

Cours d'ouverture

1. apprentissage supervisé
2. apprentissage par renforcement



Nicolas Bredeche
Sorbonne Université, CNRS
nicolas.bredeche(-)sorbonne-universite.fr

Apprentissage supervisé

Réseaux de neurones artificiel et
algorithme de retro-propagation du gradient

● Apprentissage supervisé

- On dispose d'exemples positifs et négatifs d'un concept
- On souhaite apprendre comment étiqueter un nouvel exemple
- Applications: approximation ou modélisation de fonctions, identification d'objet/texte/son, détection des mauvais payeurs, prédiction de séries temporelles, application au contrôle robotique, etc.

Apprentissage supervisé

● Le problème:

$$\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \{-1, 1\} \text{ or } \mathbf{R}$$

● Une fonction de perte:

$$\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbf{R}^+$$

● Objectif:

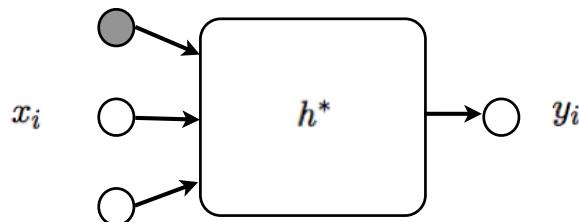
- Trouver l'hypothèse qui minimise l'erreur de prédiction

$$h^* : \mathcal{X} \mapsto \mathcal{Y}$$

$$\text{tel que: } L(h^*) = \operatorname{argmin}\{L(h), h \in H\}$$

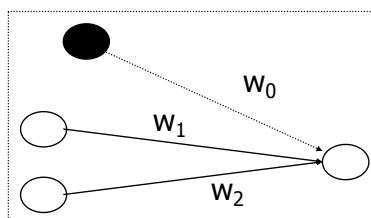
Hypothèse cherchée

$$\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \{-1, 1\} \text{ or } \mathbf{R}$$



h^* peut être représenté par de multiples formalismes... un réseau de neurones, des règles, un arbre de décision, SVM, etc.

Fonctionnement général



$$f_{activation}(w_0 + \sum_{i=1}^n w_i x_i)$$

- Perceptron [Rosenblatt, 1957]
 - Neurones formels / Perceptron linéaire à seuil
 - Fonction d'activation (ex.: sigmoïde ou Heaviside)
 - Neurone de biais (=1)
- Notations
 - \mathbf{x}_i : somme avant seuillage du neurone i
 - \mathbf{a}_i : somme après seuillage du neurone i ("activation" du neurone)
 - \mathbf{w}_0 : poids du neurone de biais

- Calcul de l'erreur quadratique

$$E(inputs) = 1/2 \sum_{i \in outputs} (a_i^* - a_i)^2$$

Méthodologie:

On trace l'erreur (en y) par rapport au nombre d'itérations de l'algorithme (en x)

- Rétro propagation du gradient de l'erreur

$$w_{ji}^{t+1} = w_{ji}^t + \mu * (a_i^* - a_i) * x_j$$

w_{ji} : poids de l'arc reliant le neurone d'entrée j et le neurone de sortie i

x_j : valeur du neurone d'entrée j

a_i : valeur du neurone de sortie i

a_i^* : valeur désirée du neurone de sortie i

μ : pas d'apprentissage

Limites

- Perceptron simple

- Problème lorsque les entrées sont inter-dépendantes
 - ▶ calculer XOR, estimer si le nombre d'entrées à 1 est paire, etc.
- Un perceptron agit comme un séparateur linéaire

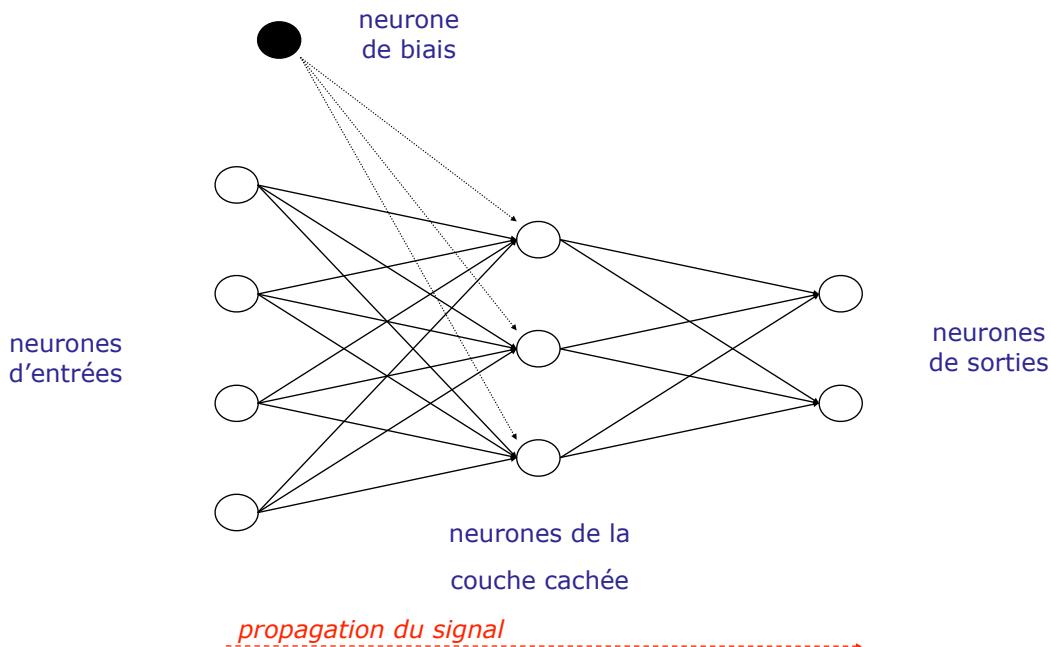
- Solution: ajouter une couche cachée!

- Théoriquement: approximateur universel
- Mais: il faut définir l'algorithme d'apprentissage
 - ▶ comment estimer l'erreur due aux neurones de la couche cachée?
- => **perceptron multi-couches**

Perceptron Multi-Couches (MLP)

9

[LeCun, 1984] [Rumelhart et McLelland, 1984][Werbos, 1974][Parker, 1982]



Propriété requise : fonction d'activation non-linéaire et dérivable

Perceptron Multi-Couches (MLP)

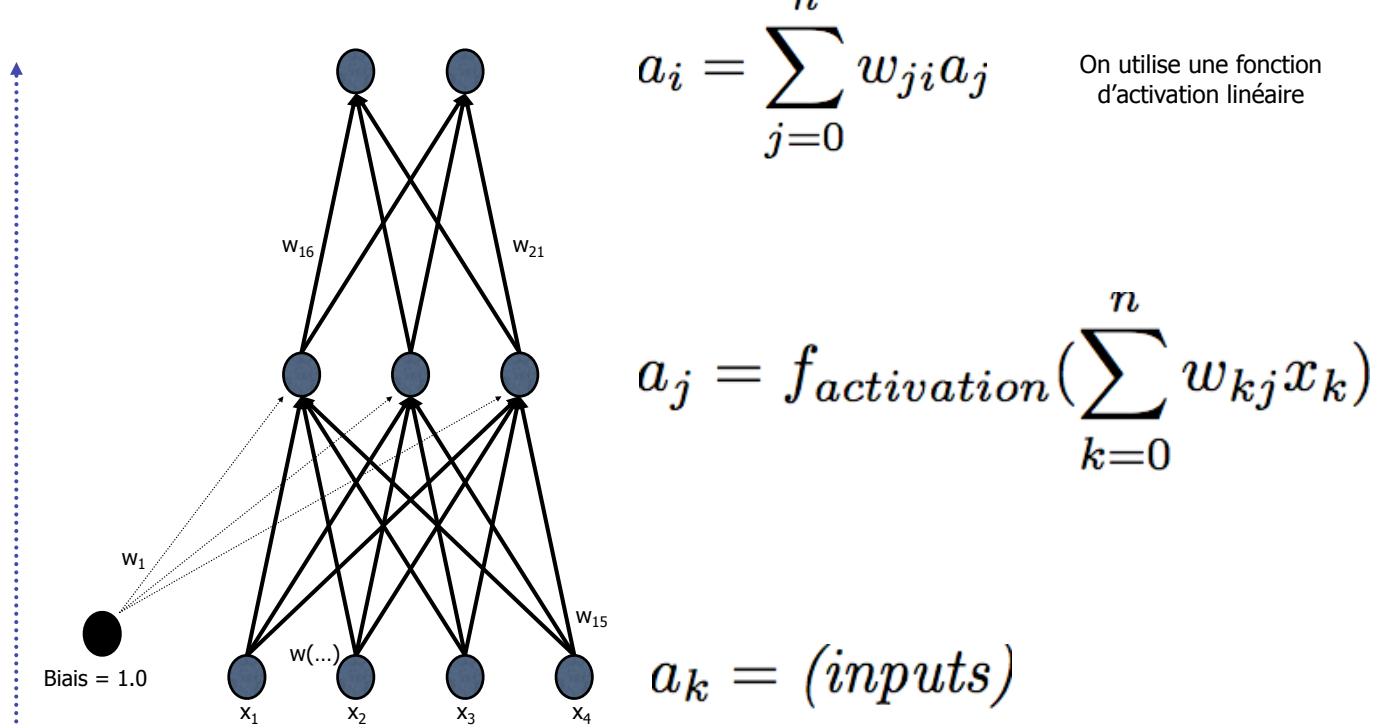
10

[LeCun, 1984] [Rumelhart et McLelland, 1984][Werbos, 1974][Parker, 1982]

- Propriétés requises :
 - couche cachée: fonction d'activation non-linéaire et dérivable
- En pratique:
 - couche cachée:
 - ▶ fonction sigmoïde $f(x) = 1/(1 + e^{-kx})$ equiv. à : $f(x) = (1 + \tanh(x))/2$
dérivé: $f'(x) = f(x)(1 - f(x))$
 - ▶ tangente hyperbolique
 - ▶ Rectified Linear Unit (ReLU)
 - couche de sortie : fonction linéaire
- Remarques
 - paramètre clé: le nombre de neurones cachés
 - ▶ Remarque: évidemment, la topologie, le nombre de couche, ont aussi une influence, mais sont aussi plus difficile à régler.

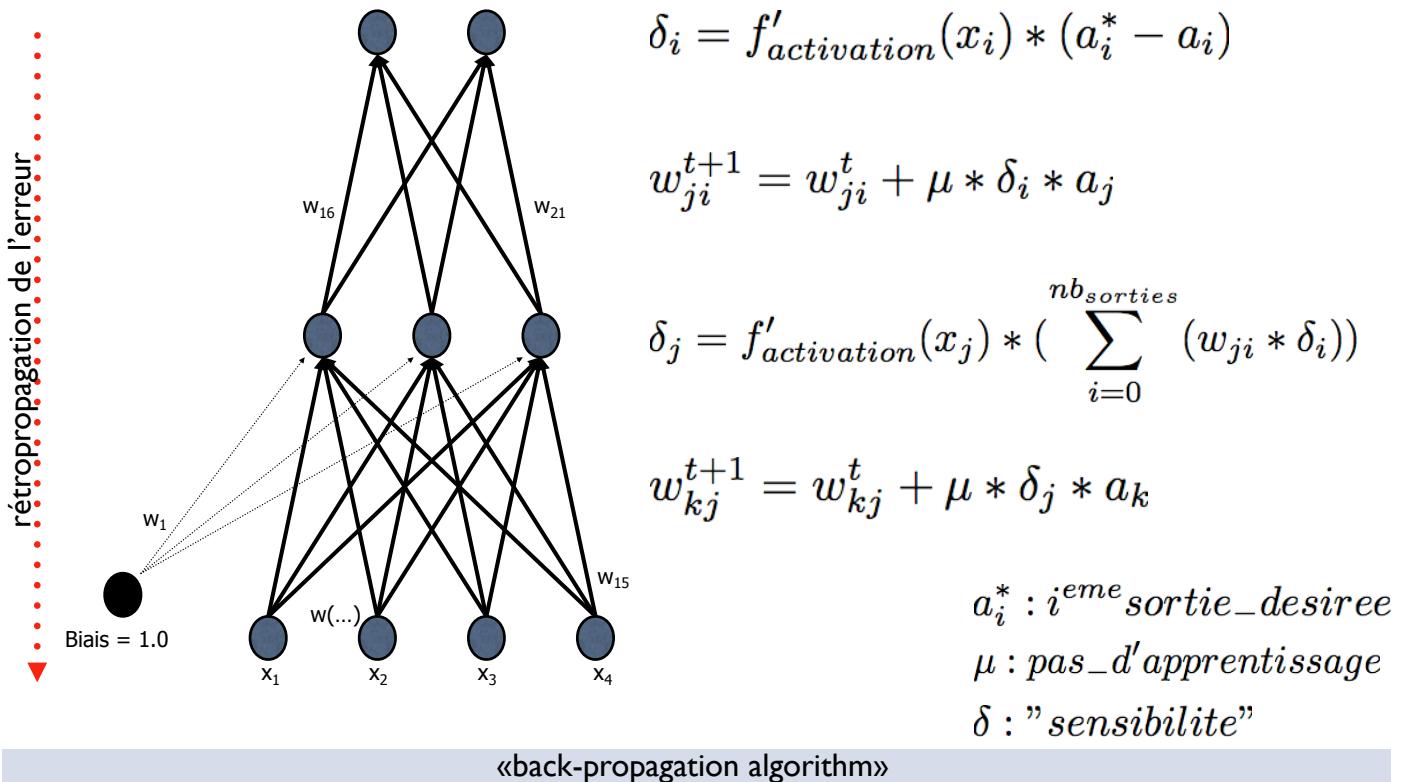
Propagation du signal dans un MLP

11

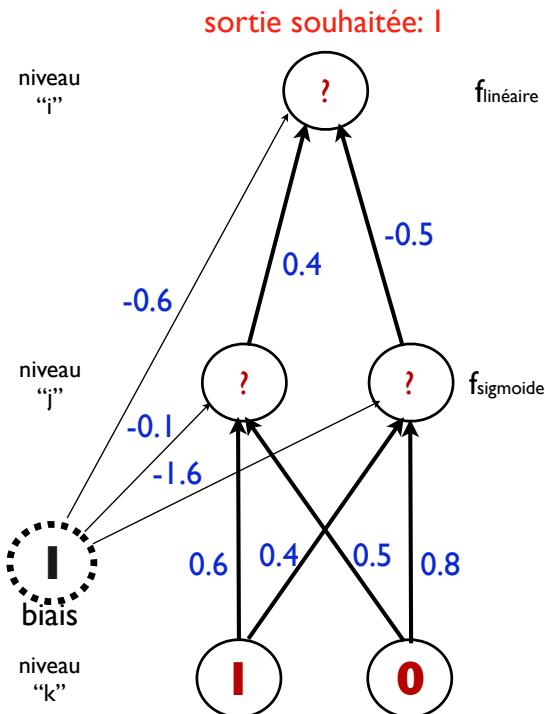


Rétro-propagation de l'erreur

12



Mise en pratique : propagation et rétro-propagation¹³



Imaginons:

- apprentissage de la fonction OU
- ici: exemple "1 OU 0"
- sortie attendue: "1"

Objectif:

Régler les poids du réseau.

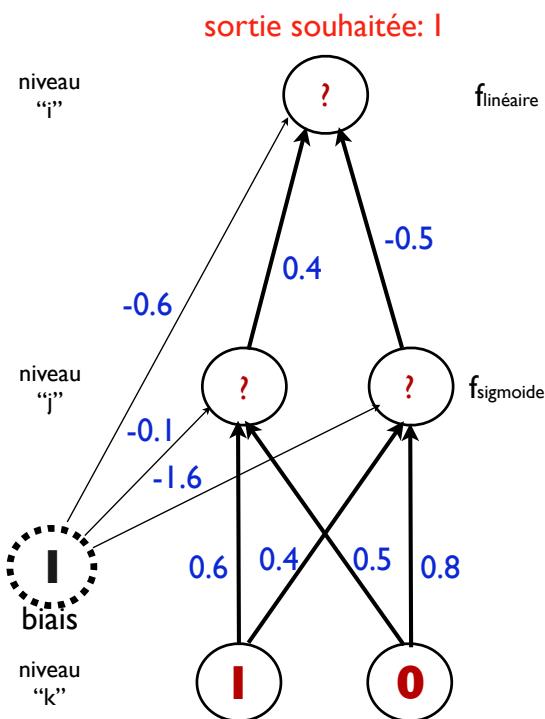
Etude pratique:

1. propagation du signal
2. rétro-propagation de l'erreur

Remarque: en pratique, il faudrait faire cela un grand nombre de fois pour chaque exemple de la base d'apprentissage.

activation: sigmoïde $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties
dérivé sigmoïde: $f'(x) = f(x)(1 - f(x))$ Ici, on fixe mu=0.1 (pas d'apprentissage)

Etape 1 : propagation du signal



$$a_k = (\text{inputs})$$

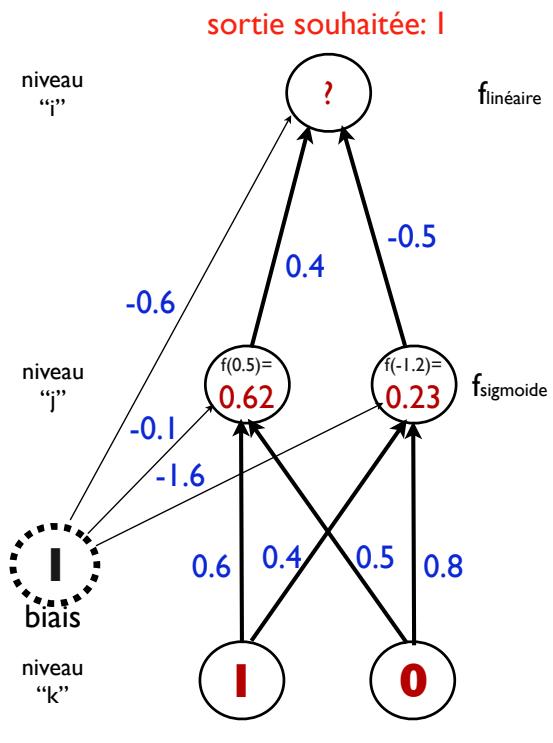
$$a_{k=1} = 1$$

$$a_{k=2} = 0$$

activation: sigmoïde $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

Etape I : propagation du signal

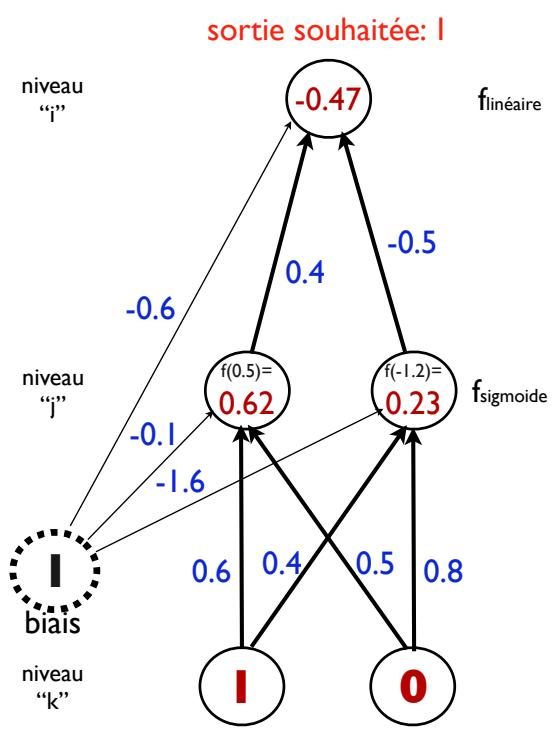
15



activation: sigmoïde $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

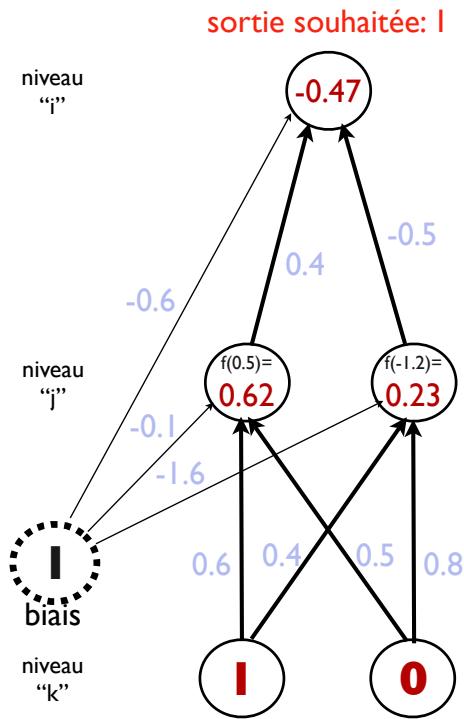
Etape I : propagation du signal

16



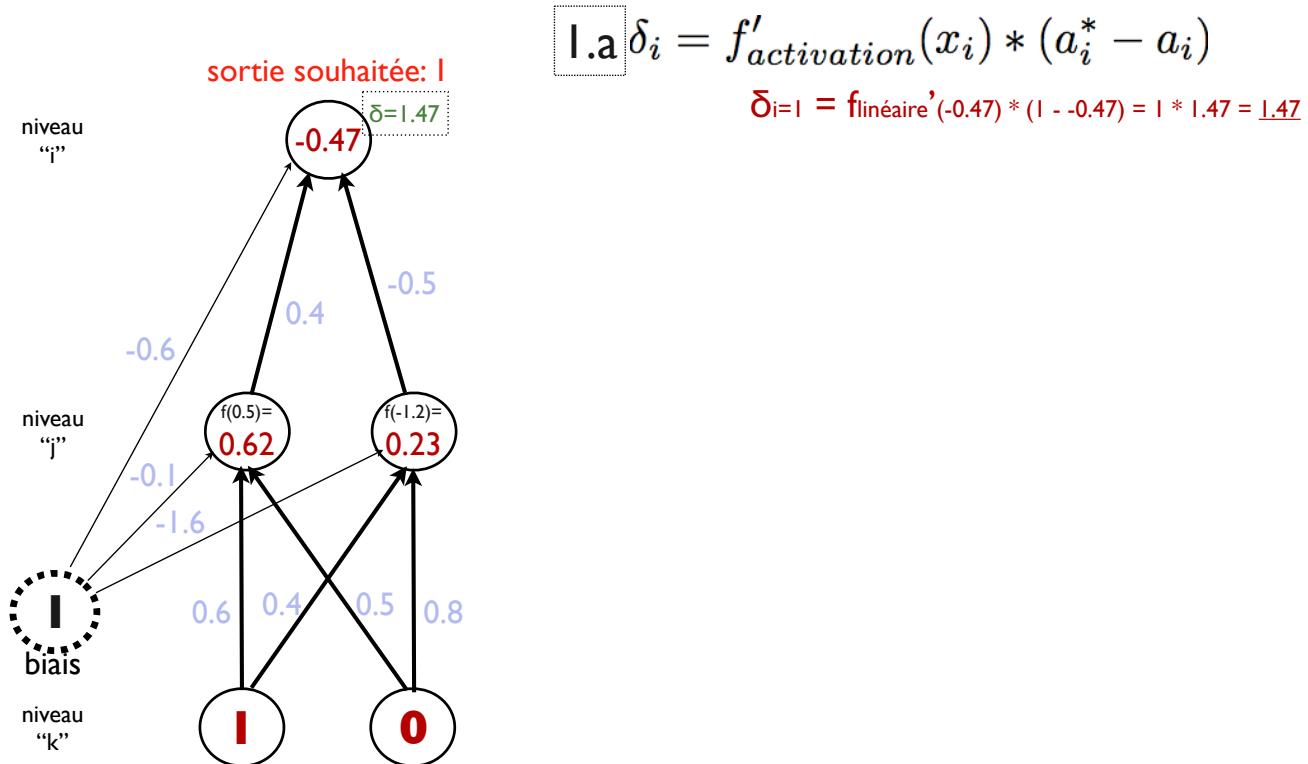
activation: sigmoïde $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

Etape 2 : rétro-propagation de l'erreur et correction¹⁷



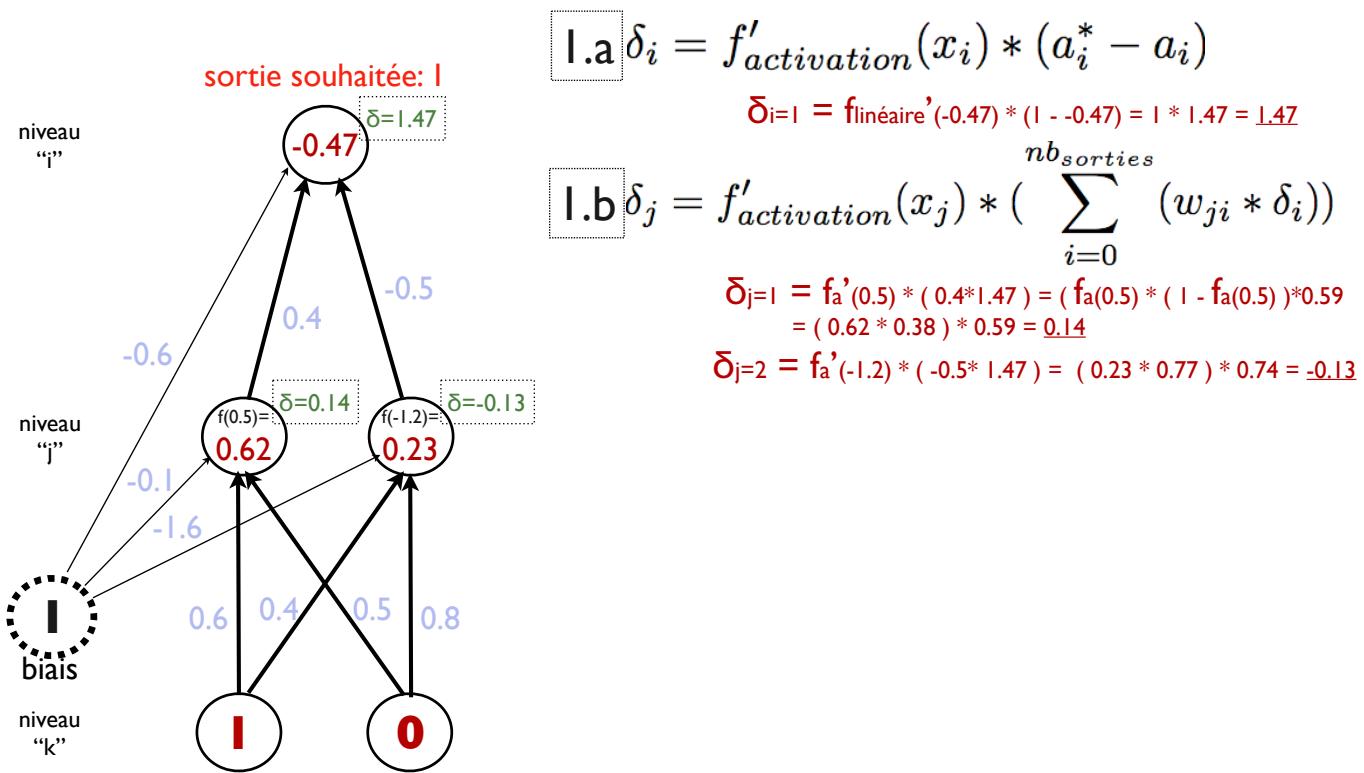
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{lin}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

Etape 2 : rétro-propagation de l'erreur et correction¹⁸



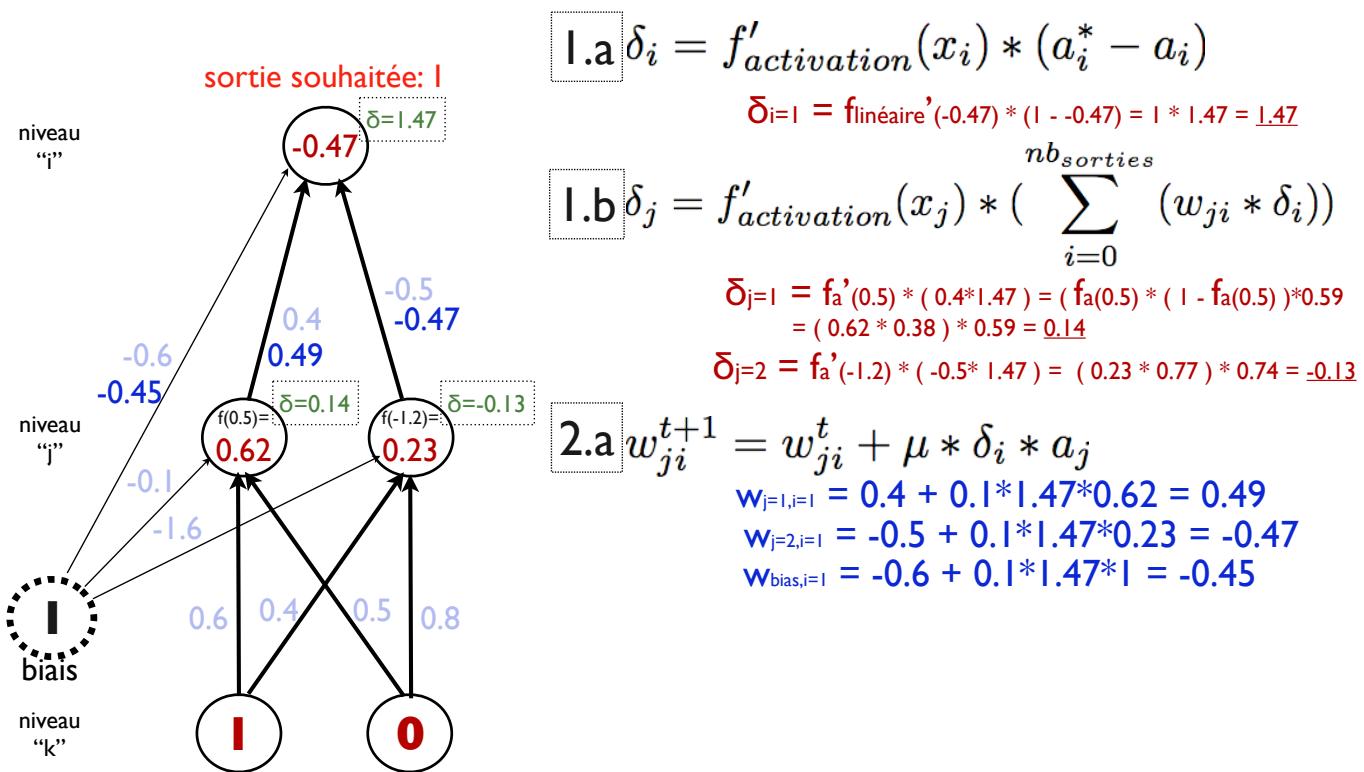
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{lin}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

Etape 2 : rétro-propagation de l'erreur et correction¹⁹



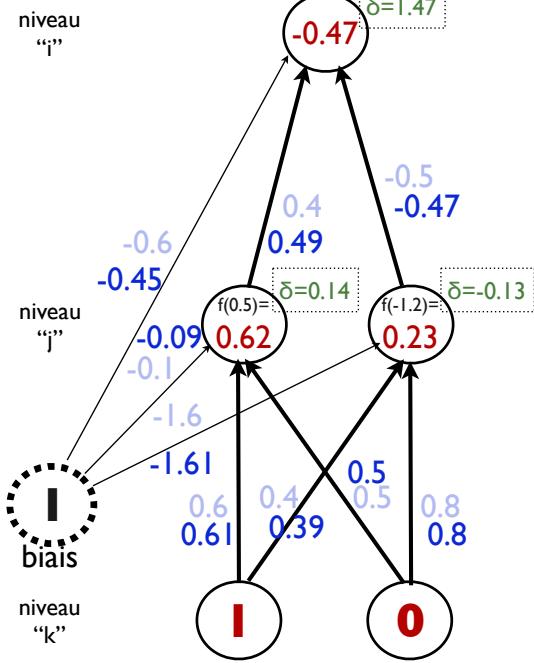
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a=f(x)$ -- $f'_{lin}(x)=1$ -- Ici, on fixe $\mu=0.1$ (pas d'apprentissage)

Etape 2 : rétro-propagation de l'erreur et correction²⁰



sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a=f(x)$ -- $f'_{lin}(x)=1$ -- Ici, on fixe $\mu=0.1$ (pas d'apprentissage)

Etape 2 : rétro-propagation de l'erreur et correction²¹



$$1.a \quad \delta_i = f'_{activation}(x_i) * (a^* - a_i)$$

$$\delta_{i=1} = f_{\text{linéaire}}'(-0.47) * (1 - -0.47) = 1 * 1.47 = 1.47$$

$$1.b \quad \delta_j = f'_{activation}(x_j) * \left(\sum_{i=0}^{nb_{\text{sorties}}} (w_{ji} * \delta_i) \right)$$

$$\begin{aligned} \delta_{j=1} &= f_a'(0.5) * (0.4 * 1.47) = (f_a(0.5) * (1 - f_a(0.5))) * 0.59 \\ &= (0.62 * 0.38) * 0.59 = 0.14 \end{aligned}$$

$$\delta_{j=2} = f_a'(-1.2) * (-0.5 * 1.47) = (0.23 * 0.77) * 0.74 = -0.13$$

$$2.a \quad w_{ji}^{t+1} = w_{ji}^t + \mu * \delta_i * a_j$$

$$w_{j=1,i=1} = 0.4 + 0.1 * 1.47 * 0.62 = 0.49$$

$$w_{j=2,i=1} = -0.5 + 0.1 * 1.47 * 0.23 = -0.47$$

$$w_{\text{bias},i=1} = -0.6 + 0.1 * 1.47 * 1 = -0.45$$

$$2.b \quad w_{kj}^{t+1} = w_{kj}^t + \mu * \delta_j * a_k$$

$$w_{k=1,j=1} = 0.6 + 0.1 * 0.14 * 1 = 0.61$$

$$w_{k=2,j=1} = 0.5 + 0.1 * 0.14 * 0 = 0.5$$

$$w_{\text{bias},j=1} = -0.1 + 0.1 * 0.14 * 1 = -0.09$$

$$w_{k=1,j=2} = 0.4 + 0.1 * -0.13 * 1 = 0.39$$

$$w_{k=2,j=2} = 0.8 + 0.1 * -0.13 * 0 = 0.8$$

$$w_{\text{bias},j=2} = -1.6 + 0.1 * -0.13 * 1 = -1.61$$

sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{\text{lin}}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

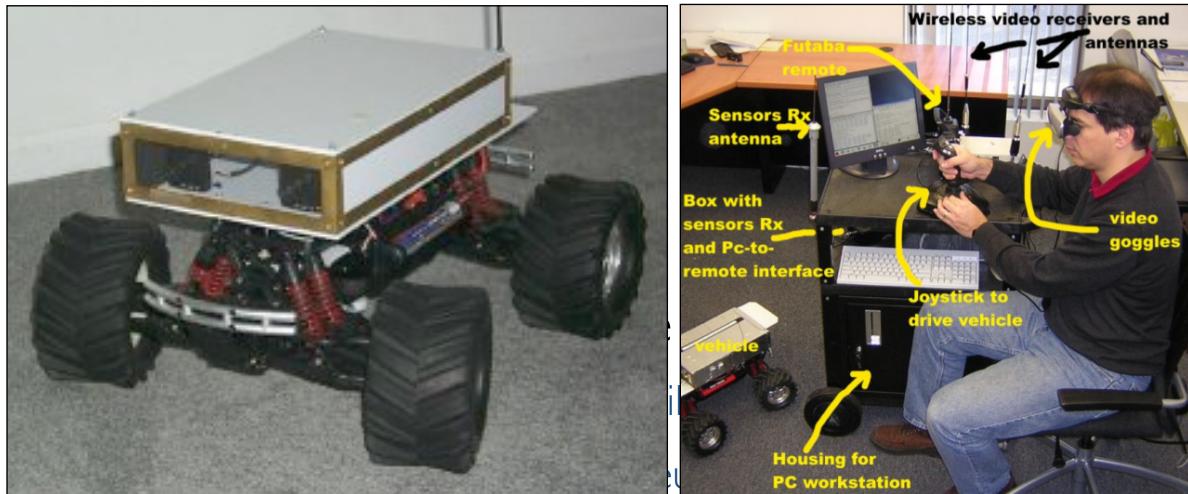
Etude de cas

Apprentissage supervisé et réseau convolutionnel appliqué à la reconnaissance de caractères et à la robotique autonome

“Dave”: autonomous off-road vehicle

23

[LeCun et al., 2003-2004]



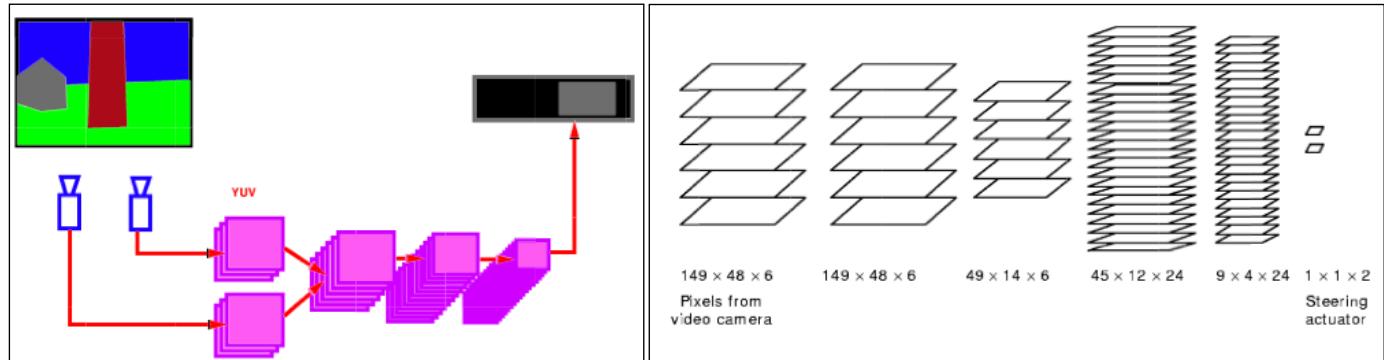
source: <http://yann.lecun.com/>

24



données d'apprentissage (exemple)

source: DARPA Final technical report. 2004. <http://yann.lecun.com/>



- Problème d'Apprentissage

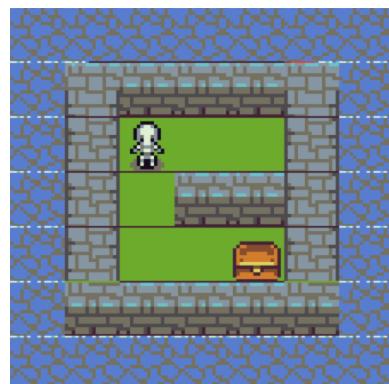
- données générées pendant pilotage par un humain
- apprendre la fonction: $f(\text{senseurs}) \Rightarrow \text{actionneurs}$

source: <http://yann.lecun.com/>

Apprentissage par renforcement

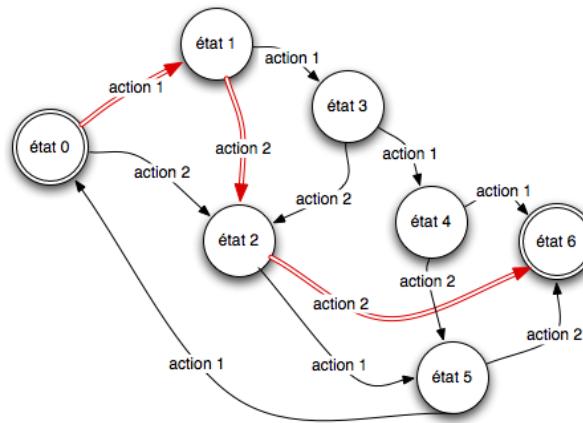
● Apprentissage par renforcement

- Description du problème
 - ▶ Problème nécessitant une séquence d'actions
 - ▶ Récompense retardée p/r aux actions ("credit assignment problem")
- Objectif: découvrir une politique pour la prise de décision
- Applications: contrôle (en robotique), publicité sur internet, recommandation, ...



● Objectif:

- trouver la politique optimale qui maximise la récompense globale
- on veut $\forall s_t, \pi^*(s_t) \rightarrow a_t^*$ avec un espérance de gain max.
- Notations:
 - a_t^* action optimale à t ; ($a_t^* \in A$)
 - s_t état courant à t ; ($s_t \in S$)



- On souhaite estimer la fonction de valeur V et trouver la politique π
- Difficultés: exploration vs. exploitation, temporal credit-assignment problem

Q-learning

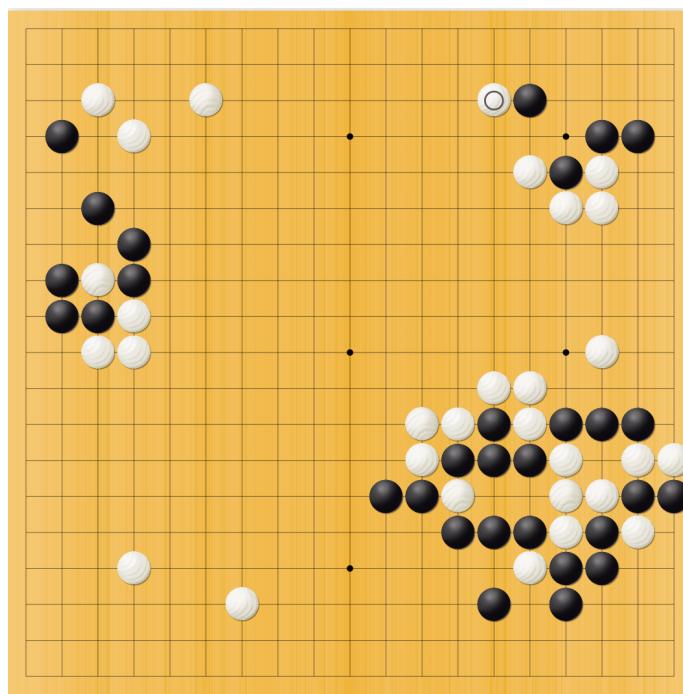
$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\underbrace{R_{t+1} + \gamma}_{\text{reward}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\substack{\text{learned value} \\ \text{estimate of optimal future value}}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right)$$

- Politique:
 - greedy, epsilon-greedy, softmax

Etude de cas

AlphaGO

32



doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver^{1*}, Aja Huang^{1*}, Chris J. Maddison¹, Arthur Guez¹, Laurent Sifre¹, George van den Driessche¹, Julian Schrittwieser¹, Ioannis Antonoglou¹, Veda Panneershelvam¹, Marc Lanctot¹, Sander Dieleman¹, Dominik Grewe¹, John Nham², Nal Kalchbrenner¹, Ilya Sutskever², Timothy Lillicrap¹, Madeleine Leach¹, Koray Kavukcuoglu¹, Thore Graepel¹ & Demis Hassabis¹

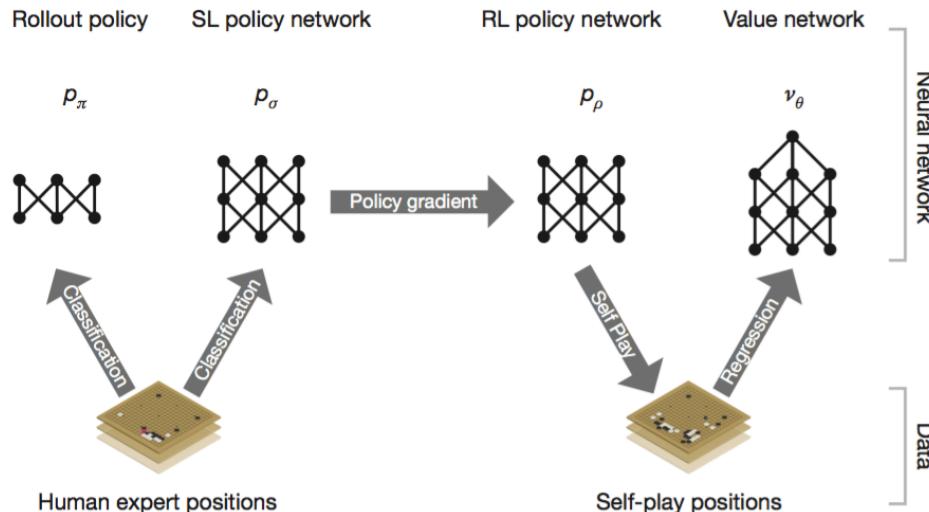
Apprentissage supervisé à partir de parties jouées par des humains
une méthode rollout complète l'apprentissage supervisé.

=> apprend à prédire le jeu de l'expert

Apprentissage par renforcement contre lui-même ("self-play")

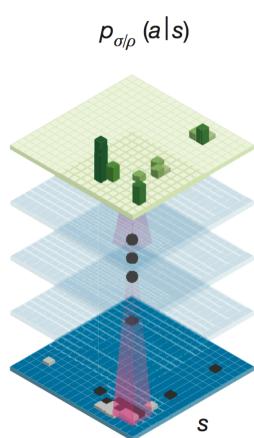
=> améliore la stratégie (policy network)

=> apprend à estimer la qualité d'une position (value network)

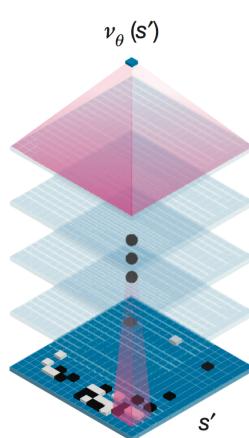


Silver et al. (2015)

Policy network



Value network



Détails:

1. policy network: à partir de l'état du jeu, donne une distribution de probabilité pour le prochain coup
2. value network: idem, mais donne une estimation de la qualité du jeu pr à la probabilité de gagner

Policy network:

13 couches

apprentissage: (1) appr. supervisé sur des parties existantes (2) appr. par renf. sur des self-play

Value network:

entrée: 19x19x48 (48 features) + 1 entrée donnant la couleur du joueur

13 couches + dernière couche avec un seul neurone (fonction d'activation tanh)

apprentissage: regression sur les self-play

Silver et al. (2015)

Conclusions

Synthèse

36

- Ce que l'on a vu
 - Apprentissage supervisé: MLP, rétro-propagation
 - Apprentissage par renforcement: Q-learning
- A retenir
 - Plusieurs types de problème d'apprentissage...
 - ...auxquels correspondent à chaque fois des méthodes dédiées

Fin du cours