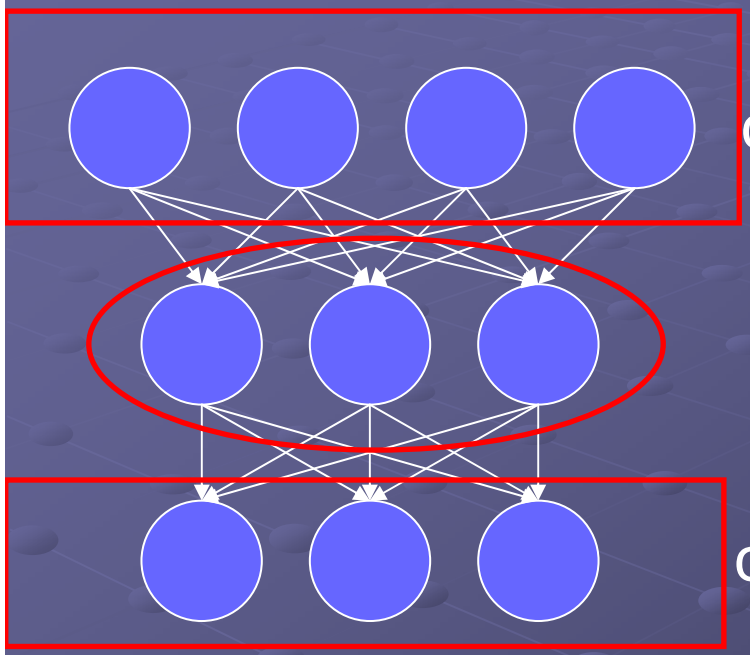



- Un graphe acyclique ***G*** à ***n*** sommets possède au plus ***n*-1** arêtes.
- Un réseau de neurones formels peut se représenter par un graphe orienté *valué* (chaque arc porte une valeur), et dont chaque sommet est valué aussi de son *seuil d'activation*.

# Topologies de réseaux

- Feed-forward : propagation vers l'avant de l'information
  - Réseau multi-couche, dans lequel chacun des neurones d'une couche (*n-1*) est connecté à tous les neurones de la couche n → réseau multi-perceptrone
- Récurents : réseaux bouclés

# Structure d'Interconnexion (feedforward)

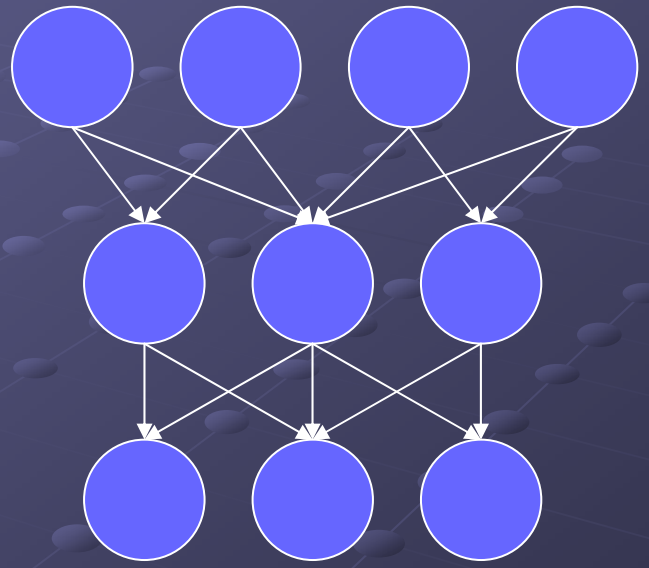


couche d'entrée

couche cachée

couche de sortie

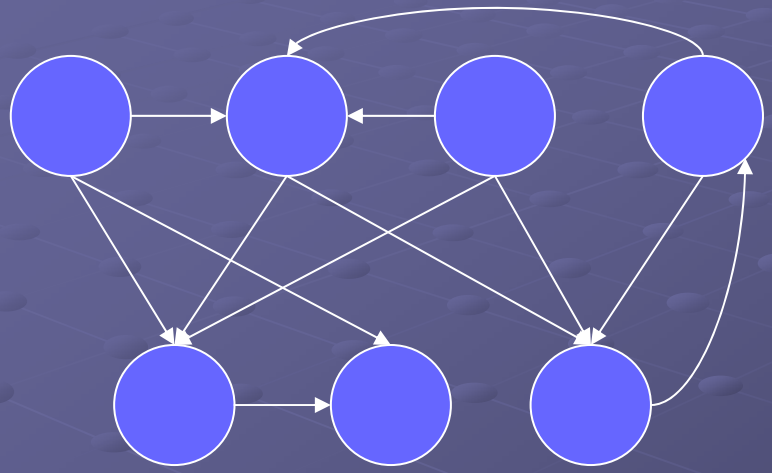
réseau multicouche

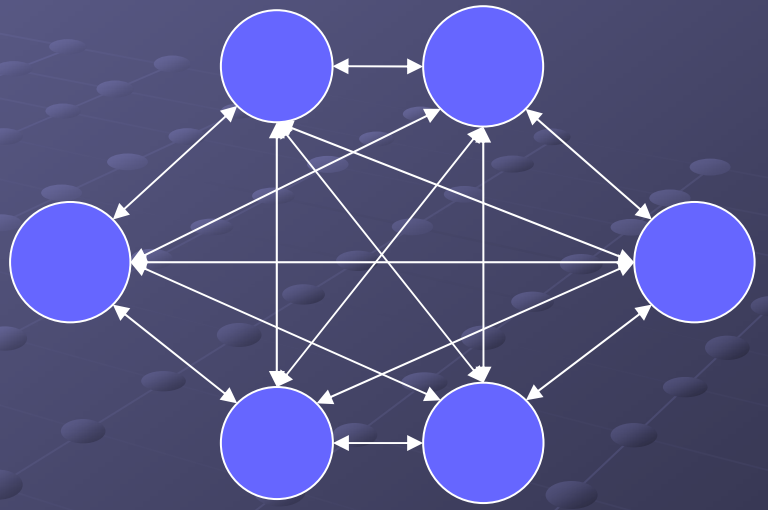


réseau à connections locales

***propagation des activations : de l 'entrée vers la sortie***

# Structure d'Interconnexion
## modèle récurrent (feedback network)





***propagation des activations :***

***<u>synchrone</u> : toutes les unités sont mises à jour simultanément***

***<u>asynchrone</u> : les unités sont mises à jours séquentiellement***

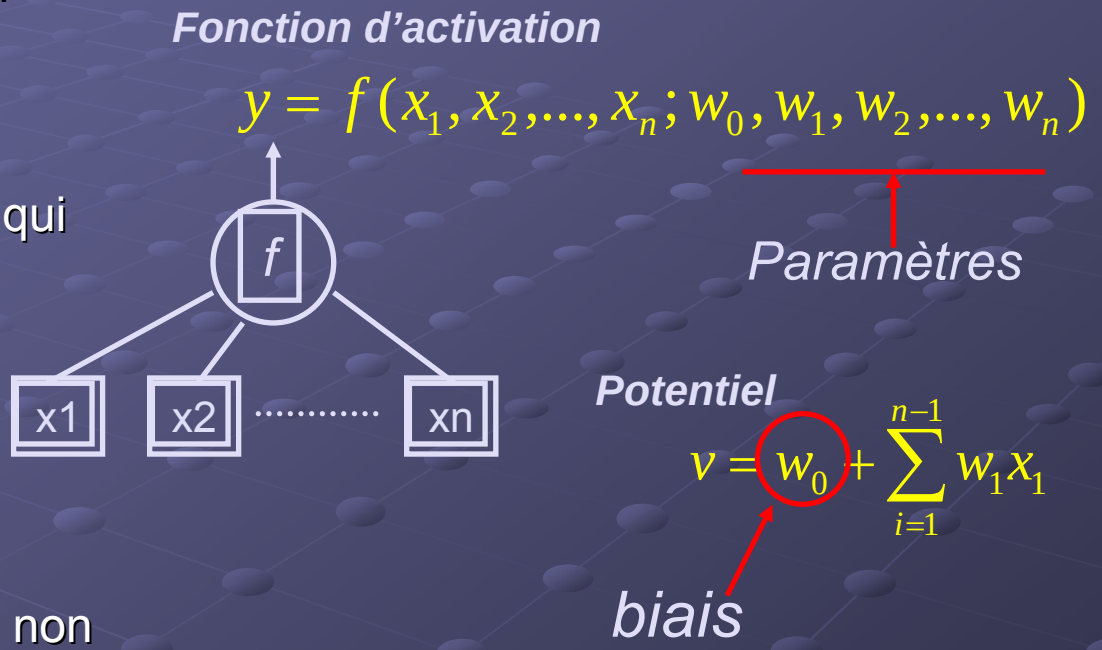
# Un neurone artificiel : une pâle imitation de la nature?

Il n'en demeure pas moins que les réseaux de neurones biologiques, de par leur multiples interconnexions (parfois récursives), de par ***leur mécanisme d'inhibition et d'activation, de par leur manière d'évoluer et de s'adapter tout on long de la vie d'un organisme vivant, ont inspiré les réseaux de neurones artificiels*** et continuent d'influencer le développement de nouveaux modèles plus adaptés à la résolution de certains problèmes.

Les caractéristiques dominantes et communes aux réseaux de neurones biologiques et artificiels sont leur capacité d'apprendre et de généraliser pour qu'une même unité, le neurone, puisse servir à plus d'une tâche à la fois.

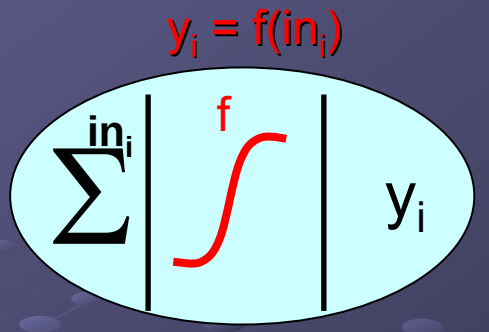
# Les RNA

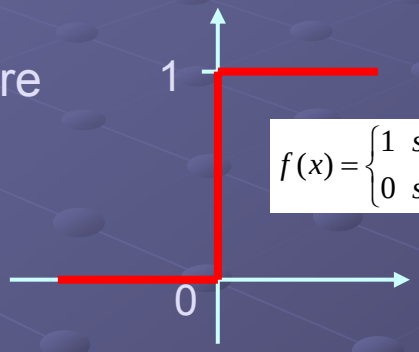
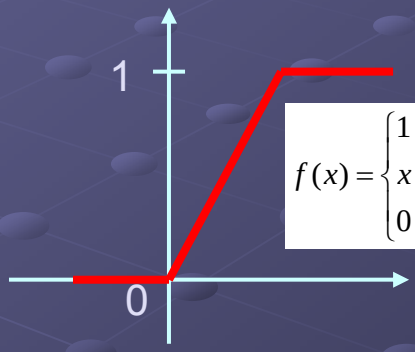
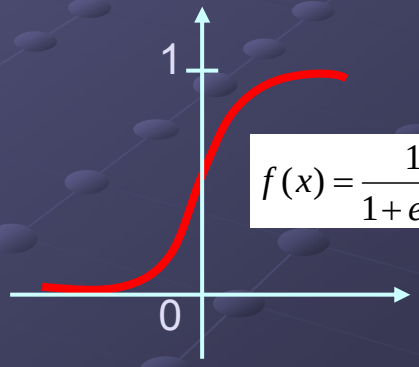
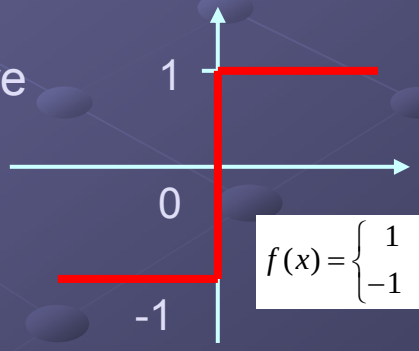
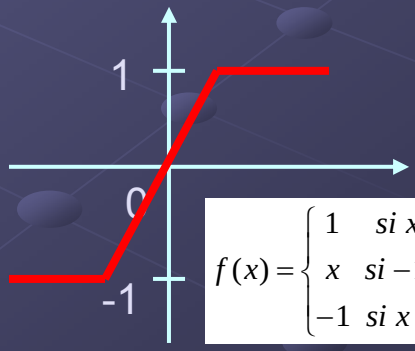
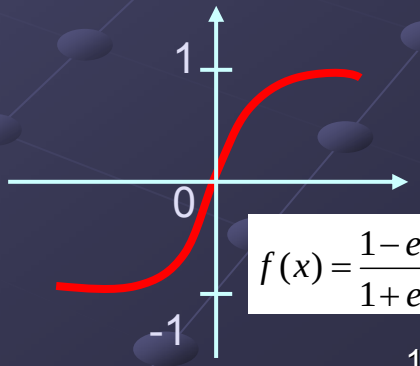
- Modélisation non-linéaire par apprentissage numérique
- Apprentissage = procédures cognitives de généralisation qui permettent de faire des prédictions
- Approximation non-linéaire parcimonieuse
- Un neurone est une fonction non linéaire, paramétrée, à valeurs bornées.



# Quelques fonctions d'activation…

***L'utilisation d'une fonction d'activation non-linéaire permet au RNA de modéliser des équations dont la sortie n'est pas une combinaison linéaire des entrées.***
Cette caractéristique confère au RNA de grandes capacités de modélisation fortement appréciées pour la résolution de problèmes non-linéaires. Voici quelques exemples de fonctions d'activation :




| | Échelon | Rampe | Sigmoïde |
|---|---|---|---|
| Unipolaire |   $f(x)=\begin{cases}1 & si\ x\geq 0\\ 0 & si\ x<0\end{cases}$ |   $f(x)=\begin{cases}1 & si\ x>1\\ x & si\ 0\leq x\leq 1\\ 0 & si\ x<0\end{cases}$ |   $f(x)=\frac{1}{1+e^{-\alpha x}}$ |
| Bipolaire |   $f(x)=\begin{cases}1 & si\ x\geq 0\\ -1 & si\ x<0\end{cases}$ |   $f(x)=\begin{cases}1 & si\ x>1\\ x & si\ -1\leq x\leq 1\\ -1 & si\ x<-1\end{cases}$ |   $f(x)=\frac{1-e^{-\alpha x}}{1+e^{-\alpha x}}$ |

# Les réseaux de neurones artificiels : une solution d'avenir!

*« Quelle machine pourrait effectuer ce travail à ma place? »*

*« Quelle machine pourrait penser à une solution à ma place? »*

Les réseaux de neurones artificiels (RNA) s'inscrivent dans l'effort de conception est mis sur le développement ***d'algorithmes d'apprentissage*** afin de doter un système d'autonomie et de capacités d'adaptation. Parfois, ces systèmes intelligents arrivent même à « découvrir » de nouvelles solutions à des problèmes fort complexes et difficilement accessibles pour un cerveau … humain.

*« Pourquoi les RNA ont-ils de l'avenir? »*

Parce que les RNA répondent à plusieurs critères de l'entreprise moderne, dont le fait de trouver des solutions rapidement à des problèmes de plus en plus complexes. De plus, ***les RNA sont robustes, versatiles et peuvent s'adapter.***

# Algorithmes d'apprentissage

Comme le cerveau humain, les réseaux de neurones artificiels (RNA) peuvent apprendre par expérience.

Les algorithmes d'apprentissage modifient la valeur des poids entre les neurones ainsi que la valeur des biais de façon à améliorer la performance du RNA.

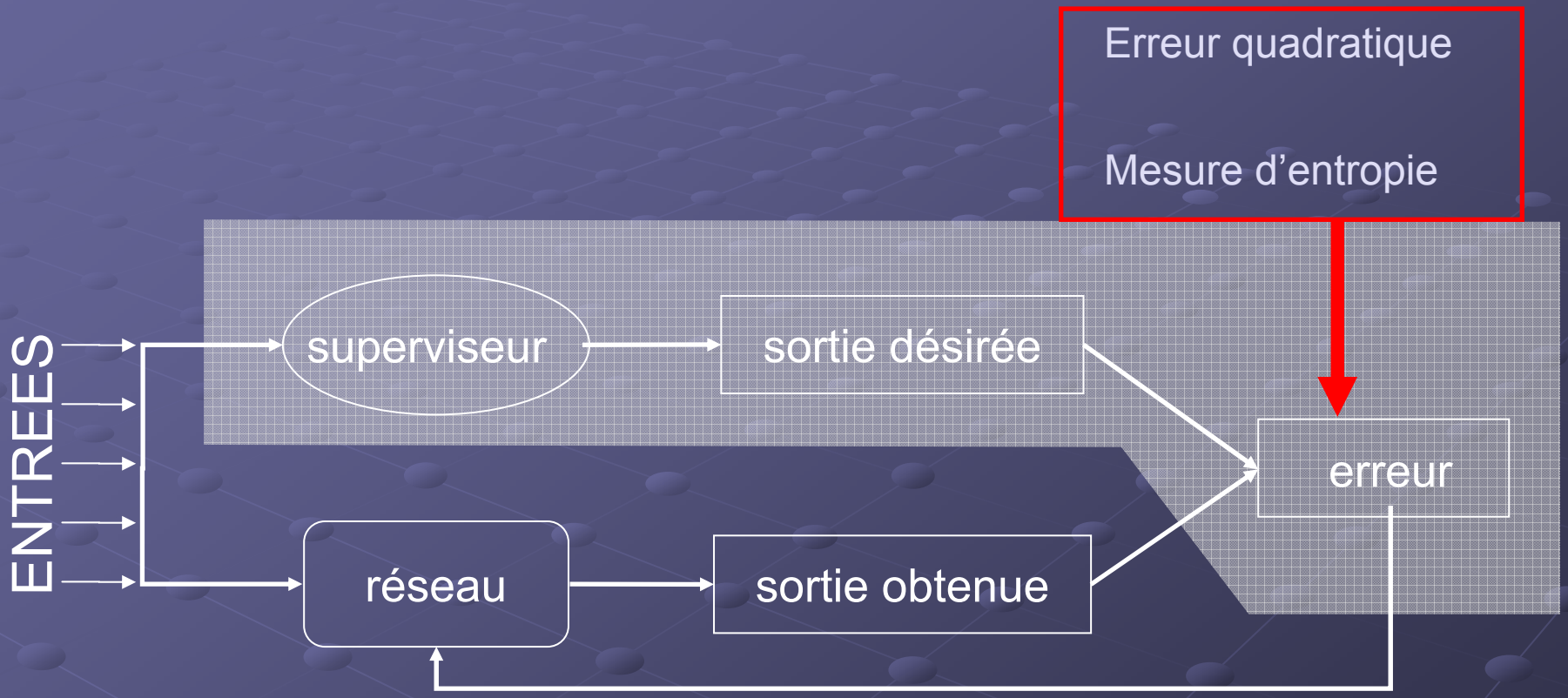
Trois grandes classes d'apprentissage existent :

Apprentissage non-supervisé : il n'y pas de connaissances à priori des sorties désirés pour des entrées données. En fait, c'est de l'apprentissage par exploration où l'algorithme d'apprentissage ajuste les poids des liens entre neurones de façon à maximiser la qualité de classification des entrées.

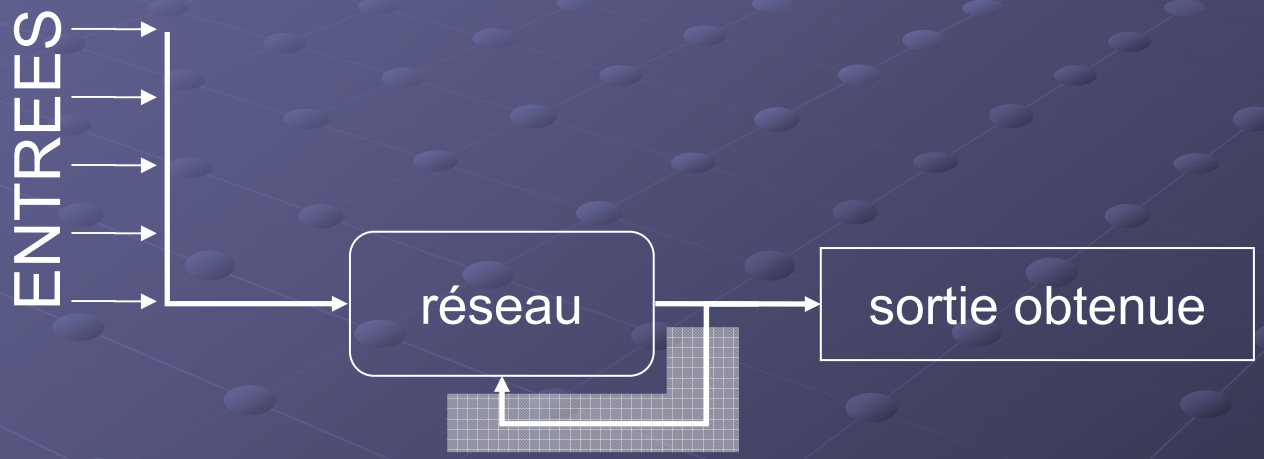
Apprentissage par renforcement : Dans ce cas, bien que les sorties idéales ne soient pas connues directement, il y a un moyen quelconque de connaître si les sorties du RNA s'approchent ou s'éloignent du but visé. Ainsi, les poids sont ajustés de façons plus ou moins aléatoire et la modification est conservée si l'impact est positif ou rejetée sinon.

Apprentissage supervisé (back propagation) : Cet algorithme d'apprentissage ne peut être utilisé que lorsque les combinaisons d'entrées-sorties désirés sont connues. L'apprentissage est alors facilité et par là, beaucoup plus rapide que pour les deux autres algorithmes puisque l'ajustement des poids est fait directement à partir de l'erreur, soit la différence entre la sortie obtenue par le RNA et la sortie désirée.

# Apprentissage supervisé



# Apprentissage non supervisé



# Algorithmes d'apprentissage

Pré-apprentissage : Une fois l'apprentissage terminée, la structure et les poids entre les neurones sont fixés. Lorsque la tâche à effectuer par le RNA ne change pas significativement avec le temps, un RNA statique peut être très efficace et beaucoup plus simple d'implémentation. Par ailleurs, il n'est alors pas aussi important de se soucier de la simplicité de la fonction d'apprentissage et de la rapidité de convergence du RNA.

Apprentissage continu : Un apprentissage continu est requis lorsque les combinaisons d'entrées-sorties idéales risquent de changer avec le temps ou selon diverses conditions externes. Le gain de flexibilité inhérent à la capacité d'adaptation du RNA s'obtient au détriment d'une implémentation plus complexe du système puisque l'algorithme d'apprentissage doit alors être intégré à la structure de calcul du RNA.

1. pour un système en temps réel, la rapidité d'adaptation ou de convergence de la fonction d'apprentissage devient alors un facteur très important.
2. la complexité d'implémentation de la fonction d'apprentissage devient également un élément à considérer dans le choix de l'algorithme d'apprentissage, parfois au détriment de la performance.

# Apprentissage d'un réseau de neurones

**L 'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré.**

- L'ajustement du poids des liens entre les neurones peut s'effectuer selon diverses équations mathématiques, dont la plus populaire est sans aucun doute la loi de Hebb.
- le choix de l'équation d'adaptation des poids dépend en grande partie de la topologie du réseau de neurones utilisé.
- Notons qu'il est aussi possible de faire évoluer l'architecture du réseau, soit le nombre de neurones et les interconnexions, mais avec d'autres types d'algorithmes d'apprentissage.

# Historique

James [1890] :

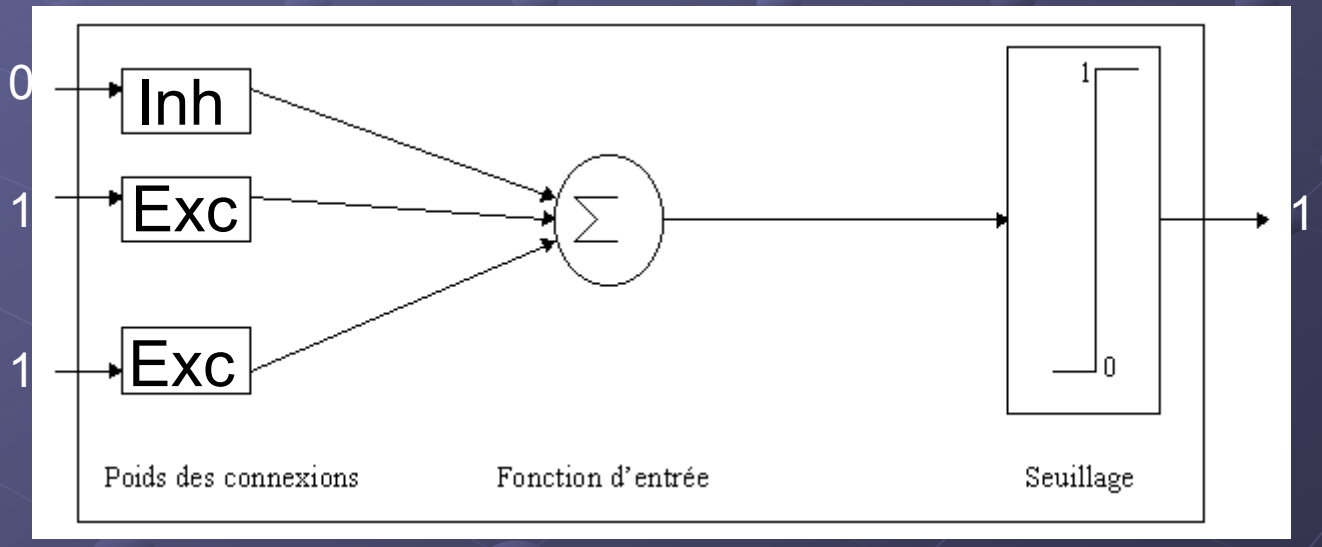
mémoire associative et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.

*When two elementary brain-processes have been active together or in immediate succession, one of them, on reoccurring, tends to propagate its excitement into the other*

# Historique

## McCulloch & Pitts [1943]

*Ceux sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (tout au moins au niveau théorique).*



# Historique

## Hebb [1949]

- *explique le conditionnement chez l'animal par les propriétés des neurones eux-mêmes.*
- *un conditionnement de type pavlovien tel que, nourrir tous les jours à la même heure un chien, entraîne chez cet animal la sécrétion de salive à cette heure précise même en l'absence de nourriture.*

***Loi d'apprentissage***

le conditionnement est une propriété des neurones:

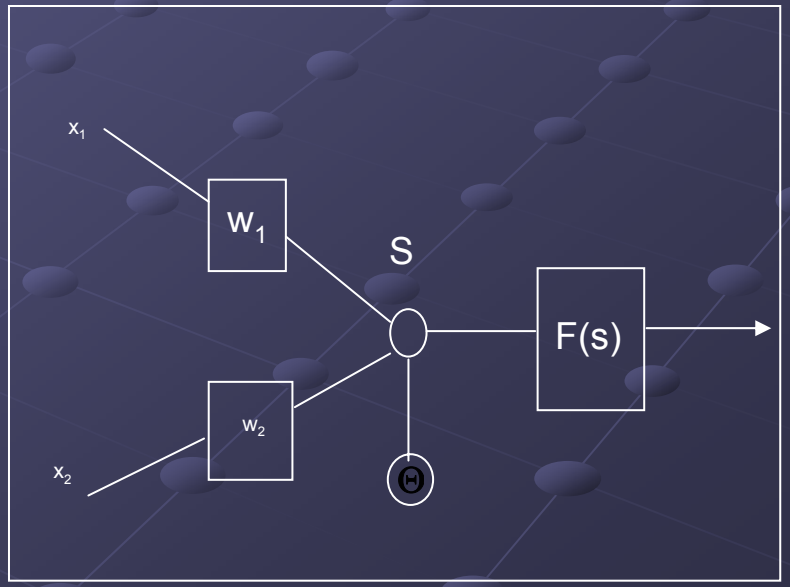
# Historique

Rosenblatt [1957] :

le perceptron, premier modèle opérationnel reconnaissance d 'une configuration apprise tolérance aux bruits

- Le perceptron se compose de deux couches :
  - la rétine
  - couche de sortie qui donne la réponse correspondant à la stimulation présente en entrée.
- Les cellules de la première couche répondent en oui/non. La réponse 'oui' correspond à une valeur '1' et la réponse 'non' correspond à une valeur '0' à la sortie du neurone.
- Les cellules d'entrée sont reliées aux cellules de sortie grâce à des synapses d'intensité variable.
- L'apprentissage du perceptron s'effectue en modifiant l'intensité de ces synapses.

$$\mathbf{sgn}(s) = \begin{cases} +1 & \text{ssi } s > 0 \\ -1 & \text{ssi } s <= 0 \end{cases} = F(s)$$



# Historique

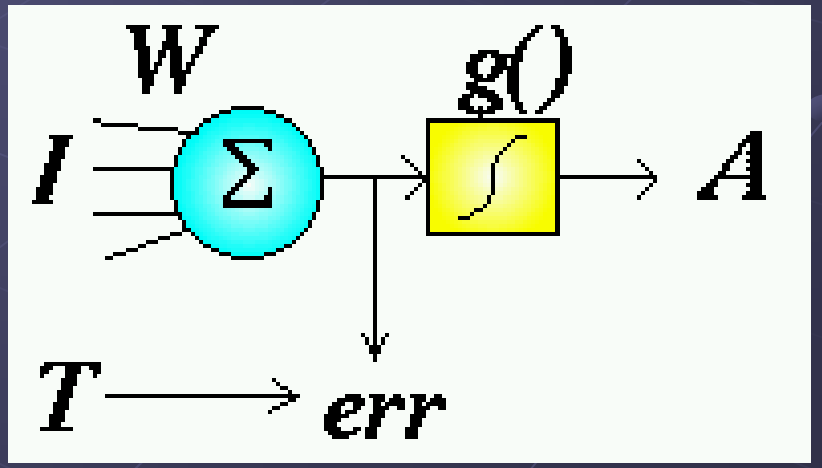
Widrow [1960] :

adaline, *adaptive linear element*

*Celle-ci est à l'origine de **l'algorithme de rétropropagation de gradient** très utilisé aujourd'hui avec les Perceptrons multicouches.*

*B. Widrow a créé dès cette époque une des premières firmes proposant neuro-ordinateurs et neuro-composants, la "Memistor Corporation".*

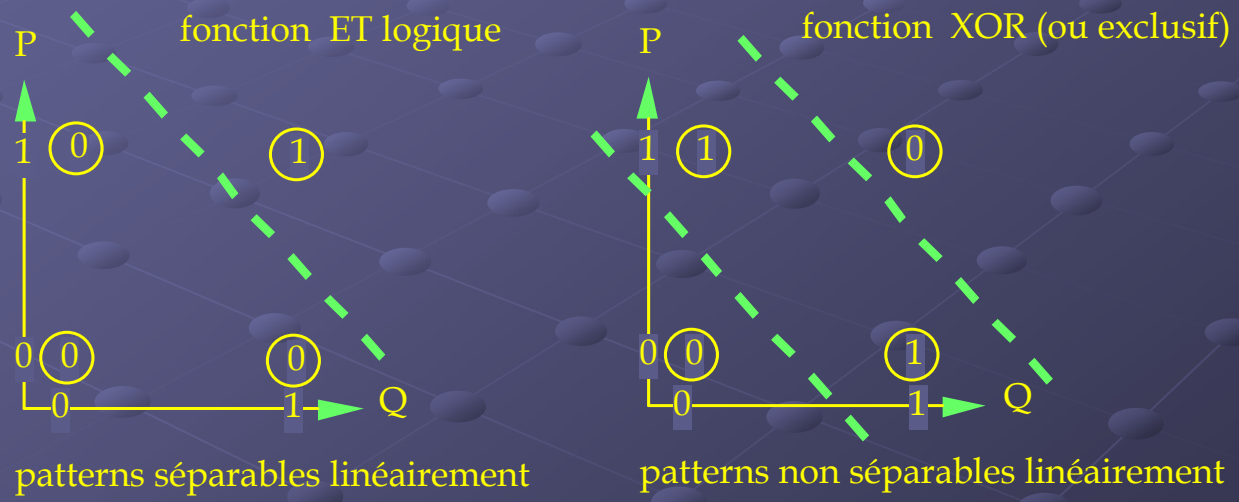
Après chaque motif d'apprentissage présenté, la correction s'applique au poids proportionnellement à l'erreur. La correction est calculée *avant* le seuillage :



# Historique

## Minsky & Papert [1969] :

impossibilité de classer des configurations non linéairement séparables abandon (financier) des recherches sur les RNA



# Historique
## *l'ombre et le renouveau*

[Hopfield [1982] :

modèle des verres de spins. Hopfield fixe préalablement le comportement à atteindre pour son modèle et construit à partir de là, la structure et la loi d'apprentissage correspondant au résultat escompté. Ce modèle est aujourd'hui encore ***très utilisé pour des problèmes d'optimisation***.

Boltzmann [1983] :

première réponse à Minsky et Papert

[1985] :

la rétro-propagation du gradient et le perceptron multicouche. Dès cette découverte, nous avons la possibilité de réaliser une fonction non linéaire d'entrée/sortie sur un réseau en décomposant cette fonction en une suite d'étapes linéairements séparable De nos jours, les réseaux multicouches et la rétropropagation de gradient reste le modèle le plus étudié et le plus productif au niveau des applications.

Rumelhart, McClelland, … [1985] :

le groupe *Parallel Distributed Processing*

# Algorithmes d'apprentissage


Loi de Hebb : ***Cette règle très simple émet l'hypothèse que lorsqu'un neurone « A » est excité par un neurone « B » de façon répétitive ou persistante, l'efficacité (ou le poids) de l'axone reliant ces deux neurones devrait alors être augmentée.***

- Loi de Hebb : $\Delta w_{ji} = R\, a_i\, a_j$

Loi de Hopfield : ***Cette loi se base sur la même hypothèse que la loi de Hebb mais ajoute une variable supplémentaire pour contrôler le taux de variation du poids entre les neurones avec une constante d'apprentissage qui assure à la fois la vitesse de convergence et la stabilité du RNA.***

Loi Delta : ***Cette loi est aussi une version modifiée de la loi de Hebb. Les poids des liens entre les neurones sont continuellement modifiés de façon à réduire la différence (le delta) entre la sortie désirée et la valeur calculée de la sortie du neurone. Les poids sont modifiés de façon à minimiser l'erreur quadratique à la sortie du RNA. L'erreur est alors propagée des neurones de sortie vers les neurones des couches inférieures, une couche à la fois.***

- Règle de Widrow-Hoff (delta rule) : $\Delta w_{ji} = R(d_i - a_i)a_j$

# Algorithmes d'apprentissage

***Loi de Hebb :***

Si deux unités connectées sont actives simultanément, le poids de leur connexion est augmenté ou diminué. R est une constante positive qui représente ***la force d'apprentissage*** (learning rate).

| | ai = -1 | ai = 1 |
|---|---|---|
| aj = -1 | ΔWji = R | ΔWji = -R |
| aj = 1 | ΔWji = -R | ΔWji = R |


$$\Delta W_{ji} = R a_i a_j$$

# Algorithmes d'apprentissage

## ***Loi de Widrow-Hoff (delta rule) :***

$a_i$ activation produite par le réseau

$d_i$ réponse désirée par l'expert humain

Par exemple si la sortie est inférieure à la réponse désirée, il va falloir augmenter le poids de la connexion à condition bien sûr que l'unité j soit excitatrice (égale à 1). On est dans l'hypothèse d'unités booléennes {0,1}.

| | ai = 0 | ai = 1 |
|---|---|---|
| di = 0 | ΔWji = 0 | ΔWji = -R |
| di = 1 | ΔWji = R | ΔWji = 0 |


$$\Delta W_{ji} = R(d_i - a_i)a_j$$

# Algorithmes d'apprentissage

***Loi de Grossberg :***

On augmente les poids qui entrent sur l'unité gagnante $a_i$ s'ils sont trop faibles, pour les rapprocher du vecteur d'entrée $a_j$. C'est la règle d'apprentissage utilisée dans les cartes auto-organisatrices de Kohonen (SOM)

$$\Delta W_{ji} = R a_i \left( a_j - W_{ji} \right)$$

# Réseaux Statiques Feedforward

# Apprentissage d'un modèle statique

- Méthodes non adaptatives d'apprentissage :
  - minimisation de la fonction de coût des moindres carrés qui tient compte simultanément de tous les exemples
- Méthodes adaptatives
  - Modifier les paramètres du modèles successivement en fonction du coût partielle

# Similitude comme Distance

$$d : R^n \times R^n \rightarrow [0,\infty)$$

$$d(x,y) = 0 \quad si\ x = y$$
$$d(x,y) = d(y,x)$$
$$d(x,y) \leq d(x,z) + d(z,y) \quad \forall z$$

## Distance de Minkowski

$$d_M = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

***p=2 on obtient la distance euclidienne***
***p=1 on obtient la distance Manhattan qui pour***
***des données binaires devient la distance Hamming ou Tanimoto***

# Fonctions discriminantes linéaires

- La première approche dans la classification de formes est géométrique. La manière la plus simple de séparer deux objets dans un plan XoY est d'intercaler entre ces objets une droite de séparation.
- La droite de séparation est écrite sous la forme

$$w_1 x_1 + w_2 x_2 + \Theta = 0$$

- Cette relation fait apparaître des termes de poids $w_1$ et $w_2$ de même qu'un terme de polarisation $\Theta$. Les poids jouent le rôles de balancier permettant d'ajuster la pente de la droite tandis que le terme de polarisation permet un déplacement vers les y positifs ou négatif de la droite séparatrice.

# Fonctions discriminantes linéaires

- **On peut écrire également l'équation sous une forme plus connue, soit:**

**Si** $x_1 = x$ **et** $x_2 = y$

$w_2 x_2 = -\Theta - w_1 x_1$ **implique que** $w_2 y = -\Theta - w_1 x$

**et** $y = \dfrac{-w_1 x - \Theta}{w_2} = -\dfrac{w_1 x}{w_2} - \dfrac{\Theta}{w_2}$

# Règle d'apprentissage du perceptron pour les fonction linéaires discriminantes

- Le perceptron doit trouver l'ensemble des valeurs à donner aux synapses pour que les configurations d'entrée se traduisent par des réponses voulues. Pour cela, on utilise la *règle d'apprentissage* de **Windrow-Hoff**.
- Pour apprendre, le perceptron doit savoir qu'il a commis une erreur, et doit connaître la réponse qu'il aurait dû donner. De ce fait, on parle *d'apprentissage supervisé*.
- La règle d'apprentissage est locale dans ce sens que chaque cellule de sortie apprend sans avoir besoin de connaître la réponse des autres cellules.
- La cellule ne modifie l'intensité de ses synapses (apprend) que lorsqu'elle se trompe.

# Algorithme du perceptron

a) Initialisation des poids avec des valeurs aléatoires.
b) Présentation à l'entrée d'un vecteur de données.
c) Calcul de la fonction de sortie et vérification de l'erreur entre la sortie calculée et la sortie prévue.
d) Ajustement des poids selon l'erreur obtenue.
e) retourner à l'étape b).

# Algorithme du Perceptron pour un réseau simple couche.

**Initialisation des poids de pondération**

(1) **Initialiser $w_1, w_2$ et $\Theta$ par des valeurs aléatoires**

**Paramètres caractérisant l'objet à l'entrée du réseau**

$$(2) \quad \vec{x} = \begin{cases} x_1 = x \\ x_2 = y \end{cases}$$

**Calcul de la règle de propagation pour être appliquée à la fonction d'activation**

$$(3) \quad s = \sum_j w_j x_j + \Theta$$

**Calcul de la fonction d'activation (fonction sgn)**

$$(4) \quad F(s) = y_0 = \begin{cases} 1 & \text{ssi} \quad s > 0 \\ -1 & \quad\quad s <= 0 \end{cases}$$

**Vecteur de classement du paramètre apparaissant à l'entrée**

(5) **Calcul de $d(\vec{x})$**

**la sortie attendue - la sortie calculée par le perceptron courant**

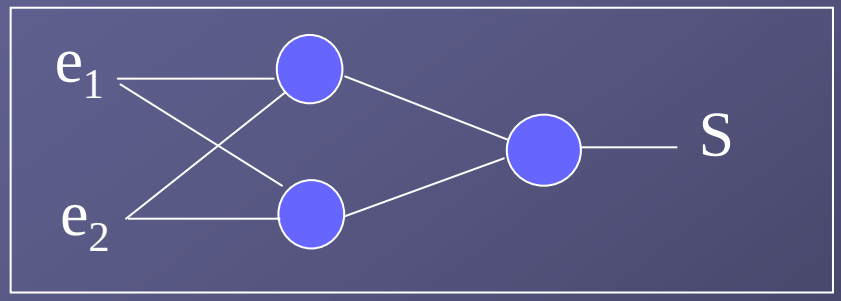
**Règle d'apprentissage pour la mise à jour des poids**

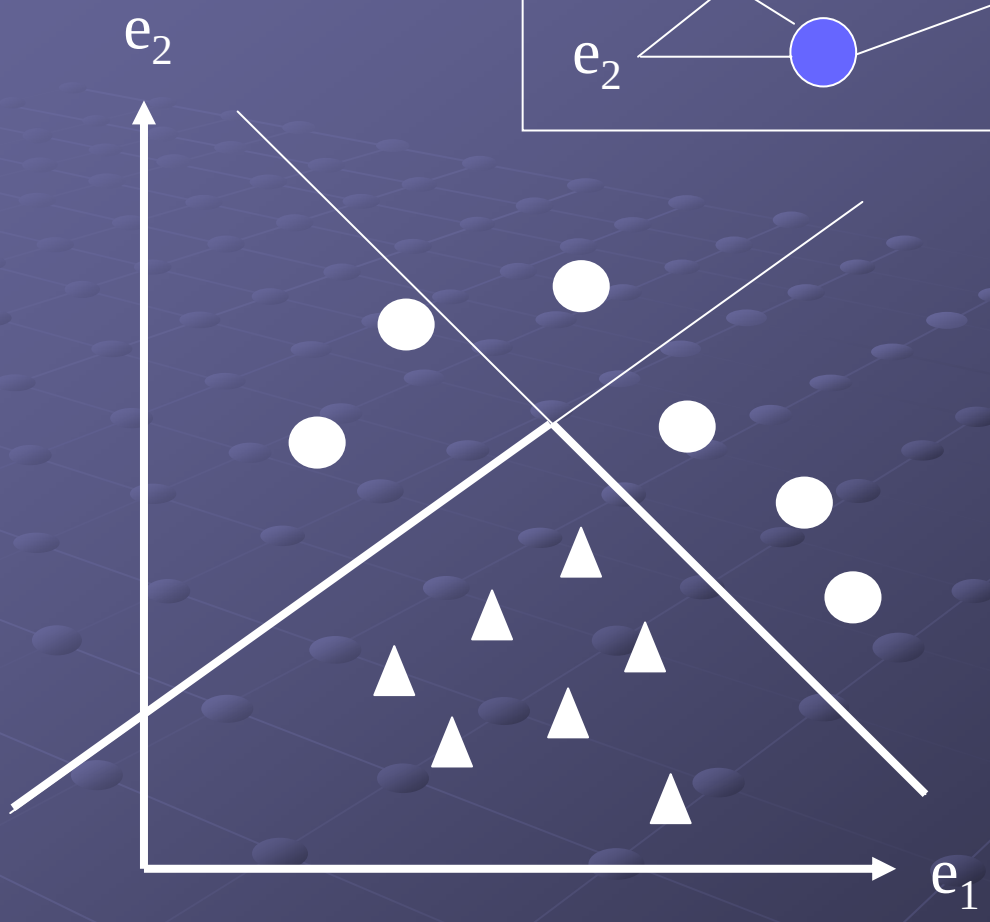
$$(6)\quad \Delta w_i = d(\vec{x})x_i$$
$$\Delta\Theta = d(\vec{x})$$

**Mise à jour des poids**

$$(7)\quad w_i = w_i + \Delta w_i$$
$$\Theta = \Theta + \Delta\Theta$$

Dans l'apprentissage du Perceptron, les poids sont ajustés **seulement si un motifs est mal classé**.



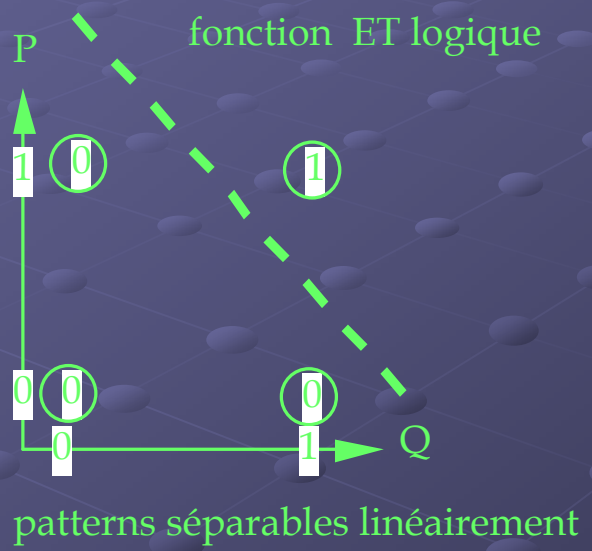



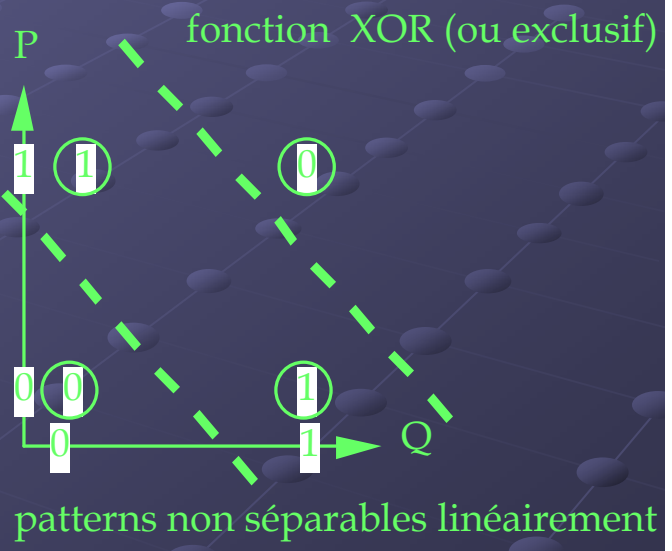

**Réseaux directs à couches**

# Limite du perceptron

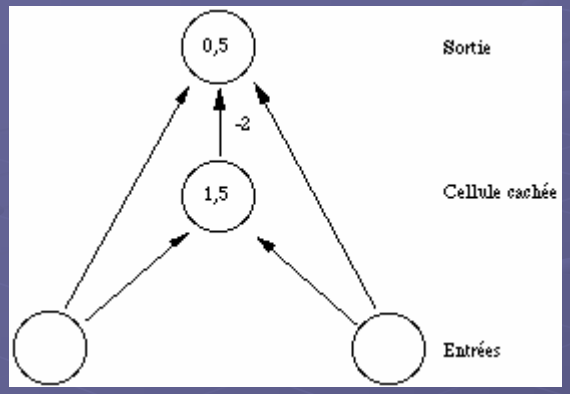
Le perceptron est incapable de distinguer les patterns non séparables linéairement [Minsky 69]

| | | | | |
|---|---|---|---|---|
| input P | 0 | 1 | 0 | 1 |
| input Q | 1 | 0 | 0 | 1 |
| **ET** | **0** | **0** | **0** | **1** |
| **XOR** | **1** | **1** | **0** | **0** |



# Le perceptron multicouche: architecture




S Y S T E M E

Environnement

bias

1

bias

1

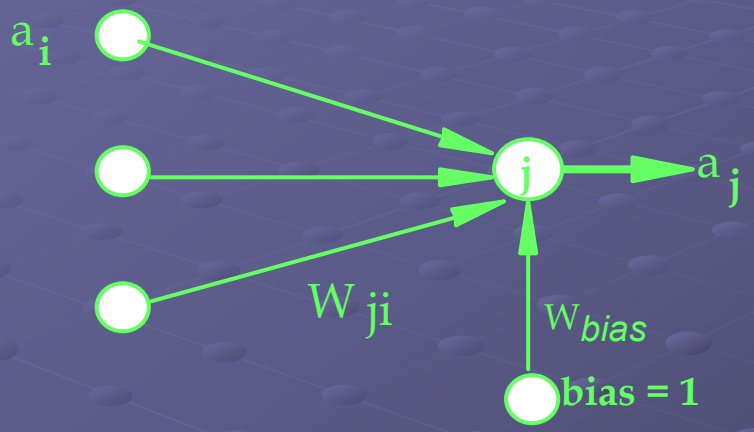
couche d'entrée (input) **i**

couche cachée (hidden) **j**

couche de sortie (output) **k**

Environnement

# Le perceptron multicouche : activation




$$x_j = \sum w_{ji} a_i$$

$$a_j = f(x_j)$$

fonction sigmoïde

$$a = f(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = f(x).(1-f(x))$$

fonction tangente hyperbolique

$$a = f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$f'(x) = (1+f(x)).(1-f(x))$$

# Apprentissage Backpropagation

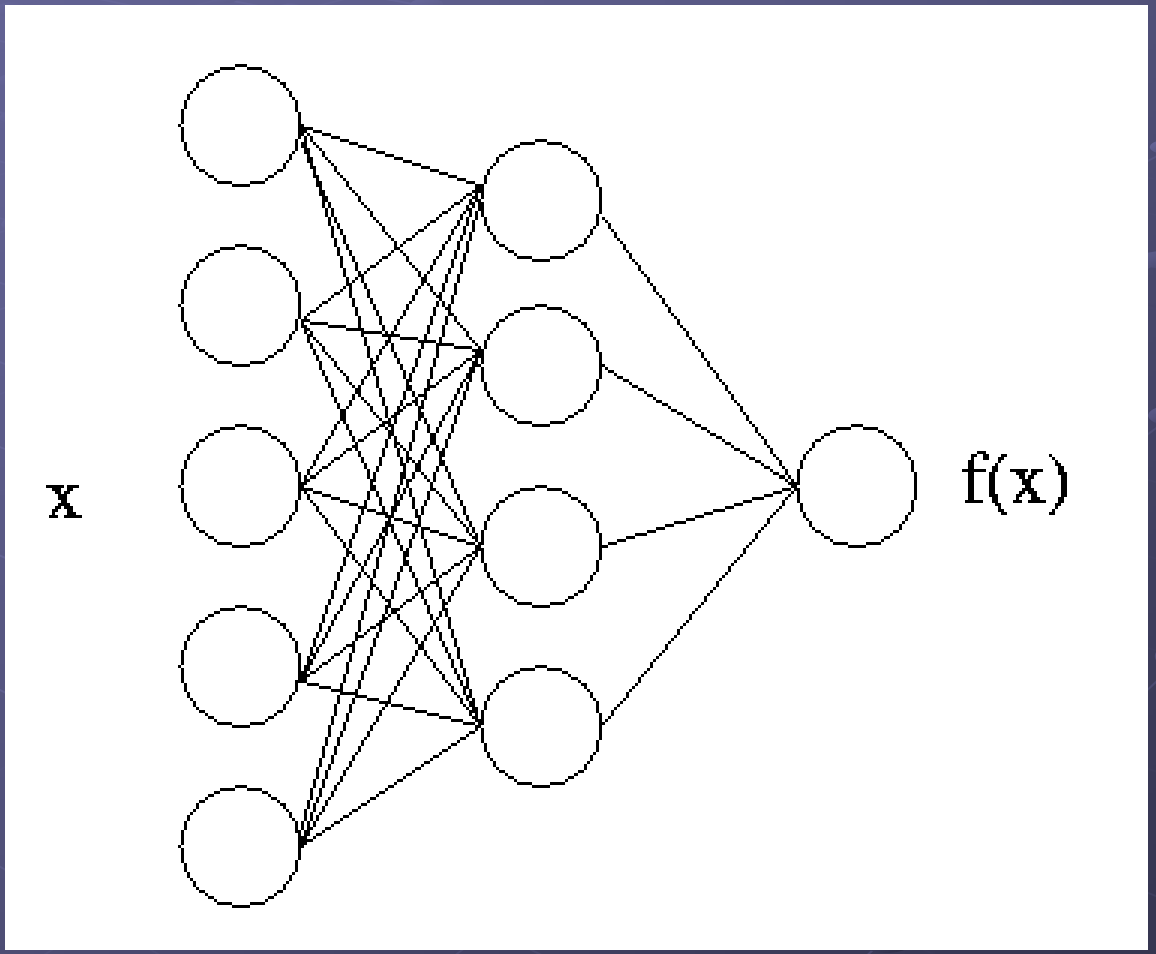
- ***Qu'est-ce qu'un gradient?***

  Le gradient pour un neurone est ***l'erreur relative à ce neurone***. Il peut en quelque sorte être vu comme la contribution du neurone à l'erreur globale

$$\frac{\partial E}{\partial w_{ij}}$$

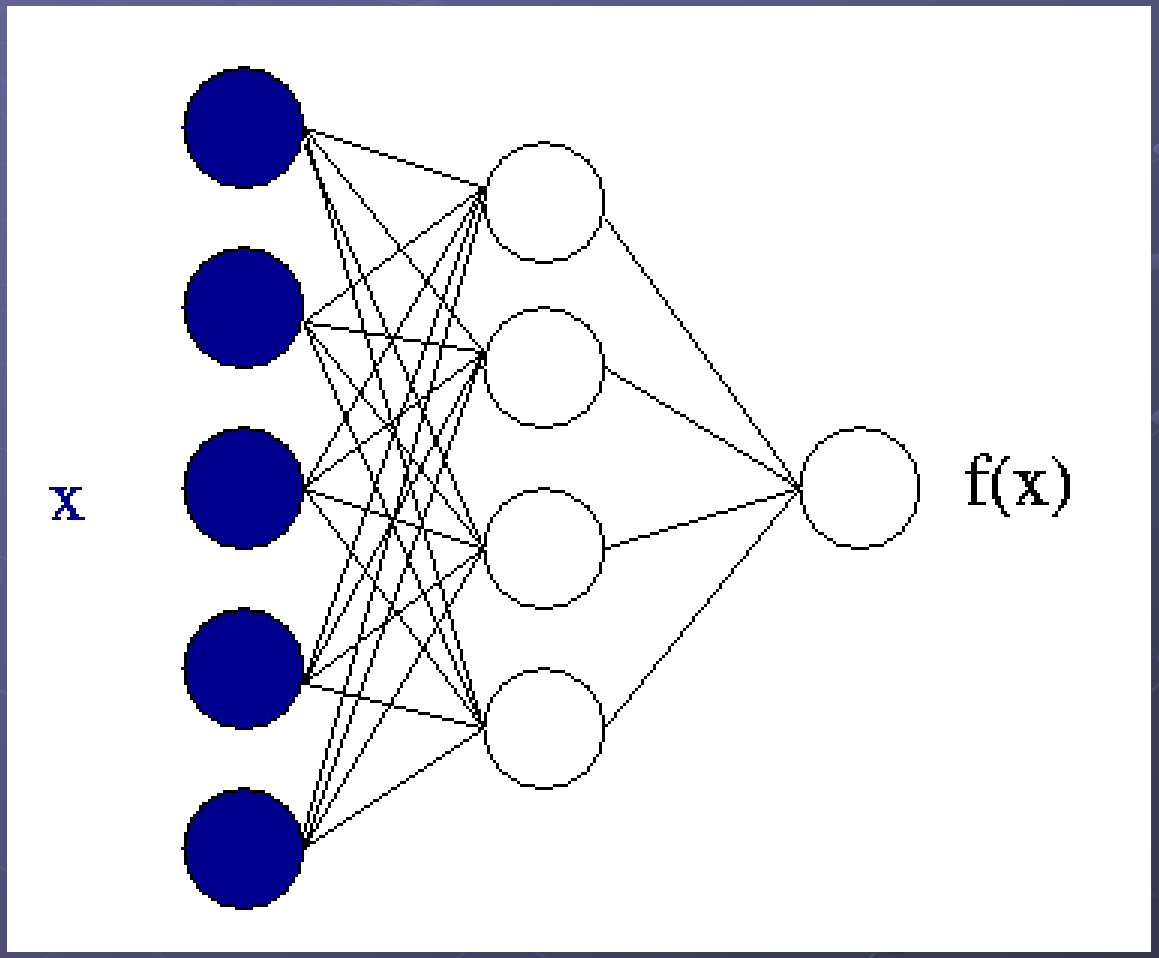
- A chaque passe arrière, on calcule le gradient de chaque neurone, en commençant par ceux de la couche de sortie (car les gradients des couches inférieures se calculent à partir des gradients des couches supérieures)

# Apprentissage Backpropagation



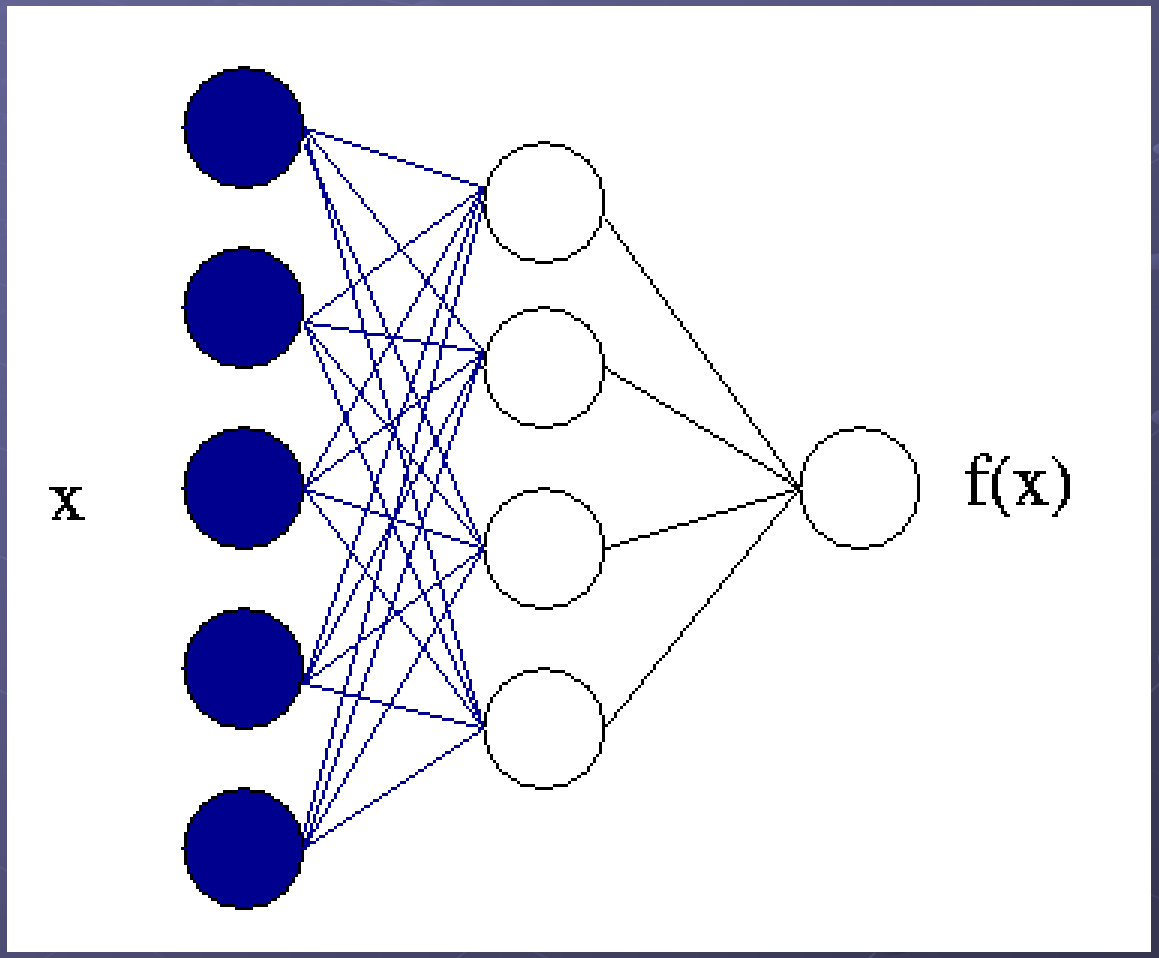

**Propagation de l'activité**

# Apprentissage Backpropagation



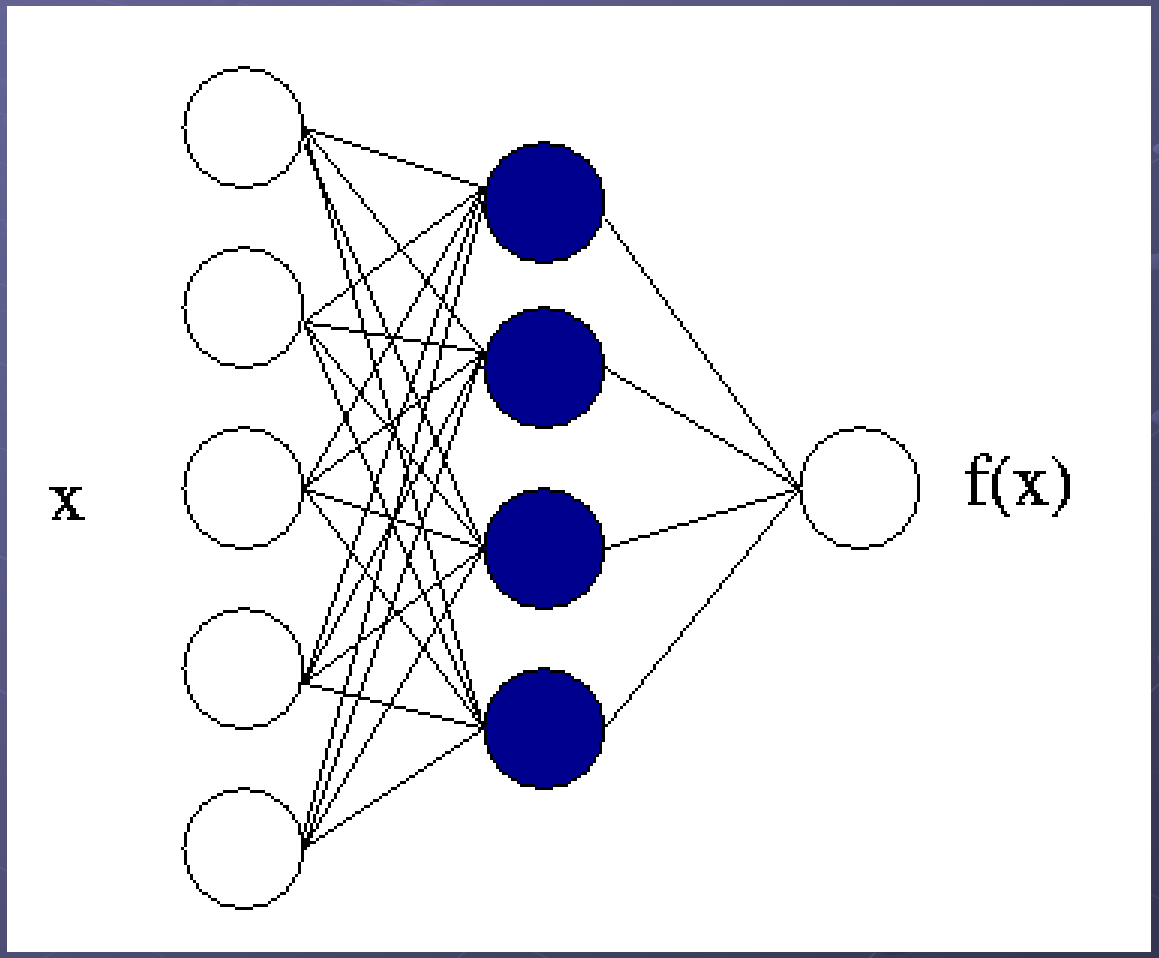

**Propagation de l'activité**

# Apprentissage Backpropagation



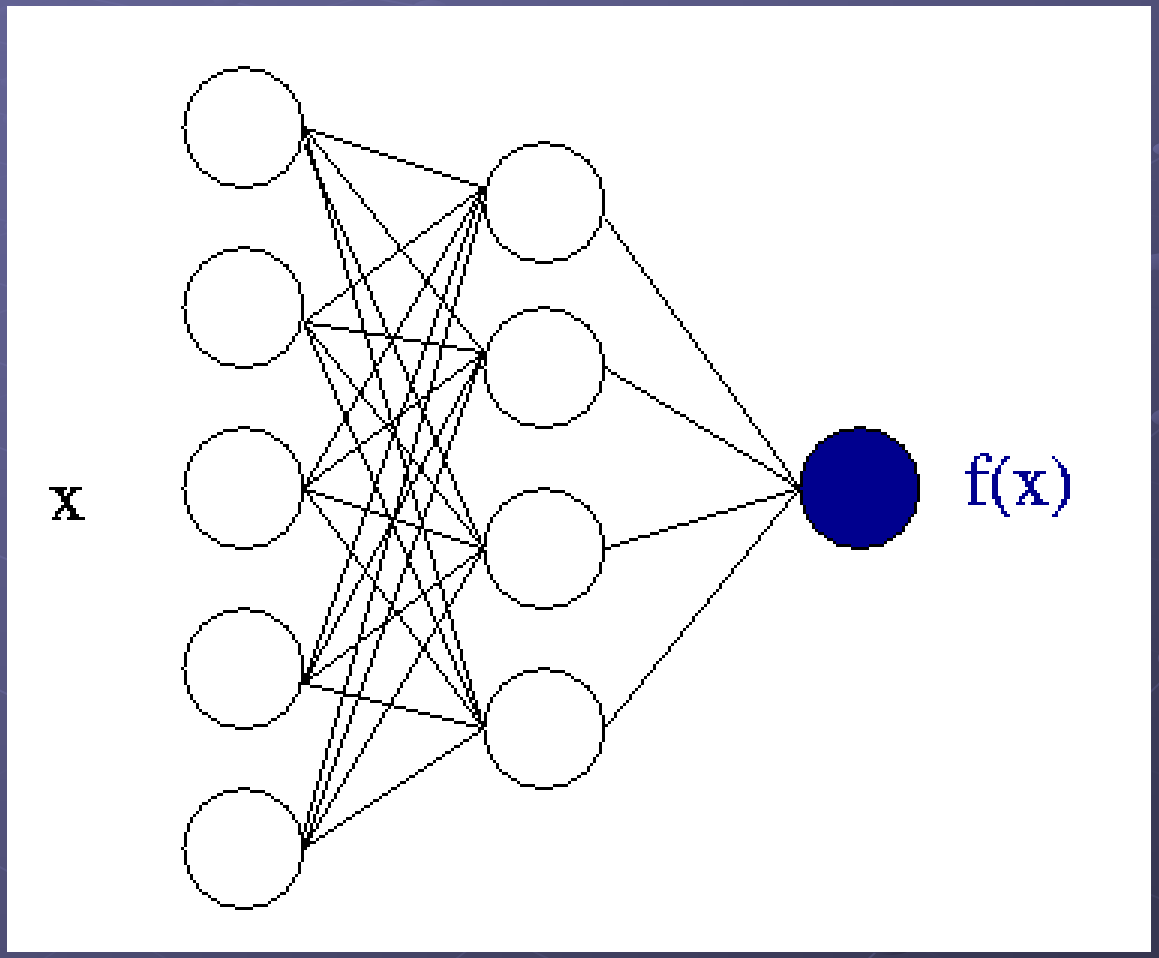

**Propagation de l'activité**

# Apprentissage Backpropagation




**Propagation de l'activité**

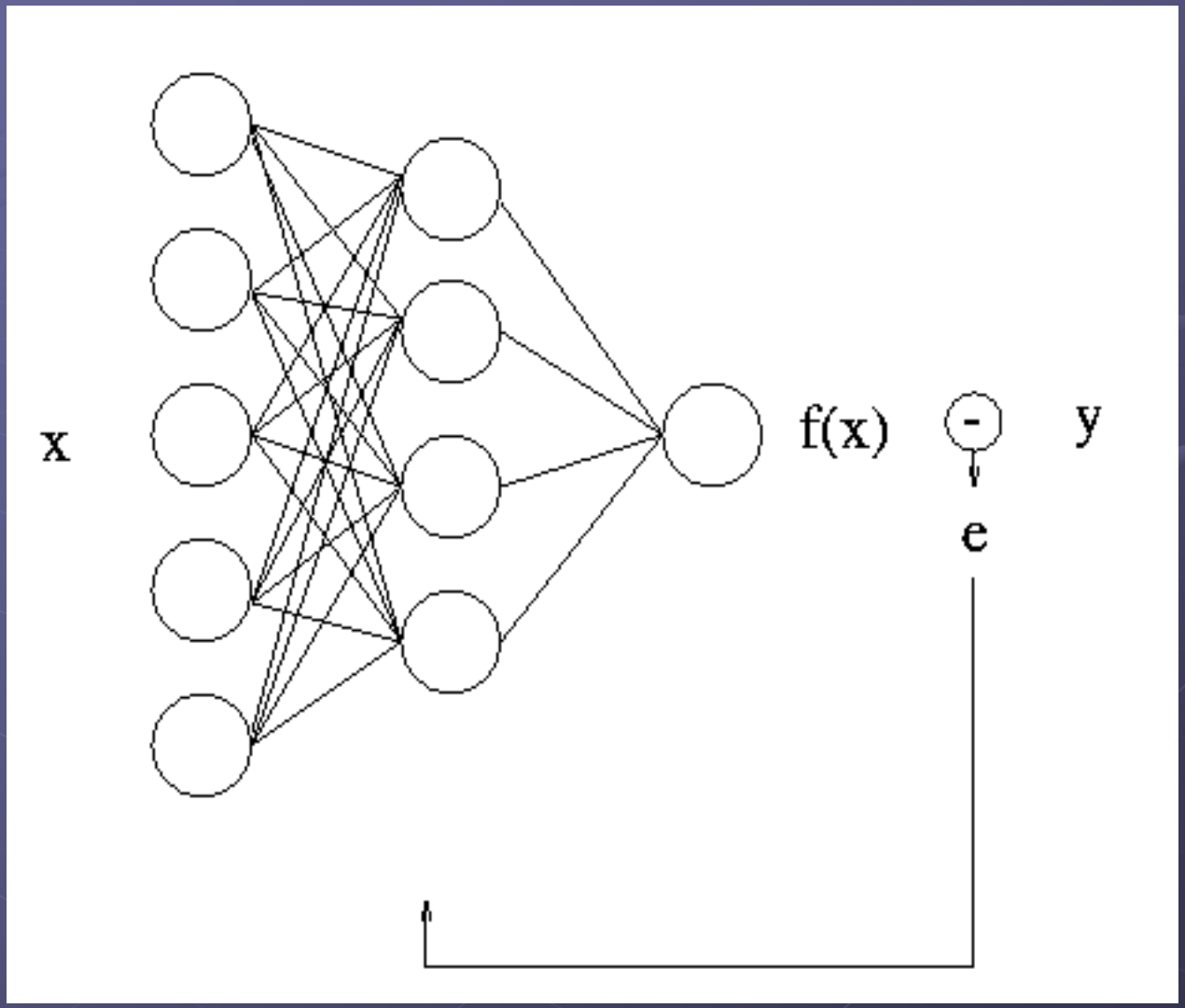
# Apprentissage Backpropagation




**Propagation de l'activité**

# Apprentissage Backpropagation

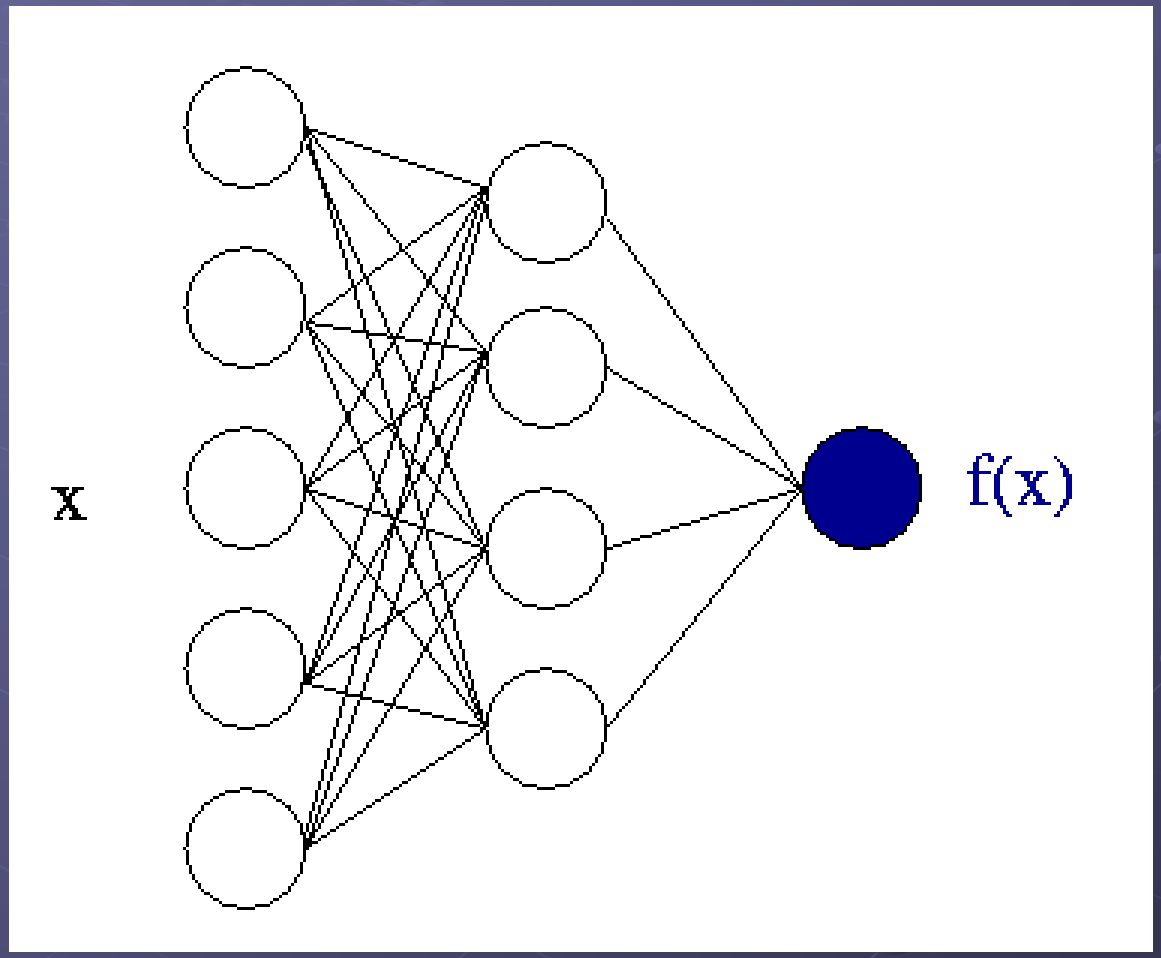
$$\frac{\partial E}{\partial w_{ij}}$$




**Rétropropagation de l'erreur**

# Apprentissage Backpropagation

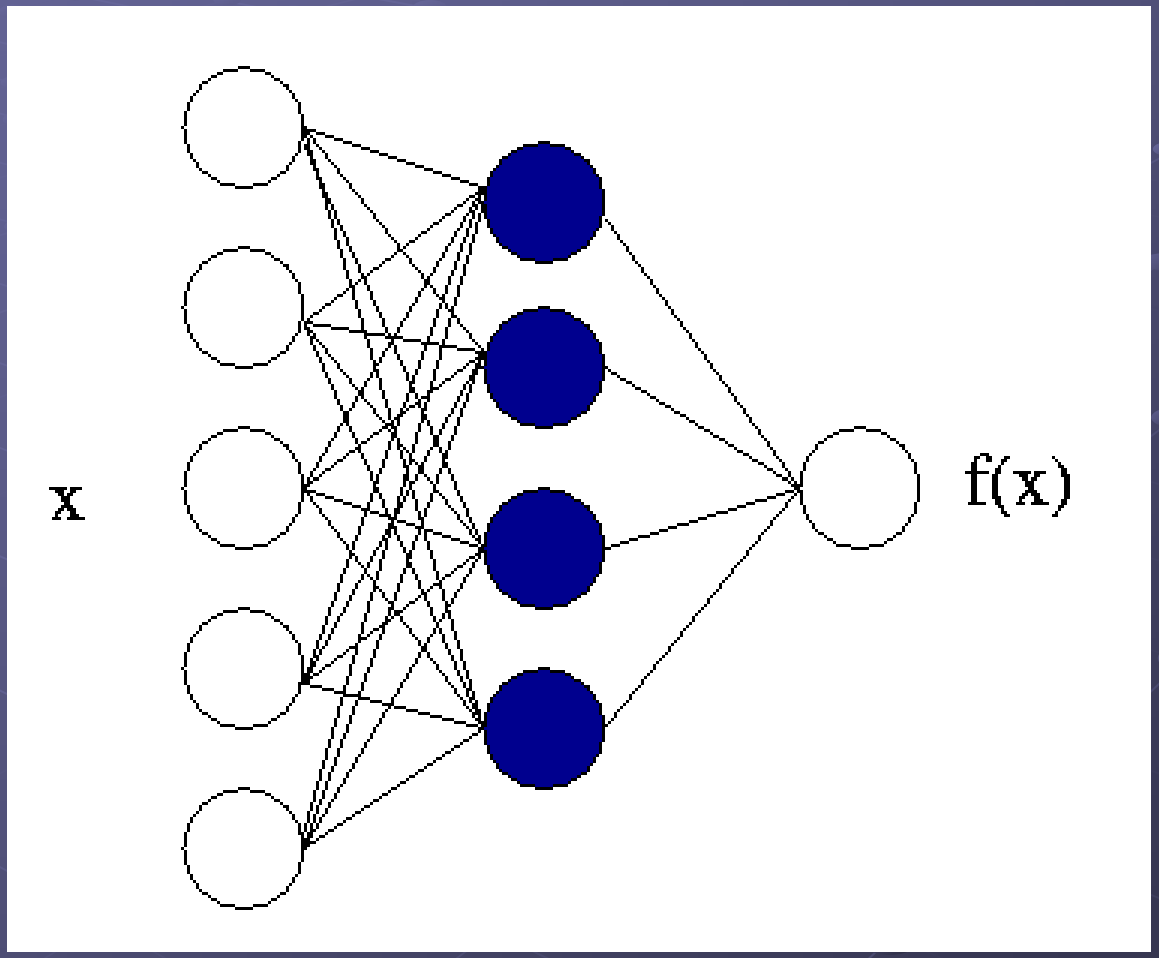
$$\frac{\partial E}{\partial w_{ij}}$$




**Rétropropagation de l'erreur**

# Apprentissage Backpropagation

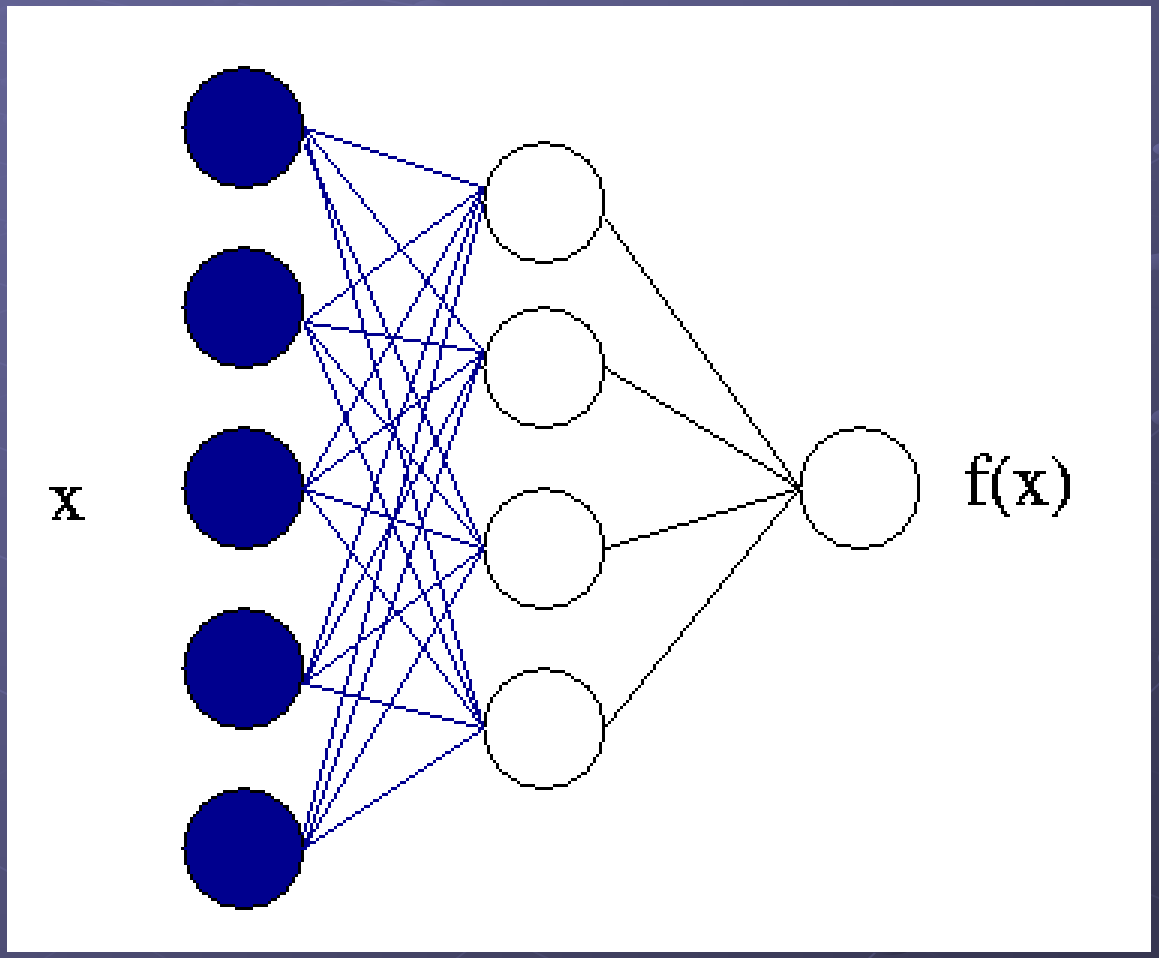
$$\frac{\partial E}{\partial w_{ij}}$$




**Rétropropagation de l'erreur**

# Apprentissage Backpropagation

$$\frac{\partial E}{\partial w_{ij}}$$




**Rétropropagation de l'erreur**

# Apprentissage : « Back propagation »

1er étape : Initialiser les poids des liens entre les neurones. Souvent *une valeur entre 0 et 1*, déterminée aléatoirement, est assignée à chacun des poids.

2e étape : Application d'un vecteur entrées-sorties à apprendre.

3e étape : Calcul des sorties du RNA à partir des entrées qui lui sont appliquées et calcul de l'erreur entre ces sorties et les sorties idéales à apprendre.

4e étape : Correction des poids des liens entre les neurones de la couche de sortie et de la première couche cachée selon l'erreur présente en sortie.

5e étape : Propagation de l'erreur sur la couche précédente et correction des poids des liens entre les neurones de la couche cachée et ceux en entrées.

6e étape : Boucler à la 2e étape avec un nouveau vecteur d'entrées-sorties tant que les performances du RNA (erreur sur les sorties) ne sont pas satisfaisantes.

# Le perceptron multicouche apprentissage : retropropagation de l'erreur

$$S_j = \sum_i a_i W_{ij}$$

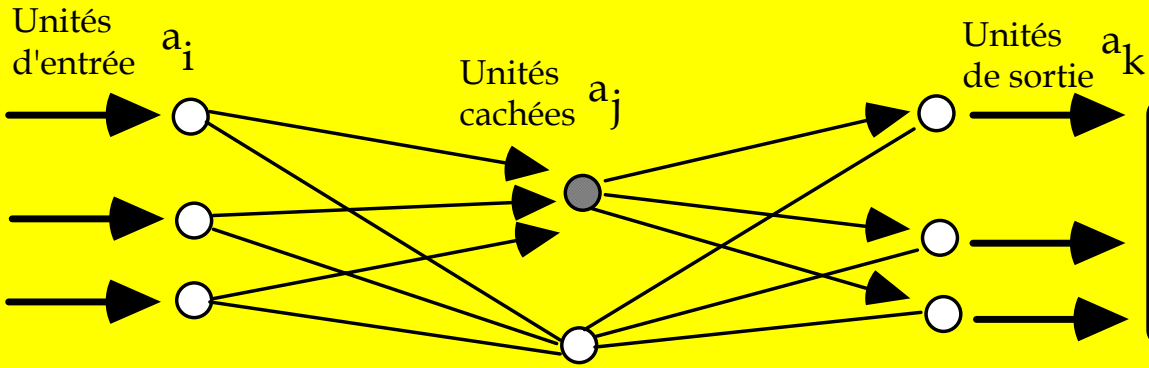
$$a_j = f(S_j)$$

1 Calcul activations unités cachées

$$S_k = \sum_j a_j W_{jk}$$

$$a_k = f(S_k)$$

2 Calcul activations unités de sortie




Calcul Erreur 3 entre sorties désirées et sorties obtenues

$$e_k = d_k - a_k$$

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

$$\delta_j = \left(\sum_k W_{jk}\delta_k\right).f'(S_j)$$

Calcul de l'erreur 5 sur les unités cachées

Calcul de l'erreur 4 sur les unités de sortie

$$\delta_k = e_k.f'(S_k)$$

6 **Ajustement des poids**

Apprentissage des unités cachées

Apprentissage des unités de sortie

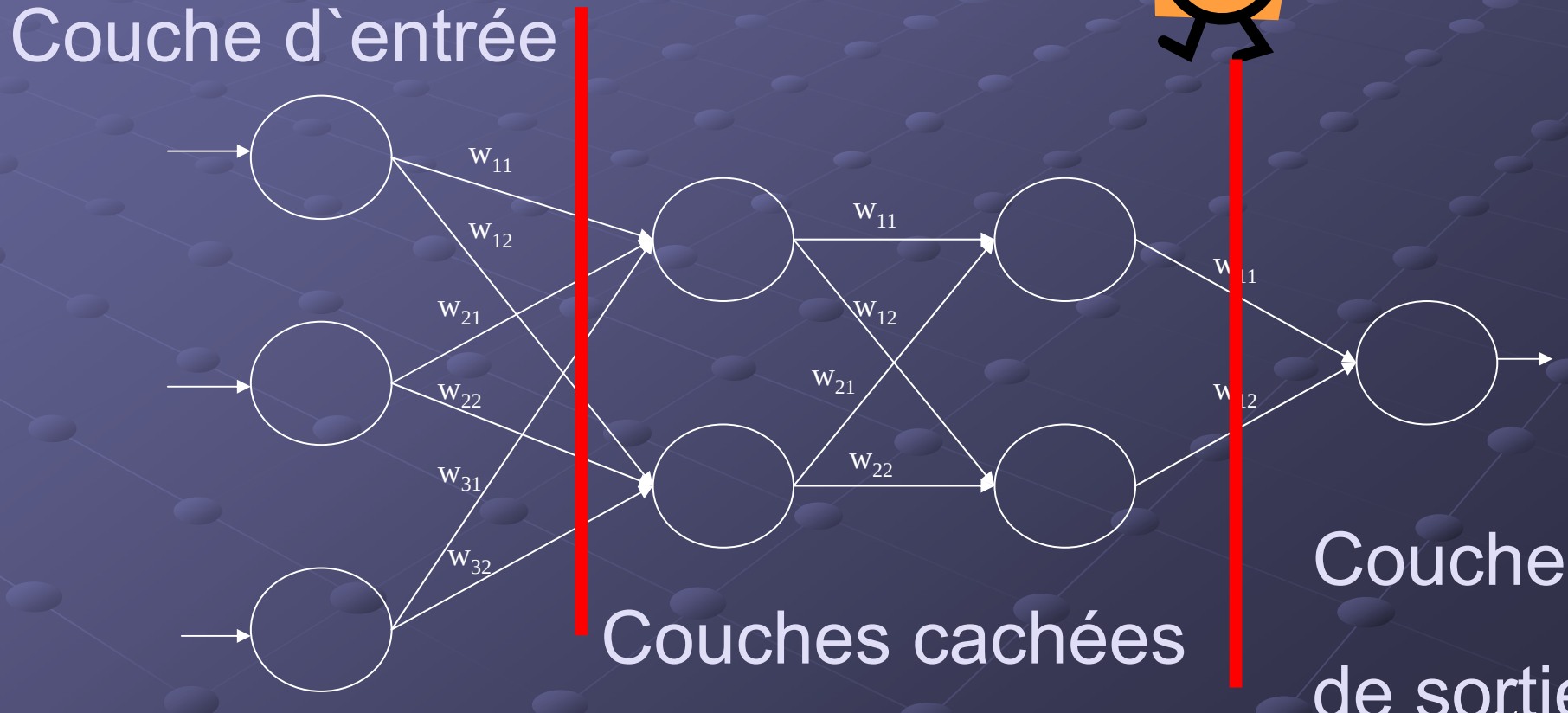
$$\Delta W_{ij} = \varepsilon \delta_j a_i$$

$$\Delta W_{jk} = \varepsilon \delta_k a_j$$

# MLP: Multi-layer perceptron

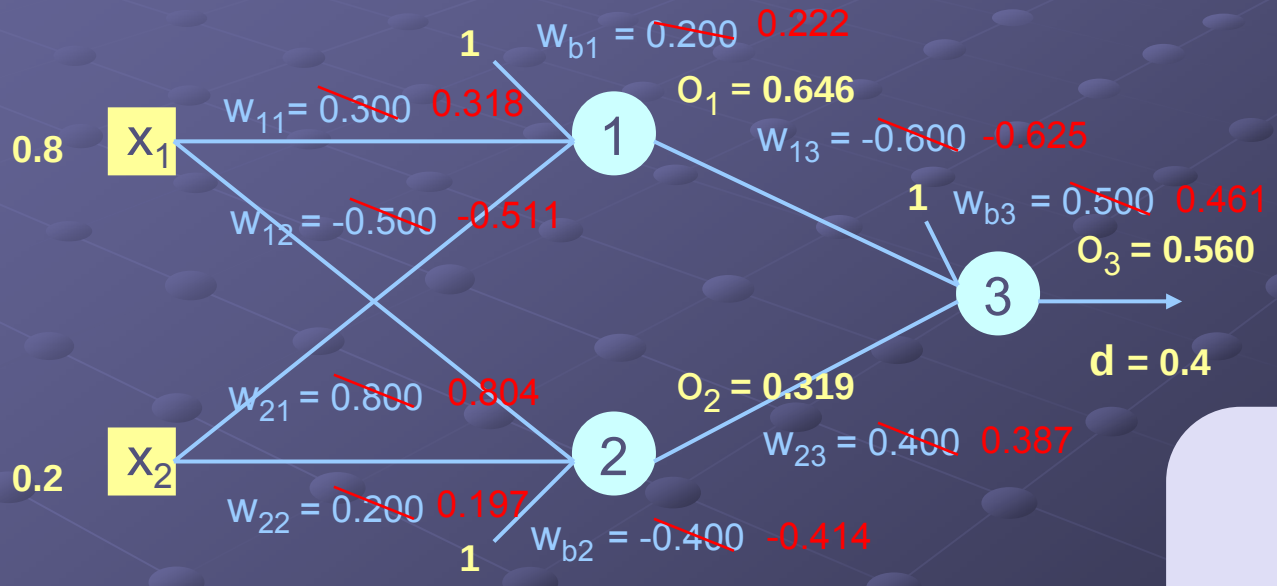
- Exemple:

Comment calculer $w_{ij}$ ????



# Exemple d'apprentissage : « Back propagation »

La figure ci-dessous présente un RNA dont les poids des liens entre les neurones et celui des biais ont été déterminés aléatoirement. Dans cet exemple, le vecteur à apprendre comporte les entrées $x_1 = 0.8$ et $x_2 = 0.2$, pour une sortie souhaitée $d = 0.4$. La fonction d'activation est la sigmoïde (équation (1) ) et sa dérivée est exprimée par l'équation (2). L'équation (3) illustre comment calculer la sortie d'un neurone et l'équation (4) comment modifier les poids entre les neurones lors de l'apprentissage (taux d'apprentissage $\mu = 1$). Remarquez que les poids sont toujours incrémentés selon: « $\mu$ • *erreur en sortie* • *f'(somme des entrées)* • *entrée du lien* ».




**Erreur de sortie de chaque neurone**

$e_3 = d - o_3 = -0.16$

$e_2 = w_{23} \cdot e_3 = -0.064$

$e_1 = w_{13} \cdot e_3 = 0.096$

$$f(x) = \frac{1}{1+e^{-x}} \quad (1)$$

$$f'(x) = f(x)\cdot[1-f(x)] \quad (2)$$

$$o_j = f\left(\sum_{i=1}^{N=2} x_j \cdot w_{ij} + w_{bj}\right) \quad (3)$$

$$w_{b1}(n+1) = w_{b1}(n) + \mu \cdot e_1 \cdot f'\left(\sum_{i=1}^{N=2} x_i \cdot w_{i1} + w_{b1}\right) \cdot 1 \quad (4)$$

$$w_{b1}(n+1) = 0.2 + (1 \cdot 0.096 \cdot f'(0.60) \cdot 1) = 0.222$$

# Le perceptron multicouche
## Règle du delta généralisé

- Les paramètres de l'apprentissage

$$\Delta W_{ij}^{t} = \varepsilon \delta_j a_i + \alpha W_{ij}^{t-1}$$

  - La force d'apprentissage ε ou taux d'apprentissage
  - Le momentum α
  - Gradient local δ
  - Cumulative Delta-Rule $\alpha W_{ij}^{t-1}$
- Les performances du réseaux
  - Erreur globale sur le jeu de test
  - Généralisation

# RPROP=Algorithme d 'apprentissage pour un réseau de type MLP

**Remarque :** que fait l 'algorithme Backprop traditionnel?

Il modifie les poids à partir de $\frac{\partial E}{\partial w_{ij}}$

**Problème :** la grandeur de cette différentielle ne représente pas vraiment la grandeur de la modification nécessaire du changement de poids

**Solution de RPROP** : ne pas se baser sur la valeur de cette différentielle, ne ***tenir compte que de ses changements de signe***

# Fonctionnement de RPROP

Backprop

$$\Delta w_{ij} = -u \frac{\partial E}{\partial w_{ij}}$$

Rprop

$$\Delta w_{ij} = -sign(\frac{\partial E}{\partial w_{ij}})\Delta_{ij}$$

# Fonctionnement de RPROP

- Calcul du gradient pour la couche externe :

$$\delta_i = (o_i^e - o_i) o_i (1 - o_i)$$

où $o_i$ est l'output du neurone $i$

$o_i^e$ est l'output attendu du neurone $i$

$o_i(1-o_i) = f'(o_i)$

# Fonctionnement de RPROP

- Calcul du gradient pour les couches internes

$$\delta_j = o_j(1-o_j)\sum_k w_{jk}\delta_k$$

où $o_j$ est l'output du neurone (interne) j

$k$ représente les indices des neurones de la couche supérieure

# Fonctionnement de RPROP

- Une fois qu'on connaît les gradients locaux de chaque neurone, on peut calculer l 'influence de chaque poids sur l 'erreur:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_j y_i$$

# Fonctionnement de RPROP

$$\Delta w_{ij}^{(t)} = -sign\left(\frac{\partial E}{\partial w_{ij}}\right)\Delta_{ij}$$

Sauf si la différentielle a changé de signe par rapport à (t-1) : à ce moment-là, on est passé au-dessus d ’un minimum local, et on revient donc au poids précédent (backtracking):

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)} \quad \text{, if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$$

# Fonctionnement de RPROP

- Qu'est-ce que

$\Delta_{ij}$ ?

= update-value : valeur de modification du poids, qui évolue en fonction des changements de signe des différentielles de ce même poids

- Les update-values et les poids ne sont changés qu'après chaque époque (batch learning). Pendant une époque, on additionne les différentielles obtenues après chaque présentation d 'un élément de l 'ensemble d 'apprentissage.

# Fonctionnement de RPROP

$$\Delta_{ij}^{(t)} = \eta^{+}\Delta_{ij}^{(t-1)} \quad , \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$$

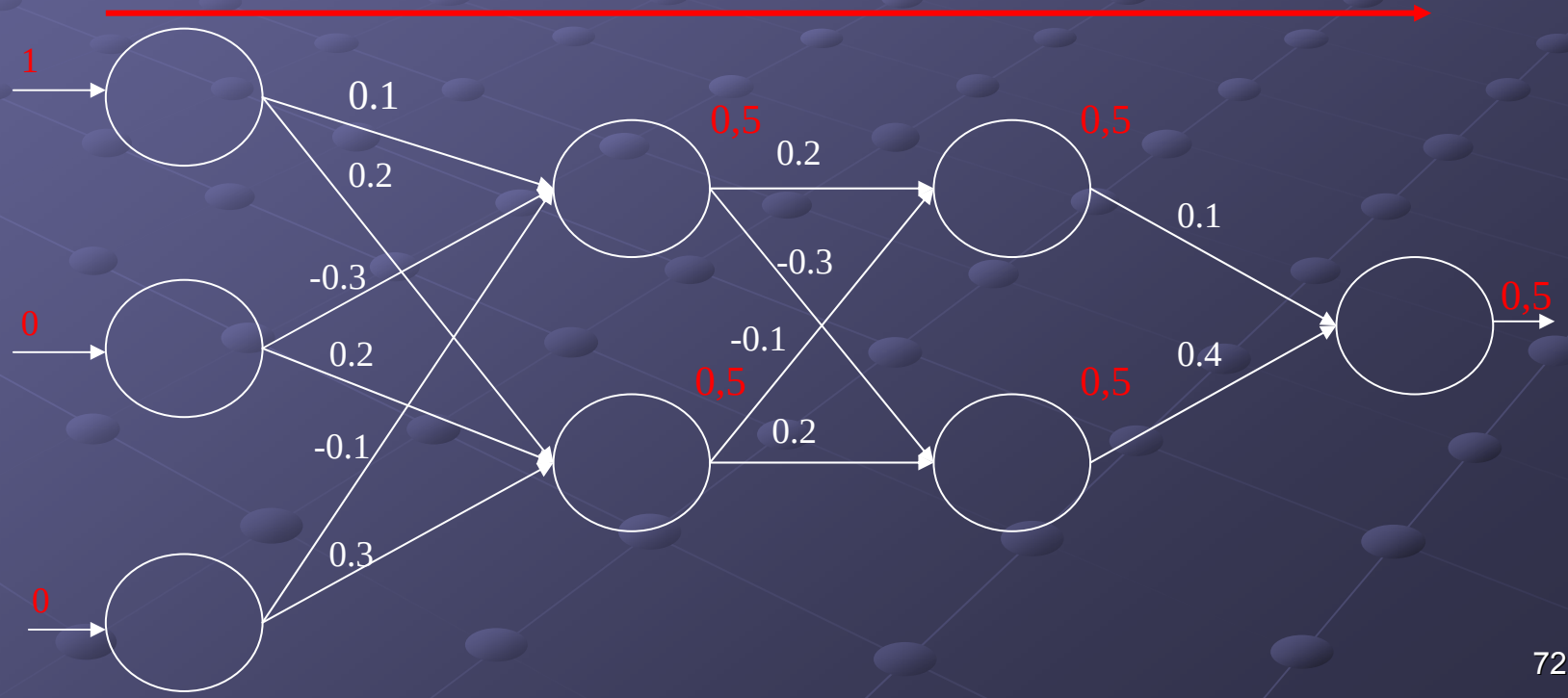
$$\eta\Delta_{ij}^{(t-1)} \quad , \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$$

$$\Delta_{ij}^{(t-1)} \quad , \text{else}$$

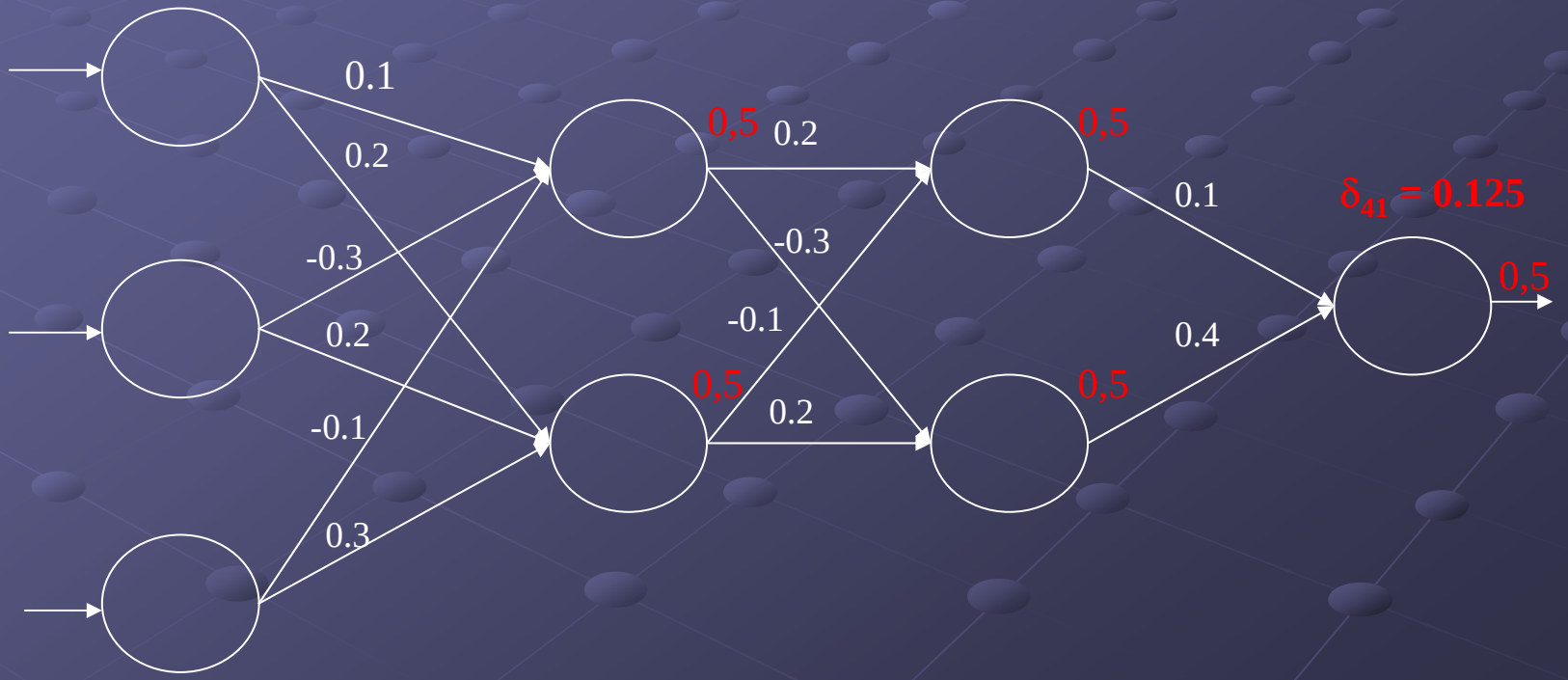
$$\text{where } 0 < \eta^{-} < 1 < \eta^{+}$$

# Fonctionnement de RPROP

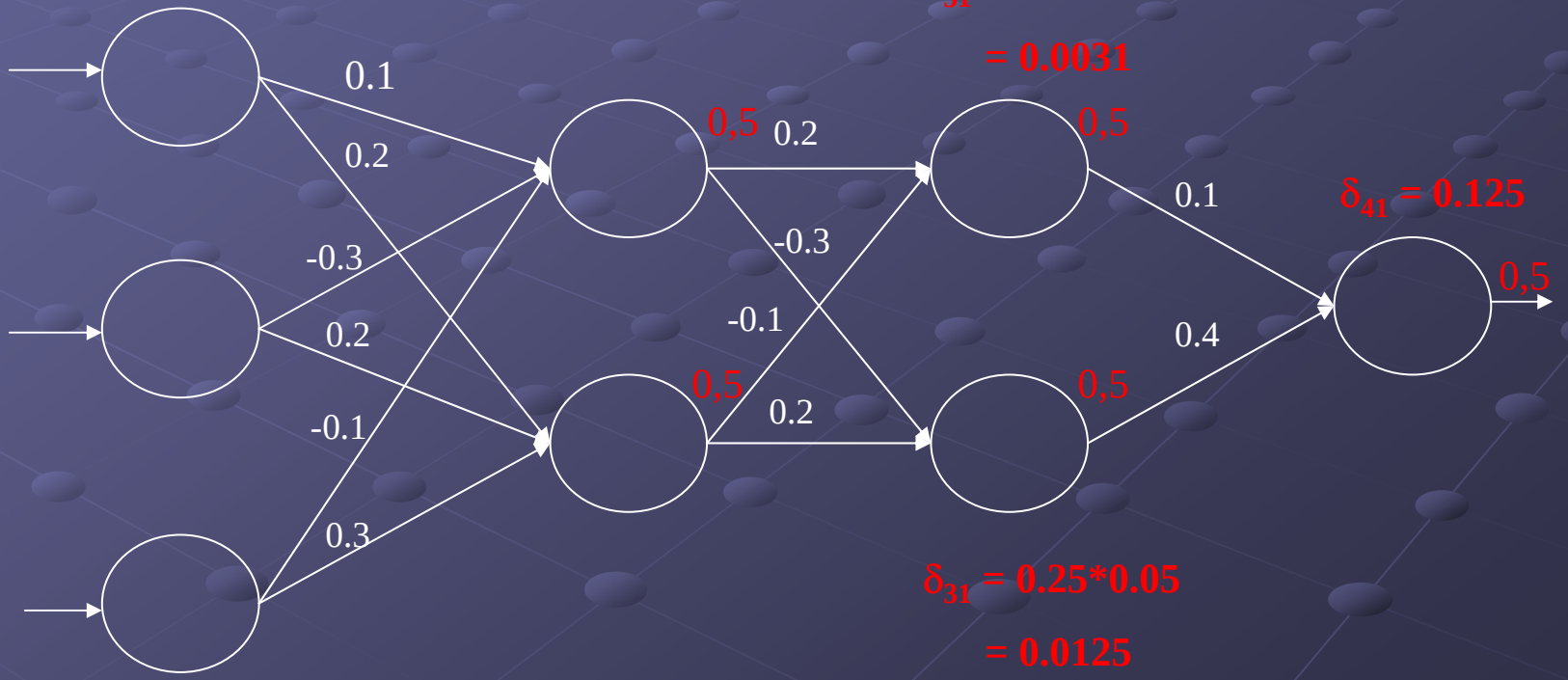
- Exemple de calcul de gradient : on donne l ’élément [1,0,0] et on attend la réponse 1



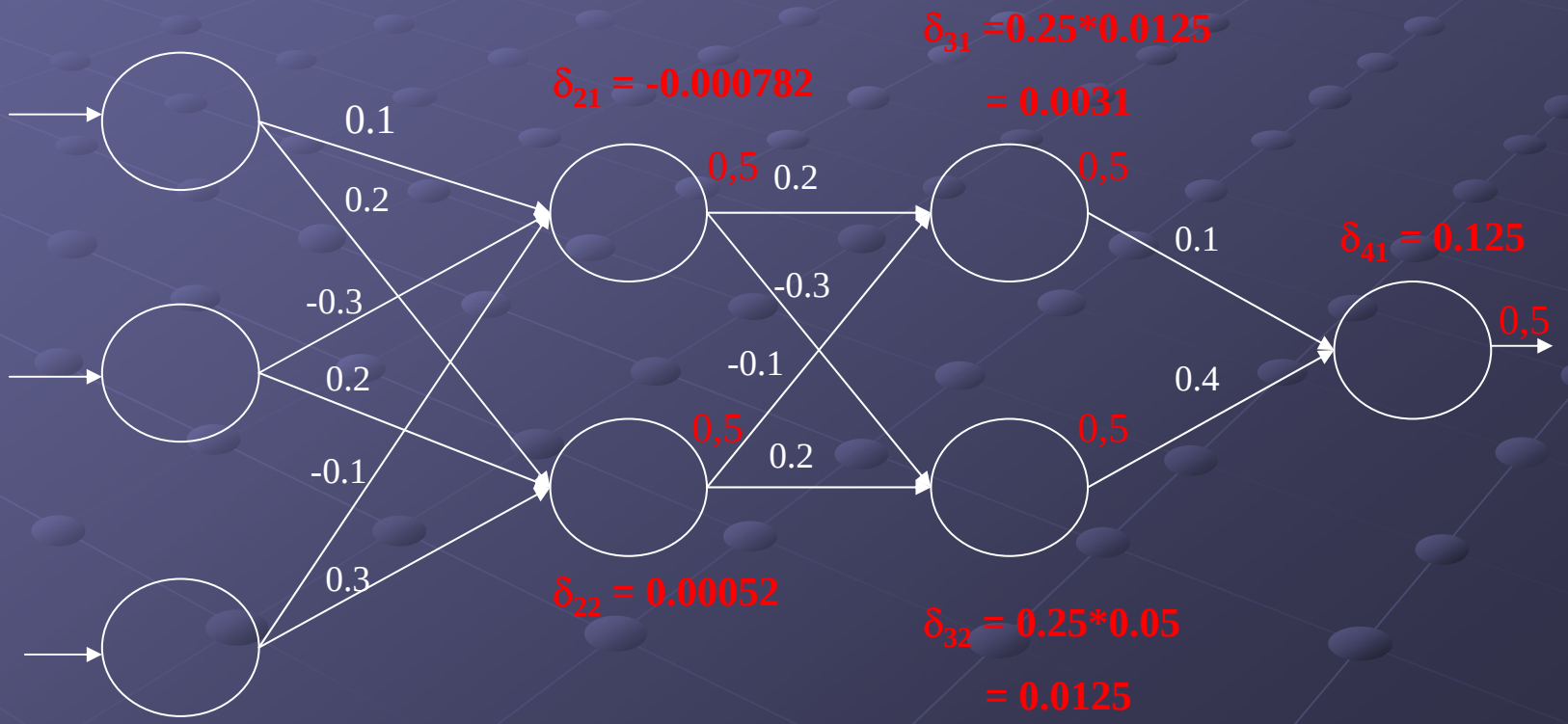
# Fonctionnement de RPROP



# Fonctionnement de RPROP



# Fonctionnement de RPROP



# Algorithme RPROP

**For all weights and biases{**

**if** $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0)$ **then** {

$\Delta_{ij}(t) =$ **minimum** $(\Delta_{ij}(t-1) * \eta^+, \Delta_{max})$

$\Delta w_{ij}(t) =$ **- sign** $(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

}

**else if** $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0)$ **then** {

$\Delta_{ij}(t) =$ **maximum** $(\Delta_{ij}(t-1) * \eta^-, \Delta_{min})$

$w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1)$

$\frac{\partial E}{\partial w_{ij}}(t) = 0$

}

**else if** $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0)$ **then** {

$\Delta w_{ij}(t) =$ **- sign** $(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

}

}

# Paramètres

- $\Delta_0$ : update-value initiale, pour chacun des poids (généralement 0.1)
- $\eta^+$ , $\eta^-$ : généralement 1.2 et 0.5 (on diminue de moitié car on ne sait pas de combien on a dépassé l 'erreur)
- $\Delta_{Max}$ $\Delta_{Min}$ : 50 et $1e^{-6}$
- Avantage de Rprop : le choix des paramètres influence peu la convergence

# Paramètres

- exemple

**10-5-10 encoder**

| Algo | $u/\Delta_0$ | Epochs | $WR(u/\Delta_0)$ |
|---|---|---|---|
| BP | 1.9 | 121 | [1.1,2.6] |
| RPROP | 2.0 | 19 | [0.05,2.0] |

# Approximation parcimonieuse Hornik

- Toute fonction bornée suffisamment régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.

- On montre [Barron 1993] que, si l'approximation dépend des paramètres ajustables de manière non linéaire, elle est plus parcimonieuse que si elle dépend linéairement des paramètres. L'écart entre l'approximation réalisée par un réseau de neurones et la fonction à approcher est inversement proportionnel au nombre de neurones cachés.

# Applications des R.N.A.

- Approximation de fonctions : les fonctions trop compliquées peuvent être approximées, grâce au réseau, par une somme de fonctions plus simples comme des polynômes ou des sigmoïdes.

- Optimisation de trajectoires : On peut, par exemple, déterminer quelle est la meilleure trajectoire pour un avion, une fusée…

- Reconnaissance : un réseau peut servir à reconnaître des caractères. Cela est déjà utilisé à la Poste pour lire les codes postaux, ou même dans certaines banques pour lire les chèques. Il est aussi possible de retrouver la prononciation des mots à partir d'un texte.

- Prévision : on utilise de plus en plus les réseaux pour faire des prévisions en marketing (prédiction de comportement, de possibilité de vente d'un produit, …) ou pour le trafic routier… Mais les prévisions en météo ou en bourse sont trop compliquées à réaliser.

- Contrôle : on peut contrôler les produits dans une industrie.

- Robotique : certains robots sont dotés de réseaux de neurones. Des entreprises japonaises se vantent déjà de leur utilisation, même pour des produits électroménagers ou informatiques.