IFT 615 – Intelligence Artificielle

Réseau de neurones artificielle

Professeur: Froduald Kabanza
Assistants: D’Jeff Nkashama

Université de Sherbrooke

Sujets couverts

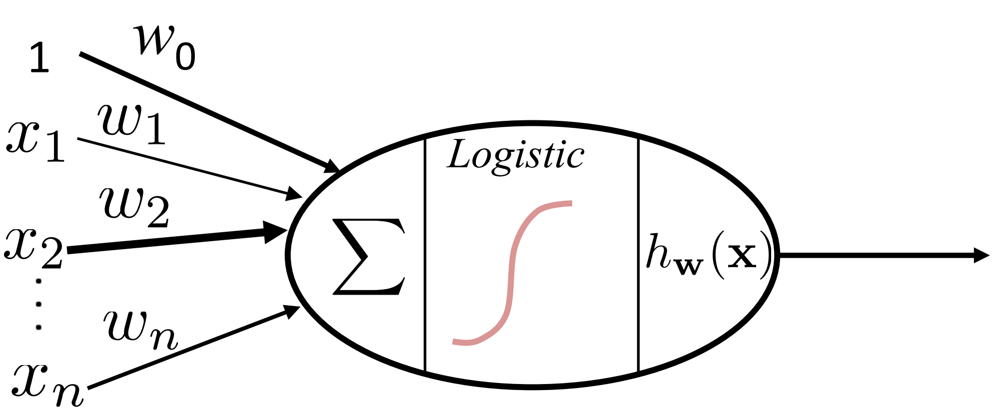
- Régression logistique
- Réseau de neurones artificiel
- Rétropropagation des gradients

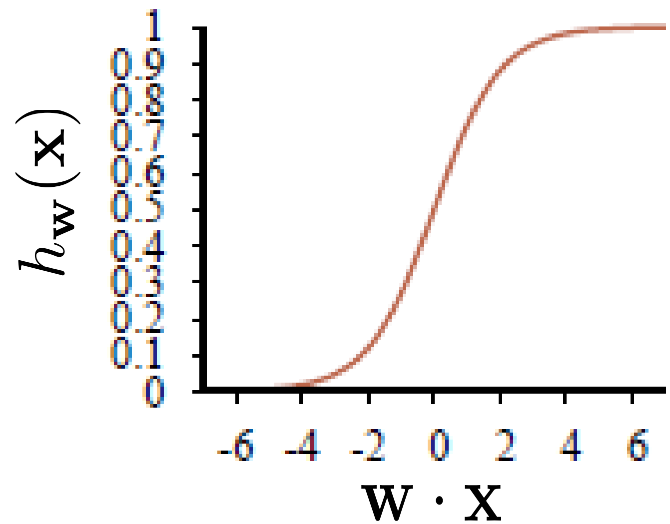
Troisième algorithme: régression logistique

- **Idée**: plutôt que de prédire une classe, prédire une probabilité d'appartenir à la classe 1 (ou la classe 0, ça marche aussi)

$$p(y=1|x)=h_w(x)=\sigma(w.x)= \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Pour choisir une classe, prendre la plus probable selon le modèle
  - si hw(x)≥ 0.5 choisir la classe 1
  - sinon, choisir la classe 0





Dérivation de la règle d'apprentissage

- Deux choix

a) $Loss(y_t, hw(x_t)) = (y - h_w(x_t))^2$

Voir livre Section 19.6.5

b) $Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) = -y_t \log h_{\mathbf{w}}(\mathbf{x}_t) - (1 - y_t) \log(1 - h_{\mathbf{w}}(\mathbf{x}_t))$

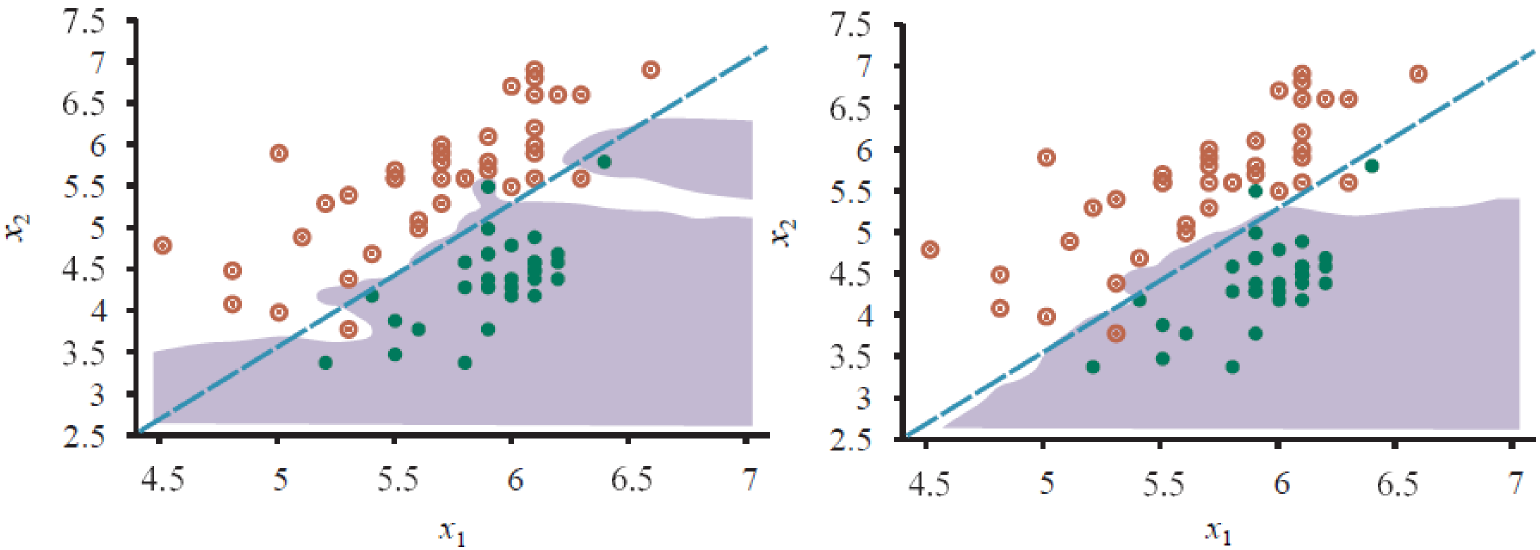
Voir capsule d'Hugo Larochelle sur la regression logistique

- Les deux mènent à même règle que pour le Perceptron, mais la définition de $h_w(x_t)$ est différente

$$w_i \leftarrow w_i + \alpha(y_t - h_{\mathbf{w}}(\mathbf{x}_t))x_{t,i} \quad \forall i$$

Limitation des classifieurs linéaires

- Si les données d'entraînement sont séparables linéairement, le Perceptron et la régression logistique vont trouver cette séparation



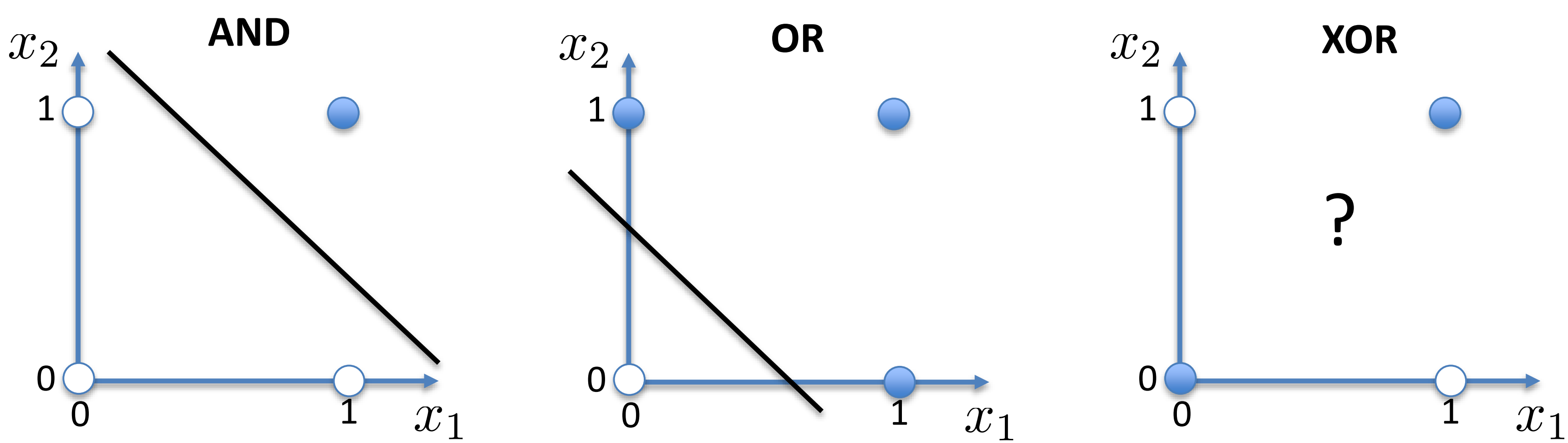
$(k = 1)$ $(k = 5)$

prédiction de classe par k plus proche voisins

- *k* plus proche voisins est non-linéaire, mais coûteux en mémoire et temps de calcul (pas approprié pour des problèmes avec beaucoup de données)

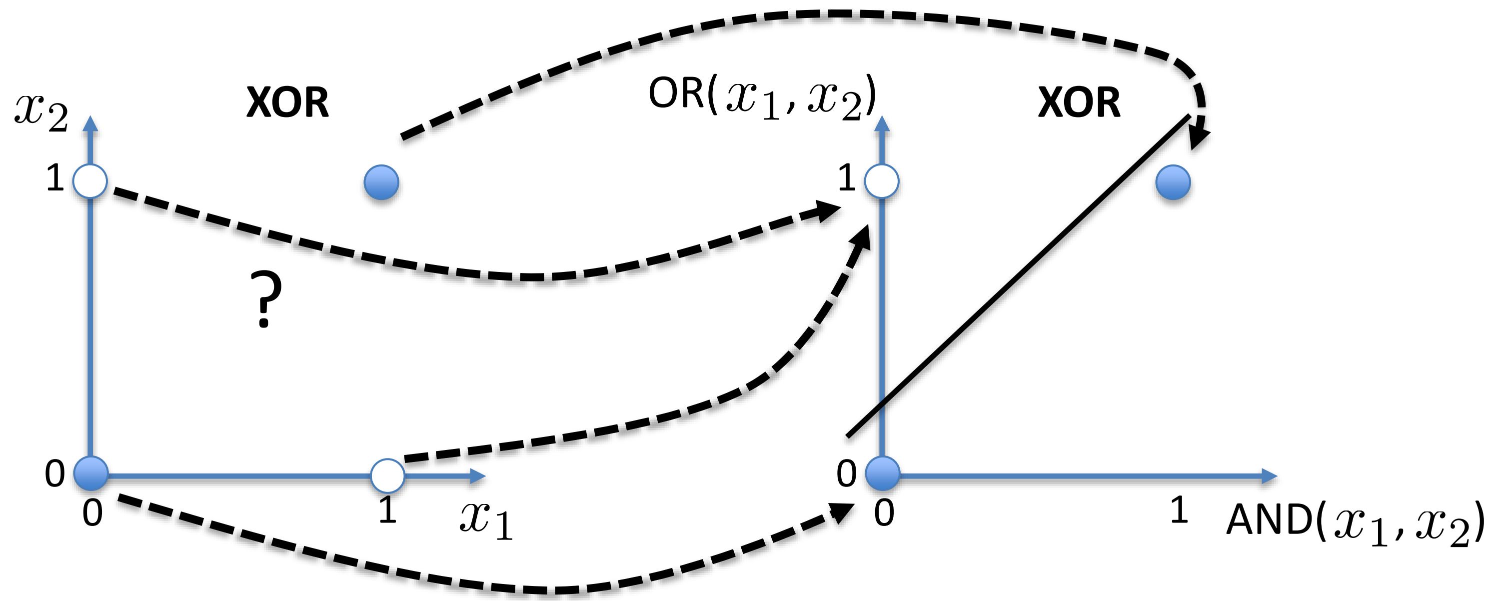
Limitation des classifieurs linéaires

- Cependant, la majorité des problèmes de classification ne sont pas linéaires
- En fait, un classifieur linéaire ne peut même pas apprendre XOR!



Limitation des classifieurs linéaires

- Par contre, on pourrait transformer l'entrée de façon à rendre le problème linéairement séparable sous cette nouvelle représentation
- Dans le cas de XOR, on pourrait remplacer
  - x_1 par AND(x_1 ,x_2) et
  - x_2 par OR(x_1, x_2)

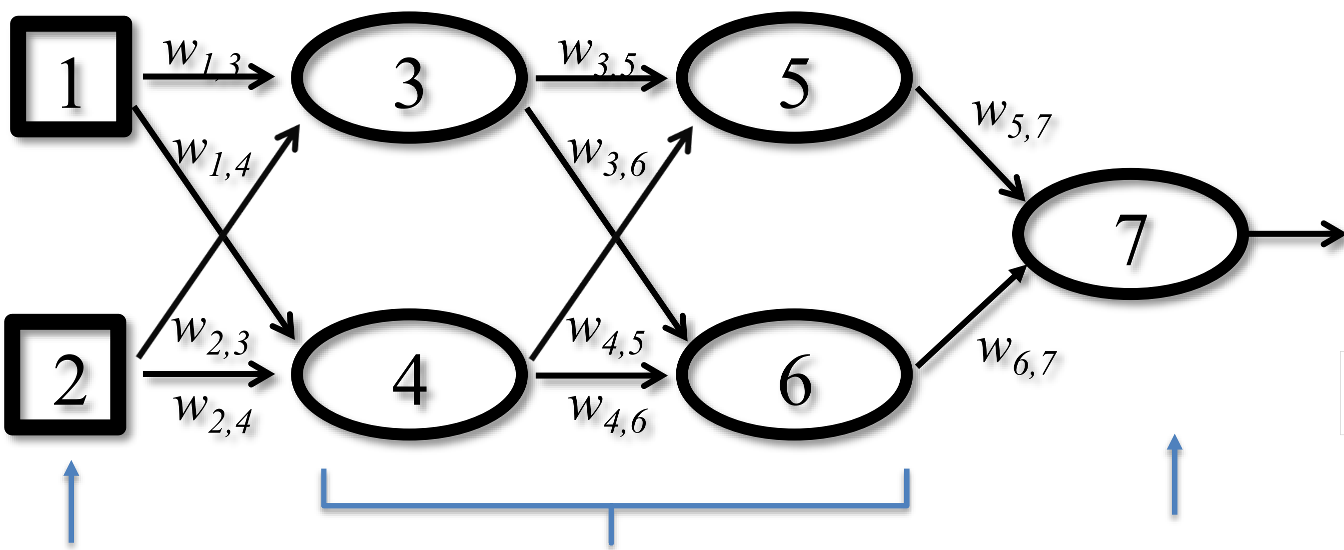


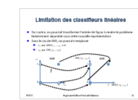
Introduction

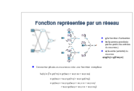
RÉSEAU DE NEURONES ARTIFICIEL

Quatrième algorithme: réseau de neurones artificiel

- **Idée**: apprendre les poids du classifieur linéaire **et** une transformation qui va rendre le problème linéairement séparable

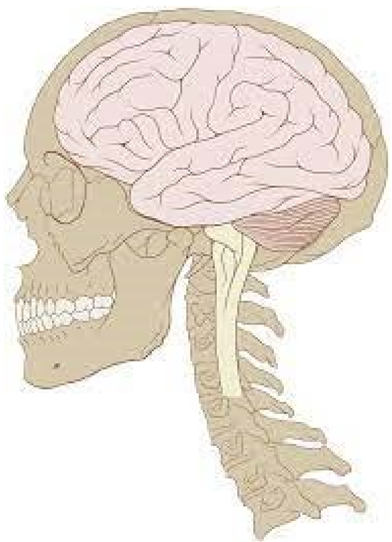


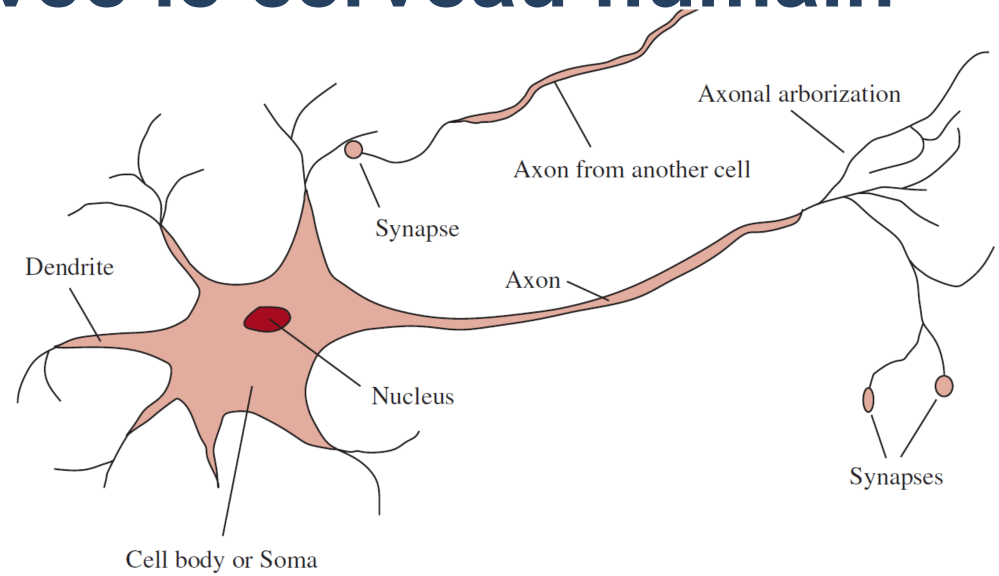


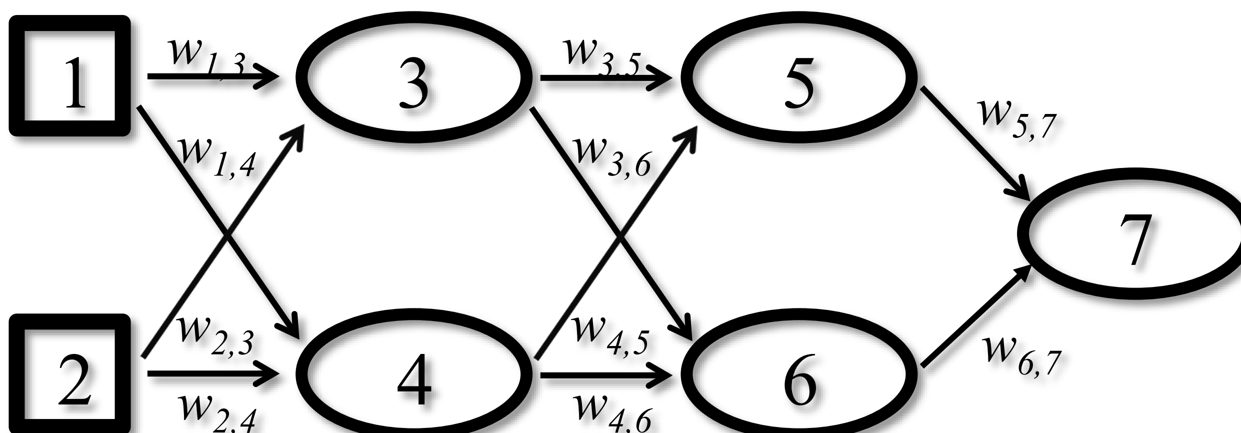


Entrées Couche cachée Couche de sortie

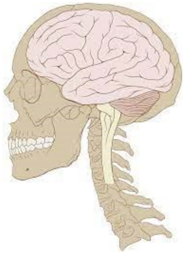
Analogie avec le cerveau humain







Analogie avec le cerveau humain

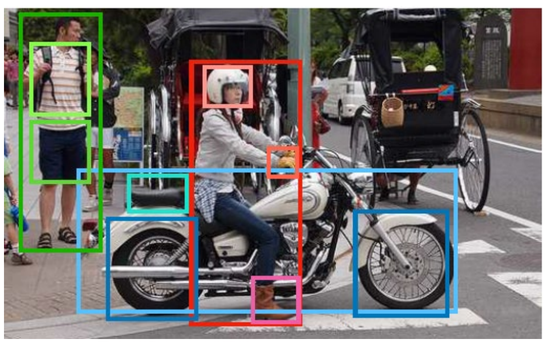
  

	Supercomputer	Personal Computer	Human Brain
Computational units	10^6 GPUs + CPUs	8 CPU cores	10^6 columns
	10^{15} transistors	10^{10} transistors	10^{11} neurons
Storage units	10^{16} bytes RAM	10^{10} bytes RAM	10^{11} neurons
	10^{17} bytes disk	10^{12} bytes disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{18}	10^{10}	10^{17}

Tiré du livre, publié en 2020 – les chiffres ne sont plus à jour, mais donnent une idée

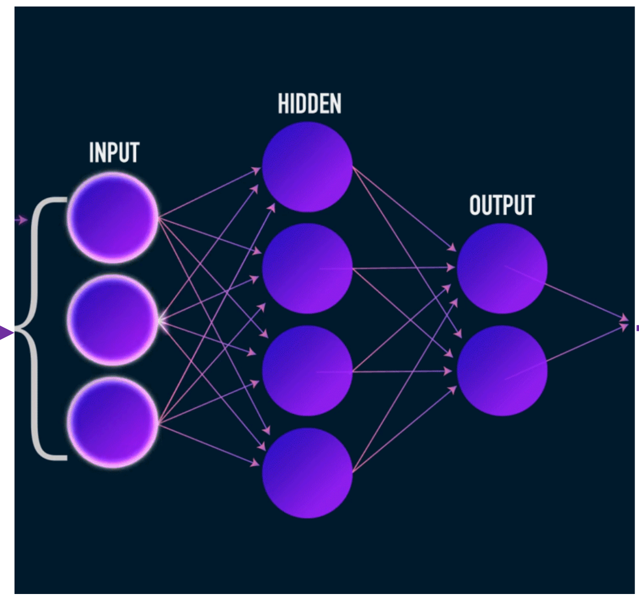
Comme un reseau de neurones fonctionne?

Video stream



Zellars et al., CVPR 2018

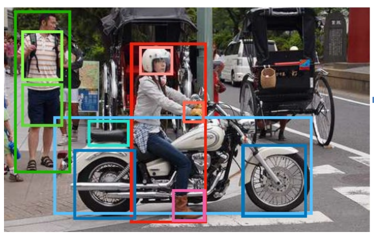
Data Processing



A person riding a bike

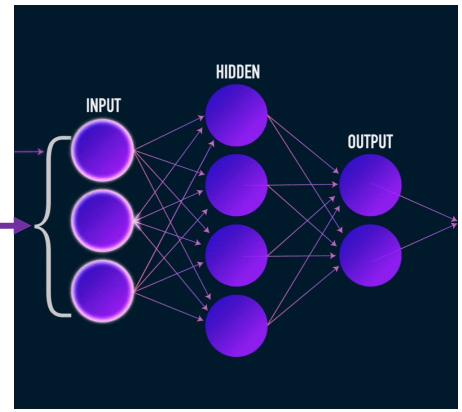
Comment un réseau de neurones fonctionne?

Video stream

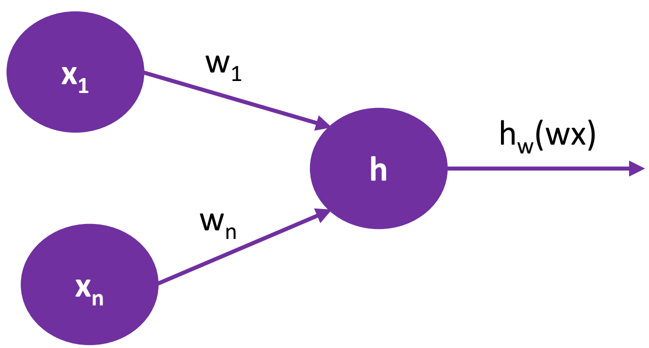


Zellars et al., CVPR 2018

Data Processing



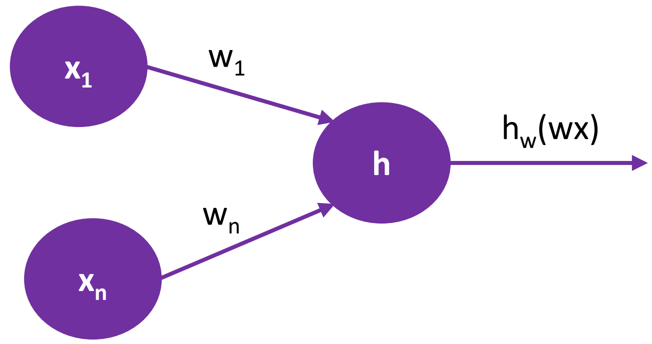
A person riding a bike

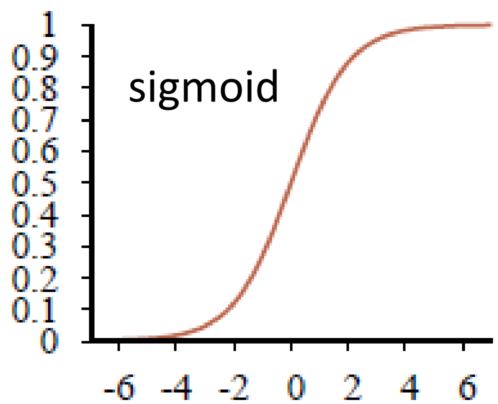


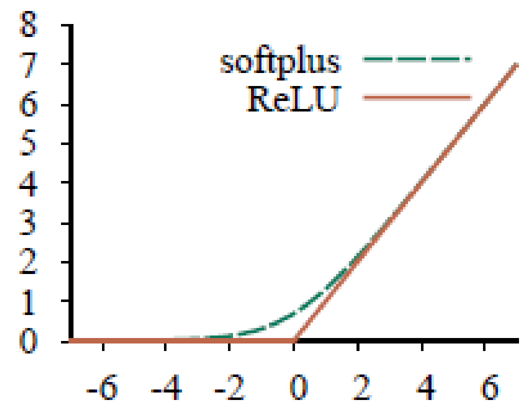
h est la function d'activation

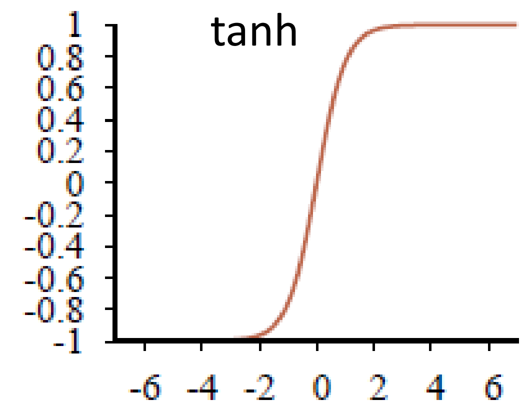
- Sigmoid
- ReLu
- Softplus
- Tanh

Fonctions d'activation









$$\sigma(w.x) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

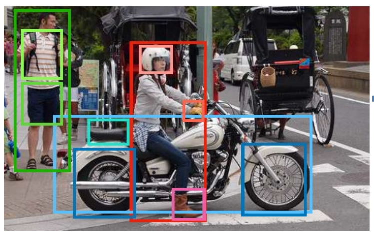
$$\text{ReLu}(w.x) = \max(0, w.x)$$

$$\text{softplus}(w.x) = \log(1+e^{w.x})$$

$$\tanh = \frac{e^{w.x} - 1}{e^{w.x} + 1}$$

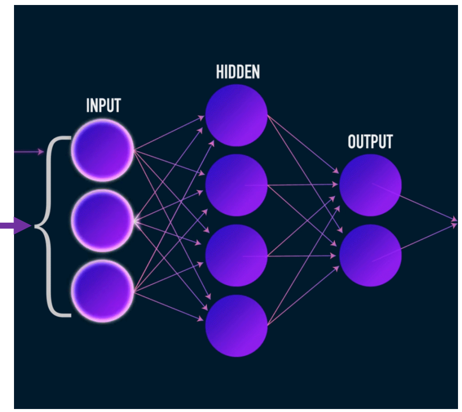
Comment un reseau de neurone apprend?

Video stream



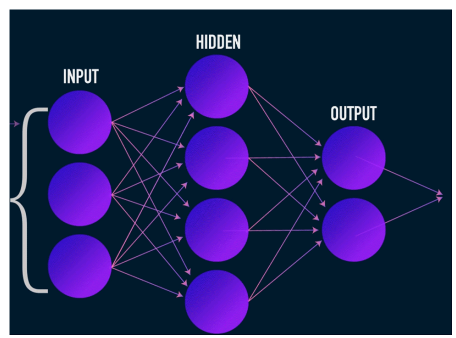
Zellars et al., CVPR 2018

Data Processing



A person riding a horse

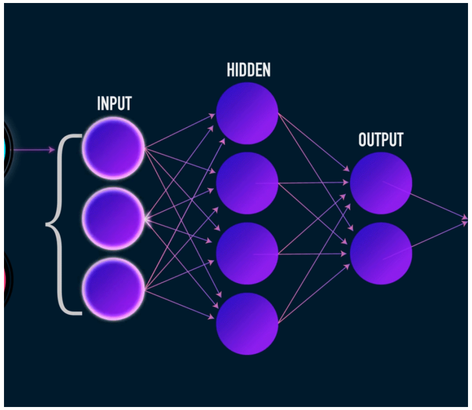
Expected*: A person riding a bike*



Back propagation

Plusieurs architectures pour différentes applications

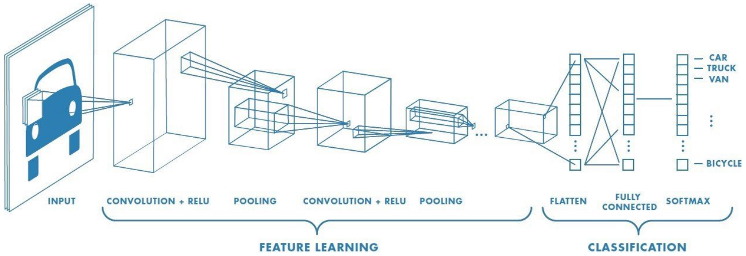
Multilayer Perceptron (MLP)



Plusieurs architectures pour différentes applications

Multilayer Perceptron (MLP)

Convolutional Neural Network (CNN)

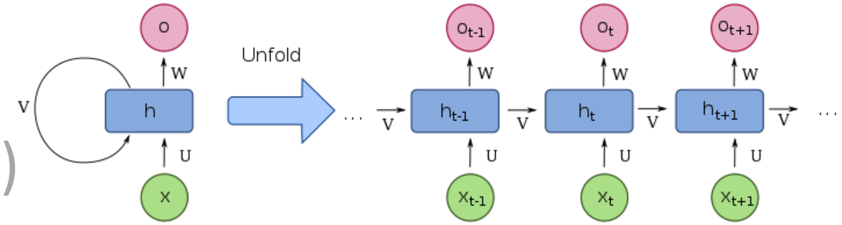


Plusieurs architectures pour différentes applications

Multilayer Perceptron (MLP)

Convolutional Neural Network (CNN)

Recurrent Neural Network (RNN)



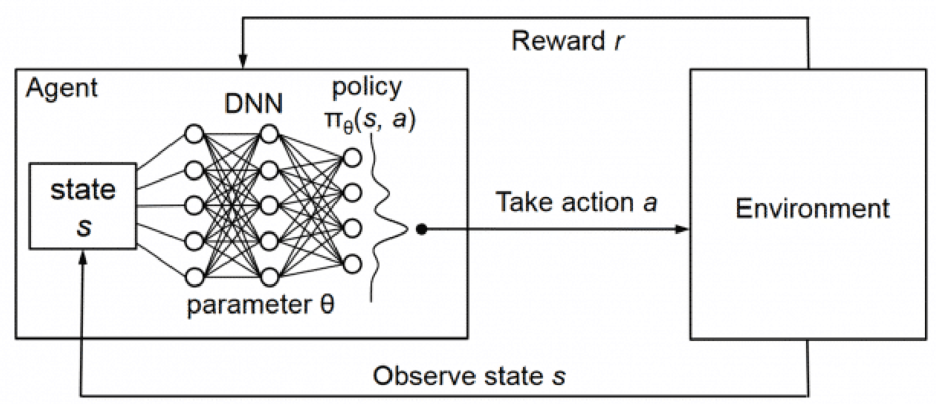
Plusieurs architectures pour différentes applications

Multilayer Perceptron (MLP)

Convolutional Neural Network (CNN)

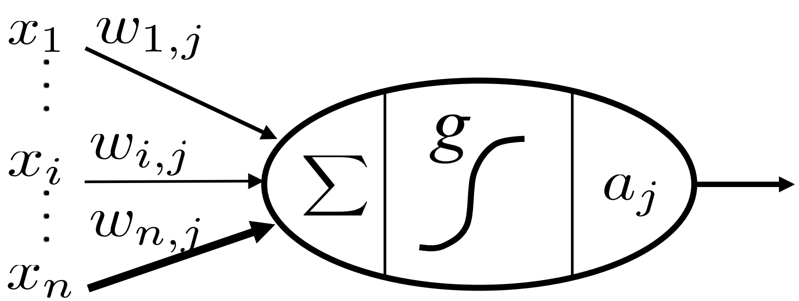
Recurrent Neural Network (RNN)

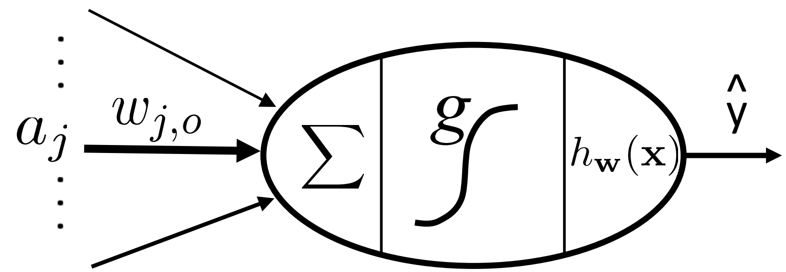
Deep Q-Learning (DQN)

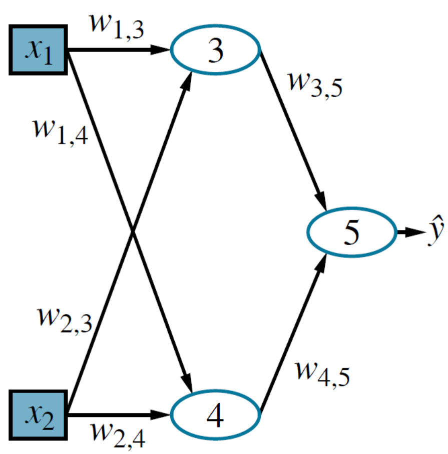


DÉRIVATION DE LA RÈGLE D'APPRENTISSAGE POUR LE PERCEPTRON MULTICOUCHES

Notations





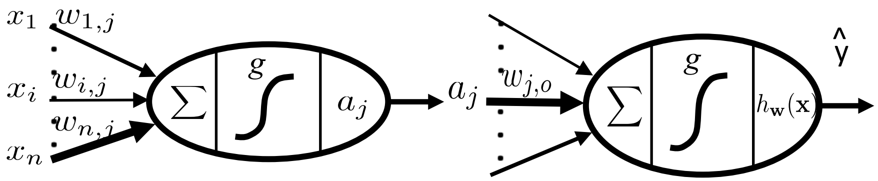


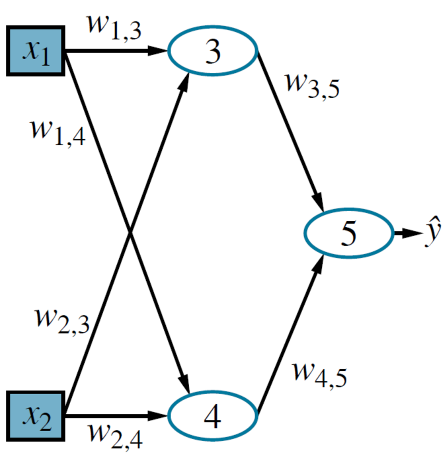
On note:

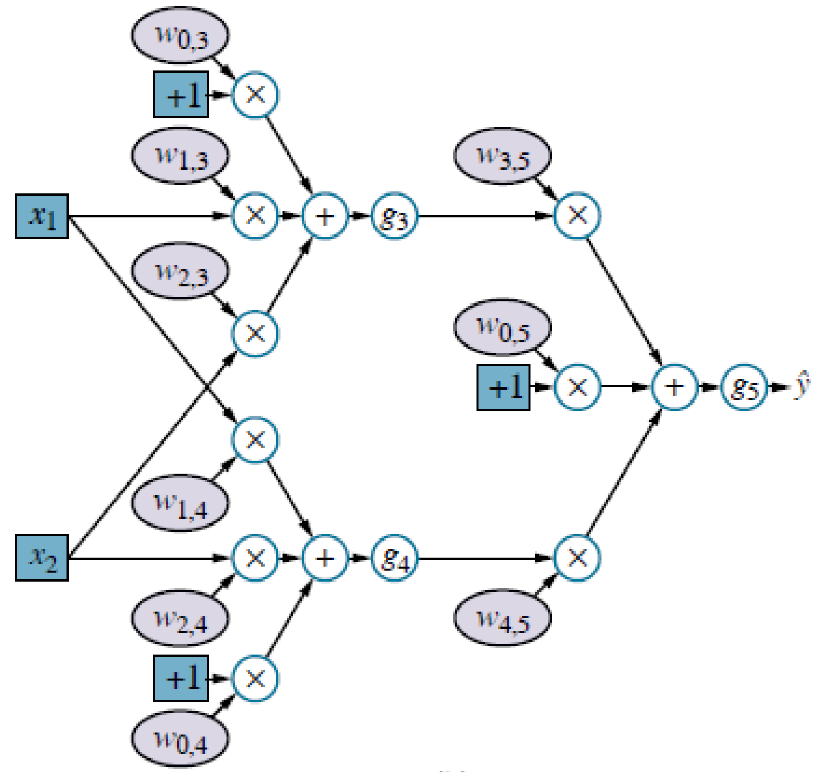
- ***g*** la fonction d'activation
- **in$_j$** la somme pondérée par les poids des entrées du neurone j
- ***a$_j$*** la sortie (activité) du neurone

On a donc $a_j = g(in_j) = g(\Sigma_i w_{i,j} a_i)$

Notations





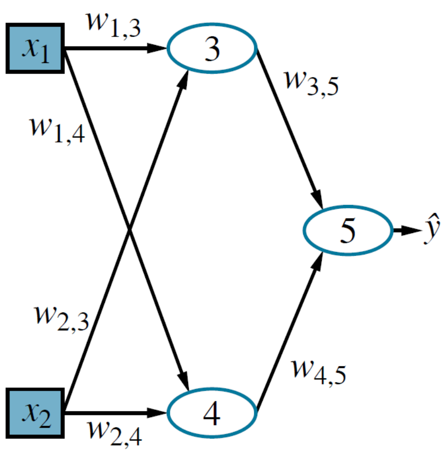


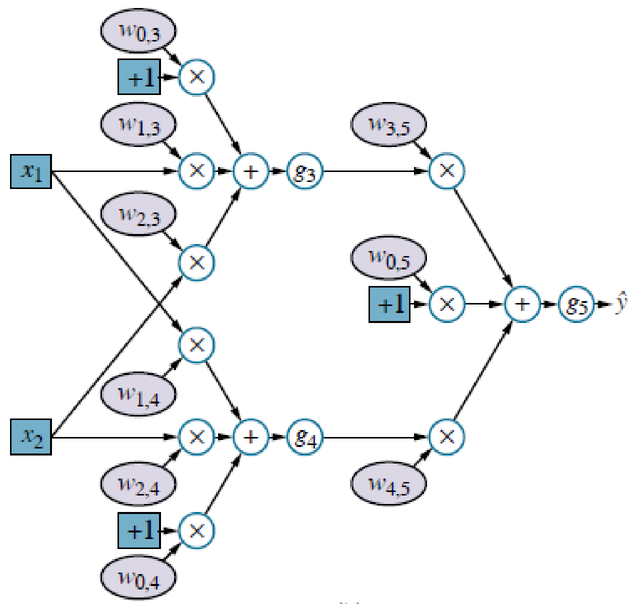
On note:

- ***g*** la fonction d'activation
- $\mathbf{in_j}$ la somme pondérée par les poids des entrées du neurone j
- $\boldsymbol{a_j}$ la sortie (activité) du neurone

On a donc $\boldsymbol{a_j = g(in_j) = g(\Sigma_i\, w_{i,j}\, a_i)}$

Fonction représentée par un réseau





- ***g*** la fonction d'activation
- **$\mathbf{in_j}$** la somme pondérée par les poids des entrées du neurone j
- ***a_j*** la sortie (activité) du neurone

$$a_j = g(in_j) = g(\Sigma_i w_{i,j} a_i)$$

- Connecter plusieurs neurones crée une fonction complexe

$$\begin{aligned}
h_w(x) = \hat{y} = g_5(in_5) &= g_5(w_{0,5} + w_{3,5}\, a_3 + w_{4,5}\, a_4) \\
&= g_5(w_{0,5} + w_{3,5}\, g_3(in_3) + w_{4,5}\, g_4(in_4)) \\
&= g_5(w_{0,5} + w_{3,5}\, g_3(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) \\
&\qquad + w_{4,5}\, g_4(w_{0,4} + w_{1,4}\, x_1 + w_{2,4}\, x_2))
\end{aligned}$$

Dérivation de la règle d'apprentissage

- La dérivation de la règle d'apprentissage se fait encore avec les gradients

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i, j$$

- Par l'application de la dérivée en chaîne, on peut décomposer cette règle d'apprentissage comme suit:

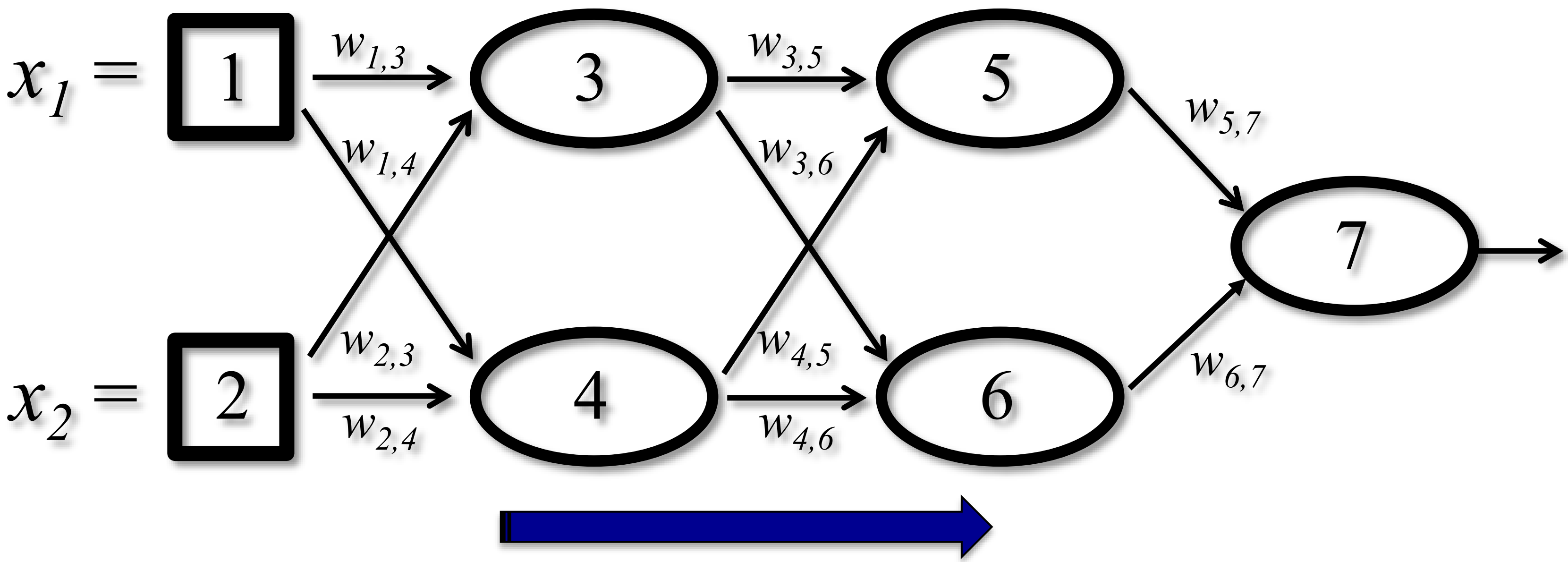
$$w_{i,j} \leftarrow w_{i,j} - \alpha \underbrace{\underbrace{\frac{\partial}{\partial a_j} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))}_{\text{gradient du coût p/r au neurone}} \underbrace{\frac{\partial}{\partial in_j} g(in_j)}_{\text{gradient du neurone p/r à la somme des entrées}}}_{-\Delta[j]} \underbrace{\underbrace{\frac{\partial}{\partial w_{i,j}} in_j}_{\text{gradient de la somme p/r au poids } w_{i,j}}}_{a_i}$$

- Donc la règle de mise à jour peut être écrite comme suite:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \ a_i \ \Delta[j]$$

Visualisation de la rétropropagation

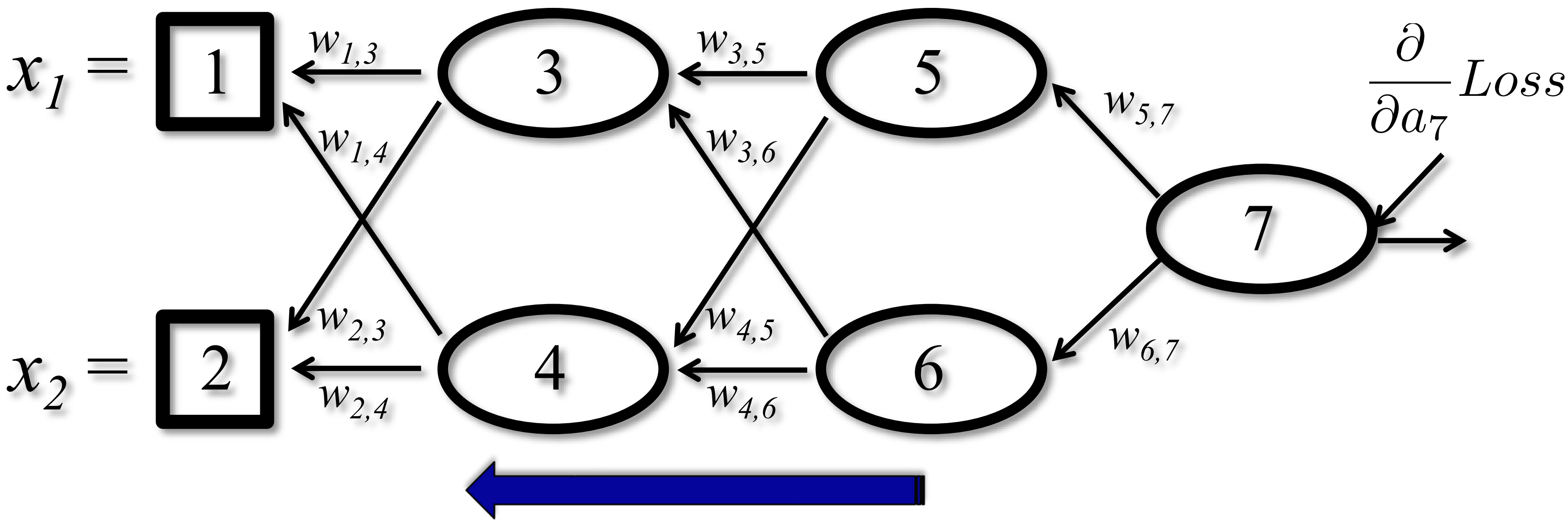
- L'algorithme d'apprentissage commence par une **propagation avant**



$$a_k = g\left(\sum_j w_{j,k} a_j\right)$$

Visualisation de la rétropropagation

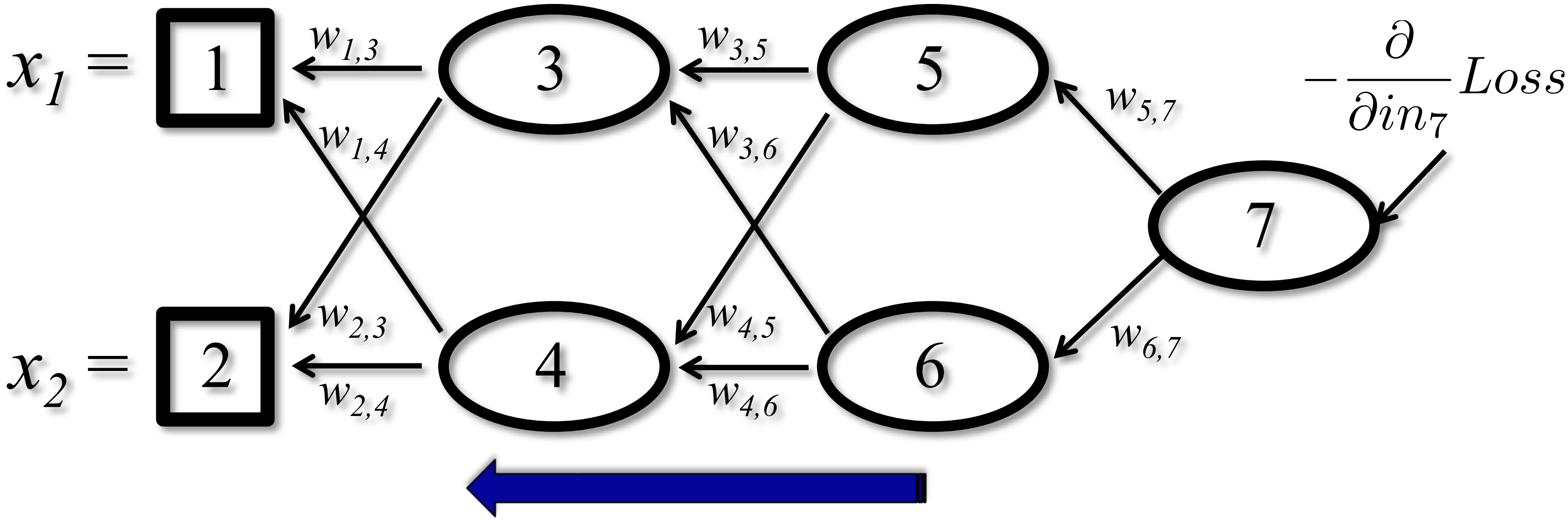
- Ensuite, le gradient sur la sortie est calculé, et le gradient rétropropagé



$$\frac{\partial}{\partial a_j} Loss = \sum_k \frac{\partial}{\partial a_k} Loss \; \frac{\partial}{\partial a_j} a_k$$

Visualisation de la rétropropagation

- Peut propager $-\frac{\partial}{\partial in_j} Loss = \Delta[j]$

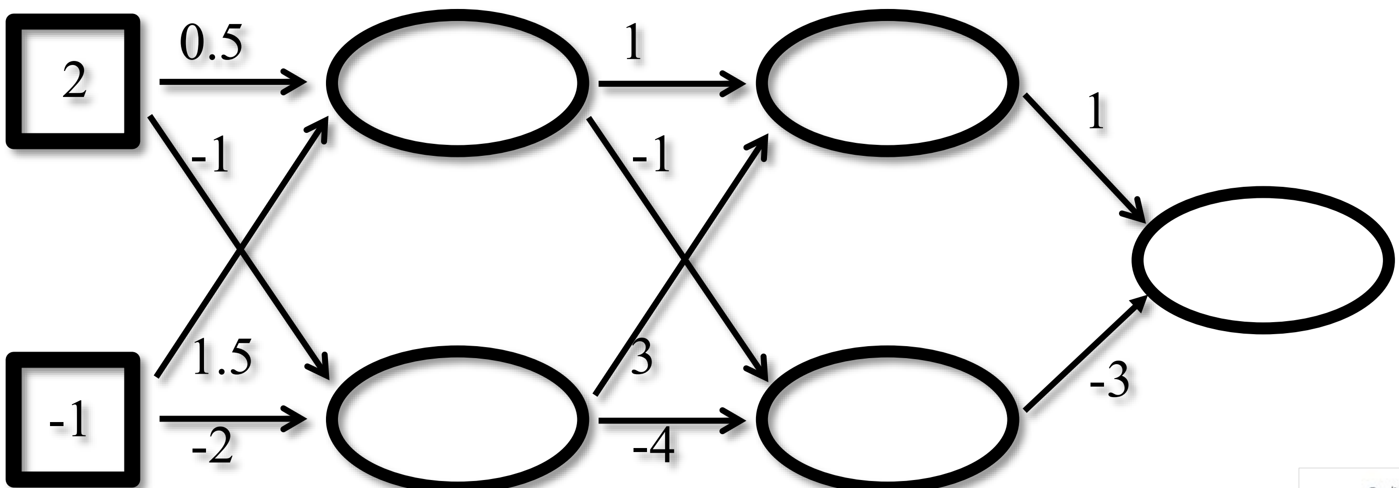


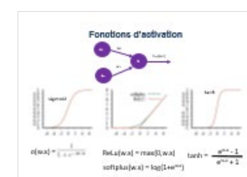
$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Exemple

$$\sigma(\mathbf{w}.\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$





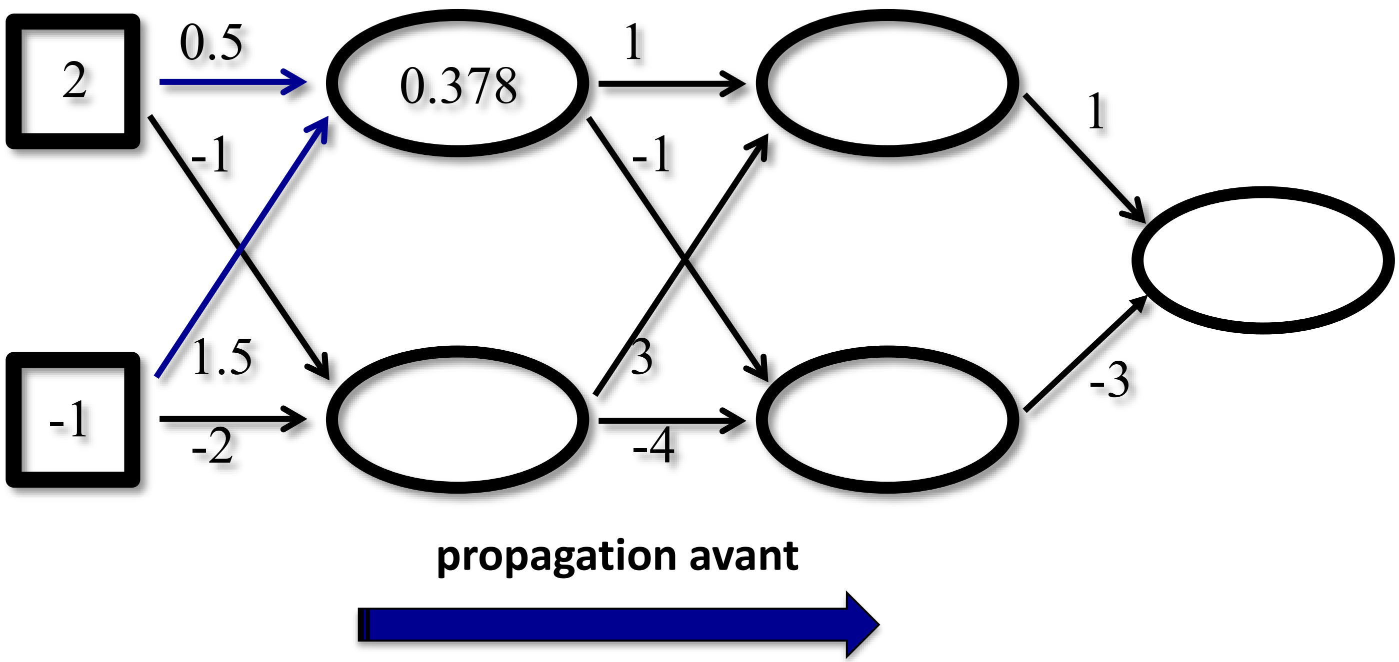
propagation avant

$$a_k = g\left(\sum_j w_{j,k} a_j\right)$$

Exemple

- Exemple: $\mathbf{x} = [2, -1]\,,\; y = 1$

$$\sigma(\mathrm{w.x}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

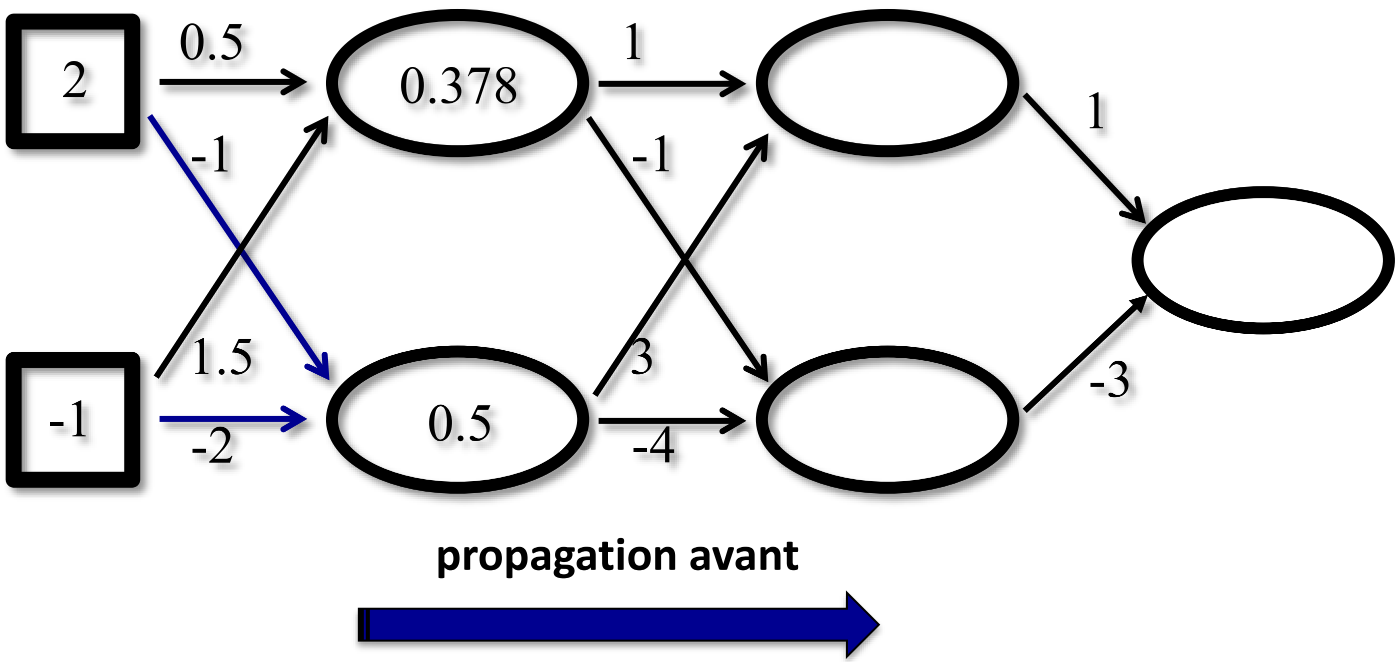


$$\sigma(0.5 * 2 + 1.5 * -1) = \sigma(-0.5) = 0.378$$

Exemple

$$\sigma(\text{w.x}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$

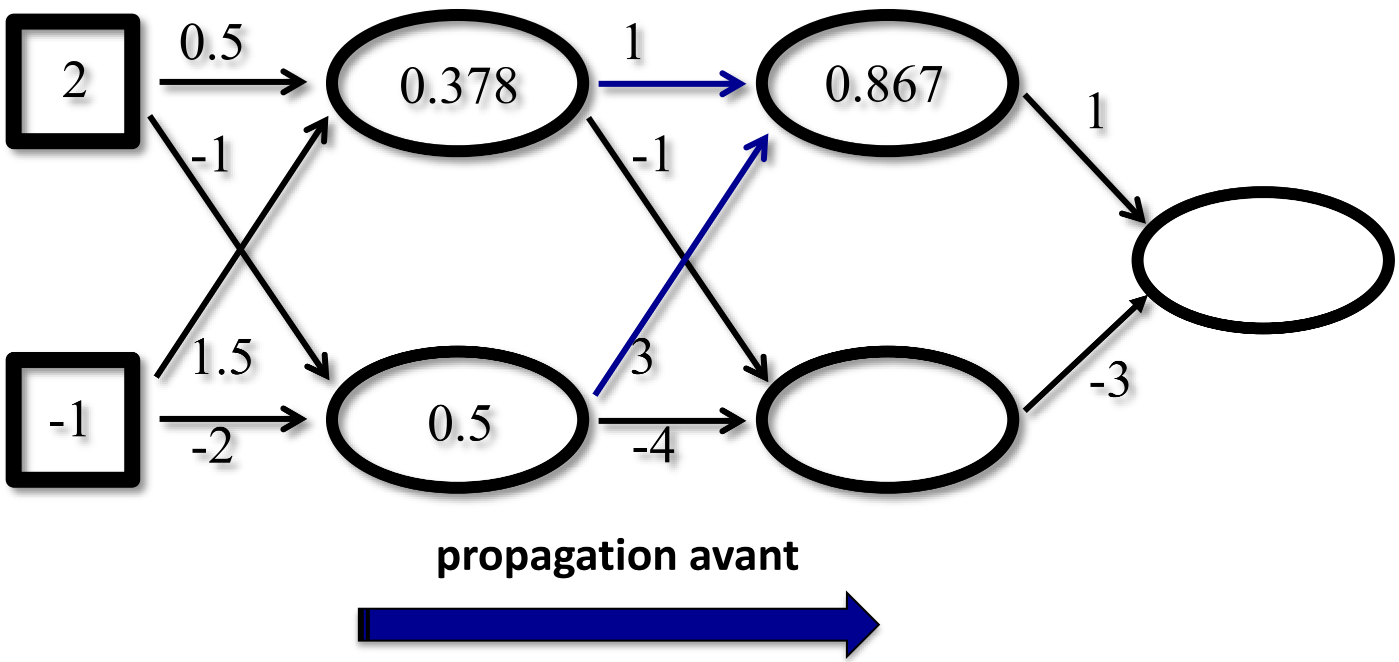


σ(-1 * 2 + -2 * -1) = σ(0) = 0.5

Exemple

$$\sigma(w.x) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

- Exemple: $\mathbf{x} = [2, -1]\,,\ y = 1$

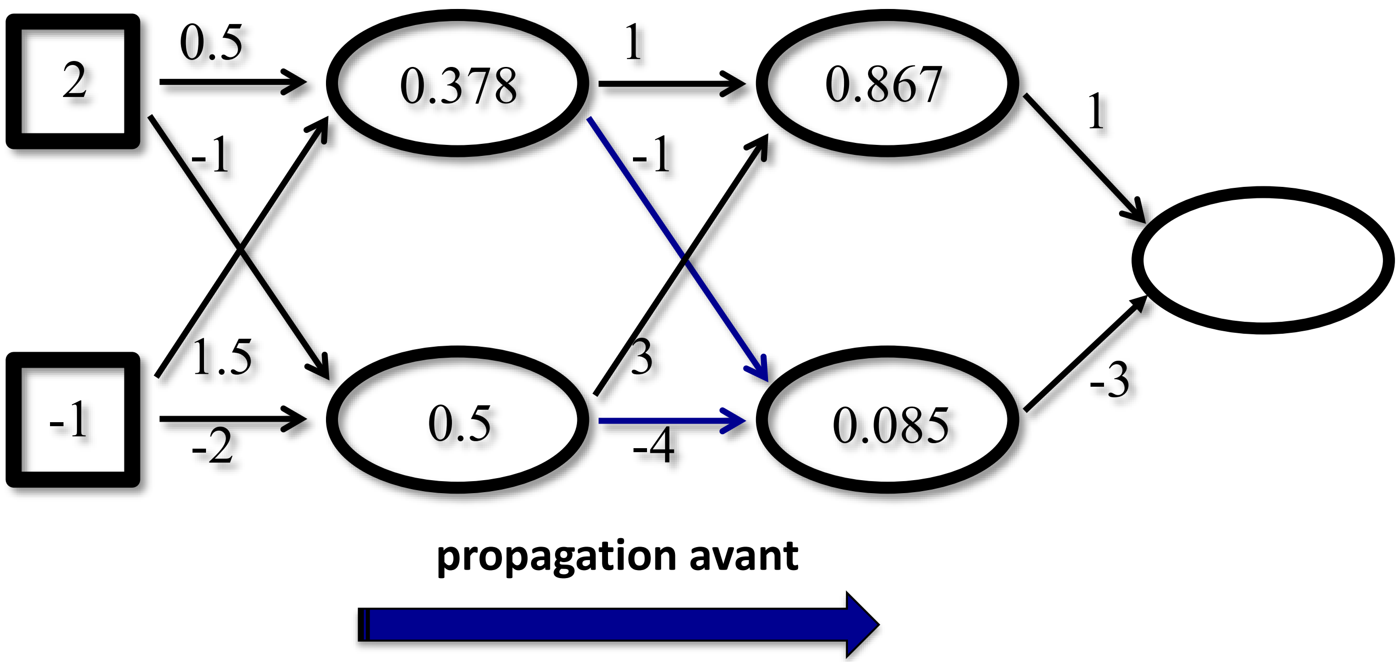


$$\sigma(1 * 0.378 + 3 * 0.5) = \sigma(1.878) = 0.867$$

Exemple

$$\sigma(\text{w.x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$

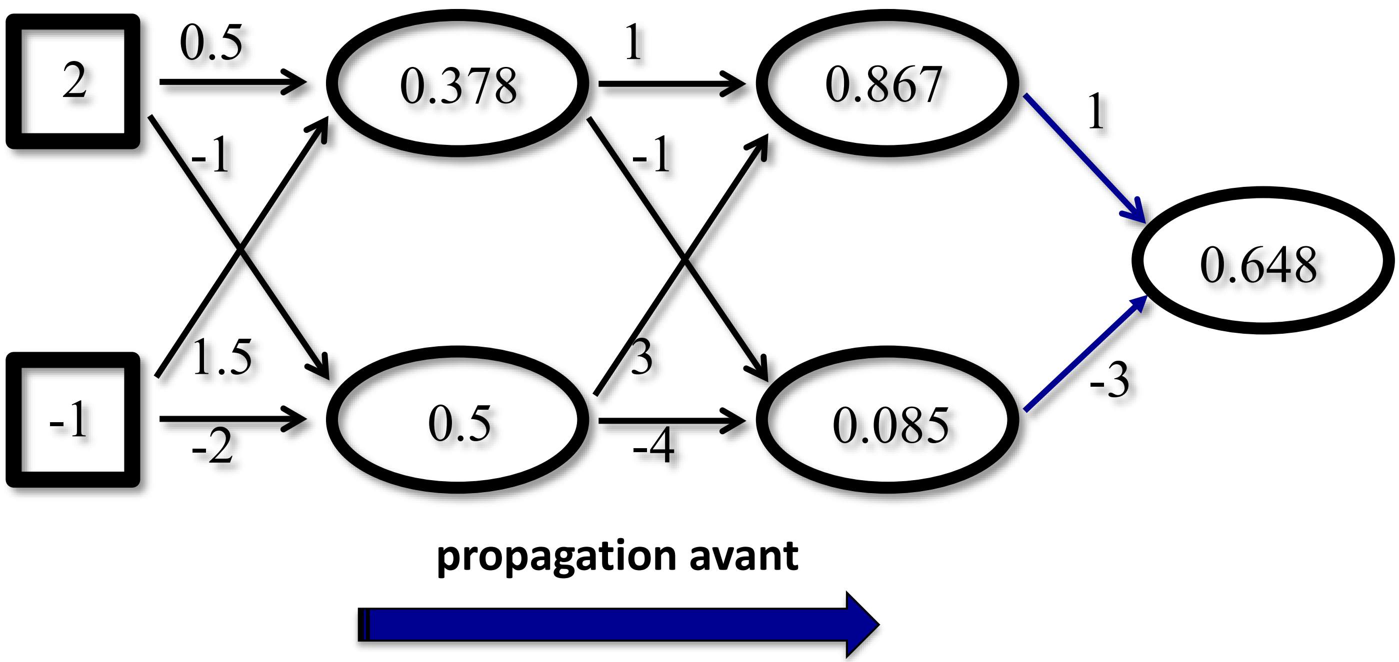


σ(-1 * 0.378 + -4 * 0.5) = σ(-2.378) = 0.085

Exemple

$$\sigma(\text{w.x}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

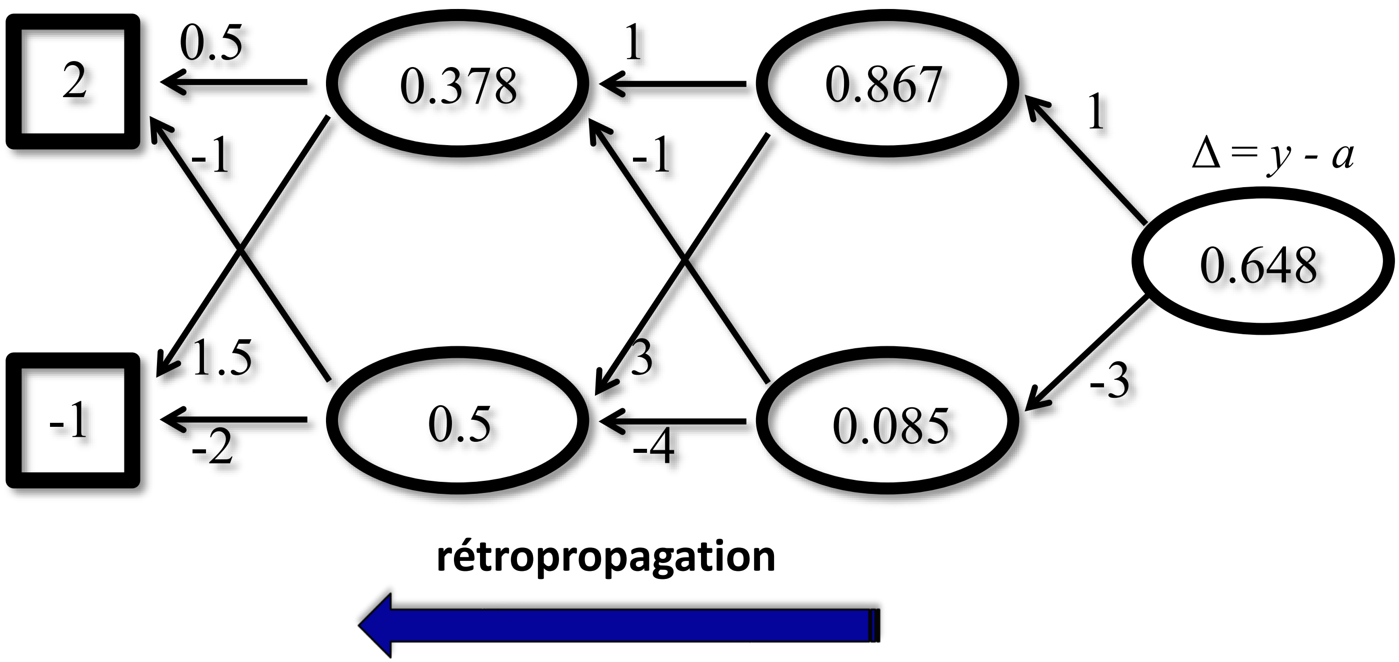
- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$



$$\sigma(1 * 0.867 + -3 * 0.085) = \sigma(0.612) = 0.648$$

Exemple

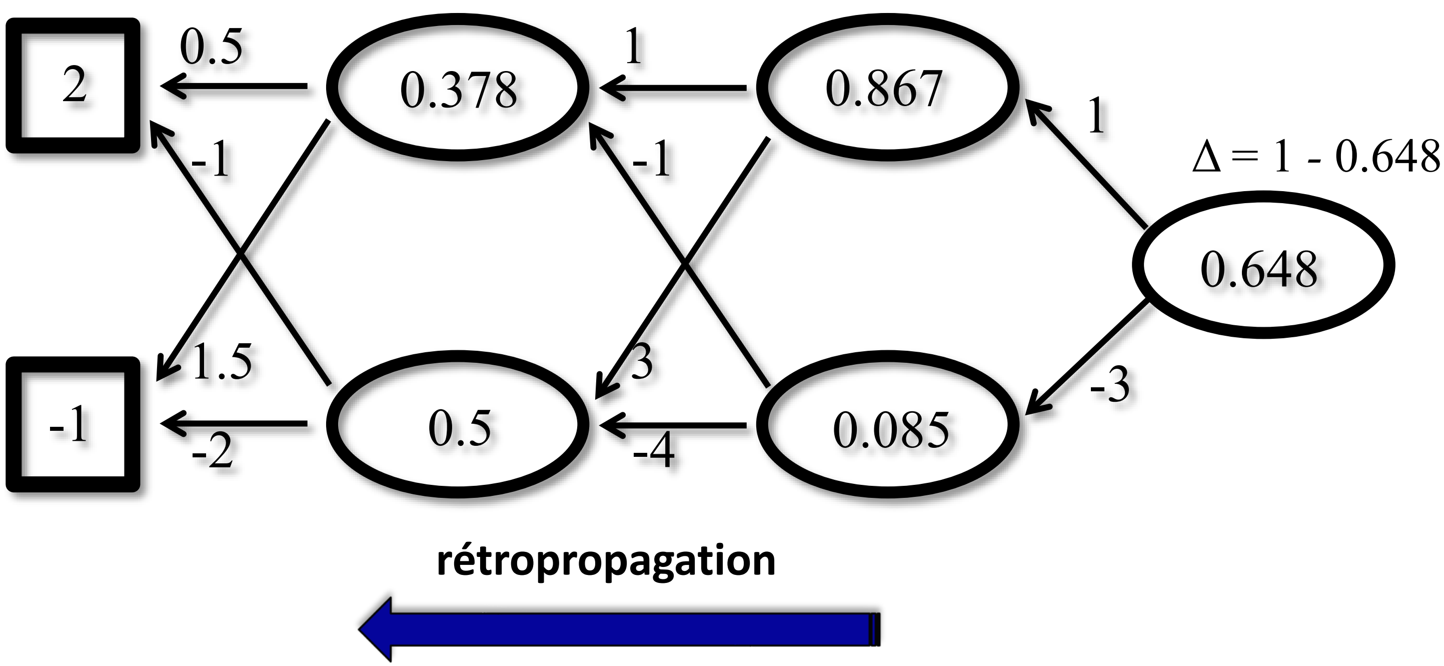
- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$



$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Exemple

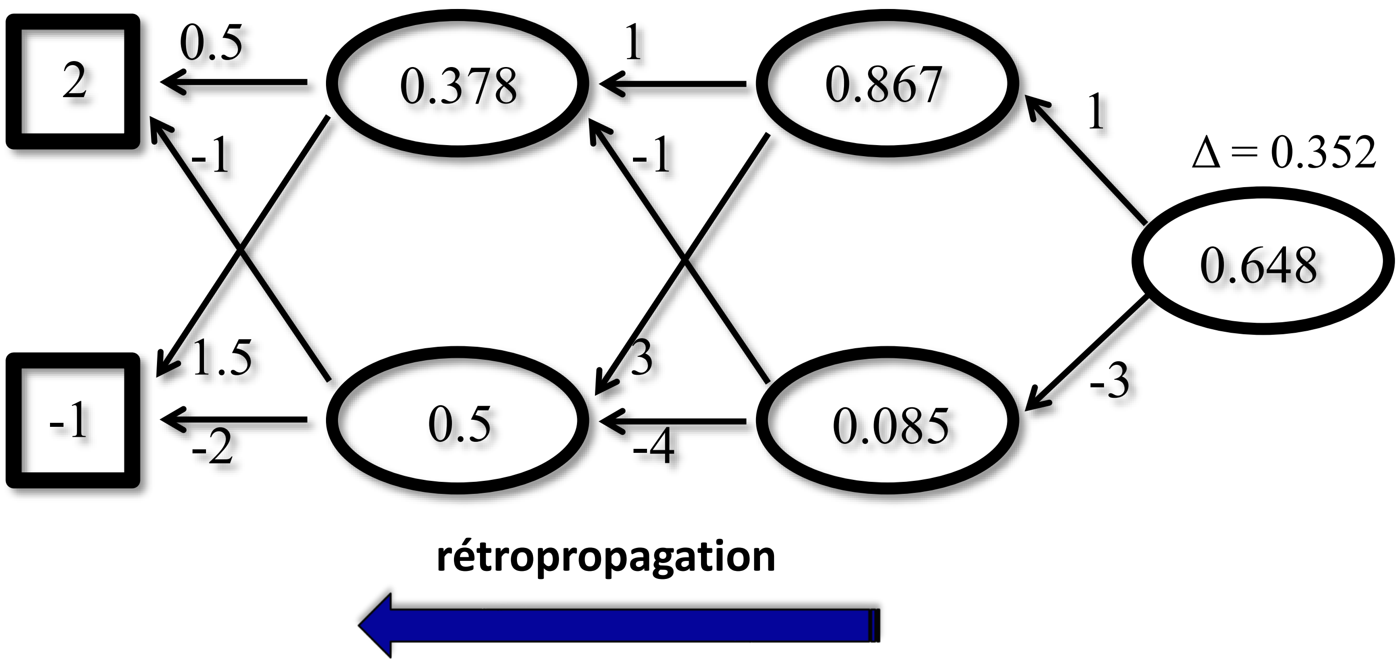
- Exemple: $\mathbf{x} = [2, -1]\,,\; y = 1$



$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Exemple

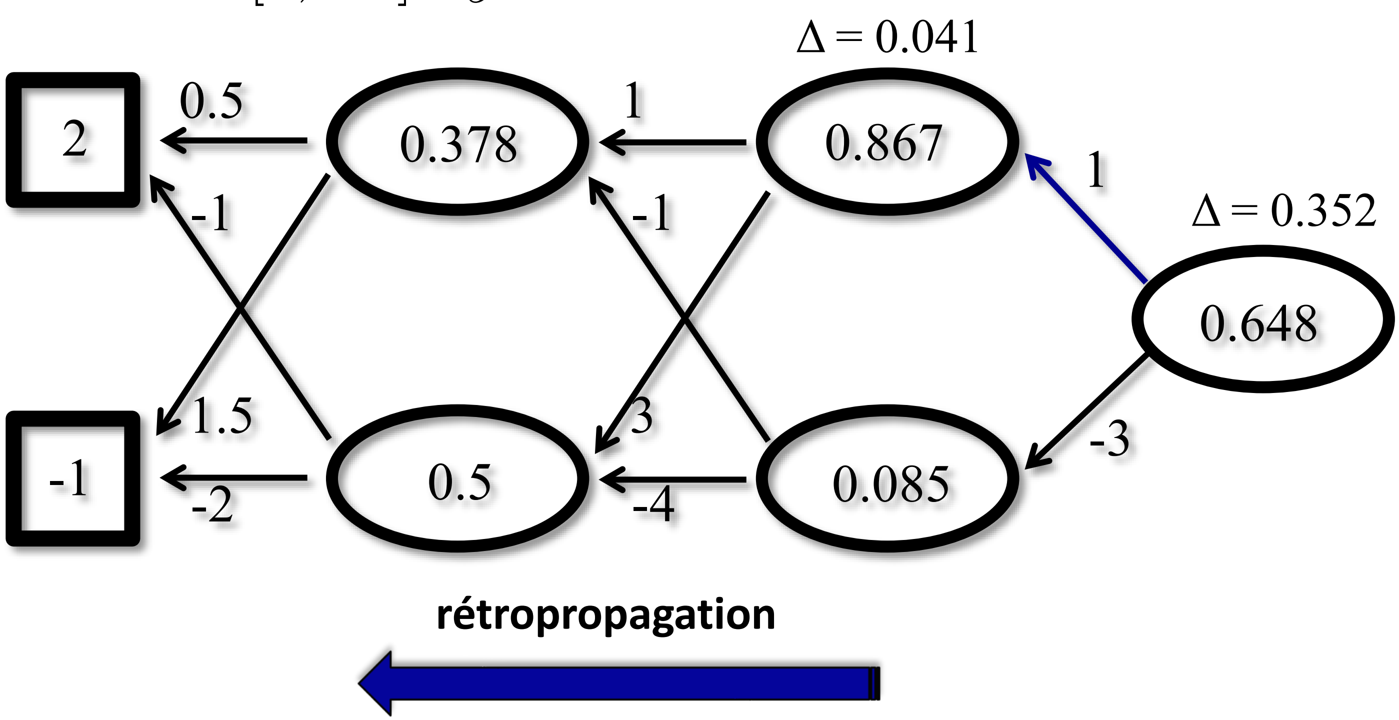
- Exemple: $\mathbf{x} = [2, -1]\,,\ y = 1$



$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Exemple

- Exemple: $\mathbf{x} = [2, -1]\,,\ y = 1$

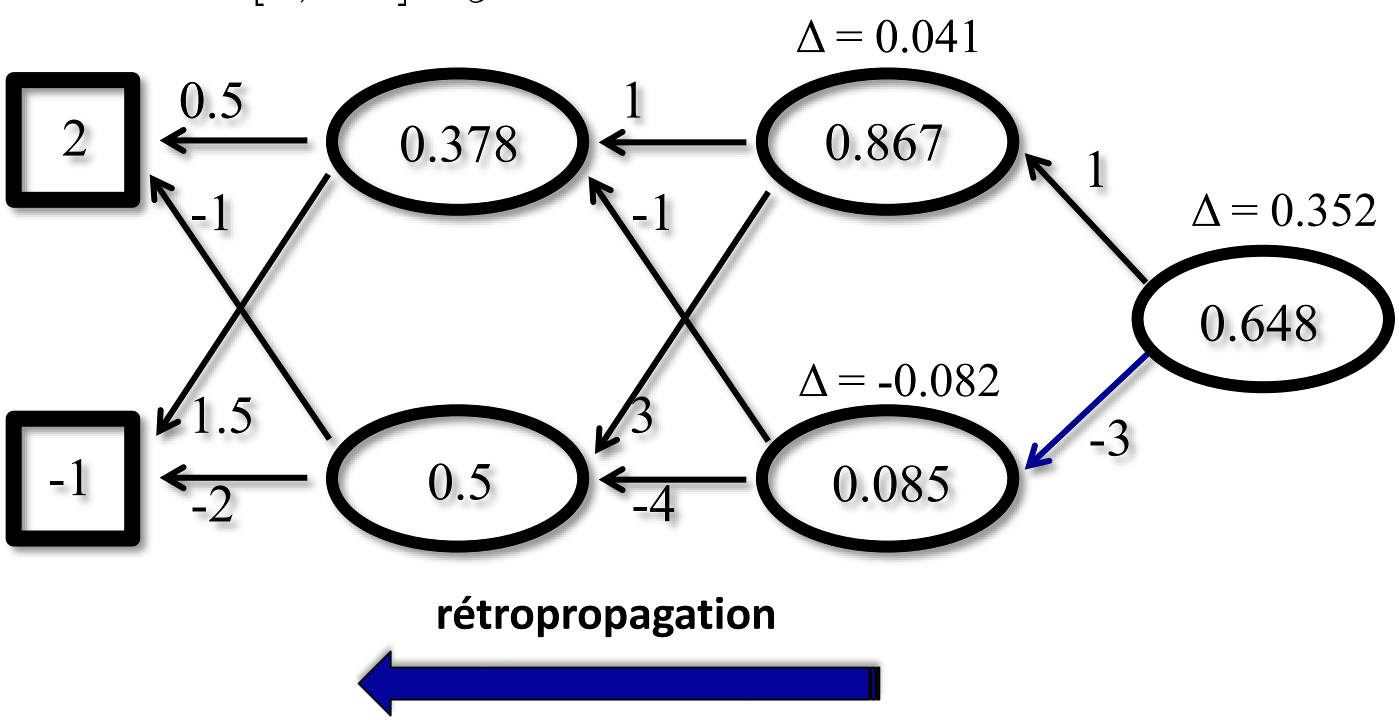


$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

$\Delta = 0.867 * (1\text{-}0.867) * 1 * 0.352 = 0.041$

Exemple

- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$

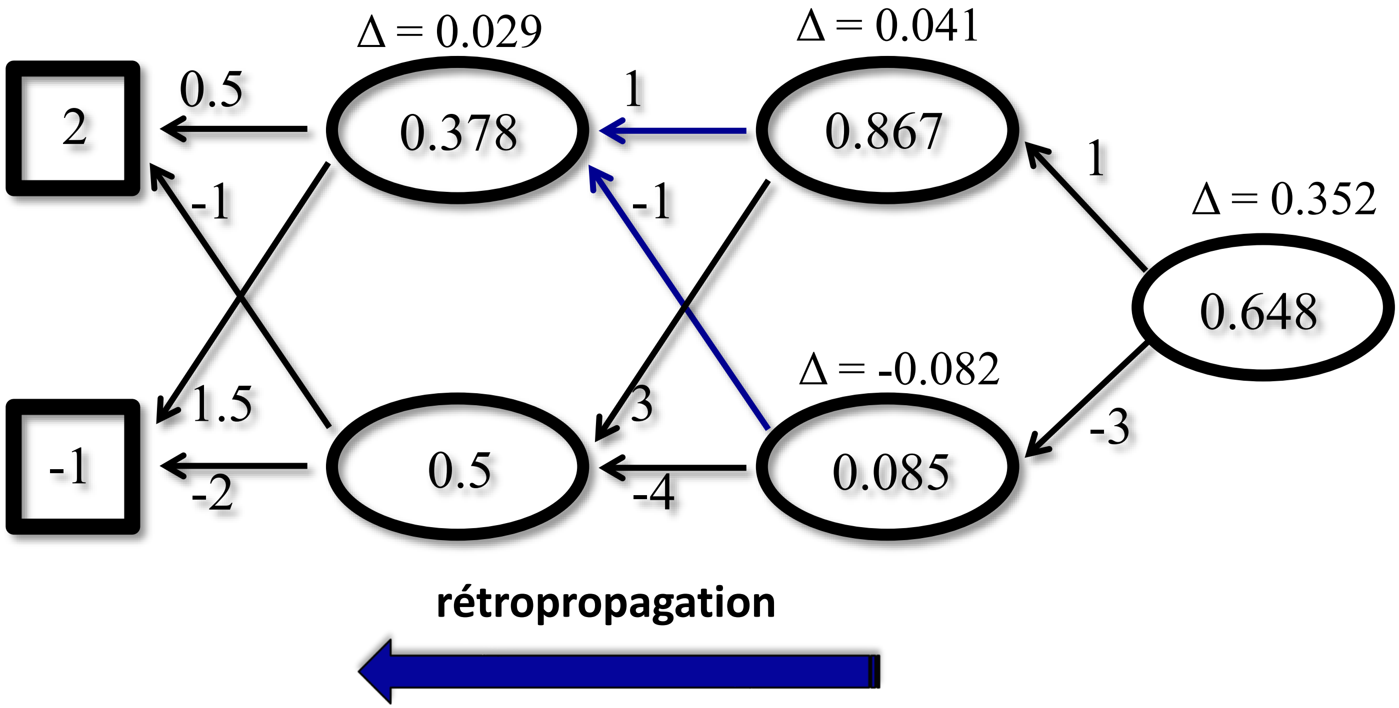


$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Δ = 0.085 * (1-0.085) * -3 * 0.352 = -0.082

Exemple

- Exemple: $\mathbf{x} = [2, -1]\,,\ y = 1$



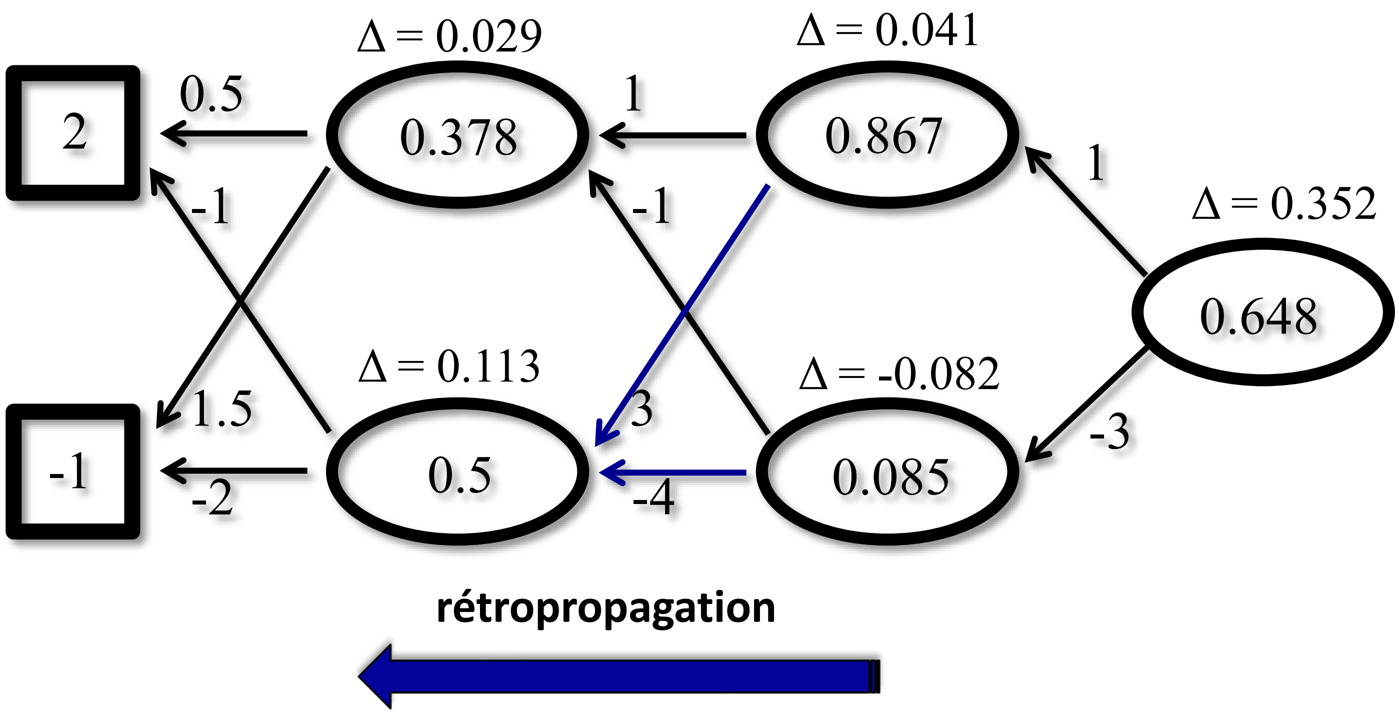
$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Δ = 0.378 * (1-0.378) * (1 * 0.041 + -1 * -0.082) = 0.029

Exemple

$$\sigma(w.x) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- Exemple: $\mathbf{x} = [2, -1]\,,\ y = 1$

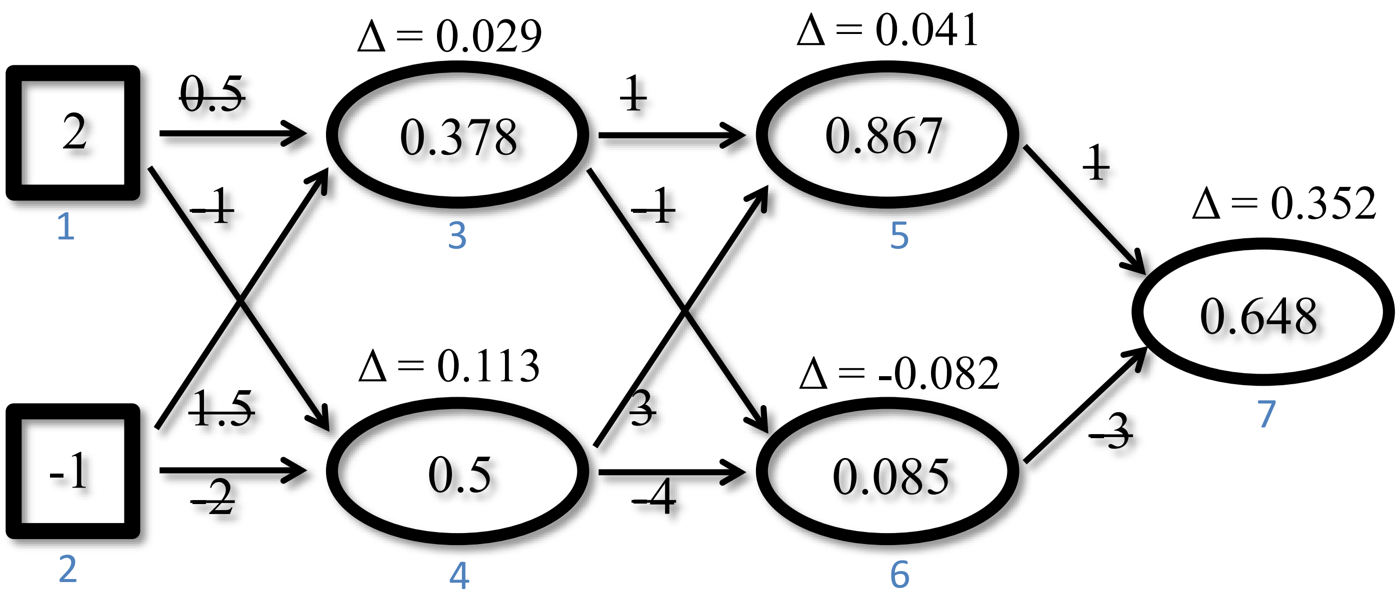


$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

$$\Delta = 0.5 * (1\text{-}0.5) * (3 * 0.041 + \text{-}4 * \text{-}0.082) = 0.113$$

Exemple

- Exemple: $\mathbf{x} = [2, -1]$, $y = 1$



$$w_{i,j} \leftarrow w_{i,j} + \alpha\ a_i\ \Delta[j]$$

mise à jour (α=0.1)

$w_{1,3} \leftarrow 0.5 + 0.1 * 2 * 0.029 = 0.506$

$w_{1,4} \leftarrow -1 + 0.1 * 2 * 0.113 = -0.977$

$w_{2,3} \leftarrow 1.5 + 0.1 * -1 * 0.029 = 1.497$

$w_{2,4} \leftarrow -2 + 0.1 * -1 * 0.113 = -2.011$

$w_{3,5} \leftarrow 1 + 0.1 * 0.378 * 0.041 = 1.002$

$w_{3,6} \leftarrow -1 + 0.1 * 0.378 * -0.082 = -1.003$

$w_{4,5} \leftarrow 3 + 0.1 * 0.5 * 0.041 = 3.002$

$w_{4,6} \leftarrow -4 + 0.1 * 0.5 * -0.082 = -4.004$

$w_{5,7} \leftarrow 1 + 0.1 * 0.867 * 0.352 = 1.031$

$w_{6,7} \leftarrow -3 + 0.1 * 0.085 * 0.352 = -2.997$

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
  inputs: *examples*, a set of examples, each with input vector $\mathbf{x}$ and output vector $\mathbf{y}$
    network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

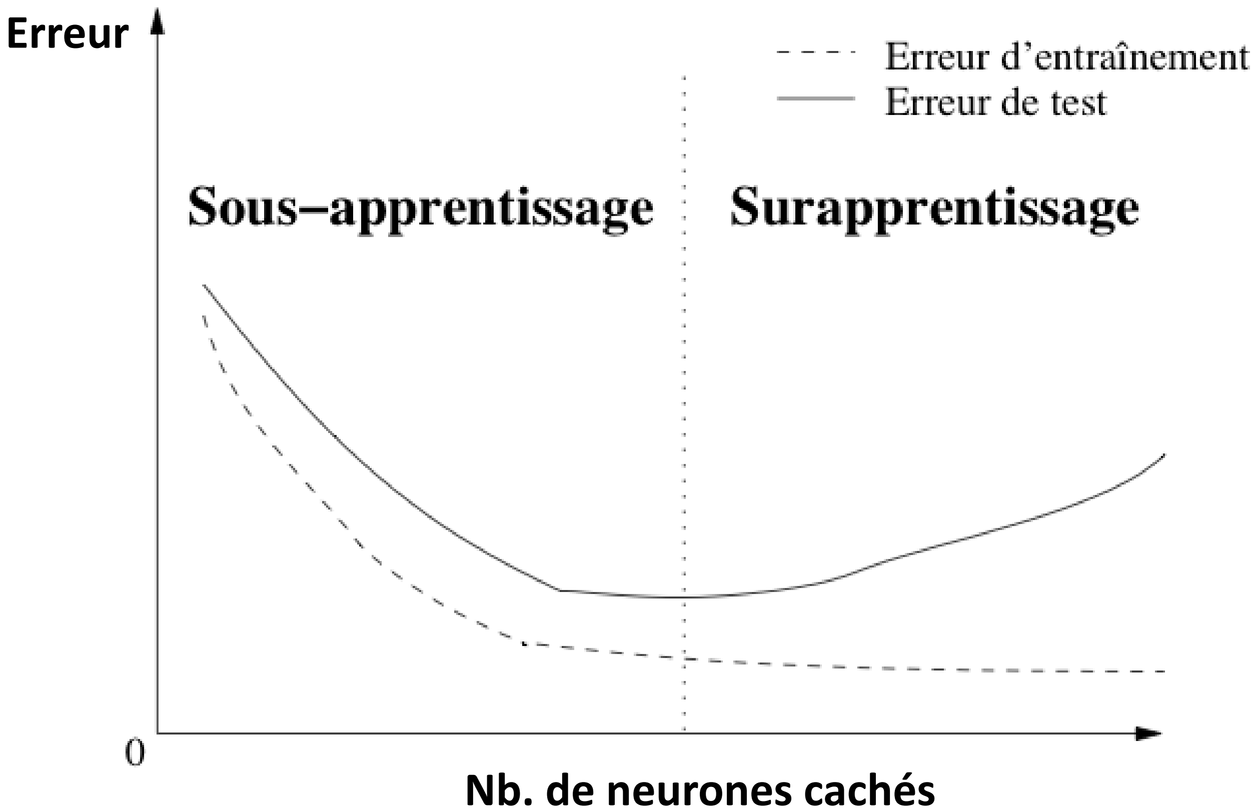
  for each weight $w_{i,j}$ **in** *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  repeat
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      / Propagate the inputs forward to compute the outputs */*
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      / Propagate deltas backward from output layer to input layer */*
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow y_j - a_j \quad (= -\partial Loss / \partial in_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g(in_i)(1 - g(in_i)) \sum_j w_{i,j} \Delta[j]$
      / Update every weight in network using deltas */*
      for each weight $w_{i,j}$ **in** *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

Pour vous pratiquer

- playground.tensorflow.org
- TP 2
- tensorflow.org

Retour sur la notion de généralisation

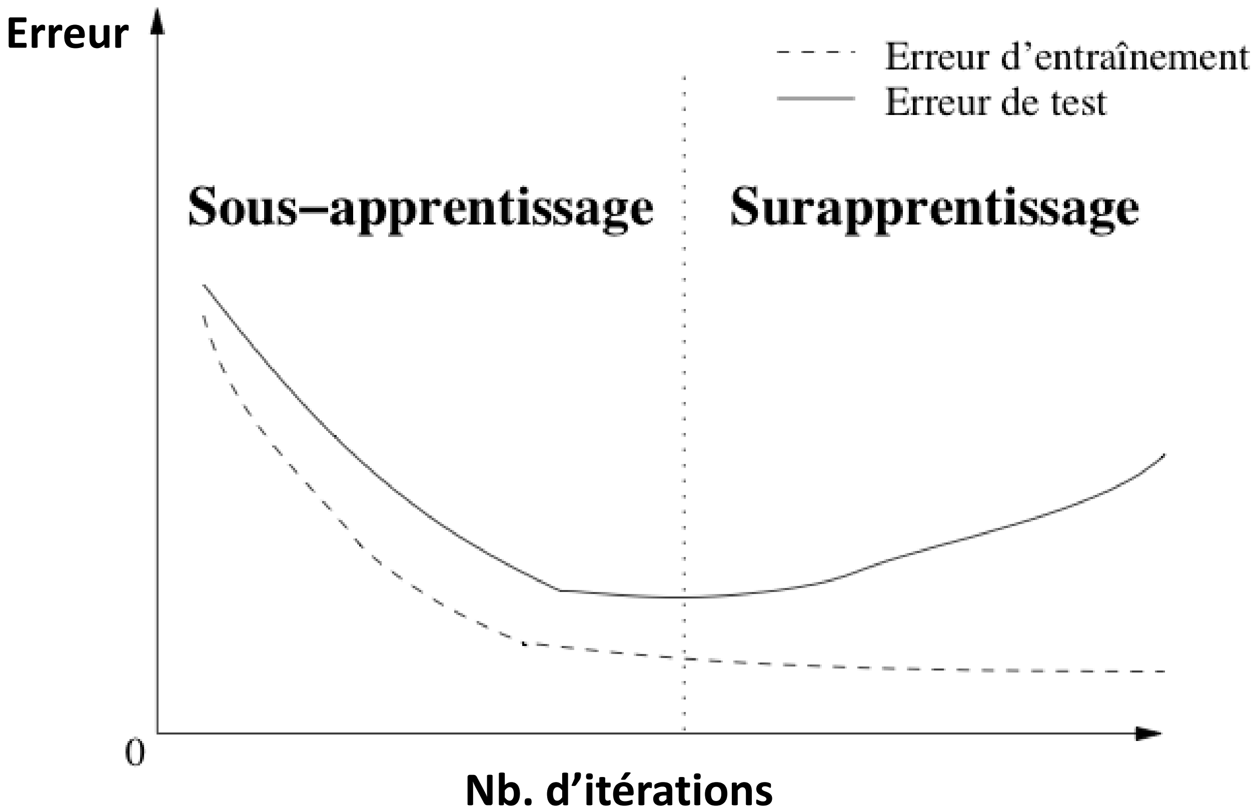
- Comment choisir le nombre de neurones cachés?



On veut trouver ce point, sans sous-apprentissage ni surapprentissage

Retour sur la notion de généralisation

- Quand arrêter l'entraînement?



Hyper-paramètres

- Dans tous les algorithmes d'apprentissage qu'on a vu jusqu'à maintenant, il y avait des « options » à déterminer
  - *k* plus proche voisins: la valeur de « k »
  - Perceptron et régression logistique: le taux d'apprentissage α, nb. itérations N
  - réseau de neurones: taux d'apprentissage, nb. d'itérations, nombre de neurones cachés, fonction d'activation $g(\cdot)$
- On appelle ces « options » des **hyper-paramètres**
  - choisir la valeur qui marche le mieux sur l'ensemble d'entraînement est en général une mauvaise idée (mène à du surapprentissage)
    - pour le *k* plus proche voisin, l'optimal sera toujours *k=1*
  - on ne peut pas utiliser l'ensemble de test non plus, **ça serait tricher!**
  - en pratique, on garde un autre ensemble de côté, l'**ensemble de validation**, pour choisir la valeur de ce paramètre
- Sélectionner les valeurs d'hyper-paramètres est une forme d'apprentissage

Procédure d'évaluation complète

- Utilisation typique d'un algorithme d'apprentissage
  - séparer nos données en 3 ensembles:
    entraînement (70%), validation (15%) et test (15%)
  - faire une liste de valeurs des hyper-paramètres à essayer
  - pour chaque élément de cette liste, lancer l'algorithme d'apprentissage sur l'ensemble d'entraînement et mesurer la performance sur l'ensemble de validation
  - réutiliser la valeur des hyper-paramètres avec la meilleure performance en validation, pour calculer la performance sur l'ensemble de test
- La performance sur l'ensemble de test est alors une **estimation non-biaisée** (non-optimiste) de la performance de généralisation de l'algorithme
- On peut utiliser la performance pour comparer des algorithmes d'apprentissage différents

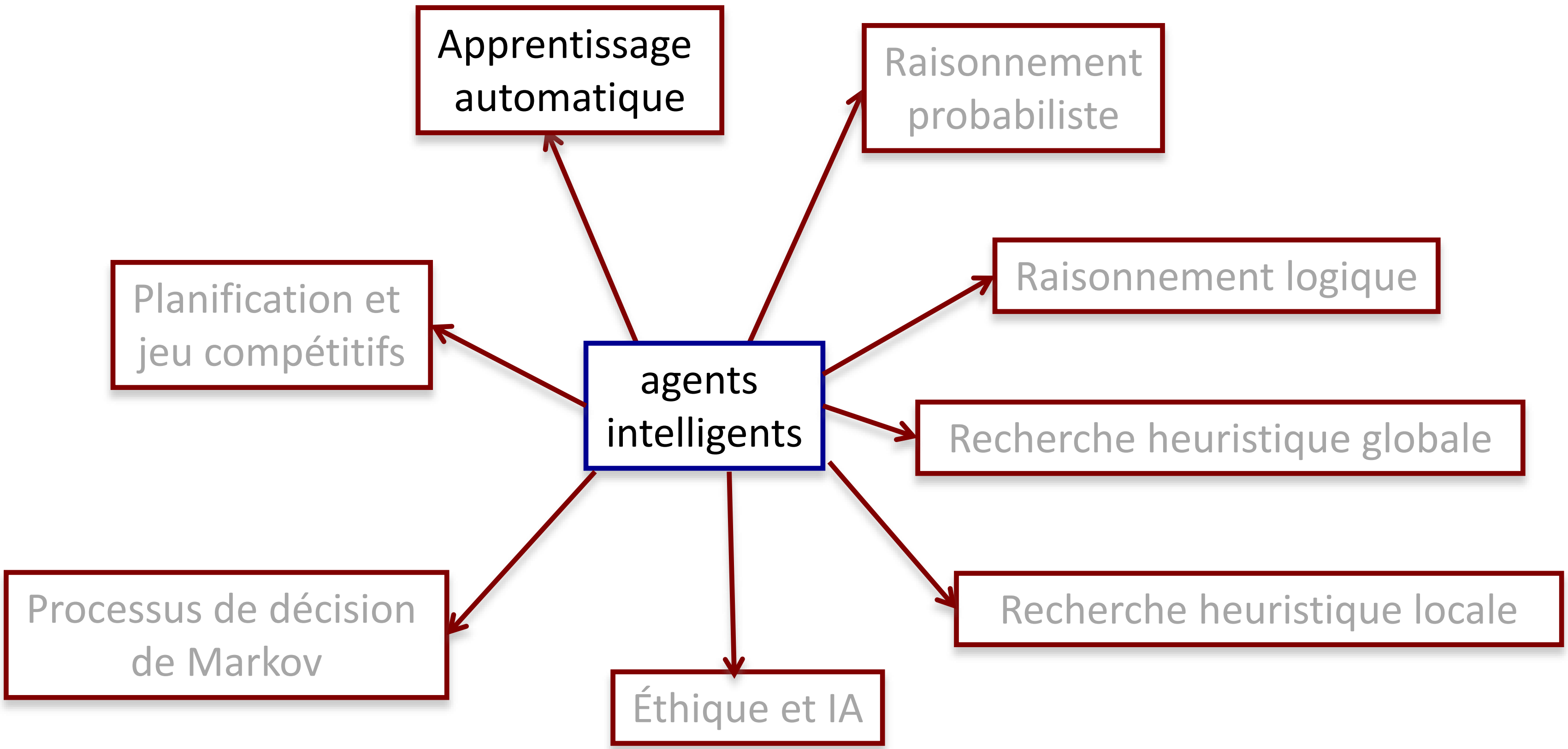
Autres définitions

- **Capacité** d'un modèle h : habilité d'un modèle à réduire son erreur d'entraînement, à mémoriser ces données
- **Modèle paramétrique**: modèle dont la capacité n'augmente pas avec le nombre de données (Perceptron, régression logistique, réseau de neurones avec un **nombre de neurones fixe**)
- **Modèle non-paramétrique**: l'inverse de paramétrique, la capacité augmente avec la taille de l'ensemble d'entraînement (*k* plus proche voisin, réseau de neurones avec un **nombre de neurones adapté aux données d'entraînement**)
- **Époque**: une itération complète sur tous les exemples d'entraînement
- **Fonction d'activation**: fonction non-linéaire $g(\cdot)$ des neurones cachés

Conclusion

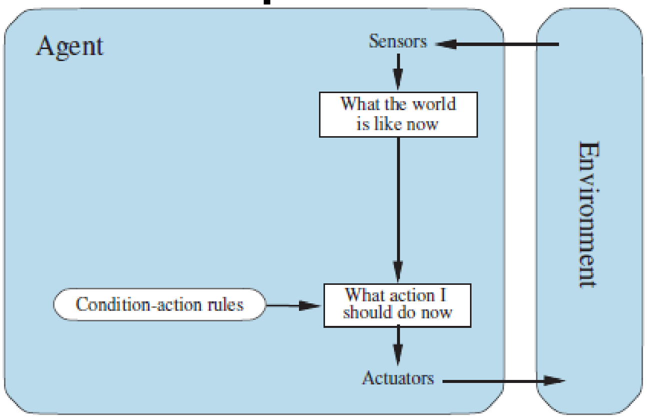
- L'apprentissage automatique permet d'extraire une expertise (humaine) à partir de données
- Nous avons vu le cas spécifique de la classification
  - il existe plusieurs autres problèmes pour lesquels l'apprentissage automatique peut être utile (voir les autres cours IA mentioné dans l'introduction du cours: IFT599, IFT603, IFT 607, IFT608)
- L'algorithme des *k* plus proches voisins est simple et puissant (non-linéaire), mais peut être lent avec de grands ensembles de données
- Les algorithmes linéaires du Perceptron et de la régression logistique sont moins puissants mais efficaces
- Les réseaux de neurones artificiels peuvent avoir l'expressivité (capacité) d'une classifieur des *k* plus proches voisins, tout en étant plus efficace

Concept et algorithmes couverts par le cours

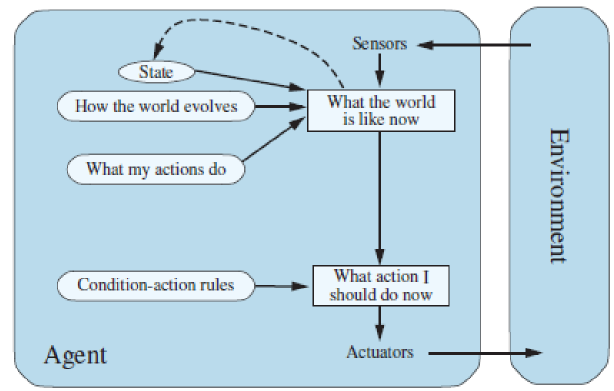


Apprentissage automatique : pour quel type d'agent?

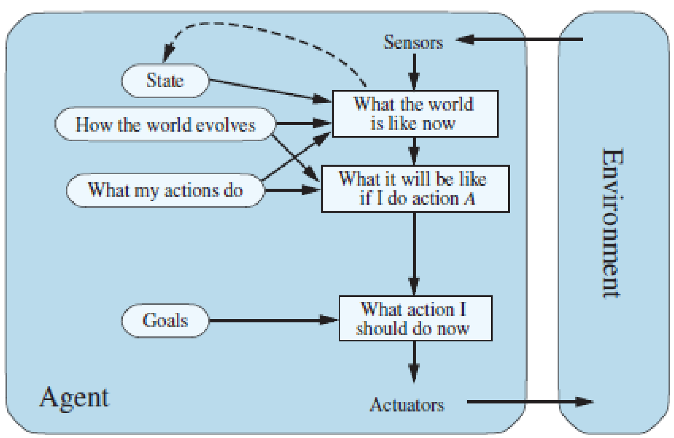
Simple reflex



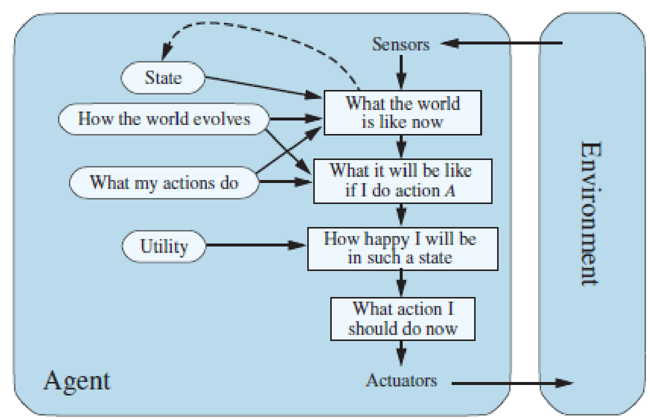
Model-based reflex



Goal-based



Utiliy-based



Vous devriez être capable de...

- Simuler les algorithmes vus
  - régression logistique
  - réseau de neurones
- Décrire le développement et l'évaluation (de façon non-biaisée) d'un système basé sur un algorithme d'apprentissage automatique
- Comprendre les notions de sous-apprentissage et surapprentissage
- Savoir ce qu'est un hyper-paramètre

Rappel – Non couverte à l’examen

DÉRIVATION EN CHAÎNE

Dérivation en chaîne

- Si on peut écrire une fonction $f(x)$ à partir d'un résultat intermédiaire $g(x)$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

- De façon récurrente, si on peut exprimer $g(x)$ à partir de $h(x)$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \frac{\partial g(x)}{\partial h(x)} \frac{\partial h(x)}{\partial x}$$

- Si on peut écrire une fonction $f(x)$ à partir de résultats intermédiaires $g_i(x)$, alors on peut écrire la dérivée partielle

$$\frac{\partial f(x)}{\partial x} = \sum_i \frac{\partial f(x)}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$$

Dérivation en chaîne

- Exemple: $f(x) = 4\exp(x) + 3(1+x)^3$

- On considère $g_1(x) = \exp(x)$ et $g_2(x) = 1 + x$
- Donc on peut écrire $f(x) = 4g_1(x) + 3g_2(x)^3$
- On peut obtenir la dérivée partielle avec les morceaux:

$$\frac{\partial f(x)}{\partial g_1(x)} = 4 \qquad \frac{\partial g_1(x)}{\partial x} = \exp(x)$$

$$\frac{\partial f(x)}{\partial g_2(x)} = 9g_2(x)^2 \qquad \frac{\partial g_2(x)}{\partial x} = 1$$

- Donc: $\frac{\partial f(x)}{\partial x} = 4\exp(x) + 9g_2(x) = 4\exp(x) + 9(1+x)^2$