

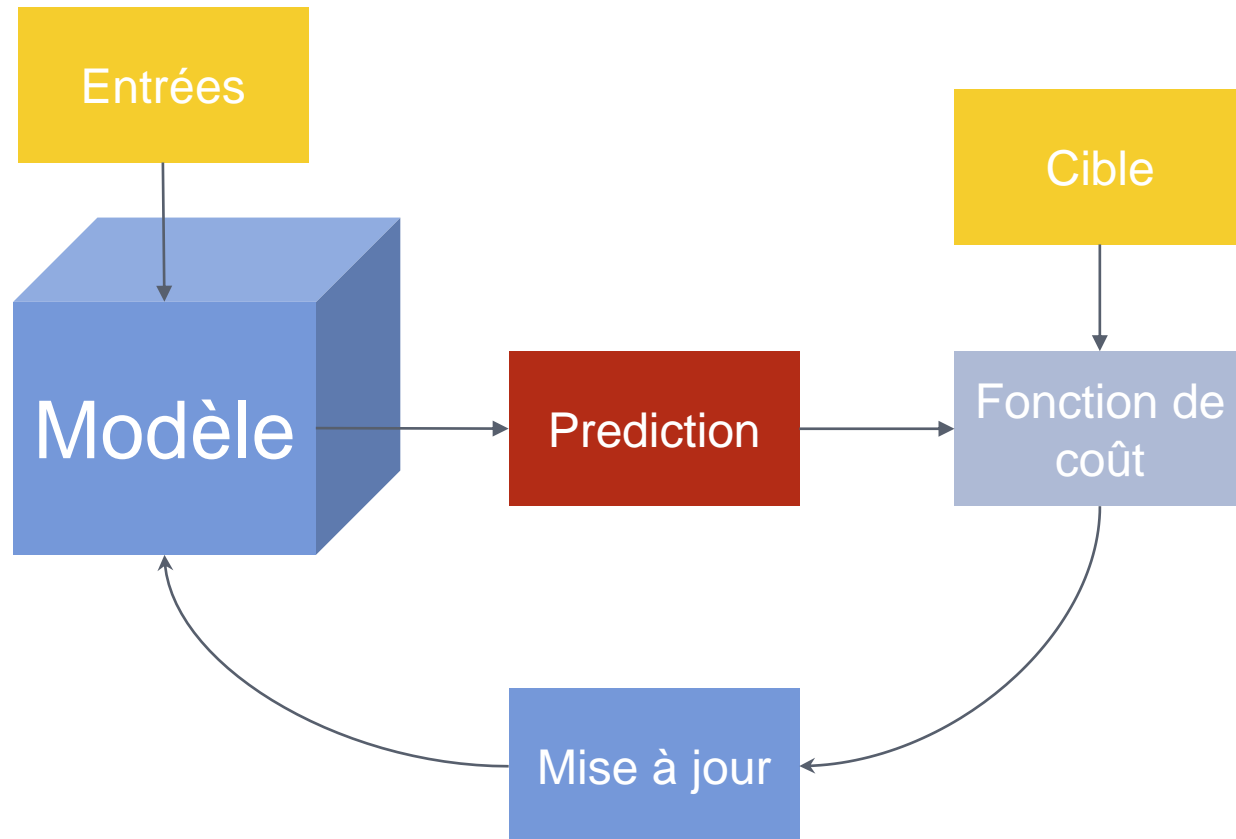
# Réseaux de Neurones

ANN : Artificial Neural Network

# I. RAPPELS

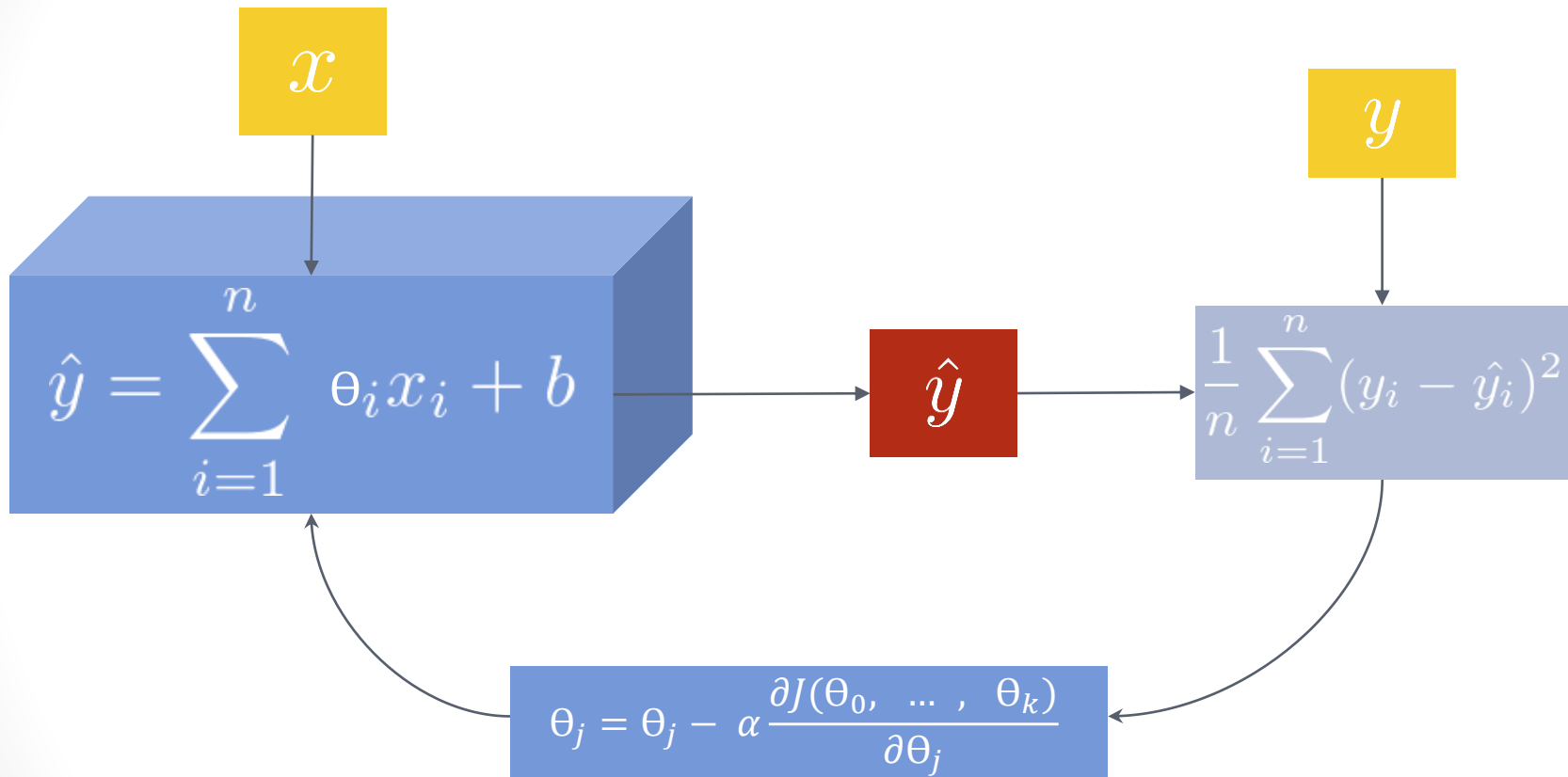


# Apprentissage





# Apprentissage (maths)



# Fonction hypothèse et de coût



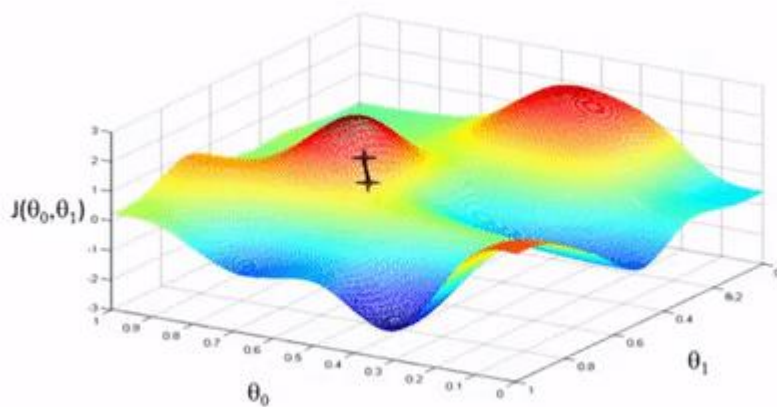
- La fonction **hypothèse** est celle qu'on essaie de paramétrer pour obtenir la bonne sortie à partir des entrées.
- Le type de fonction hypothèse caractérise la méthode : ANN, Arbre, Régression...
- La fonction **de coût** est la fonction que l'on va chercher à minimiser dans l'apprentissage. On l'appelle aussi **Loss**.
- La fonction de coût peut varier suivant le problème considéré.
- Nous verrons bientôt (Descente de gradient) qu'une fonction de coût devrait être continue, dérivable et convexe.





# Descente de Gradient


- Ainsi, un problème de machine Learning devient un problème d'optimisation: Comment minimiser la fonction de coût en fonction des paramètres  $(\theta_0, \dots, \theta_k)$ .
- Pour cela, nous disposons d'un algorithme classique : La Descente de Gradient.

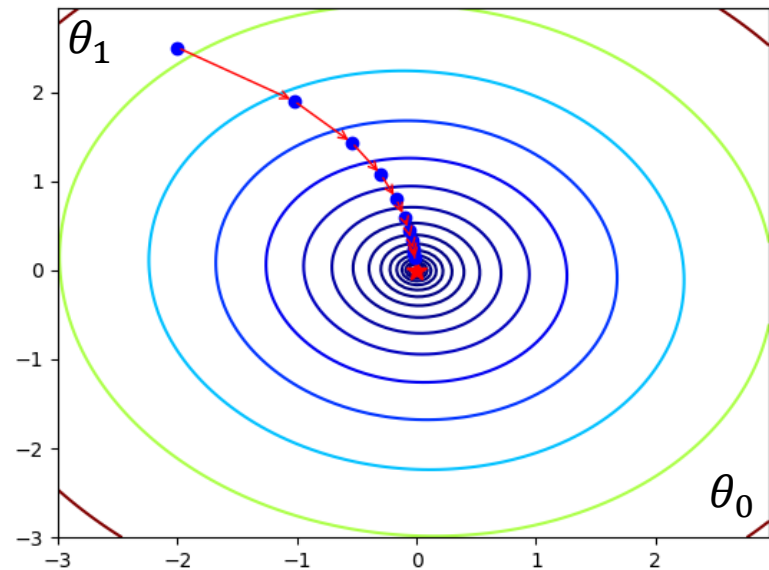
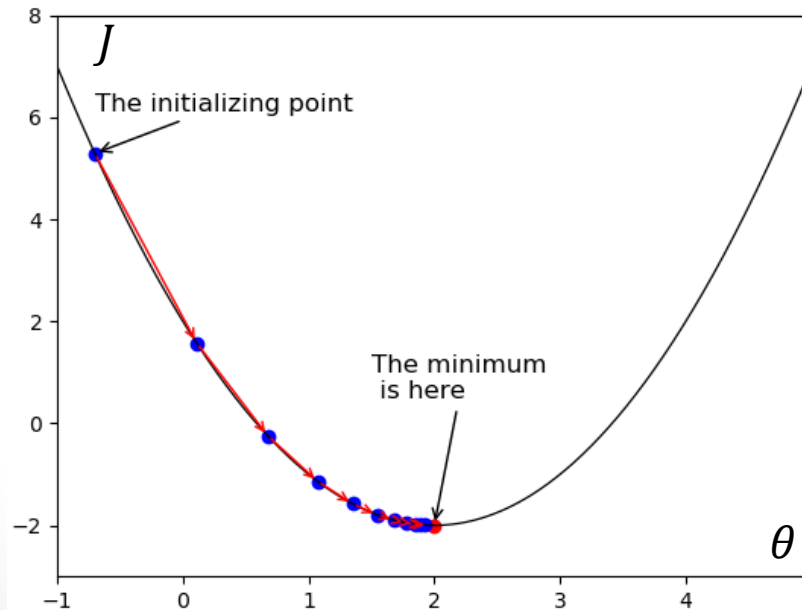


- Intuitivement, L'algorithme est analogue à la descente d'une colline douce par un randonneur.
- A chaque pas, il va chercher la direction qui descend le plus

# Descente de Gradient



- On applique la descente simultanément pour tous les paramètres  $\theta_j$
- $J$  est la fonction de coût.  $\alpha$  est le **Learning Rate** ( $> 0$ ).
- Pour  $j$  dans  $\{0, 1, \dots, k\}$   $\theta_j = \theta_j - \alpha \frac{\partial J(\theta_0, \dots, \theta_k)}{\partial \theta_j}$  



## II. DES ANN AUX DNN

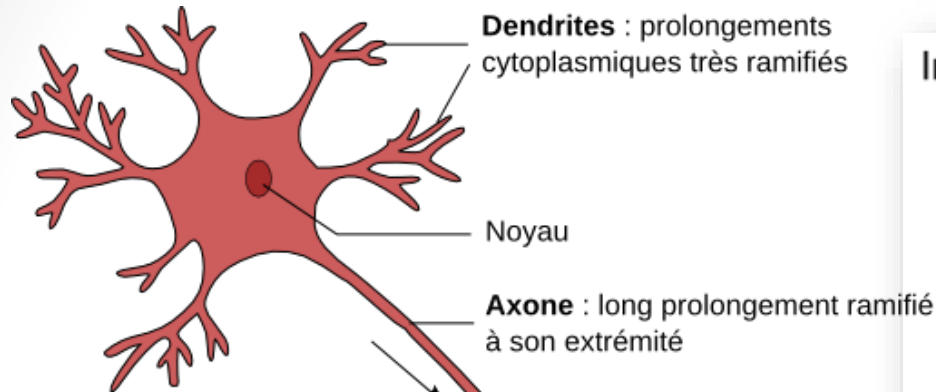
ANN : Artificial Neural Network.

DNN : Deep Neural Network.

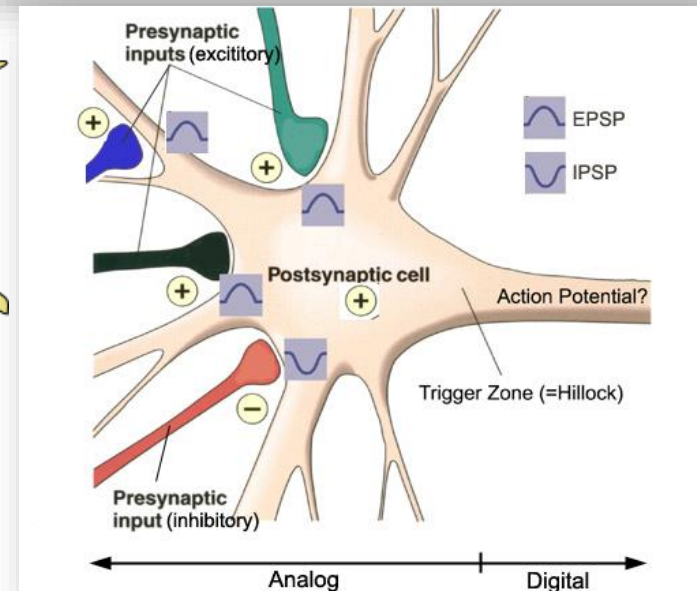
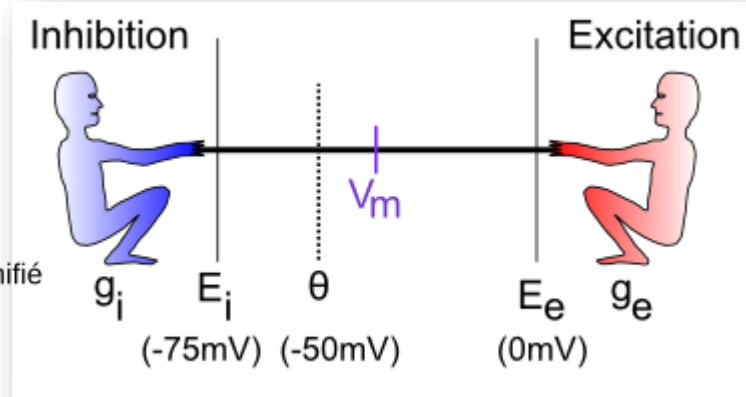
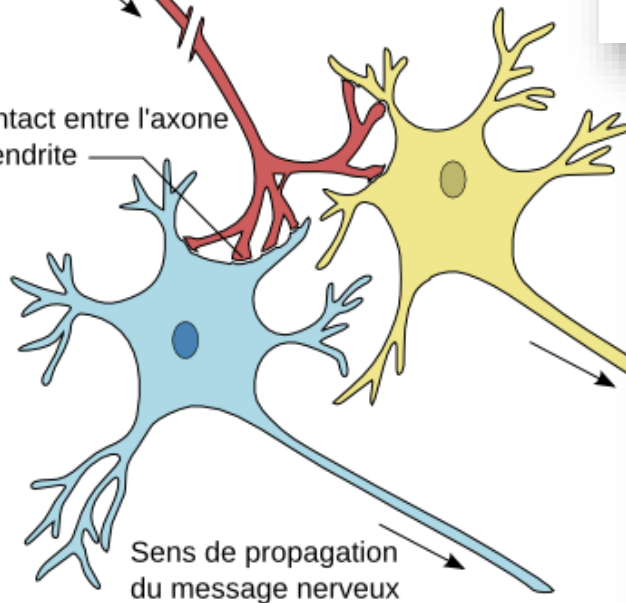


# La métaphore biologique

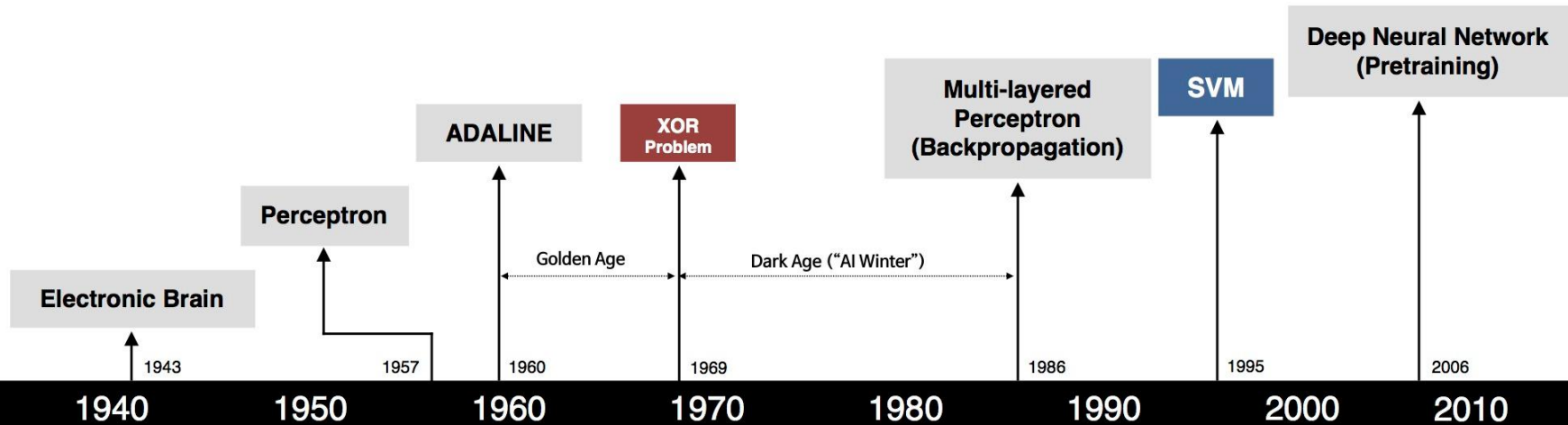
L'activation de l'axone dépend de la proportion des signaux excitateurs et inhibiteurs



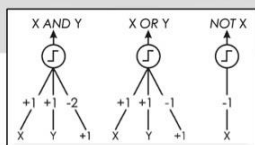
**Synapse** : zone de contact entre l'axone d'un neurone et une dendrite d'un autre neurone



# Les grandes étapes



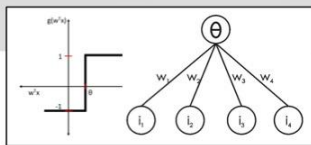
S. McCulloch - W. Pitts



- Adjustable Weights
- Weights are not Learned



F. Rosenblatt



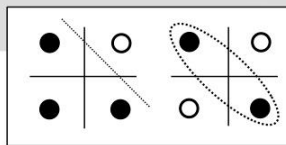
- Learnable Weights and Threshold



B. Widrow - M. Hoff



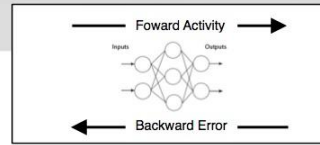
M. Minsky - S. Papert



- XOR Problem



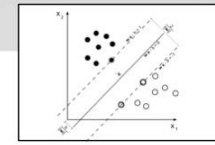
D. Rumelhart - G. Hinton - R. Williams



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



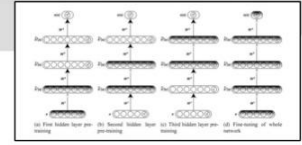
V. Vapnik - C. Cortes



- Limitations of learning prior knowledge
- Kernel function: Human Intervention



G. Hinton - S. Ruslan



- Hierarchical feature Learning



# Années 40 et 50

- 1943 : Premiers circuits électroniques cherchant à simuler un neurone (« en dur »).
- 50's :
  - 1950 : Article fondateur de Turing : « Computing Machinery and Intelligence ». Test de Turing.
  - 1957 : Le perceptron. Premier algorithme d'apprentissage:
  - Les valeurs  $W$  sont apprises

$i_1, \dots, i_n$  : inputs

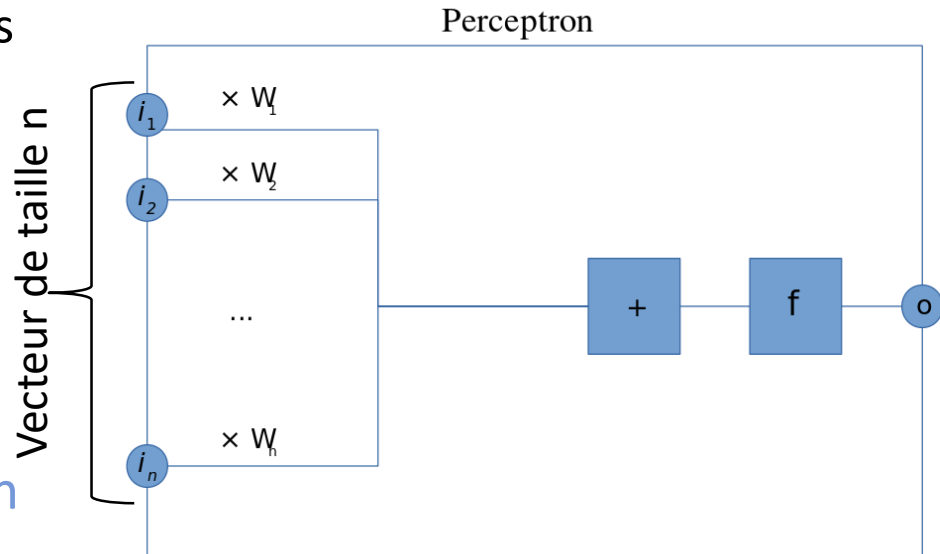
$W_1, \dots, W_n$  : weights

$o$  : sortie attendue

$f(x) = 1$  si  $x > 0$ ,

0 sinon

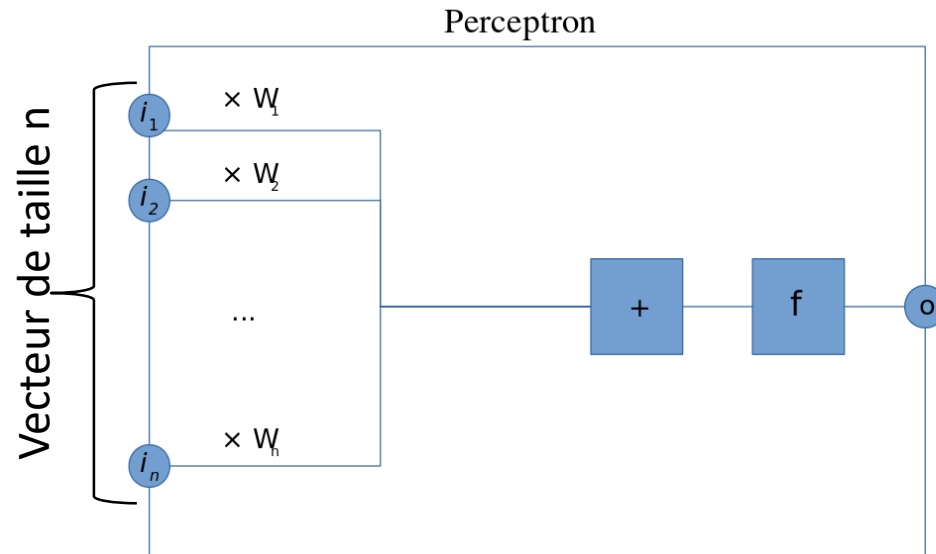
$f$  est la **fonction d'activation**



$$o = f\left(\sum_{k=1}^n i_k \cdot W_k\right)$$



# Une ligne d'apprentissage



Attendu  $d = 0$

Obtenu  $y = 1$

La somme obtenue est trop grande, il faut réduire les poids  $w$

*On suppose que les valeurs  $i$  sont entre 0 et 1*

$$W_1(t+1) = W(t) + r(d-y) i_1 = W_1(t) - r \cdot i_1$$

$i_1 = 0$  : Pas de changement.

$i_1 = 1$  :  $r$  est retranché au poids.

NB: On néglige l'ajout de biais pour le moment

# Années 60 et 70

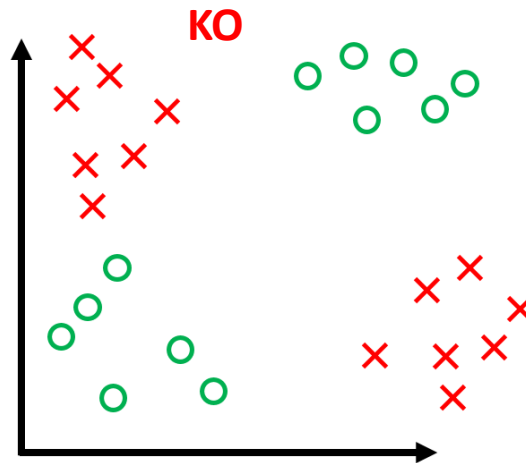
- 1960 : Adaline (ADaptive LInear Neuron)

On remplace  $(d-y)$  dans le Perceptron par  $S - \sum w_i(t) \cdot x_{ji}$  où  $S$  est la valeur correcte pour la somme.

⇒ **Premier Age d'or**

- 1969 : XOR Problème démontré insoluble. => **AI Winter**

*En langage moderne, le perceptron est un classificateur linéaire.*



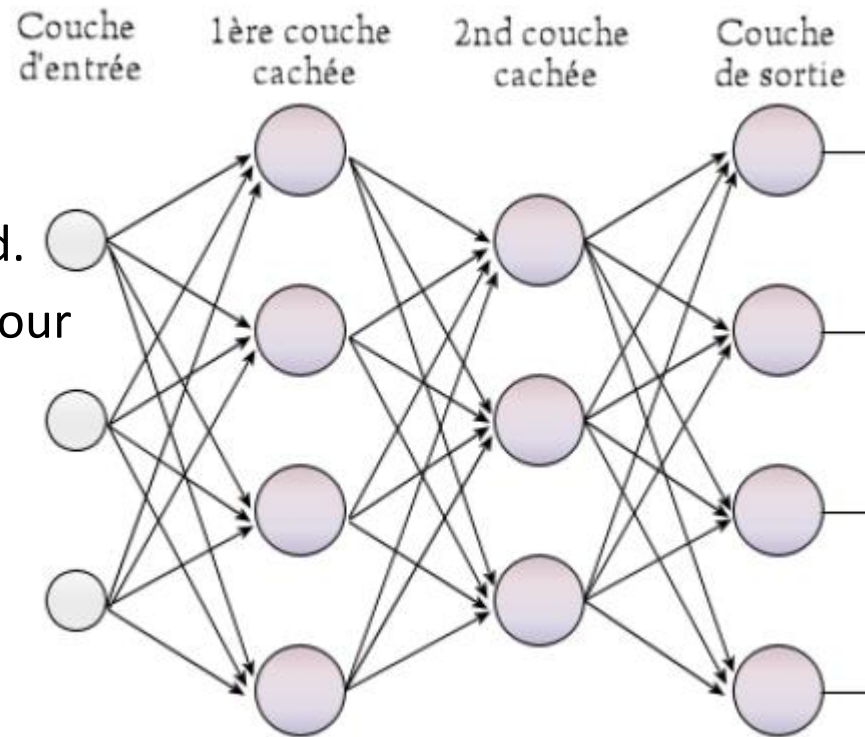
XOR		
A	B	$R = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

# Années 80 et 90



- **1986 : MultiLayers Perceptron et BackPropagation**

- Assemblage de perceptrons
- La fonction  $f$  évolue vers le sigmoïd.
- L'algo de backpropagation utilisé pour l'apprentissage.
- Début des réseaux convolutifs (reconnaissance d'images)



**Universal approximation theorem :** Tout problème (y compris XOR) peut être représenté par un perceptron multicouches.

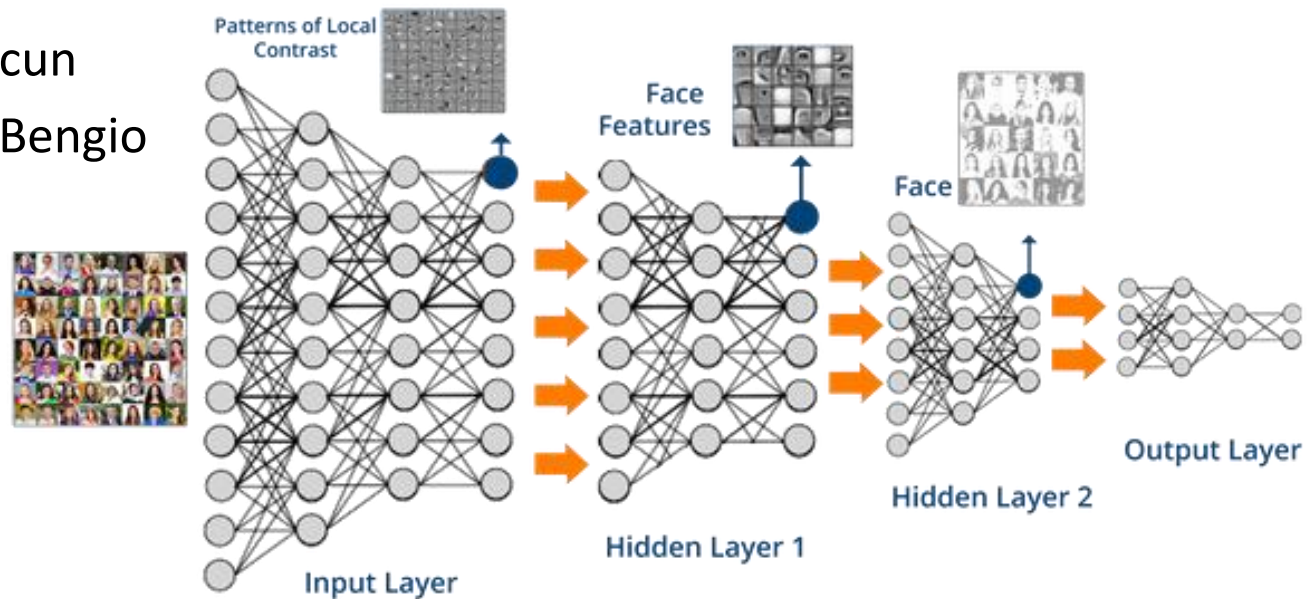
(En 1999 : Equivalence avec une machine de Turing Universel.)

Reste marginal. Supplanté par SVM (Support Vector Machine 1995)



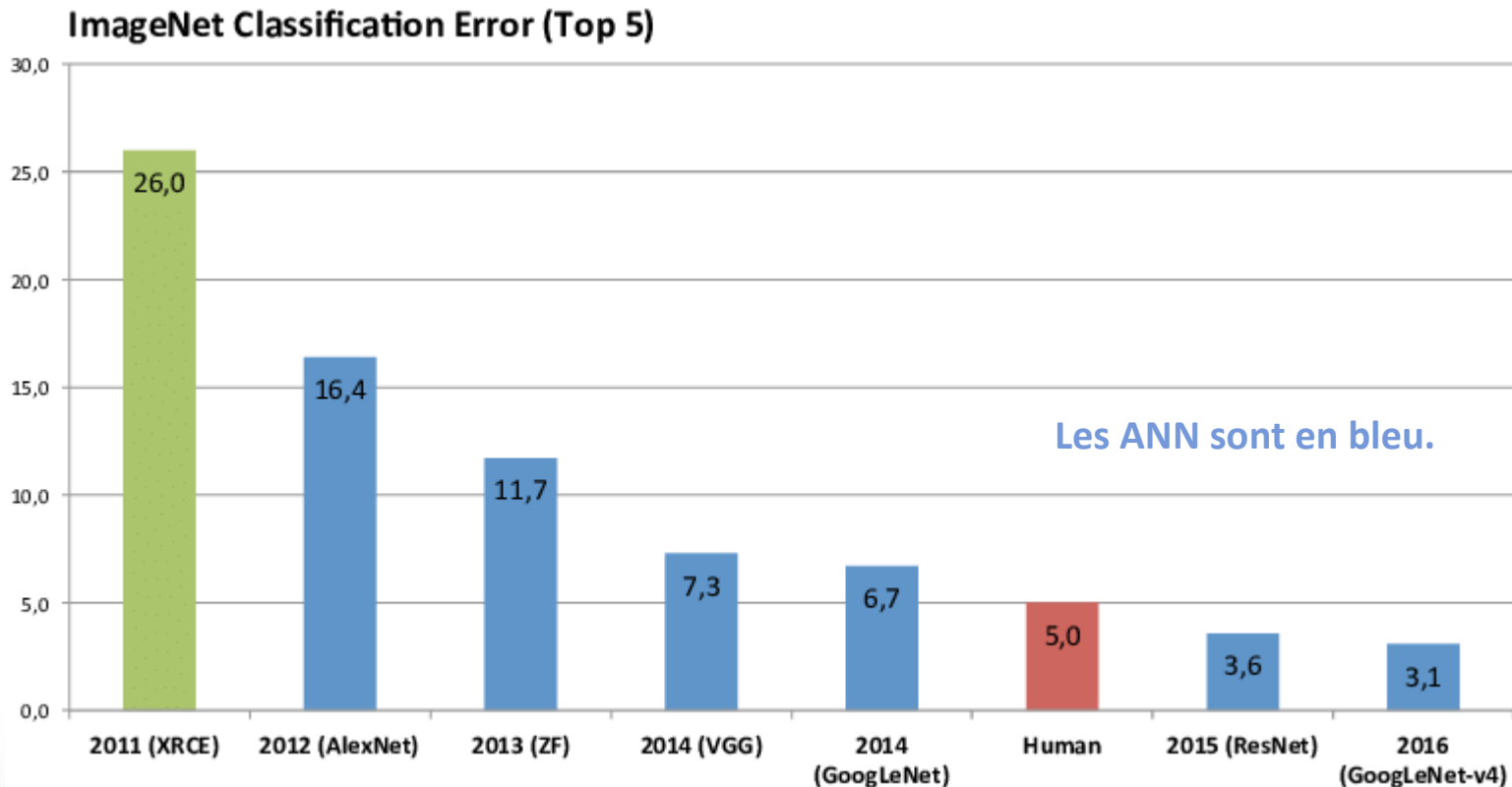
# Années 2000

- 2006 : Apparition du concept de deep Learning
- Progression continue, mais la technique reste marginale.  
“Neural networks are the second best way to do almost anything”
- La flamme est maintenue par the « Deep Learning Conspiracy »:
  - Geoff Hinton
  - Yann Lecun
  - Yoshua Bengio



# 2012 : Le Choc

- ImageNet Large Scale Visual Recognition Challenge est une compétition annuelle de reconnaissance d'image basée sur 10 millions d'images annotées.



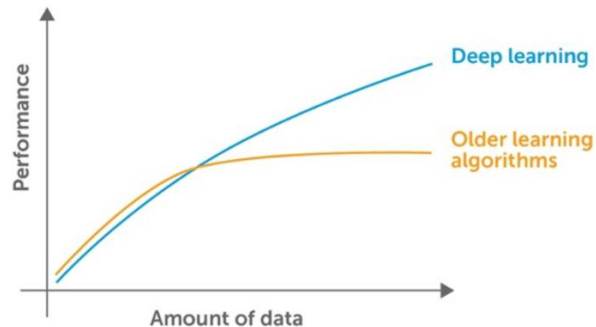


# Les raisons de l'émergence du Deep Learning



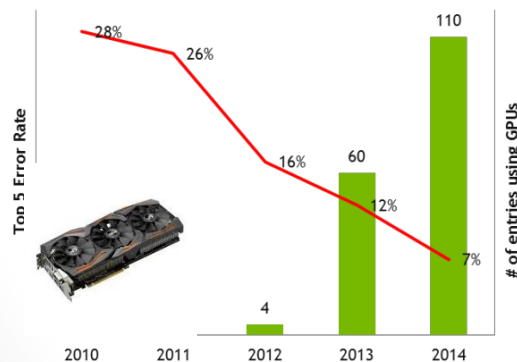
## Deep Learning Hypothesis

“Le succès du Deep Learning est largement un succès de l'Ingénierie”

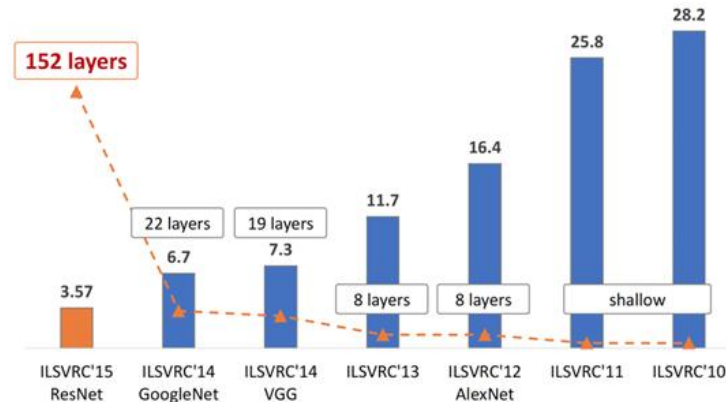


Taille des jeux de données (Big Data).

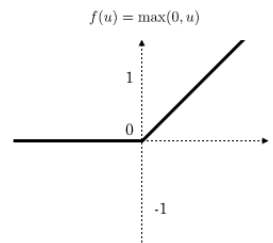
IMAGENET



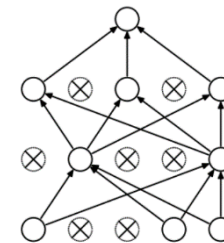
Parallélisation massive (GPU).



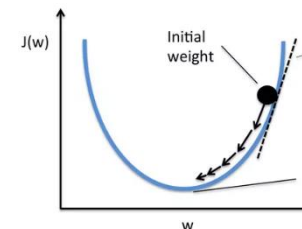
Amélioration de l'Architecture.



Fonction d'activation



Régularisation



Descente de Gradient

Améliorations algorithmiques

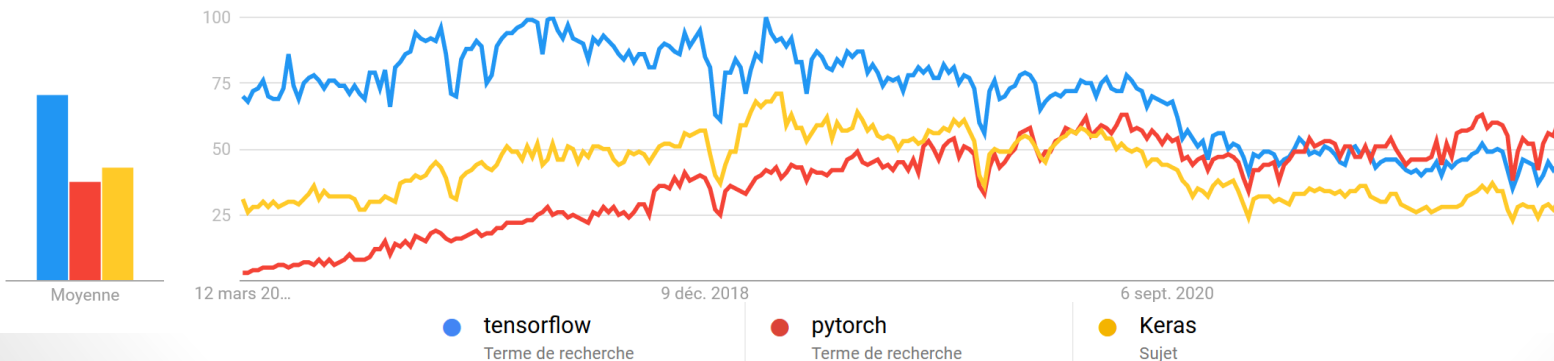
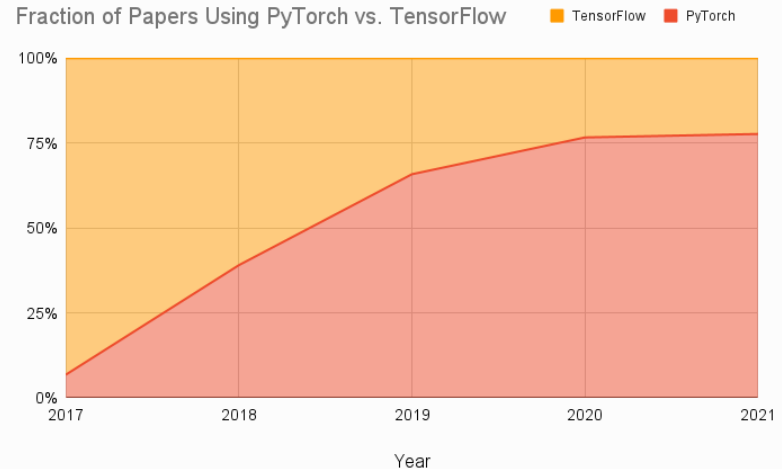
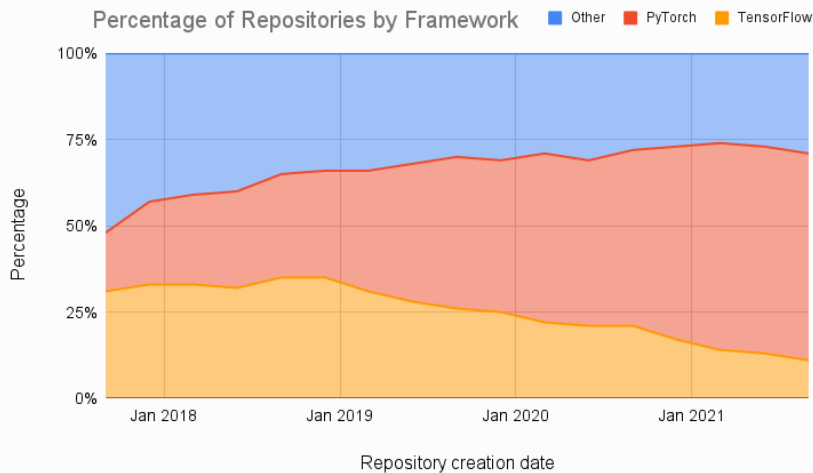
# Librairies de Deep Learning



2 Framework principaux:

- TensorFlow de Google AI
  - PyTorch de Facebook AI
- ➔ Choix de Pytorch.

Convergence des deux Frameworks.

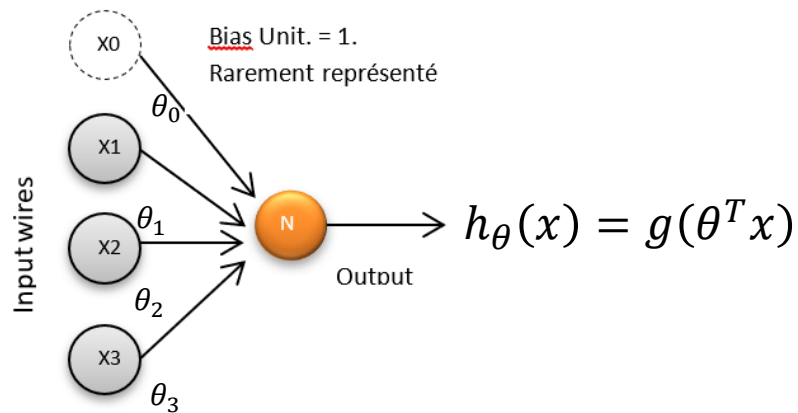


### III. ARCHITECTURE DU RÉSEAU DE NEURONE



# Réseaux de Neurone

- Le réseau de neurone est un cas particulier de fonction hypothèse.
- La fonction de coût peut être variable.

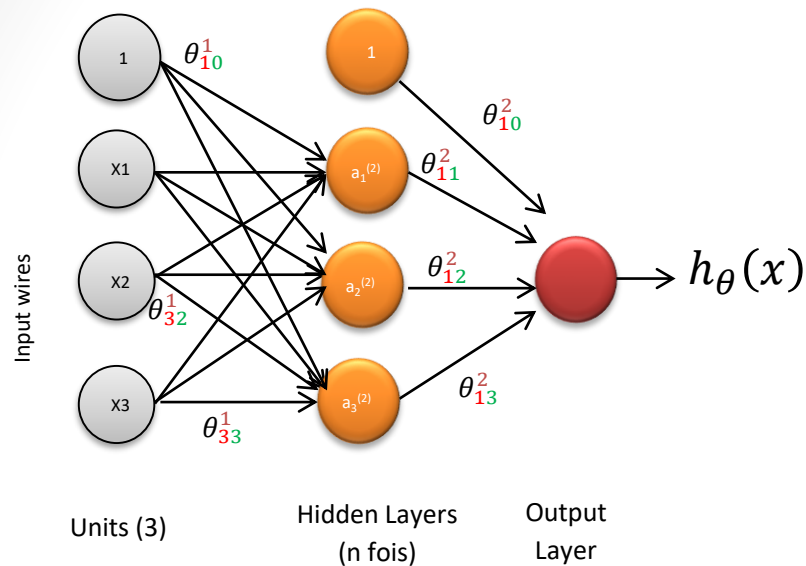


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

- $g()$  est la fonction d'activation.



# h() Réseaux de Neurone



$\theta_{ij}^l$  couche  
Neurone d'arrivée Neurone de départ

$$\begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} = g(\Theta^{(1)} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}) = g\left(\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}\right)$$

$Dim \Theta^{(1)} = 3 \times 4$

1

$$h_{\theta}(x) = g(\Theta^{(2)} * \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}) = g([z^{(3)}])$$

$$Dim \Theta^{(2)} = 1 \times 4$$

2

[ 23 ]



# Back Propagation

- Voici un exemple de fonction de coût d'un réseau de neurone (pour une classification 0/1) :

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \ln \left( h_{\theta}(x^{(i)}) \right)_k + \left( 1 - y_k^{(i)} \right) \ln \left( 1 - \left( h_{\theta}(x^{(i)}) \right)_k \right) \right]$$

- Pour appliquer l'algorithme de descente de gradient, nous allons calculer les dérivées partielles :  $\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}}$
- L'algorithme qui permet de calculer cette dérivée partielle est appelée **Back Propagation**.
- Elle porte ce nom, car l'algorithme part de l'output layer pour remonter jusqu'à l'input layer.

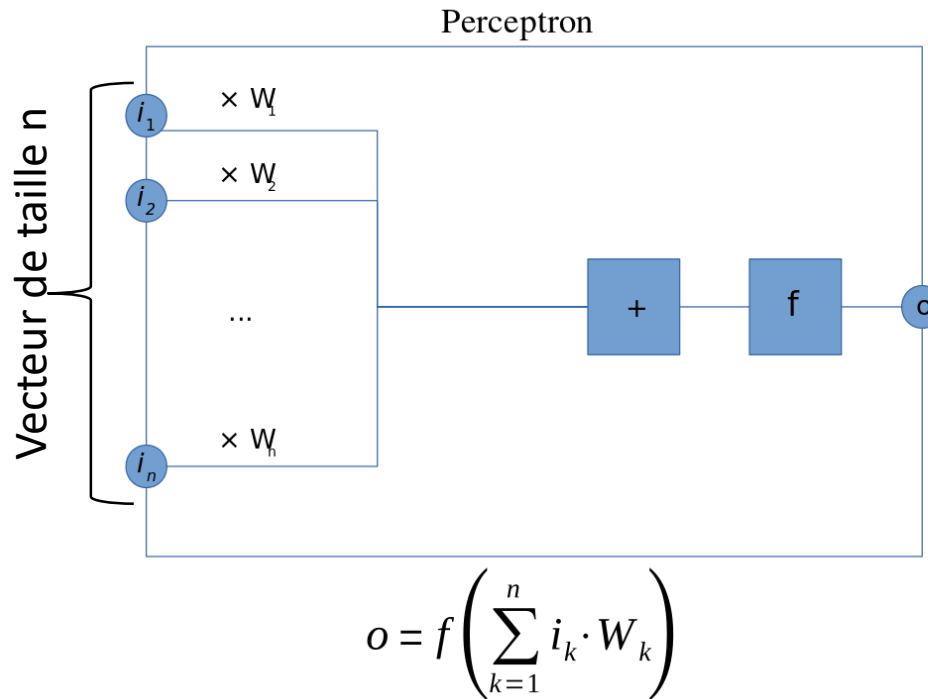


<https://youtu.be/OgSA7liZMXI>



# Fonction d'activation

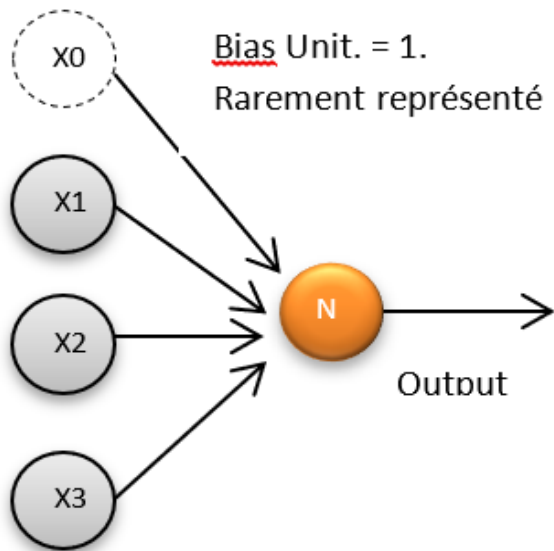
- $f$  est la fonction d'activation.
- Doit être continues/ dérivable.





# Fonction d'activation

- Exemple d'unité logistique : Perceptron + biais + sigmoïd



$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

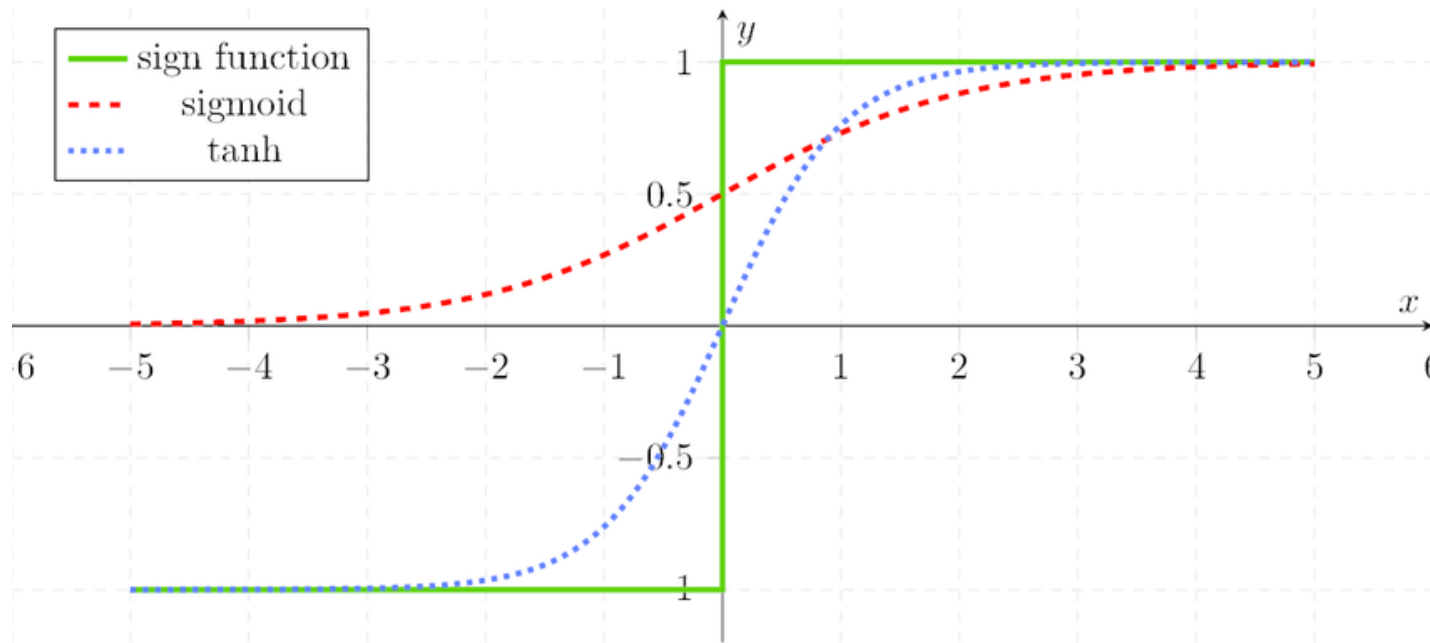
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$



# Fonctions d'activation classiques



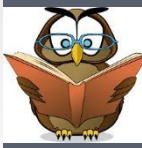
- sign est quasiment la fonction du perceptron.
- sigmoid et tanh permettent de passer d'une valeur continue à une valeur 0/1 ou -1/1



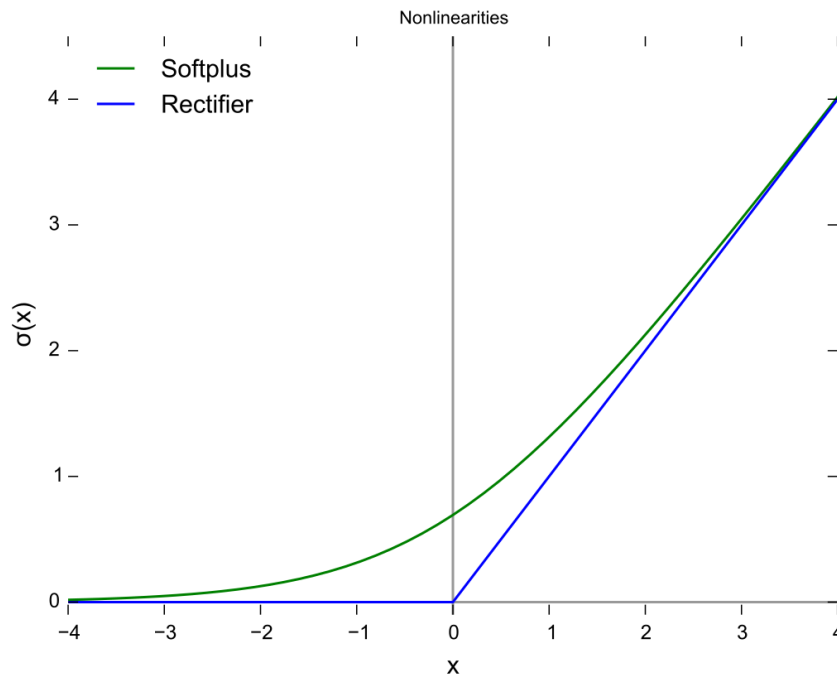
$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

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

# La Fonction Relu



- Rectified Linear Unit (appelé aussi « **Rectifier** »).
- La fonction RELU a permis l'efficacité du Deep Learning en allégeant le « **Vanishing gradient problem** » : Les poids des premières couches évoluent beaucoup trop lentement pendant l'apprentissage.



$$\begin{aligned} \text{relu}(z) &= 0 \text{ si } z < 0 \\ &= z \text{ si } z > 0 \end{aligned}$$

$$\text{softplus}(z) = \ln(1 + e^z)$$

$$\frac{d(\text{softplus}(z))}{dz} = \text{sigmoid}(z)$$

- La librairie doit gérer la non-différenciabilité de relu en 0.
- Softplus permet d'éviter ce problème.
- Relu est aujourd'hui la fonction d'activation de loin la plus utilisée.