INTRODUCTION AUX RESEAUX DE NEURONES

Pr. Fabien Moutarde
Center for Robotics (CAOR)
MINES ParisTech
PSL Research University

Fabien.Moutarde@mines-paristech.fr

http://people.mines-paristech.fr/fabien.moutarde

HISTORIQUE

- **Compréhension et modélisation cerveau**

↓

- **Essai imitation pour reproduire fonctions évoluées**

↓

- **Outils mathématiques pour l'ingénieur**

DOMAINES D'APPLICATIONS

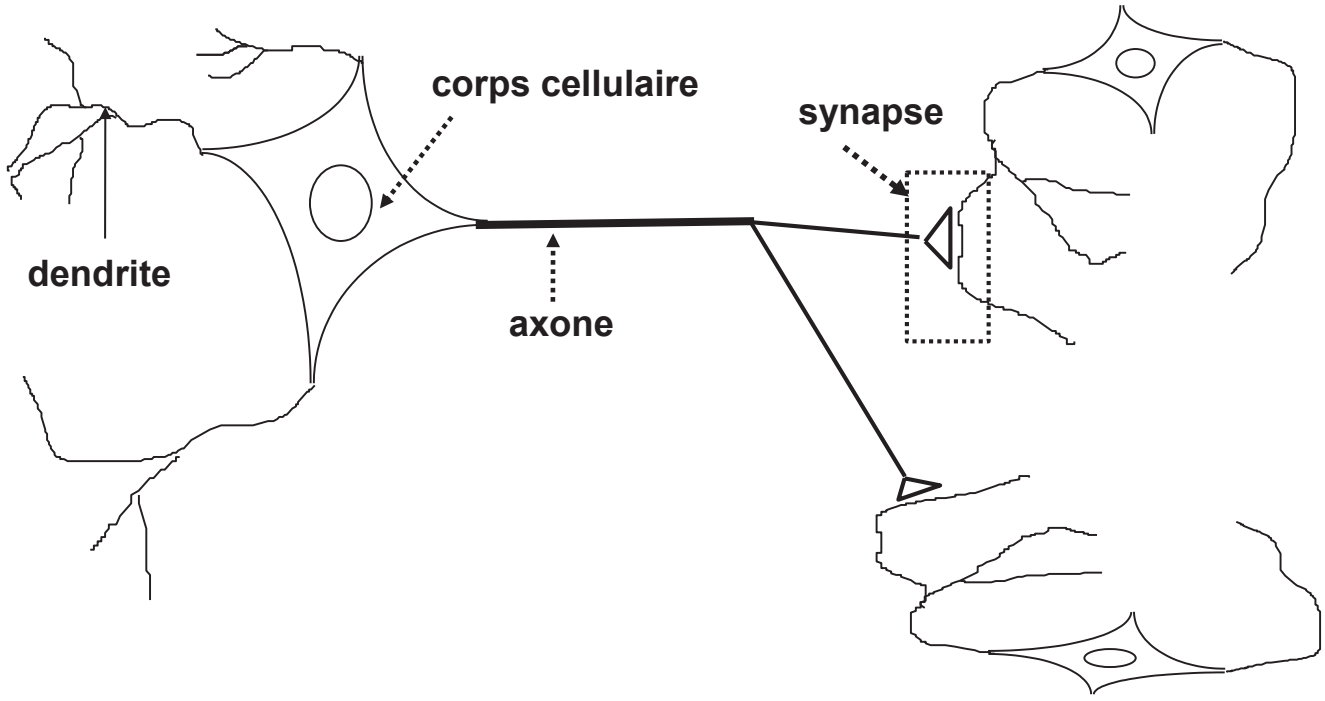
Apprentissage d'une relation entrée-sortie connue par des exemples :

- Reconnaissance de formes
- Reconnaissance vocale
- Classification, diagnostic

- Identification
- Prédiction
- Filtrage
- Commande, régulation

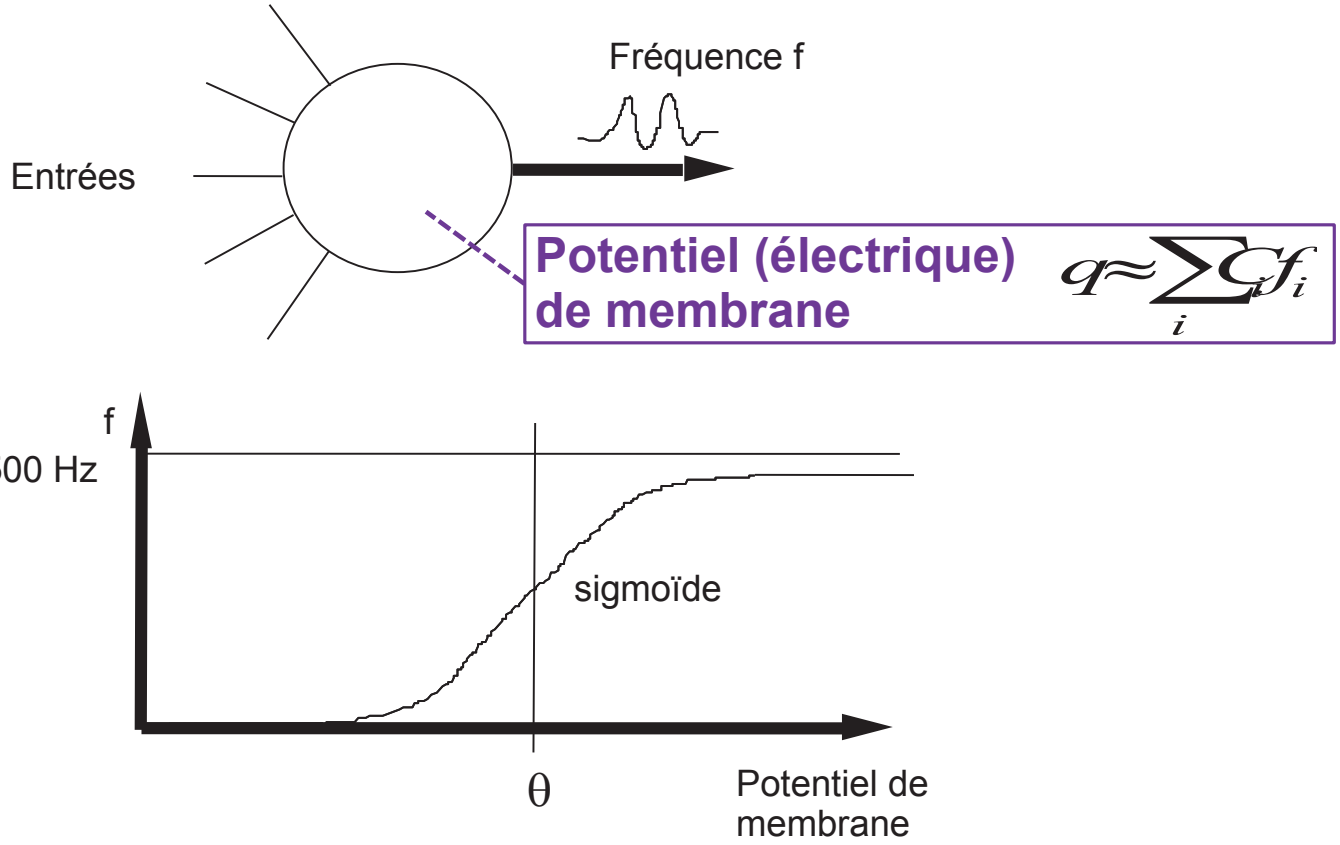
+ Optimisation combinatoire (réseaux de Hopfield)

NEURONES BIOLOGIQUES



- Signal électrique dendrites --> corps --> axone --> synapses

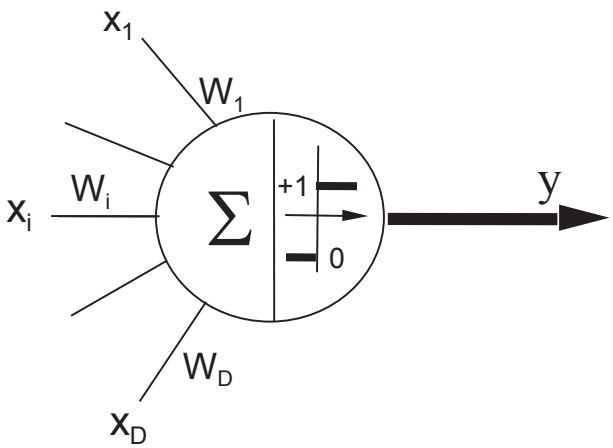
MODELISATION DES NEURONES



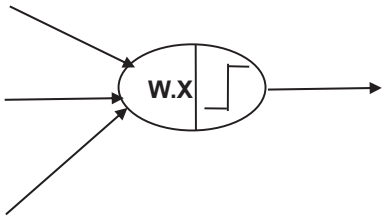
NAISSANCE DU NEURONE FORMEL

- **LES DEBUTS : Mc Culloch et Pitts (1943)**

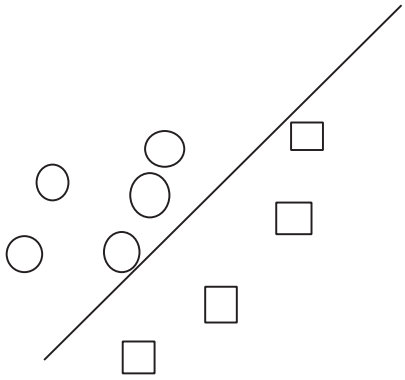
  - **modèle de neurone simple**
  - **but de modéliser le cerveau**



SEPARATION LINEAIRE PAR UN SEUL NEURONE



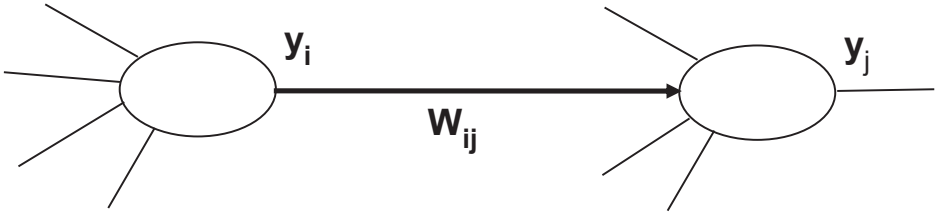
la séparation linéaire



MODELE DE L'APPRENTISSAGE

- **Règle de Hebb (1949)**

Renforcement des connections entre neurones
<u>activés simultanément</u>



$$W_{ij}(t+dt) = W_{ij}(t) + \lambda y_i(t) y_j(t)$$

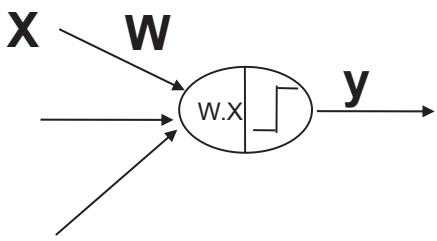
PREMIERS "RESEAUX DE NEURONES FORMELS"

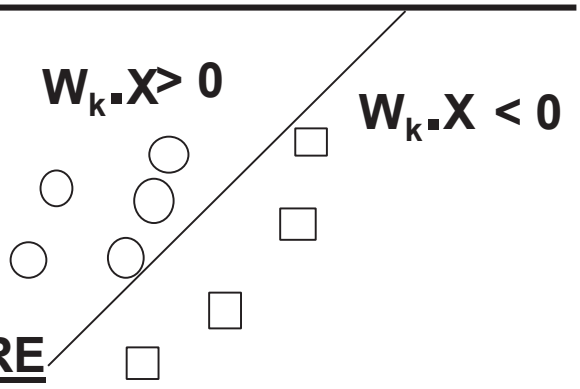
- **LE PERCEPTRON (Rosenblatt, 1957)**
- **ADALINE (Widrow, 1962)**

neurone formel Mac Culloch & Pitts
+
règle d'apprentissage de Hebb

réalisation de fonctions booléennes
par apprentissage à partir d'exemples

APPRENTISSAGE DU PERCEPTRON





==> SEPARATION LINEAIRE

loi d'apprentissage :

$W_{k+1} = W_k + dX$ si X mal classé (d : sortie désirée)
$W_{k+1} = W_k$ si X bien classé

- **Convergence de l'algorithme dans le cas séparable**
- **Impossible de savoir si un problème est non séparable.**

LIMITES DES PREMIERS MODELES

- **PERCEPTRONS, le livre de Minsky et Papert (1969)**

Etude approfondie du perceptron et de ses limites intrinsèques :

- certaines fonctions booléennes impossibles à apprendre
(XOR, ...)
- limitation aux séparations linéaires

- **Les progrès des ordinateurs séquentiels ont poussé le développement du traitement symbolique et de l'Intelligence Artificielle (systèmes experts) et l'abandon *temporaire* des réseaux de neurones.**

LE 1er RENOUVEAU DU NEURONAL

- **La mécanique statistique au secours des réseaux de neurones (J.J. Hopfield, 82)**

--> intérêt de la dynamique des réseaux totalement connectés

- **LA RETROPROPAGATION DU GRADIENT et la résolution du Credit Assignment Problem (Rumelhart 86, Le Cun 85) : *apprentissage de réseaux à couches cachées*.**

- **Puissance de calcul des ordinateurs**
==> résolution empirique de problèmes réels.

- **Des résultats mathématiques :**
approximateurs *universels* (et parcimonieux)

- **Vers années 2000 : utilisés, mais moins populaires que SVMs et boosting**

2e « REVIVAL » RECENT : DEEP LEARNING

- **Depuis 2006, intérêt croissant et excellents résultats avec *« deep » neural networks*, à plusieurs couches cachées de grandes dimensions :**
  - **initialisation « intelligente » non-supervisée**
  - **apprentissage usuel ensuite → « fine-tuning » des poids**
  - **couches cachées ➔ features appris (bas niveau sur première couche cachée, et de + en + haut niveau sur couches suivantes)**
- **Depuis ~2010, les *Convolutional* Neural Networks ont permis résultats spectaculaires en reconnaissance visuelle d'objets, reconnaissance vocale, et analyse du langage naturel**

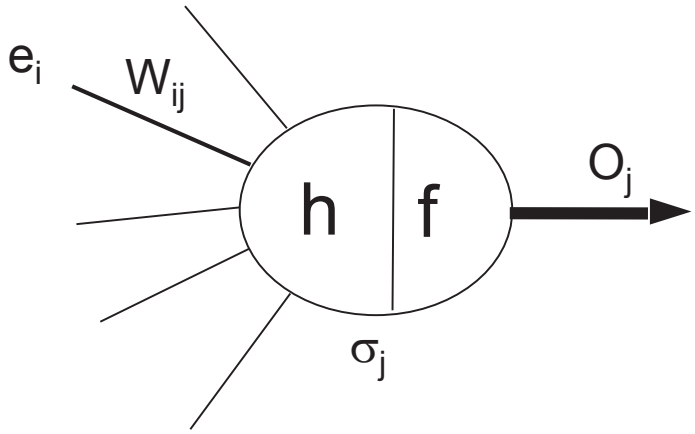
Les réseaux de neurones formels

DEFINITIONS DU NEURONE FORMEL

Une définition très générale : processeur qui applique une opération simple à ses entrées et que l'on peut relier à d'autres pour former un réseau qui peut réaliser une relation entrée-sortie quelconque.

DEFINITION USUELLE : processeur très simple qui calcule une somme pondérée et qui applique à cette somme une fonction de transfert non linéaire (échelon, sigmoïde, gaussienne, ...)

LE NEURONE FORMEL



e_i : entrées du neurone
σ_j : potentiel du neurone
O_j : sortie du neurone

W_{ij} : poids (synaptiques)
h : fonction d'entrée (calcul du potentiel = Σ, dist, noyau, …)
f : fonction d'activation (ou de transfert)

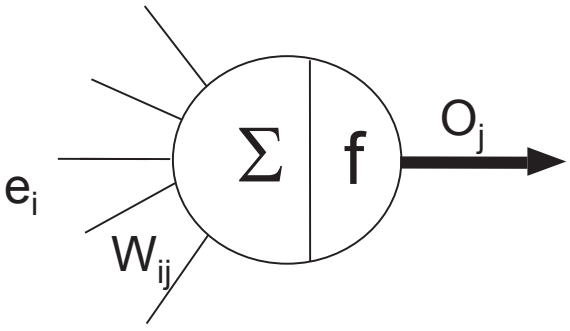
$$\sigma_j = f(h(e_i, \{W_{ij}, i=0 \text{ à } k_j\}))$$

$$O_j = f(\sigma_j)$$

La combinaison (h,f) définit le type de neurone

Summating artificial "neurons"

PRINCIPLE

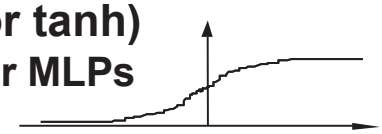


$$O_j = f\left(W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i \right)$$

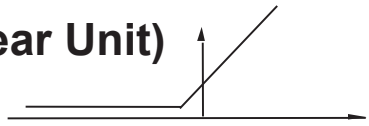
W_{0j} = "bias"

ACTIVATION FUNCTIONS

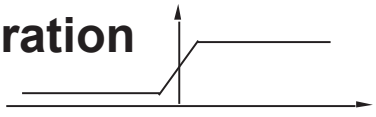
- **Threshold** (Heaviside or sign)
  → ***binary*** neurons
- **Sigmoïd** (logistic or tanh)
  → most common for MLPs



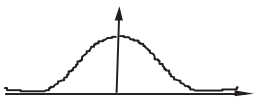
- **Identity** → ***linear*** neurons
- **ReLU** (Rectified Linear Unit)



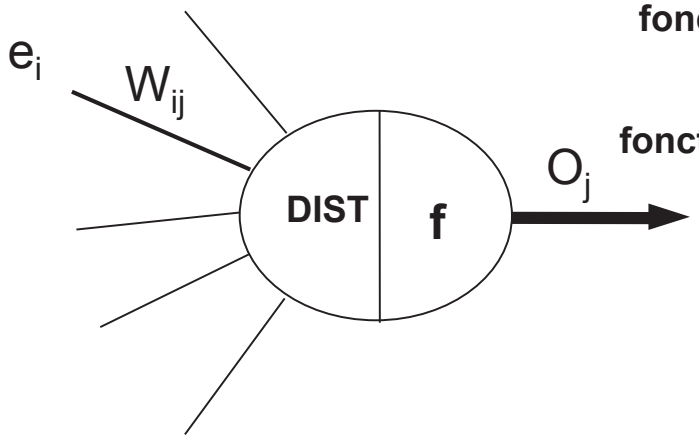
- **Saturation**



- **Gaussian**



LE NEURONE DISTANCE

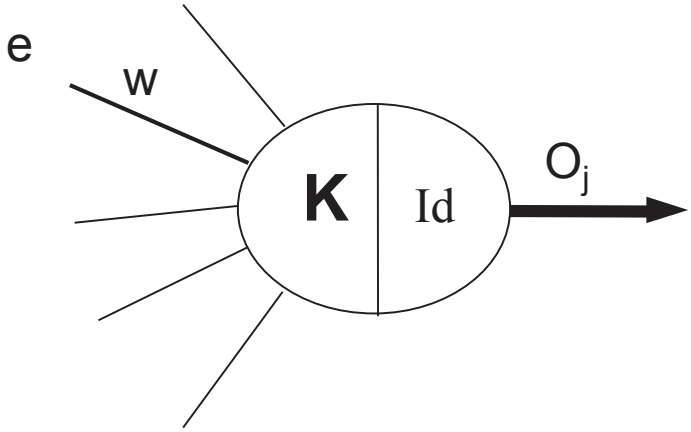


fonction d'entrée : $h\left(\begin{pmatrix} e_1 \\ \dots \\ e_k \end{pmatrix}, \begin{pmatrix} W_{1j} \\ \dots \\ W_{kj} \end{pmatrix}\right) = \sum_i (e_i - W_{ij})^2$

fonction d'activation = Identité ou gaussienne

L'activation représente donc la distance entre le vecteur d'entrée $(e_i)_i$ et le vecteur poids $(W_{ij})_i$

NEURONES DE TYPE NOYAU



fonction d'entrée : $h(e,w) = K(e,w)$

avec K symétrique et « positif »
au sens de Mercer : $\forall \psi$ tq $\int \psi^2(x)dx < \infty$,
alors $\int K(u,v)\psi(u)\psi(v)dudv \geq 0$

fonction d'activation = identité

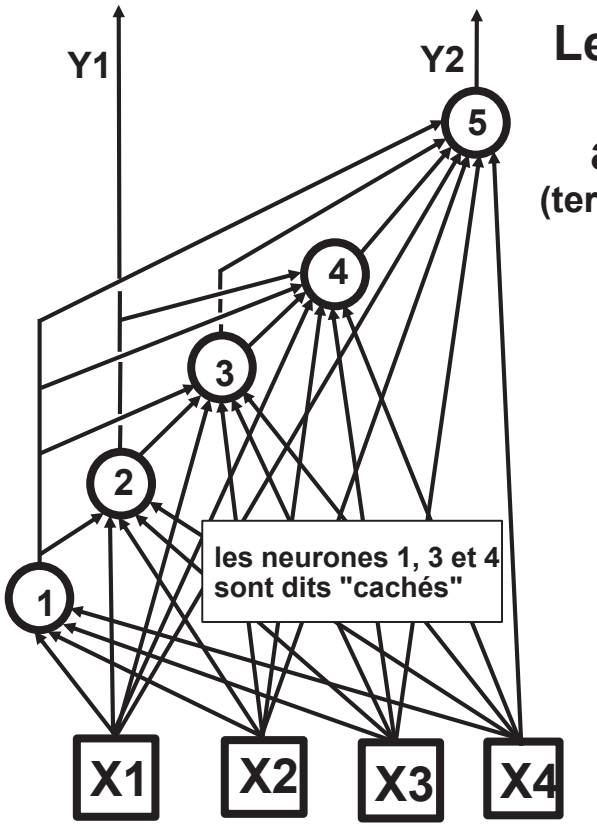
Exemples de noyaux « admissibles » :

- **Polynomiaux : $K(u,v) = [u.v + 1]^p$**
- **Radial Basis Function : $K(u,v) = \exp(-\|u-v\|^2 / 2\sigma^2)$**
  - ➔ **équivalent à distance+activation gaussienne**
- **Sigmoïdes : $K(u,v) = \tanh(u.v+\theta)$**
  - ➔ **équivalent à sommateur+activation sigmoïde**

DEUX FAMILLES DE RESEAUX

- **RESEAUX NON BOUCLES : sont utilisés en classification, reconnaissance des formes (caractères, parole, ...), en prédiction**

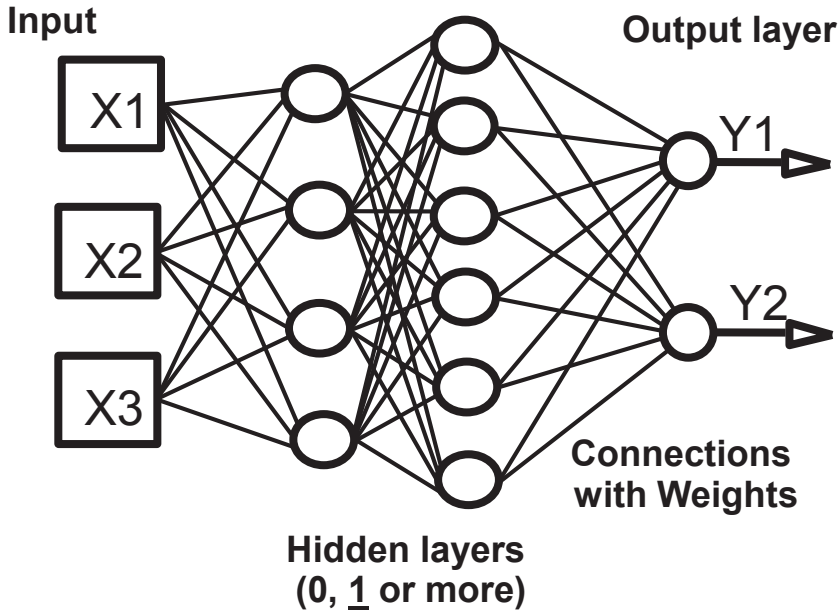
- **RESEAUX BOUCLES : sont utilisés comme mémoire associative (Hopfield, réseaux à attracteurs) ou pour des tâches de traitement du signal ou de commande.**



Les neurones peuvent être ordonnés d'une façon telle qu'il n'y a aucune connection "vers l'arrière"
(terme anglais : "FEEDFORWARD" neural network)

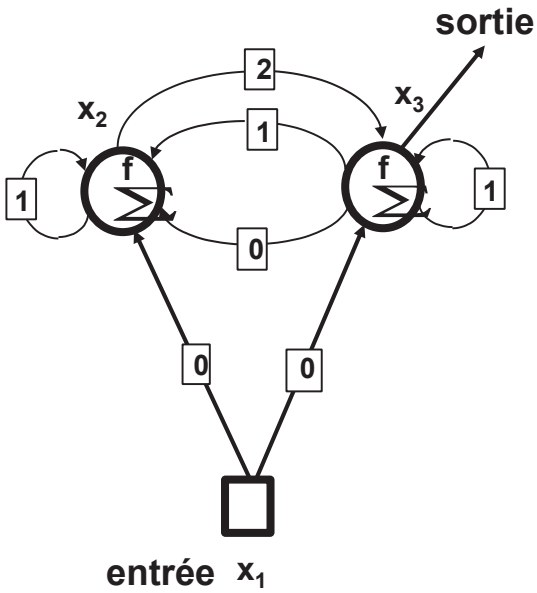
Le temps n'intervient pas comme variable fonctionnelle
(i.e. le réseau n'a pas de mémoire et ses sorties ne dépendent pas de son passé)

Feed-forward Multi-layer Neural Networks

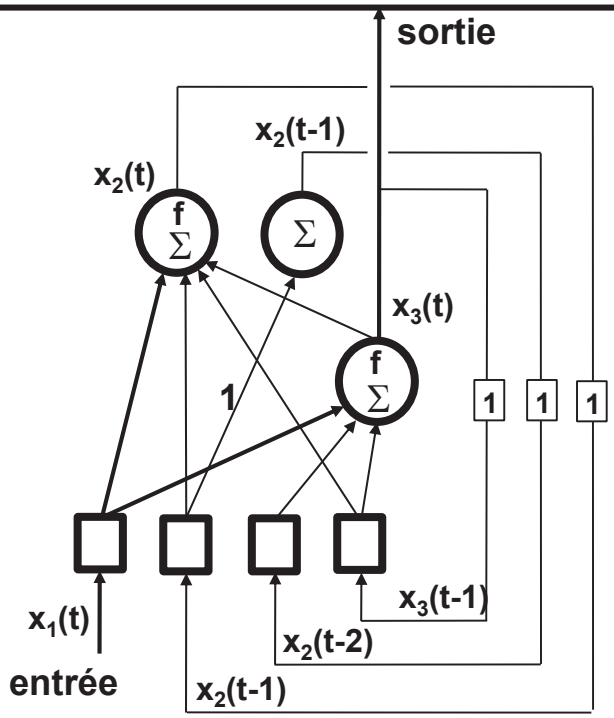


For "Multi-Layer Perceptron" (MLP), neurons type generally "summating with sigmoid activation"

RESEAUX BOUCLES

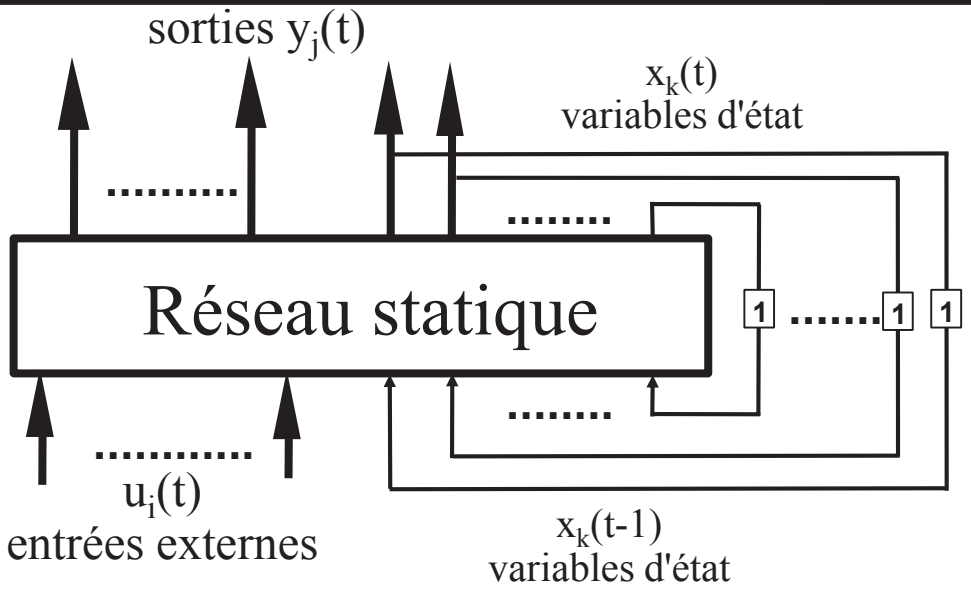


A chaque connection est associé un délai



Forme équivalente

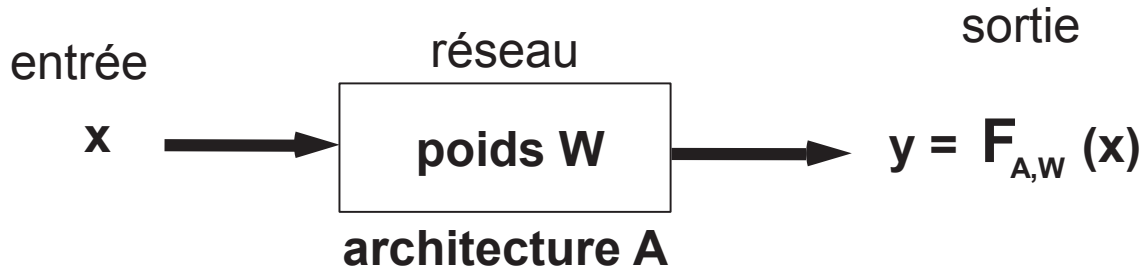
RESEAUX BOUCLES : forme canonique



FORME CANONIQUE DES RESEAUX BOUCLES

Les sorties à t dépendent non seulement des entrées externes à t, mais aussi (via les variables d'état) de toute la séquence des entrées externes précédentes (et de l'initialisation des variables d'état)

UTILISATION D'UN RESEAU



- **Deux modes :**
  - **apprentissage : à partir d'exemples de couples (entrée, sortie) , le réseau modifie**
    - **les paramètres W (poids des connexions)**
    - **éventuellement son architecture A (en créant/éliminant neurones ou connexions)**
  - **reconnaissance :**
    calcul de la sortie associée à une entrée

GENERALITES ET FORMALISME

- L'apprentissage supervisé est l'adaptation des coefficients synaptiques d'un réseau afin que pour chaque exemple, la sortie du réseau corresponde à la sortie désirée.

- FONCTION DE COÛT : problème de minimisation

Soient n exemples (X_p;D_p) et Y_p la sortie donnée par le réseau
On définit l'erreur qui peut être quadratique de la forme

$$E(W) = \sum_p \left(Y_p - D_p\right)^2$$

L'apprentissage revient à déterminer W =ArgMin(E).

En général, on utilise des méthodes de gradient, total, partiel ou stochastique :

$W(t+1) = W(t) - \varepsilon \frac{\partial E}{\partial W}$ *[+ μ(t)(W(t)-W(t-1))]*

Training of Multi Layer Perceptrons (MLP)

- Training by Stochastic Gradient Descent (SGD), using *back-propagation*:
  - Input 1 (or a few) random training sample(s)
  - Propagate
  - Calculate error (loss)
  - Back-propagate through all layers from end to input, to compute gradient and update weight

Back-propagation Principle

Smart method for efficient computing of gradient (w.r.t. weights) of a Neural Network cost function, based on chain rule for derivation.

Cost function is $Q(t) = \sum_m \text{loss}(Y_m, D_m)$, where m runs over training set examples

Usually, $\text{loss}(Y_m, D_m) = ||Y_m - D_m||^2$ [quadratic error]

Total gradient:

$$W(t+1) = W(t) - \lambda(t)\, \text{grad}_W(Q(t)) + \mu(t)(W(t)-W(t-1))$$

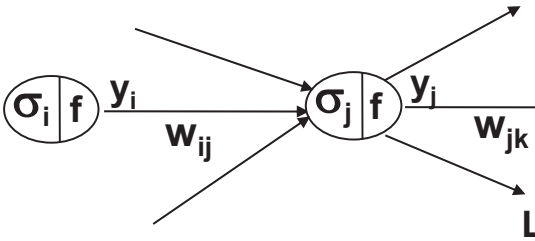
Stochastic gradient:

$$W(t+1) = W(t) - \lambda(t)\, \text{grad}_W(Q_m(t)) + \mu(t)(W(t)-W(t-1))$$

where $Q_m = \text{loss}(Y_m, D_m)$, is error computed on only ONE example randomly drawn from training set at every iteration and $\lambda(t)$ = learning rate (fixed, decreasing or adaptive), $\mu(t)$ = momentum

Now, how to compute dQ_m/dW_{ij}?

Backprop through fully-connected layers: use of chain rule derivative computation



$$dE_m/dW_{ij} = (dE_m/d\sigma_j)(d\sigma_j/dW_{ij}) = (dE_m/d\sigma_j)\, y_i$$

Let $\delta_j = (dE_m/d\sigma_j)$. Then $W_{ij}(t+1) = W_{ij}(t) - \lambda(t)\, y_i\, \delta_j$

(and $W_{0j}(t+1) = W_{0j}(t) - \lambda(t)\delta_j$)

If neuron j is output, $\delta_j = (dE_m/d\sigma_j) = (dE_m/dy_j)(dy_j/d\sigma_j)$ with $E_m = ||Y_m - D_m||^2$

so $\delta_j = 2(y_j - D_j) f'(\sigma_j)$ if neuron j is an output

Otherwise, $\delta_j = (dE_m/d\sigma_j) = \sum_k (dE_m/d\sigma_k)(d\sigma_k/d\sigma_j) = \sum_k \delta_k (d\sigma_k/d\sigma_j) = \sum_k \delta_k W_{jk} (dy_j/d\sigma_j)$

so $\delta_j = (\sum_k W_{jk}\delta_k) f'(\sigma_j)$ if neuron j is "hidden"

➔ all the δ_j can be computed successively from last layer to upstream layers by "error backpropagation" from output

LE THEOREME D'APPROXIMATION UNIVERSELLE

Cybenko 89

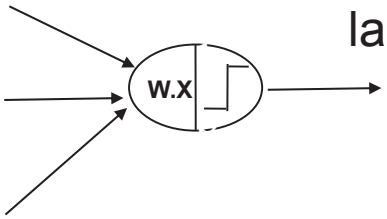
- **Pour toute fonction F continue définie et bornée sur un ensemble borné, et pour tout ε, il existe un réseau à 1 couche cachée de neurones sigmoïdes qui approxime F à ε près.**

…Mais on ne sait pas commment le trouver à coup sûr, et ce réseau a peut-être énormément de neurones cachés...

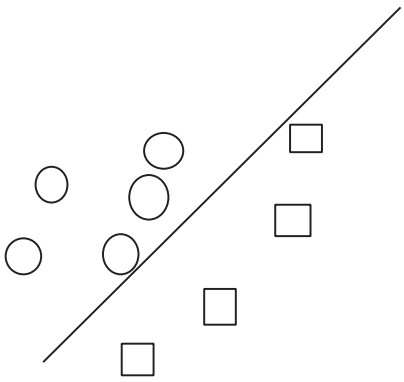
Sussman 92

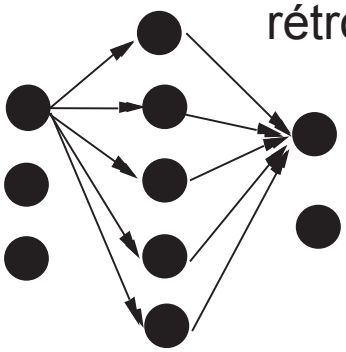
- **Les réseaux à une couche cachée forment une famille d'approximateurs parcimonieux : à nombre égal de paramètres on approxime correctement plus de fonctions qu'avec des polynômes**

Rétropropagation v.s. réseau monocouche

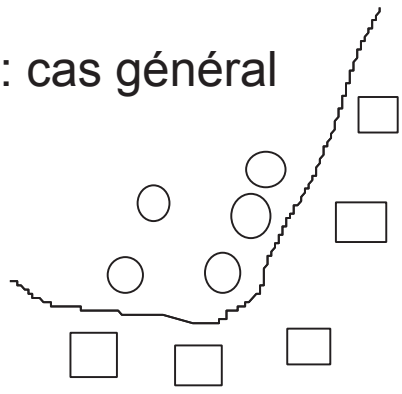


la séparation linéaire





rétropropagation : cas général



Caractéristiques des réseaux neuronaux multicouches

AVANTAGES

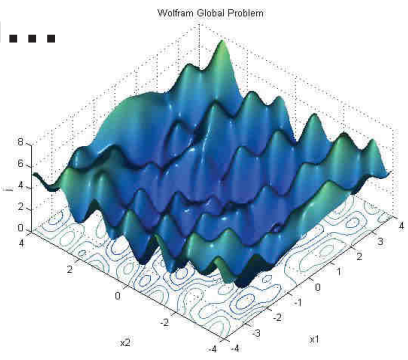
- Approximateur universel parcimonieux (et classifieur universel)
- Rapidité d'exécution (++ si multi-processeurs ou chip)
- Robustesse des solutions, résistance au bruit des données
- Facilité de développement

INCONVENIENTS

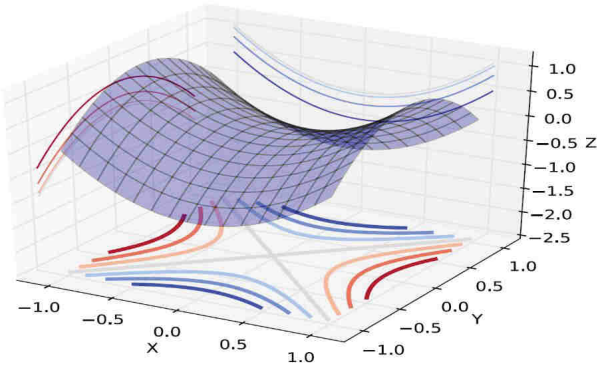
- *Le choix de l'architecture est critique*
- Le temps d'apprentissage peut être long
- *Présence de minima locaux de la fonction de coût*
- Difficultés d'interprétation des résultats en terme de connaissance

Why gradient descent works *despites non-convexity*?

- Local minima dominate in low-Dim…

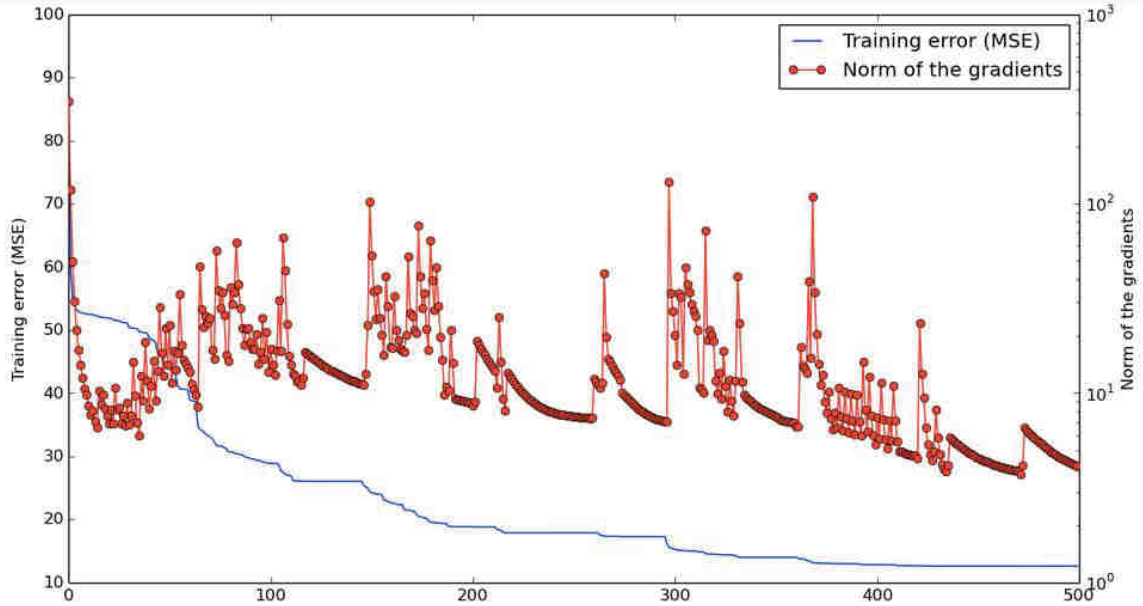


- …but recent work has shown ***saddle points*** dominate in high-Dim



- Furthermore, most local minima are close to the global minimum

Saddle points in training curves



- **Oscillating between two behaviors:**
  - **Slowly approaching a saddle point**
  - **Escaping it**

METHODOLOGIE POUR L'APPRENTISSAGE SUPERVISE DE RESEAUX NEURONAUX A COUCHES

Méthodo : bases d'apprentissage, de validation, et de test

- Espace des entrées possibles infini (si valeurs continues), ensemble d'apprentissage = ss-ensemble (≈ échantillonage)
- Erreur nulle sur tous les exemples ≠ bons résultats sur tout l'espace des entrées possibles (cf erreur de généralisation ≠ erreur empirique…)

↓

- collecter suffisamment d'exemples représentatifs
- mettre de côté un ensemble d'exemples qui ne servira que de *"base de test"* pour estimer le taux de généralisation final (ie quand apprentissage terminé)
- séparer les exemples restants en une base d'apprentissage et une *"base de validation"* cette dernière servira pour arrêter l'apprentissage quand l'erreur cesse de baisser en test (éviter le "sur-apprentissage")

Méthodo : validation (*eg* croisée) pour optimiser paramètres

Pour bien maximiser GENERALISATION, tous paramètres d'apprentissage (taille couche cachée, pas de gradient, nombre d'itérations, etc…) doivent être optimisés par validation :

- soit avec base validation séparée (découpage *aléatoire* des exemples en Apprentissage+Validation)
- soit par *validation croisée :*
  - estimer erreur sur plusieurs ensembles de validation (k-fold ou « leave-one-out » puis en calculer la moyenne des erreurs

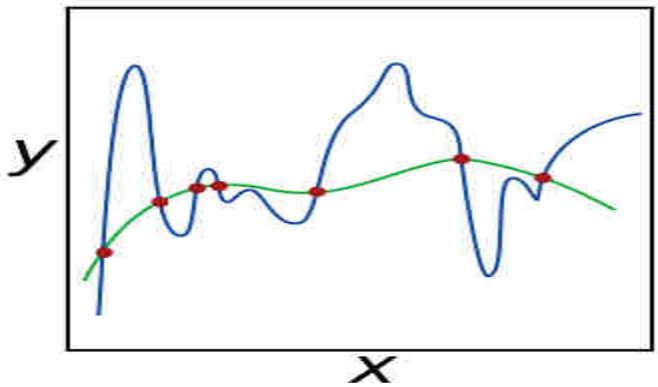
3-fold cross-validation :

- apprendre sur E1∪E2 et estimer sur E3
- apprendre sur E1∪E3 et estimer sur E2
- apprendre sur E2∪E3 et estimer sur E1
- moyenner errE1,errE2,errE3

Some Neural Networks training « tricks »

- Importance of <u>input normalization</u>
  (zero mean, unit variance)
- Importance of <u>weights initialization</u>
  random but SMALL and prop. to 1/sqrt(nbInputs)
- Decreasing (or adaptive) <u>learning rate</u>
- Importance of <u>training set size</u>
  If a Neyral Net has a LARGE number of free parameters,
  ➔ train it with a sufficiently large training-set !
- Avoid overfitting by Use of L1 or L2 <u>regularization</u>

Avoid <u>overfitting</u> using L1/L2 regularization (« weight decay »)



Trying to fit too many free parameters with not enough information can lead to overfitting

<u>Regularization</u> = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just <u>norm L2 or L1 of vector of all weights</u>:

$$K = \sum_m(\text{loss}(Y_m, D_m)) + \beta \sum_{ij} |W_{ij}|^p \quad \text{with p=2 (L2) or p=1 (L1)}$$

➔ name "Weight decay"

QUELQUES REFERENCES SUR LES RESEAUX NEURONAUX

- Des livres :
  - ***Réseaux de neurones : méthodologie et applications*, G. Dreyfus et al., Eyrolles, 2002.**
  - ***Réseaux de neurones formels pour la modélisation, la commande, et la classification*, L. Personnaz et I. Rivals, CNRS éditions, collection Sciences et Techniques de l'Ingénieur, 2003.**
  - ***Réseaux de neurones : de la physique à la psychologie*, J.-P. Nadal, Armand Colin, 1993.**
- Sur le Web :
  - La « FAQ » sur les réseaux neuronaux :
    `http://www.faqs.org/faqs/ai-faq/neural-nets/part1/`
  - Les pages du groupe «neuronal » de l'UTC de Compiègne :
    `http://www.hds.utc.fr/~grandval/rna/`
  - Les pages du cours «neuronal » de l'INSA de Rouen :
    `http://asi.insa-rouen.fr/~scanu/coursRNA/`
  - Voir aussi les liens depuis ma homepage :
    `http://www.ensmp.fr/~moutarde/rech_neuronal.html`