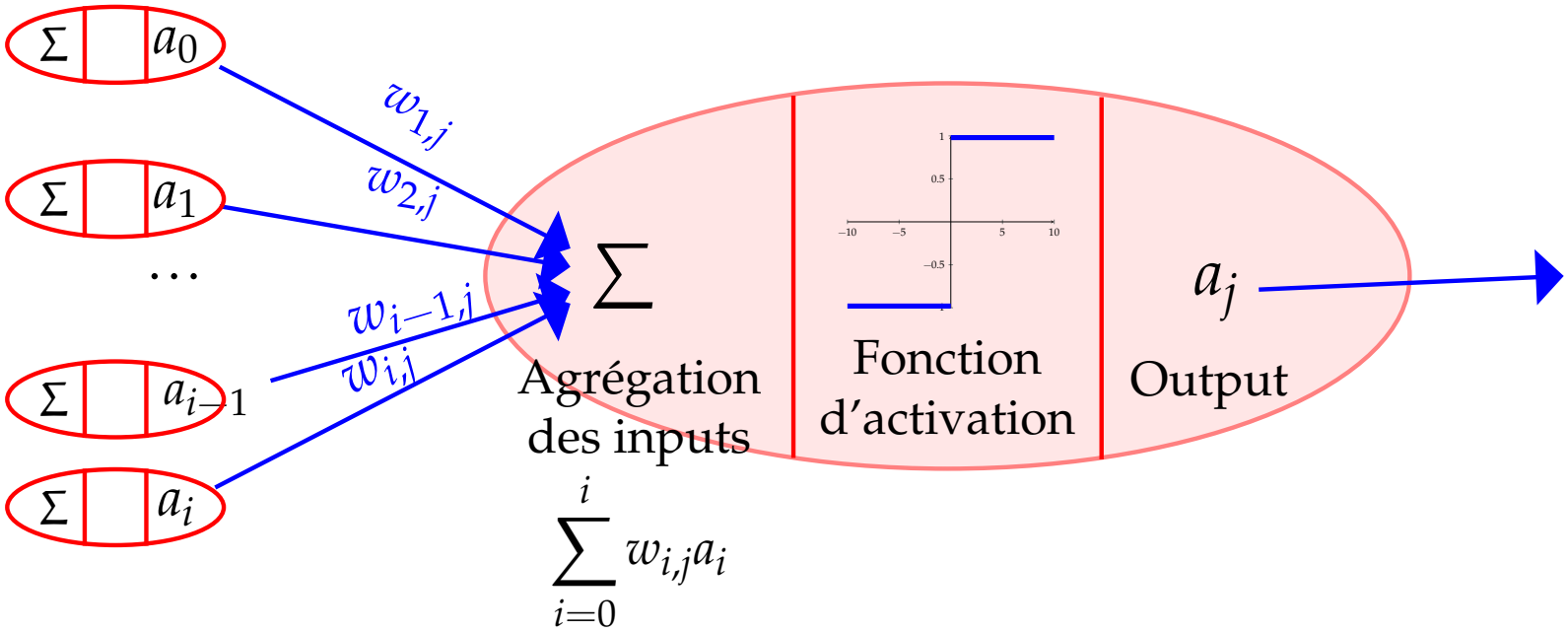
Modèle d'un neurone : le perceptron



Apprendre pour **un** perceptron **linéaire**

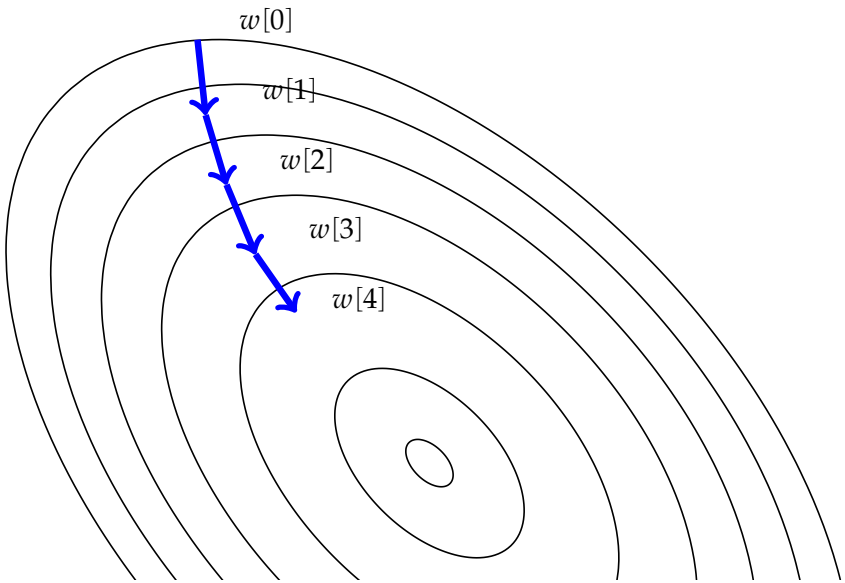
- On veut apprendre un vecteur de poids $\overrightarrow{w}$
- on va utiliser le principe la **descente de gradient** (la plus grande pente)
- L'erreur d'apprentissage est souvent mesurée par

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

où

- D est l'ensemble de exemples d'apprentissage
- t_d est la vraie classification de l'instance d
- o_d est la réponse du peceptron pour l'instance d

Descente de gradient de l'erreur



le gradient indique la direction de la pente la plus forte
La règle de mise à jour sera donc :

$$\overrightarrow{w} \leftarrow \overrightarrow{w} + \Delta\overrightarrow{w} \qquad w_i \leftarrow w_i + \Delta w_i$$

$$\text{où } \Delta\overrightarrow{w} = -\eta\nabla E(\overrightarrow{w}) \qquad \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

- η est le taux d'apprentissage qui determine la taille du pas que l'on fait en descendant
- le signe négatif indique que l'on veut descendre

Heureusement, calculer la dérivée est facile ici !

Descente de gradient avec fonction d'activation linéaire

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2\right) \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{i,d})
\end{aligned}
$$

La règle de mise à jour est donc

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{i,d}$$

Algorithme de descente de gradient avec fonction d'activation linéaire

```
1  initialise chaque w_i avec une valeur au hasard
2  tant que l'erreur est trop grande
3      Pour chaque i ∈ {1,...,n}
4          Δw_i = 0
5      Pour chaque example (x⃗, t) ∈ D
6          calculer la sortie o
7          Pour chaque i ∈ {1,...,n}
8              Δw_i = Δw_i + η(t − o)x_i
9      Pour chaque i ∈ {1,...,n}
10         w_i ← w_i + Δw_i
```

Algorithme pour **une** unité avec une fonction d'activation **linéaire**

Descente de gradient stochastique

- la descente peut être lente
- s'il y a plusieurs minimaux locaux, on n'a pas de garantie de trouver le minimum global

On utilise souvent une variante qui consiste à mettre à jour les poids après l'examen de chaque point (et pas après de les avoir tous examinés)

- c'est souvent une bonne approximation
- demande un peu moins de calculs
- permet parfois d'éviter de tomber dans un minimum local
  ⇨ plus de chances de tomber dans me minimum global

On veut apprendre des fonctions non linéaires

- on veut représenter des fonctions non linéaires
- avec la discontinuité du perceptron, on ne peut pas calculer de dérivées
- on remplace la fonction d'activation par une fonction **sigmoïde** (ou fonction logistique) qui est un approximation continue et différentiable de la fonction seuil

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

- on peut aussi utiliser la fonction tangente hyperbolique

$$tanh(y) = \frac{e^{y} - e^{-y}}{e^{y} + e^{-y}}$$

On peut refaire les calculs précédents pour calculer la nouvelle fonction de mise à jour pour ces fonctions d'activation.

Il faut passer au **réseau** de neurones!

Le **deep** learning consiste à avoir beaucoup de couches de neurones (ex DeepMind utilise des réseaux de 13 couches).

On va avoir des réseaux **multi-couches**

- une couche pour avoir les données
- couche cachée 1 : une couche de neurones connectés aux neurones de la couche d'entrée
- couche cachée 2 : couche de neurones connectés aux neurones de la couche d'entrée ou au neurones de la couche cachée 1
- …
- couche cachée k : couche de neurones connectés aux neurones de la couche d'entrée, des couches 1, 2, …, k-1
- …
- couche de sortie

Problème : on sait mesurer l'erreur pour la couche de sortie (en comparant avec les étiquettes)
⇨ comment mesurer l'erreur pour les neurones des couches cachées ?

Fonction d'erreurs

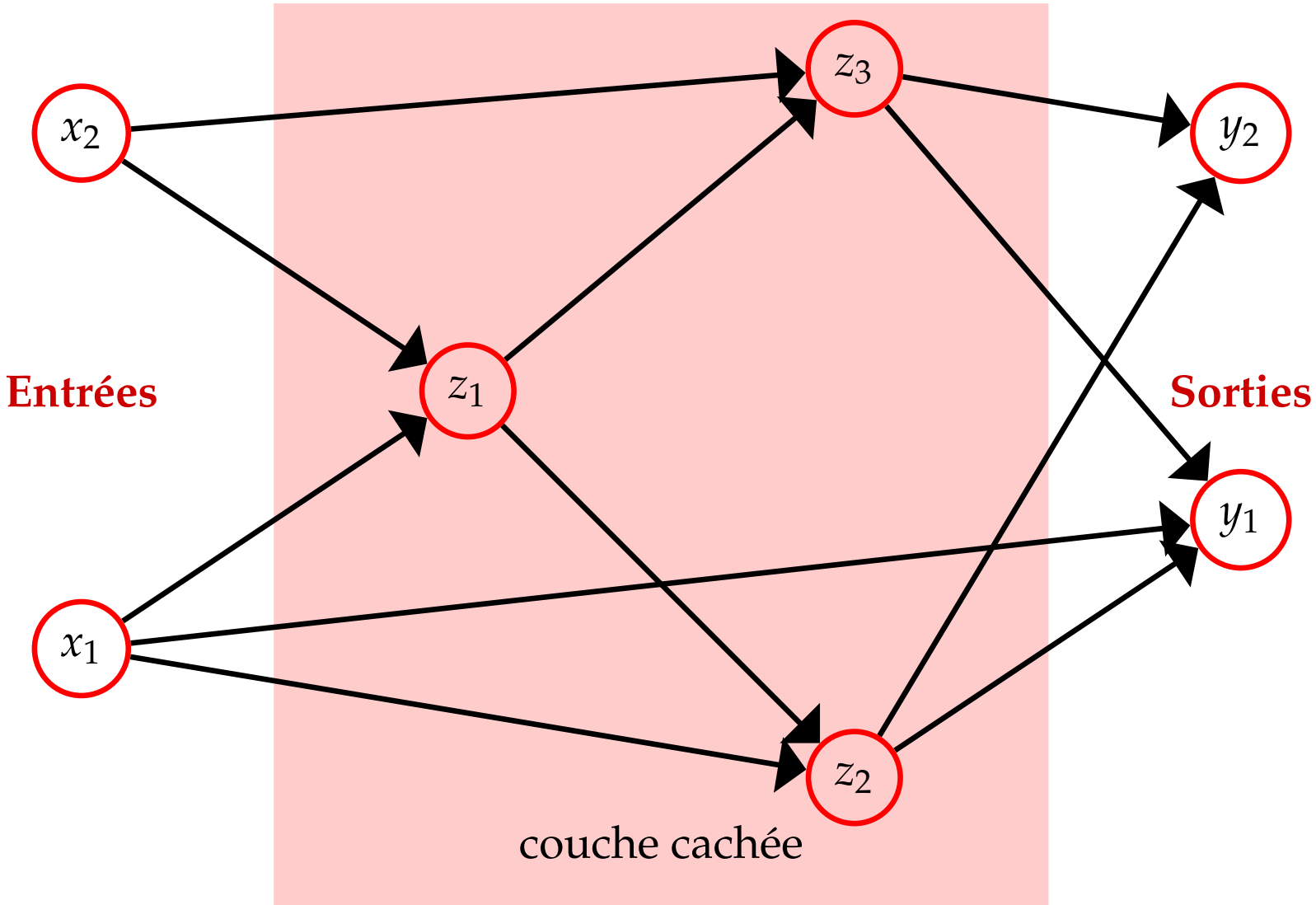
L'erreur d'apprentissage est mesurée par

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in sorties} (t_{k,d} - o_{k,d})^2$$

où

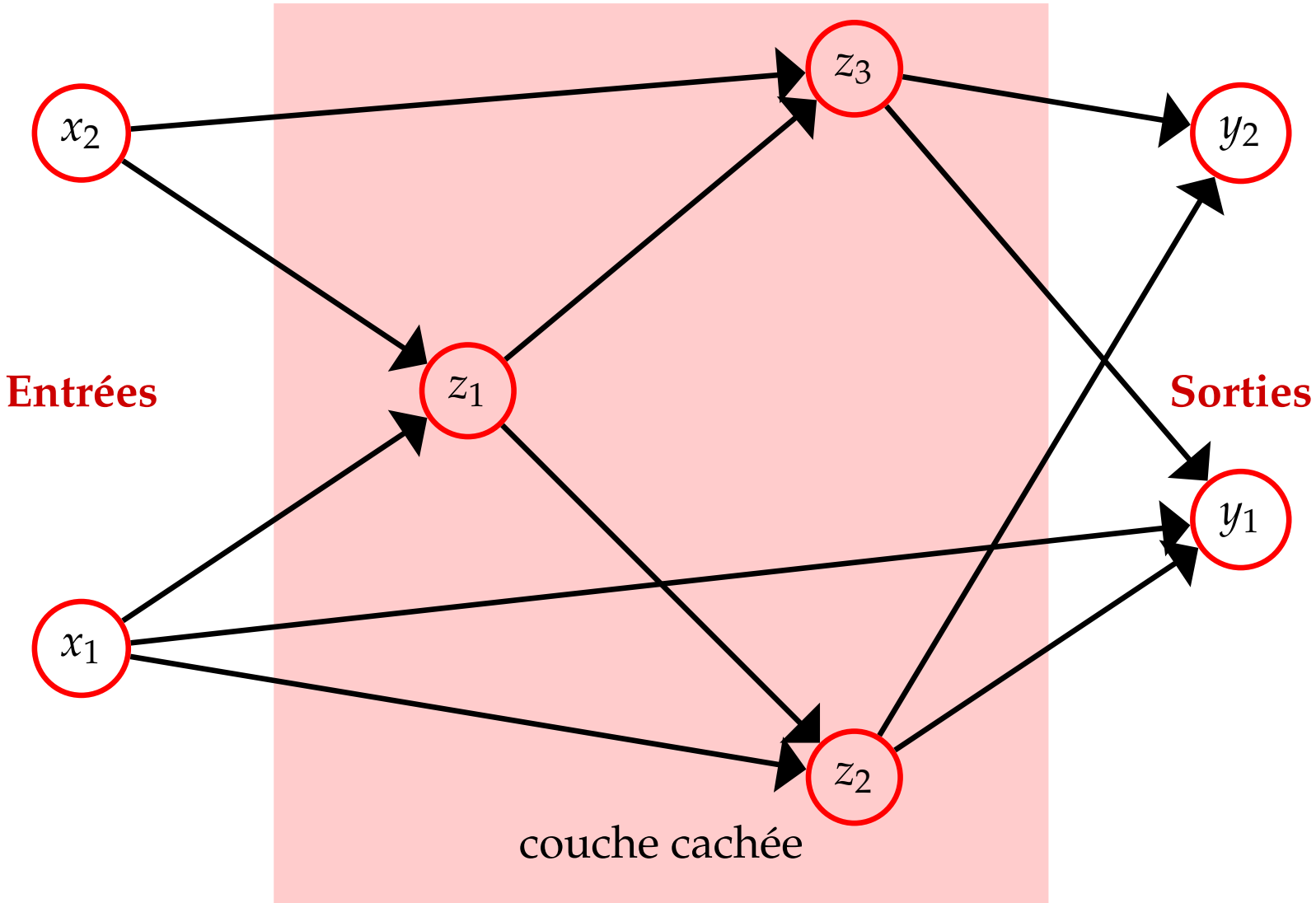
- D est l'ensemble des exemples d'apprentissage
- $t_{k,d}$ est la vraie classification de l'instance d pour la sortie k
- $o_{k,d}$ est la valeur de la sortie k pour l'instance d

Backpropagation



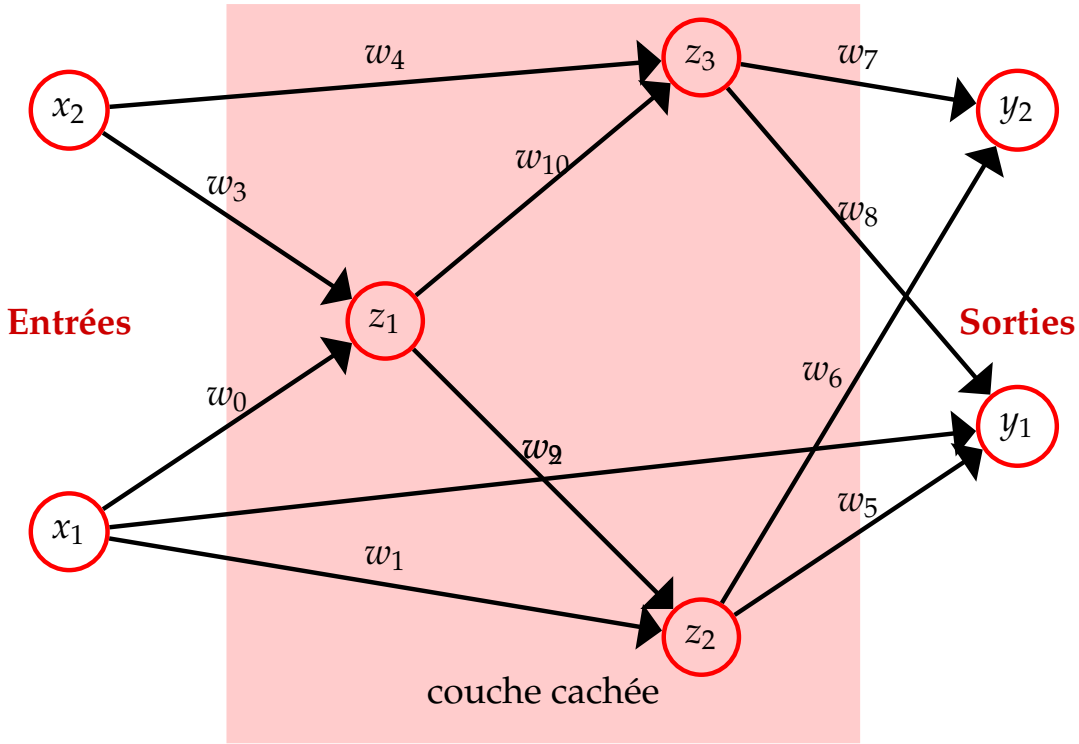
on peut calculer l'erreur aux sorties
mais il faut calculer une erreur pour les neurones de la couche interne !

Backpropagation



idée : on partage l'erreur en fonction des poids

Backpropagation



- z_3 contribue à la décision sur y_2 avec un poids de w_7
  ⇨on attribue à z_3 une partie de l'erreur de y_2 avec un poids de w_7
- z_3 contribue à la décision sur y_1 avec un poids de w_8
  ⇨on attribue à z_3 une partie de l'erreur de y_1 avec un poids de w_8

L'erreur de z_3 sera donc $w_7 \times erreur(y_2) + w_8 * erreur(y_1)$

Backpropagation algorithm

- x_{ji} est l'entrée du noeud j venant du noeud i we w_{ji} est le poids correspondant.
- δ_n est l'erreur associée à l'unité n. Cette erreur joue le rôle du terme $t-o$ dans le cas d'une seule unité.

initialise chaque w_i avec une valeur au hasard
tant que l'erreur est trop grande
    Pour chaque example $(\overrightarrow{x}, t) \in D$
        1- calcul de l'état du réseau par propagation
        2- rétro propagation des erreurs
            a- **pour chaque** unité de sortie k, calcul du terme d'erreur
                $\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$
            b- **pour chaque** unité cachée h, calcul du terme d'erreur
                $\delta_h \leftarrow o_h(1-o_h) \sum_{k \in sorties} w_{k,h}\delta_k$
            c- mise à jour des poids w_{ji}
                $w_{j,i} \leftarrow w_{j,i} + \eta \delta_j x_{ji}$

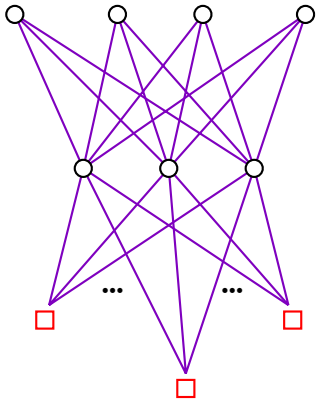
Propriétés

- backpropagation converge vers un minimum local (aucune garantie que le minimum soit global)
- en pratique, il donne de très bons résultats
  dans la pratique, le grand nombre de dimensions peut donner des opportunités pour ne pas être bloqué dans un minimum local
- pour le moment, on n'a pas assez de théorie qui explique les bons résultats en pratique
  - ajouter un terme de "momentum"
  - entrainer plusieur RNA avec les mêmes données mais différents poids de départ
    (boosting)

Quelles fonctions peut on représenter avec un RNA ?

- fonctions booléennes
- fonctions continues (théoriquement : toute fonction continue peut être approximée avec une erreur arbitraire par un réseau avec deux niveaux d'unités)
- fonctions arbitraires (théoriquement : toute fonction peut être approximée avec une précision aribitraire par un réseau avec 3 niveaux d'unités)

Exemple : reconnaissance de forme



20 personnes, environ 32 photos par personnes, leur tête peut être de face, tournée à gauche, droite, regardant en haut, différentes expressions (content, triste, en colère, neutre), avec ou sans lunettes de soleil. image de 120x128 pixels, noire et blanc, intensité entre 0 et 255.

taux de réussite de 90% pour apprendre l'orientation de la tête et reconnaître une des 20 personnes