وزارة التعليـــم العـــالـــي و الـبحـث الـعلمـــي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جـا معة جـيجـل
Université de Jijel
كليـة العلوم والتكنولوجيا
Faculté des Sciences et de la Technologie
قسم الهندسة الكهربائية
Département de Génie Electrique

Réseaux de neurones artificiels

Master II : Electrotechnique Industrielle

Module : Techniques d'intelligences artificielles

Introduction

- A première vue, la biologie et la science de l'ingénieur n'ont rien en commun, et pourtant nous pouvons affirmer que certaines connaissances en biologie constituent des inspirations de certains domaines de la science de l'ingénieur.

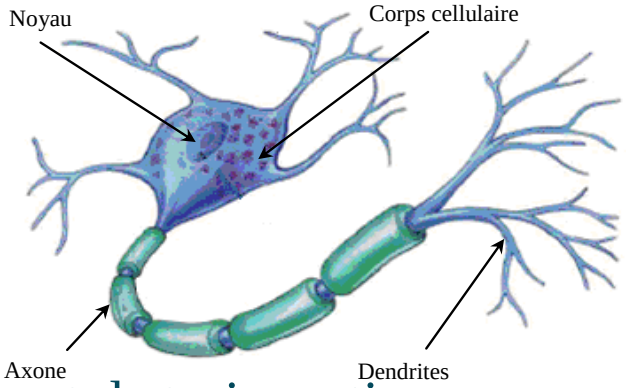
- Le rêve de créer une machine dotée d'une forme d'intelligence est présente depuis fort longtemps dans l'imagination humaine. Comment l'homme fait-il pour penser, raisonner ou même éprouver des sentiments? Des recherches menées par des scientifiques ont aboutis à l'étude des réseaux de neurones artificiels.

Introduction

- L'objectif de ce cours est multiple:
  - Rappeler les définitions fondamentales relatives aux réseaux de neurones ainsi que leur propriétés mathématiques.
  - Décrire les principaux types des réseaux de neurones ainsi que des détailles sur les types de réseaux de neurones les plus utilisés dans notre domaine (MLP et RBF) et plus particulièrement ces propriétés et sa mise en œuvre.

Le neurone biologique

- Le neurone est l'unité de base du système nerveux. Il possède environ cent milliards de neurones, il présente cinq caractéristiques:
  - Recevoir des signaux provenant de neurones voisins
  - Intégrer ces signaux
  - Engendrer un flux nerveux
  - Le conduire
  - Le transmettre à un autre neurone.



- Le neurone est une cellule constituée principalement de trois parties:
  - Les dendrites: Sont des fines extensions, elles collecte les signaux venant d'autres cellule ou de l'extérieur.
  - Le corps cellulaire: Il contient le noyau du neurone, ils recueil les informations reçues.
  - L'axone: Il sert comme moyen de transport des signaux émis par le neurone.

Le neurone artificiel

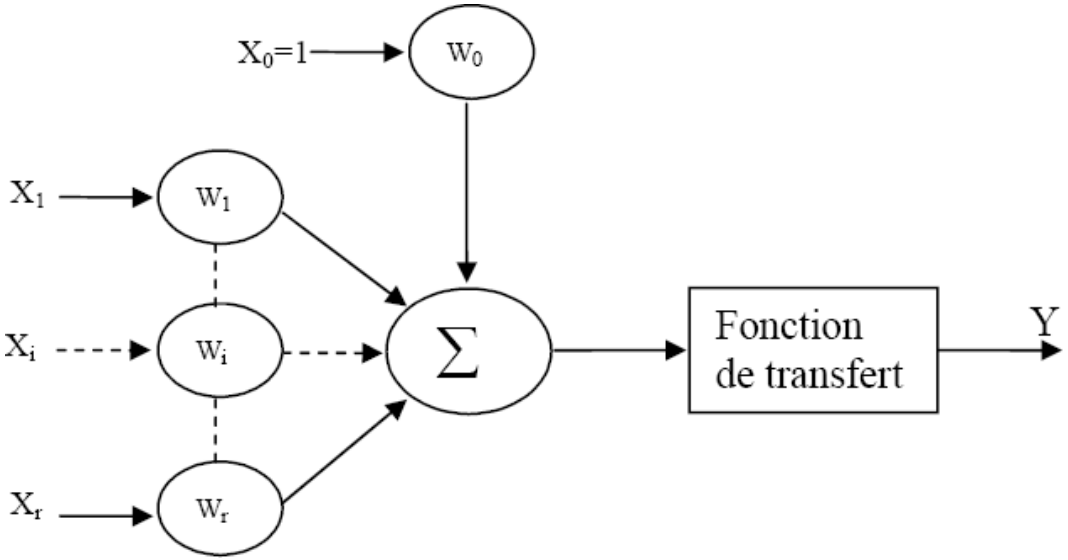
- Les réseaux de neurones artificiels (formels) sont à l'origine, une tentative de modélisation mathématique du cerveau humain. Un neurone formel est un processeur très simple, simulé sur ordinateur ou réalisé sur circuit intégré, modélisant le fonctionnement d'un neurone biologique.

Neurone artificiel élémentaire

- Chaque neurone reçoit un nombre variable d'entrées en provenance des neurones amonts. A chacune de ces entrées est associée:

Le neurone artificiel

- Un poids *Wi*: abréviation de Weight (poids) représentatif de la force de la connexion,
- Le seuil *W0*: peut être envisagé comme le coefficient de pondération de l'entrée *X0*, dont la valeur est fixée à 1.



- Chaque neurone élémentaire est doté d'une fonction de transfert (fonction d'activation) qui donne une sortie unique *Y*, et se ramifie ensuite pour alimenter un autre neurone aval.

Le neurone artificiel

- La phase qui suit c'est le calcul de la somme pondérée des entrées X_i, à partir de cette valeur, une fonction de transfert f calcule la valeur de l'état du neurone selon l'expression suivante:

$$Y = f(W_0 + \sum_{i=1}^{r} W_i \cdot X_i) \qquad (1)$$

Ou bien:

$$Y = f(\sum_{i=0}^{r} W_i \cdot X_i) \qquad (2)$$

Où:

r est le nombre des entrées

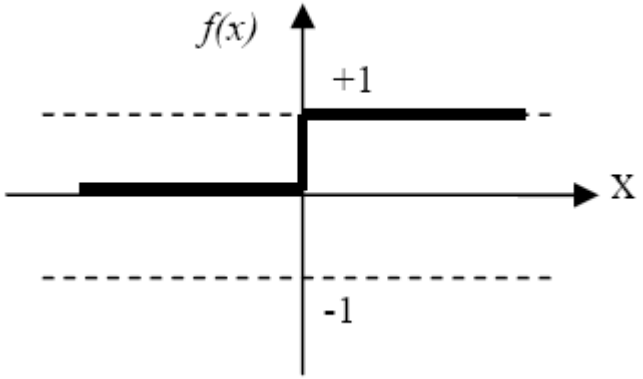
C'est cette valeur qui sera transmise aux neurones avals.

Fonctions de transfert

- Il existe de nombreuses formes possibles pour les fonctions de transfert. Les plus courantes sont présentées dans ce qui suit, avec leurs équations mathématiques.

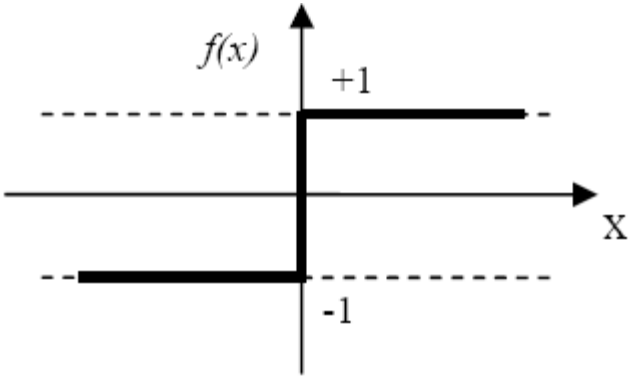
Binaire

$$f(x) \begin{cases} 0 \text{ si } x<0 \\ 1 \text{ si } x\geq 0 \end{cases}$$



Signe

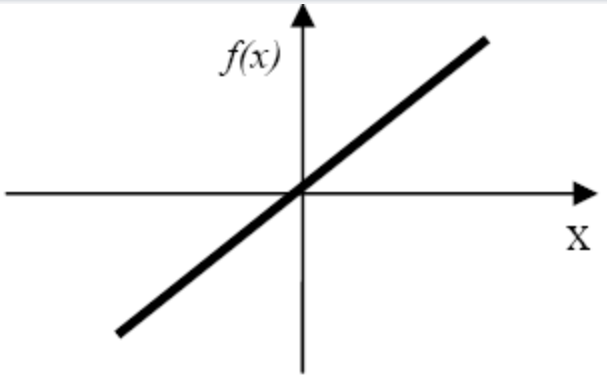
$$f(x) \begin{cases} 1 \text{ si } x>0 \\ -1 \text{ si } x\leq 0 \end{cases}$$



Fonctions de transfert

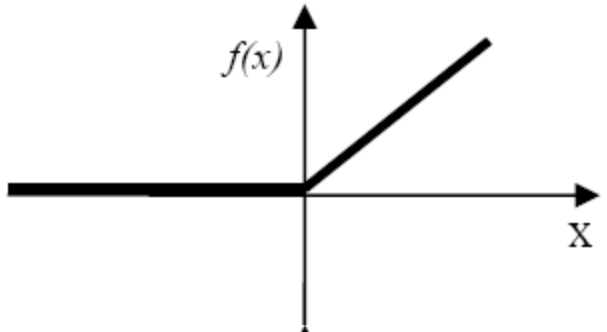
Identité

$$F(x)=x$$



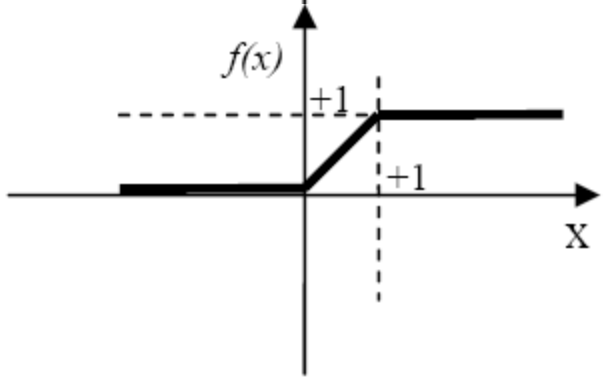
Linéaire positif

$$f(x)\begin{cases} 1 \text{ si } x<0 \\ x \text{ si } x\geq 0 \end{cases}$$



Saturé positif

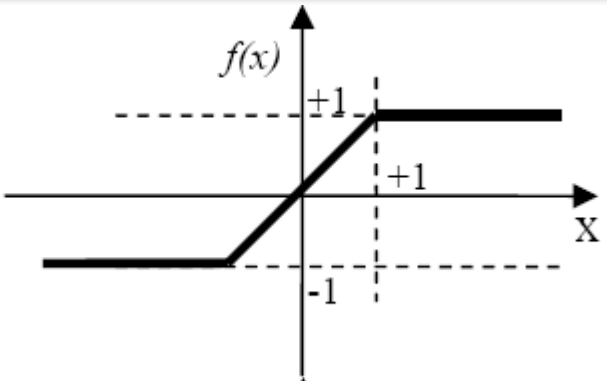
$$f(x)\begin{cases} 0 \text{ si } x<0 \\ 1 \text{ si } x\geq 1 \\ x \text{ si non} \end{cases}$$



Fonctions de transfert

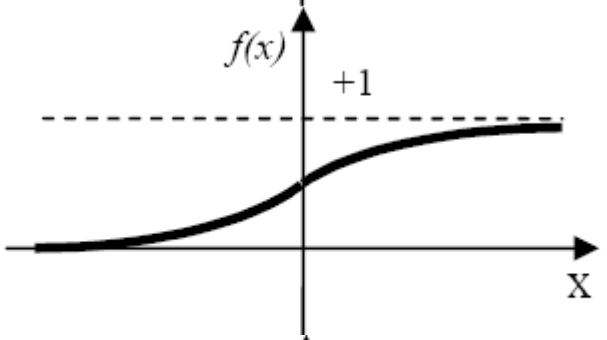
Saturé symétrique

$$f(x) \begin{cases} -1 \text{ si } x \leq -1 \\ 1 \text{ si } x \geq 1 \\ x \text{ si non} \end{cases}$$



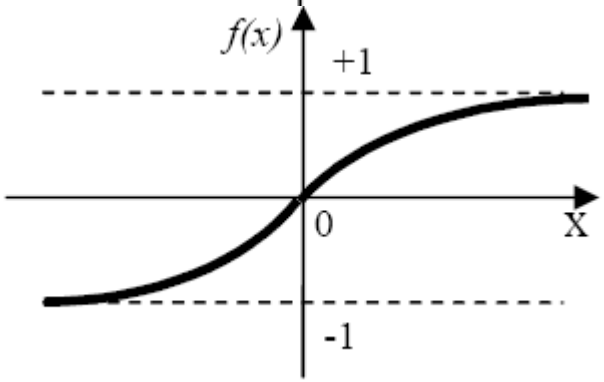
Sigmoïde

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



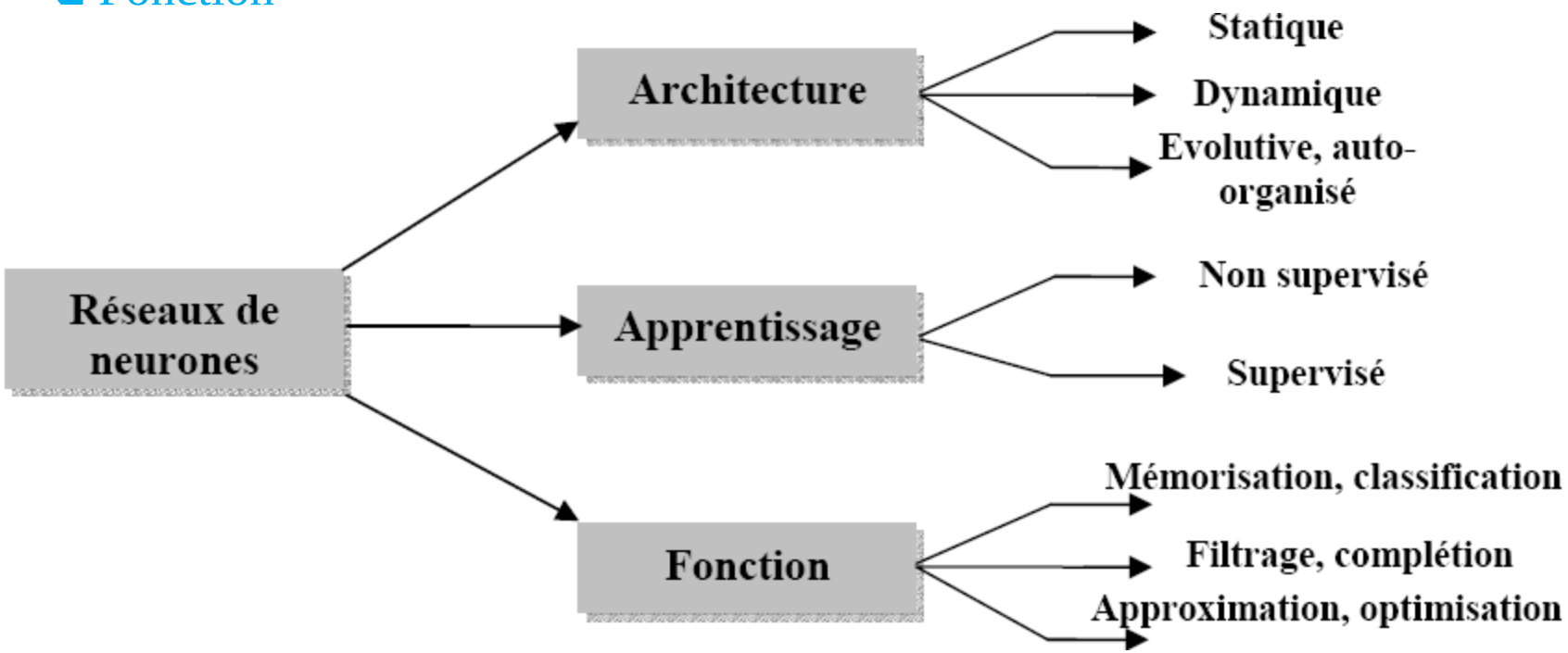
Tan-sigmoïde (Tanh)

$$f(x) = \frac{2}{1 + e^{-X}} - 1$$



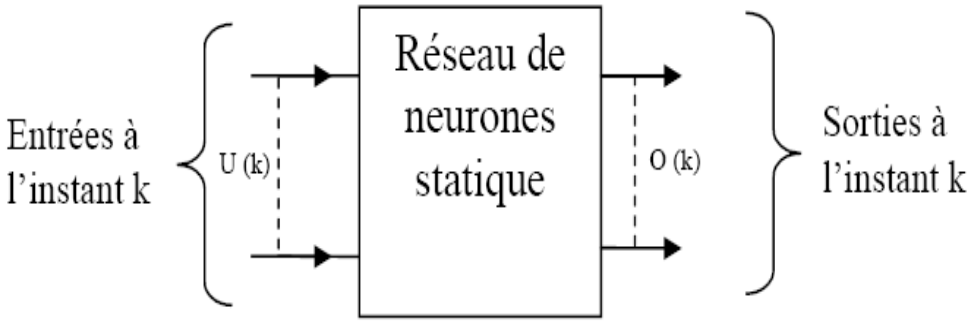
Classification des réseaux de neurones

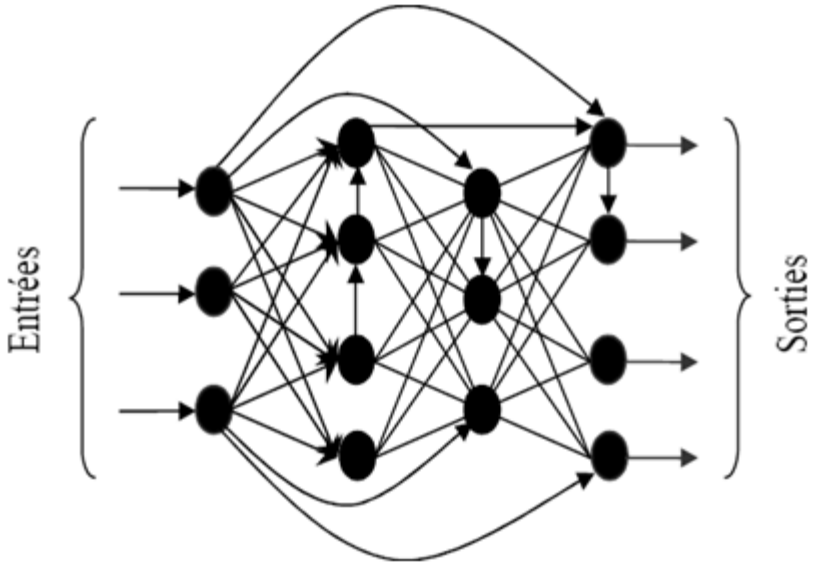
- Il existe trois différentes possibilités suivant lesquelles on peut classer les réseaux de neurones:
  - Architecture
  - Apprentissage
  - Fonction



Architecture

- Il existe trois types de réseaux à architectures différentes:
  - **Réseaux statiques:**
    - Ce type de réseaux est organisé généralement en couches de neurones.
    - Le neurone d'une couche reçoit ses entrées de la couche précédente.
    - Dans ces réseaux, il n'existe pas de retour de l'information (*feedback*).
    - Ils sont utilisés pour la classification ou l'approximation de fonction.

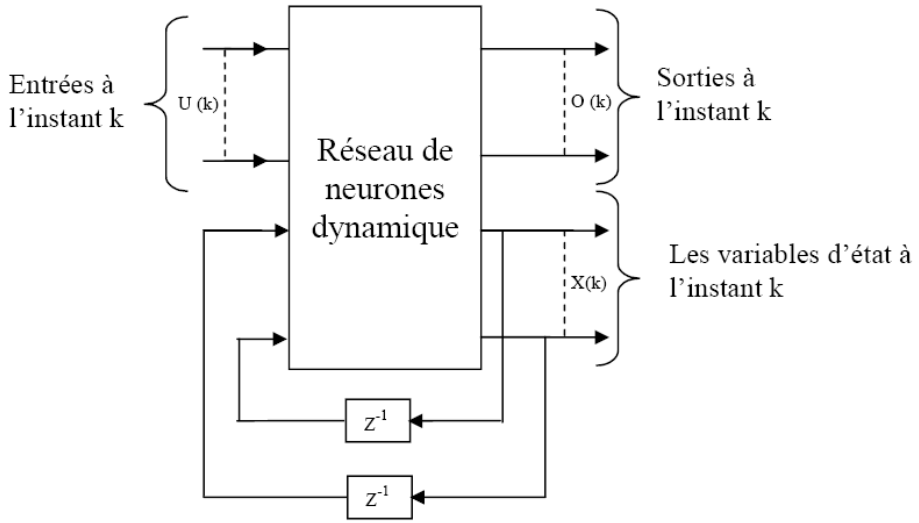


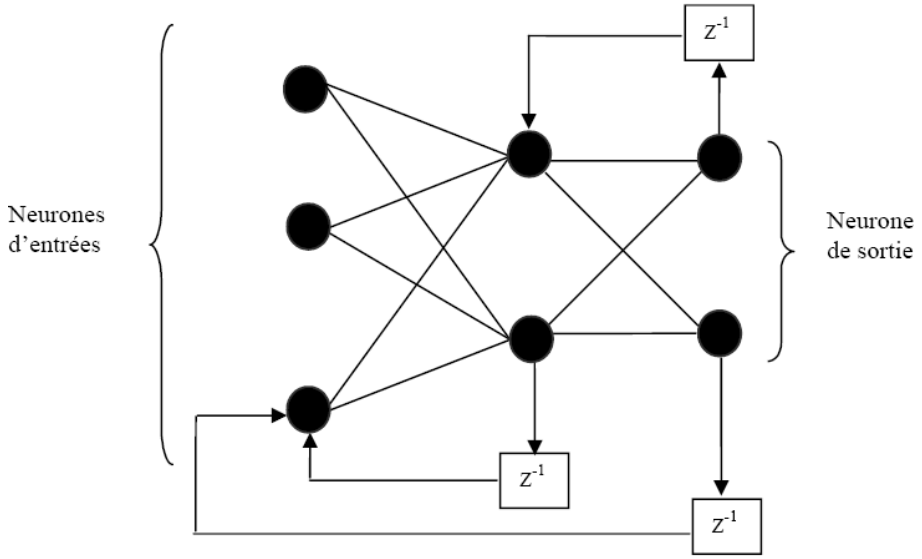


Architecture

Réseaux dynamiques:

- L'introduction de *feedback*, rend le réseau dynamique.
- Prise en compte de l'aspect temporel du problème.
- Souvent utilisés pour les problèmes de la commande de systèmes dynamiques et de mémorisation.





Réseaux à architecture auto-organisée:

- Les réseaux auto organisés sont des réseaux de neurones qui changent leur structure interne pendant l'utilisation.

Apprentissage

- L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré, il fait appel à des exemples de comportement.

- **Objectif:**
  - Amélioration des performances futures du réseau, sur la base d'une connaissance acquise au fur et à mesure des expériences passées.
  - Adaptation des poids de façon à ce que le réseau apprend le comportement du processus, via une base de données acquise on-line (en temps réel) ou off-line (en temps différé).

Apprentissage supervisé

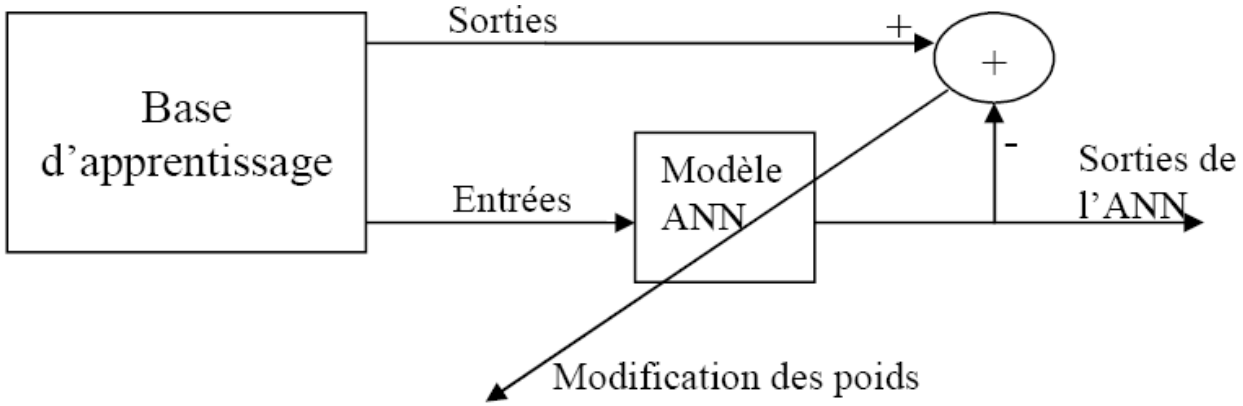
Apprentissage supervisé:

- Il se fait en présence d'un superviseur (Teacher).
- On désire apprendre au réseau un comportement de référence précis.

Exemple

- On fournit une série d'exemples x et de résultats y.
- L'algorithme doit trouver w tel que $y = f_w(x)$ (avec une bonne généralisation)

Remarque : Parmi les algorithmes d'apprentissage
- Méthode de Backpropagation
- Méthodes du Gradient stochastique (à voir pour les prochains cours)



Apprentissage semi-supervisé et non supervisé

Apprentissage semi-supervisé (renforcement)

- On fournit des exemples et des indications sur le résultat (vrai/faux).
- Dans ce cas, le réseau réajuste ses poids suivant un critère de performance. Celui-ci renforce les poids du réseau si le critère est favorable et les punit dans le cas contraire.

Apprentissage non supervisé:

- Nécessite la présence des entrées seulement.
- L'apprentissage non supervisé est bien adapté à la modélisation des données complexes (images, sons, ...).

Exemple

- On fournit seulement des exemples x à l'algorithme
- Il doit trouver w tel que les x soient correctement groupés selon f_w (avec une bonne généralisation)

Réseaux de neurones à apprentissage supervisée

- Le réseau à apprentissage supervisé représente la classe de réseaux qui a fait l'objet de plus grand nombre de travaux de recherche, et qui a connu une grande évolution depuis les années 90. Parmi ces réseaux:
  - Perceptron multicouches MLP
  - Réseau à base de fonction radiale RBF
  - Réseau d'ordre supérieur Pi-sigma et RPN

- Le but est de ramener le réseau vers le comportement désiré (imposé par le superviseur), ceci se fait par la recherche du vecteur des poids optimal *w*, parmi toutes les combinaisons possibles dans l'espace des poids.
- Dans ce genre d'apprentissage, il faut déterminer une fonction qui mesure l'écart entre les sorties désirées et celles fournies par le réseau (fonction d'erreur ou critère):

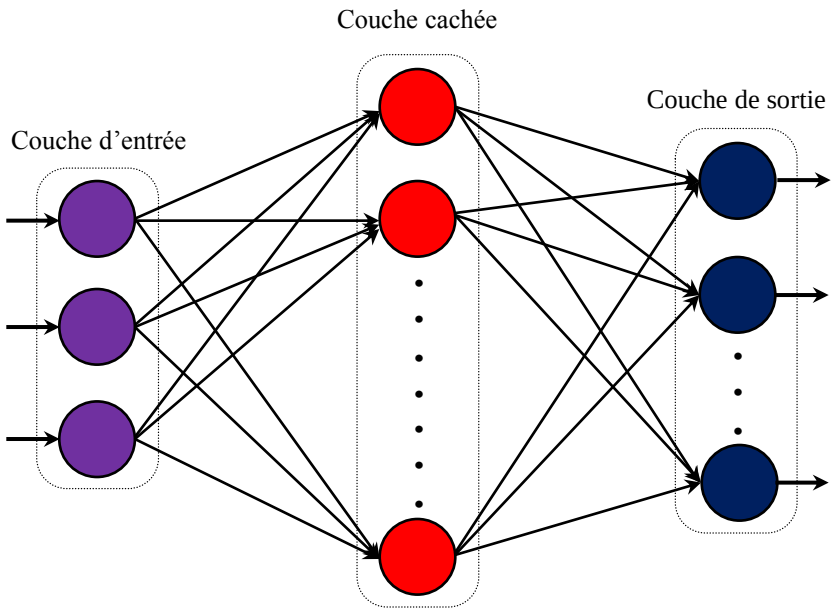
$$E = \psi(Y_k - d_k)$$

Où d_k est la sortie désirée et Y_k représente l'estimation du réseau (sortie du réseau).

Perceptron multicouche MLP

- Parmi les types des réseaux de neurones les plus utilisés on trouve le MLP avec son algorithme d'apprentissage, la rétro-propagation des erreurs.
- Le perceptron multicouche est un réseau orienté de neurones artificiels organisé en couches, où l'information voyage dans un seul sens, de la couche d'entrée vers la couche de sortie.

Remarque: La couche d'entrée présente toujours une couche virtuelle associée aux entrées du système, elle ne contient aucun neurone.



Propriétés fondamentales du MLP

- **L'approximation universelle**

▪ *Toute fonction bornée suffisamment régulière peut être approchée uniformément, avec une précision arbitraire, dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachée en nombre fini, possédant tous la même fonction d'activation, et un neurone de sortie linéaire.*

- **La capacité de généralisation**

▪ *L'intérêt des réseaux de neurones réside précisément dans leur capacité à la généralisation, c'est-à-dire leur aptitude de donner une réponse satisfaisante à une entrée qui ne fait pas partie des exemples à partir des quels il a appris.*

Mise en œuvre du réseau de neurones MLP

- La mise en œuvre des réseaux de neurones comporte à la fois:

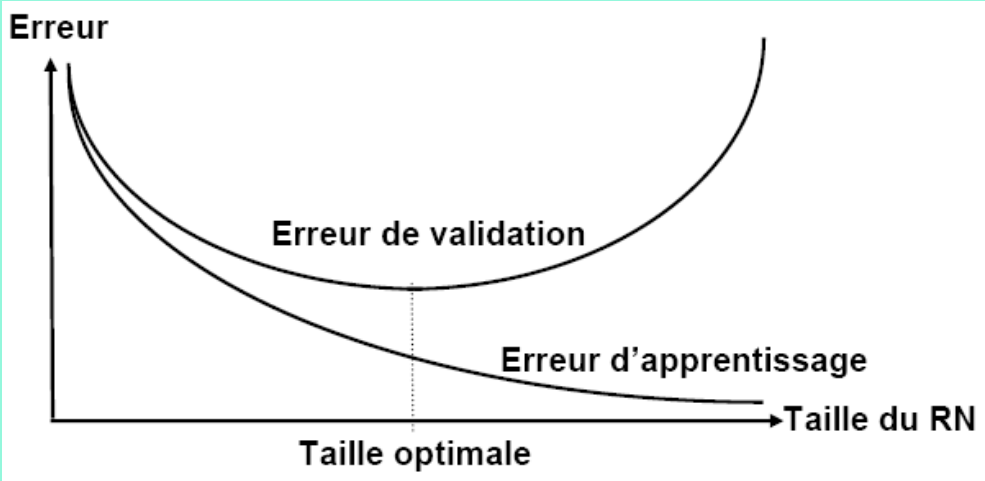
▪ Partie conception: Permettre de choisir la meilleure configuration possible du réseau de neurones.

▪ Partie de calcul numérique: Réaliser l'apprentissage d'un réseau de neurones.

Partie conception

- En vue de perfectionner le fonctionnement du MLP d'un côté et minimisé au maximum le temps de calcul d'autre part:
  - **Recherche d'une architecture optimale du réseau:** (Nombre de couche et Nombre de neurones par couche) effectuer des apprentissages pour des RN de différentes tailles et de sélectionner la configuration qui a une erreur la plus faible sur la base de validation.

Problème de sur-paramétrisation



  - **Détermination des entrées pertinentes:** Choisir les grandeurs qui ont une influence significative sur le phénomène que l'on cherche à modéliser.

Partie conception

- **Conditionnement:** Avant tout apprentissage, il est préférable de conditionner (normaliser) toutes les variables d'entrées et de sorties.

- **Initialisation des poids et biais:** Les poids et biais obtenus par le réseau à la fin de l'apprentissage dépendent en partie de ceux choisis au départ de l'apprentissage.

Apprentissage des réseaux MLP

- L'apprentissage d'un réseau de neurones est défini comme un problème d'optimisation qui consiste à trouver les coefficients du réseau minimisant une fonction d'erreur globale (fonction de coût).
- La fonction la plus couramment utilisée, est celle dite fonction d'erreur quadratique définie pour chaque exemple n ($n \in N$):

Où: N exemples d'apprentissage
$x(n)$ vecteur d'entrée
$y(n)$ vecteur de sortie
$d(n)$ valeurs désirées
r nombre de neurones de sortie.

$$e(n) = \frac{1}{2}\sum_{j=1}^{r}[d_j(n) - y_j(n)]^2$$

- Pour tout l'ensemble d'apprentissage N on peut définir la fonction de coût (appelée aussi l'erreur quadratique moyenne *EQM*):

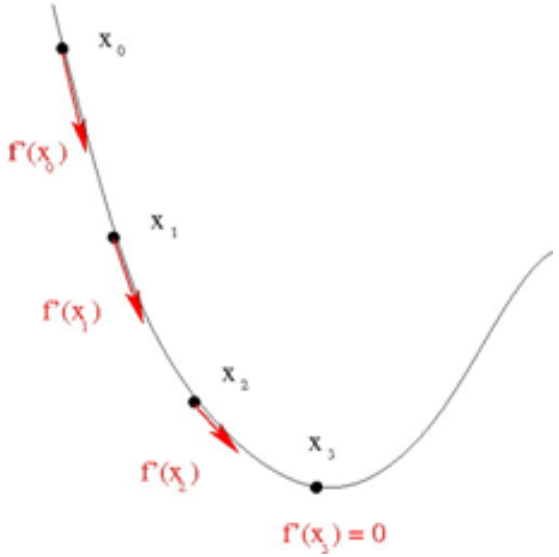
$$E(n) = \frac{1}{N}\sum_{n=1}^{N} e\ (n)$$

Algorithme de la rétropropagation

- Le principe de l'algorithme d'apprentissage (la rétropropagation) est de calculer la contribution des poids du réseau à l'erreur.
- Il consiste simplement en une descente de gradient, qui est une méthode d'optimisation universelle.
- On cherche à minimiser une fonction de coût (qui représente l'erreur entre la sortie désirée et la sortie obtenue), en suivant les lignes opposées de plus grande pente.

Mise en œuvre de l'algorithme

- Soit le couple ($x(n)$, $d(n)$) désignant la nième donnée d'apprentissage du réseau.



Où:

$$\vec{x}(n) = < x_1(n), \ldots\ldots, x_P(n) >$$

$$\vec{d}(n) = < d_1(n), \ldots\ldots, d_q(n) >$$

Correspondent respectivement aux « p » entrées et aux « q » sorties désirées du système.

Algorithme de la rétropropagation

- L'algorithme de la rétropropagation consiste alors à mesurer l'erreur entre les sorties désirée, et les sorties observées $y(n)$:

$$\vec{y}(n) =< y_1(n),....., y_q(n) >$$

- Et à rétropropager cette erreur à travers les couches du réseau en allant des sorties vers les entrées.

- L'algorithme de rétropropagation procède à l'adaptation des poids neurone par neurone en commencent par la couche de sortie:
  - Soit l'erreur observée $e_j(n)$ pour le neurone de sortie j et la donnée d'entraînement (apprentissage) n :

$$e_j(n) = d_j(n) - y_j(n)$$

L'indice j représente le neurone pour lequel on veut adapter les poids.

Algorithme de la rétropropagation

- L'objectif de l'algorithme est d'adapté les poids des connexions du réseau de manière à minimiser la somme des erreurs sur tous les neurones de sortie.
- Soit $E(n)$ la somme des erreurs quadratique observées sur l'ensemble des neurones de sorties :

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

- La sortie du neurones j est définie par:

$$y_j(n) = \sigma[\sum_{i=0}^{r} w_{ji}(n).y_i(n)]$$

Où :

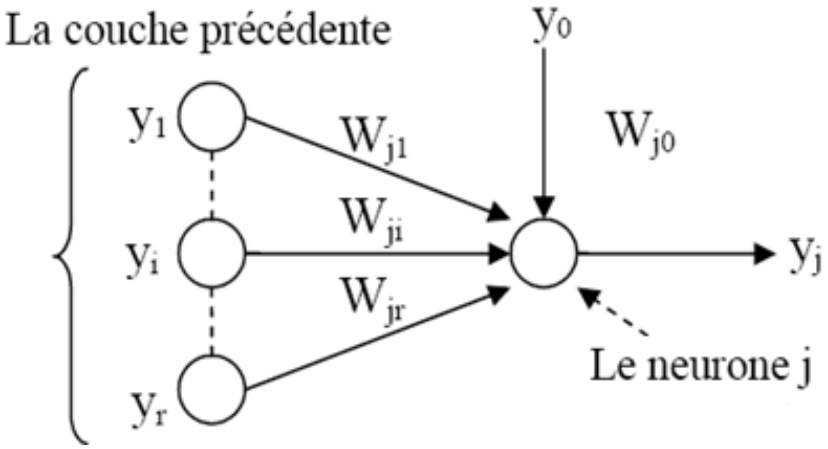
σ: fonction d'activation du neurone j,

w_{ji}: poids de connexion entre le neurone i de la couche précédente et le neurone j de la couche courante,

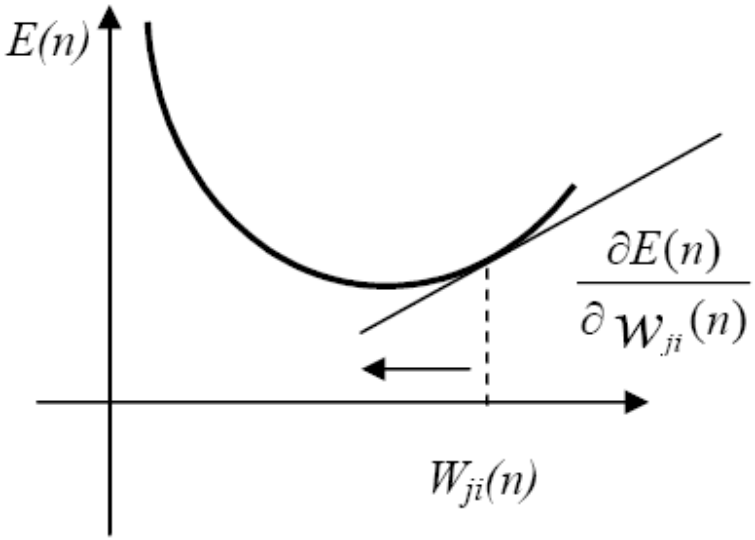
y_i : sortie du neurone i.

Algorithme de la rétropropagation

- On suppose ici que la couche précédente contient r neurones numérotés de 1 à r, le poids w_{j0} correspond au biais (seuil) du neurone j et que l'entrée $y_0(n) = 1$.



- Pour corriger l'erreur observée, il faut modifier le poids $w_{ji}(n)$ dans le sens opposé au gradient $\frac{\partial E(n)}{\partial w_{ji}(n)}$ de l'erreur.



Remarque: Cette dérivée partielle représente un facteur de sensibilité:
Si on change un peu $w_{ji}(n)$, $E(n)$ change beaucoup.

Algorithme de la rétropropagation

- Par l'application des règles des dérivées partielle on obtient :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Avec :

$$v_j(n) = \sum_{i=0}^{r} w_{ji}(n) . y_i(n)$$

Et on exprime la variation de poids $\Delta w_{ji}(n)$ sous la forme :

$$\Delta w_{ji}(n) = -\eta . \frac{\partial E(n)}{\partial w_{ji}(n)}$$

Avec : $0 \leq \eta \leq 1$ représentant un taux d'apprentissage ou gain de l'algorithme

Algorithme de la rétropropagation

Evaluons maintenant chacun des termes du gradient :

- $$\frac{\partial E(n)}{\partial e_j(n)} = \frac{1}{2} \cdot \frac{\partial e_j^2(n)}{\partial e_j(n)} = e_j(n)$$

- $$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial [d_j(n) - y_j(n)]}{\partial y_1(n)} = -1$$

- $$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial [\sigma(v_j(n))]}{\partial v_j(n)}$$

On suppose que σ est fonction sigmoïde, ce qui donne :

$$\sigma(v_j(n)) = \frac{1}{1 + \exp(-v_j(n))}$$

Algorithme de la rétropropagation

Alors

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\exp(-v_j(n))}{(1+\exp(-v_j(n)))^2}$$

Or

$$\frac{\partial y_j(n)}{\partial v_j(n)} = y_j(n).[1-y_j(n)]$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial \left[\sum_{i=0}^{r} w_{ji}(n).y_i(n)\right]}{\partial w_{ji}(n)} = y_i(n)$$

Nous obtenons donc :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\, y_j(n).[1-y_j(n)].y_i(n)$$

Algorithme de la rétropropagation

et la règle du 'delta' pour la couche de sortie s'exprime par :

$$\Delta w_{ji}(n) = -\eta . \frac{\partial E(n)}{\partial w_{ji}(n)} = \eta \delta_j(n) . y_i(n)$$

Avec :

$$\delta_j(n) = e_j(n) y_j(n).[1 - y_j(n)]$$

Considérons maintenant le cas des neurones sur la dernière couche caché (le cas des autre couches cachées est semblable).

- La variable *n* désignera toujours la donnée d'entraînement.
- Les indices *i* et *j* désignerons respectivement (comme précédemment) un neurone sur la couche précédente et un neurone sur la couche courante.
- L'indice *k* servira maintenant à désigner un neurone sur la couche suivante.

Algorithme de la rétropropagation

Reprenons l'expression de la dérivée partielle de l'erreur totale $E(n)$ par rapport à w_{ji} mais on ne dérivant pas par rapport à l'erreur $e_j(n)$, car celle-ci est inconnue dans le cas d'une couche cachée.

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

Par rapport aux résultats obtenus pour la couche de sortie, les deux derniers termes de cette équation restent inchangés, seul le premier terme sera évalué.

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\partial \left[\frac{1}{2}\sum_{k \in c} e_k^2(n)\right]}{\partial y_j(n)}$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_{k \in c} \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial y_j(n)}\right]$$

Algorithme de la rétropropagation

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_{k \in c} [e_k(n) . \frac{\partial e_k(n)}{\partial v_k(n)} . \frac{\partial v_k(n)}{\partial y_j(n)}]$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_{k \in c} [e_k(n) . \frac{\partial [d_k(n) - \sigma(v_k(n))]}{\partial v_k(n)} . \frac{\partial v_k(n)}{\partial y_j(n)}]$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_{k \in c} [e_k(n) . (-y_k(n) . [1 - y_k(n)]) . w_{kj}]$$

Ce qui donne :

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k \in c} [\delta_k(n) . w_{kj}(n)]$$

Algorithme de la rétropropagation

En substituant l'équation dans $\frac{\partial E(n)}{\partial w_{ji}(n)}$, on obtient :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -y_j(n)[1-y_j(n)]\sum_{k\in c}[\delta_k(n).w_{kj}(n)]\,y_i(n)$$

et :

$$\Delta w_{ji}(n) = -\eta.\frac{\partial E(n)}{\partial w_{ji}(n)} = \eta\,\delta_j(n).y_i(n)$$

avec :

$$\delta_j(n) = y_j(n).[1-y_j(n)].\sum_{k\in c}[\delta_k(n).w_{kj}(n)]$$

Les deux derniers équations ont démontré qu'ils sont valide pour toutes les couches cachées. Cependant, pour la première couche cachée du réseau, il faut substituer la variable $y_i(n)$ *par* l'entrée du réseau $x_i(n)$.

Mise en œuvre de l'algorithme

Comme résumer de la mise en œuvre de l'algorithme de rétropropagation standard:

- Phase 1: Initialisation de tous les poids à de petite valeurs aléatoire dans l'intervalle [-0.5,0.5];
- Phase 2: Pour chaque donnée d'entraînement n:
  - a. Calculer des sorties observées en propageant les entrées vers l'avant;
  - b. Ajuster les poids en rétropropageant l'erreur observée:

$$w_{ji}(n) = w_{ji}(n-1) + \Delta w_{ji}(n) = w_{ji}(n-1) + \eta\, \delta_j(n).y_i(n)$$

où le gradient local es défini par:

$$\begin{cases} \delta_j(n) = e_j(n)\, y_j(n).[1-y_j(n)] & \text{pour la couche de sortie} \\ \delta_j(n) = y_j(n).[1-y_j(n)].\sum_{k\in c}[\delta_k(n).w_{kj}(n)] & \text{pour une couche cachée} \end{cases}$$

Avec $0 \le \eta \le 1$ est le taux d'apprentissage

Mise en œuvre de l'algorithme

- le choix de η est empirique,
- si η est trop petit, le nombre d'itérations peut être très élevé,
- si η est trop grand, les valeurs de la suite risquent d'osciller autour du minimum sans converger.

- Phase 3: Répéter les étape 1 et 2 jusqu'à un nombre maximum d'itération ou jusqu'à ce que la valeur de l'erreur quadratique moyenne (EQM) soit inférieur à un certain seuil;

Remarque: le but d'atteindre EQM inférieur à un seuil n'est pas sûre, alors pour éviter le problème de la boucle ouverte, on fixe un nombre d'itérations maximum, généralement l'ordre des centaines, dans ce cas là l'algorithme cherche à minimiser EQM en Ni itérations successive tel que : Ni est inférieur au nombre d'itérations maximum.

Rétropropagation avec Momentum

Un taux d'apprentissage très petit assure la convergence mais très lentement, un facteur plus grand génère des changements plus importants dans les valeurs de poids (problème d'oscillation).
D.E Rumelhart a proposé une solution qui consiste à utiliser les changements des poids précédents pour la réadaptation des poids actuels.

L'équation d'adaptation devient donc :

$$\Delta w_{ji}(n) = \eta \delta_j y_i + \alpha_m \Delta w_{ji}(n-1)$$

$$w_{ji}(n+1) = w_{ji}(n) + \eta \delta_j y_i + \alpha_m \Delta w_{ji}(n-1)$$

Le terme $\alpha_m \Delta w_{ji}(n-1)$ est appelé Momentum

Le paramètre α_m est utilisé pour pondérer l'effet de ce terme. Sa valeur est généralement prise entre 0.1 et 0.9.

L'utilisation de cette méthode permet de faire sortir les poids des minimums locaux, afin de chercher d'autres optimums, ce qui donne beaucoup de chances d'aboutir à un minimum global.

Rétropropagation avec Momentum et taux d'apprentissage η adaptatif

Pour ne pas engendrer des oscillations et une instabilité dans la recherche du minimum, le mieux est de choisir un taux d'apprentissage η adaptatif.

Kung a proposé une méthode d'adaptation qui consiste à adapter le taux d'apprentissage et le momentum au même titre que les poids.

$$\eta_d = \eta.\lambda_a \quad (\lambda_a < 1)$$ Diminuer le taux

$$\eta_a = \eta.\lambda_a \quad (\lambda_a > 1)$$ Augmenter le taux

λ_a est un paramètre d'optimisation qui peut être supérieur à 1 en cas d'augmentation de taux d'apprentissage ou inférieur à 1 dans le cas contraire.

La même chose pour le momentum α_m, la règle d'adaptation de momentum est définit comme suit :

$$\alpha_i(n) = \begin{cases} (1-(R\%/100))\alpha_i(n-1) & \text{quand } E(n) > E(n-1) \\ (1+(R\%/100))\alpha_i(n-1) & \text{quand } E(n) \le E(n-1) \end{cases}$$

Avec:

R% est le pourcentage du taux de changement de α_i entre deux itérations successives

Rétropropagation avec l'algorithme quasi-Newtonien

La méthode quasi-Newtonienne est une alternative de la méthode de gradient. Elle est basée sur la méthode de Newton qui consiste à calculer les nouveaux poids de la couche L, en utilisant.

$$w_{ij}^{l}(t+1) = w_{ij}^{l}(t) - \mathrm{H}^{-1}(n)G(n)$$

où :

$H(n)$: est la matrice Hessienne

$G(n)$: est le gradient de la même fonction

La mise en œuvre de cette méthode est complexe (exige le calcul des deuxième dérivées).

Il y'a une classe d'algorithmes qui sont basés sur la méthode de Newton, mais qui n'exige pas le calcul des deuxième dérivées. Ceux-ci s'appellent les méthodes quasi-newtoniennes. Elles mettent à jour une matrice approximative de la matrice Hessienne à chaque itération de l'algorithme, en fonction du gradient.

- Méthode de Levenberg-Marquardt (LM)
- Méthode de Broydan-Fletcher-Goldfarb-Shanno (BFGS)

Neurones à Fonctions Radiales de Base (RBF)

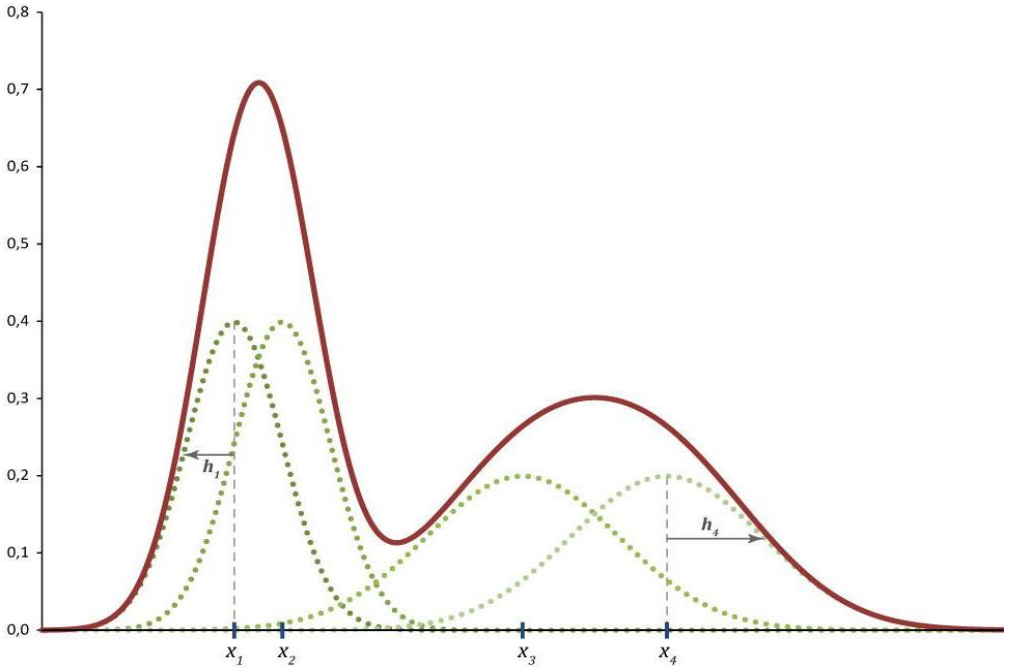
Formalisme

L'approximation de fonction non linéaire peut également s'effectuer à l'aide d'une somme de fonctions noyaux ("kernels"). Si ces noyaux sont fixés en largeur et position, la sortie dépend linéairement des poids. Pour une fonction continue ϕ d'une variable vectorielle Y, son estimation par une somme de N_c noyaux s'écrit:

$$\varphi(\underline{Y}) = w_0 + \sum_{i=1}^{N_c} w_i \, \Phi(\|\underline{Y} - \underline{c}_i\|)$$

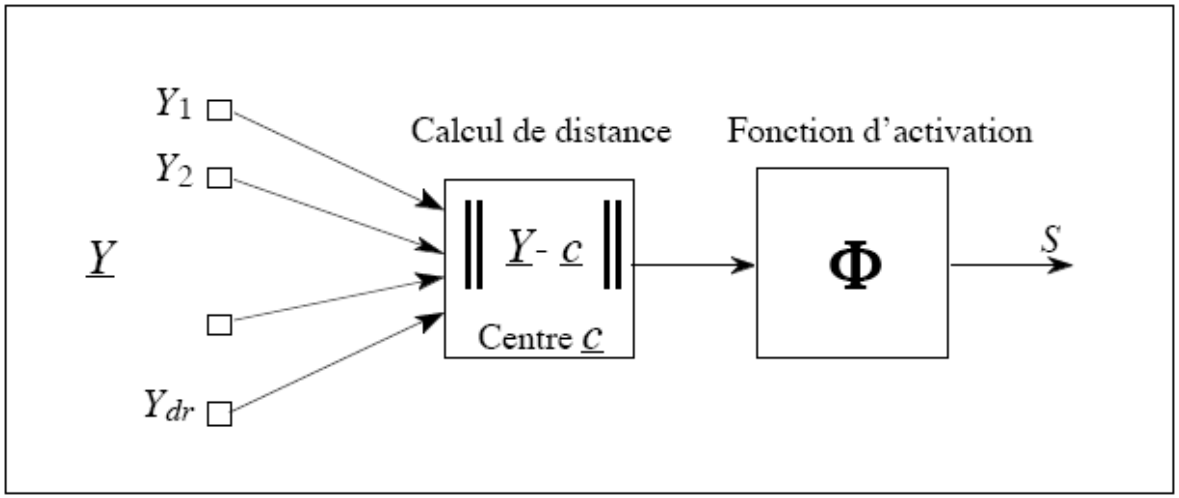
où Φ désigne la fonction noyau, w_i les coefficients de pondération et c_i le centre du ième noyau.

Cette formulation est la base des réseaux de neurones à fonctions radiales de base ou RBF (Radial Basis Function).



Neurones à Fonctions Radiales de Base (RBF)

- Dans le modèle de neurone à fonction radiale de base, chaque neurone élémentaire calcule la distance entre l'entrée et son centre qu'il fait passer ensuite dans une non linéarité Φ.
- Dans ce modèle, l'opérateur Distance (ou Norme) remplace l'opération Produit Scalaire du neurone formel.



La sortie S du neurone s'écrit finalement sous la forme:

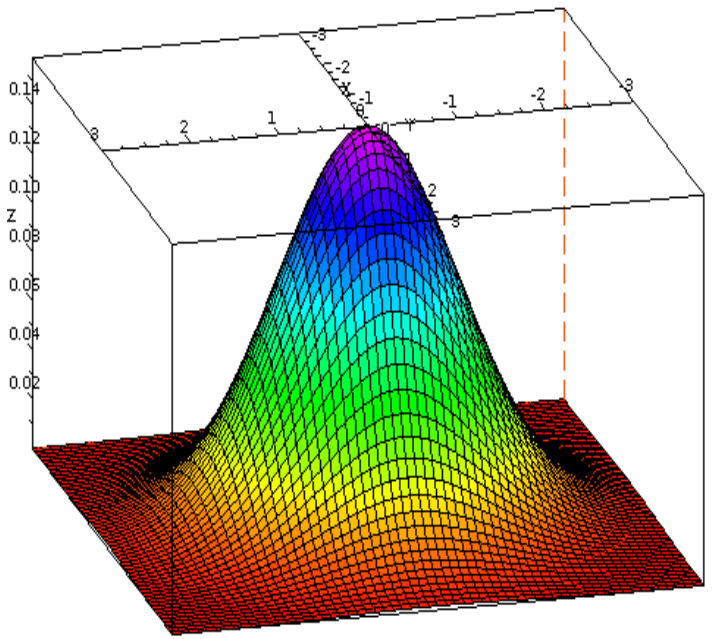
$$S = \Phi\left(\| \underline{Y} - \underline{c} \| \right)$$

Noyaux utilisés

Les noyaux utilisés comme fonction d'activation sont des fonctions définies de R vers R, symétriques radialement par rapport à un point (d'où la dénomination de neurones à fonctions radiales de base) parmi lesquelles on peut citer:

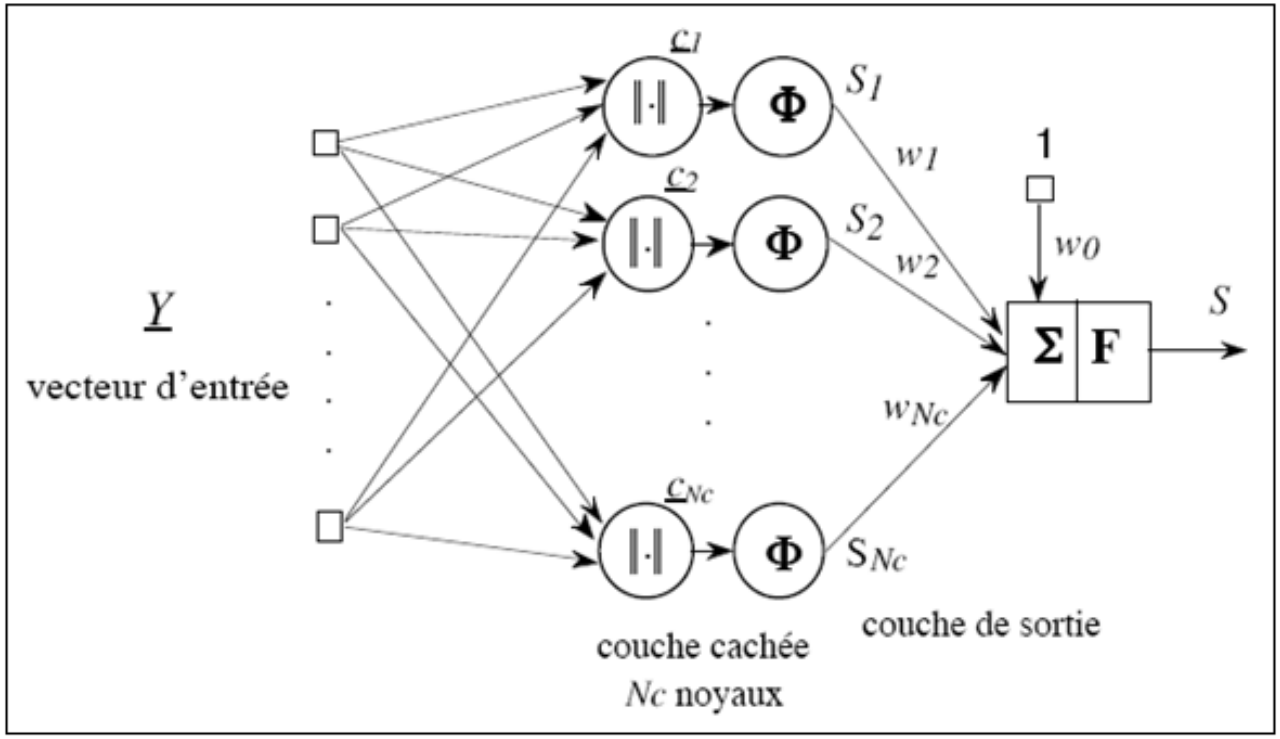
- ☑ Noyau Thin plate $\Phi(v) = v^2 \log(v)$
- ☑ Noyau Multiquadratique $\Phi(v) = \sqrt{(v^2 + \beta^2)}$
- ☑ Noyau Gaussien $\Phi(v) = \exp\left(-\frac{v^2}{2\beta^2}\right)$

Le noyau gaussien est le plus largement répandu. La valeur que prend sa sortie est d'autant plus importante que l'entrée est plus proche de son centre et elle tend vers zéro lorsque la distance entrée-centre devient importante. Le paramètre β permet de contrôler la vitesse de décroissance de la fonction Φ et il conviendra de le choisir de façon judicieuse.



Réseau de neurones RBF

L'architecture d'un réseau RBF s'organise en deux couches seulement : une couche cachée et une couche de sortie. La première couche, constituée de N_c noyaux élémentaires, effectue une transformation non linéaire de l'espace d'entrée. La couche de sortie est constituée d'un nombre de neurones de type produit scalaire à fonction d'activation linéaire.



Architecture d'un réseau de neurones RBF

Réseau de neurones RBF

La sortie d'un tel réseau s'exprime sous la forme :

$$S = \mathbf{F}\left(w_0 + \sum_{i=1}^{N_c} w_i \, \Phi(\|\underline{Y} - \underline{c}_i\|) \right)$$

Où la fonction d'activation en sortie est linéaire $\mathbf{F}(x) = x$.

Si les perceptrons multicouches MLP et les réseaux RBF sont tous deux en mesure d'approximer n'importe quelle fonction non linéaire, il convient de noter quelques différences entre ces deux types de réseaux:

- La première porte sur leur architecture (Le MLP peut comporter plusieurs couches cachées), tandis qu'elle est figée en deux couches pour les RBF.
- Le deuxième point fondamental qui les différencie est la nature des réponses qu'ils sont en mesure de fournir, gaussiennes pour les RBF et sigmoïdales pour les MLP.

Apprentissage des réseaux RBF

Les paramètres ajustables dans un réseau RBF sont :

1. Nombre de noyaux dans la couche cachée
2. La position des centres c_i
3. La valeur de l'écart type β_i associé à chaque noyau
4. Les poids de la couche de sortie w.

Différentes stratégies d'apprentissage sont alors possibles:

- Apprentissage global
- Apprentissage hybride

Apprentissage des réseaux RBF

Apprentissage global

Pour un nombre de noyaux fixé a priori, cette approche consiste à ajuster simultanément à l'aide d'un apprentissage supervisé la position des noyaux, l'écart-type relatif à chaque noyau et les poids en sortie.

Ces paramètres (c_i, β_i, w) peuvent être ajustés d'une manière itérative à l'aide d'un algorithme de gradient qui consiste à minimiser une fonction coût de type moindres carrés.

Ce type d'apprentissage peut présenter des difficultés de convergence.

Apprentissage des réseaux RBF

Apprentissage hybride

Ce dernier possède plusieurs variantes:

1- Soit on commence par le positionnement des noyaux et le choix de leur nombre et on détermine ensuite l'écart-type de chaque noyau et les poids par un algorithme de type gradient.

2- Soit l'apprentissage des deux couches s'effectue séparément:

- Premièrement: on optimise les paramètres de la couche cachée (position des noyaux, nombre des noyaux et écart-type)
- Deuxièmement: on calcul des poids de la couche de sortie.

La majorité des auteurs proposent d'adopter la procédure d'apprentissage hybride.

Apprentissage Hybride

Choix du nombre et du positionnement des noyaux

Le choix qui consiste à centrer un noyau sur chaque exemple de la base d'apprentissage est peu réaliste et conduit rapidement à une explosion de la taille du réseau si le nombre d'exemples d'apprentissage est important.

- ***Algorithme des "k-means" (k-moyens)***

Le but est de regrouper les N exemples de la base d'apprentissage en N_c groupes (ou clusters) de sorte que tout exemple soit plus proche des exemples appartenant à son groupe qu'à ceux appartenant aux autres groupes.

A chaque groupe de la partition obtenue correspondra un noyau élémentaire du réseau RBF dont le centre sera le centre de gravité du groupe.

Cet algorithme suppose néanmoins que l'on a fixé a priori le nombre de noyaux à atteindre.

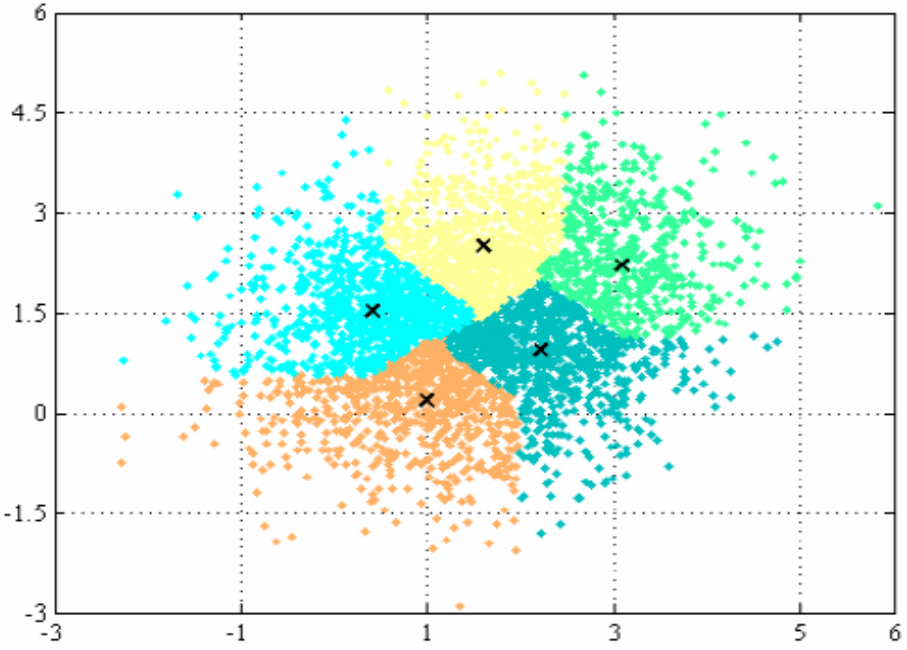
Apprentissage Hybride

Méthode d'orthogonalisation

Cette méthode va permettre la sélection des N_c centres parmi les N exemples d'apprentissage. Elle consiste à classé chaque exemple parmi la base complète en terme de contribution à la sortie souhaitée.

La méthode d'orthogonalisation offre une alternative intéressante pour construire un réseau RBF souple.

Cette méthode permet à la fois de positionner les noyaux et d'en régler le nombre.



Exemple de clustering réalisé par un k-means

Apprentissage Hybride

Choix de la largeur des noyaux

Dans un réseau RBF, le choix de la valeur de l'écart-type β_i ce fait avec précaution. Il existe trois cas les plus utilisés:

- Cas 1: L'écart-type sera constant pour tous les noyaux et déduit de la valeur maximale des distances séparant tous les noyaux par la relation:

$$\beta_i = \beta = \frac{d_{\max}}{\sqrt{2N_c}} \qquad \forall\, i \quad 1 \le i \le N_c$$

- Cas 2: la valeur de l'écart-type est directement la valeur moyenne des distances séparant tous les noyaux:

$$\beta_i = \beta = d_{moy} \qquad \forall\, i \quad 1 \le i \le N_c$$

- Cas 3: Chaque noyau possède son propre écart-type. Cette approche est particulièrement intéressante si une méthode de type "cluster" est employée:

$$\beta_i = \frac{1}{N_{CL_i}} \sum_{\underline{Y} \in cluster\, i} \left(\underline{Y} - \underline{c}_i\right)^t \left(\underline{Y} - \underline{c}_i\right)$$

Où N_{CLi} est le nombre d'exemples dans le *ième* groupe de centre c_i.