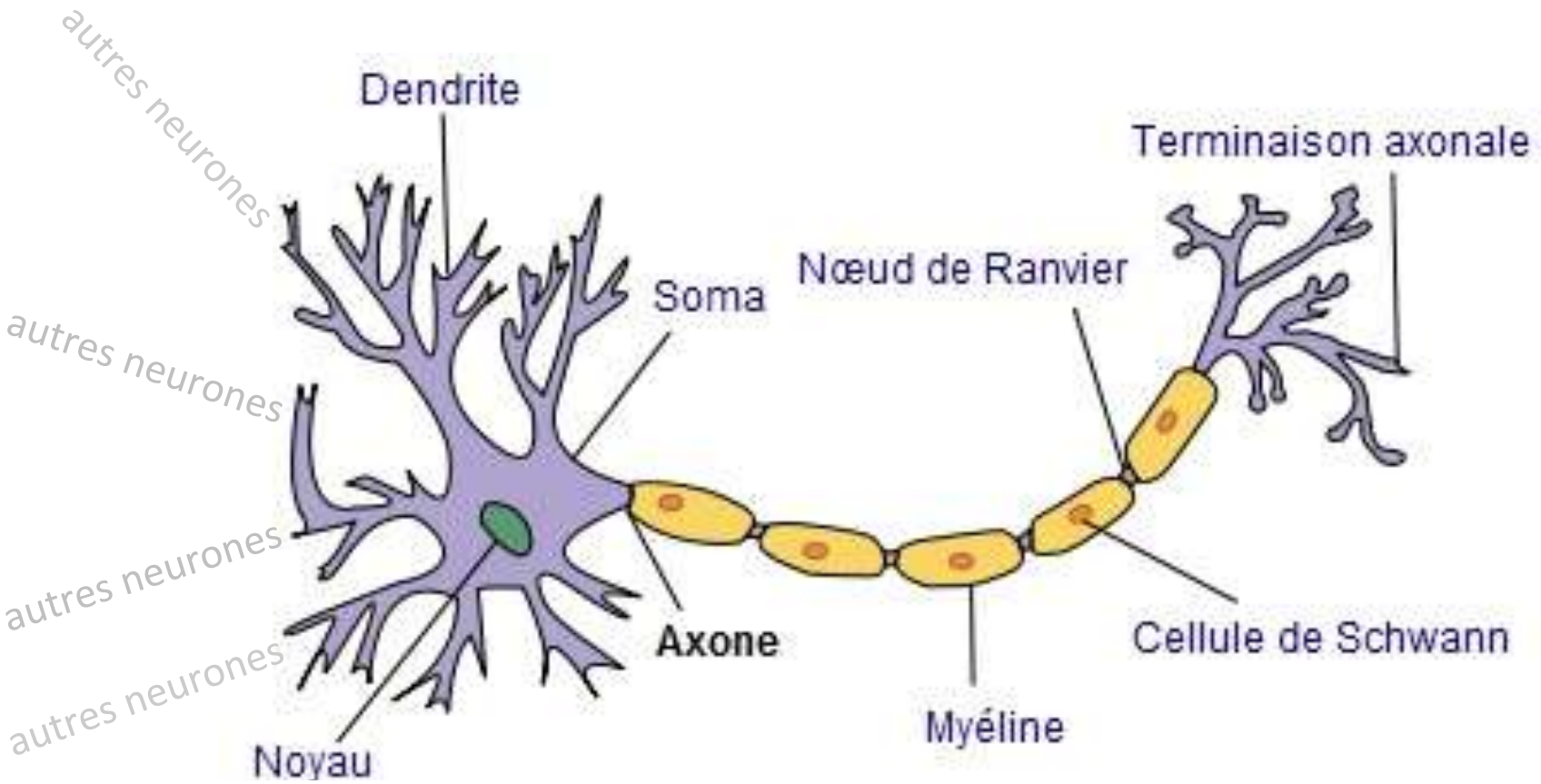
Réseaux neuronaux

Partie 1 : Modèle du neurone, Multiple Layer Perceptron

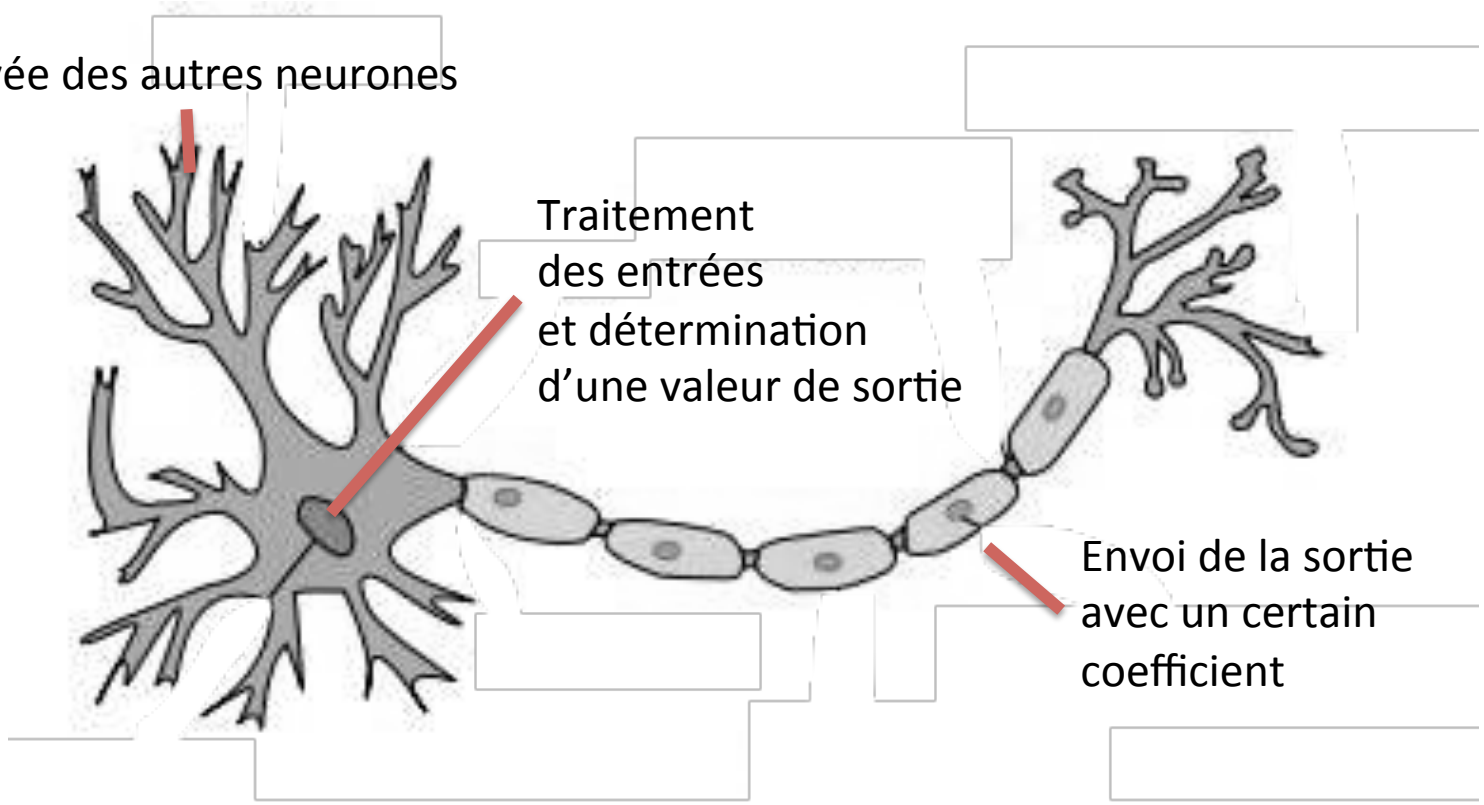
Pourquoi ?

- Visionique (reconnaissance de formes, “compréhension des images”…)
- Reconnaissance vocale
- Restitution d’image
- Reconnaissance de caractères
- Estimations boursières
- Classification des espèces animales
- Météorologie
- Industrie (apprentissage de mouvements, adaptation…)
- Jeux vidéos (ils deviennent meilleurs que nous ! :p)
- • •

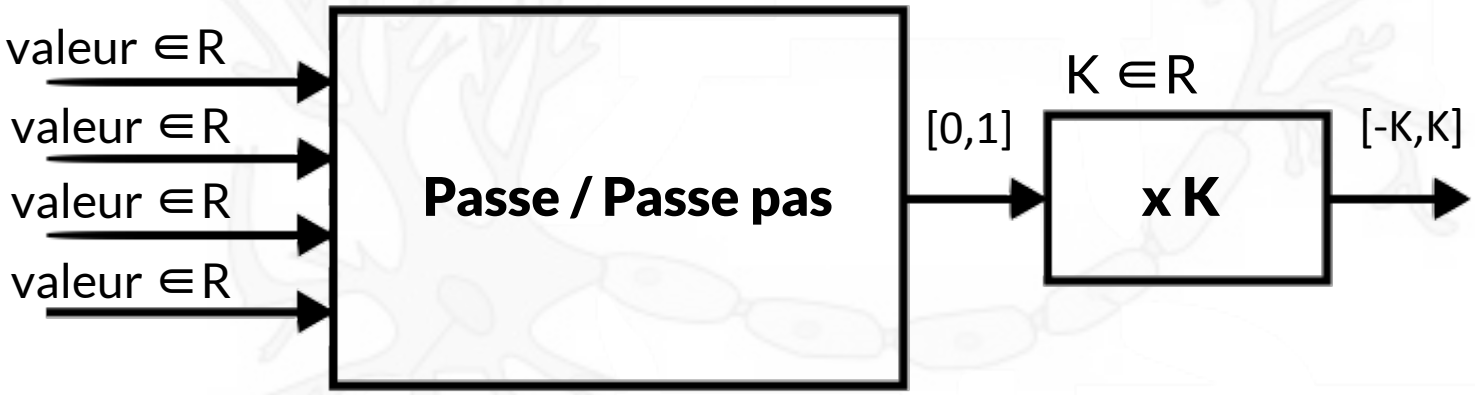
Le neurone biologique



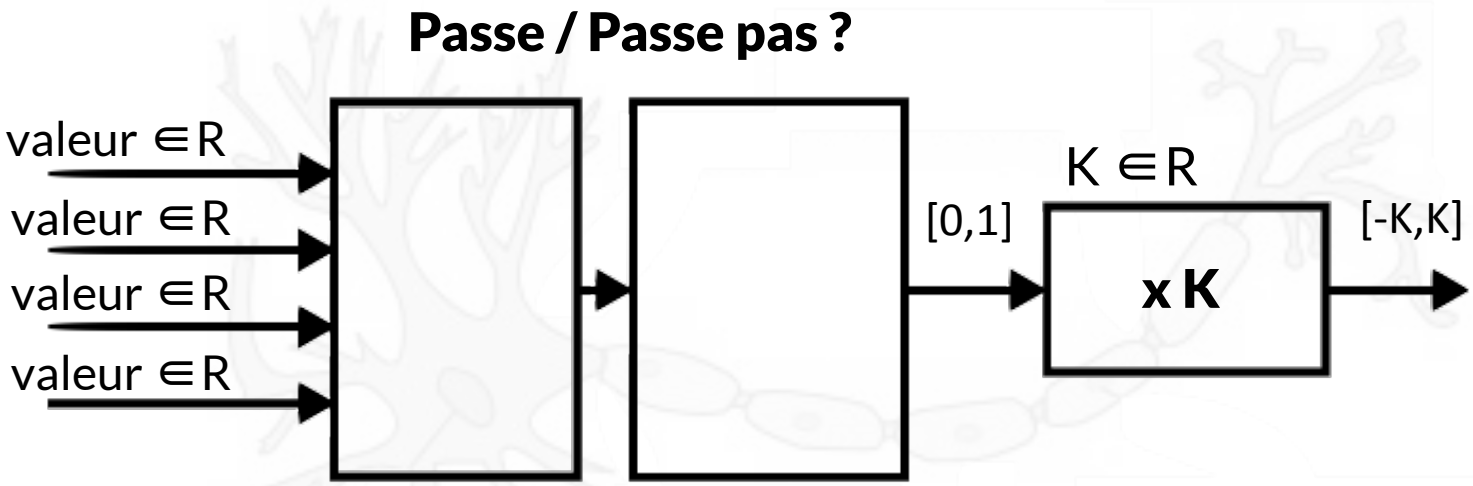
Le neurone compris par les informaticiens



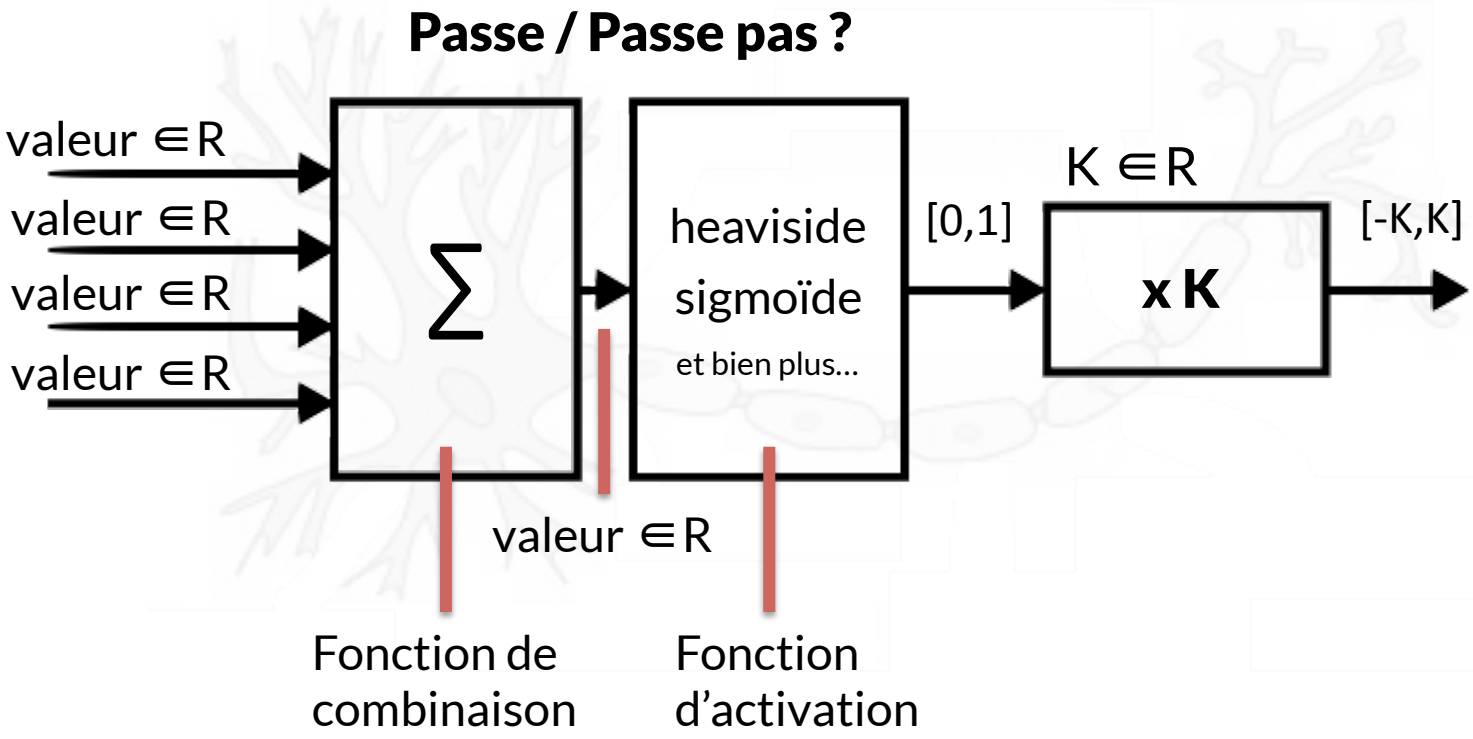
Le modèle mathématique



Le modèle mathématique

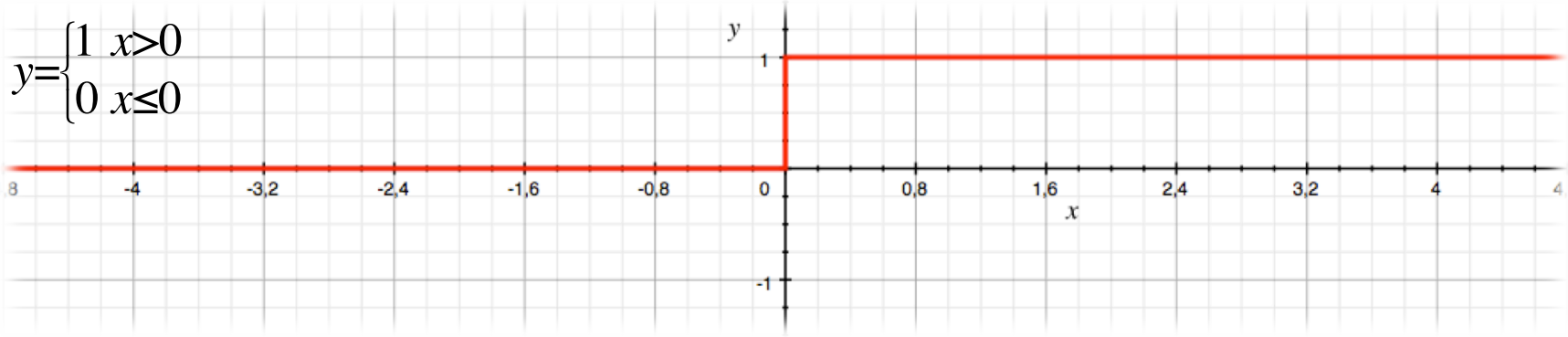


Le modèle mathématique



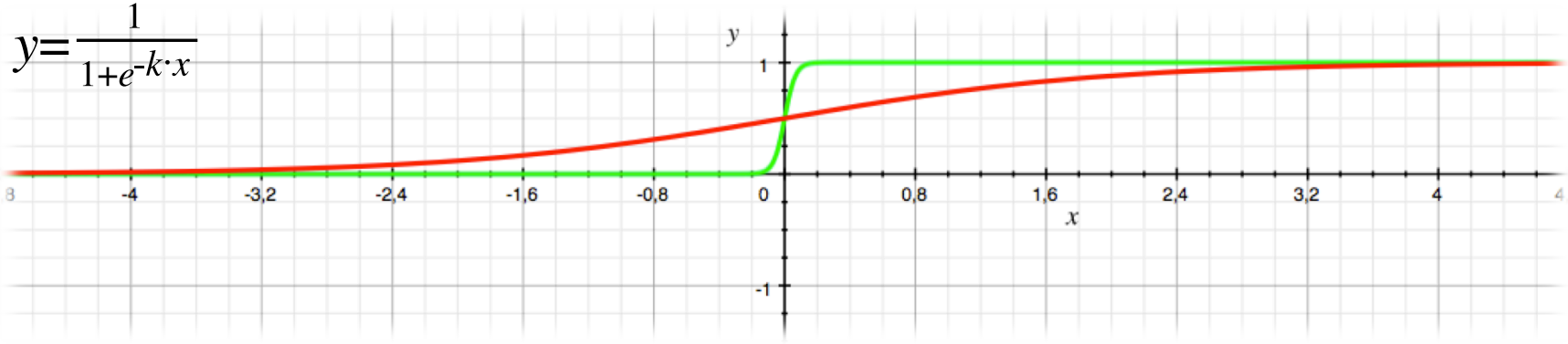
Heaviside (simple seuil)

$$y=\begin{cases}1 & x>0\\ 0 & x\leq 0\end{cases}$$



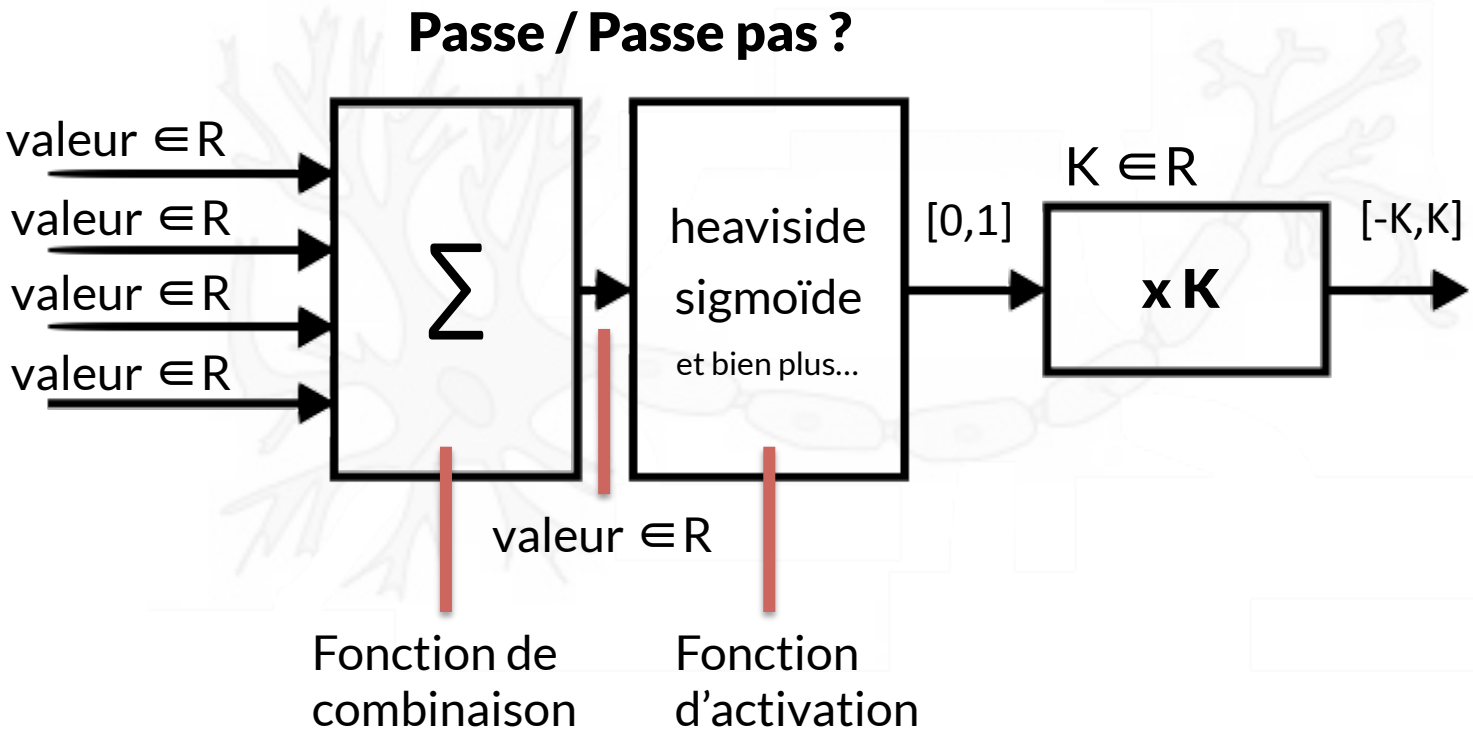
Sigmoïde (seuil progressif, avec facteur k)

$$y=\frac{1}{1+e^{-k\cdot x}}$$



[en vert k=30, en rouge k=1]

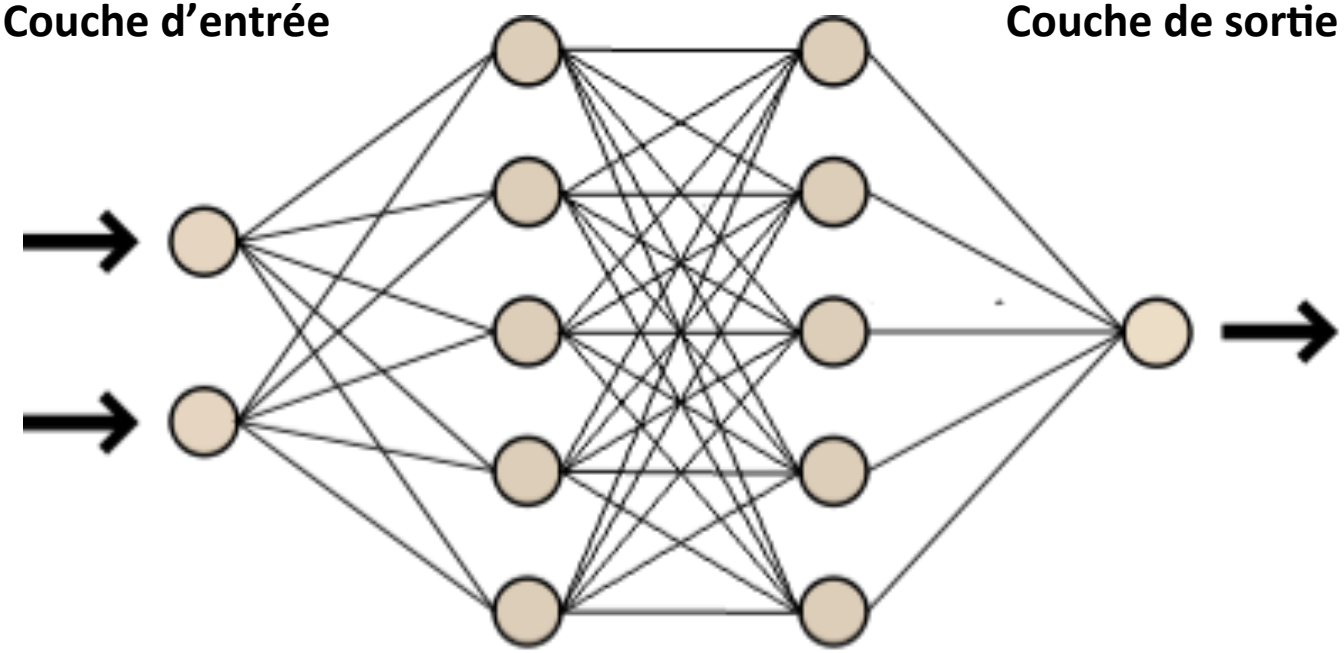
Le modèle mathématique



Maintenant on met plein de neurones !

Neurones d'entrée
Couche d'entrée

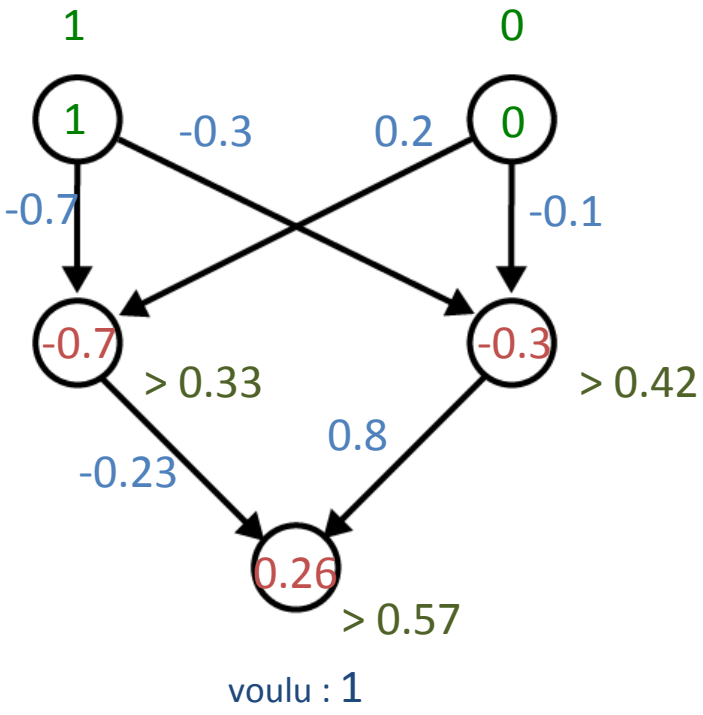
Neurones de sortie
Couche de sortie



Couches de neurones
dites **couches cachées**
(ici 2 couches cachées)

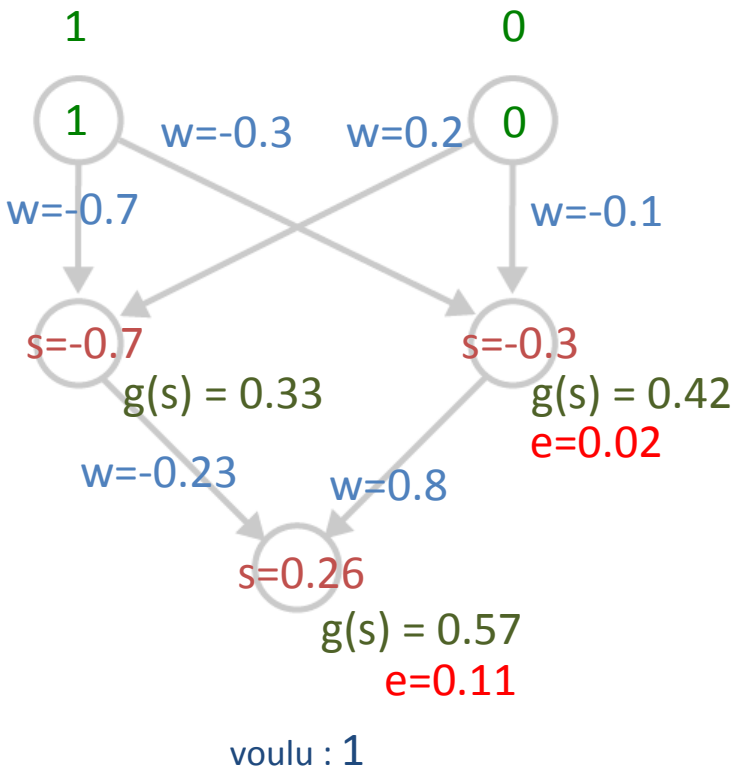
La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)



La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)



Pour *g* la fonction sigmoïde

$$g'(x) = \frac{k \cdot e^{-k \cdot x}}{(1+e^{-k \cdot x})^2}$$

$e = g'(s) * \sum(w_i * e_i)$ $= g'(-0.3) * (0.8 * 0.11) = 0.02$

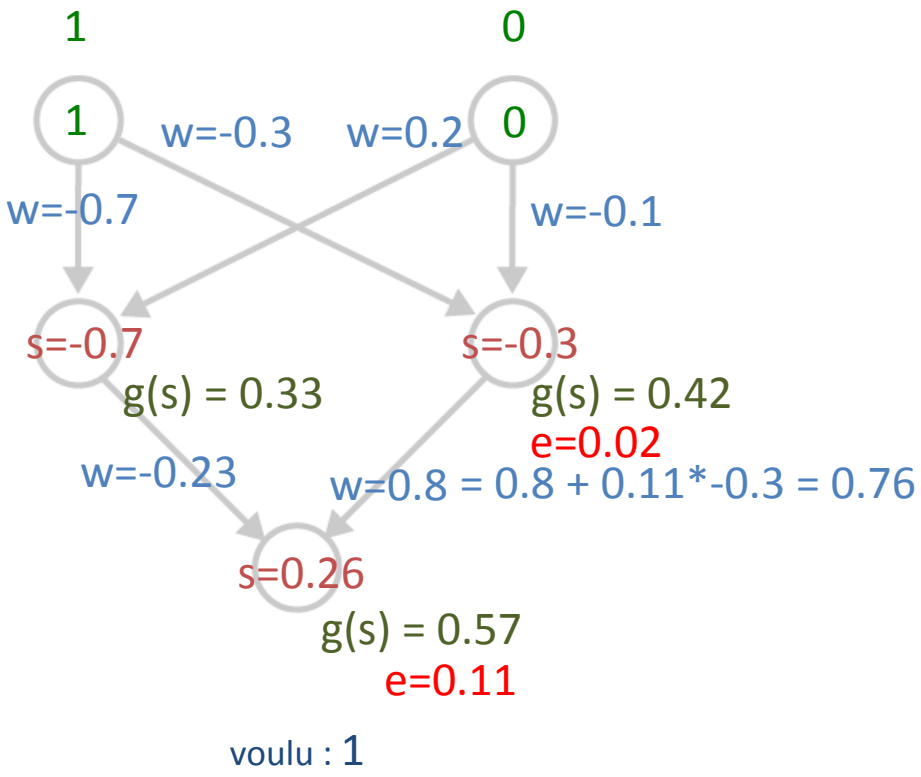
$\sum(w_i * e_i)$: poids et erreur des connexions sortantes
s = résultat de la combinaison (somme)
g = fonction d'activation (ici dérivée)

$e = g'(s) * \Delta r$ $= g'(0.26) * (1-0.57) = 0.25 * 0.43 = 0.11$

Δr = voulu - résultat
s = résultat de la combinaison (somme)
g = fonction d'activation (ici dérivée)

La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)

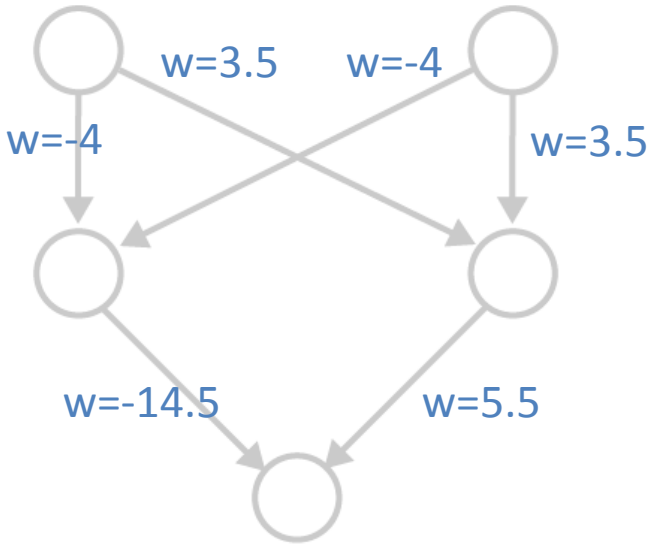


Modification des coefficients

$$w_{i->j} = w_{i->j} + \lambda . e_j . s_i$$

La partie difficile : rétropropagation de l'erreur

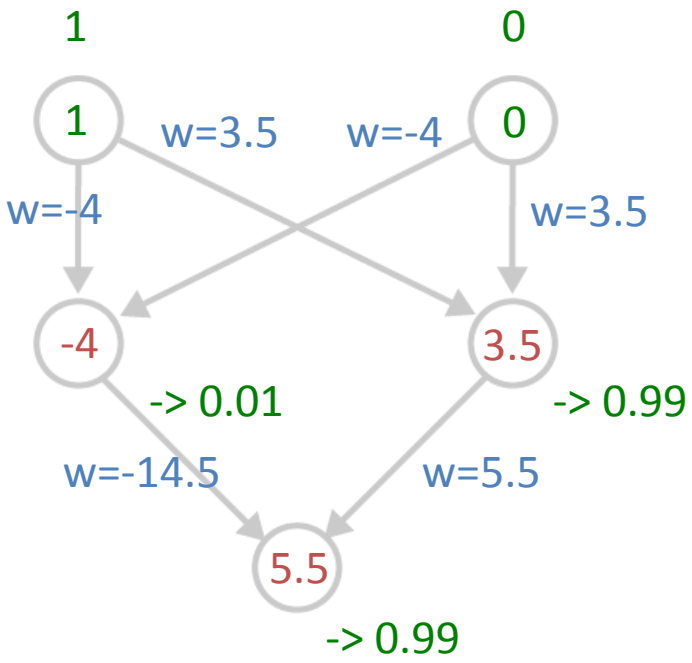
Exemple d'application : porte OR complexe (pour rien)



Au bout de quelques tests…

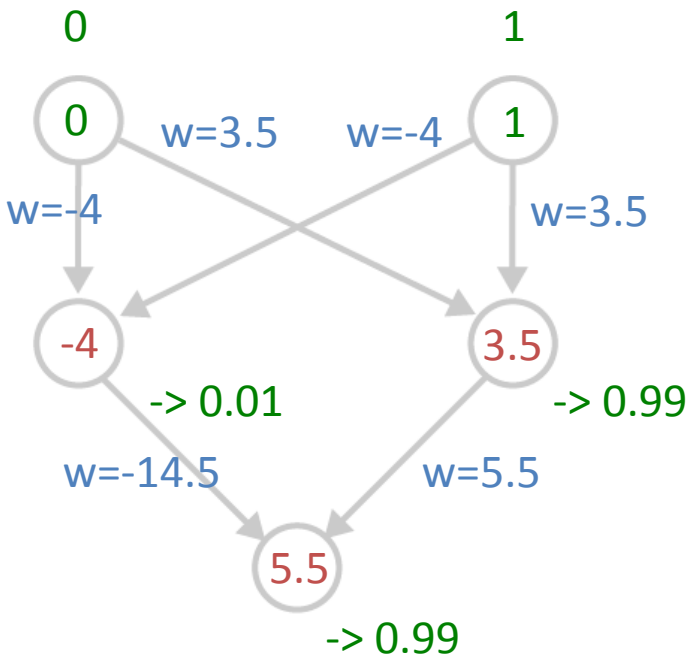
La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)



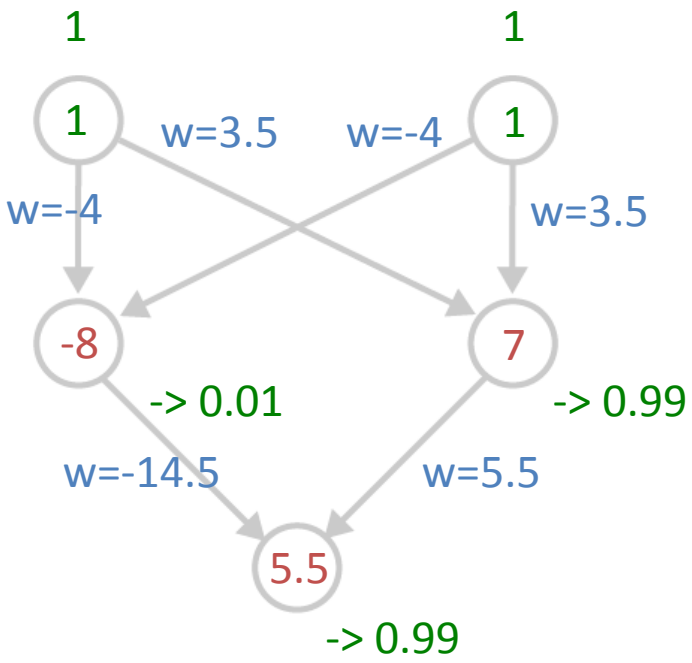
La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)



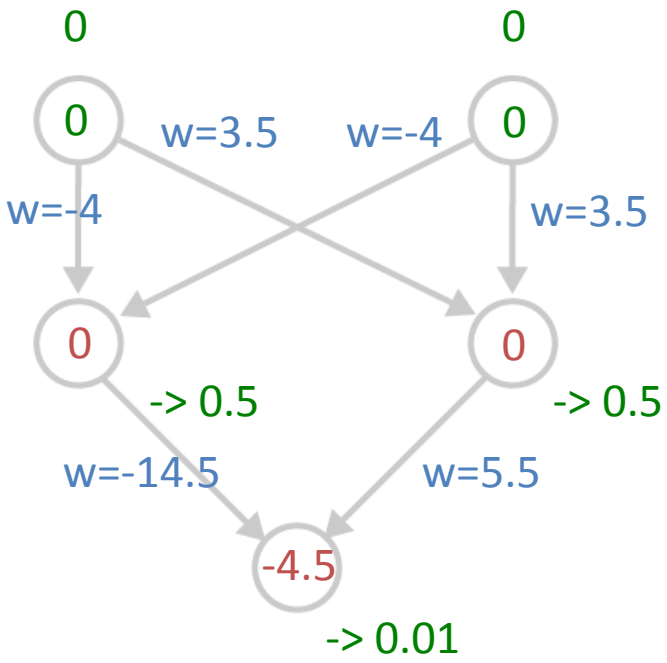
La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)



La partie difficile : rétropropagation de l'erreur

Exemple d'application : porte OR complexe (pour rien)

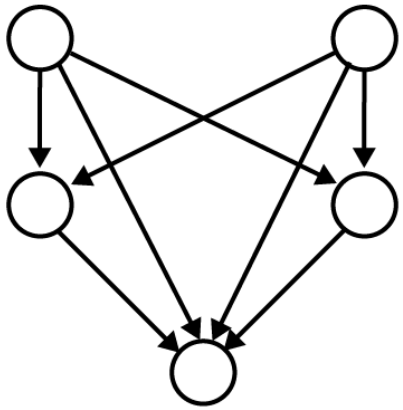


Un réseau neuronal c'est pas tout à fait autonome, il faut faire attention à :

- La structure du réseau
- La fonction d'activation (g, et k pour la sigmoïde)
- Le coefficient d'apprentissage (λ)
- Le sur-apprentissage

Conclusion, on dirait un joli logo pour un réseau XOR, c'est simple et c'est pratique !

Avez vous des question ?



Résumé des fonctions

Calcul de la somme : $x = \sum w_i . x_i$ (I les éléments précédent)

Fonction d'activation (sigmoïde) : $g(x) = y = \frac{1}{1+e^{-k \cdot x}}$

Dérivée de la fonction d'activation (sigmoïde) : $g'(x) = \frac{k \cdot e^{-k \cdot x}}{(1+e^{-k \cdot x})^2}$

Calcul de l'erreur pour la couche finale : $e = g'(x) * \Delta r$ (delta : attendu moins obtenu)

Calcul de l'erreur pour le reste des cas : $e_j = g'(x) * \sum(w_i * e_i)$ (i les éléments précédent)

Modification des valeurs : $w_{i->j} = w_{i->j} + \lambda . e_j . s_i$