

Apprentissage Automatique

Apprentissage supervisé

Apprentissage non-supervisé

Apprentissage par renforcement

Nicolas Bredèche

Professeur des Universités, UPMC

nicolas.bredèche@upmc.fr

UE IARO - L3 - UPMC - Mise à jour: 30 mars 2017

Apprentissage supervisé

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

Définition

- **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:

$$\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \begin{matrix} \text{classification} \\ \{-1, 1\} \end{matrix} \text{ or } \mathbf{R}^{\text{regression}}$$

- Une fonction de perte:

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

- Objectif:

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

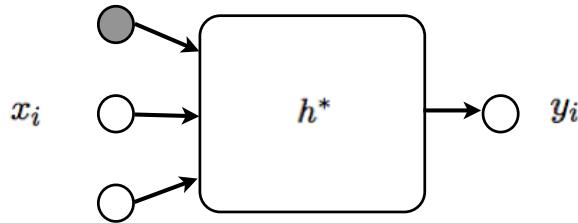
$$h^* : \mathcal{X} \mapsto \mathcal{Y}$$

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

Hypothèse cherchée

5

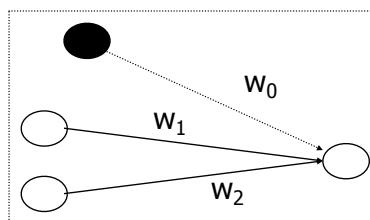
$$\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \{-1, 1\} \text{ or } \mathbf{R}$$



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

Fonctionnement général

6



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

- 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 liant 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

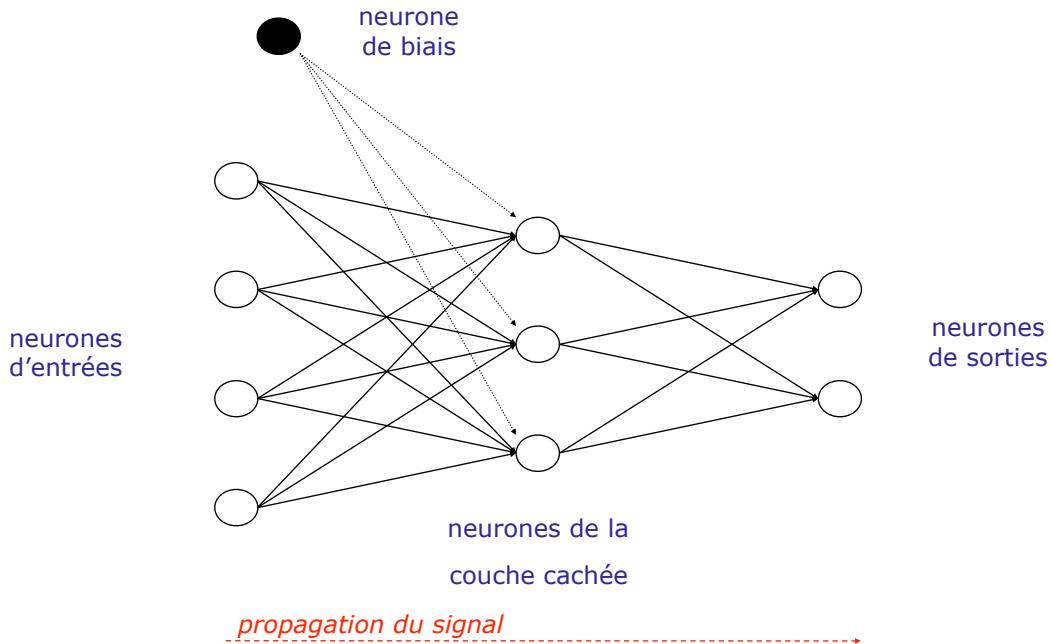
- Solution: ajouter une couche cachée!

- 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

10

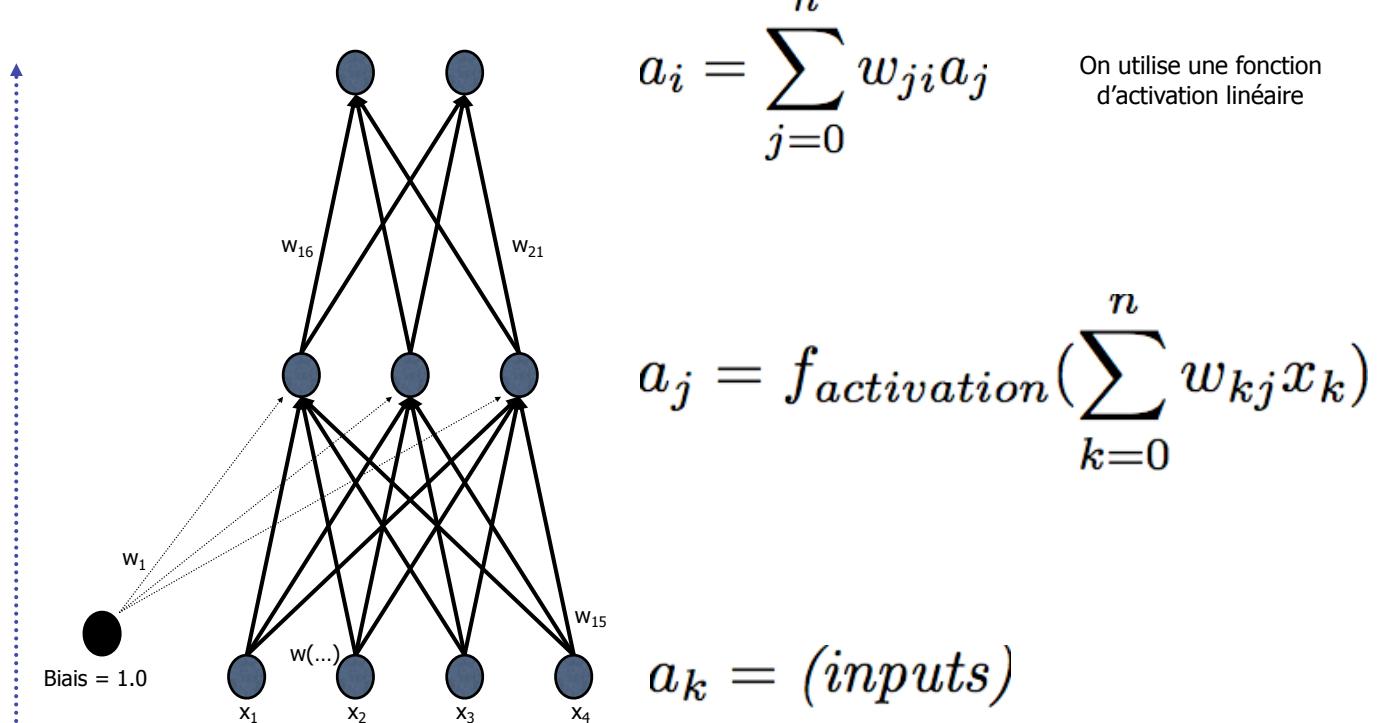
Perceptron Multi-Couches (MLP)

[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
 - 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.

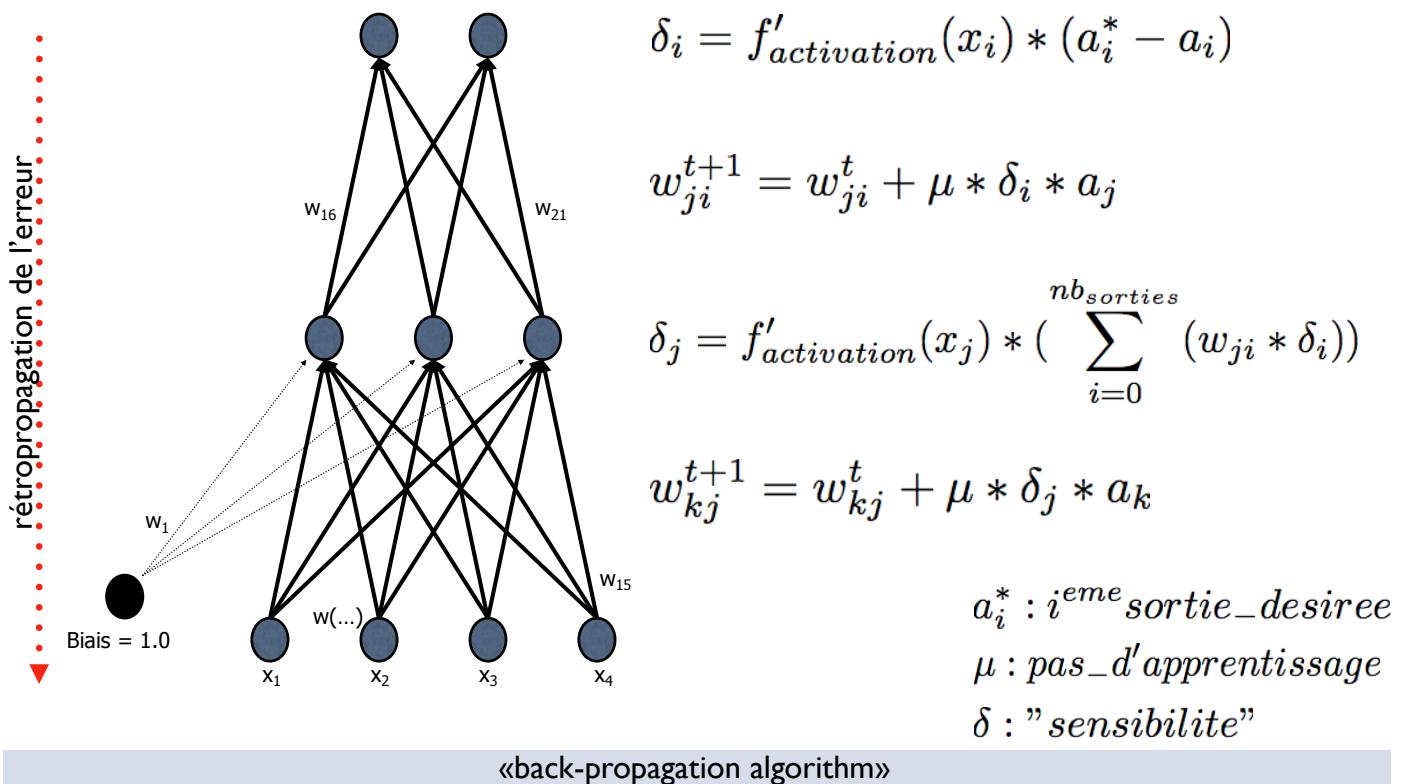
Propagation du signal dans un MLP

11

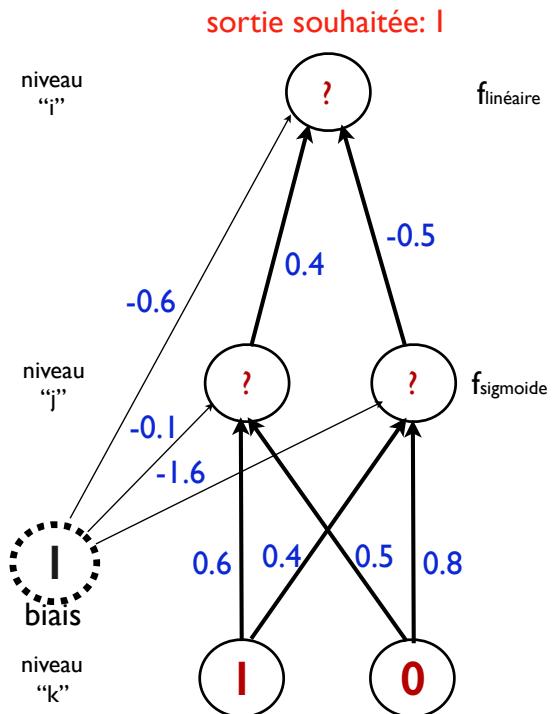


Rétro-propagation de l'erreur

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.

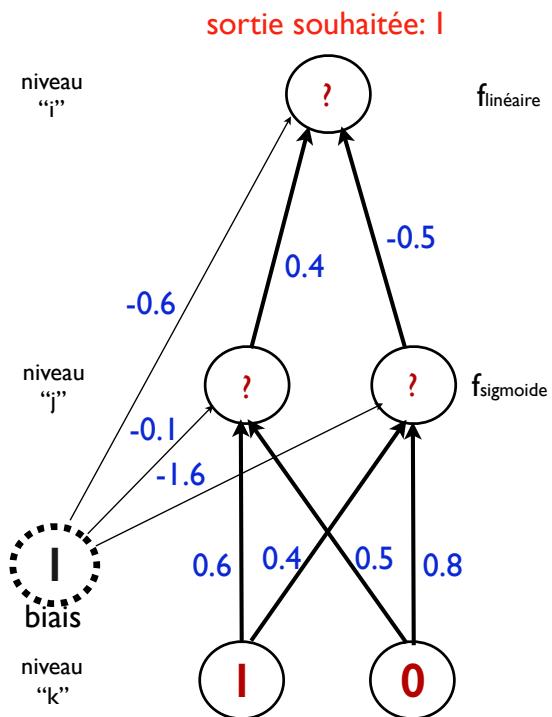
Etude pratique:

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



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

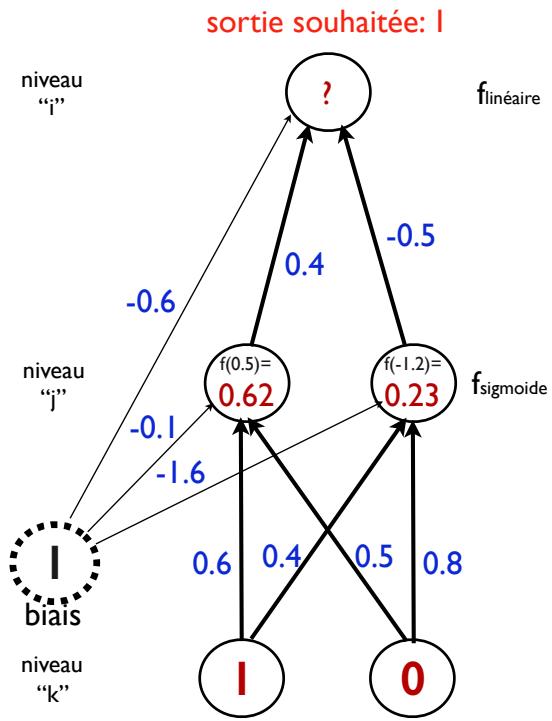
$$a_{k=1} = 1$$

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

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

Etape I : propagation du signal

15



$$a_j = f_{\text{activation}} \left(\sum_{k=0}^n w_{kj} x_k \right)$$

$$a_{j=1} = f_{\text{sig}}(0.6*1 + 0.5*0 + -0.1) = f_{\text{sig}}(0.5) = 0.62$$

$$a_{j=2} = f_{\text{sig}}(0.4*1 + 0.8*0 + -1.6) = f_{\text{sig}}(-1.2) = 0.23$$

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

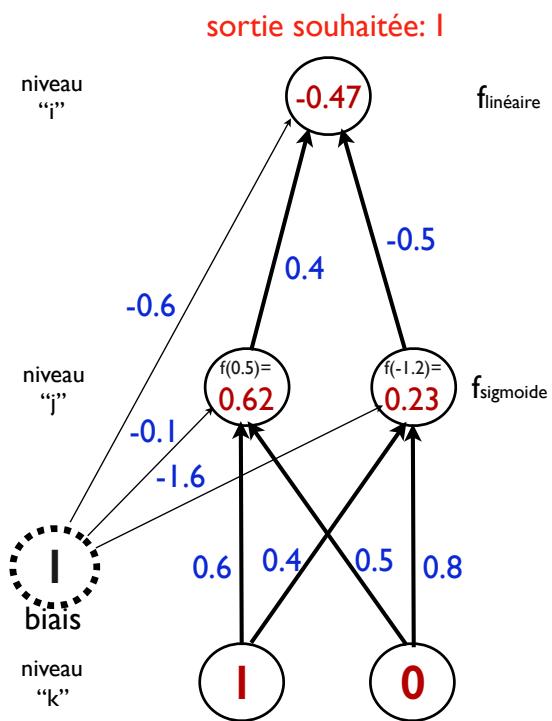
$$a_{k=1} = 1$$

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

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

Etape I : propagation du signal

16



$$a_i = \sum_{j=0}^n w_{ji} a_j$$

$$a_{i=1} = 0.4*0.62 + -0.5*0.23 + -0.6 = -0.47$$

$$a_j = f_{\text{activation}} \left(\sum_{k=0}^n w_{kj} x_k \right)$$

$$a_{j=1} = f_{\text{sig}}(0.6*1 + 0.5*0 + -0.1) = f_{\text{sig}}(0.5) = 0.62$$

$$a_{j=2} = f_{\text{sig}}(0.4*1 + 0.8*0 + -1.6) = f_{\text{sig}}(-1.2) = 0.23$$

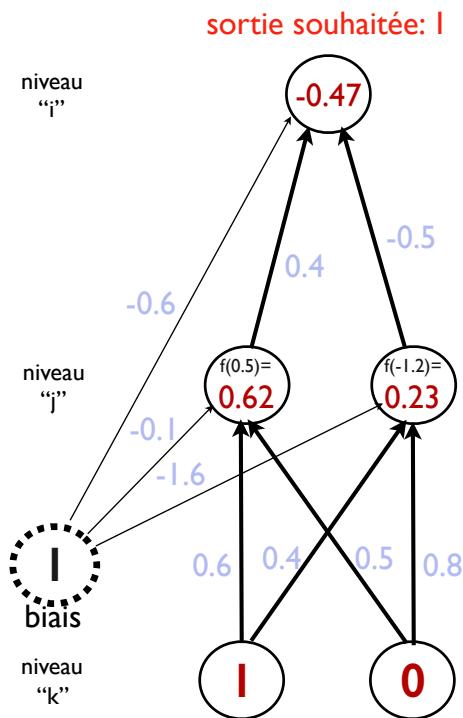
$$a_k = (\text{inputs})$$

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

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

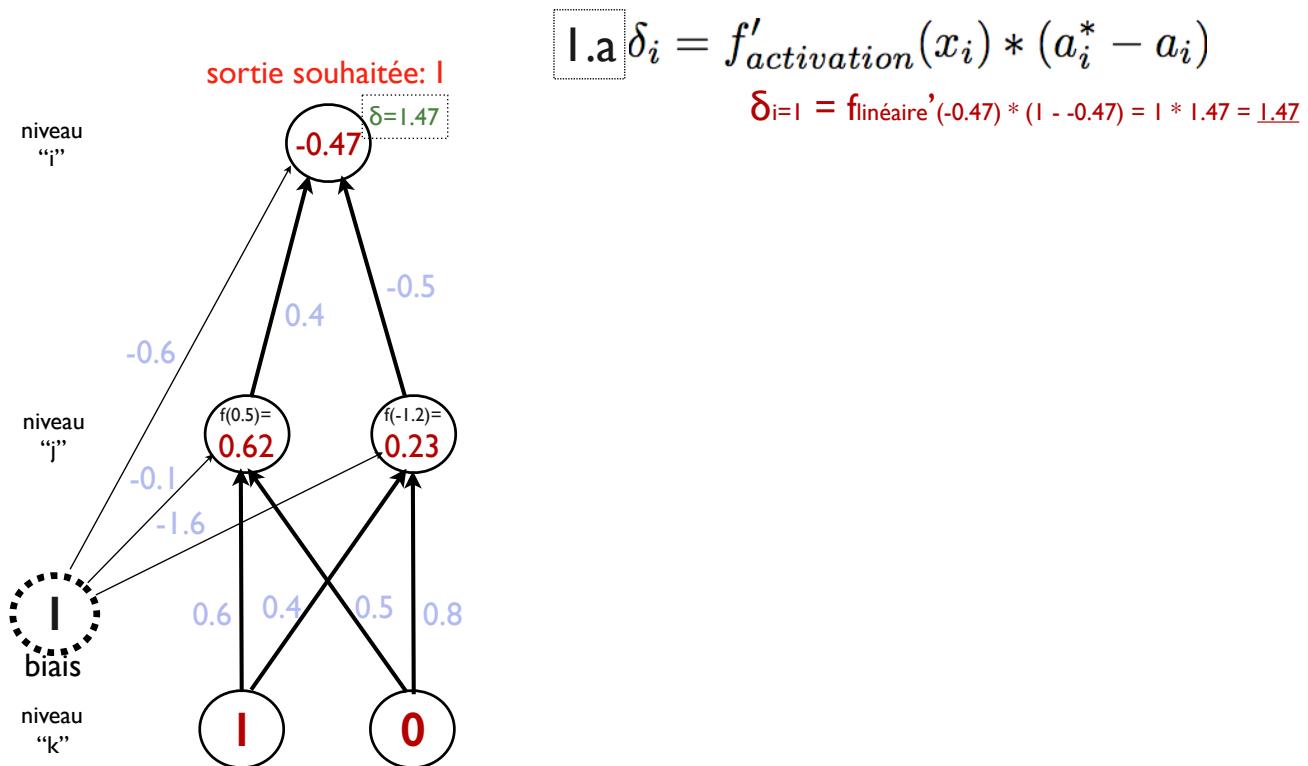
activation: sigmoïde $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

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



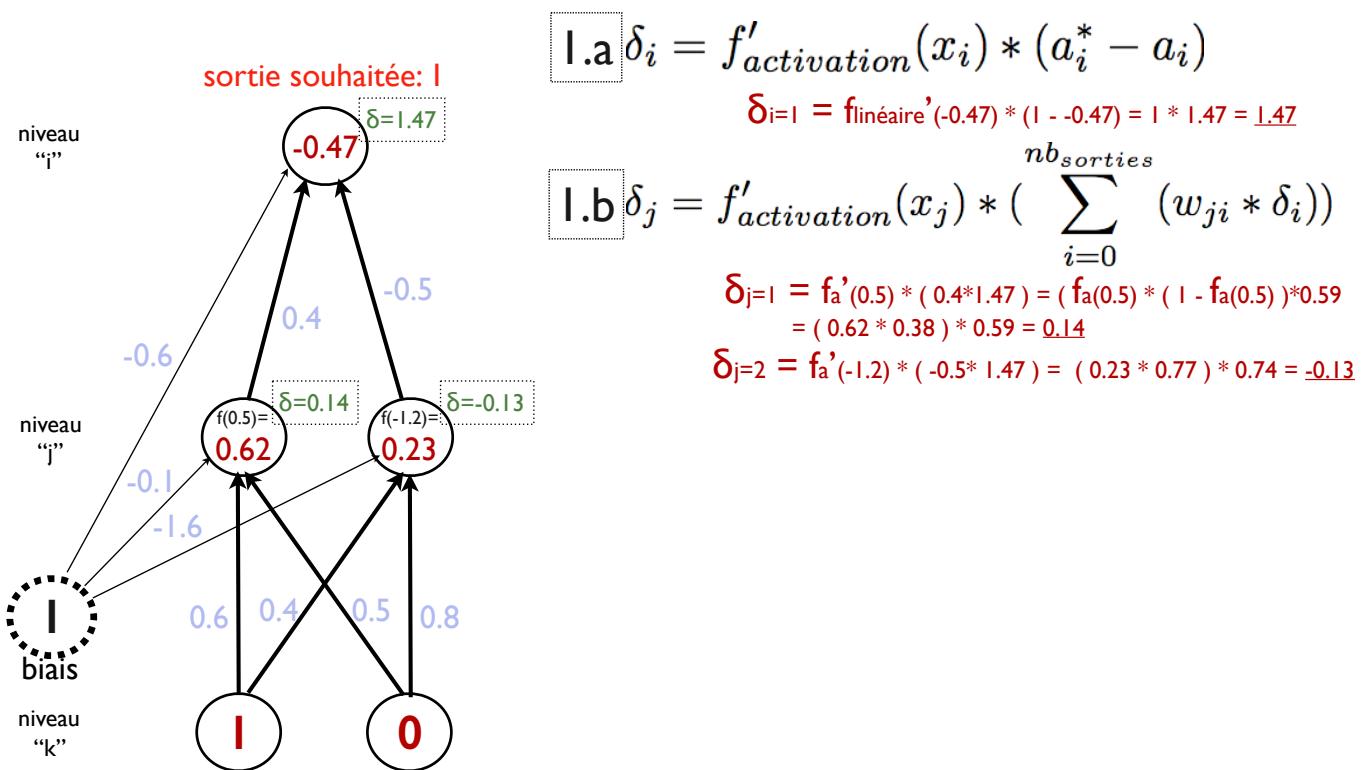
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{lin}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

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



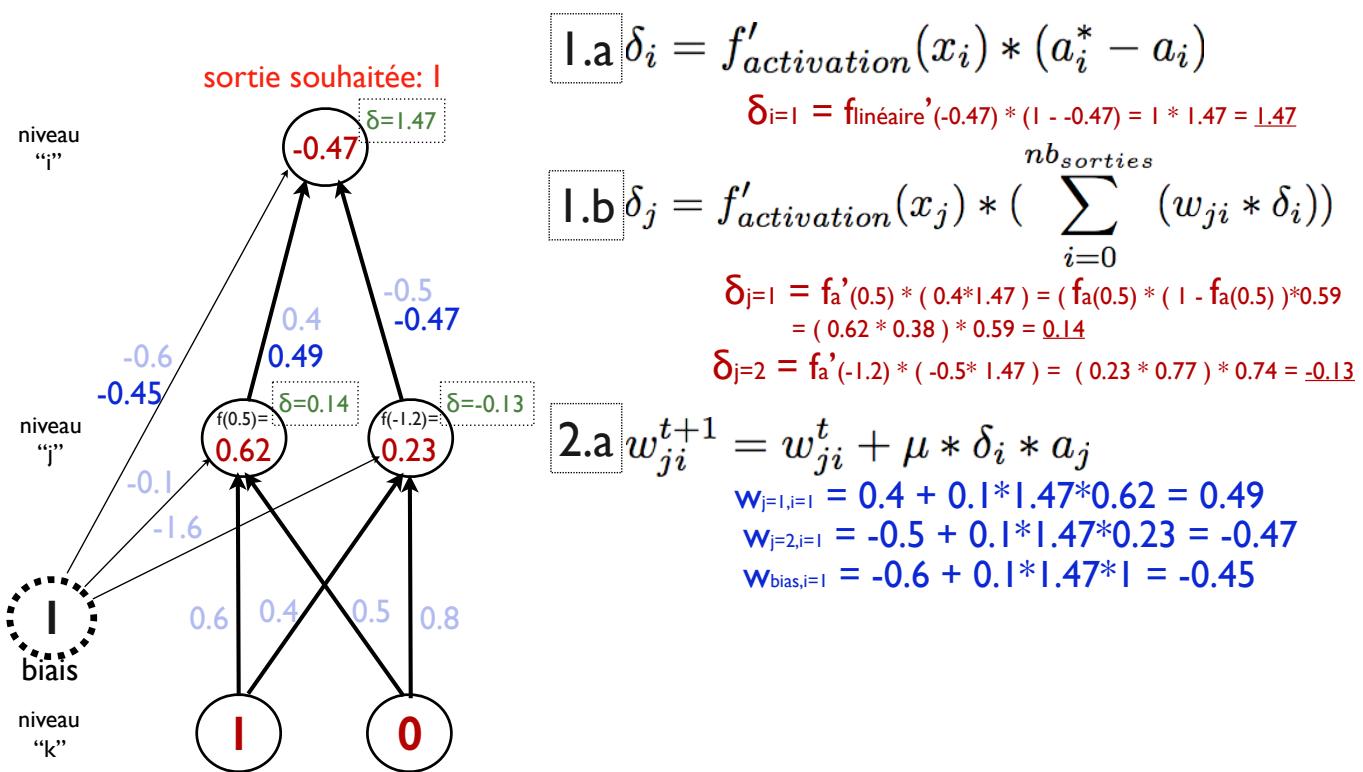
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{lin}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

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



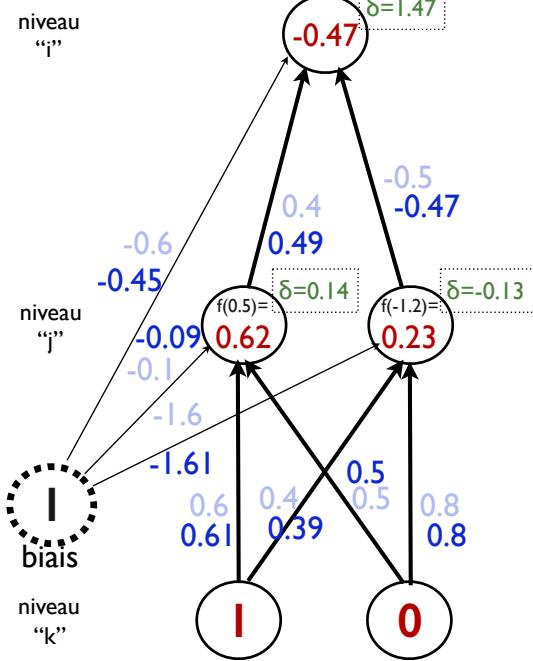
sigmoïde: $f'(x) = f(x)(1 - f(x))$ -- rappel: $a = f(x)$ -- $f'_{lin}(x) = 1$ -- Ici, on fixe $\mu = 0.1$ (pas d'apprentissage)

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 $\delta_i = f'_{activation}(x_i) * (a_i^* - a_i)$
 $\delta_{i=1} = f_{linéaire}'(-0.47) * (1 - -0.47) = 1 * 1.47 = 1.47$

1.b $\delta_j = f'_{activation}(x_j) * \left(\sum_{i=0}^{nb_{sorties}} (w_{ji} * \delta_i) \right)$
 $\delta_{j=1} = f_a'(0.5) * (0.4 * 1.47) = (f_a(0.5) * (1 - f_a(0.5))) * 0.59 = (0.62 * 0.38) * 0.59 = 0.14$
 $\delta_{j=2} = f_a'(-1.2) * (-0.5 * 1.47) = (0.23 * 0.77) * 0.74 = -0.13$

2.a $w_{ji}^{t+1} = w_{ji}^t + \mu * \delta_i * a_j$
 $w_{j=1,i=1} = 0.4 + 0.1 * 1.47 * 0.62 = 0.49$
 $w_{j=2,i=1} = -0.5 + 0.1 * 1.47 * 0.23 = -0.47$
 $w_{bias,i=1} = -0.6 + 0.1 * 1.47 * 1 = -0.45$

2.b $w_{kj}^{t+1} = w_{kj}^t + \mu * \delta_j * a_k$
 $w_{k=1,j=1} = 0.6 + 0.1 * 0.14 * 1 = 0.61$
 $w_{k=2,j=1} = 0.5 + 0.1 * 0.14 * 0 = 0.5$
 $w_{bias,j=1} = -0.1 + 0.1 * 0.14 * 1 = -0.09$
 $w_{k=1,j=2} = 0.4 + 0.1 * -0.13 * 1 = 0.39$
 $w_{k=2,j=2} = 0.8 + 0.1 * -0.13 * 0 = 0.8$
 $w_{bias,j=2} = -1.6 + 0.1 * -0.13 * 1 = -1.61$

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

Protocole expérimental

22

- Protocole d'apprentissage
 - Les exemples sont présentés plusieurs fois
 - ▶ un +/- grand nombre de fois avant convergence
 - Méthode “on-line”:
 - tirage au hasard d'un exemple, on calcule une erreur locale
 - application de l'algorithme de rétro-propagation (ie. pour chaque exemple)
 - Méthode “batch”:
 - on passe tous les exemples, on calcule une erreur globale
 - application de l'algorithme de rétro-propagation (ie. une seule fois pas passage de la base)

Critères de succès

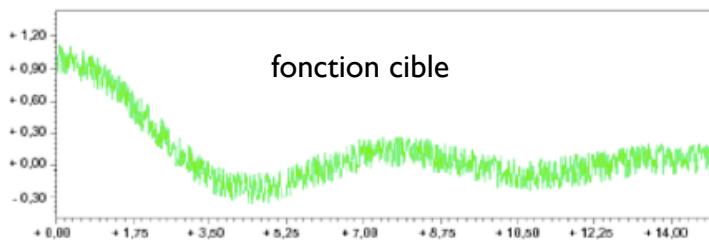


Figure 1-10. Un signal que l'on voudrait modéliser

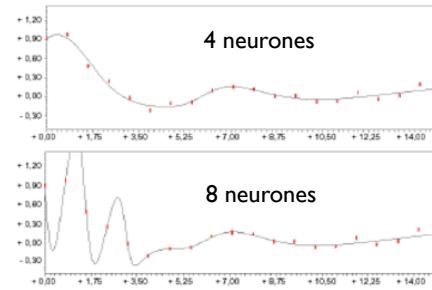


Figure 1-14
Toutes choses égales par ailleurs, le réseau de neurones le plus performant possède les meilleures propriétés de généralisation.

- Compromis mémorisation vs. généralisation
 - mémorisation: approximation +/- grossière des exemples
 - généralisation: risque de sur-apprentissage (= apprendre par coeur)
- Méthodologie:
 - Base d'apprentissage : exemples utilisés pour l'apprentissage
 - Base de test : exemples utilisés pour la validation (ex: 10-fold cross-validation)

image: source transp. A. Cornuejols

Etude de cas

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

exemple : Les codes postaux

25

- Lecun et al. 1989

- Objectif
 - Identifier les codes postaux sur les enveloppes
- Méthode
 - Réseaux de neurones + retro-propagation
- Problématique :
 - Comment entraîner un tel réseau? (trop de poids)
- Approche:
 - entraîner des détecteurs de caractéristiques particulières

80322-4129 80206

40004 14310

37878 05153

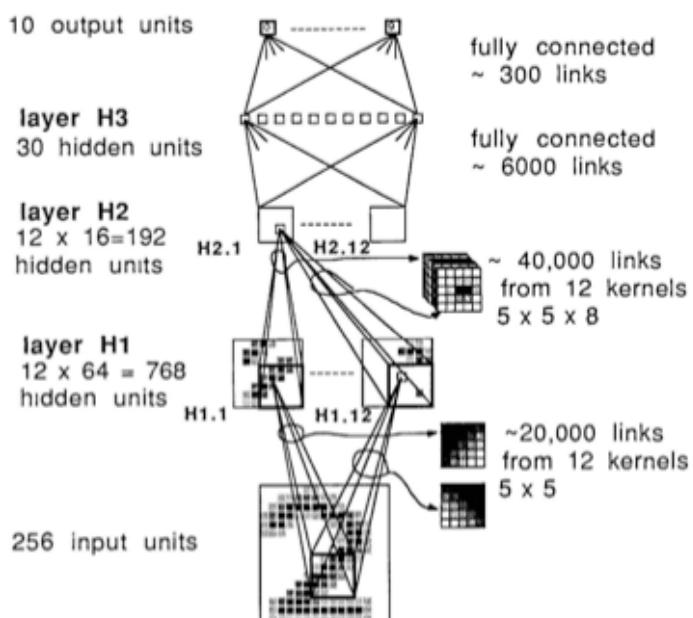
5502 75216

35460 44209

1011913485726803224414186
4359720299299722510046701
3084111591010615406103631
1064111030475262009979966
8912054708557131427955460
2011730187112991089970984
0109707597331972015519055
1075518255182814358090943
1787541655468554603546055
18255108503047520439401

Réseau convolutionnel

26



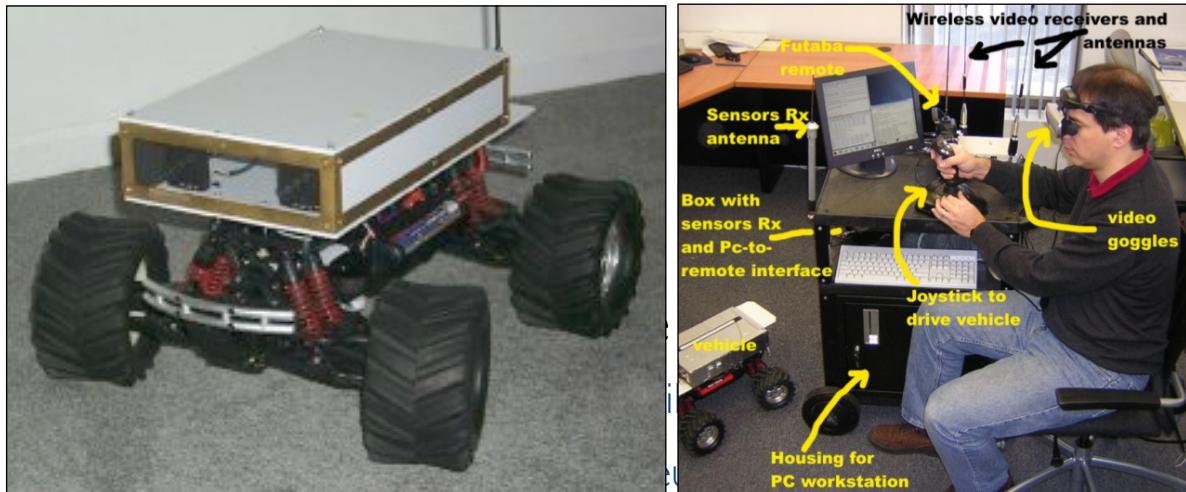
9760 poids (au lieu de 200000 poids si on connecte tout)

7300 exemples d'apprentissage, 2000 exemples de tests, 99% de précision.

“Dave”: autonomous off-road vehicle

27

[LeCun et al., 2003-2004]



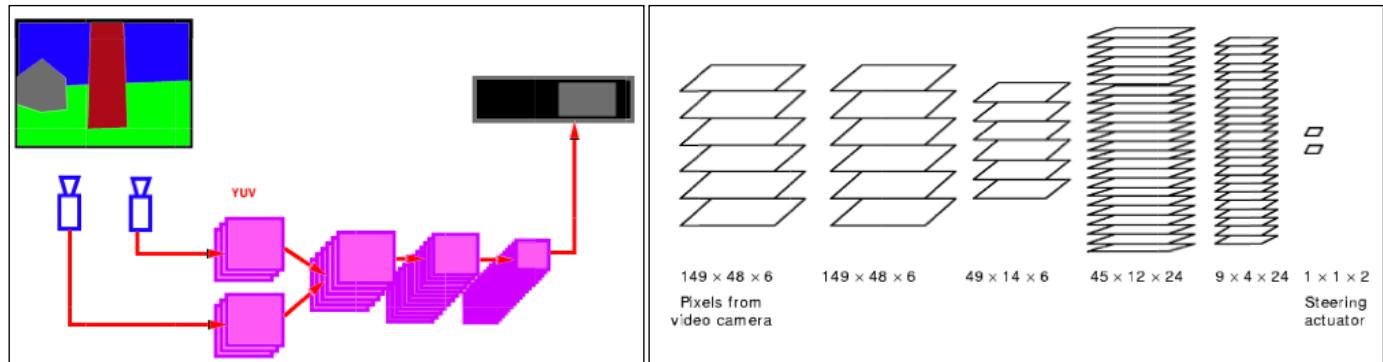
source: <http://yann.lecun.com/>

28



données d'apprentissage (exemple)

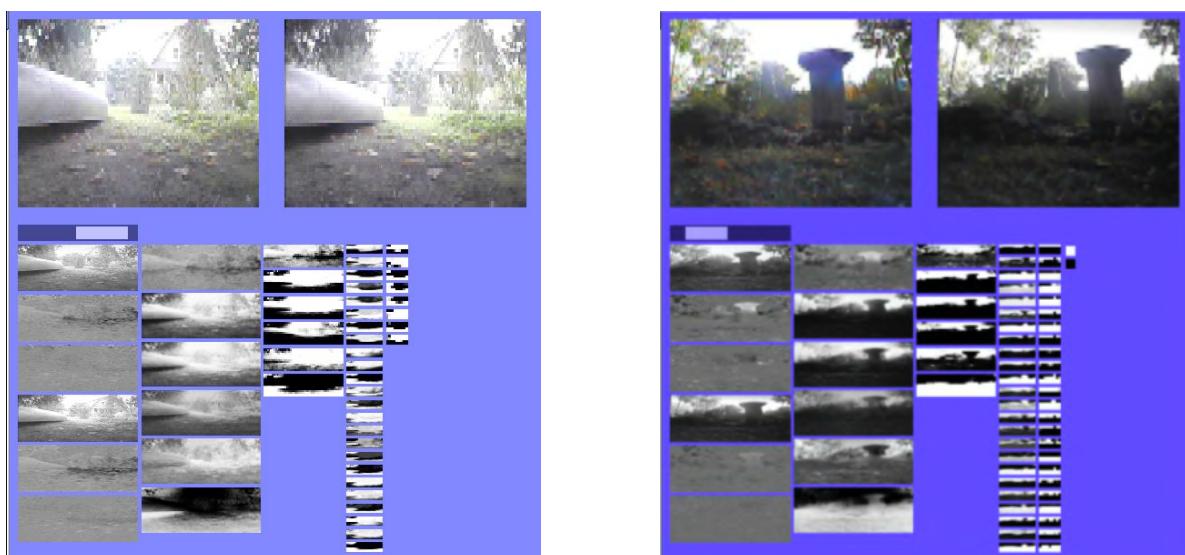
source: DARPA Final technical report. 2004. <http://yann.lecun.com/>



- Problème d'Apprentissage

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

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



traitement et décision (2 exemples)

Apprentissage non-supervisé

Définition

32

- **Apprentissage non supervisé**

- On ne dispose que de données brutes
- On souhaite organiser; stocker ou préparer ces données
 - ▶ organiser: construire des catégories
 - ▶ stocker: imiter la mémoire et le mécanisme de rappel
 - ▶ préparer: reformuler les données pour une utilisation ultérieure
- Applications: découvrir des groupes naturels, faciliter la visualisation, changer de représentation, etc.

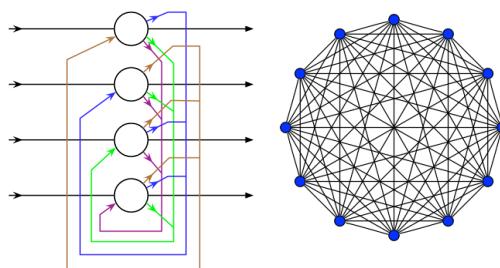
- Règle d'apprentissage non supervisée (règle de Hebb)
 - Renforce les connections
 - ▶ entre neurones proches,
 - ▶ actifs au même moment
- Mise à jour des poids:

$$\Delta W_{ij}(t) = F(x_j, x_i, \gamma, t, \theta)$$

(simplification: $\Delta W_{ij}(t) = \gamma * x_j * x_i$)

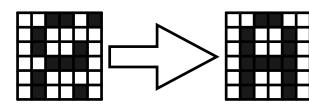
- Remarques:
 - Mécanisme d'association (renforcement stimulus-réponse)

Réseaux de Hopfield [1982]

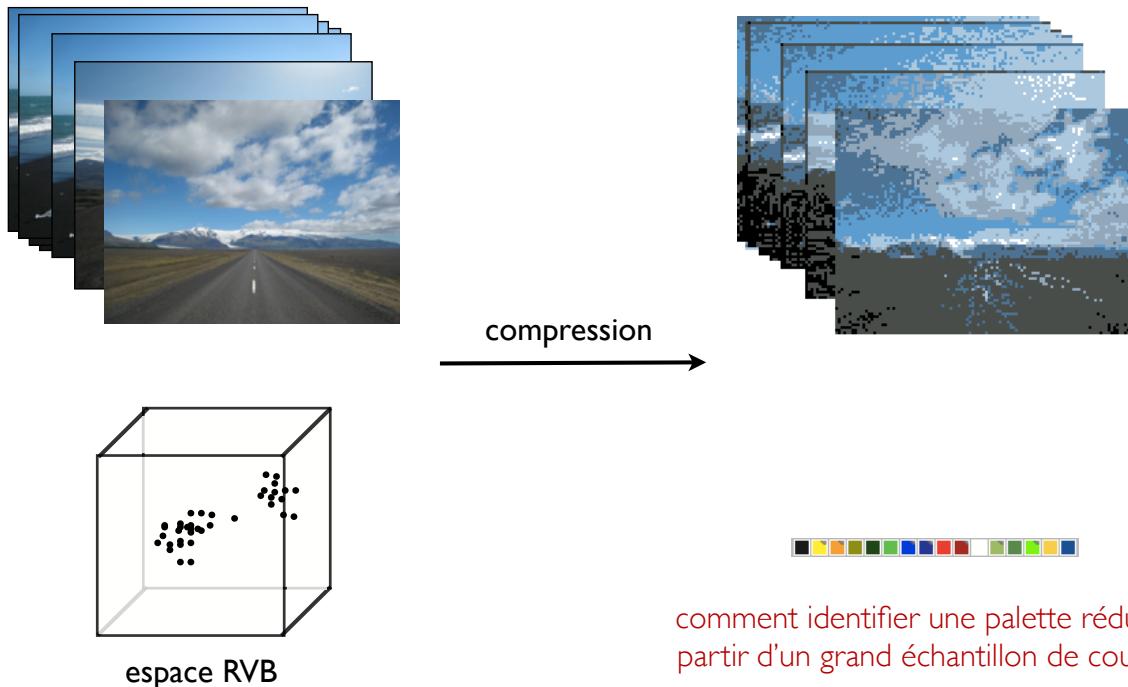


réseaux de neurones (souvent binaire) complètement connecté (connexions excitatrice ou non, asymétrique)

- Mémoire associative
 - stockage d'un motif (ie. encode un attracteur)
 - rappel du motif à partir d'une amorce (ie. convergence)
- Applications
 - Etude de la mémoire associative
 - Reconnaissance/identification d'objets/lettres/scènes, reconstruction de parties manquantes, etc.



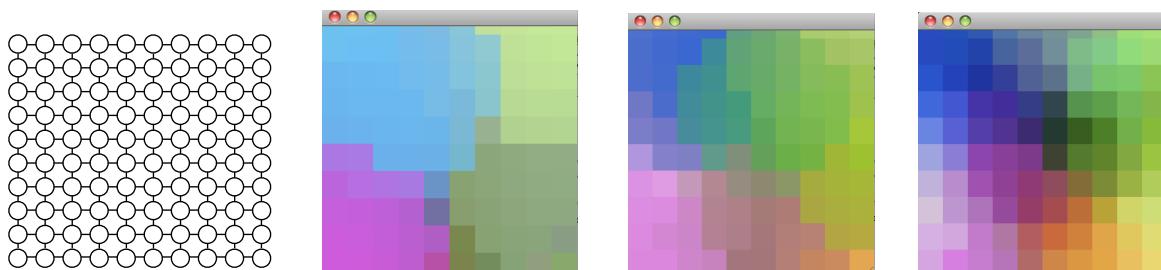
“Self-Organizing Maps” ou “Kohonen Maps”



comment identifier une palette réduite à partir d'un grand échantillon de couleur?

Carte Auto-Organisatrice

“Self-Organizing Maps” ou “Kohonen Maps”



exemple: compression de données RVB sur 100 valeurs — chaque neurone code 3 valeurs (R,V,B de référence)

● Principes

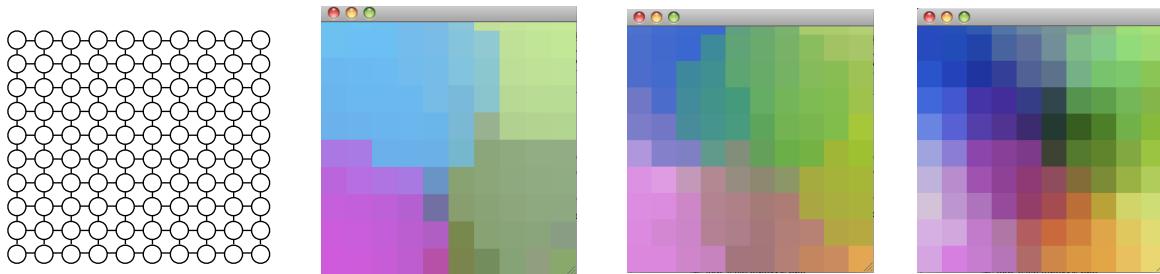
- Neurones avec états internes (= #dimensions du problème) sur une grille 2D
- Pour chaque nouvelle entrée, on cherche le neurone le plus proche
- Le lien entre ce neurone (et ses voisins) et l'entrée est renforcé

● Objectif

- regroupement de données proches (auto-organisation)
- applications: compression, visualisation, etc.

Carte Auto-Organisatrice

“Self-Organizing Maps” ou “Kohonen Maps”



exemple: compression de données RVB sur 100 valeurs — chaque neurone code 3 valeurs (R,V,B de référence)

● Algorithme

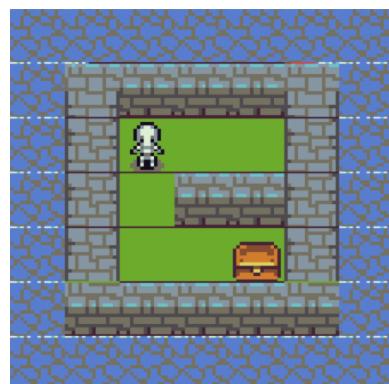
- Pour chaque entrée on calcule le BMU (« best matching unit »)
- Mise à jour des valeurs de chaque unité: $\mathbf{Wv}(t + 1) = \mathbf{Wv}(t) + \Theta(v, t)\alpha(t)(\mathbf{D}(t) - \mathbf{Wv}(t))$
 - ▶ alpha: taux d'apprentissage (décroissance monotone dans le temps)
 - ▶ omega: pondère par la distance p/r au « Best Matching Unit »
 - ▶ D(t): valeurs de l'entrée actuellement traitée
 - ▶ Wv(t): valeur de l'unité neuronale à mettre à jour

[Kohonen, 1988]

Apprentissage par renforcement

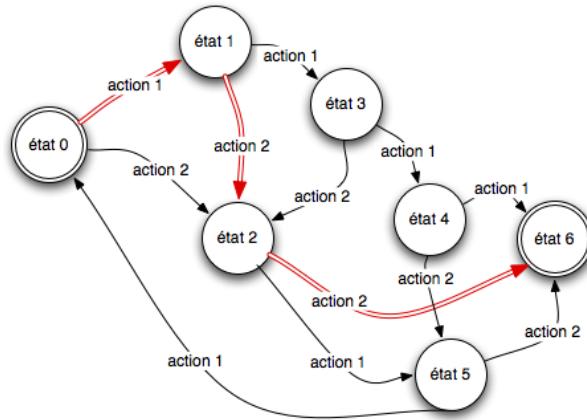
- **Apprentissage par renforcement**

- 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

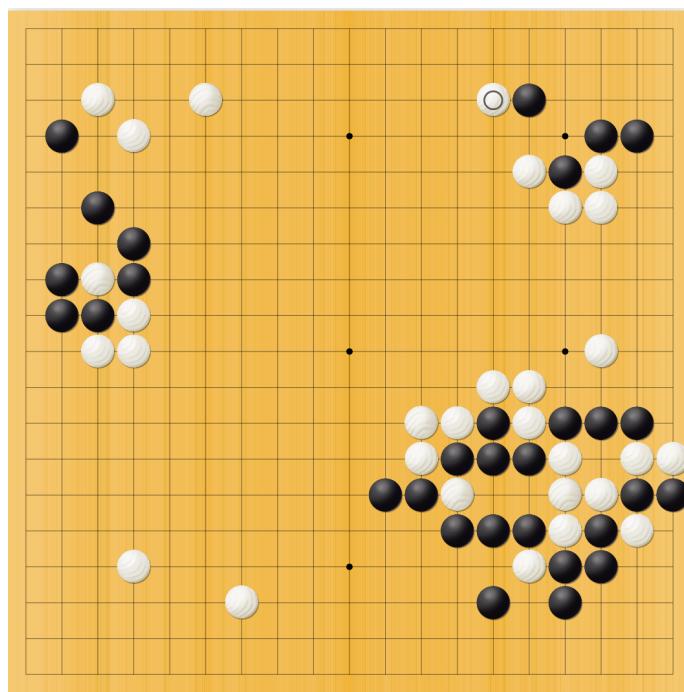
$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\underbrace{R_{t+1} + \gamma}_{\substack{\text{reward} \\ \text{discount factor}}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\substack{\text{learned value} \\ \text{estimate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

- Politique:
 - greedy, epsilon-greedy, softmax

Etude de cas

AlphaGO

44



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

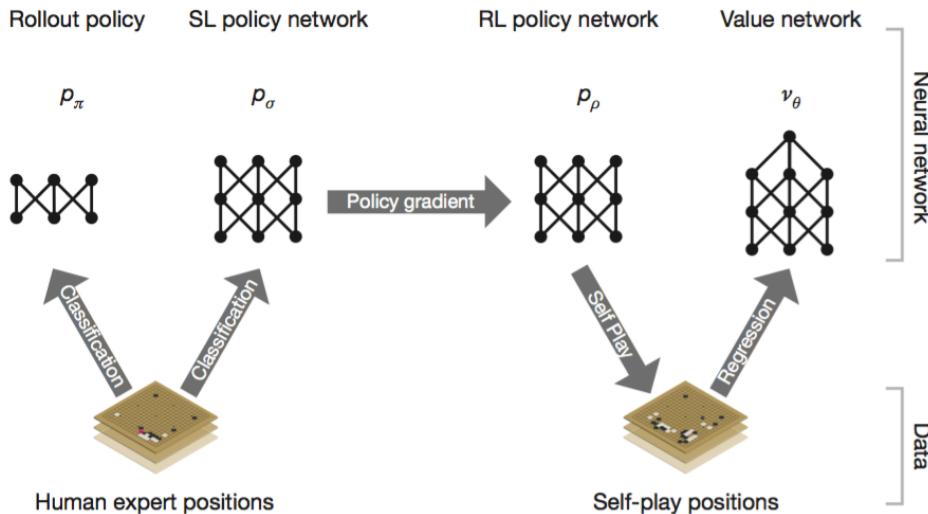
Apprentissage supervisé à partir de parties jouées par des humains une méthode rollout complète l'apprentissage supervisé.

=> 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)

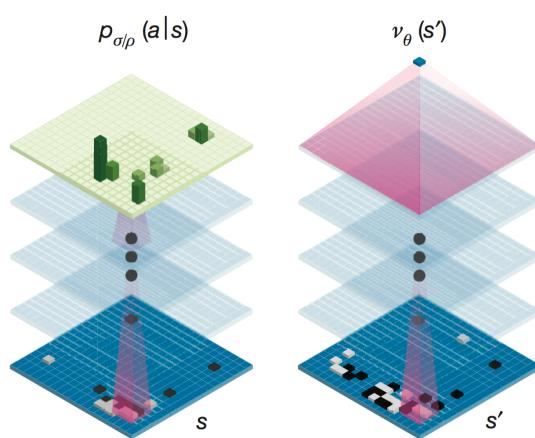


Silver et al. (2015)

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 p/r 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: $19 \times 19 \times 48$ (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

48

- Ce que l'on a vu
 - Apprentissage supervisé: MLP, rétro-propagation
 - Apprentissage non-supervisé: carte auto-organisatrice
 - Apprentissage par renforcement: Q-learning
 - ▶ Pas vu dans ce cours: recherche directe de politique (cf. cours précédent!)
- A retenir
 - Plusieurs types de problème d'apprentissage...
 - ...auxquels correspondent à chaque fois des méthodes dédiées

Fin du cours