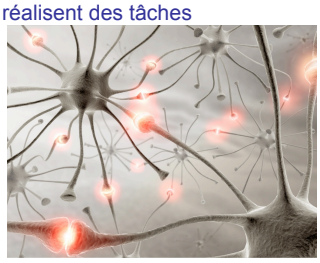
Plan du cours

- **Éléments de base (le neurone, architectures, paramètres)**
- **Critères et algorithmes d'apprentissage**
- **Réseaux de neurones artificiels supervisés**
- **Réseaux de neurones artificiels non supervisés**
- **Réseaux de neurones artificiels profonds**
- **Propriétés et liens avec l'analyse factorielle**
- **Conclusion**

L'idée d'origine (~ 1940)

• Les organismes vivants, même assez primitifs (ex. insectes), réalisent des tâches complexes de traitement de l'information :
  • orientation
  • communication
  • comportement social
  • ...



• La puissance de traitement de leur système nerveux vient de l'interconnexion(ex. 10^{14} connexions chez l'homme)
  • d'un grand nombre (ex. 10^{11} chez l'homme)
  • d'unités de traitement simples et similaires : les neurones

• La motivation initiale était de faire du neuro-mimétisme
  • toutefois, la vision des années 1940 était assez simpliste;
  • la réalité biologique c'est avéré plus complexe depuis.

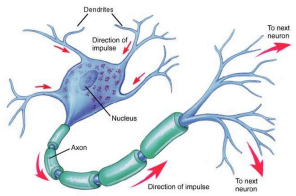
• En revanche, cette idée c'est avérée très féconde en mathématiques et en ingénierie

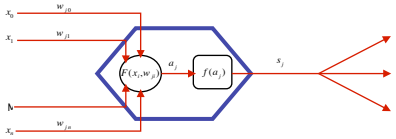
Le Neurone Formel

1943, Mc Culloch & Pitts

Un modèle et non une copie du neurone biologique
= un processeur élémentaire caractérisé par :

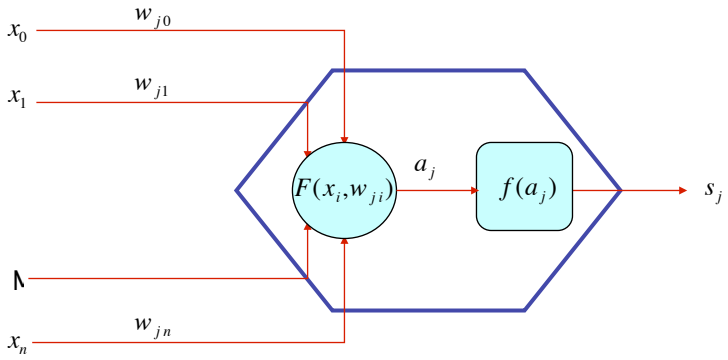
- **signaux d'entrée** $x_0, x_1, \ldots, x_n$
- **poids des connexions** $w_{i0}, w_{i1}, \ldots, w_{in}$
- **fonction d'activation** $F(x,w)$
- **état interne d'activation** $a = F(x,w)$
- **fonction de transition** $f(a)$
- **état de sortie** $s = f(a)$





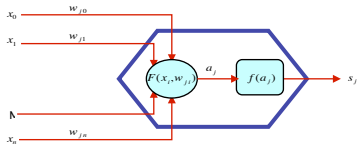
Le Neurone Formel

Définition : ***Un neurone formel (artificiel) est une unité de traitement qui reçoit des données en entrée, sous la forme d'un vecteur, et produit une sortie réelle. Cette sortie est une fonction des entrées et des poids des connexions.***



Poids et connexion

Définition : ***Une connexion entre deux unités i et j indique la possibilité d'une relation physique entre ces deux unités.***

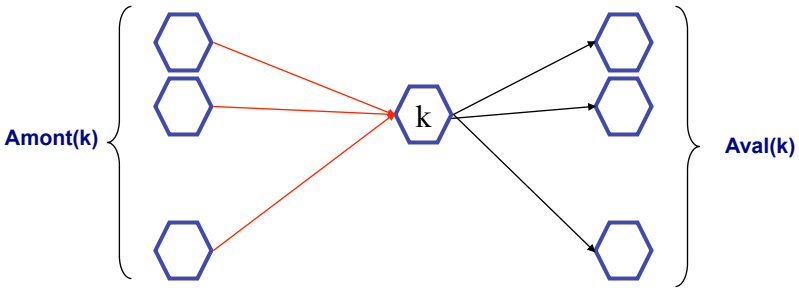


Définition : ***La valeur numérique du poids associé à une connexion entre deux unités reflète la force de la relation entre ces deux unités. Si cette valeur est positive, la connexion est dite excitatrice, sinon elle est dite inhibitrice. La convention usuelle est de noter le poids de la connexion reliant le neurone i au neurone j :*** w_{ji}

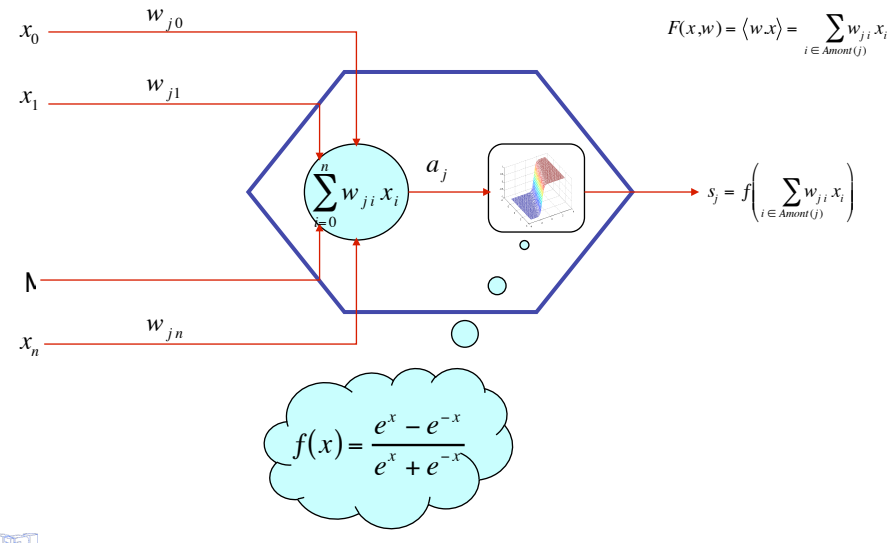
Le Neurone Formel

Définition : ***Si l'on considère l'unité k, et que l'on appelle Amont(k) l'ensemble des neurones dont les sorties servent d'entrées au neurone k, Aval(k) l'ensemble des neurones qui utilisent comme entrée la sortie du neurone k.***

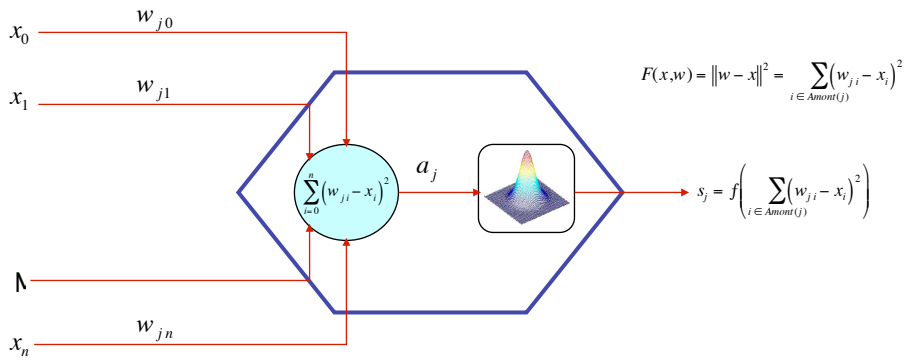
E l'ensemble des neurones d'entrée, et S l'ensemble des neurones de sortie. Alors, par définition on a : ∀ i ∈ E, Amont(i)=∅ et ∀ i ∈ S, Aval(i)= ∅.



Le Neurone Produit Scalaire



Le Neurone Distance



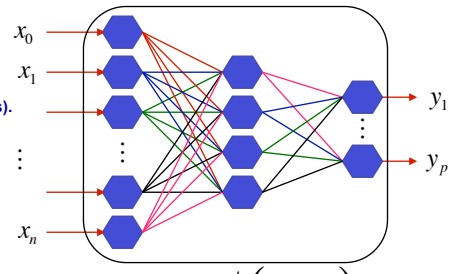
$$\|w - x\|^2 = \|w\|^2 - 2\langle w, x \rangle + \|x\|^2$$

Réseau Connexionniste / Réseau de neurones

Définition : ***Un réseau de neurones est un graphe valué orienté, constitué d'un ensemble d'unités (ou automates), réalisant des calculs élémentaires, structurées en couches successives capables d'échanger des informations au moyen de connexions qui les relient.***

- Architecture massivement parallèles.
- Système basé sur la coopération de plusieurs unités simples (neurones formels).

- Un réseau se caractérise par :
  - son architecture
  - les fonctions de ses éléments



$$y = \psi(x, w)$$

Fonctions d'un réseau connexionniste

Le Mode Apprentissage :

Le réseau est modifié en fonction des exemples d 'apprentissage

On peut modifier :

- **les poids de ses connexions**
- **son architecture**
- **les fonctions de transition des unités**

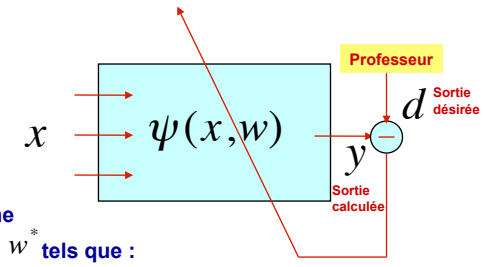
Il existe essentiellement deux sortes d 'algorithmes d 'apprentissage :

- **l'apprentissage supervisé**
- **l'apprentissage non supervisé**

Apprentissage Supervisé

On se donne :

- **N exemples étiquetés** $E_N = \left\{\left(x^1,d^1\right),\left(x^2,d^2\right),\ldots,\left(x^N,d^N\right)\right\}$ $x^k \in \Re^n$ $d^k \in \Re^p$ (entrée, Sortie désirée)
- **une architecture de réseau** A
- **des poids initiaux** w_0



On cherche, au moyen de l'algorithme d'apprentissage à trouver des poids w^* **tels que :**

- **les exemples sont reconnus :** $d^k = \psi(x^k,w)$
- **on obtient une bonne généralisation :** $d = \psi(x,w)$ **est une réponse raisonnable pour l'entrée** x

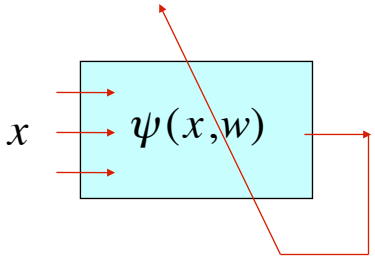
Apprentissage Non Supervisé

On se donne :

- N exemples non étiquetés $E_N = \{x^1, x^2, \ldots, x^N\}$

- une architecture de réseau A

- des poids initiaux w_0



On cherche à trouver des poids w^* **tels que :**

- les exemples sont correctement regroupés

- on obtient une bonne généralisation

Mode Reconnaissance

Le réseau est utilisé pour traiter des données

- les poids et l 'architecture du réseau sont fixés

- on lui présente un exemple x **en entrée et le réseau**

calcule une sortie $y = \psi(x,w)$ **qui dépend de l'architecture** A **et des poids** w^* **appris.**

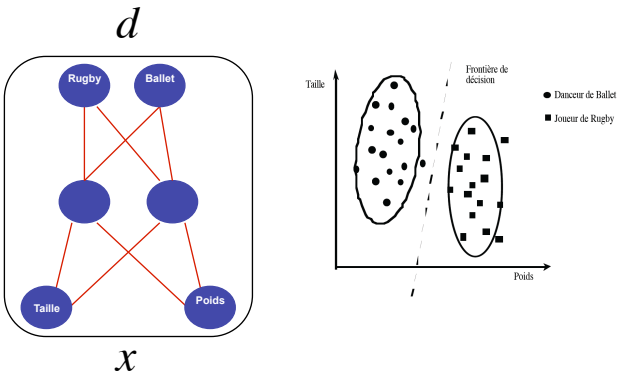
La sortie peut être :

- l'étiquette de la classe (identité)	**Classement/Discrimination**
- le N° du cluster	**Classification/Quantification**
- la valeur à prévoir	**Prévision**
- un vecteur du même type que l'entrée	**Codage/Compaction**

Utilisation des modèles connexionnistes

Grand nombre de mesures + Loi sous-jacente inconnue

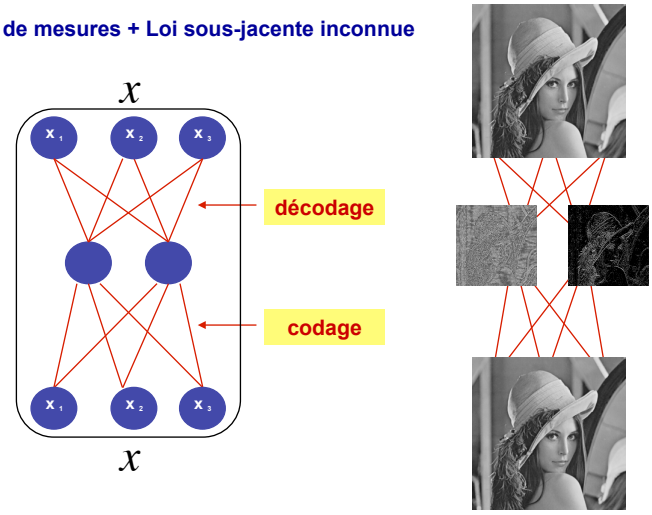
Classement



Utilisation des modèles connexionnistes

Grand nombre de mesures + Loi sous-jacente inconnue

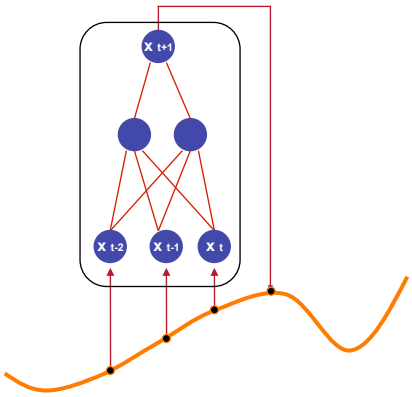
Compression

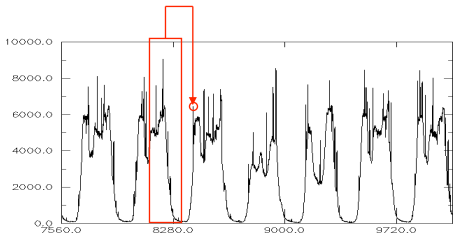


Utilisation des modèles connexionnistes

Grand nombre de mesures + Loi sous-jacente inconnue

Régression

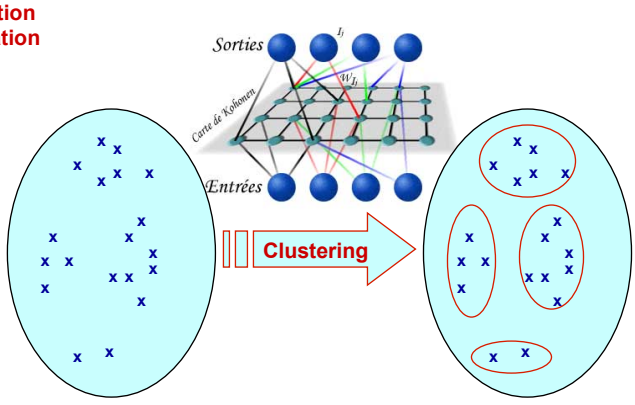




Utilisation des modèles connexionnistes

Grand nombre de mesures + Loi sous-jacente inconnue

Classification
Quantification



Apprentissage à partir d'exemples

But :

- **Construire un système qui explique des données ou une relation pertinente liant ces données**
- **Estimer un modèle statistique qui a généré les données**

Connaissances :

- **Données**
- **Domaine**

Démarche :

- **Choisir un système possédant une bonne capacité de généralisation**
- **Agir sur la complexité effective des modèles**

Formalisme

La problématique de l'apprentissage se présente souvent comme la minimisation d'une fonction de risque :

$$R(w) = \int_{z} L(z,w)\, dp(z)$$

Risque théorique

L'erreur de généralisation

Fonction de perte

(Loss function)

Où z **, à valeur dans un espace** $\mathcal{Z}$ **(des conditions extérieures ou des exemples), représente les observations d'une v.a.** Z **de distribution de probabilité** $p(z)$**, fixe mais inconnue, et** w **les paramètres du réseau.**

L'apprentissage revient à trouver les paramètres :

$$w^* = \arg\min_{w} R(w)$$

Formalisme

Le risque théorique n'est pas calculable, $p(z)$ est inconnue.

Mais un échantillon $E_N = \{z^k\}_{k=1}^N$ d'exemples indépendants, tirés de $p(z)$ est connu.

En pratique, on ne peut pas minimiser $R(w)$, on utilise alors un principe d'induction.

Le plus courant = minimisation du risque empirique (MRE) :

Risque empirique

(Erreur d 'apprentissage)

$$\tilde{R}(w) = \frac{1}{N}\sum_{k=1}^{N} L(z^k, w)$$

Choix : $w^+ = \arg\min_w \tilde{R}(w)$

Algorithmes d'Optimisation

$$\frac{\partial \tilde{R}(w)}{\partial w} = \lim_{\delta w \to 0} \frac{\tilde{R}(w+\delta w) - \tilde{R}(w)}{\delta w}$$

La règle d'adaptation :

Gradient stochastique

$$w(t+1) = w(t) - \varepsilon(t)\nabla_w \tilde{R}^k(w)$$

$$\tilde{R}^k(w) = L(z^\gamma, w)$$

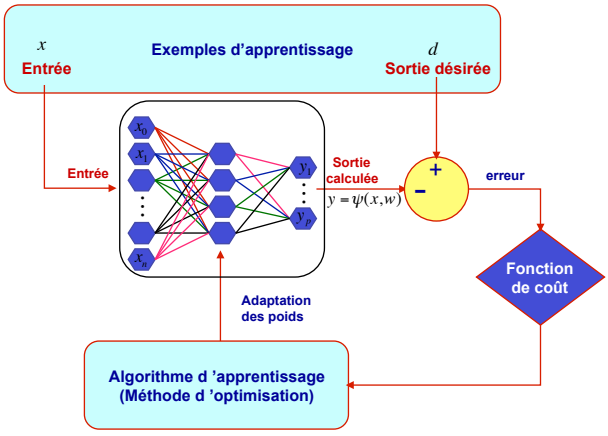
$$\nabla_w \tilde{R}^k(w) = \left(\frac{\partial \tilde{R}^k(w)}{\partial w_1}, \ldots, \frac{\partial \tilde{R}^k(w)}{\partial w_i}, \ldots, \frac{\partial \tilde{R}^k(w)}{\partial w_n}\right)$$

Gradient total

$$w(t+1) = w(t) - \frac{1}{N}\varepsilon(t)\sum_{k=1}^{N}\nabla_w \tilde{R}^k(w)$$

Systèmes d'Apprentissage Supervisé

Système d 'Apprentissage Supervisé



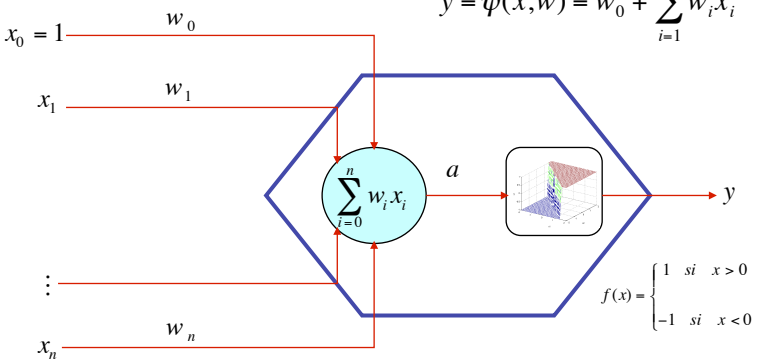
Adaline : Adaptive Linear Element

Stanford, 1960, Bernard Widrow*

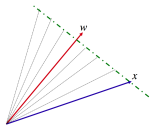
C'est un élément linéaire adaptatif : $y = \sum_{i=0}^{n} w_i x_i$

$$y = \psi(x,w) = w_0 + \sum_{i=1}^{n} w_i x_i$$

L 'unité x_0, dont l 'activation fixée à 1, est dite unité de biais. Elle permet d'introduire les termes de biais dans le réseau.



$y = \sum_{i=0}^{n} w_i x_i = \|w\| \|x\| \cos\phi$

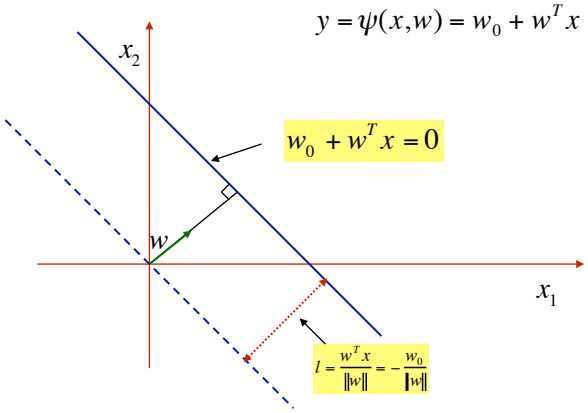


* Widrow B., Hoff M.E. (1960) : « Adaptive switching circuits », IRE WESCON Conv. Record, part 4, pp. 96-104.

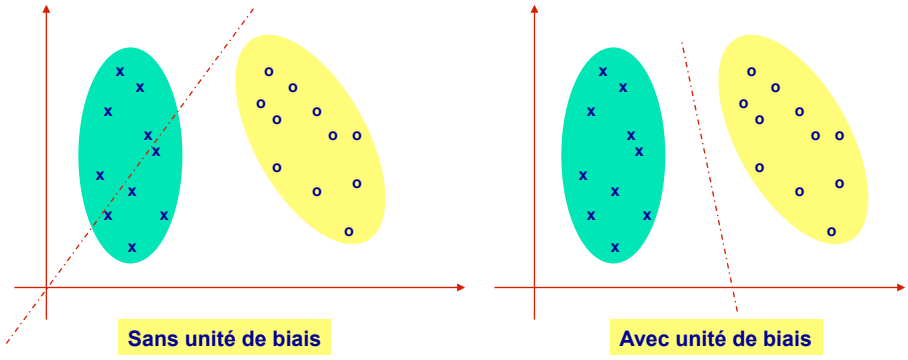
Adaline : Adaptive Linear Element

$$y = \psi(x,w) = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$y = \psi(x,w) = w_0 + w^T x$$

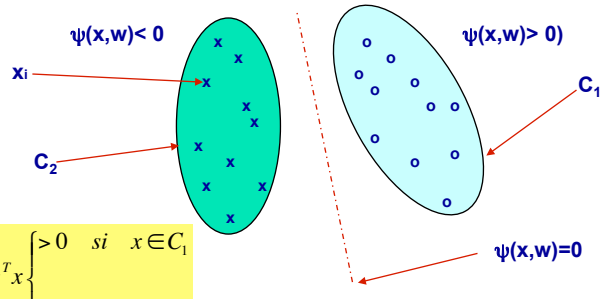


Adaline : Adaptive Linear Element



Adaline : Adaptive Linear Element

On peut l'utiliser en discrimination (classement) entre 2 classes :



$$\psi(x,w) = w_0 + w^T x \begin{cases} > 0 & si \quad x \in C_1 \\ < 0 & si \quad x \in C_2 \end{cases}$$

Adaline : Adaptive Linear Element

Si l 'on appelle $z^k = (x^k, d^k)$ la forme prise en compte à l'itération k,
On définit le carré de l'erreur instantanée associée à la forme (x^k, d^k) par :

$$\tilde{R}^k_{Adaline}(w) = \left(d^k - wx^k\right)^2$$

Sortie désirée — Sortie calculée : y

L'erreur quadratique globale ou (MSE) est définie comme la moyenne observée des carrés des erreurs instantanées sur l'ensemble de toutes les formes :

$$\tilde{R}_{Adaline}(w) = \frac{1}{N}\sum_{k=1}^{N} \tilde{R}^k_{Adaline}(w)$$

Il existe plusieurs algorithmes d'apprentissage.

Adaline : Adaptive Linear Element

Techniques de descente de gradient (la plus grande pente) :
supposons qu 'à l'instant t , les poids de l 'Adaline soient $w(t)$
et qu 'on présente la forme (x^k, d^k) , alors les poids seront modifiés par :

$$w(t+1) = w(t) - \varepsilon(t)\nabla_w \tilde{R}^k_{Adaline}(w)$$

Le pas du gradient — Le gradient instantané

$$\nabla_w \tilde{R}^k_{Adaline}(w) = \frac{\partial \tilde{R}^k_{Adaline}(w)}{\partial w} = -2\left(d^k - wx^k\right)x^k$$

Cette règle est appelée règle du gradient stochastique
ou règle de Widrow-Hoff
ou règle du delta de Widrow-Hoff
ou règle μ-LMS (Least Mean Square)

Adaline : Adaptive Linear Element

1- Tirer au hasard w_0

2- Présenter une forme $\left(x^k, d^k\right)$

3- Déterminer la valeur de l'écart

$$e^k(t) = \left(d^k - wx^k\right)$$

4- Calculer une approximation du gradient

$$\nabla_w \tilde{R}^k_{Adaline}(w) = -2e^k(t)x^k$$

5- Adapter les poids $w(t)$

$$w(t+1) = w(t) - \varepsilon(t)\nabla_w \tilde{R}^k_{Adaline}(w)$$

Où $\varepsilon(t)$ est le pas du gradient.

6- Répéter de 2 à 4 jusqu'à l'obtention d'une valeur acceptable de l'erreur

Adaline : Exemples

Données : Table de vérité d 'une fonction booléenne de 2 variables

x_1	x_2	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Fonction : ψ(x,w)= (x_1 ou x_2)

$$E = \left\{ \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} -1 \\ 1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} -1 \\ -1 \end{pmatrix}, -1 \right) \right\}$$

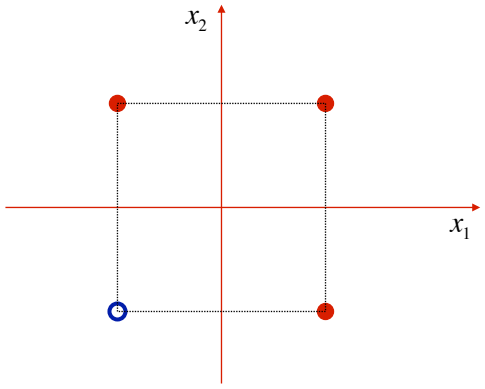
Problème :

Trouver un Adaline capable d 'apprendre la table de vérité d'une fonction booléenne de 2 variables

Adaline : Exemples

Fonction : ψ(x,w)= (x_1 ou x_2)

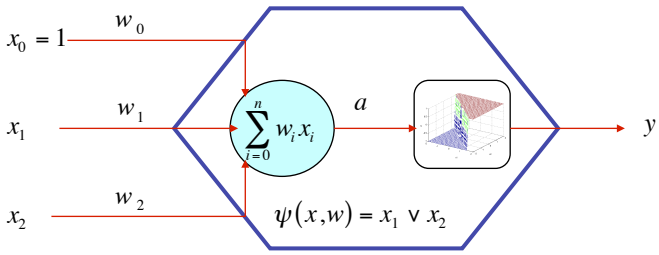
x_1	x_2	d
1	1	1
1	-1	1
-1	1	1
-1	-1	-1



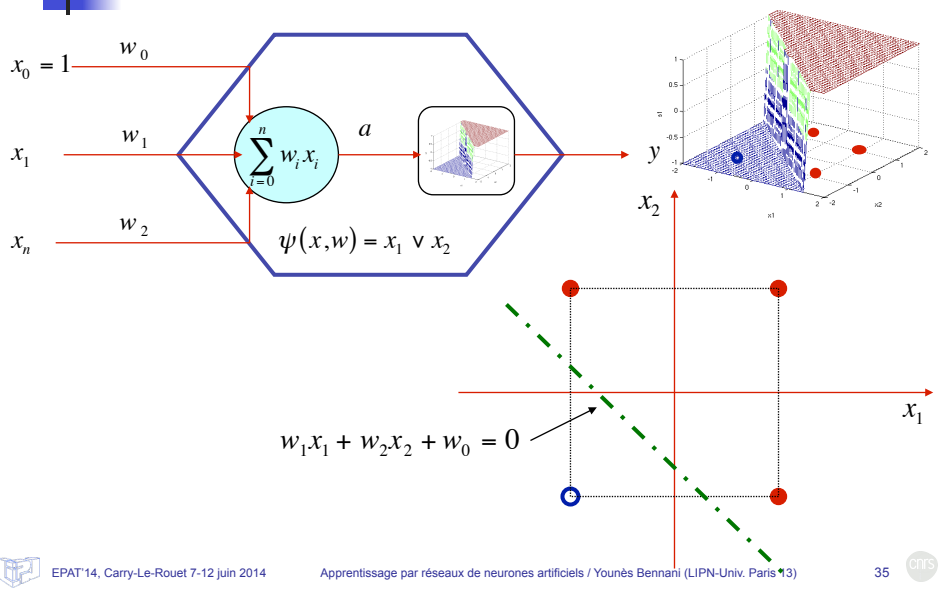
Adaline : Exemples

$$E_N = \left\{\left(x^1,d^1\right),\left(x^2,d^2\right),\ldots,\left(x^N,d^N\right)\right\}$$

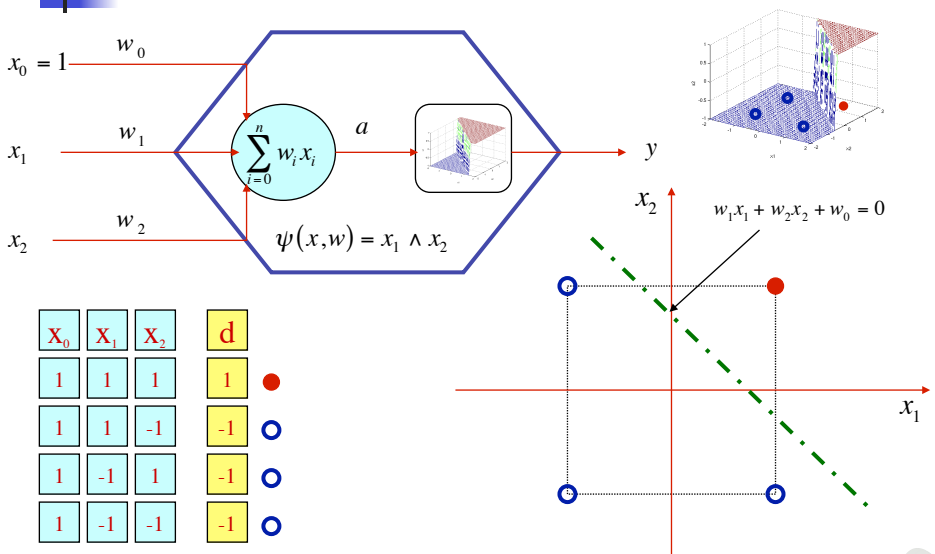
x_0	x_1	x_2	d
1	1	1	1
1	1	-1	1
1	-1	1	1
1	-1	-1	-1



Adaline : Exemples



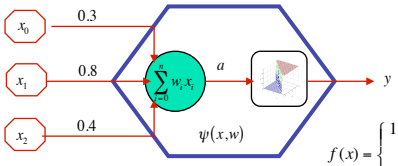
Adaline : Exemples



x_0	x_1	x_2	d
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Exercice à faire

$$E = \left\{ \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, -1 \right); \left(\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, 1 \right) \right\}$$



$$f(x) = \begin{cases} 1 & si \quad x > 0 \\ -1 & si \quad x < 0 \end{cases}$$

Représenter dans un repère orthogonal l'ensemble des échantillons.

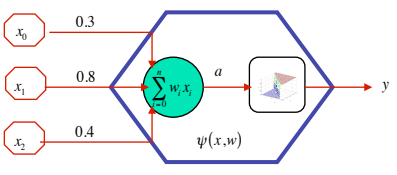
Utiliser l 'algorithme Adaline pour adapter les poids du modèle (ε=0.1).

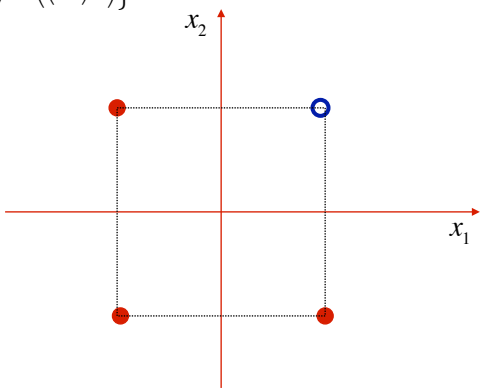
Donner l 'équation de l 'hyperplan séparant les deux classes.

Représenter l 'hyperplan dans le même repère orthogonal

Représentation graphique

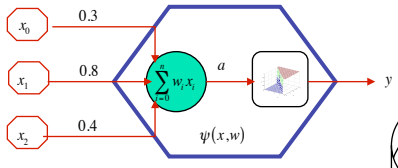
$$E = \left\{ \left(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, -1 \right); \left(\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, 1 \right) \right\}$$





Adaptation des poids

$$D = \left\{ \begin{pmatrix} \begin{pmatrix} 1 \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{pmatrix}, -1 \end{pmatrix}; \begin{pmatrix} \begin{pmatrix} 1 \\ \begin{pmatrix} 1 \\ -1 \end{pmatrix} \end{pmatrix}, 1 \end{pmatrix}; \begin{pmatrix} \begin{pmatrix} 1 \\ \begin{pmatrix} -1 \\ 1 \end{pmatrix} \end{pmatrix}, 1 \end{pmatrix}; \begin{pmatrix} \begin{pmatrix} 1 \\ \begin{pmatrix} -1 \\ -1 \end{pmatrix} \end{pmatrix}, 1 \end{pmatrix} \right\}$$

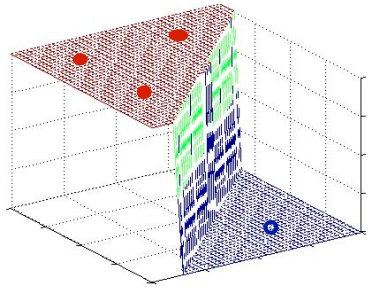


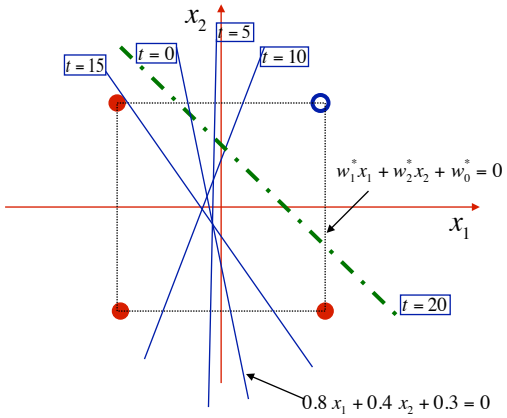
$$\begin{pmatrix} w_0(t+1) \\ w_1(t+1) \\ \vdots \\ w_n(t+1) \end{pmatrix} = \begin{pmatrix} w_0(t) \\ w_1(t) \\ \vdots \\ w_n(t) \end{pmatrix} - 2\,\varepsilon(t) \left[d^k - \begin{pmatrix} w_0(t) & w_1(t) & \dots & w_n(t) \end{pmatrix} \begin{pmatrix} x_0^k \\ x_1^k \\ \vdots \\ x_n^k \end{pmatrix} \right] \begin{pmatrix} x_0^k \\ x_1^k \\ \vdots \\ x_n^k \end{pmatrix}$$

$$\begin{pmatrix} w_0(t+1) \\ w_1(t+1) \\ w_2(t+1) \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.8 \\ 0.4 \end{pmatrix} - 2 \times 0.1 \times \left[-1 - \begin{pmatrix} 0.3 & 0.8 & 0.4 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right] \times \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} w_0(t+1) \\ w_1(t+1) \\ w_2(t+1) \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.8 \\ 0.4 \end{pmatrix} - \begin{pmatrix} -0.5 \\ -0.5 \\ -0.5 \end{pmatrix} \rightarrow \begin{pmatrix} w_0(t+1) \\ w_1(t+1) \\ w_2(t+1) \end{pmatrix} = \begin{pmatrix} 0.8 \\ 1.3 \\ 0.9 \end{pmatrix}$$

Évolution de l'apprentissage

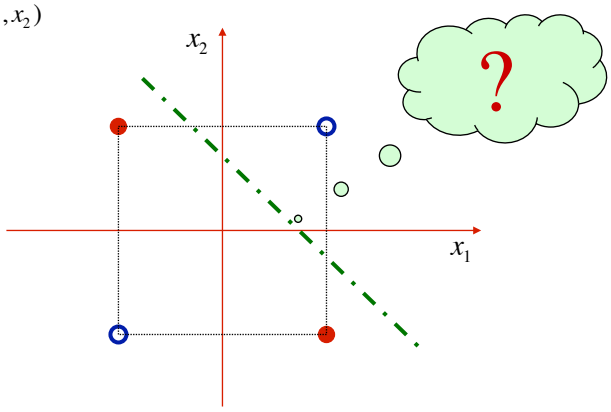




Adaline : limites

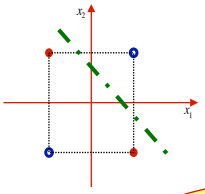
$$\psi(x,w) = x_1 \oplus x_2 = XOR(x_1, x_2)$$

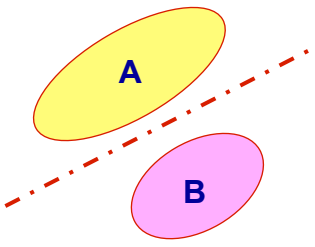
	x_0	x_1	x_2	d
x^1	1	1	1	-1
x^2	1	1	-1	1
x^3	1	-1	1	1
x^4	1	-1	-1	-1



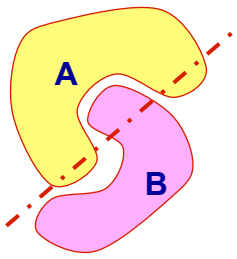
Séparabilité linéaire

« Deux classes d'objets, décrits dans un espace de dimension n, sont dits « linéairement séparables » s'ils se trouvent de part et d'autre d'un hyperplan dans l'espace des descripteurs»





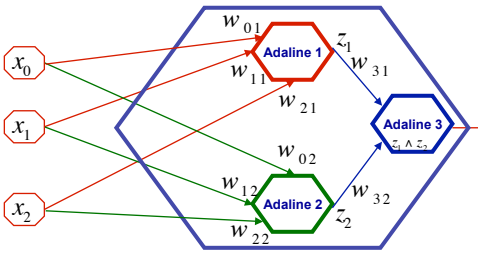
Linéairement séparable



Non-linéairement séparable

Madaline : Multi-Adaptive Linear Element

Madaline = un ensemble d'Adalines parallèles

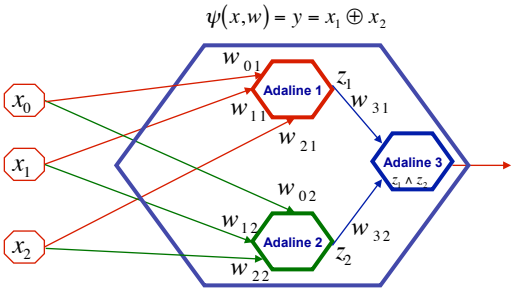


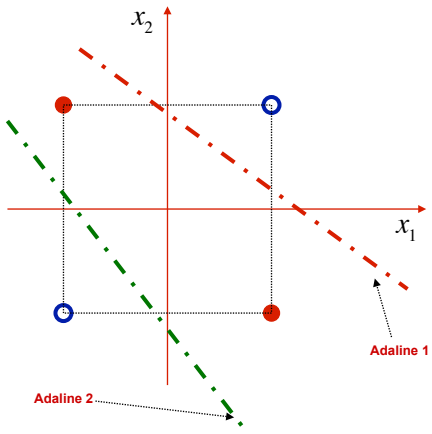
$$\psi(x,w) = y = x_1 \oplus x_2$$

	x_0	x_1	x_2	Adaline 1 d	Adaline 2 d	Adaline 3 d
x^1	1	1	1	-1	1	-1
x^2	1	1	-1	1	1	1
x^3	1	-1	1	1	1	1
x^4	1	-1	-1	1	-1	-1

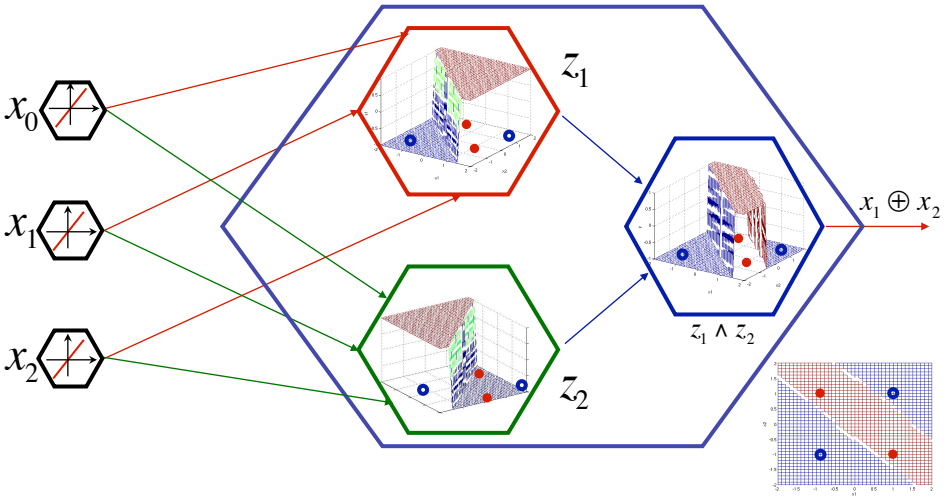
Madaline : Multi-Adaptive Linear Element

$$\psi(x,w) = y = x_1 \oplus x_2$$



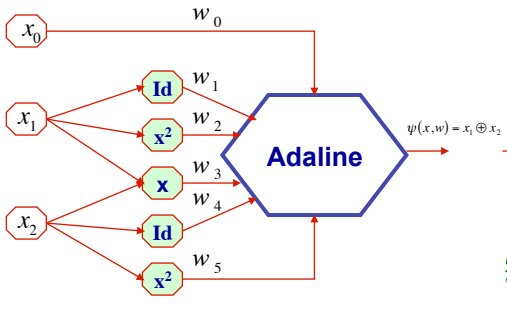


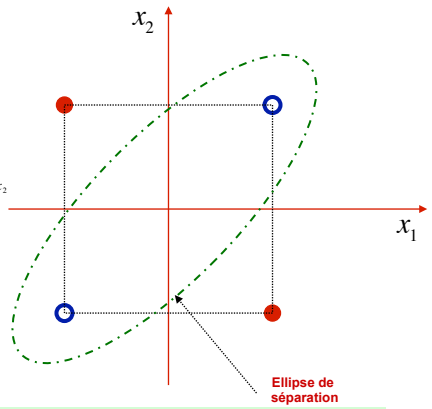
Madaline : Multi-Adaptive Linear Element



Madaline : Multi-Adaptive Linear Element

Adaline avec pré-traitement polynomial





$$w_1 x_1 + w_2 x_1^2 + w_3 x_1 x_2 + w_4 x_2 + w_5 x_2^2 + w_0 = 0$$

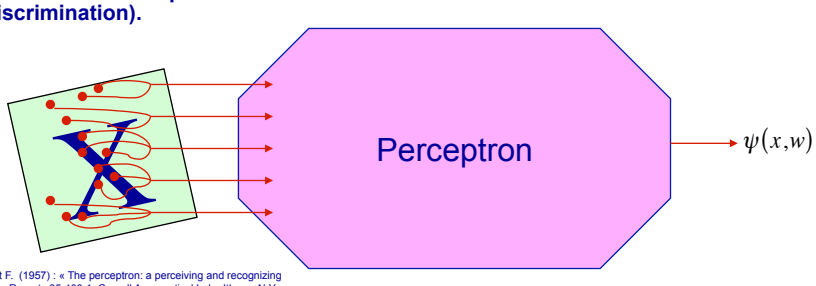
Perceptron

Rosenblatt F., 1957, 1962*

1957, Frank Rosenblatt

Le perceptron ne désigne pas un seul modèle mais regroupe une importante famille d 'algorithmes.

Le perceptron = machine adaptative employée pour résoudre des problèmes de classement (discrimination).



* Rosenblatt F. (1957) : « The perceptron: a perceiving and recognizing automaton », Reports 85-460-1, Cornell Aeronautical Lab., Ithaca, N.Y.
* Rosenblatt F. (1962) : « Principles of Neurodynamics: perceptrons and theory of brain mechanisms », Spartan Books, Washington.

Perceptron

Rosenblatt F., 1957, 1962

la rétine R

qui reçoit les informations de l'extérieur

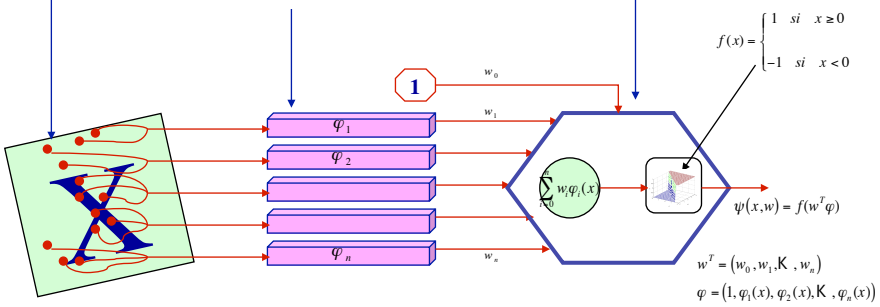
les cellules d'association

chaque cellule possède une fonction de transition définie sur la rétine :

$$\varphi_i(x): \quad R \to \Re$$

la cellule de décision

$$\psi(x,w) = f\left(w_0 + \sum_{i=1}^{n} w_i\, \varphi_i(x)\right)$$



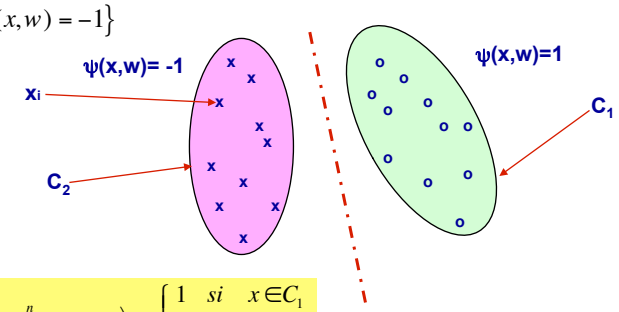
Perceptron

Cas de deux classes

Un perceptron peut être vu comme un classificateur à 2 classes :

$$C_1 = \{x \in R : \psi(x,w) = 1\}$$

$$C_2 = \{x \in R : \psi(x,w) = -1\}$$



$$\psi(x,w) = f\left(w_0 + \sum_{i=1}^{n} w_i \varphi_i(x)\right) = \begin{cases} 1 & si \quad x \in C_1 \\ -1 & si \quad x \in C_2 \end{cases}$$

Perceptron

Cas de deux classes

Si l 'on appelle (x^k, d^k) la forme prise en compte à l'itération k,
On définit le carré de l'erreur instantanée associée à la forme (x^k, d^k) par :

$$\begin{cases} d^k = 1 & si \quad x^k \in C_1 \\ d^k = -1 & si \quad x^k \in C_2 \end{cases} \Rightarrow \begin{cases} w^T \varphi^k > 0 & pour \quad x^k \in C_1 \\ w^T \varphi^k < 0 & pour \quad x^k \in C_2 \end{cases} \Rightarrow \forall\, x^k, w^T(\varphi^k\, d^k) > 0$$

$$\varphi^k = (1, \varphi_1(x^k), \varphi_2(x^k), \ldots, \varphi_n(x^k))$$

$$\tilde{R}^k_{Perceptron}(w) = -\, w^T(\varphi^k\, d^k)$$

L'erreur quadratique globale ou (MSE) est :

$$\tilde{R}_{Perceptron}(w) = \frac{1}{N}\sum_{k=1}^{N} \tilde{R}^k_{Perceptron}(w) = -\sum_{\{k : x^k \ mal\ classé\}} w^T(\varphi^k\, d^k)$$

Il existe plusieurs algorithmes d'apprentissage.

Perceptron

Cas de deux classes

Techniques de descente de gradient (la plus grande pente) : supposons qu 'à l'instant t , les poids du Perceptron soient $w(t)$ et qu 'on présente la forme (x^k, d^k) , alors les poids seront modifiés par :

$$w(t+1) = w(t) - \varepsilon(t) \nabla_w \tilde{R}^k_{Perceptron}(w)$$

Le pas du gradient — Le gradient instantané

$$\nabla_w \tilde{R}^k_{Perceptron}(w) = \frac{\partial \tilde{R}^k_{Perceptron}(w)}{\partial w} = -\varphi^k \, d^k$$

$$\varphi^k = (1, \varphi_1(x^k), \varphi_2(x^k), \mathrm{K}, \varphi_n(x^k))$$

Critère d'arrêt = taux de classement satisfaisant.

Perceptron

Algorithme d 'apprentissage : cas de 2 classes

1- A t=0, tirer au hasard $w(0) = (w_0(0), w_1(0), \mathrm{K}, w_n(0))^T$

2- Présenter une forme (x^k, d^k)

3- Calculer la sortie du perceptron et la comparer à y^k

$$f\left(w_0 + \sum_{i=1}^{n} w_i \varphi_i(x)\right) = f\left(w^T \varphi^k\right)$$

4- Si x^k est bien classé : $f(w^T \varphi^k) = d^k$

$$w(t+1) = w(t)$$

Si x^k est mal classé : $f(w^T \varphi^k) \neq d^k$

$$w(t+1) = w(t) + \varepsilon(t) \, \varphi^k \, d^k$$

$$\begin{pmatrix} w_0(t+1) \\ w_1(t+1) \\ \vdots \\ w_n(t+1) \end{pmatrix} = \begin{pmatrix} w_0(t) \\ w_1(t) \\ \vdots \\ w_n(t) \end{pmatrix} + \varepsilon(t) \, d^k \begin{pmatrix} 1 \\ \varphi_1(x^k) \\ \vdots \\ \varphi_n(x^k) \end{pmatrix}$$

$$\varphi^k = (1, \varphi_1(x^k), \varphi_2(x^k), \mathrm{K}, \varphi_n(x^k))$$

Où $\varepsilon(t)$ est le pas du gradient.

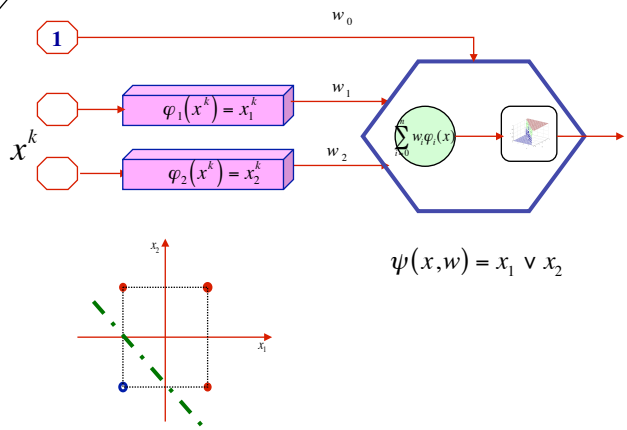
5- Répéter de 2 à 4 jusqu'à l'obtention d'une valeur acceptable de l'erreur

Exemple de Perceptron

Applet : http://lcn.epfl.ch/tutorial/french/perceptron/html/index.html

$$E_N = \left\{\left(x^1,d^1\right),\left(x^2,d^2\right),\ldots,\left(x^N,d^N\right)\right\}$$

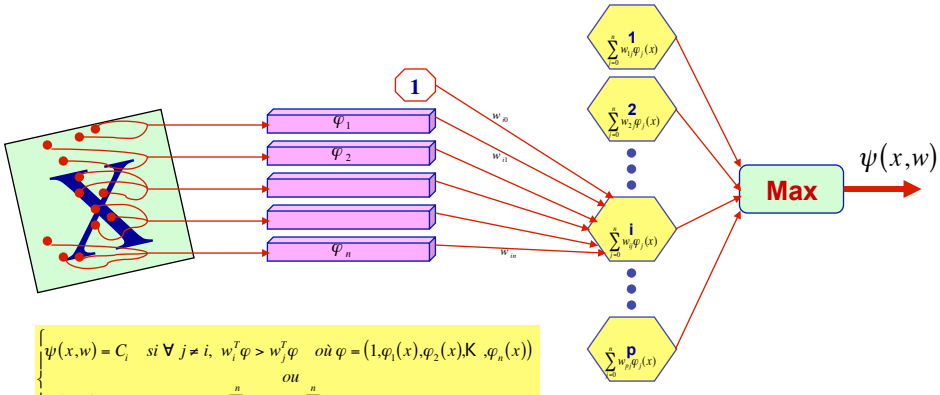
	x_0	x_1	x_2	d
x^1	1	1	1	1
x^2	1	1	-1	1
x^3	1	-1	1	1
x^4	1	-1	-1	-1



$$\psi(x,w) = x_1 \vee x_2$$

Perceptron

Cas de p classes : C_1, C_2, …, C_p



$$\begin{cases} \psi(x,w) = C_i \quad si\ \forall\ j \neq i,\ w_i^T\varphi > w_j^T\varphi \quad o\grave{u}\ \varphi = (1,\varphi_1(x),\varphi_2(x),\mathrm{K}\ ,\varphi_n(x)) \\ \qquad\qquad\qquad\qquad ou \\ \psi(x,w) = C_i \quad si\ \forall\ j \neq i,\ \sum_{k=1}^{n} w_{ik}\varphi_k > \sum_{k=1}^{n} w_{jk}\varphi_k \end{cases}$$

Perceptron

Algorithme d 'apprentissage : cas de p classes

1- A t=0, tirer au hasard la matrice des poids $w(0)$

2- Présenter une forme $x^k \in C_i$

3- Calculer la sortie $\psi(x^k, w)$ du perceptron et la comparer à C_i

$$\psi(x^k, w) = C_j \quad \Leftrightarrow \quad j = Arg \max_l \left(w_l^T \varphi^k\right)$$

4- Si x^k est bien classé : $C_j = C_i$

$$w(t+1) = w(t)$$

Si x^k est mal classé : $C_j \neq C_i$

$$w_i(t+1) = w_i(t) + \varepsilon(t)\varphi^k$$

$$w_j(t+1) = w_j(t) - \varepsilon(t)\varphi^k$$

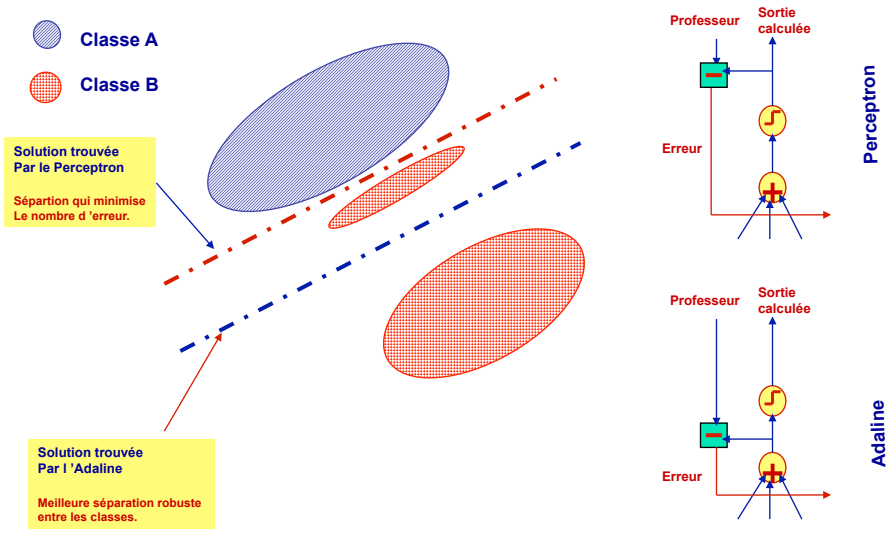
$$w_l(t+1) = w_l(t) \quad \forall\, l \neq i \text{ et } l \neq j$$

Où $\varepsilon(t)$ est le pas du gradient.

5- Répéter de 2 à 4 jusqu'à l'obtention d'une valeur acceptable de l'erreur

Perceptron et Adaline

Cas de 2 classes



Théorème de convergence du Perceptron

« Si un ensemble de formes est linéairement séparable, alors l'algorithme d'apprentissage du Perceptron converge vers une solution correcte en un nombre fini d'itération. »

Arbib M.A. (1987) : « Brains, Machines, and Mathematics »
Berlin, Springer-Verlag.

Rosenblatt F. (1962) : « Principles of Neurodynamics »
N.Y., Spartan.

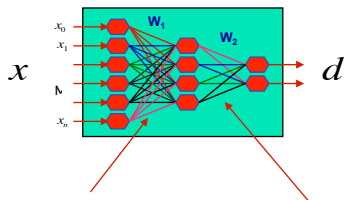
Block H.D. (1962) : « The Perceptron: A Model for Brain Functioning »
Reviews of Modern Physics 34, 123-135.

Minsky M.L. & Papert S.A. (1969) : « Perceptrons »
Cambridge, MIT Press.

Diederich S. & Opper M. (1987) : « Learning of Correlated Patterns Spin-Glass Networks by Local Learning Rules »
Physical Review letters 58, 949-952.

Architecture multi-couches

Le « credit assignment problem »
On se donne un réseau en couches, et un ensemble d 'exemples composés de paires entrées-sorties.



On ne connaît pas les sorties Désirées des unités cachées !

On ne peut pas appliquer l 'algorithme d 'apprentissage du Perceptron pour déterminer W_1

Perceptron Appliquer l 'algorithme d 'apprentissage pour déterminer W_2

L 'algorithme de rétro-propagation du gradient apporte une solution d 'une simplicité déroutante à ce problème.

L'après pause connexionniste

Problèmes de contrôle

1963, Bryson A., Denham W., Dreyfuss S.,

"la rétro-propagation du gradient"

Les années 80, la portée exacte de l 'ouvrage de Minsky & Papert sera correctement perçue.

La "redécouverte" de la "rétro-propagation du gradient"

1986, LeCun Y.

1986, Rumelhart D., Hinton G.E., Williams R.

Bryson A., Denham W., Dreyfuss S. (1963) : « Optimal Programming Problem With Ineduality Constraints. I: Necessary Conditions for Extremal Solutions », AIAA Journal, Vol. 1, pp. 25-44.

LeCun Y. (1986) : « Learning Processes in Asymmetric Threshold Network »
Disordered Systems and Biological Organizations, Les Houches, France, Springer, pp. 223-240.

Rumelhart D., Hinton G.E., Williams R. (1986) : « Learning Internal Representations by Error Propagation »
In Parallel Distributed Processing: exploring the microstructure of cognition, Vol I, Badford Books, Cambridge, MA, pp. 318-362, MIT Press.

Perceptron Multi-Couche (PMC)

Multi-Layer Perceptron (MLP)

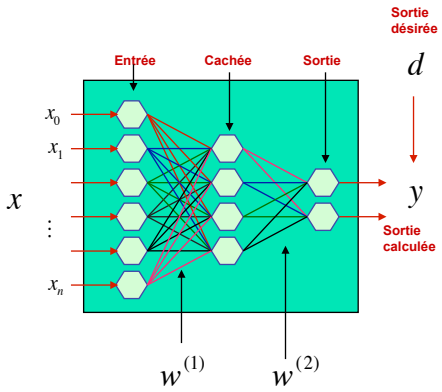
Architecture :
semblable à celle du Perceptron ou de Madaline
+ des couches de traitement intermédiaire
(couches cachées)

- Couches externes :
Entrée (e unités),
Sortie (s unités)
- Couches internes :
Cachées (c unités)

Notation : < e I c I s > **exemple : < 6 I 4 I 2 >**

But : $E_N = \{(x^1,d^1),(x^2,d^2),\ldots,(x^N,d^N)\}$

$$\{x^k \rightarrow d^k\}^{k=1\ldots N}$$



Perceptron Multi-Couche (PMC)
Multi-Layer Perceptron (MLP)

Notations :

E : l 'ensemble des unités d 'entrée

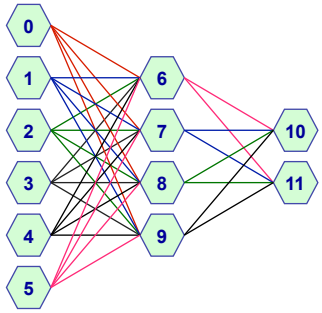
S : l 'ensemble des unités de sortie

$Amont(k)$: l 'ensemble des unités dont les sorties servent d'entrées à l 'unité k

$Aval(k)$: l 'ensemble des unités qui utilisent comme entrée la sortie de k

Par définition, on a :

$$\forall i \in E, \quad Amont(i) = \varnothing$$
$$\forall i \in S, \quad Aval(i) = \varnothing$$

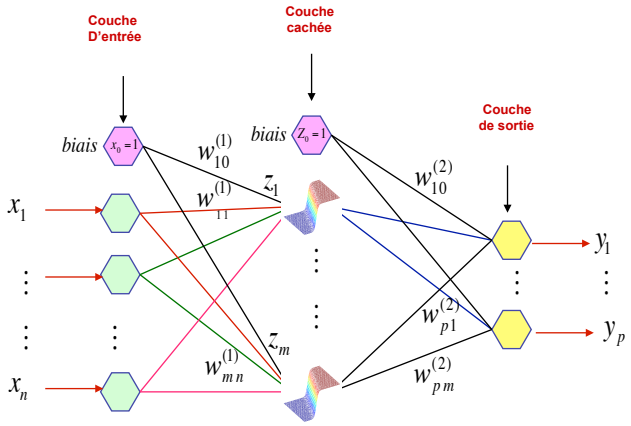


$Amont(7) = \{0,1,2,3,4,5\}$ $\quad Aval(7) = \{10,11\}$

$Amont(2) = \varnothing$ $\quad Aval(2) = \{6,7,8,9\}$

$Amont(11) = \{6,7,8,9\}$ $\quad Aval(11) = \varnothing$

Perceptron Multi-Couche (PMC)
Multi-Layer Perceptron (MLP)



Perceptron Multi-Couche (PMC)

Multi-Layer Perceptron (MLP)

L'activation de la j[ème] cellule cachée est :

$$a_j = w_{0j}^{(1)} + \sum_{i \in Amont(j)} w_{ij}^{(1)} x_i$$

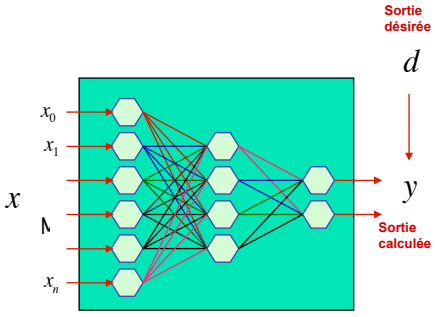
La sortie de cette i[ème] cellule cachée s'obtient par une transformation non linéaire de l'activation :

$$z_j = f(a_j)$$

De la même façon, l'activation et la sortie de la k[ième] unité de sortie peuvent s'obtenir comme suit :

$$a_k = w_{0k}^{(2)} + \sum_{j \in Amont(k)} w_{jk}^{(2)} z_j$$

$$y_k = f(a_k)$$



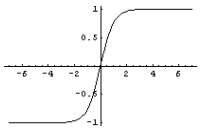
Si on combine le calcul des sorties des cellules cachées et celui des cellules de sortie on obtient pour la k[ième] sortie du réseau l'expression suivante :

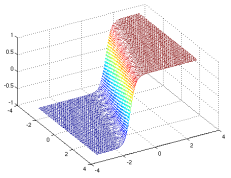
$$y_k = f\left(w_{0k}^{(2)} + \sum_{j \in Amont(k)} w_{jk}^{(2)} f\left(w_{0j}^{(1)} + \sum_{i \in Amont(j)}^{n} w_{ij}^{(1)} x_i \right) \right)$$

Perceptron Multi-Couche (PMC)

Types de fonction *f*

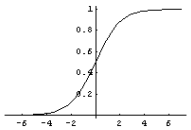
$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$





$$f(x) = \frac{e^x - 1}{e^x + 1}$$

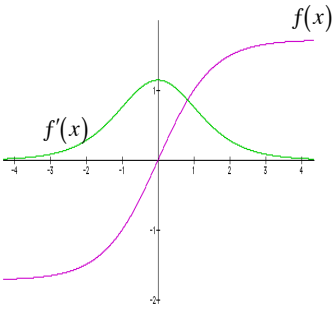
$$f(x) = \frac{1}{1 + e^{-x}} \rightarrow f'(x) = f(x)(1 - f(x))$$



$$f(x) = \frac{x}{1 + |x|}$$

Perceptron Multi-Couche (PMC)

Types de fonction *f*



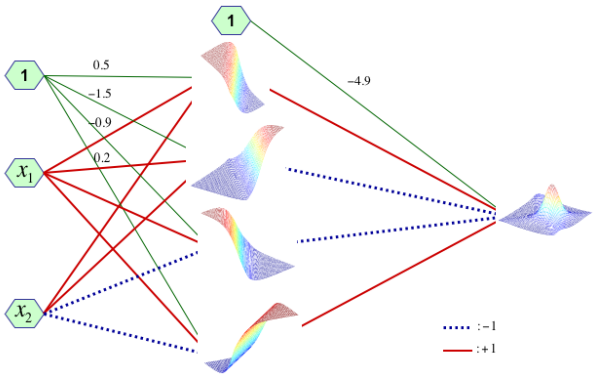
$$\tilde{f}(\tilde{x}) = \tanh(\tilde{x}) \quad \equiv \quad f(x) = \frac{1}{1+e^{-x}}$$

Transformations linéaires :

$$\tilde{x} = \frac{x}{2} \quad et \quad \tilde{f} = 2.f - 1$$

entrées sorties

Surfaces de séparation et PMC



Perceptron Multi-Couche (PMC)

Critères d 'apprentissage

$$R_{mse}(w) = \frac{1}{2}\sum_{k=1}^{N}\left(d^k - y^k\right)^2$$

$$R_{mse-pondéré}(w) = \sum_{k=1}^{N}\left(\left(d^k - y^k\right)\sum^{-1}\left(d^k - y^k\right)\right) \qquad avec \quad \sum^{-1} = \frac{1}{N}\sum_{k=1}^{N}\left(d^k - y^k\right)\left(d^k - y^k\right)^T$$

$$R_{multiple-\log istic}(w) = \sum_{k=1}^{N}\sum_{p=1}^{n}\left[d_p^k \log\frac{d_p^k}{y_p^k} + \left(1 - d_p^k\right)\log\frac{1 - d_p^k}{1 - y_p^k}\right]$$

$$R_{\log-likelihood}(w) = \sum_{k=1}^{N} d^k \log\frac{d^k}{p^k} \qquad avec \qquad p^k = \frac{e^{y^k}}{\sum_j e^{y^j}}$$

Perceptron Multi-Couche (PMC)

Règles d'adaptation

$$R(w) = \int_Z L(z,w)\, dp(z) \quad \rightarrow \quad \tilde{R}(w) = \frac{1}{N}\sum_{k=1}^{N} L\left(z^k, w\right)$$

$$w^+ = \arg\min_w \tilde{R}(w)$$

$$z = (x,d) \qquad v.a.(X,D) \qquad p(x,d) = p(x)p(d/x)$$

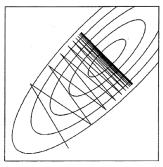
$$\tilde{R}_{mlp}^k(w) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i^k - d_i^k\right)^2$$

$$\tilde{R}_{mlp}^k(w) = \frac{1}{2}\sum_{m=1}^{n}\left(f\left(w_{m0}^{(2)} + \sum_{j\in Amont(m)} w_{mj}^{(2)} f\left(w_{j0}^{(1)} + \sum_{i\in Amont(j)} w_{ji}^{(1)} x_i\right)\right) - d_m^k\right)^2$$

Perceptron Multi-Couche (PMC)

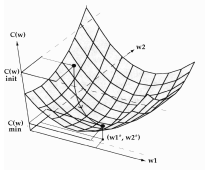
Règles d'adaptation

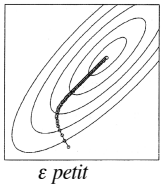
$$\tilde{R}^k_{mlp}(w) = \frac{1}{2}\sum_{m=1}^{n}\left(f\left(w^{(2)}_{m0} + \sum_{j\in Amont(m)} w^{(2)}_{mj} f\left(w^{(1)}_{j0} + \sum_{i\in Amont(j)} w^{(1)}_{ji} x_i\right)\right) - d^k_m\right)^2$$



ε grand

$$w_{ji}(t+1) = w_{ji}(t) - \varepsilon(t)\nabla_{w_{ji}}\tilde{R}^k_{mlp}(w)$$



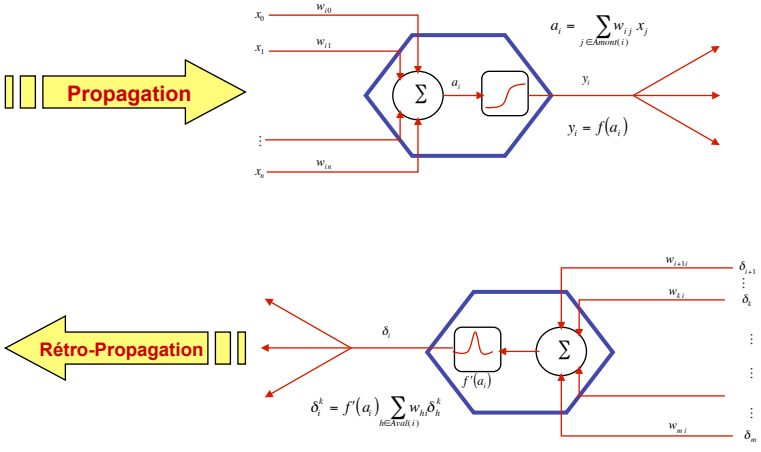


ε petit

$$\nabla_{w_{ji}}\tilde{R}^k_{mlp}(w) = \delta^k_j s_i$$

$$\begin{cases} \delta^k_j = f'(a_j)\left(y^k_j - d^k_j\right) & si \quad j \in Sortie \\ \delta^k_j = f'(a_j)\sum_{h\in Aval(j)} w_{hj}\,\delta^k_h & si \quad i \notin Sortie \end{cases}$$

Rétro-propagation du gradient



Perceptron Multi-Couche

Algorithme d 'apprentissage

1- Tirer au hasard w_0

2- Présentation d'une forme (x^k, y^k)

3- Calcul de l'état du réseau par propagation

$$x_j = f(a_j) \qquad a_j = w_{j0} + \sum_{i \in Amont(j)} w_{ji} x_i$$

4- calcul des signaux d 'erreur

$$\begin{cases} \delta_j^k = f'(a_j)(y_j^k - d_j^k) & si \quad j \in Sortie \\ \delta_j^k = f'(a_j) \sum_{h \in Aval(j)} w_{hj}\, \delta_h^k & si \quad i \notin Sortie \end{cases}$$

5- Adaptation les poids

$$w_{ji}(t+1) = w_{ji}(t) - \varepsilon(t)\delta_j^k s_i$$

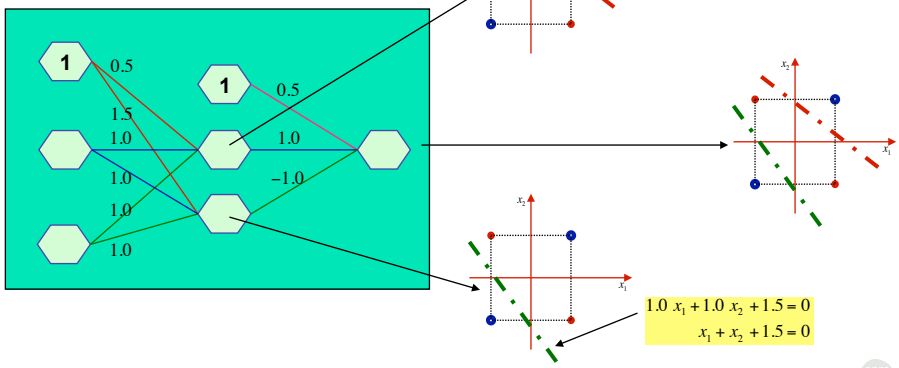
Où $\varepsilon(t)$ est le pas du gradient.

6- Répéter de 2 à 5 jusqu'à l'obtention d'une valeur acceptable de l'erreur

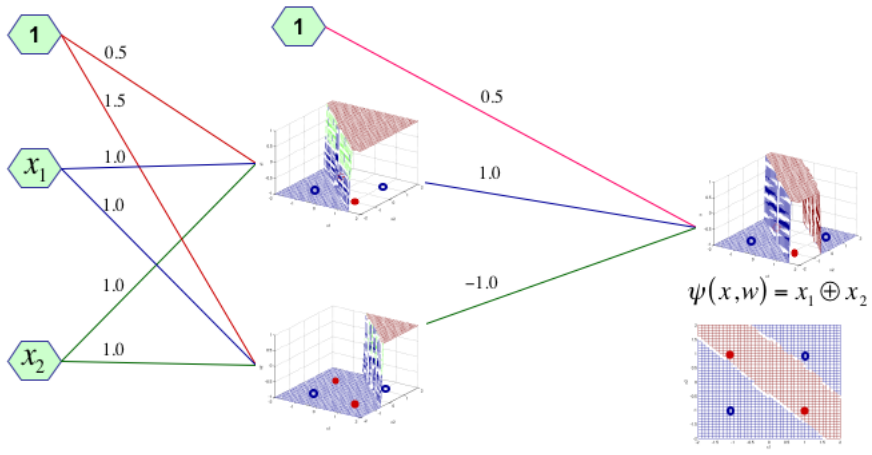
Exemple de PMC (MLP)

$$D = \left\{ \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, -1 \right); \left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} -1 \\ 1 \end{pmatrix}, 1 \right); \left(\begin{pmatrix} -1 \\ -1 \end{pmatrix}, -1 \right) \right\}$$

$\psi(x, w) = x_1 \oplus x_2$



Exemple de PMC (MLP)



Approximation universelle

Théorème [Hornik 1989*] :
Les PMC à trois couches avec des sorties linéaires, des cellules cachées saturantes croissantes "en nombre suffisant" peuvent approcher avec une précision arbitraire toute fonction bornée mesurable d'un espace de dimension finie dans un autre.
Un PMC est un approximateur universel.

Remarques :
– En théorie, il n'y aurait besoin d'aucune autre structure de réseau. Toutefois, dans les applications, il peut s'avérer plus pratique d'utiliser plusieurs couches, des sorties non linéaires, etc.

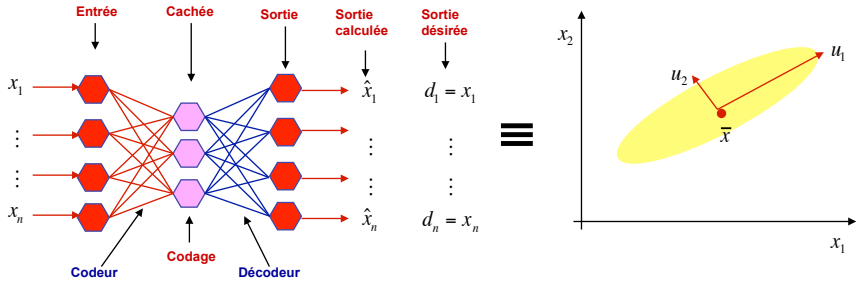
– C'est un théorème purement existentiel. Il ne dit pas comment déterminer un nombre approprié de neurones dans la couche cachée et les valeurs des poids pour approximer une fonction donnée avec une précision donnée !

Liens avec l'analyse factorielle

PMC & ACP * **

Un PMC auto-associatif avec des sorties linéaires, peut réaliser une transformation de Karhunen-Loève.

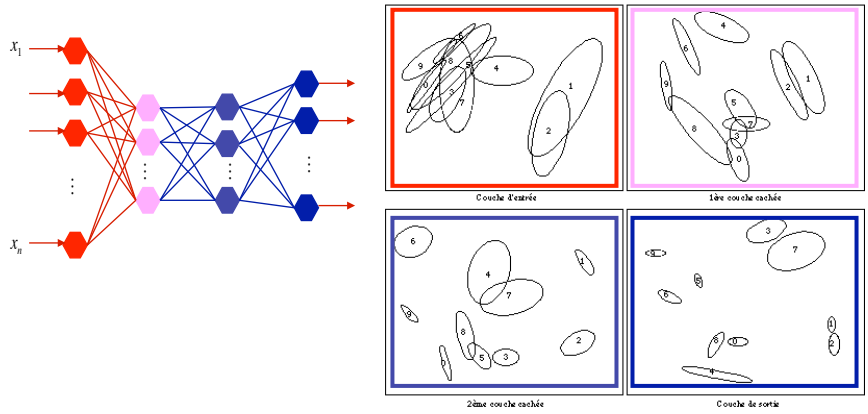
Un PMC *auto-associatif* est équivalent à une Analyse en Composantes Principales.



* Bourlard H. & Kamp Y. (1988) : «Auto-association by multilayer perceptrons and singular value decomposition»
Biological Cybernetics, Vol. 59, pp. 291-294.
** Baldi P. & Hornik K. (1989) : «Neural networks and principal component analysis: Learning from examples without local minima »
Neural Networks, Vol. 2, N° 1, pp. 53-58.

Liens avec l'analyse factorielle

PMC & AD * : analyse et interprétation dans le cas non-linéaire

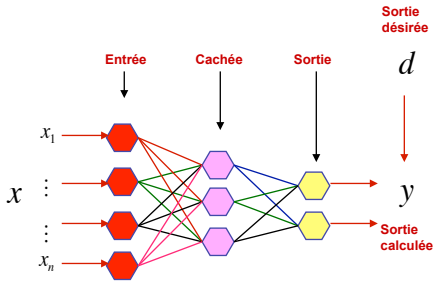


* Bennani Y. (1992) : «Approches Connexionnistes pour la Modélisation et l 'Identification»
Thèse de Doctorat, LRI-Université Paris 11, Orsay.

* Gallinari P., Thiria S., Badran F., Fogelman-Soulie F. (1991) : « On the relations between discriminant analysis and multilayer perceptrons»
Neural Networks, Vol. 4, N° 3, pp. 349-360.

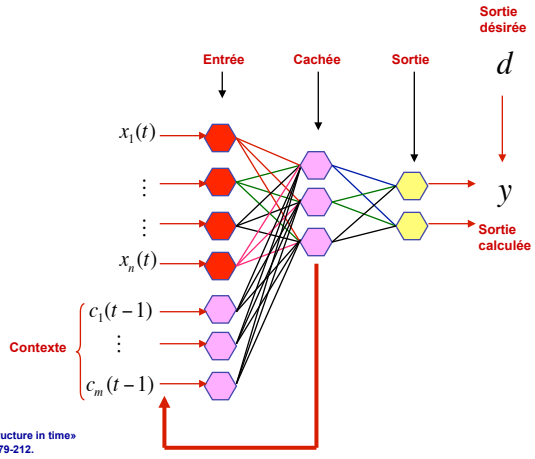
Réseaux structurés

Connexions complètes



Réseaux structurés

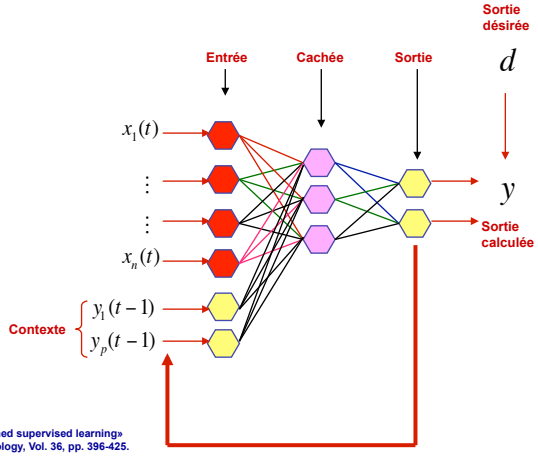
Connexions complètes avec contexte [Elman*]



* Elman J.L. (1990) : «Finding structure in time» Cognitive Science, Vol. 14, pp. 179-212.

Réseaux structurés

Connexions complètes avec contexte [Jordan*]

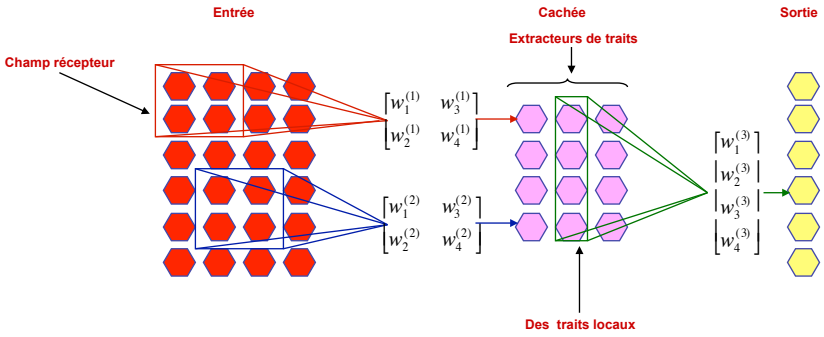


* Jordan M.I. (1992) : «Constrained supervised learning» Journal of Mathematical Psychology, Vol. 36, pp. 396-425.

Réseaux structurés

Connexions locales

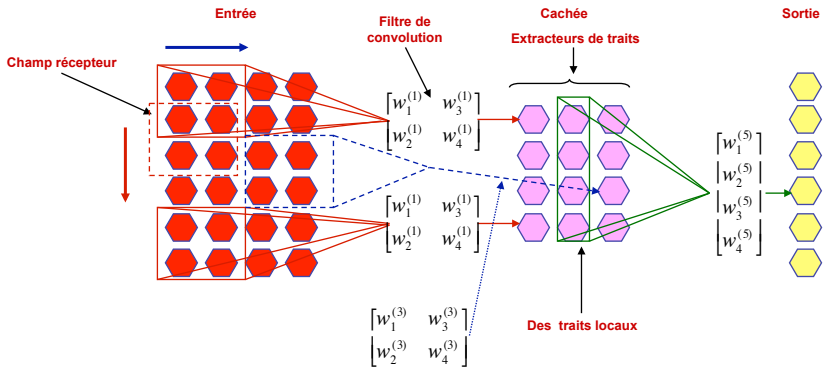
L'utilisation de connexions locales diminue très fortement le nombre de poids d'un réseau.



Réseaux structurés

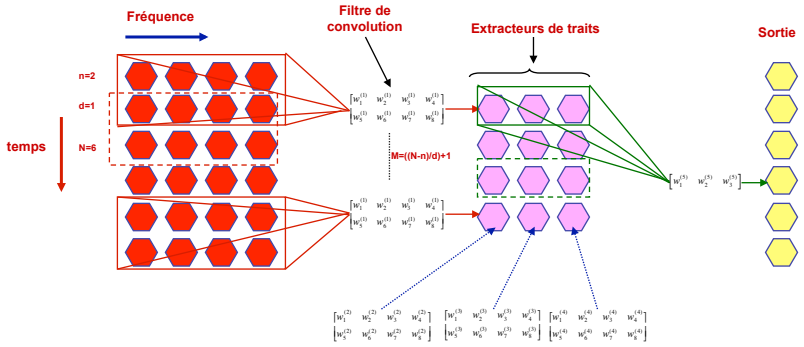
Connexions contraintes ou à poids partagés

Une propriété intéressante du mécanisme de partage des poids tient au nombre très faible de paramètres libres.

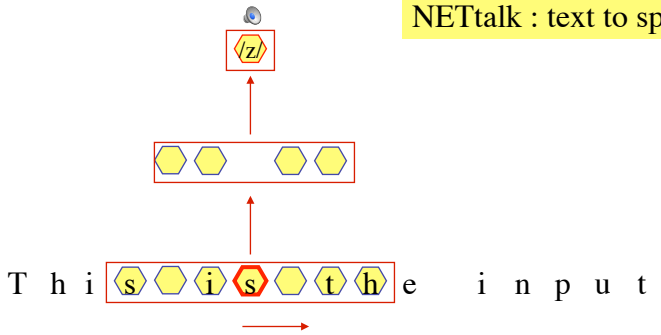


Réseaux structurés

TDNN (Time Delay Neural Network)



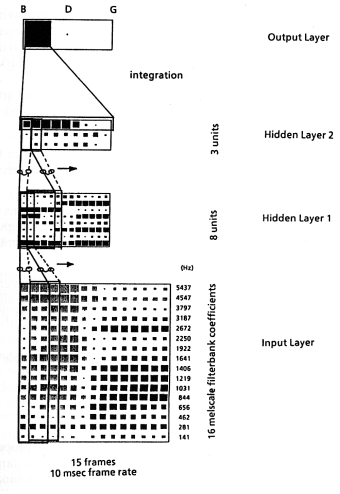
Réseaux structurés
Synthèse de la parole [Sejnowski & Rosenberg*]



NETtalk : text to speech

* Sejnowski T.J. & Rosenberg C.R. (1987) :
«Parallel Networks that learn to pronounce English text»
Complex systems, Vol 1, pp. 145-168.

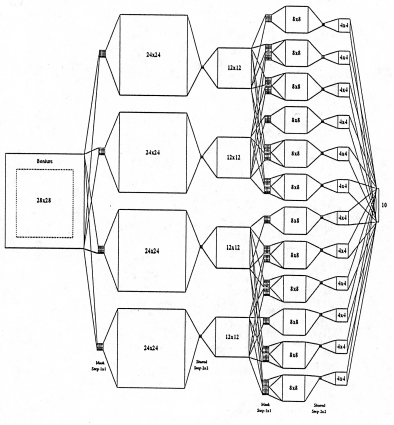
Réseaux structurés
Reconnaissance de la parole [Alex Waibel*]

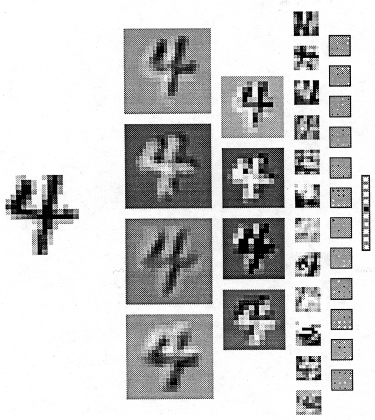


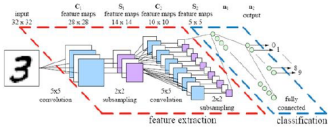
* Waibel A., Hanazawa T., Hinton G., Shikano K., Lang K. (1987) :
«Phoneme recognition using Time-Delay Neural Networks»
Tech. Rep. ATR, TR-1-006.

Réseaux structurés : Réseaux à convolution

LeNet pour la reconnaissance de chiffres
[Yann LeCun*]



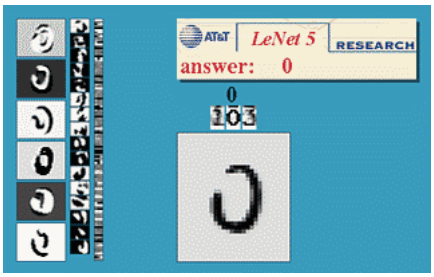




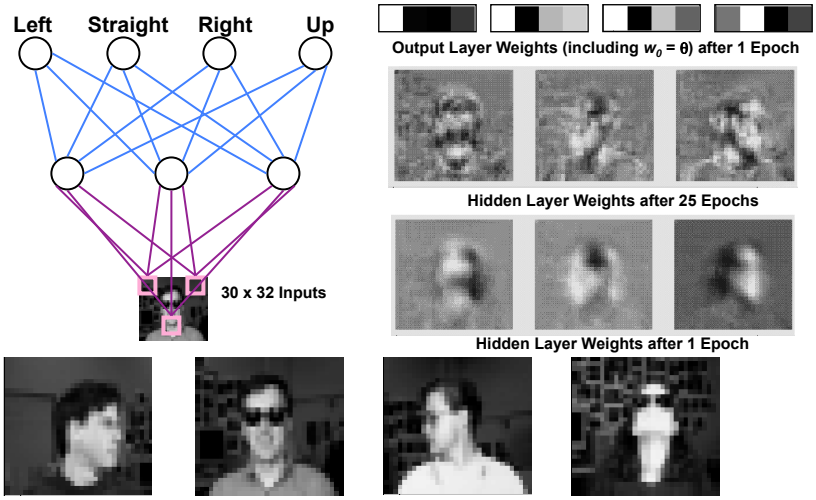
* Le Cun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D. (1989) : «Back-propagation applied to handwritten zip code recognition» Neural Computation, Vol. 1, pp. 541-551.

Réseaux structurés

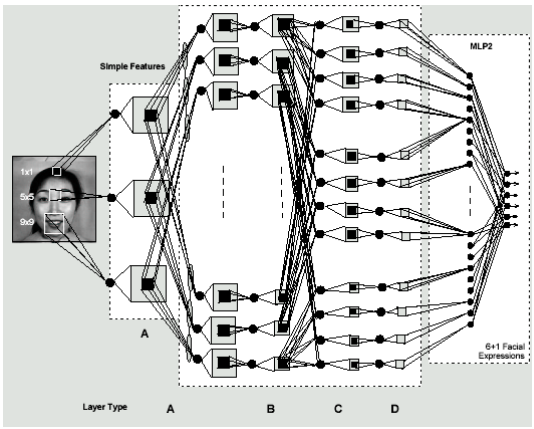
LeNet pour la reconnaissance de chiffres
[Yann LeCun*]



Réseaux structurés
reconnaissance de la position du visage



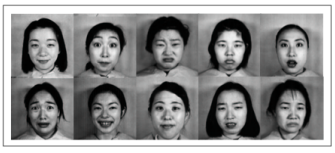
Réseaux structurés
reconnaissance d'expressions faciales



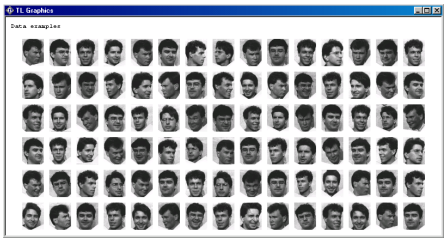
210 images (246x256 : TIFF)

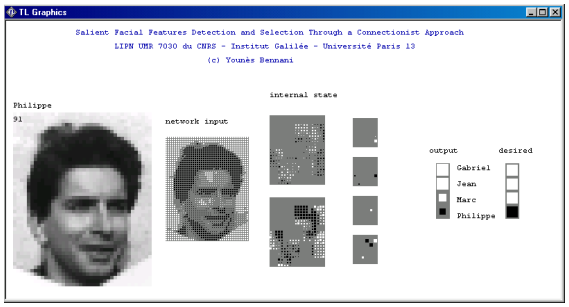
10 femmes japonaises

6 expressions + 1 position neutre



Réseaux structurés
identification de visages



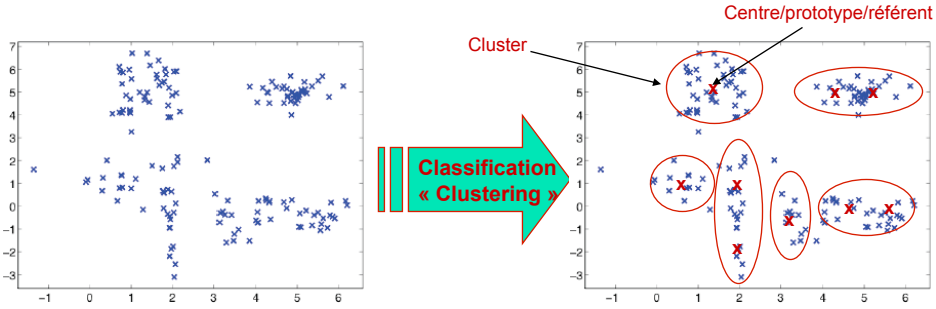


Systèmes d'Apprentissage Non Supervisé

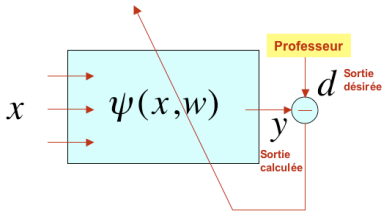
Classification non supervisée « Clustering »

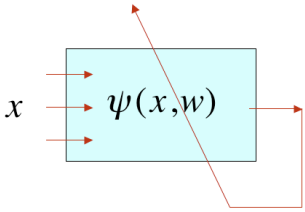
Objectif :
Définir sur un ensemble d'objets 2 à 2 comparables (ou une matrice de (dis)similarités), une partition (en groupes disjoints et complémentaires) qui respecte au mieux les ressemblances entre objets.

La ressemblance de 2 objets est grande lorsqu'ils figurent dans le même groupe et petite dans le cas contraire.



Supervisée vs Non-Supervisé





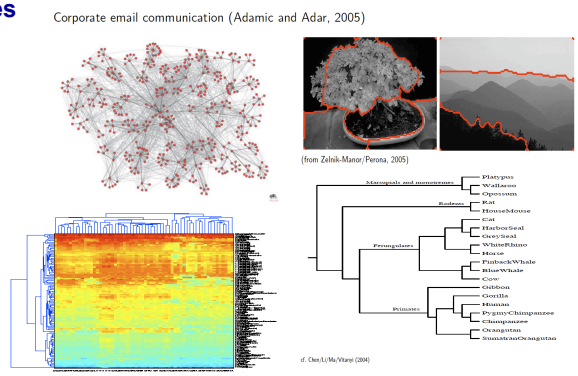
		Supervisé	Non supervisé
Coût	:	Erreur de classification	Inertie, …
Données	:	Etiquetées $x_N = \{(x^1,d^1),(x^2,d^2),\ldots,(x^N,d^N)\}$	Non étiquetées $x_N = \{x^1,x^2,\ldots,x^N\}$
Objectif	:	Prédiction	Exploration
# Classes	:	Connu	Inconnu
Généralisation	:	Perf. N^lles données	Qualité sur les mêmes données
Théorie	:	Mature	Jeune, très active

Classification Non-Supervisée

Domaines d'applications

Outil très utilisé en analyse exploratoire de données dans de nombreux domaines d'applications :

- **Sciences sociales**
- **Biologie**
- **Médecine**
- **Astronomie**
- **Diagnostic**
- **Images**
- **Marketing**
- **Web mining**
- **Texte mining**
- **Graphe mining**
- **Data mining**
- **…**



SOM : Self Organizing Map

Cartes auto-organisatrices/Cartes topologiques/Cartes de Kohonen

Développé par Kohonen, à partir des travaux de Willshaw & Von Der Malsburg

Cherche à transformer des signaux de dimension quelconque, en signaux à une ou deux dimensions.

Projeter les données initiales sur un espace discret et régulier de faible dimension.

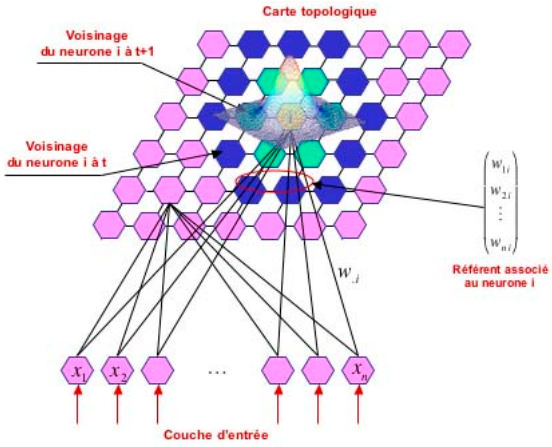
Les espaces utilisés sont des treillis réguliers dont chacun des nœuds est occupé par un automate + notion de voisinage entre automates.

On a, dans ces cartes, la propriété suivante :

des données similaires auront des projections proches sur la carte.

SOM : Self Organizing Map

Architecture du réseau



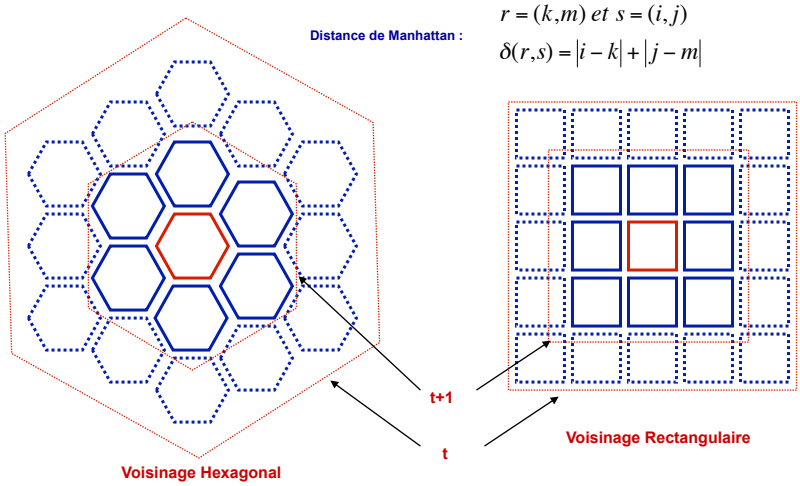
SOM : Self Organizing Map

Types de voisinage

Distance de Manhattan :

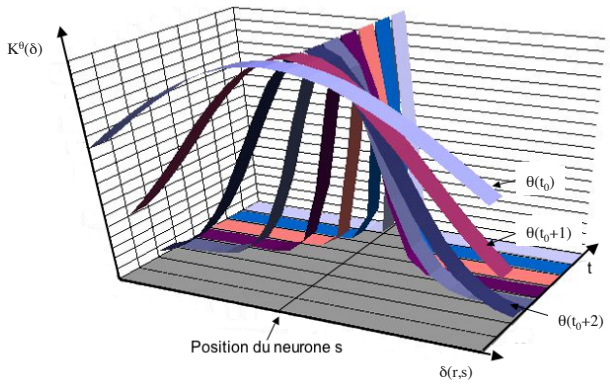
$$r = (k,m) \; et \; s = (i,j)$$

$$\delta(r,s) = |i-k| + |j-m|$$



SOM : Self Organizing Map

Fonction de voisinage



$$K^{\theta}(\delta) = \frac{1}{\theta(t)} e^{-\frac{\delta^2}{\theta^2(t)}}$$

$$\theta(t) = \theta_i \left(\frac{\theta_f}{\theta_i}\right)^{\frac{t}{N_{iter}-1}}$$

$$r = (k,m) \; et \; s = (i,j)$$
$$\delta(r,s) = |i-k| + |j-m|$$

SOM : Self Organizing Map

Formulation

Minimiser la fonction de coût :

$$R_{SOM}(w,\chi) = \sum_{i=1}^{N} \sum_{c=1}^{K} K^{\theta}(\delta(c,\chi(\vec{x}_i)) \left\| \vec{x}_i - \vec{w}_c \right\|^2$$

Calcul des gradients :

$$\nabla_{\vec{w}}(R_{SOM}) = \frac{\partial R_{SOM}}{\partial \vec{w}}$$

Règle d'adaptation :

$$\vec{w}_j^t = \vec{w}_j^{t-1} - \varepsilon^t \nabla_{\vec{w}}(R_{SOM})$$
$$\vec{w}_j^t = \vec{w}_j^{t-1} - \varepsilon^t K^{\theta}(\delta(j,\chi(\vec{x}_i)) \left(\vec{w}_j^{t-1} - \vec{x}_i\right)$$

Fonction d'affectation :

$$\chi(\vec{x}_i) = \arg\min_{1 \le c \le K} \left\| \vec{x}_i - \vec{w}_c \right\|^2$$

Fonction de voisinage :

$$K^{\theta}(\delta) = \frac{1}{\theta(t)} e^{-\frac{\delta^2}{\theta^2(t)}}$$

Distance entre unités :

$$r = (k,m) \; et \; s = (i,j)$$
$$\delta(r,s) = |i-k| + |j-m|$$

Température :

$$\theta(t) = \theta_i \left(\frac{\theta_f}{\theta_i}\right)^{\frac{t}{N_{iter}-1}}$$

SOM : Self Organizing Map

Algorithme d'apprentissage

Phase d'initialisation

1. Initialisation :

$$t = 0$$

$$w^0 = \left\{\vec{w}_1^0, \vec{w}_2^0, \ldots, \vec{w}_K^0\right\} \quad \vec{w}_i \in \Re^n$$

Phase de compétition

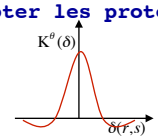
2. Présenter un exemple $\vec{x}$ d'apprentissage

3. Déterminer le gagnant :

$$\chi(\vec{x}_i) = \arg\min_{1 \le c \le K} \left\|\vec{x}_i - \vec{w}_c\right\|^2$$

Phase d'adaptation

4. Adapter les prototypes :



$$\vec{w}_j^t = \vec{w}_j^{t-1} - \varepsilon^t \mathrm{K}^\theta(\delta(j, \chi(\vec{x}_i)))\left(\vec{w}_j^{t-1} - \vec{x}_i\right)$$

$$\mathrm{K}^\theta(\delta) = \frac{1}{\theta^t(t)} e^{-\frac{\delta^2}{\theta^2(t)}} \qquad \theta(t) = \theta_i\left(\frac{\theta_f}{\theta_i}\right)^{\frac{t}{N_{iter}-1}} \qquad \varepsilon(t) = \varepsilon_i\left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{\frac{t}{N_{iter}-1}}$$

5. Incrémenter le nombre d'itération :

$$t = t + 1$$

6. Test d'arrêt : si $t < N_{iter}$ aller en 2.

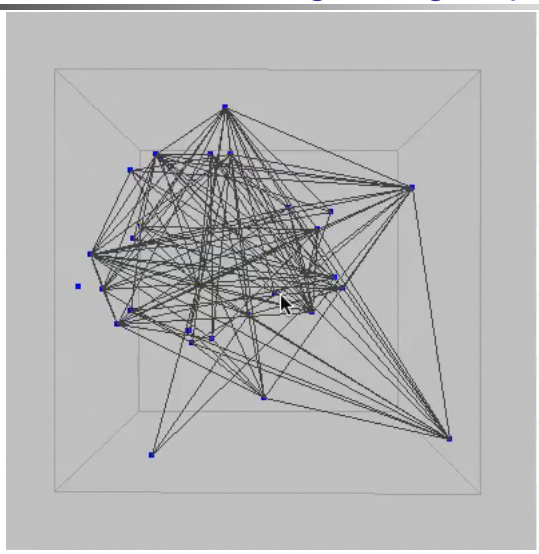
$\theta_i = 2, \quad \theta_f = 0.5$

$\varepsilon_i = 0.5, \quad \varepsilon_f = 0.005$

$N_{iter} = 10000$

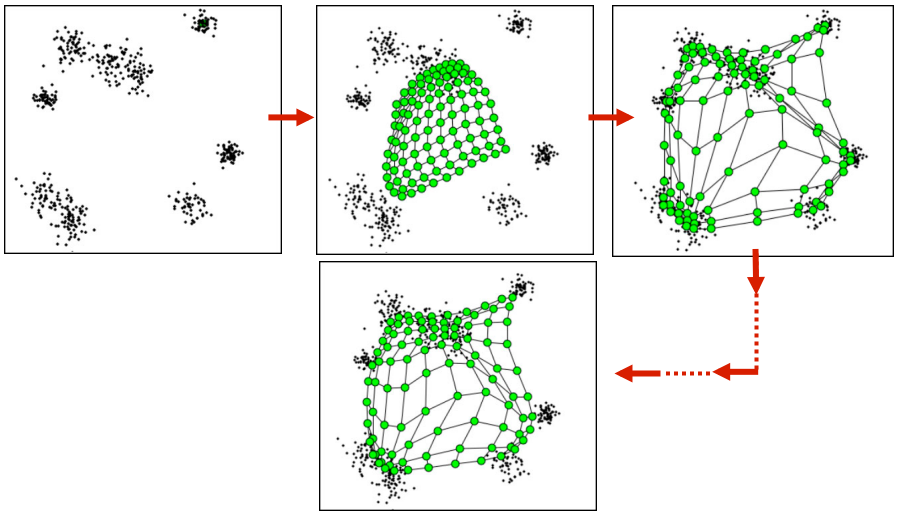
O(tkN), avec N le nombre d'objets,
t le nombre d'itérations et en général t et k << N

SOM : Self Organizing Map

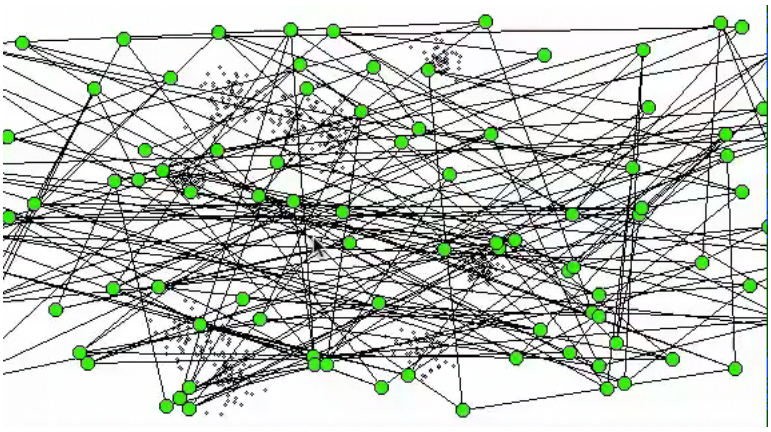


SOM : Self Organizing Map

Phase d'auto-organisation

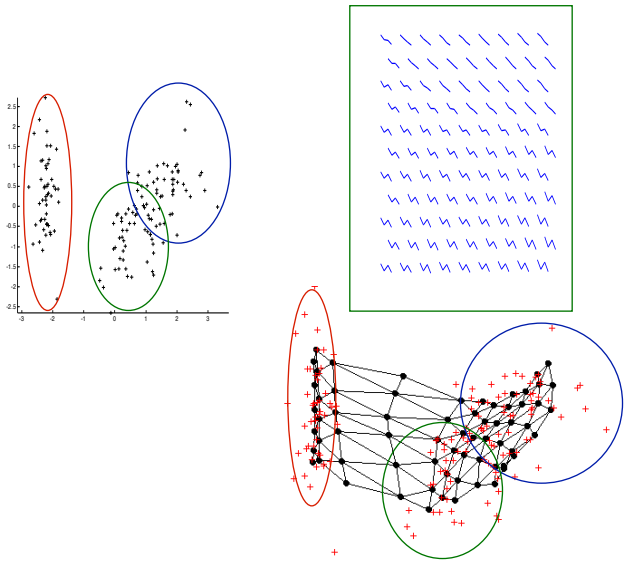


SOM : Self Organizing Map



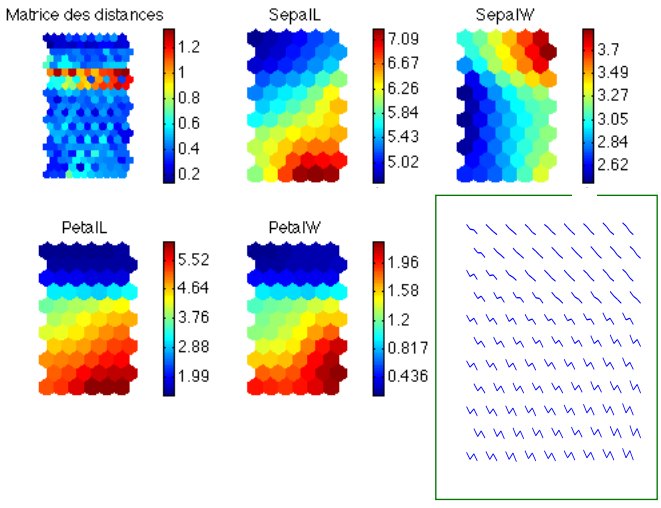
Apprentissage Non-Supervisé

SOM : Self Organizing Map : Iris



Apprentissage Non-Supervisé

SOM : Self Organizing Map : Iris

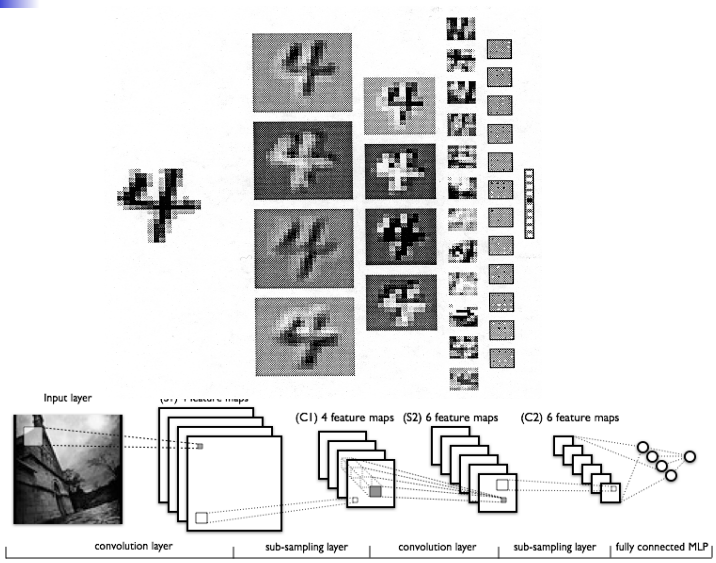


Systèmes d'Apprentissage profond

Deep Learning (Travaux pionniers)

- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks
- Many layered MLP with backpropagation
  - Tried early but without much success
    - Lent
    - Diffusion du gradient
- Présentation du deep networks avec apprentissage non supervisé

Convolutional Neural Networks



Apprentissage des Deep Networks

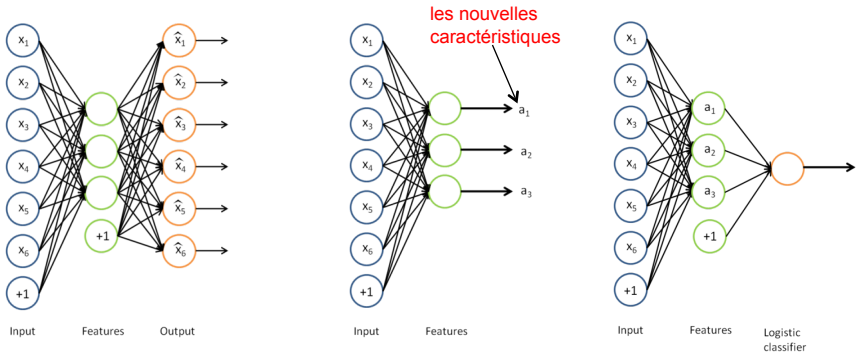
- Construire un espace de représentation (feature space)
  - Notez que c'est ce que nous faisons avec les noyaux SVM, ou les couches cachées dans MLP, etc, mais maintenant, nous allons construire l'espace de représentation en utilisant les architectures profondes.
  - Apprentissage non supervisé entre les couches peut décomposer le problème en sous-problèmes distribués (avec des niveaux d'abstraction plus élevés) à être encore décomposé à des couches successives

Le défi d'entraîner des réseaux de neurones profonds

- Difficultés d'apprentissage supervisé des réseaux profonds
  - Les premières couches du MLP ne sont pas bien apprises
    - Diffusion du Gradient - erreur s'atténue à mesure qu'elle se propage aux couches précédentes : le gradient se propage « mal » de la sortie vers l'entrée.
    - Conduit à un apprentissage très lent.
    - Plus le réseau est profond plus le degré de non-linéarité du réseau augmente, ce qui augmenterait les chances de trouver ces obstacles à l'optimisation.
    - Les couches inférieures restent avec des transformations pas très utiles de l'entrée.
    - Besoin d'un moyen pour aider les premières couches à faire un travail efficace
  - Souvent pas suffisamment de données étiquetées disponibles
    - Pouvons-nous utiliser des approches non supervisées / semi-supervisées pour profiter des données non étiquetées
  - Réseaux profonds ont tendance à avoir des problèmes de minima locaux plus que les réseaux peu profonds pendant l'apprentissage supervisé

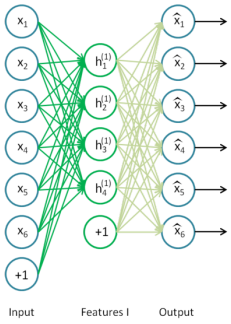
Auto-Encoders

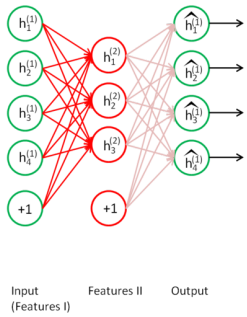
- Un type d'apprentissage non supervisé qui tente de découvrir les caractéristiques génériques des données
  - Apprendre la fonction identité par apprentissage des sous-caractéristiques importantes des données
  - Compression, etc
  - Peut utiliser seulement les nouvelles caractéristiques dans la nouvelle série de d'apprentissage ou concaténer les deux



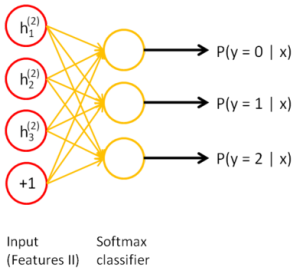
Stacked Auto-Encoders

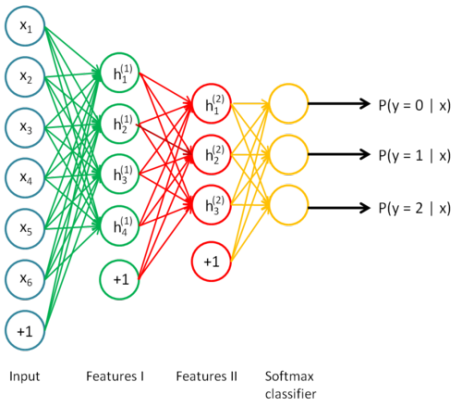
- Bengio (2007)
  Empilez nombreux auto-encodeurs en succession
  Déposer la couche de sortie de décodage à chaque fois



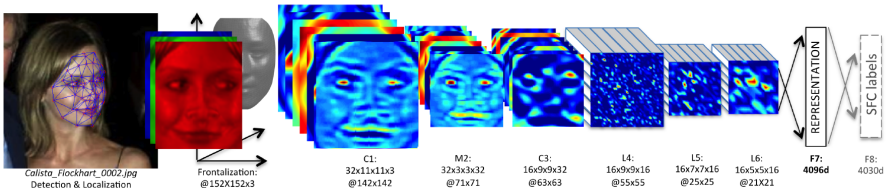


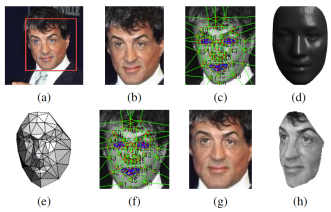
Stacked Auto-Encoders





Un exemple : Deep Face





DeepFace: Closing the Gap to Human-Level Performance in Face Verification

Yaniv Taigman Ming Yang Marc'Aurelio Ranzato Lior Wolf

Facebook AI Group
Menlo Park, CA, USA
{yaniv, mingyang, ranzato}@fb.com

Tel Aviv University
Tel Aviv, Israel
wolf@cs.tau.ac.il

Deep Learning

References

Bengio, Y. (2008). Learning deep architectures for AI. *Foundations and Trends in Machine Learning, to appear.*

Bengio, Y., Delalleau, O., & Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, pp. 107–114. MIT Press, Cambridge, MA.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19*, pp. 153–160. MIT Press.

Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., & Weston, J. (Eds.), *Large Scale Kernel Machines*. MIT Press.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In Schölkopf, B., Platt, J., & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19*. MIT Press.

Conclusion

• Aujourd'hui, on observe un retour en force des réseaux de neurones artificiels avec des résultats impressionnants.

• Les réseaux de neurones artificiels (RN) forment un ensemble de techniques
  • matures
  • efficaces
  • avec une base théorique solide
  • largement utilisées dans de nombreux domaines
    • Médecine (diagnostic, prothèses, conseils d'urgence)
    • Prospection minière et pétrolière
    • Reconnaissance vocale, écriture manuscrite
    • Télécommunication (compression des données)
    • Finance (estimation immobilier, détection fausses déclarations, prédiction des cours)
    • Industrie (mesure, prédiction contrôle)
    • Transports (pilotage automatique, détection de risques, détection de pannes)

De quoi n'a-t-on pas parlé ?

• Réseaux de neurones récurrents

• Autres modèles supervisés et non supervisés (LVQ, RBF, ART, …)

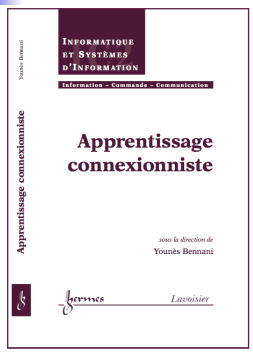
• Architectures hybrides, modulaires

• Analyse du comportement (Dilemme Biais-Variance, VC-dim, …)

• Problème de la généralisation

• Ajustement de la capacité de généralisation
  • Régularisation Formelle
  • Régularisation Structurelle
• …

Bibliographie



• *Apprentissage Connexionniste*
Younès Bennani
Editions Hermès Science (2006)
ISBN: 2-7462-1337-0. (100 €)

• *Réseaux de neurones*
G. Drefus, JM Martinez, M. Samuelides, MB Gordon, F. Badran, S. Thiria, L. Hérault
Editions Eyrolles(2002) ISBN: 2-212-11019-7. (50 €)

• *Neural Networks for Pattern Recognition*
Christopher M. Bishop
Clarendon Press - Oxford (1995)

• *Neural Smithing*
Supervised Learning in Feedforward Artificial Neural Networks
Russell D. Reed & Robert J. Marks
Massachusetts Institute of Technology Press (1999)

• *Pattern Recognition and Neural Networks*
B.D. Ripley
Cambridge University Press (1996)

• *Neural Networks*
James A. Freeman & David M. Skapura
Addison-Wesley Publishing Compagny (1991)

• *Adaptive Pattern Recognition and Neural Networks*
Yoh-Han Pao
Addison-Wesley Publishing Compagny (1989)

• *Neural Networks in Computer Intelligence*
LiMin Fu
Massachusetts Institute of Technology Press (1994)