Réseaux de neurones artificiels : architectures et applications

Patrice Wira

patrice.wira@uha.fr
www.trop.mips.uha.fr

Université de Haute Alsace
Laboratoire MIPS (Modélisation, Intelligence, Processus, Systèmes)

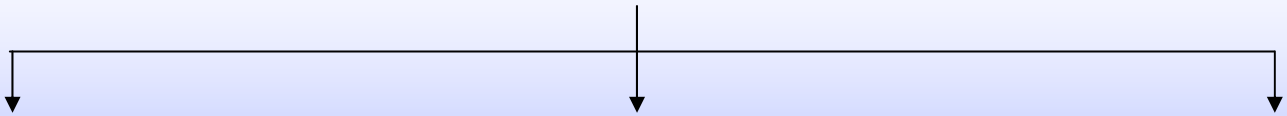
Avril 2009

Université de Haute Alsace

- *à Mulhouse et Colmar (France)*
- *8000 étudiants*
- *impliquée dans 3 pôles de compétitivités à vocation internationale (Fibre Grand Est, Innovation thérapeutique, Véhicule du Futur)*

Laboratoire MIPS
(Modélisation, Intelligence, Processus, Systèmes)

- *thématique principale : véhicule intelligent*
- *100 personnes (75 permanents + 25 doctorants)*
- *3 axes de recherche*



axe automatique	axe signal et image	axe informatique et réseau
MIAM : applications au véhicule TROP : contrôle neuromimétique	LABEL : traitement des images FOTI : traitement optique des informations	GRT : réseaux mobiles et qualité de services LSI : modèle objet et pattern des lois de commande

Master 1 et 2 AII
Automatique et Informatique Industrielle
options Recherche ou Professionnel

http://www.trop.mips.uha.fr/master/

Master recherche Sciences et Technologies
mention information systèmes communication
spécialité information systèmes communications
Référence : 17315-20098

http://www.campusfrance.org/
(portail obligatoire pour accéder à l'enseignement supérieur français)

Réseaux de neurones artificiels : architectures et applications

– Sommaire –

Réseaux de neurones artificiels : architectures et applications

Introduction

Les origines biologiques des réseaux de neurones

Les neurones sont considérés comme le support physique de l'intelligence. Ils fascinent puisque comprendre et savoir utiliser l'intelligence permet de réaliser des buts inimaginables.

Depuis quelques années, on cherche à copier les réseaux de neurones pour faire des lois de commande intelligentes.

Ce cours introduit l'approche « *traitement du signal* » des

réseaux de neurones artificiels

ou

réseaux neuromimétiques.

Eléments historiques

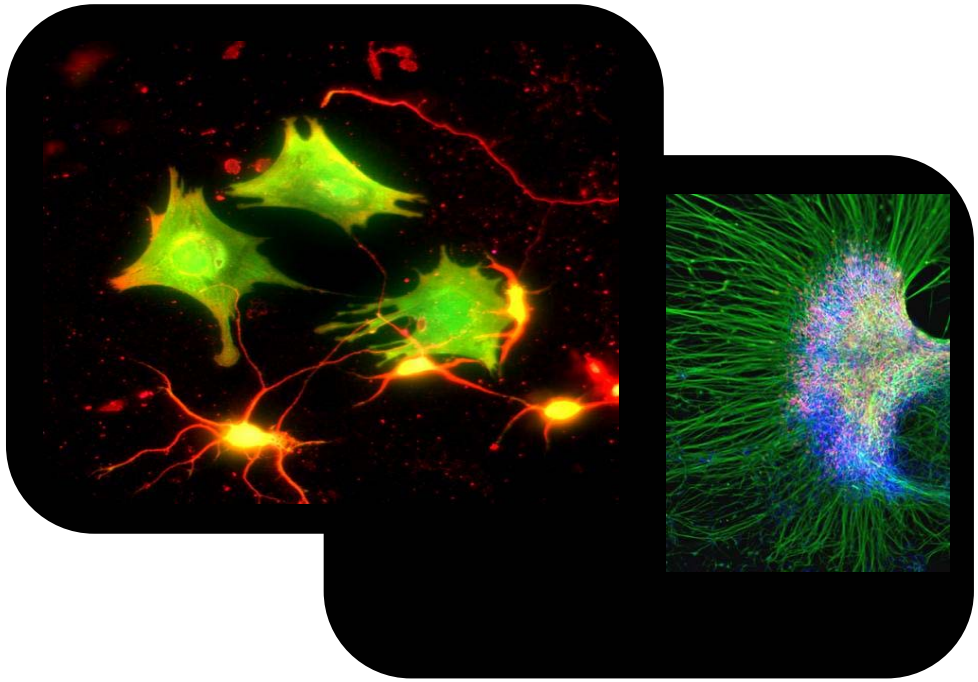
Dès 1943, Mac Culloch et Pitts ont proposé des neurones formels mimant les neurones biologiques et capables de mémoriser des fonctions booléennes simples.

Les réseaux de neurones artificiels réalisé à partir de ce types de neurones sont ainsi inspirés du systèmes nerveux. Ils sont conçus pour reproduire certaines caractéristiques des mémoires biologiques par le fait qu'ils sont:

- massivement parallèles ;
- capables d'apprendre ;
- capables de mémoriser l'information dans les connexions entre les neurones ;
- capables de traiter des informations incomplètes.

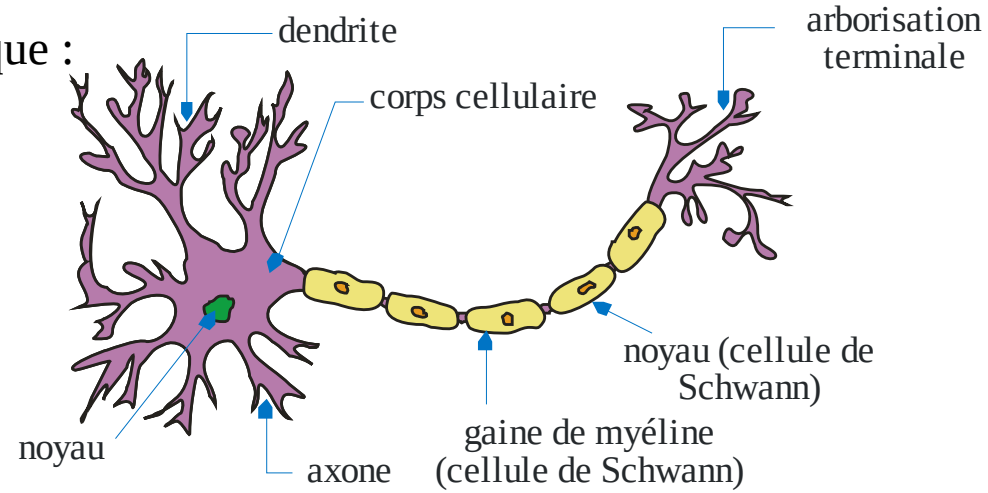
Les neurones biologiques

vue de plusieurs neurones biologiques :

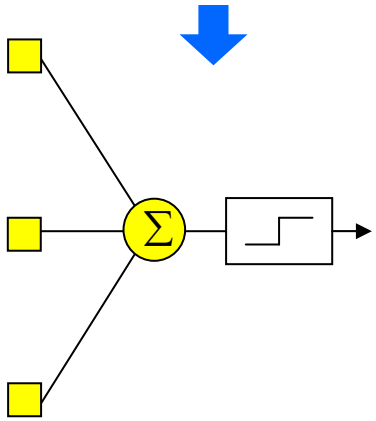


Du neurone biologique au neurone formel

simplification d'un neurone biologique :



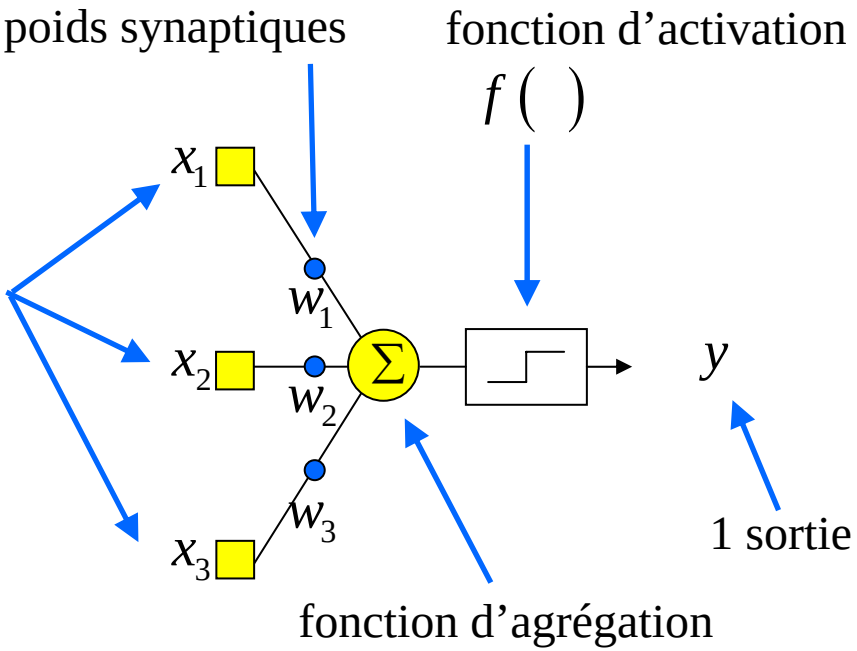
un neurone formel :



Architecture d'un neurone formel

Le neurone formel : exemple avec 3 entrées

Un neurone est une cellule qui "concentre" des signaux, qui les traite et qui les traduit en une réponse.



Couplage synaptiques et apprentissage

En 1949, Hebb a mis en évidence l'importance du couplage synaptique dans l'apprentissage par renforcement ou dégénérescence des liaisons inter-neuronales lors de l'interaction du cerveau avec le milieu extérieur.

Le premier modèle opérationnel est le Perceptron inspiré du modèle visuel et capable d'apprentissage. Il a été proposé en 1958 par Rosenblatt.

Les limites du Perceptron monocouche du point de vue performance ont été montrées en 1969 par les mathématiciens Minsky et Papert.

Les travaux de Hopfield en 1982 ont montrés que des réseaux de neurones artificiels étaient capables de résoudre des problèmes d'optimisation et ceux de Kohonen (1982) ont montrés qu'ils étaient capables des résoudre des tâches de classification et de reconnaissance.

Exemples d'application

Aujourd'hui, les réseaux de neurones artificiels ont de nombreuses applications dans des secteurs très variés :

• traitement d'images : reconnaissance de caractères et de signatures, compression d'images, reconnaissance de forme, cryptage, classification, etc.

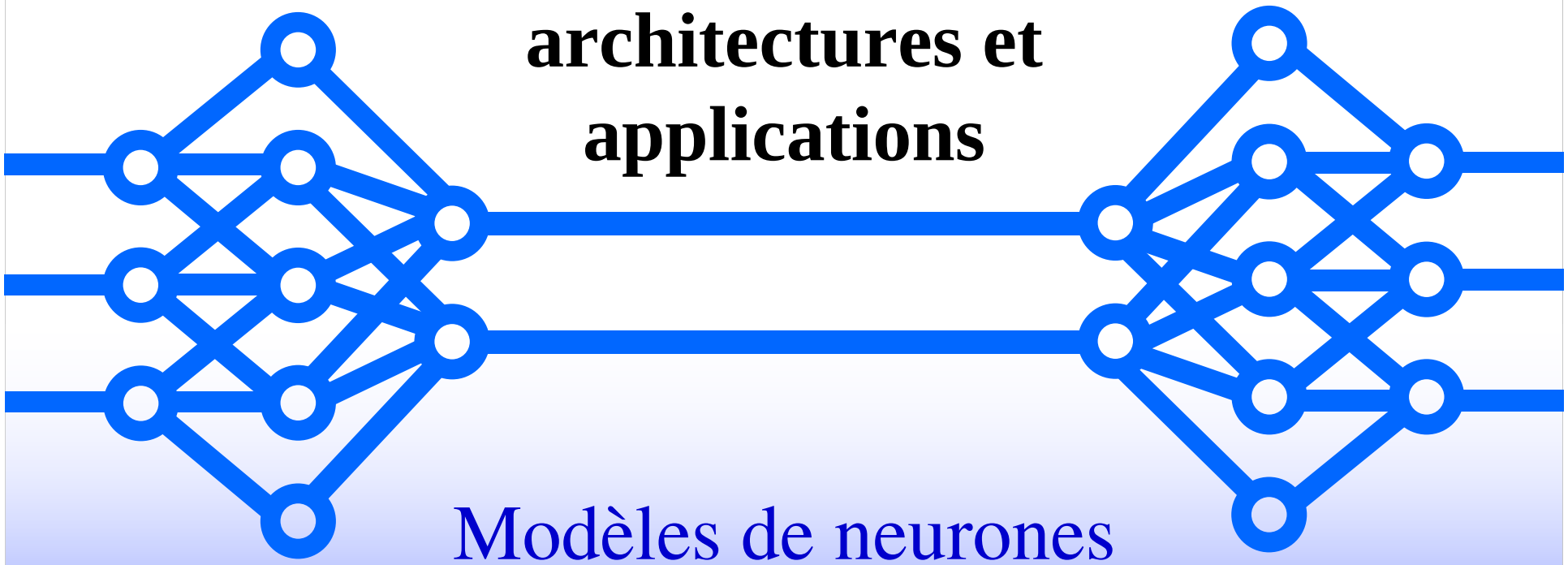
• traitement du signal : filtrage, classification, identification de source, traitement de la parole…

• contrôle : commande de processus, diagnostic, contrôle qualité, asservissement de robots…

• optimisation : planification, allocation de ressource, gestion et finances, etc.

• simulation : simulation de boîte noire, prévision météorologique, recopie de modèle…

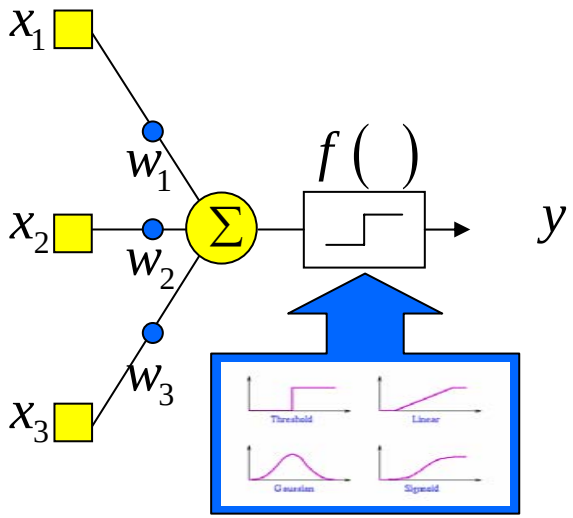
Réseaux de neurones artificiels : architectures et applications



Modèles de neurones

Le neurone formel : architecture et calculs

Le neurone formel : exemple avec 3 entrées



Calcul de la sortie du neurone :

$$y = f\left(w_0 + x_1 w_1 + x_2 w_2 + \ldots + x_n w_n\right) = f\left(w_0 + \mathbf{w}^T \mathbf{x}\right)$$

Correction des poids synaptiques :

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta \mathbf{w}_k$$

L'Adaline et le Perceptron

On distingue 2 types de neurone formel :

1. Le Perceptron de F. Rosenblatt (1958)
   + la règle de D. Hebb (1949)

2. L'Adaline
   (ADApitve LINear Element)
   de B. Widrow et T. Hoff (1960)

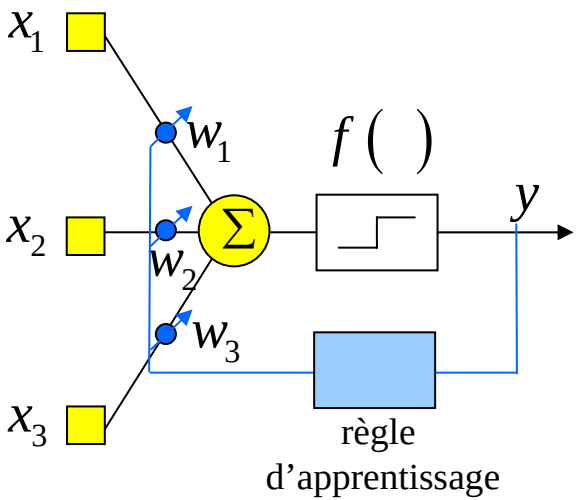
Le Perceptron

Le Perceptron de F. Rosenblatt (1958) et la règle de D. Hebb (1949)

A chaque itération :

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta\mathbf{w}_k$$

$$\Delta\mathbf{w}_k = \eta y_k \mathbf{x}_k$$



η représente le coefficient d'apprentissage
(sa valeur est généralement comprise entre 0 et 1)

Le Perceptron

Le Perceptron

Pour traiter plusieurs sorties, on utilise plusieurs Perceptron en parallèle, on parle alors de réseau monocouche.

L'utilisation d'une fonction d'activation du type « automate linéaire à seuil » permet de réaliser pour chaque neurone i une partition des vecteurs d'entrées en deux classes dont la frontière est définie par l'hyperplan de dimension n-1 (n = nombre d'entrées) et d'équation :

$$\sum_{j=1}^{n} w_{ij} x_j + w_{i0} = 0$$

Le Perceptron permet donc de ne séparer que des exemples linéairement séparables. Il ne peut pas par exemple réaliser un OU exclusif (ou XOR). Les réseaux multicouches pallient cette limitation.

L'Adaline

L'Adaline de B. Widrow et T. Hoff (1960)
(ADAptive LINear Element, la fonction d'activation est unitaire)

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta\mathbf{w}_k$$

$$\varepsilon_k = d_k - y_k = d_k - \mathbf{w}_k^T \mathbf{x}_k$$

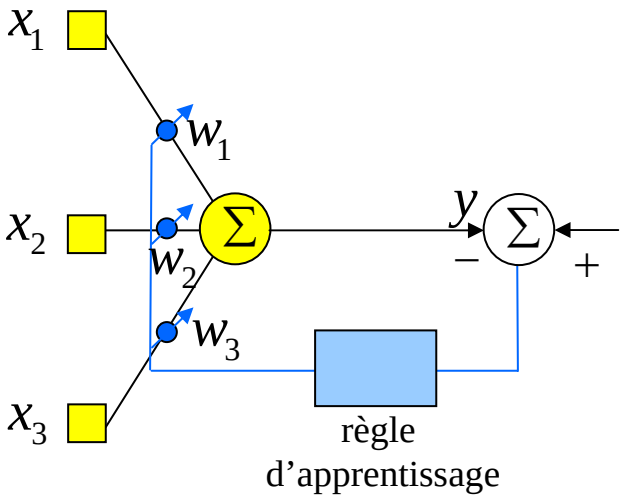
On peut utiliser 2 règles :

µ LMS

$$\Delta\mathbf{w}_k = \mu\varepsilon_k \mathbf{x}_k \alpha$$

α LMS

$$\Delta\mathbf{w}_k = \alpha \frac{\varepsilon_k \mathbf{x}_k}{\lambda + \|\mathbf{x}_k\|^2}$$



L'Adaline

L'Adaline

Pour traiter plusieurs sorties, on utilise plusieurs Adaline en parallèle, on parle alors de MAdaline (Muti-Adaline).

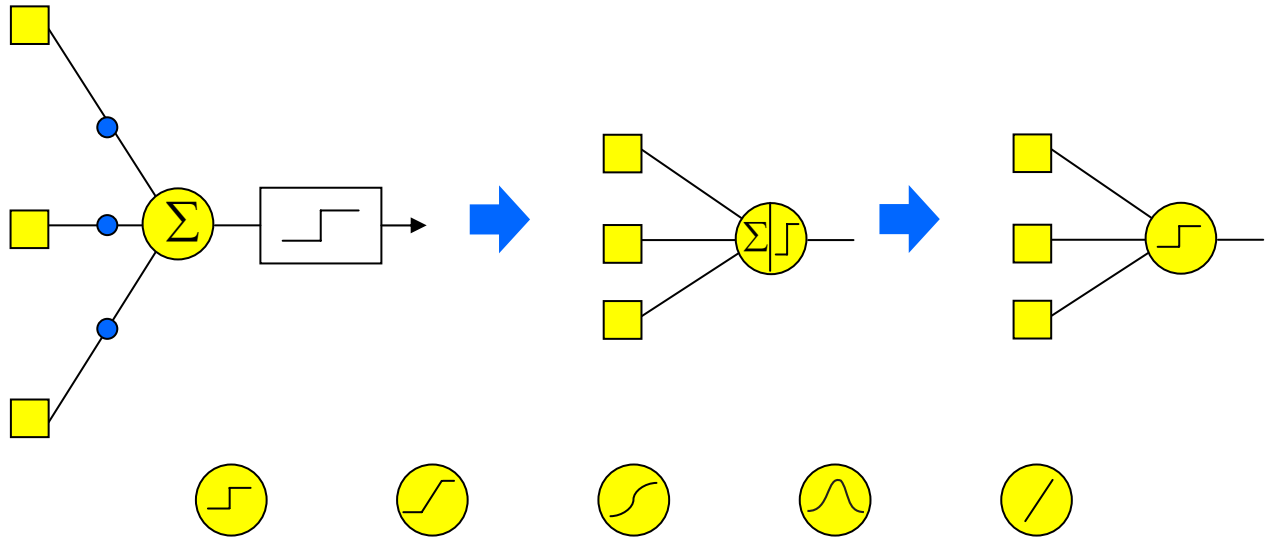
L'apprentissage d'un Adaline s'effectue en ligne et est basé sur la minimisation d'une erreur, l'erreur quadratique moyenne.

Lorsque le nombre d'entrées (exemples) tend vers l'infini, l'Adaline tend vers la solution de Gauss ou le filtre de Wiener. Les poids de l'Adaline ont alors convergé vers les coefficients optimaux (au sens des moindres carrés) et sont équivalents aux coefficients du filtre de Wiener.

Neurones formels utilisés

Que ce soit le Perceptron ou l'Adaline, ces 2 modèles sont toujours actuellement utilisés. Ils sont la base de tous réseaux de neurones artificiels dans le domaine de l'intelligence artificielle en particulier dans le traitement du signal.

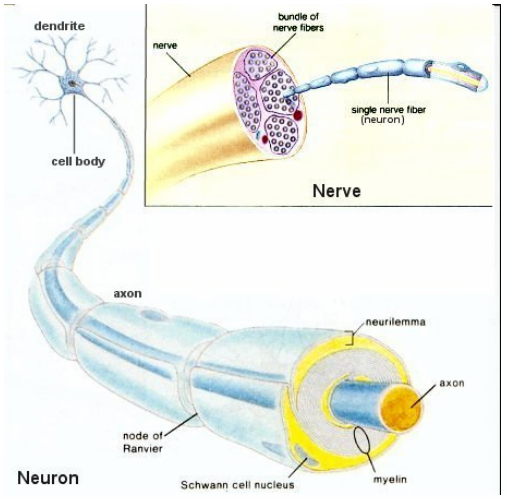
La représentation d'un neurone formel peut se simplifier :

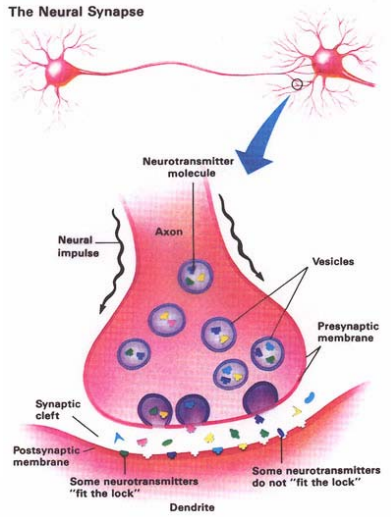


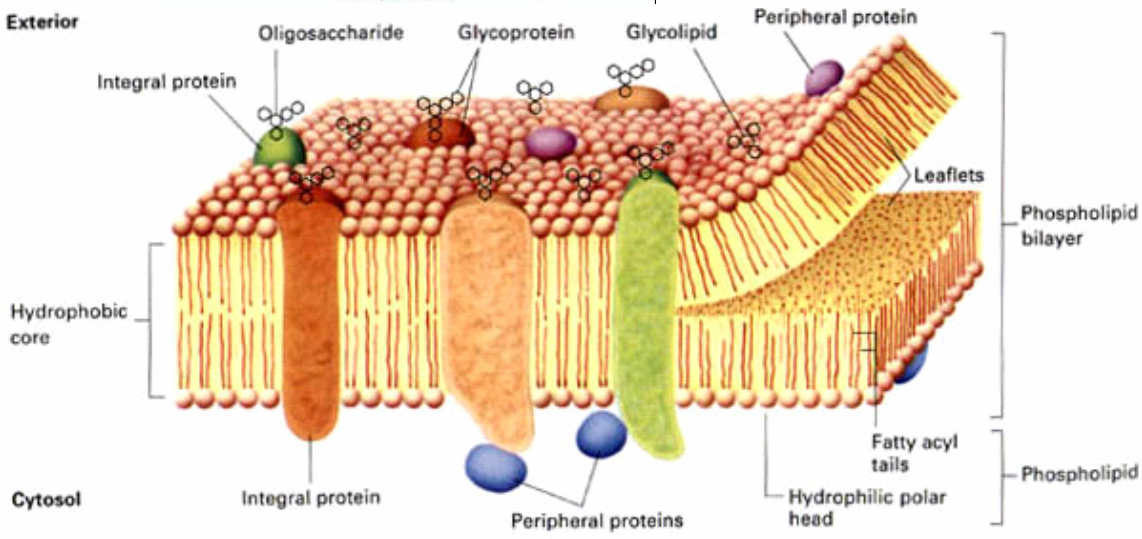
Autres neurones formels

Il existe d'autres modèles de neurone formel.

Leur utilisation dépend des objectifs…







Des neurones pour différents objectifs

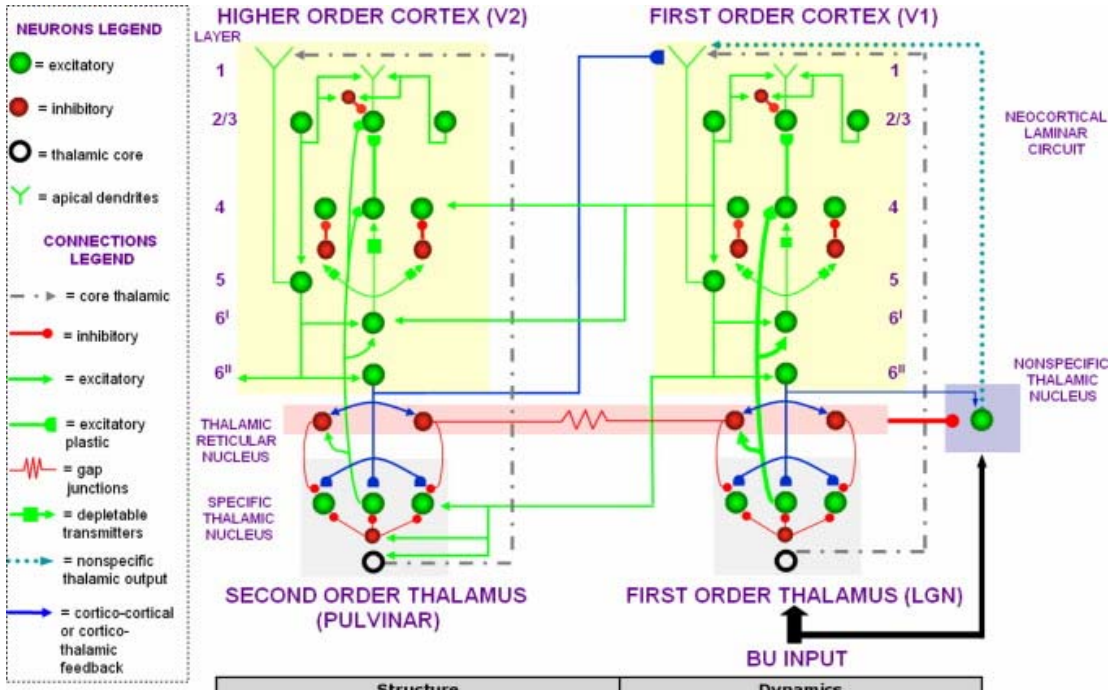
Les objectifs sont ceux des sciences différentes regroupées autours de ce qu'on appelle les neurosciences : neurologie, neurophysiologie, neuro-biologie, sciences cognitives, etc.

Elles cherchent à étudier :

- le neurone biologique,
- le cerveau, les aires de la motricité et des activités cérébrales,
- la perception,
- la formation de la pensée,
- la conscience,
- etc.

Des neurones pour différents objectifs

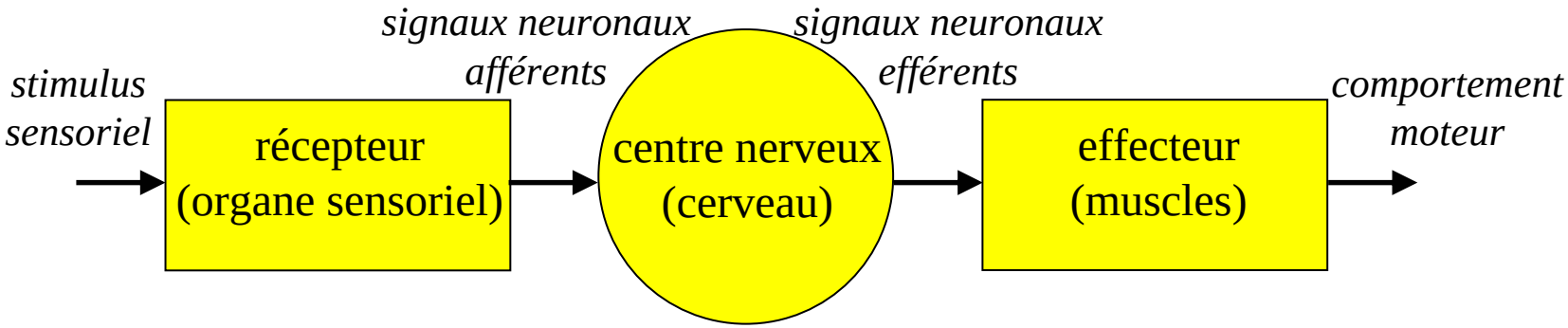
Étude de la conscience avec le modèle de Grossberg (2007) :

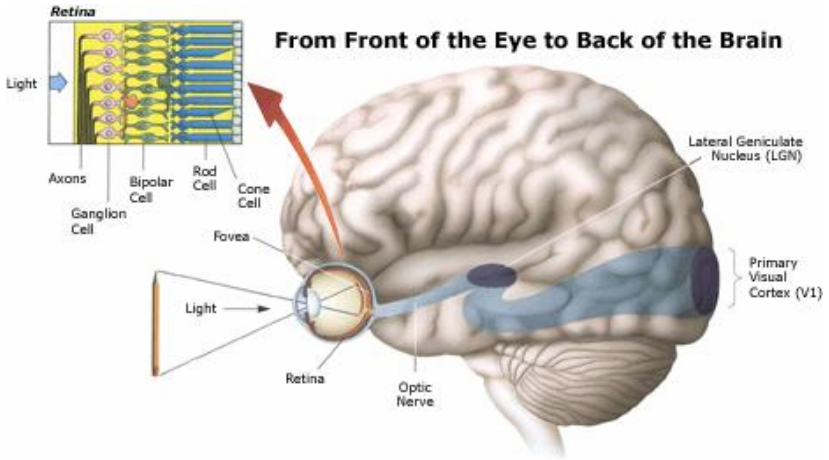


Structure	Dynamics
- Single channels - Single cells - Cell populations - Thalamic specific and nonspecific nuclei - Cortical layers - Thalamo-cortical and cortico-cortical interactions	- Synaptic STDP - Single channel neuromodulation (Ach/AHP) - Dendritic potentials and spikes - Somatic action potentials - Local field potentials/current source densities - Gamma and beta oscillations - Short-and long-range synchrony

Des neurones pour différents objectifs

Étude de la perception :

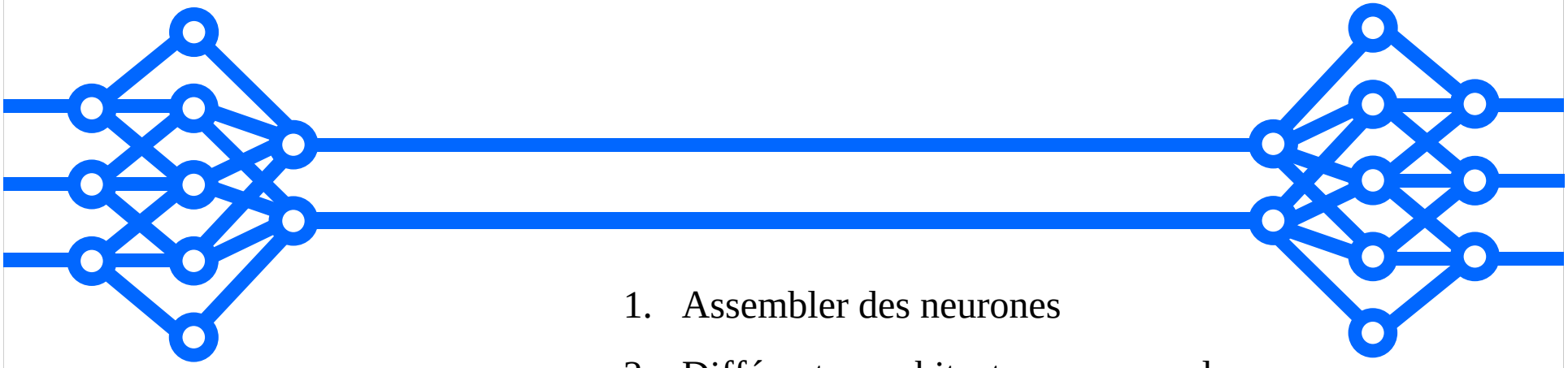




Réseaux de neurones artificiels : architectures et applications

Structures des réseaux de neurones artificiels

Structures des réseaux de neurones artificiels

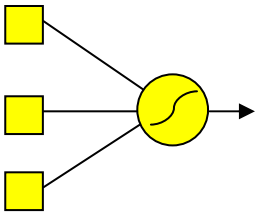


1. Assembler des neurones
2. Différentes architectures neuronales
3. Structure du réseau multicouche
4. Stratégies d'apprentissage
5. Apprentissage supervisé/non supervisé
6. Exemple : apprendre un processus
7. Apprentissage d'un réseau multicouche
8. Apprentissage d'une carte auto-organisatrice

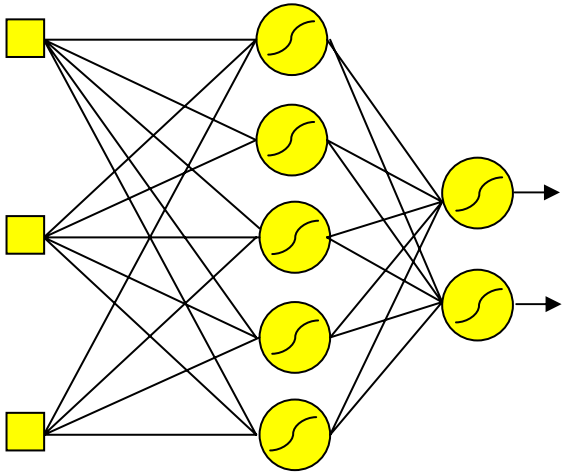
Assembler des neurones

Objectif : créer des méthodes adaptatives d'identification et des lois de commandes intelligentes, basées sur le principe de l'apprentissage, donc sur des réseaux de neurones artificiels.

Assembler des neurones c'est multiplier leurs capacités d'apprentissage.



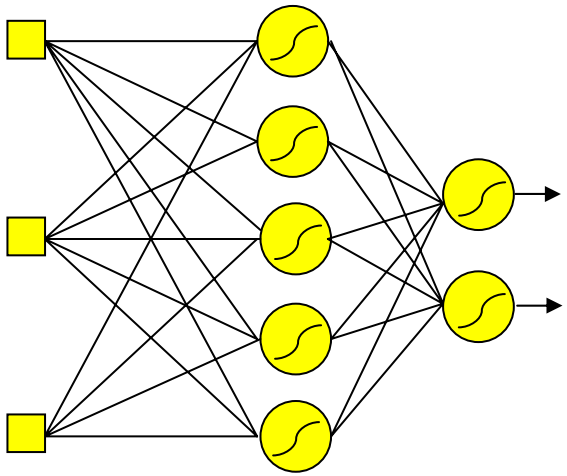
neurone formel de base



réseau de neurones artificiels

Assembler des neurones

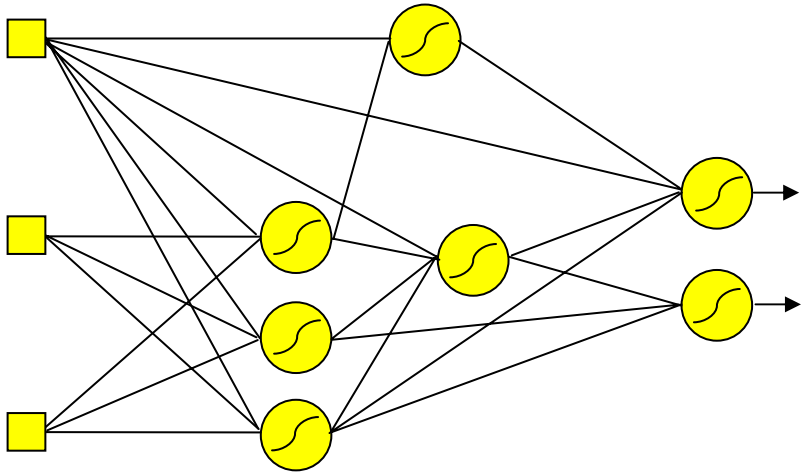
Exemple d'un réseau multicouche complètement connecté



Le réseau suivant est constitué de 5+2 neurones (3 entrées). Ce réseau est bien ordonné. On y distingue des couches et les neurones sont entièrement connectés entre les couches.

Assembler des neurones

Exemple d'un réseau non ordonné et partiellement connecté



Il est relativement aisé de mettre des neurones ensemble, mais il est plus difficile de les faire apprendre. Les architectures généralisées ou ordonnées sont plus simples à « manipuler ».

Différentes architectures neuronales

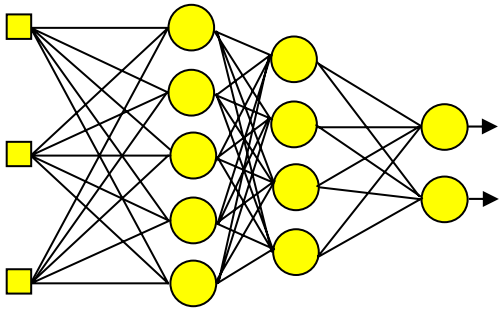
L'architectures d'un réseau de neurones est l'organisation des neurones entre eux au sein d'un même réseau. Autrement dit, il s'agit de la façon dont ils ordonnés et connectés.

La majorité des réseaux de neurones utilise le même type de neurones. Quelques architectures plus rares se basent sur des neurones dédiés. L'architecture d'un réseau de neurones dépend de la tâche à apprendre (problème à résoudre).

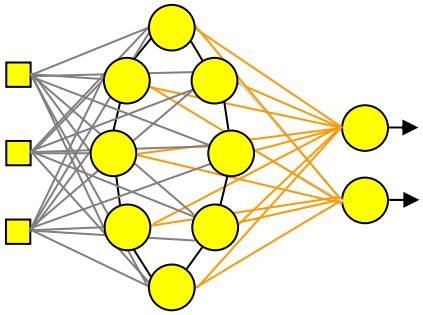
Un réseau de neurone est en général composé de plusieurs couches de neurones, des entrées jusqu'aux sorties.

Différentes architectures neuronales

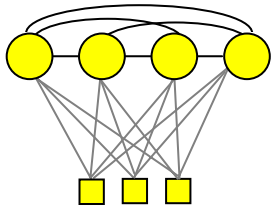
Quelques architectures :



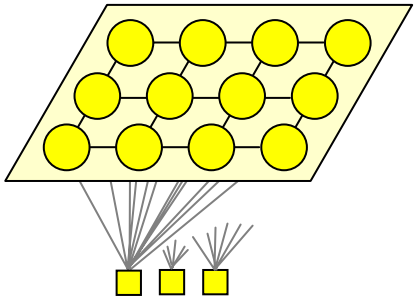
multilayer feedforward network



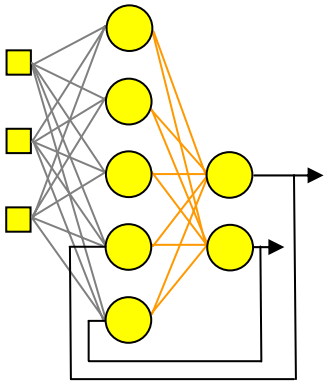
fully recurrent network



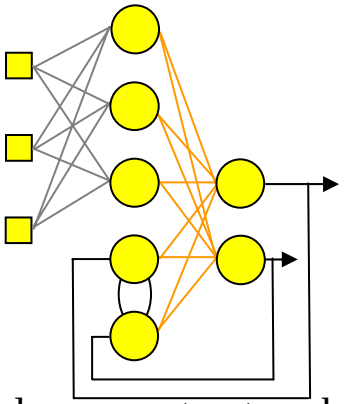
competitive network



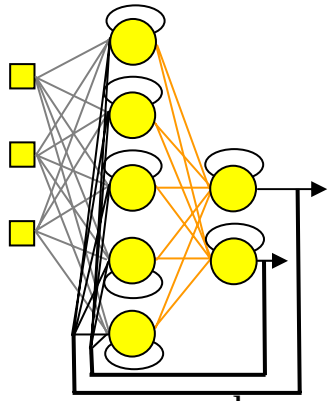
self-organizing map



Jordan network



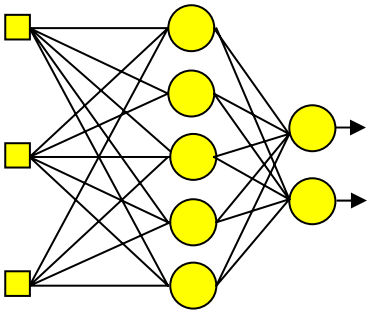
partial recurrent network with dual connections

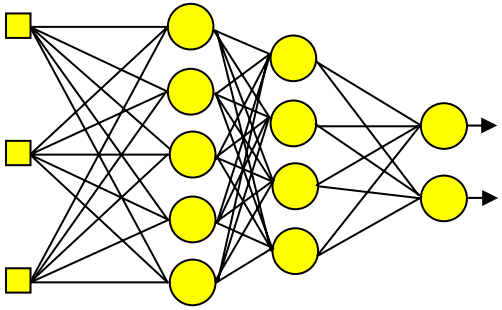


recurrent network with self connections

Structure du réseau multicouche

Le réseau de neurone multicouche : 2 exemples
(multilayer feedforward network, ou MultiLayer Perceptron = MLP)





Ce réseau a :
1 couche de 3 entrées
1 couche cachée (5 neurones)
1 couche de sortie (2 neurones)

Ce réseau a :
1 couche de 3 entrées
2 couches cachées
(de 5 et 4 neurones)
1 couche de sortie (2 neurones)

Structure du réseau multicouche

Un réseau de neurone multicouche est composé d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente. Chaque couche (i) est composée de Ni neurones, prenant leurs entrées sur les Ni-1 neurones de la couche précédente. À chaque synapse est associée un poids synaptique, de sorte que les Ni-1 sont multipliés par ce poids, puis additionnés par les neurones de niveau i, ce qui est équivalent à multiplier le vecteur d'entrée par une matrice de transformation.

Mettre différentes couches l'une derrière l'autre reviendrait à mettre en cascade plusieurs matrices de transformation et équivaudrait à une seule matrice, produit des autres, s'il n'y avait à chaque couche, la fonction d'activation qui introduit une non linéarité à chaque étape.

Un réseau de neurones dont les sorties seraient linéaires n'aurait que peu d'intérêt.

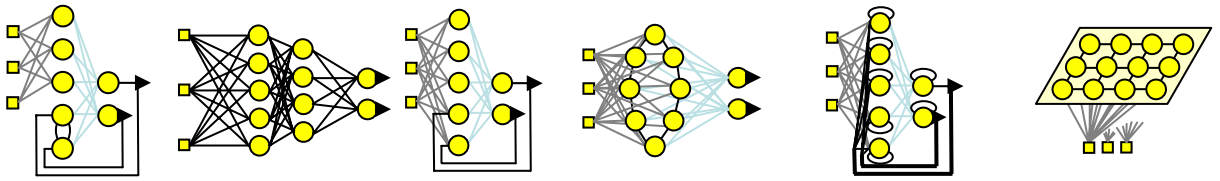
Stratégies d'apprentissage

L'apprentissage au sein des différentes architectures dépend de l'architecture du réseau et de l'environnement du problème.

Les 2 règles d'apprentissage pour mettre corriger les poids d'un neurone (règle de Hebb et de Widrow) ne concernent qu'un neurone seul.

Ces règles peuvent servir pour mettre à jour les poids d'un neurone, de certains réseaux de neurones, mais ne peuvent être généralisées et s'appliquer à n'importe quelle architecture.

Chaque architecture possède ses spécificités et nécessite une règle d'adaptation des poids qui lui est propre.



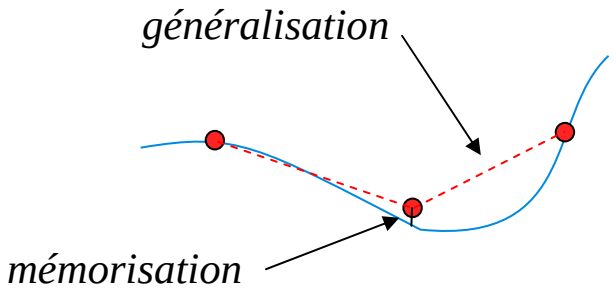
Stratégies d'apprentissage

L'apprentissage n'est pas modélisable dans le cadre de la logique déductive : celle-ci en effet procède à partir de connaissances déjà établies dont on tire des connaissances dérivées. Or il s'agit ici de la démarche inverse : par observations limitées tirer des généralisations plausibles.

La notion d'apprentissage recouvre deux réalités :

• **La mémorisation** : le fait d'assimiler sous une forme dense des exemples éventuellement nombreux,

• **La généralisation** : le fait d'être capable, grâce aux exemples appris, de traiter des exemples distincts, encore non rencontrés, mais similaires.



Stratégies d'apprentissage

Le concept de la **mémorisation** et celui de la **généralisation** sont partiellement en opposition. Si on privilégie l'un, on élaborera un système qui ne traitera pas forcément de façon très efficace l'autre.

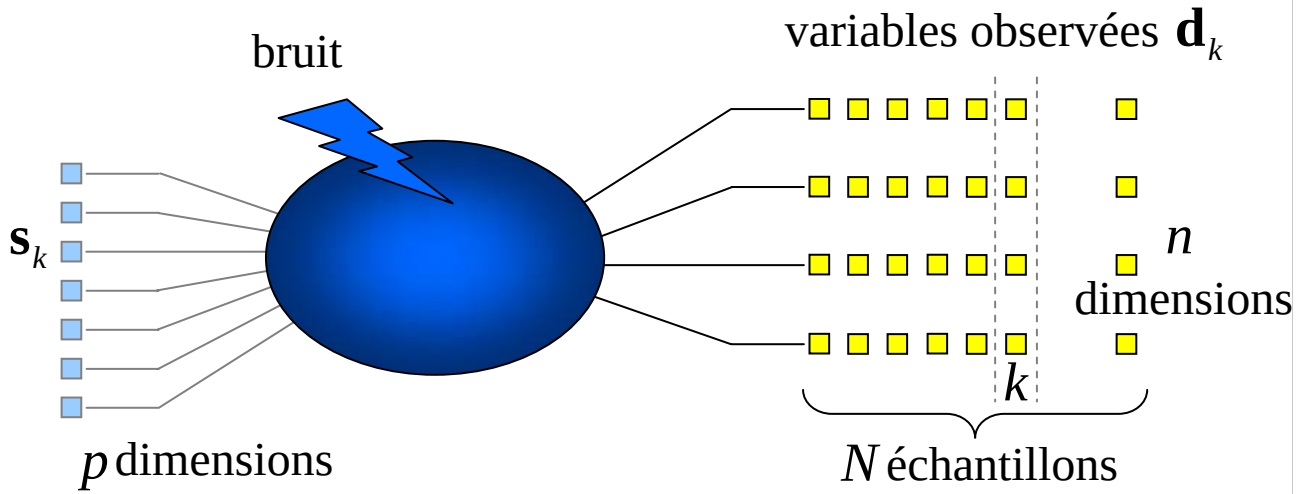
Dans le cas des systèmes d'apprentissage statistique, utilisés pour optimiser les modèles statistiques classiques, réseaux de neurones et automates markoviens, c'est la généralisation qui est préférée.

Il faut trouver un compromis en choisissant un coefficient d'apprentissage satisfaisant (par essai-erreur).

Stratégies d'apprentissage

Données et processus :

Analyser un processus, c'est à dire comprendre son fonctionnement, caractériser ses états (normaux ou…), savoir ce qui est significatif, quelle est sa structure, requiert l'observation d'un certain nombre (parfois grand) de variables issues de ce processus et liées à son fonctionnement et son environnement.



Stratégies d'apprentissage

Données et processus :

Si les données ne sont pas représentatives du processus, on ne pourra l'apprendre que partiellement.

Le choix des données est crucial pour un bon apprentissage.
Il faut bien choisir :

- le type des données,
- leur nature,
- leur pertinence/importance,
- la quantité,
- l'ordre dans lequel elles sont prises en compte,
- etc.

Stratégies d'apprentissage

Apprentissage en ligne / apprentissage "batch" (en paquet)

L'apprentissage est un moyen flexible et efficace d'extraire une structure stochastique d'un environnement. Deux types différents d'apprentissage sont utilisés, a savoir l'apprentissage en paquet et l'apprentissage en ligne.

La procédure d'apprentissage en paquet utilise tous les exemples d'entraînement de façon répétée de sorte que sa performance est comparable a celle d'une procédure d'estimation statistique.

L'apprentissage en ligne est plus dynamique, en mettant à jour l'estimation courante par l'observation des nouvelles données une par une. C'est une procédure itérative. L'apprentissage en ligne est en général lent mais est recommandé dans des environnements changeants.

Stratégies d'apprentissage

Apprentissage en ligne / apprentissage "batch" (en paquet)

Il est possible de combiner les 2, dans l'ordre :

1. On récupère un série d'observation à partir du processus à modéliser (en quantité et pertinence suffisante).

2. On effectue un "préapprentissage" : les données récoltées servent à faire converger les poids vers des valeurs proches de la solution finale.

3. On utilise le réseau de neurone tout en utilisant un apprentissage en ligne pour affiner la valeur des poids et pourvoir tenir compte de toutes variations éventuelles (dérives, perturbations, changements de mode de fonctionnement, etc.)

Stratégies d'apprentissage

Apprentissage en ligne / apprentissage "batch" (en paquet)

L'apprentissage "batch"

• requiert souvent de charger en mémoire l'ensemble des poids et des données d'apprentissage (entrées et éventuellement sorties désirées correspondantes),
• ne peut satisfaire une contrainte de temps-réel,
• autorise de réinitialiser sans risque l'apprentissage.

L'apprentissage en ligne

• prend en compte les observations itérativement, au fur et à mesure et demande de ce fait moins de mémoire, moins de calculs,
• est compatible au temps-réel,
• "subit" l'ordre dans lequel les observations sont accessibles.

On cherche à optimiser le plus possible les ressources algorithmiques.

Stratégies d'apprentissage

Apprentissage global / apprentissage local

L'apprentissage global est le processus qui met à jour l'intégralité des poids d'un réseau. Dans ce cas, tous les poids sont modifiés à chaque itération.

L'apprentissage local vise à ne mettre à jour que certains poids à chaque itération. Plusieurs techniques existent :

- à travers le concept de voisinage,
- à travers des règles spécifiques d'apprentissage,
- etc.

Stratégies d'apprentissage

L'apprentissage, c'est le processus (de calculs) qui permet de mettre à jour les poids des neurones à partir d'une ou plusieurs mesures.

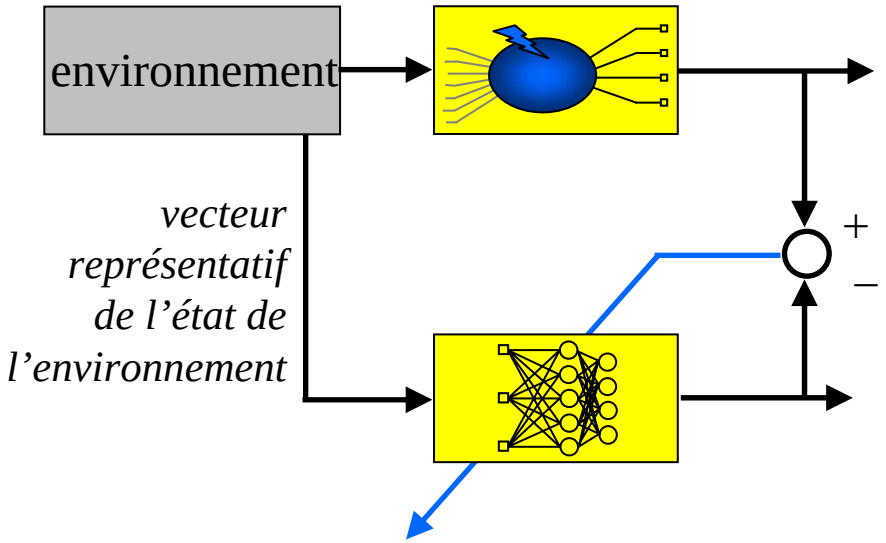
Il existe 3 types d'apprentissage :

- l'apprentissage supervisé,
- l'apprentissage non supervisé,
- et l'apprentissage par assignation de crédit.

Apprentissage supervisé/non supervisé

Apprentissage supervisé

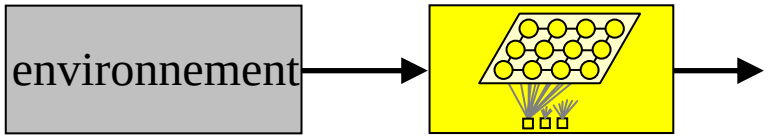
L'apprentissage supervisé, en anglais "Supervised Learning", doit disposer d'un comportement de référence précis pour pouvoir l'inculquer au réseau neuronal.



Apprentissage supervisé/non supervisé

Apprentissage non supervisé

L'apprentissage supervisé s'effectue sous le contrôle d'un expert, alors que l'apprentissage non supervisé (appelé en anglais "unsupervised learning") est autodidacte.



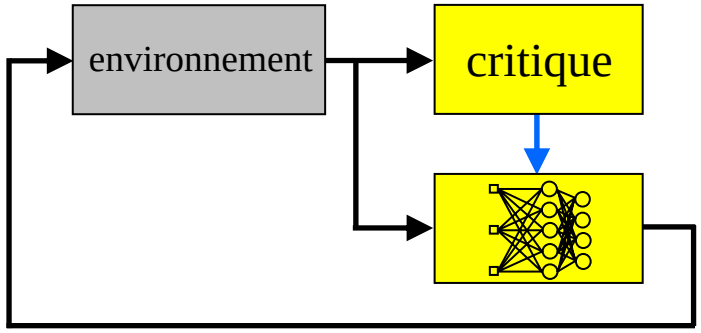
Les paramètres internes du réseau ne sont modifiés qu'avec les seuls stimuli, aucune réponse désirée n'est prise en considération.

Par nature, ce type d'apprentissage construit une représentation interne de la connaissance issue de l'environnement.

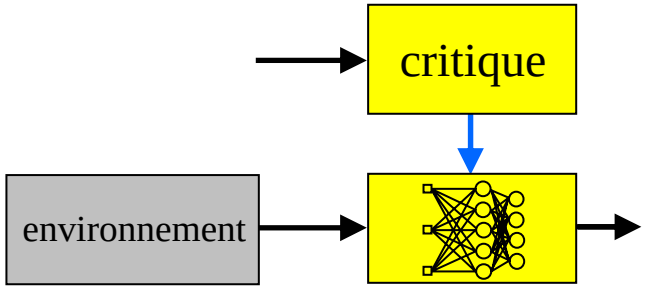
Apprentissage supervisé/non supervisé

Apprentissage par assignation de crédits

Un apprentissage qui ne nécessite pas de comportement de référence explicite mais seulement d'informations grossières, comme un encouragement ou une pénalisation, est appelé apprentissage par renforcement.



Ce type d'apprentissage fait partie d'un schéma plus large d'apprentissage, le "Credit-Assignment Problem" ou CAP.

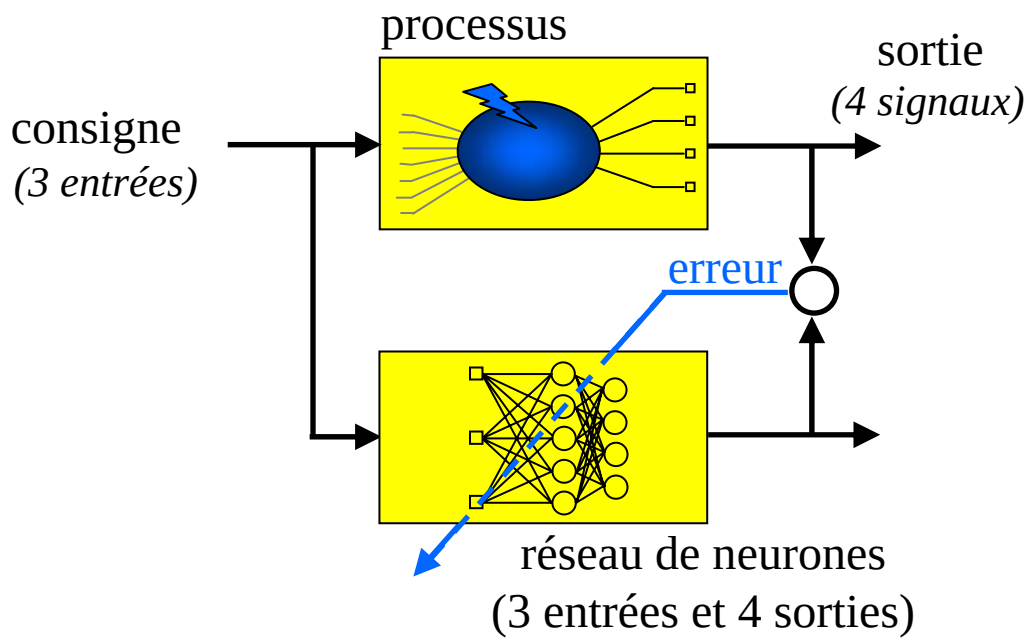


renforcement heuristique

Exemple : apprendre un processus

Apprendre un processus ou un système

De le cas précis où on veut apprendre un système, les mêmes entrées donnent les mêmes réponses (le réseau de neurones cherche à imiter le processus).



Exemple : apprendre un processus

Apprendre un processus ou un système

De le cas précis où on veut apprendre un système :

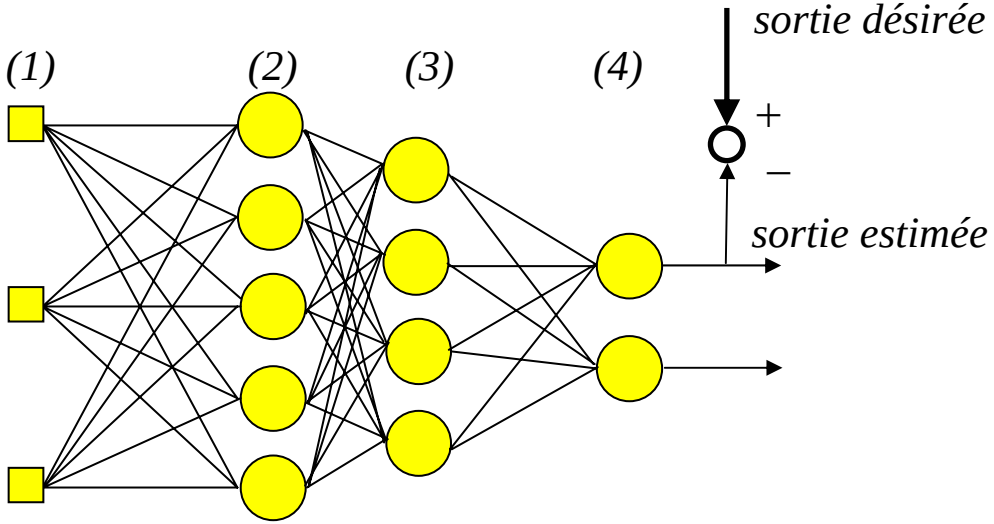
- on utilise un apprentissage supervisé (pour imiter le processus),
- le réseau de neurones à le même nombre d'entrées que le système,
- le réseau de neurones à le même nombre de sorties que le système.

Il faut déterminer les autres paramètres du réseau : apprentissage en ligne/batch, règle et coefficient d'apprentissage, nombre de couches internes et nombre de neurones par couche, etc. On procède généralement par essai-erreur…

Apprentissage d'un réseau multicouche

La rétropropagation de l'erreur dans un réseau multicouche ("MLP with backpropagation") est un apprentissage supervisé.

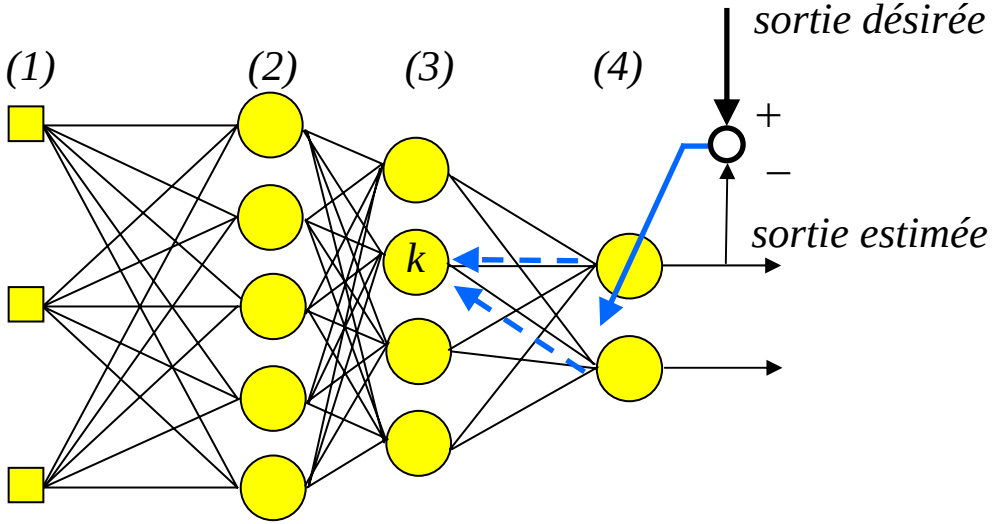
On présente un vecteur d'entrée pour lequel on détermine la sortie du réseau. L'ensemble des poids des liaisons synaptiques détermine le fonctionnement du réseau de neurones. On compare les sorties des neurones de la couche de sortie avec les valeurs modèles (= sorties désirées) et on calcule l'erreur de chacun.



Apprentissage d'un réseau multicouche

L'erreur d'un neurone *k* dans une couche *j* est calculée à partir des erreurs des neurones de la couche *j*+1 pondérées par les poids :

$$\varepsilon_k^{(j)} = f\left(\sum_i w_{ki} x_i\right) . \sum_{i \in j+1} \left(w_{ik}^{(j+1)} . \varepsilon_i^{(j+1)}\right)$$

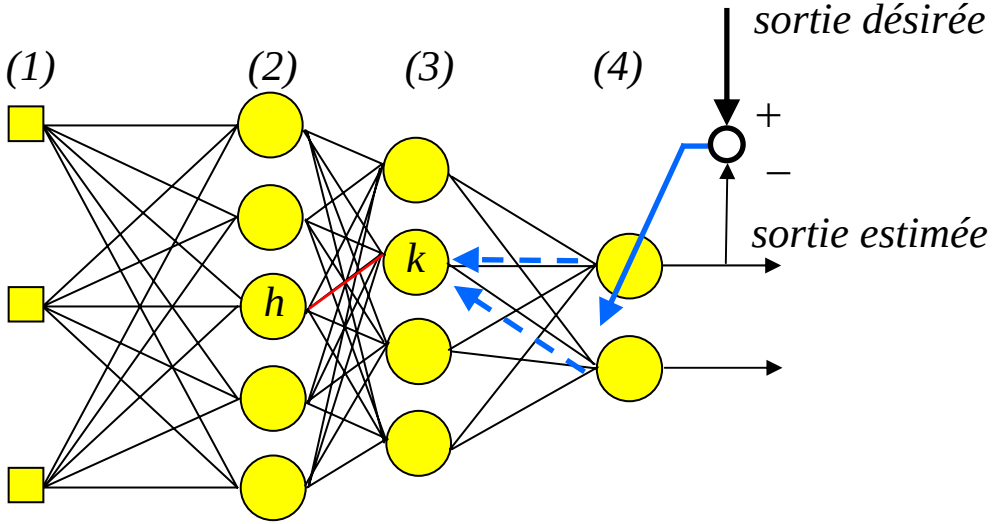


Apprentissage d'un réseau multicouche

En appliquant la méthode du gradient, l'adaptation des poids se fait selon :

$$\Delta w_{kh}^{(j)} = -\alpha.\varepsilon_k^{(j)}.y_h^{(j-1)} = -\alpha\left[f\left(\sum_i w_{ki}x_i\right).\sum_{i\in j+1}(w_{ik}^{(j+1)}.\varepsilon_i^{(j+1)})\right].y_h^{(j-1)}$$

$$w_{kh}^{(j)}(t+1) = \Delta w_{kh}^{(j)} + w_{kh}^{(j)}(t)$$



Apprentissage d'un réseau multicouche

Nous avons vu la rétropropagation de l'erreur qui consiste à rétropropager l'erreur commise par un neurone à ses synapses et aux neurones qui y sont reliés.

L'erreur est d'abord définie à la sortie, par rapport à des valeurs souhaitées (mesurées sur un système) puis propagées de la couche de sortie jusqu'à la couche des entrées pour mettre à jour l'ensemble des poids entre les neurones des couches successives.

Ainsi ce principe consiste à corriger les erreurs selon l'importance des éléments qui ont justement participé à la réalisation de ces erreurs : les poids synaptiques qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale.

Apprentissage d'un réseau multicouche

Plusieurs règles d'apprentissage peuvent s'appliquer aux réseaux multicouches, parmi elles :

• une méthode batch : l'algorithme de Levenberg-Marquardt (LM)
• des méthodes itératives : Backpropagation through time (BPTT) et la descente de gradient,
• des méthodes itératives semi-directes : le gradient conjugué, l'algorithme EM (espérance-maximisation), le recurrent Extended Kalman Filter (EKF).
• …

Remarque :
la régression linéaire est une méthode d'optimisation directe.

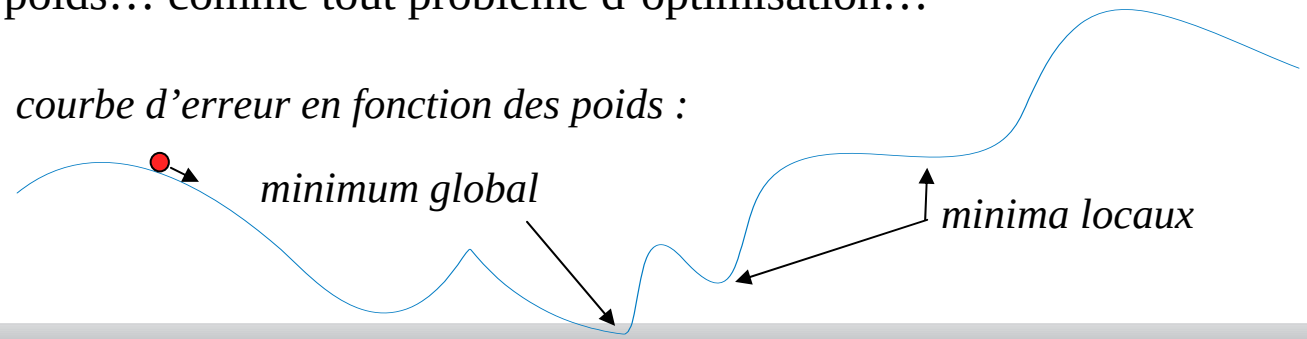
Apprentissage d'un réseau multicouche

convergence/divergence

La rétropropagation de l'erreur avec un coefficient d'apprentissage faible conduit à une convergence lente alors qu'un coefficient élevé risque de produire des oscillations.

Pour les méthodes du second ordre, il faut connaître les dérivées des fonctions d'activation car elles nécessitent la dérivée de l'erreur par rapport aux poids.

Toutes ces méthodes dépendent beaucoup des valeurs initiales des poids… comme tout problème d'optimisation…

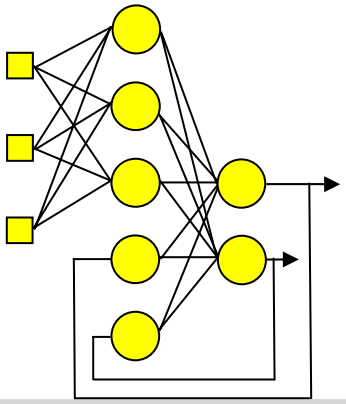


Apprentissage d'un réseau multicouche

Réseau multicouche récurrent

Le réseau multicouche qui a été présenté est un réseau statique. Il est également appelé réseau « feed-forward » car le flux d'information est dirigé des entrées vers les sorties.

Une extension de ce réseau est le réseau multicouche récurrent qui présente un caractère dynamique : ses poids dépendent non seulement des entrées apprises, mais également des sorties précédentes.



Dans cet exemple, les 2 sorties sont rebouclées comme des entrées des neurones de la couche cachée.

Apprentissage d'un réseau multicouche

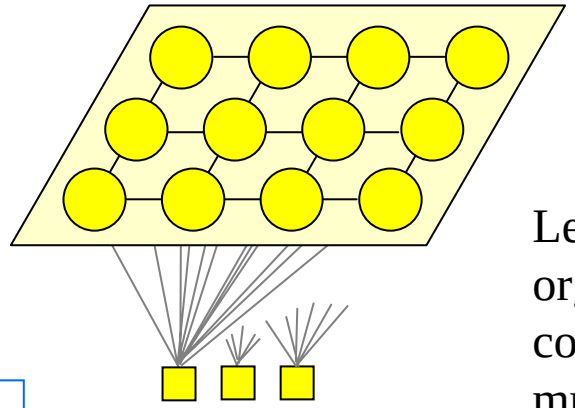
Réseau multicouche récurrent

Le réseau multicouche récurrent est particulièrement adapté pour estimer des processus et des systèmes dynamiques et non linéaires car :

• il présente de manière inhérent un caractère dynamique,
• chaque neurones introduit une portion de non linéarité,
• il est capable d'apprendre et peut donc s'adapter en ligne à toute variation de paramètres du système ou tout changement de son environnement (variation de bruit, dérive, etc.)

Apprentissage d'une carte auto-organisatrice

L'auto-organisation d'une carte de Kohonen est un apprentissage non supervisé (SOM : Self-Organizing Map).

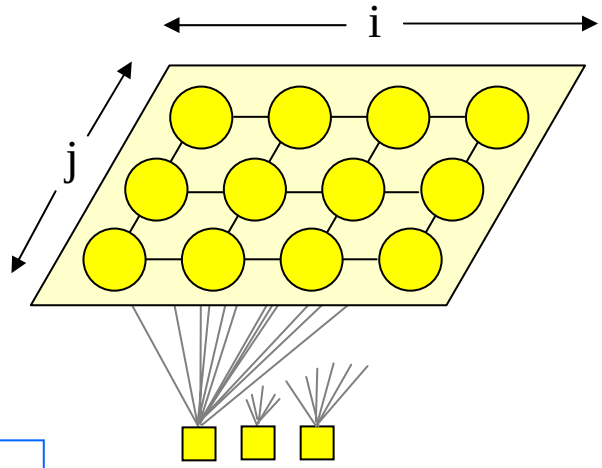


Les neurones d'une carte de Kohonen sont organisés dans une seule couche qui peut être considérée comme une grille multidimensionnelle.

L'exemple ci-contre comprend :
1 couche = 1 grille bidimensionnelle de 3 × 4 neurones
3 entrées

Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation :



Il n'y a pas de couche de sortie, pas plus que des neurones de sortie.

La position (indices i et j) d'un neurone au sein de la grille est fondamentale et permet de définir la notion de voisinage.

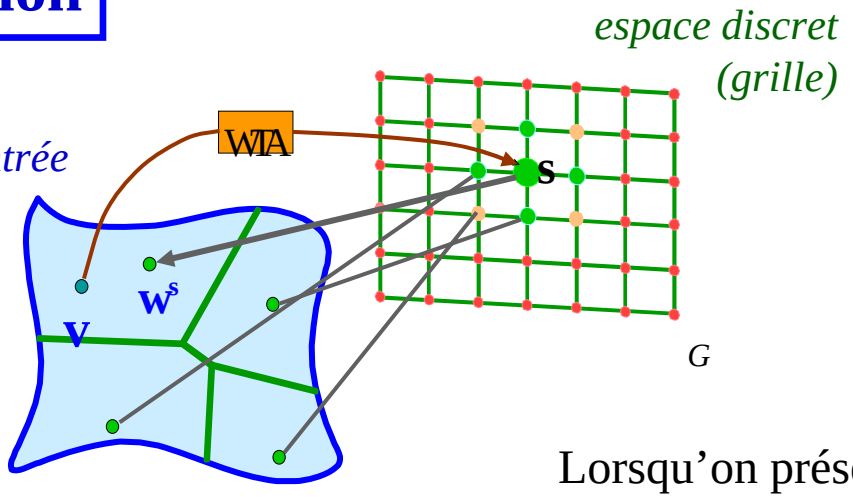
Chaque neurone possède des poids qui définissent sa position dans l'espace des entrées.

L'apprentissage s'effectue en 2 étapes :
- la compétition entre les neurones,
- l'adaptation des neurones.

Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : la compétition

compétition



distance la plus courte :

$$\mathbf{s} = \arg\min_{r \in G} \left\| \mathbf{x}_t - \mathbf{m}_r \right\|$$

Lorsqu'on présente un vecteur d'entrée, on cherche le neurone le plus proche. Pour cela on calcule la distance entre l'entrée $\mathbf{x}_t$ et les poids des neurones $\mathbf{m}_r$.

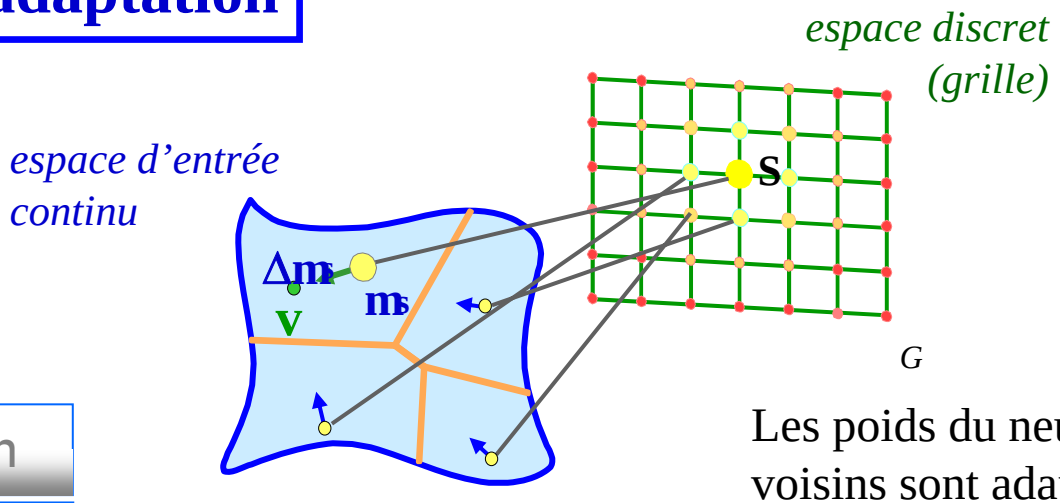
$r = \{i, j\} \in G$ est le vecteur d'indice des neurones

$\mathbf{s} \in G$ est l'indice du neurone vainqueur

Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : l'adapation

adaptation



Les poids du neurone vainqueur et de ses voisins sont adaptés :

$$\Delta \mathbf{m}_r = \mu \, h_{rs} \cdot (\mathbf{x} - \mathbf{m}_r)$$

μ	est un coefficient d'apprentissage
$\mathbf{m}_r$	est le vecteur poids des neurones
h_{rs}	est une fonction de voisinage

Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : optimisation des paramètres

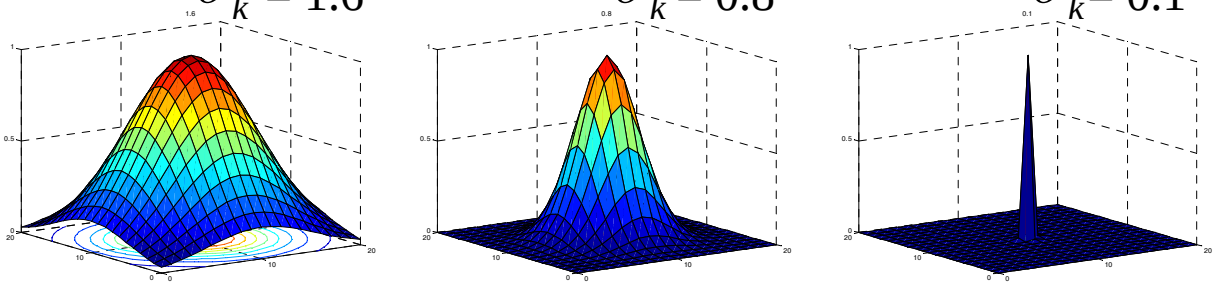
La fonction de voisinage permet d'adapter les poids de plusieurs neurones, pas seulement ceux du vainqueur.

Elle s'écrit $h_{rs} = \exp\left(-\frac{d_{r,s}^2}{2\sigma^2}\right)$ est généralement modifiée au cours du temps

en réduisant son rayon : $\sigma_k = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{\frac{k}{k_{\max}}}$

Exemple :

$\sigma_k = 1.6$ $\sigma_k = 0.8$ $\sigma_k = 0.1$



Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : exemple 1

Une carte de Kohonen est utilisée pour apprendre la série temporelle de Hénon.
Cette série est considérée comme un attracteur :

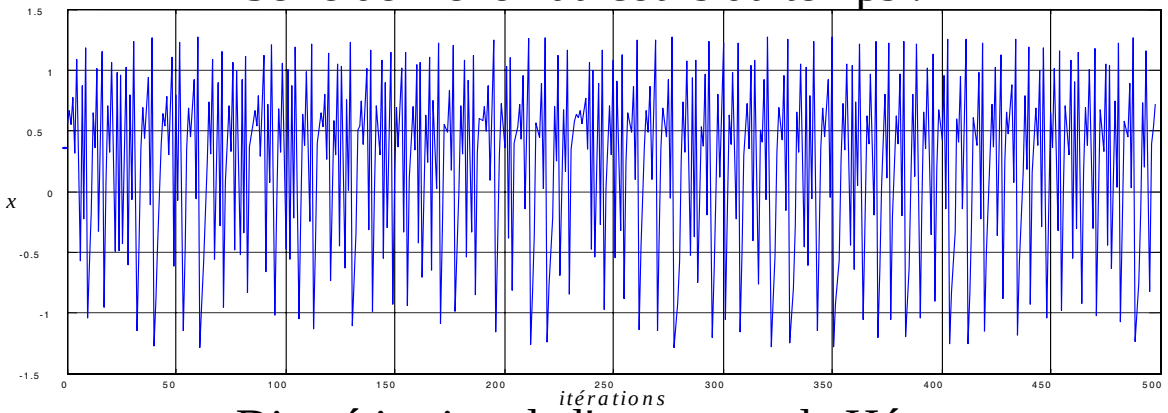
$$\begin{cases} x_k = 1 - 1.4x_{k-1}^2 + y_{k-1} \\ y_k = 0.3x_{k-1} \end{cases}$$

La carte comprend 25×25 neurones.
Le vecteur d'entrée est constitué de $\begin{bmatrix} x_k & y_k \end{bmatrix}^T$.

Apprentissage d'une carte auto-organisatrice

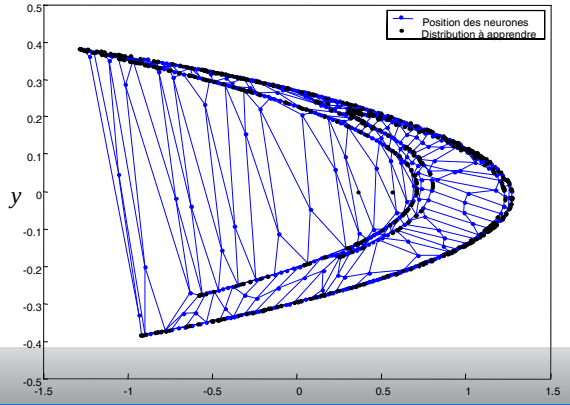
Principe de l'auto-organisation : exemple 1

Série de Hénon au cours du temps :

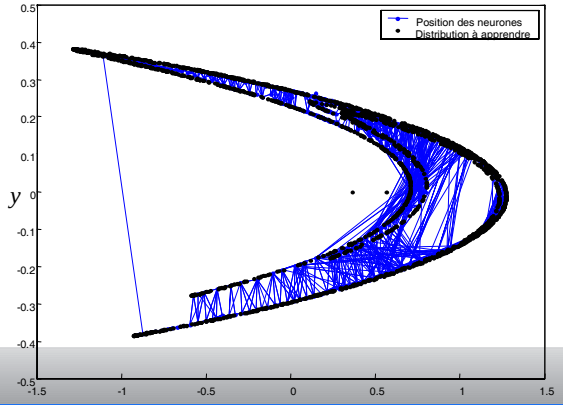


Discrétisation de l'attracteur de Hénon

par une carte de Kohonen



par une carte "neural gas"



Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : exemple 1

Les 2 cartes effectuent une discrétisation de l'espace des entrées.

La nature spécifique de la fonction de voisinage intervenant dans les cartes de Kohonen et "neural gas" leur confère des propriétés distinctes qui font aboutir l'apprentissage à une quantification différente de l'espace d'entrée.

La répartition des neurones de la carte de Kohonen préserve les relations de voisinage imposées par la grille topologique alors que le "neural gas" n'est pas basé sur un maillage ou sur une grille.

Le réseau "neural gas" optimise le positionnement des neurones lorsque la distribution des exemples est non convexe.

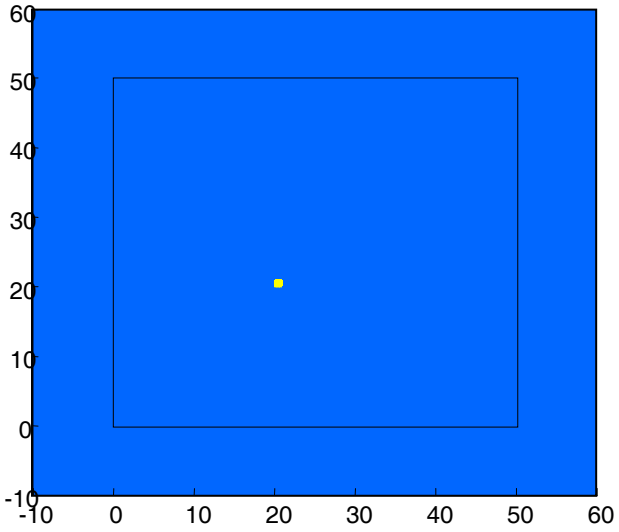
Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : exemple 2

ALGORITHME :

Etape 1 : initialisation
Etape 2 : choix d'une entrée
Etape 3 : détermination du neurone vainqueur
Etape 4 : adaptation des poids des neurones
⇨ Retour à l'étape 2

RESULTAT
(initialisation)



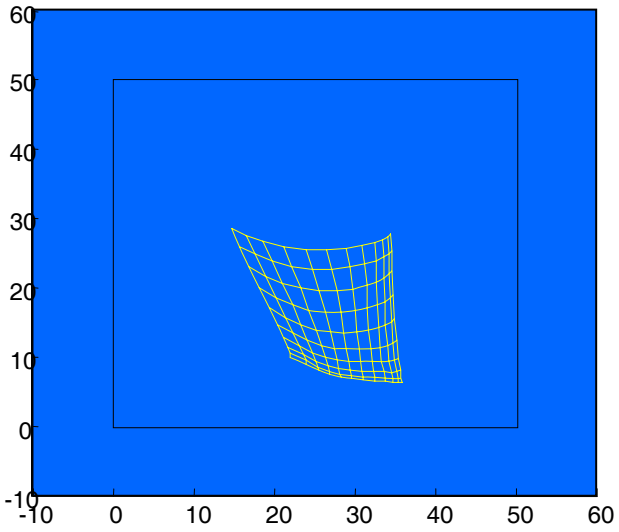
Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : exemple 2

ALGORITHME :

Etape 1 : initialisation
Etape 2 : choix d'une entrée
Etape 3 : détermination du neurone vainqueur
Etape 4 : adaptation des poids des neurones
⇨ Retour à l'étape 2

RESULTAT
(itération 100)



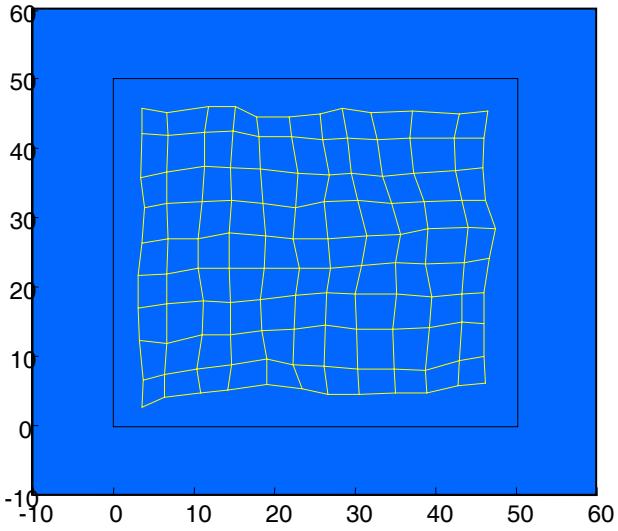
Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : exemple 2

ALGORITHME :

Etape 1 : initialisation
Etape 2 : choix d'une entrée
Etape 3 : détermination du neurone vainqueur
Etape 4 : adaptation des poids des neurones
⇨ Retour à l'étape 2

RESULTAT
(itération 1000)

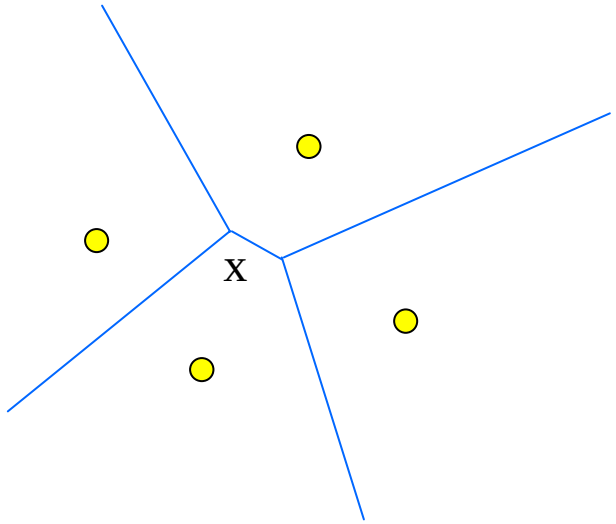


Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : vue plus générale

L'architecture des cartes auto-organisatrices appelée encore cartes topographiques (à apprentissage compétitif) est bien adaptée au problème de classification :

Chaque neurone se positionne avec ses poids dans l'espace d'entrée, et couvre une zone. Un point de cet espace se retrouve dans la zone définie par le neurone le plus proche.



L'apprentissage déplace les neurones pour les adapter à la densité de l'espace d'entrée.

Apprentissage d'une carte auto-organisatrice

Principe de l'auto-organisation : vue plus générale

Il existe différentes cartes auto-organisatrices ou carte topographique (à apprentissage compétitif) :

• la carte auto-organisatrice de Kohonen (1982) à apperntissage non supervisé,
• le neural gas,
• les SVM.

Les machines à vecteurs de support (en anglais Support Vector Machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires et ont été développés dans les années 1990.

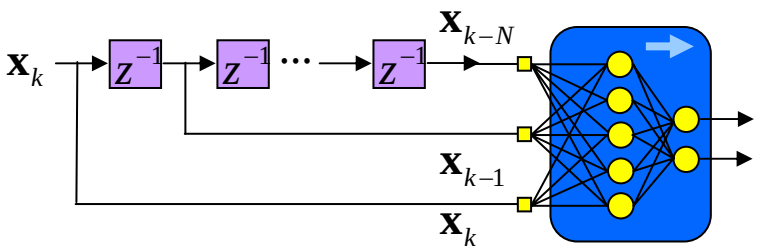
Réseaux de neurones artificiels : architectures et applications

Apprentissage de systèmes dynamiques

Apprentissage de systèmes dynamiques

Apprendre la dynamique d'un système

Principe: la dynamique du système est "captée" à l'aide de lignes à retard.

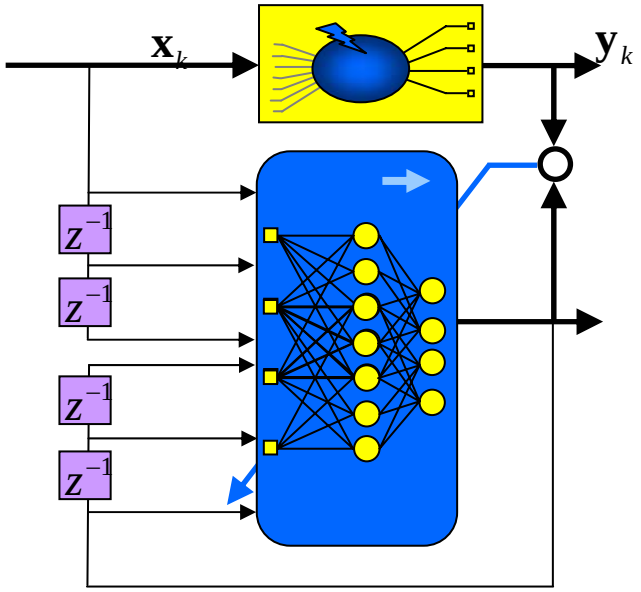


time-delayed neural network
with tapped delay line

Apprentissage de systèmes dynamiques

Apprendre la dynamique d'un système

Souvent, l'entrée et la sortie sont représentatif de la dynamique d'un système.
Dans ce cas, il faut en tenir compte explicitement pour réaliser l'apprentissage.

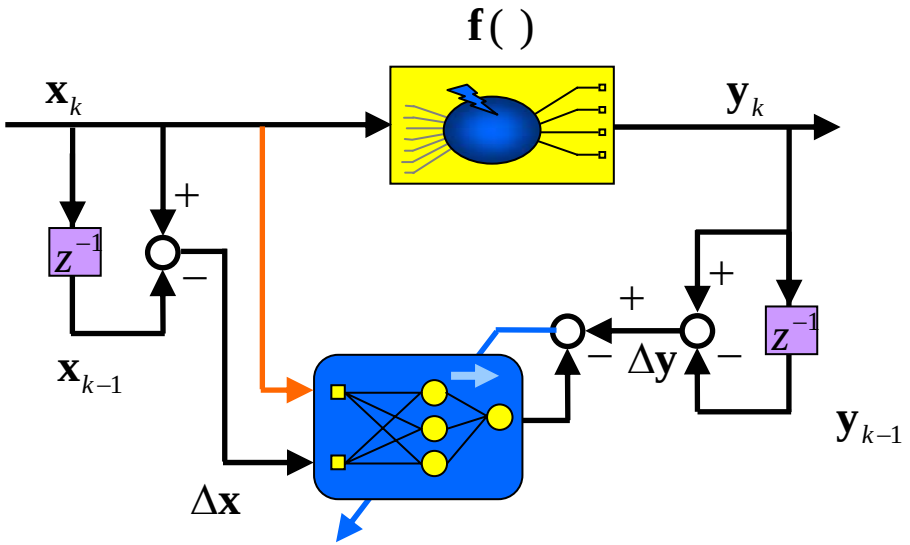


neural network with time delayed input and output

Apprentissage de systèmes dynamiques

Apprendre le Jacobien d'un système

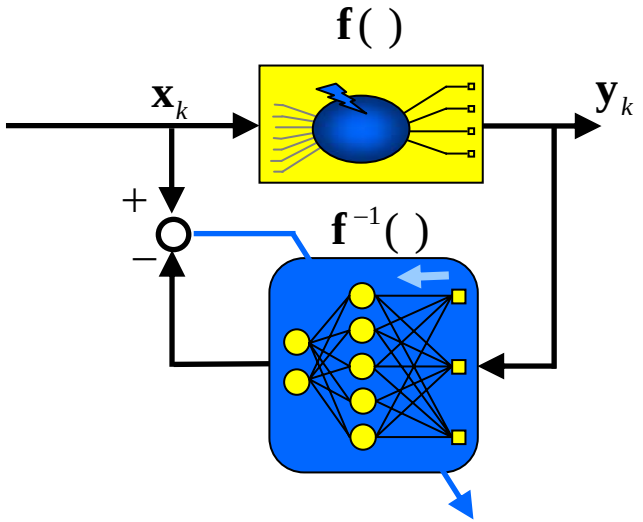
Le schéma d'apprentissage suivant les dérivées (différences entre 2 instants successifs) des entrées et des sorties pour apprendre le Jacobien du processus.



Apprentissage de systèmes dynamiques

Apprendre l'inverse d'un système

Le réseau suivant prend en entrée la sortie du processus, sa sortie désirée est l'entrée du processus (la consigne). L'apprentissage permet d'estimer l'inverse du processus.



training of the inverse dynamic model of a plant

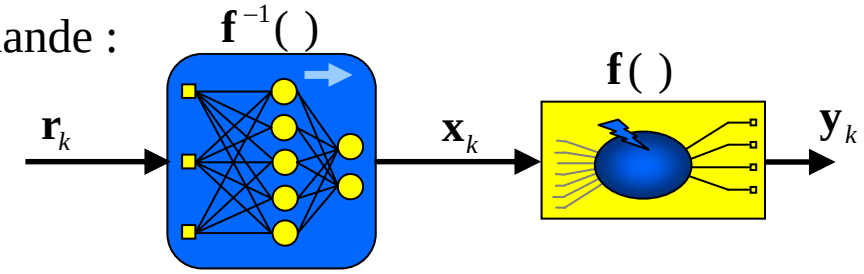
Apprentissage de systèmes dynamiques

Utilisation du réseau ayant appris l'inverse d'un système

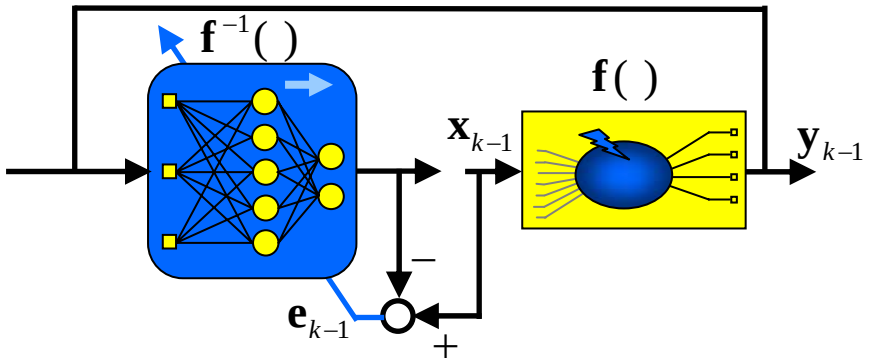
Commande simple avec apprentissage en ligne :

A chaque itération,

1. on calcule une commande :



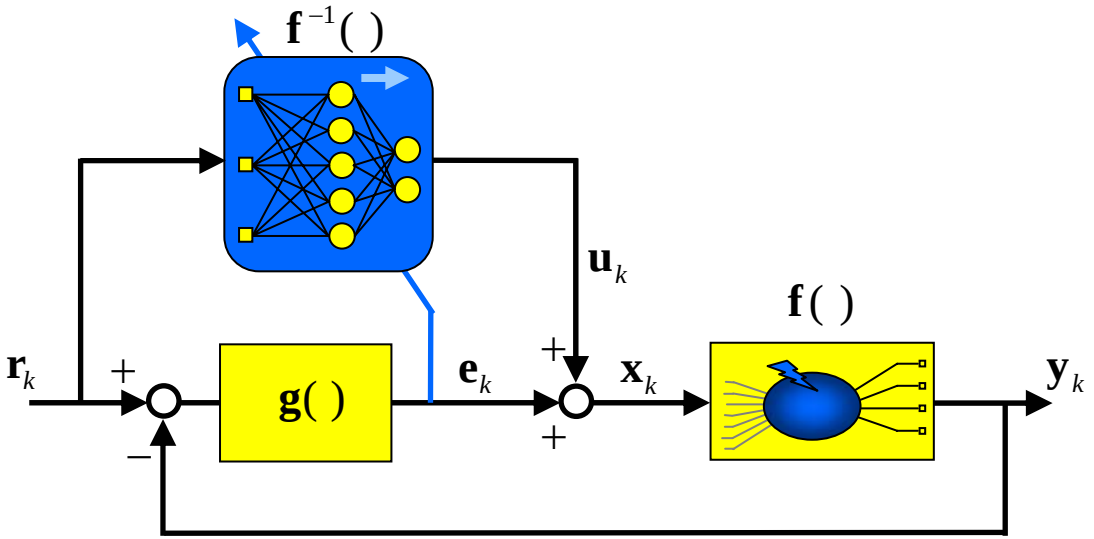
2. on adapte les poids :



Apprentissage de systèmes dynamiques

Utilisation du réseau ayant appris l'inverse d'un système

Commande plus complexe avec apprentissage en ligne :

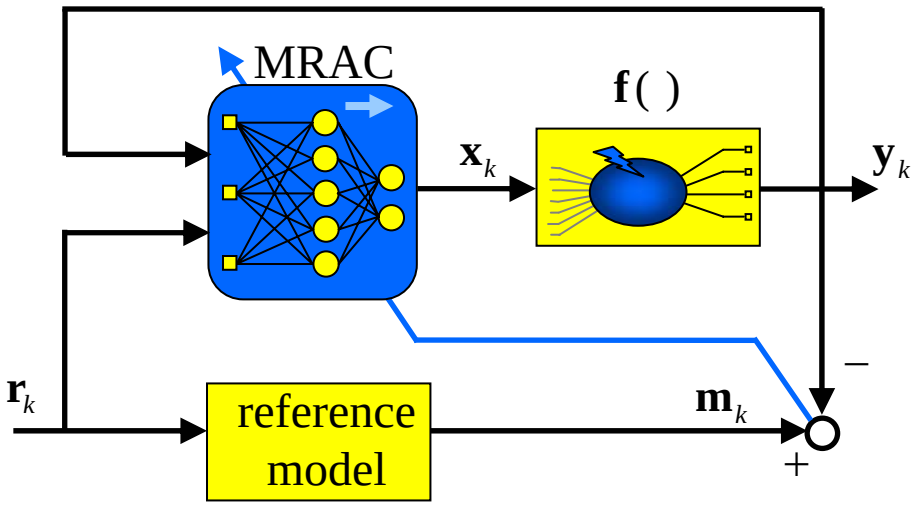


inverse dynamic model-based adaptive control of a plant

Apprentissage de systèmes dynamiques

MRAC: Model Referencing Adaptive Control

Méthode directe

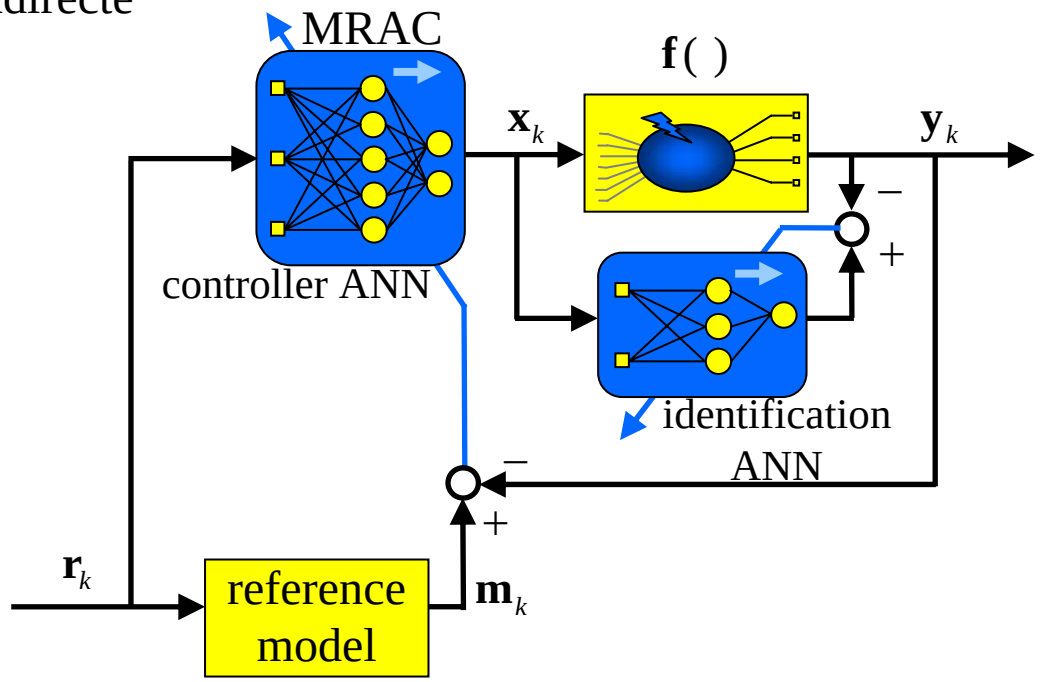


direct dynamic model-based adaptive control of a plant

Apprentissage de systèmes dynamiques

MRAC: Model Referencing Adaptive Control

Méthode indirecte

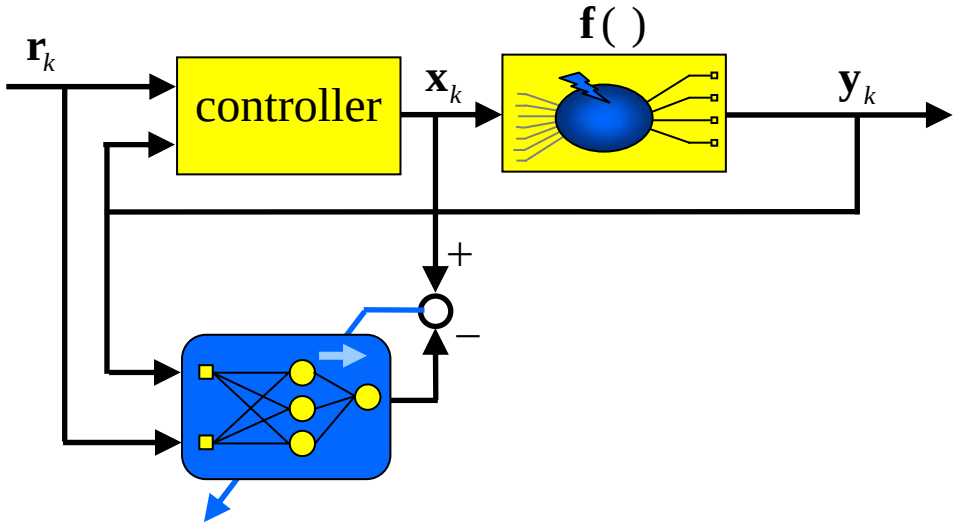


inverse dynamic model-based adaptive control of a plant

Apprentissage de systèmes dynamiques

Emuler un contrôleur existant

Il est parfois utile d'estimer la loi de commande d'un système. Dans le schéma d'apprentissage ci-dessous, le contrôleur pour être remplacé par le réseau de neurones dès que celui-ci aura appris avec une précision suffisante.

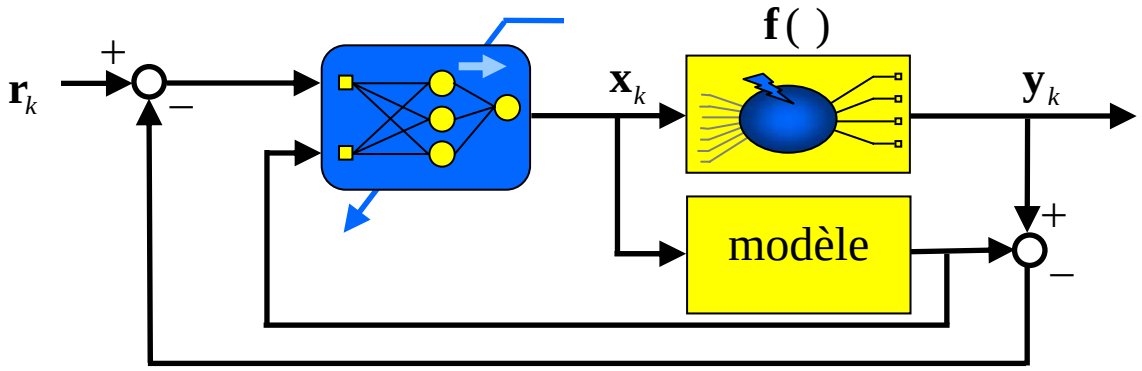


training for emulation of an actual controller

Apprentissage de systèmes dynamiques

Contrôle avec un modèle « interne »

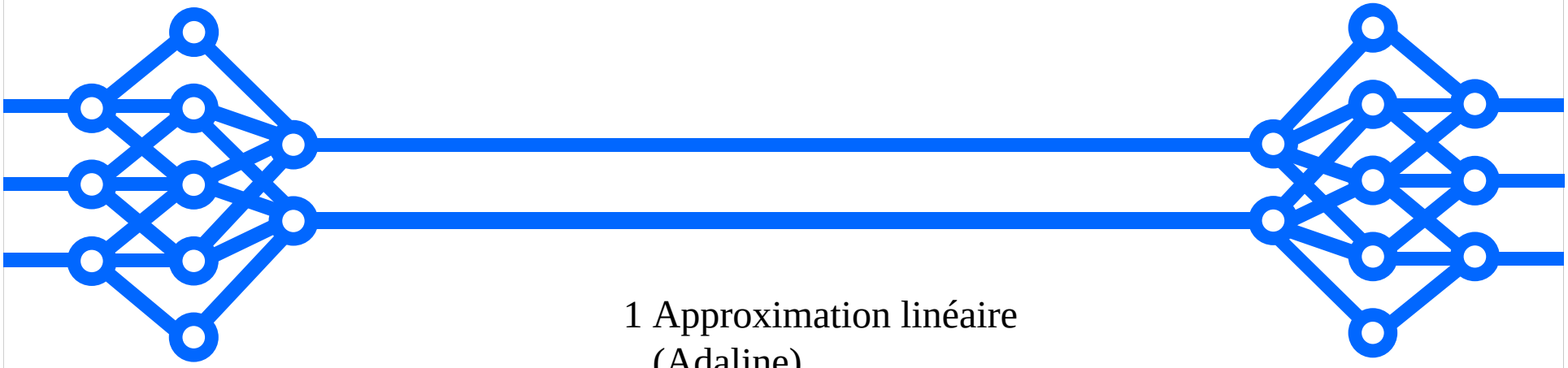
Ce schéma modifie la consigne en soustrayant les perturbations et les erreurs de modélisation.



Réseaux de neurones artificiels : architectures et applications

Application des réseaux de neurones

Applications des réseaux de neurones



1 Approximation linéaire
(Adaline)

2 Apprentissage de fonctions
(carte de Kohonen)

3 Estimation de fonctions robotiques
(carte de Kohonen et Adaline)

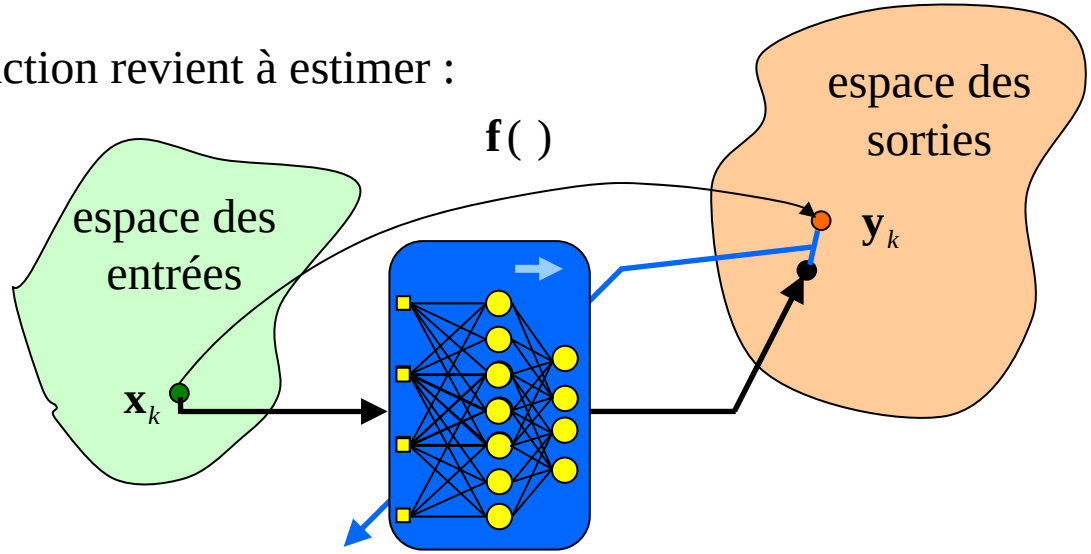
4 Identification d'une fonction de transfert
(réseau multicouche)

Applications

Estimation de fonction

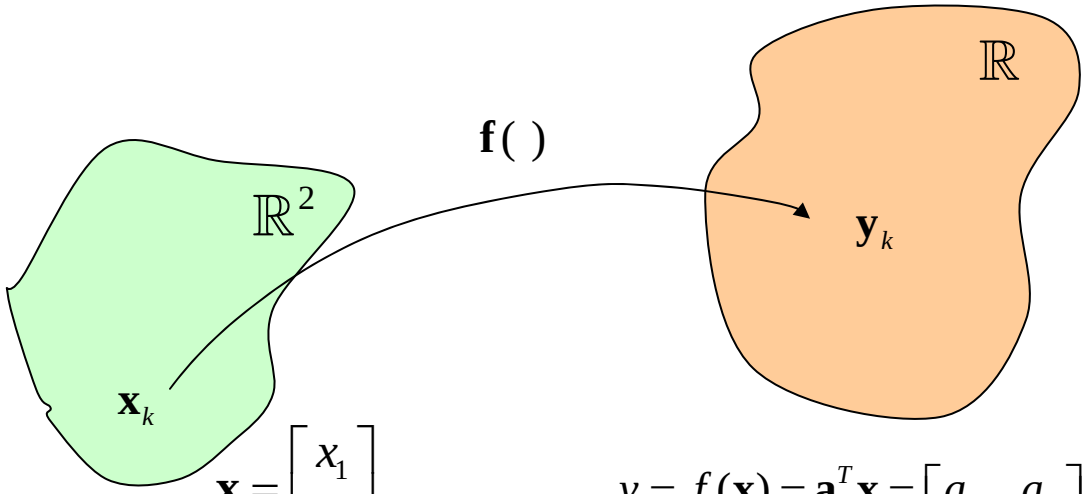
Estimer une fonction est un cas général particulièrement intéressant, car de nombreux processus peuvent être assimilés à une fonction ou une transformation.

Estimer une fonction revient à estimer :



Application 1 : Adaline

Objectif : estimer une relation linéaire à l'aide d'un Adaline.



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y = f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Application 1 : Adaline

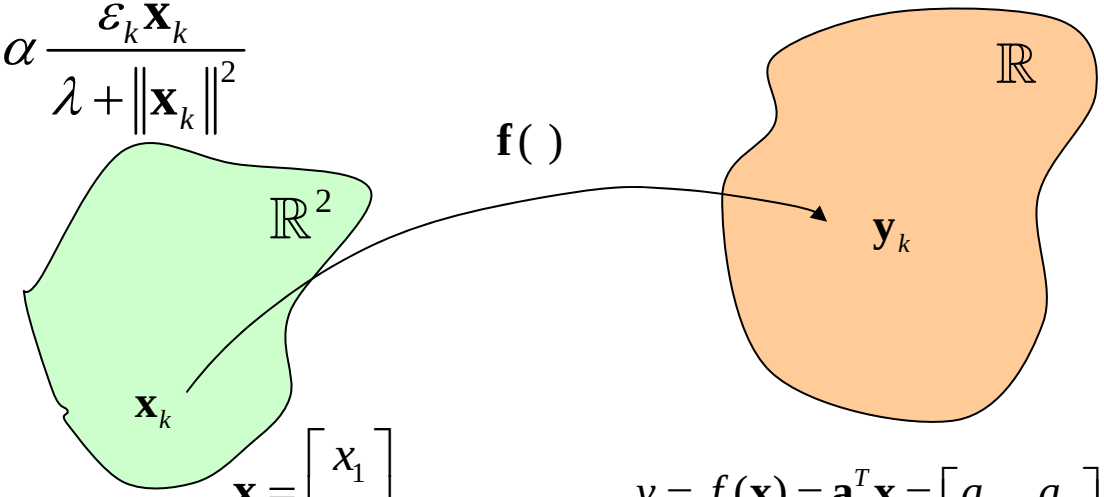
Objectif : estimer une relation linéaire à l'aide d'un Adaline.

$$\hat{y} = \mathbf{w}_k^T \mathbf{x}_k$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta\mathbf{w}_k$$

$$\varepsilon_k = y_k - \hat{y}_k = y_k - \mathbf{w}_k^T \mathbf{x}_k$$

$$\Delta\mathbf{w}_k = \alpha \frac{\varepsilon_k \mathbf{x}_k}{\lambda + \|\mathbf{x}_k\|^2}$$



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y = f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Application 1 : Adaline

Objectif : estimer une relation linéaire à l'aide d'un Adaline.

Jeu d'apprentissage :

avec

$\mathbf{a} = [0.5 \quad 1.33]^T$

$\mathbf{x}_1 = [0 \quad 1]^T, y_k = 1.33$

$\mathbf{x}_2 = [1 \quad 0]^T, y_k = 0.50$

$\mathbf{x}_3 = [1 \quad 1]^T, y_k = 1.83$

$\mathbf{x}_4 = [2 \quad 7]^T, y_k = 10.31$

$\mathbf{x}_5 = [0.5 \quad 8]^T, y_k = 10.89$

$\mathbf{x}_6 = [-2 \quad 3.14]^T, y_k = 3.17$

$\mathbf{x}_7 = [-3.14 \quad 0]^T, y_k = -1.57$

$\mathbf{x}_8 = [-1.1 \quad 10]^T, y_k = 12.75$

$\mathbf{x}_9 = [0.5 \quad -1]^T, y_k = -1.08$

$\mathbf{x}_{10} = [4.5 \quad -5]^T, y_k = -4.40$

Après apprentissage, on obtient : $\mathbf{w} = \hat{\mathbf{a}} = [0.5 \quad 1.33]^T$
avec une erreur de 10^-3 :

- en 4 itérations avec $\alpha = 1$
- en 12 itérations avec $\alpha = 0.5$
- en 26 itérations avec $\alpha = 0.1$

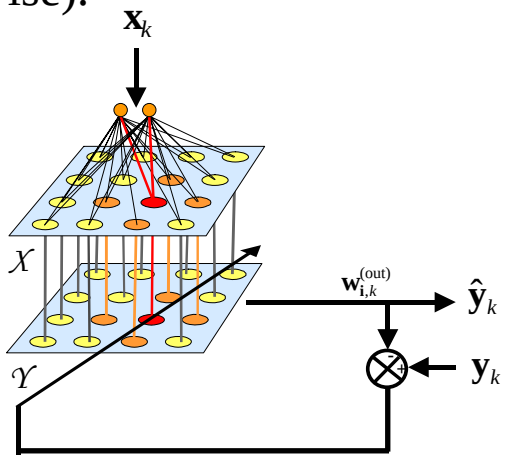
$\mathbf{w}_{init} = [-0.5 \quad 1.5]^T$

Application 2 : carte de Kohonen

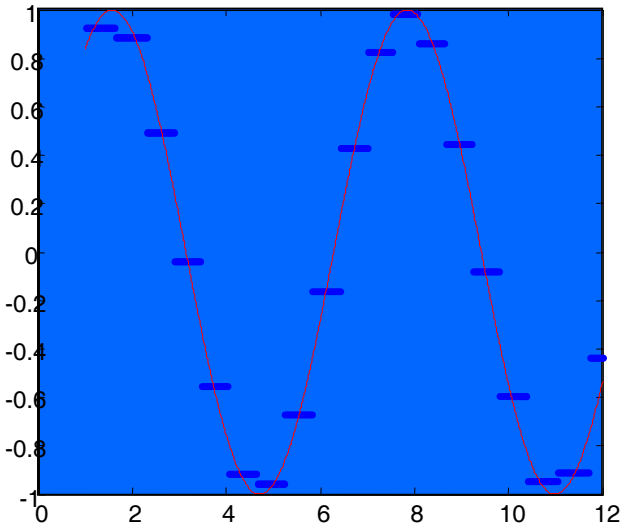
Objectif : estimer une relation non linéaire avec une carte de Kohonen.

ARCHITECTURE :

Il s'agit en fait d'une double carte avec :
1 grille dans l'espace des entrées (apprentissage non supervisé),
1 grille dans l'espace des sorties (apprentissage supervisé).



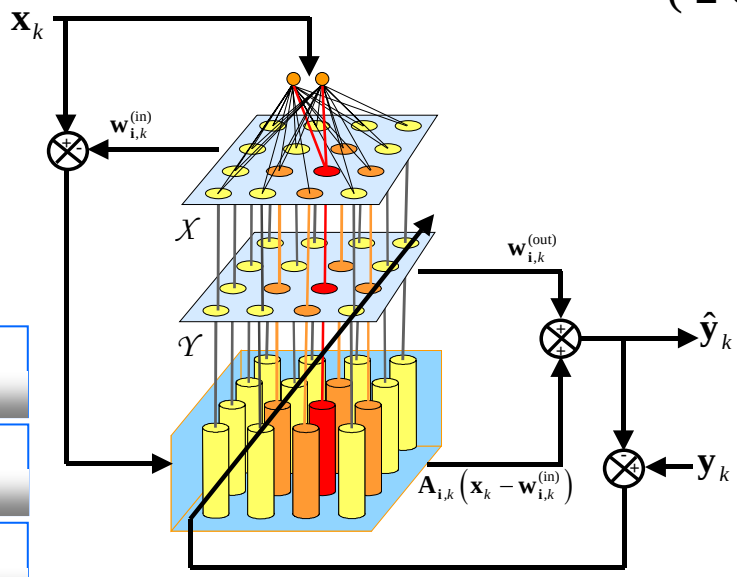
RESULTAT
(2 cartes de 1×19 neurones)



Application 2 : carte de Kohonen

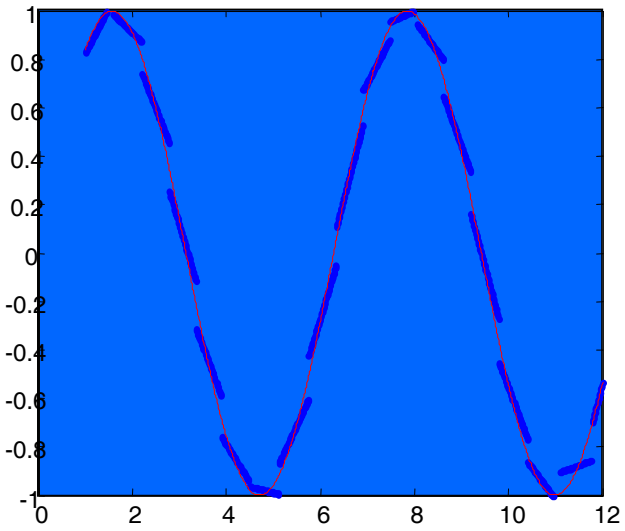
Objectif : estimer une relation non linéaire avec une carte de Kohonen.

ARCHITECTURE :



RESULTAT

(2 cartes de 1×19 neurones et 19 Adalines)

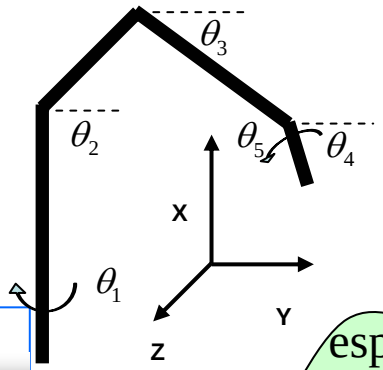


SOM-LLM

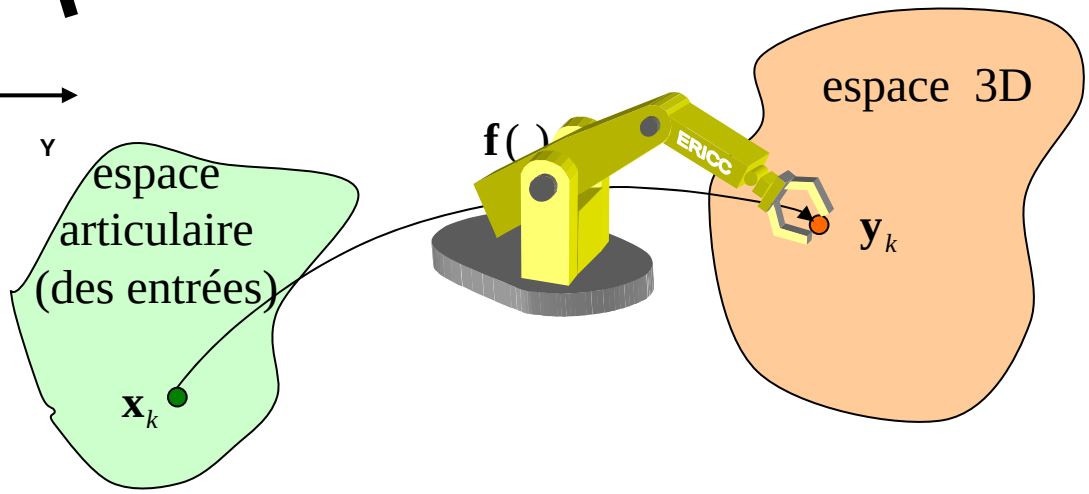
(Self-Organizing Maps-Linear Local Models)

Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.



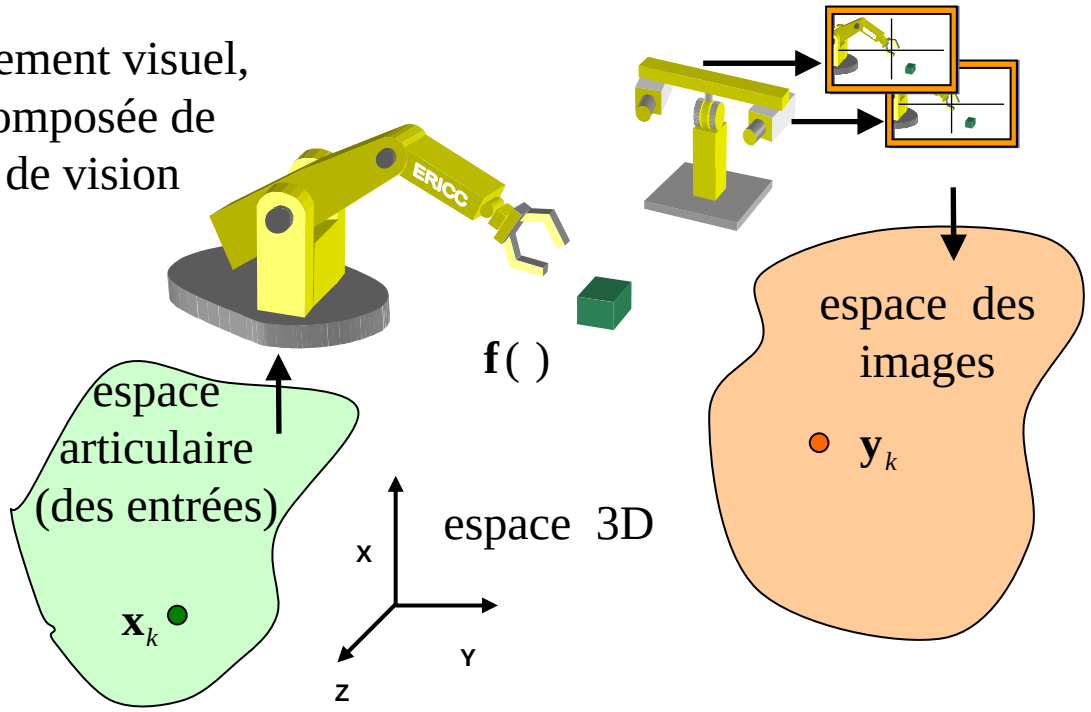
Un robot (ici un bras manipulateur) est une fonction.
Le modèle direct du robot est calculé à l'aide de fonctions trigonométriques.



Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.

Dans un schéma d'asservissement visuel, la fonction du système est composée de celle du robot et du système de vision (projection perspective).

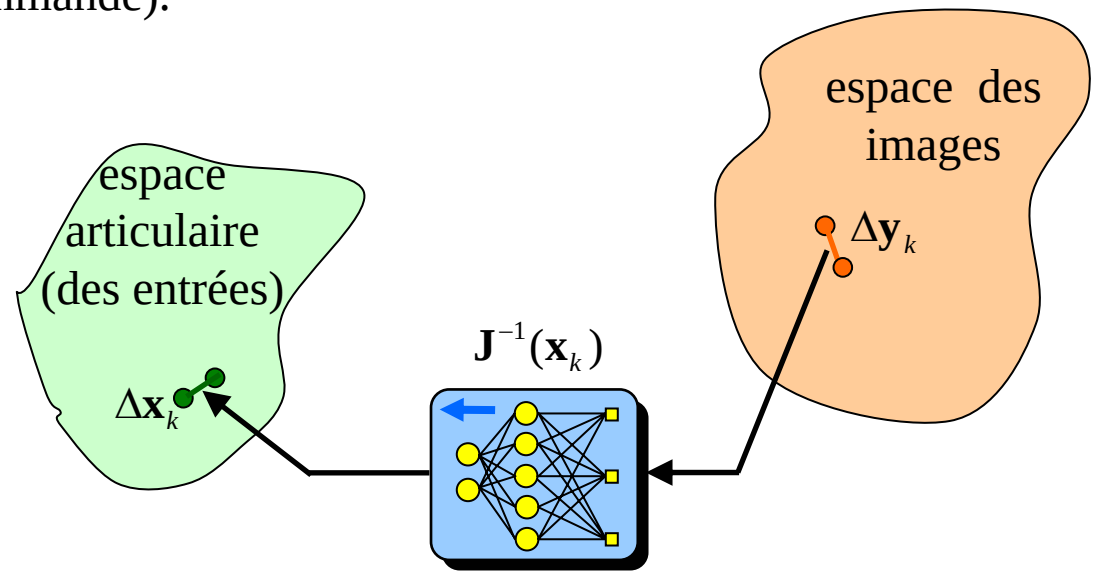


L'espace 3D est mesurée "à distance" par 2 caméras.

Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.

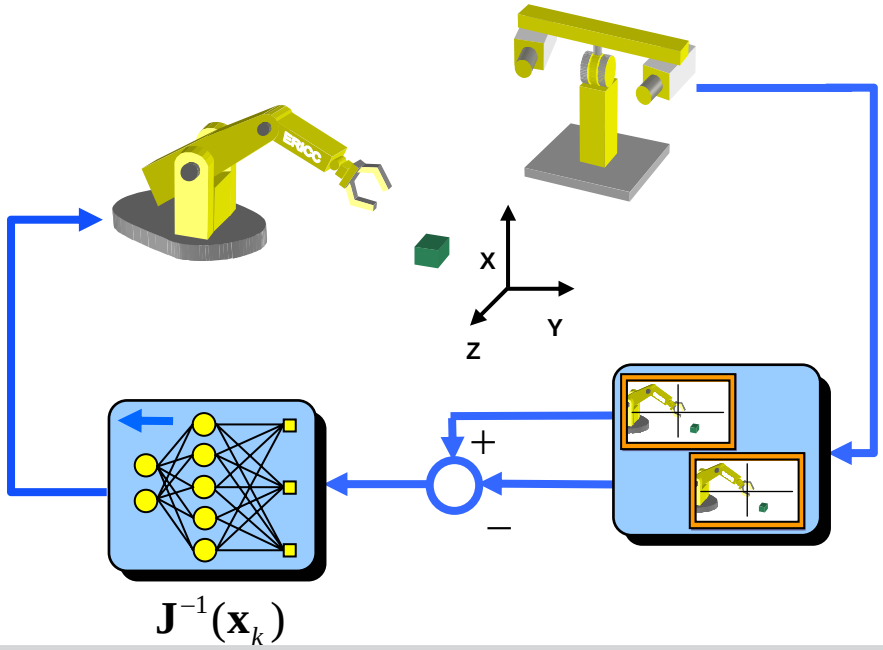
En estimant le Jacobien de la fonction , il est possible d'associer à un déplacement dans les images (une erreur), un déplacement dans les angles (une commande).



Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.

On en déduit une loi de commande :

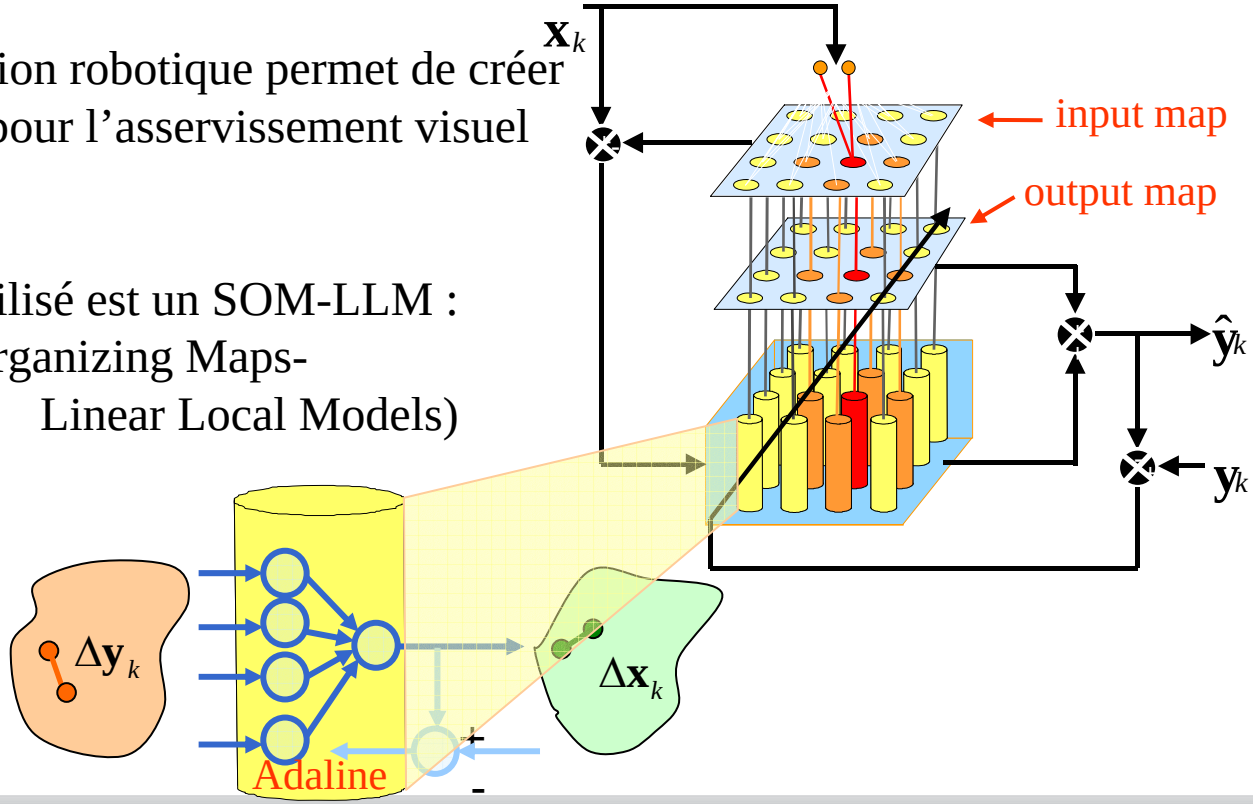


Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.

L'estimation de la fonction robotique permet de créer un contrôleur neuronal pour l'asservissement visuel du bras.

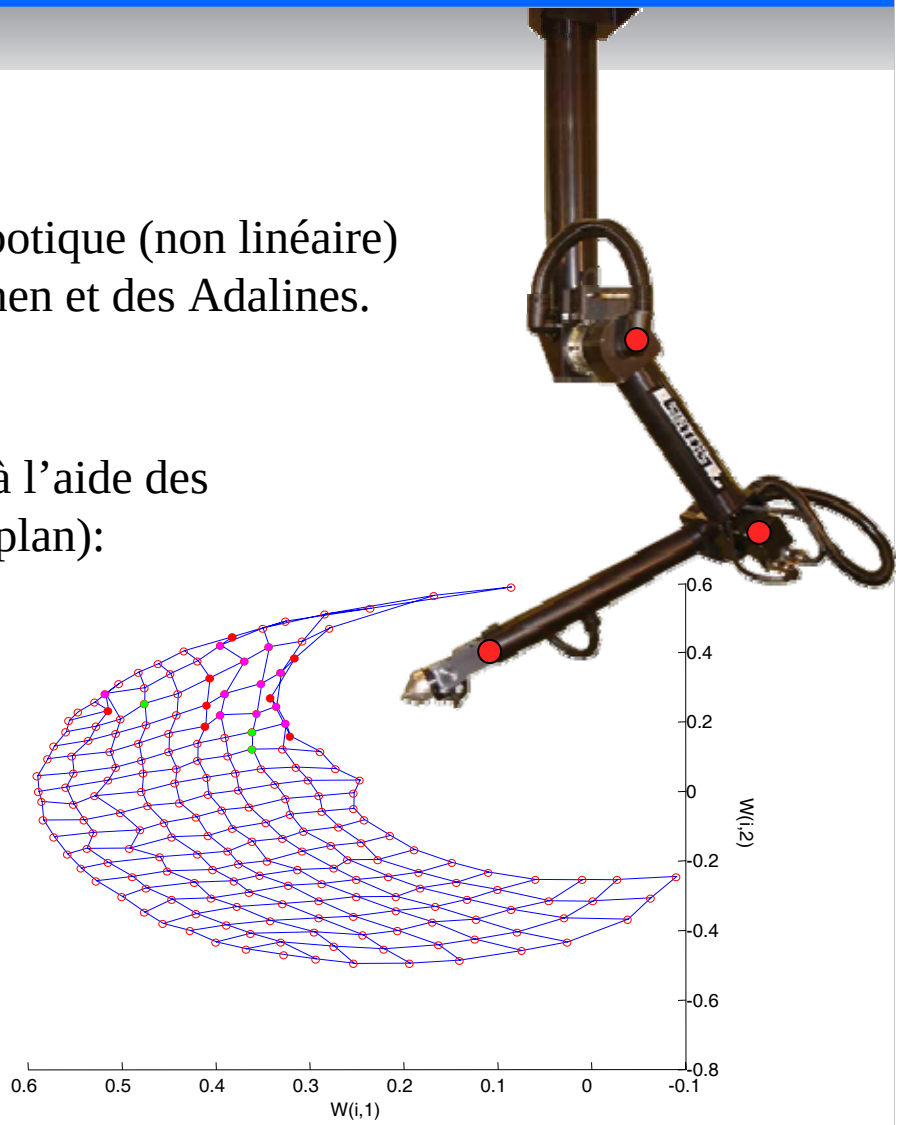
Le réseau de neurone utilisé est un SOM-LLM :
(Self-Organizing Maps-
Linear Local Models)



Application 3 : carte de Kohonen + Adaline

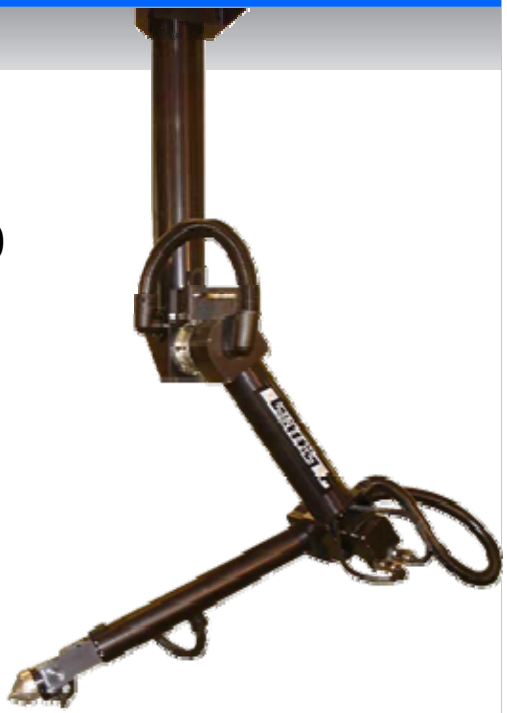
Objectif : estimer une fonction robotique (non linéaire) avec une carte de Kohonen et des Adalines.

Discrétisation de l'espace de travail du robot à l'aide des réseaux SOM-LLM (robot 3 axes dans un plan):

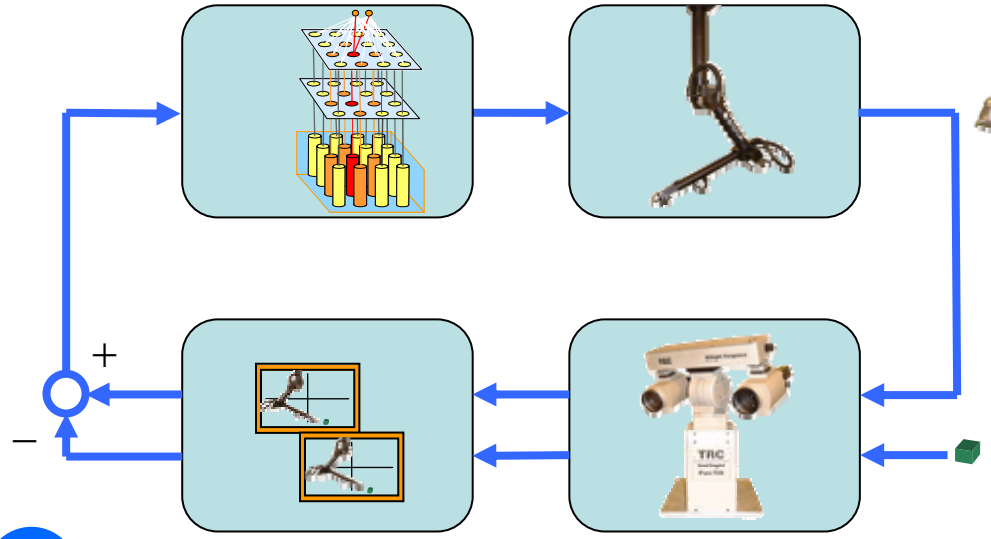


Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire)
avec une carte de Kohonen et des Adalines.

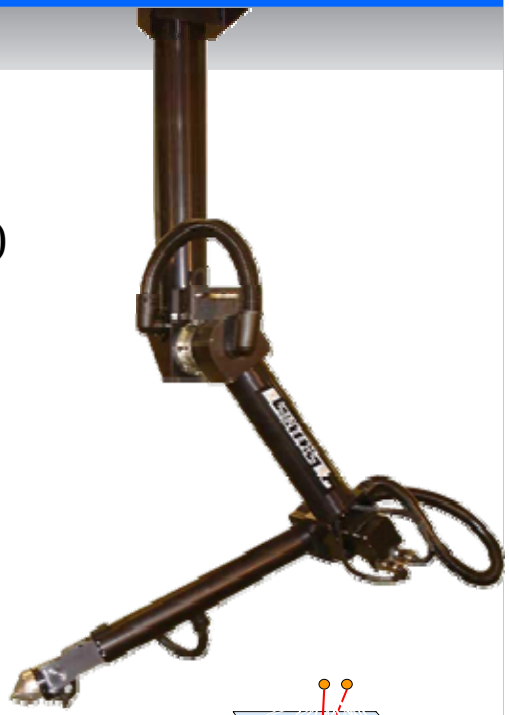


Cette approche a été validée expérimentalement :



Application 3 : carte de Kohonen + Adaline

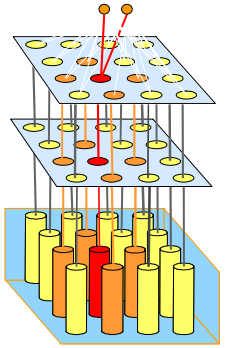
Objectif : estimer une fonction robotique (non linéaire) avec une carte de Kohonen et des Adalines.



Cette approche a été étendue :

- à des robots à 5 axes (volume)
- à des caméras qui s'orientent

⇨ 5+4 = 9 degrés de liberté



SOM-LLM

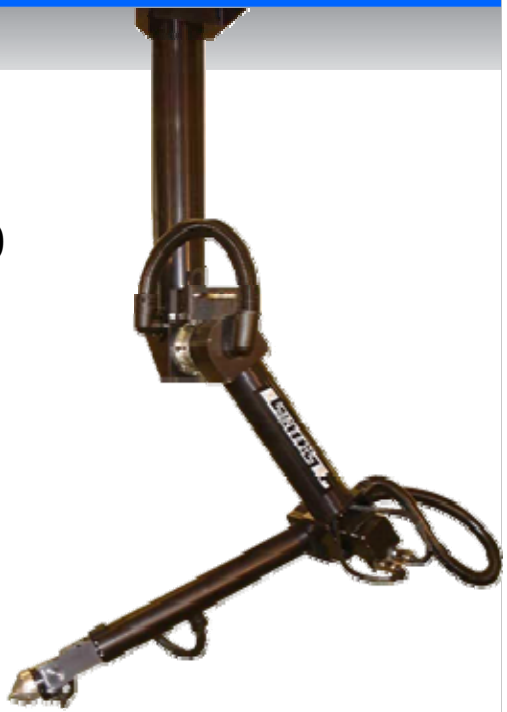
Un réseau unique ne peut pas apprendre cette fonction.

⇨ Solution :

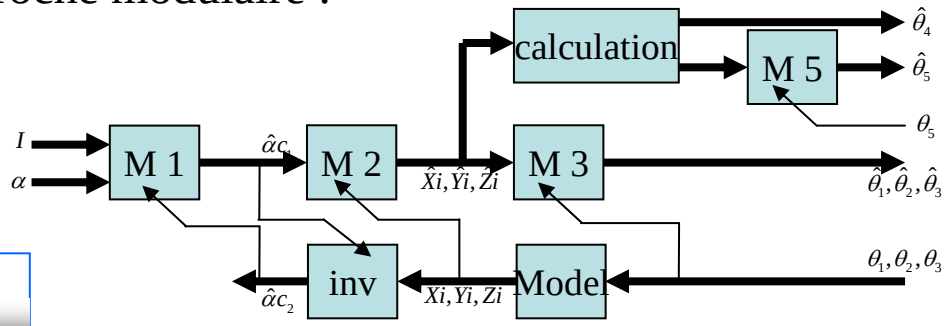
la fonction est découpée en plusieurs fonctions plus simples et chacune est apprise par un réseau de neurone indépendant.

Application 3 : carte de Kohonen + Adaline

Objectif : estimer une fonction robotique (non linéaire) avec une carte de Kohonen et des Adalines.

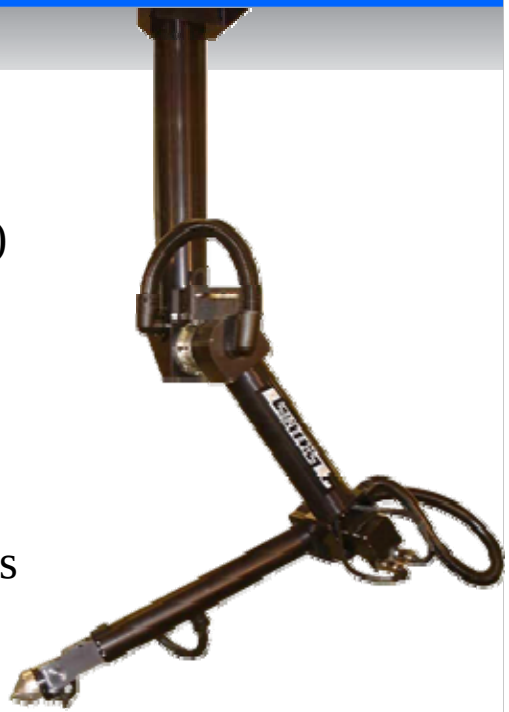


Approche modulaire :



Chaque module (M1, M2, M3 et M5) est un réseau SOM-LLM avec pas plus de 3 dimensions pour les grilles des neurones et pour l'espace des entrées (11×10 ou 11×10×4 neurones).

Application 3 : carte de Kohonen + Adaline

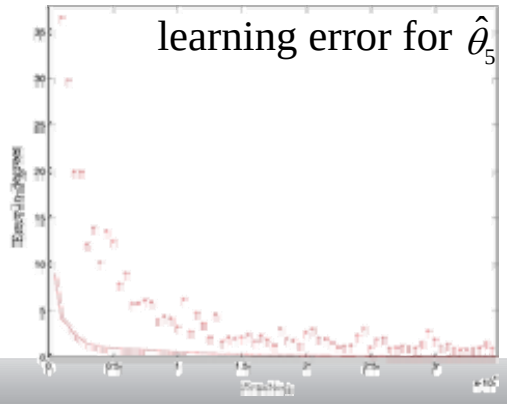


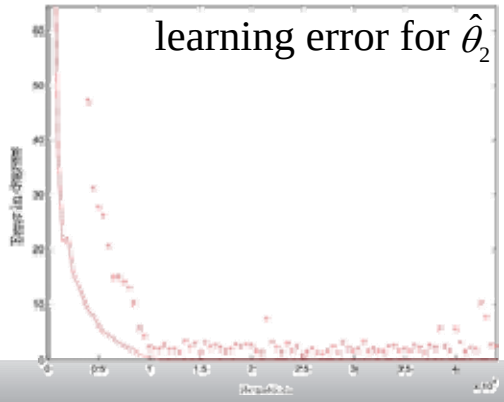
Objectif : estimer une fonction robotique (non linéaire) avec une carte de Kohonen et des Adalines.

Approche modulaire :

- elle découle d'une décomposition théorique (expression mathématique des angles)
- elle permet d'apprendre des problèmes très complexes
- elle aboutit à des erreurs moyennes de 0.15° par axe

⇨ soit une erreur inférieure à 4 mm en position et 1.5° en orientation

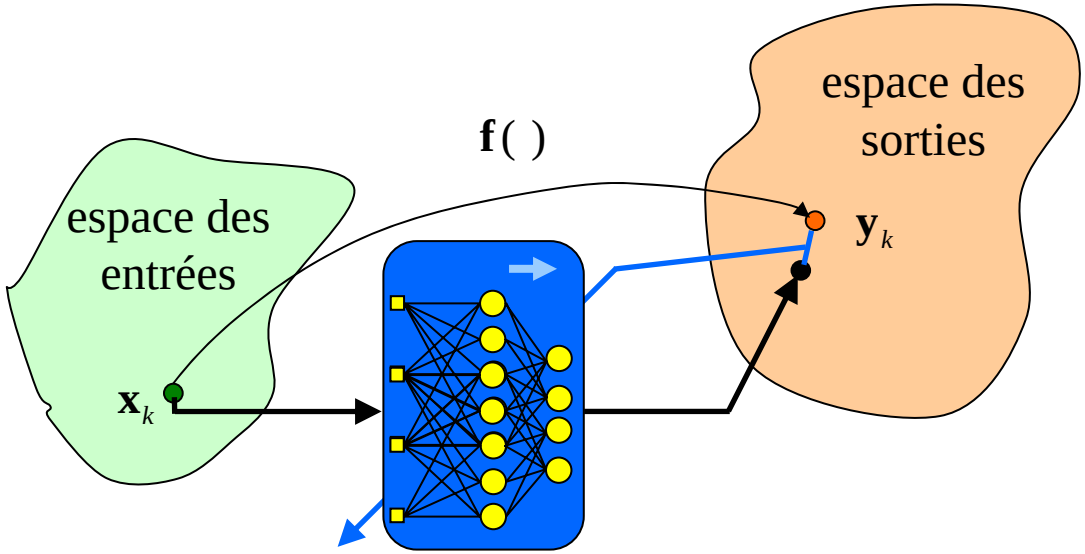




Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

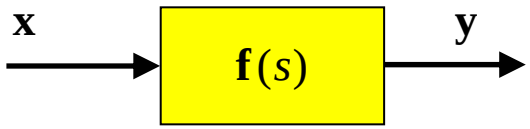
Une fonction de transfert est une fonction…



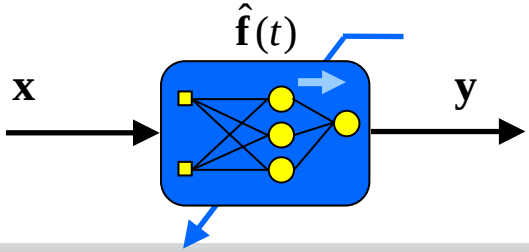
Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Approche fonction de transfert :



Approche neuronale ("black-box modeling") :



Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

En fait, on veut estimer l'onduleur et son filtre de sortie du 3ième ordre.

La fonction de transfert d'un onduleur peut se simplifier à un système du premier ordre avec un retard :

$$\mathbf{f}_{onduleur}(s) = \frac{1 - e^{sT}}{s}$$

T = période d'échantillonnage

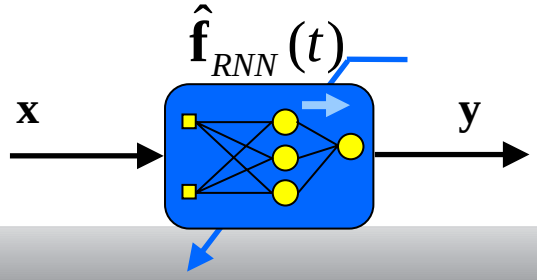
Il faut donc estimer

$$\hat{\mathbf{f}}_{RNN}(t) = \mathbf{f}_{onduleur}(t) + \mathbf{f}_{filtre}(t)$$

$t = kT$

avec

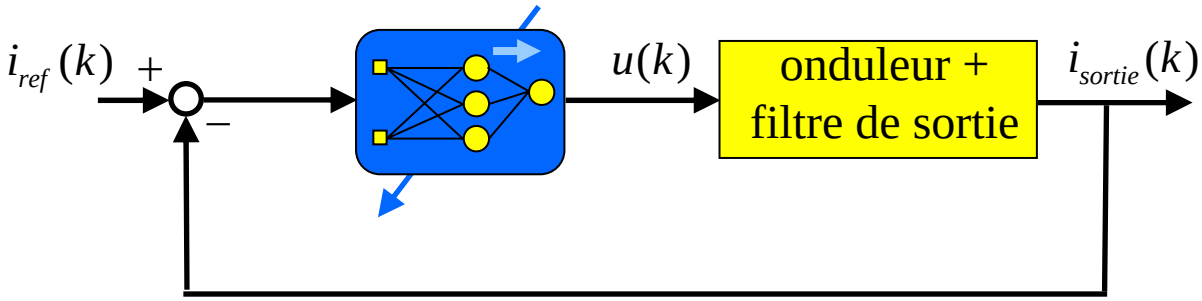
$$\mathbf{f}_{filtre}(s) = \frac{A}{a_3 s^3 + a_2 s^2 + a_1 s + a_0}$$



Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

La tâche à réaliser est la suivante :



Il faut générer un courant $i_{sortie}(k)$ identique au courant de référence $i_{ref}(k)$ grâce au signal $u(k)$ créé par la loi de commande.

Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Il est par exemple nécessaire de commander un onduleur et le filtre de sortie associé dans le schéma d'un filtre actif parallèle (pour compenser les harmoniques présentes dans un réseau de distribution électrique).

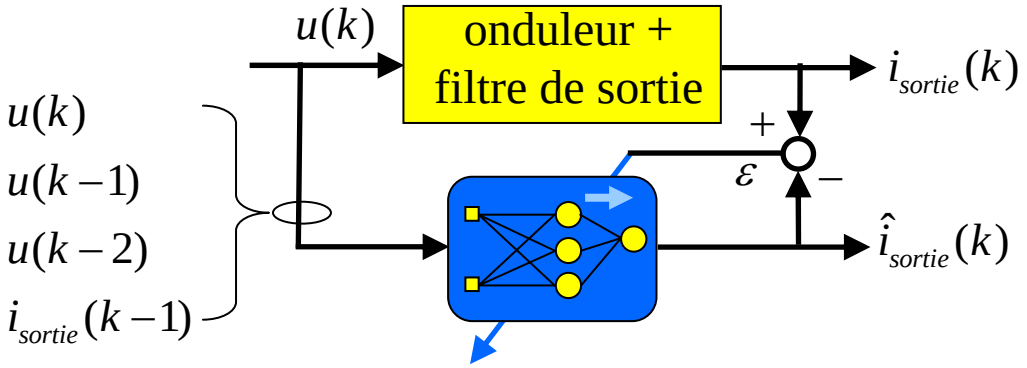
Ils (l'onduleur et le filtre de sortie) introduisent un retard entre le courant de sortie et le courant de référence. Ce retard se traduit par la suite en un déphasage non souhaitable entre le courant et la tension du réseau électrique.

Les contrôleurs généralement utilisés dans les filtres actifs parallèles sont des régulateurs PID ou RST.

Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

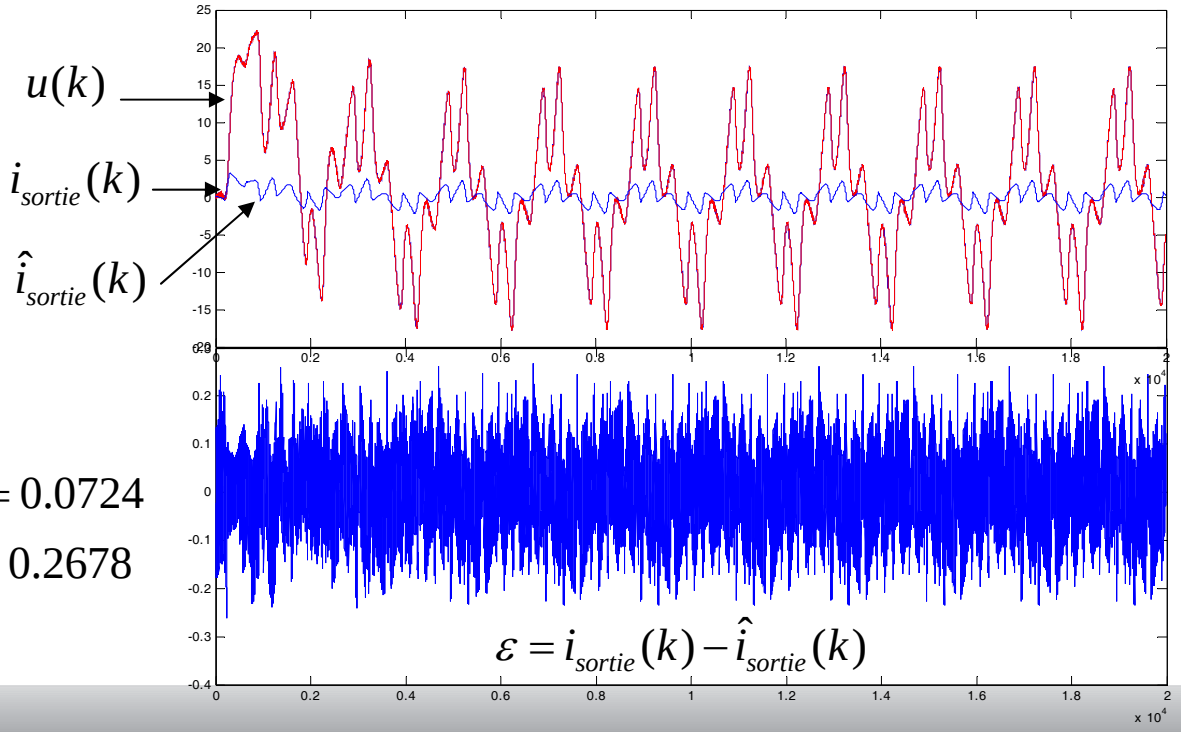
Identification : apprentissage du système direct avec un MLP 4-20-1



Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Identification : apprentissage du système direct avec un MLP 4-20-1



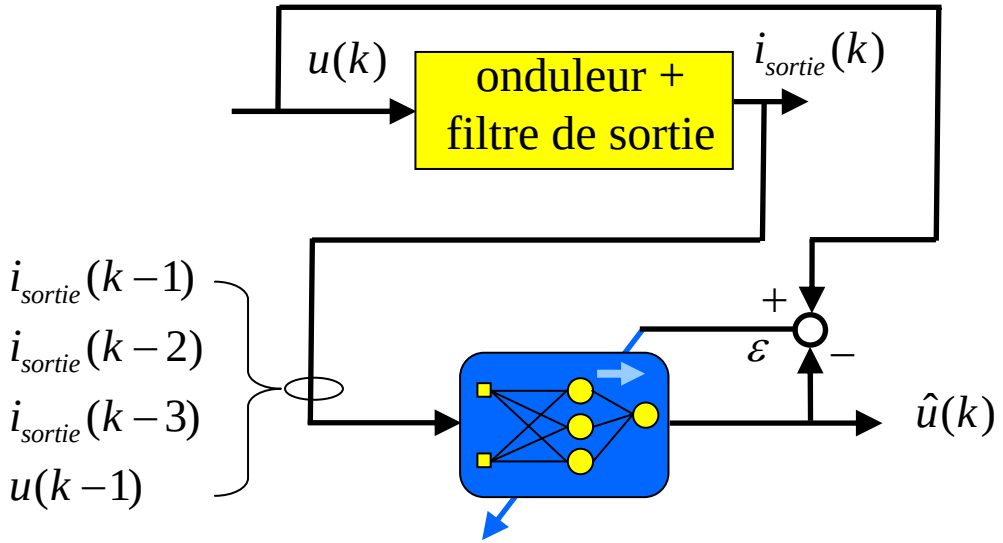
$\varepsilon_{mean} = 0.0724$

$\varepsilon_{max} = 0.2678$

Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

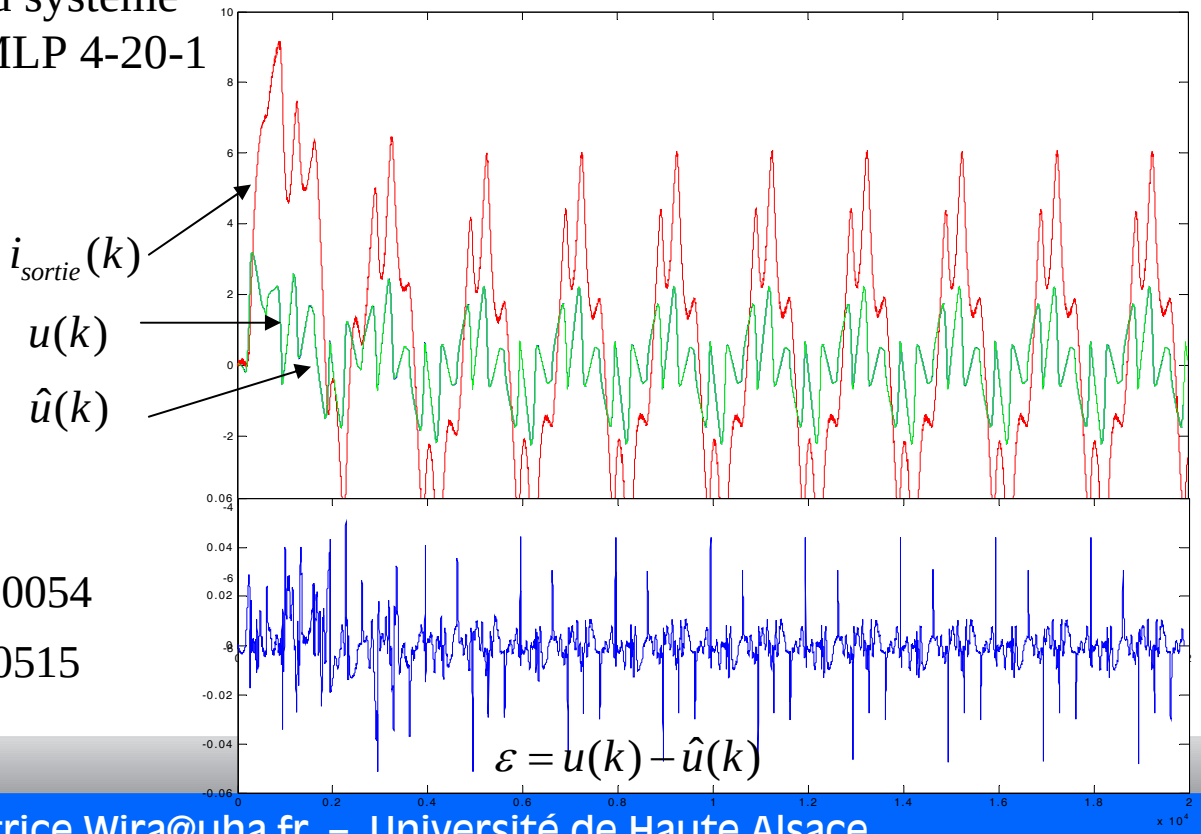
Identification : apprentissage du système inverse avec un MLP 4-20-1



Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Identification : apprentissage du système inverse avec un MLP 4-20-1



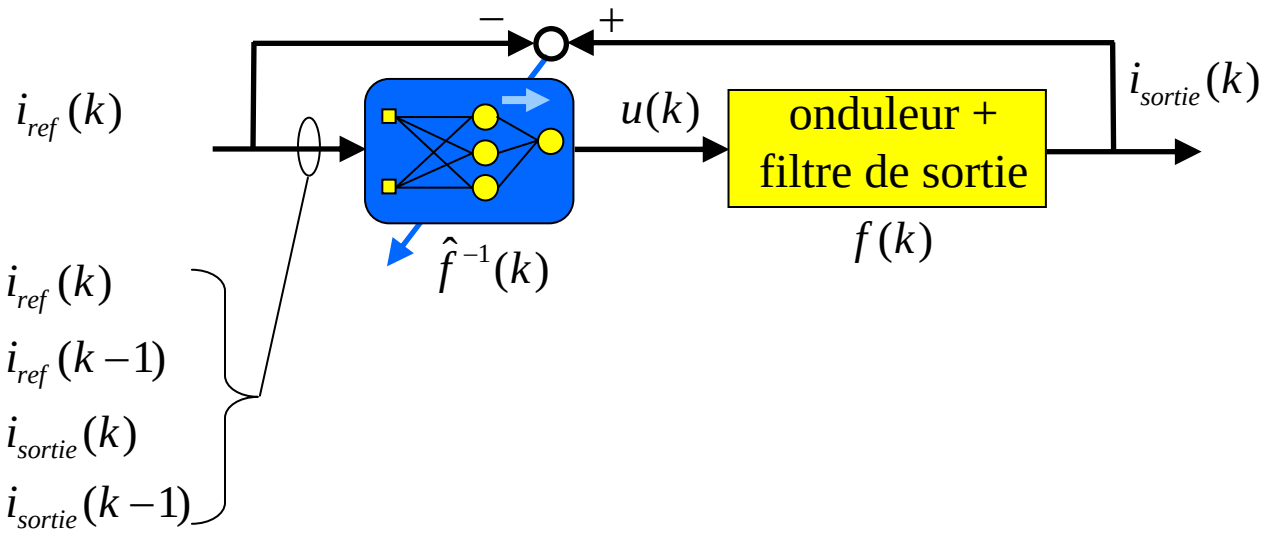
$\varepsilon_{mean} = 0.0054$

$\varepsilon_{max} = 0.0515$

Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Identification : commande directe avec un MLP 4-20-1

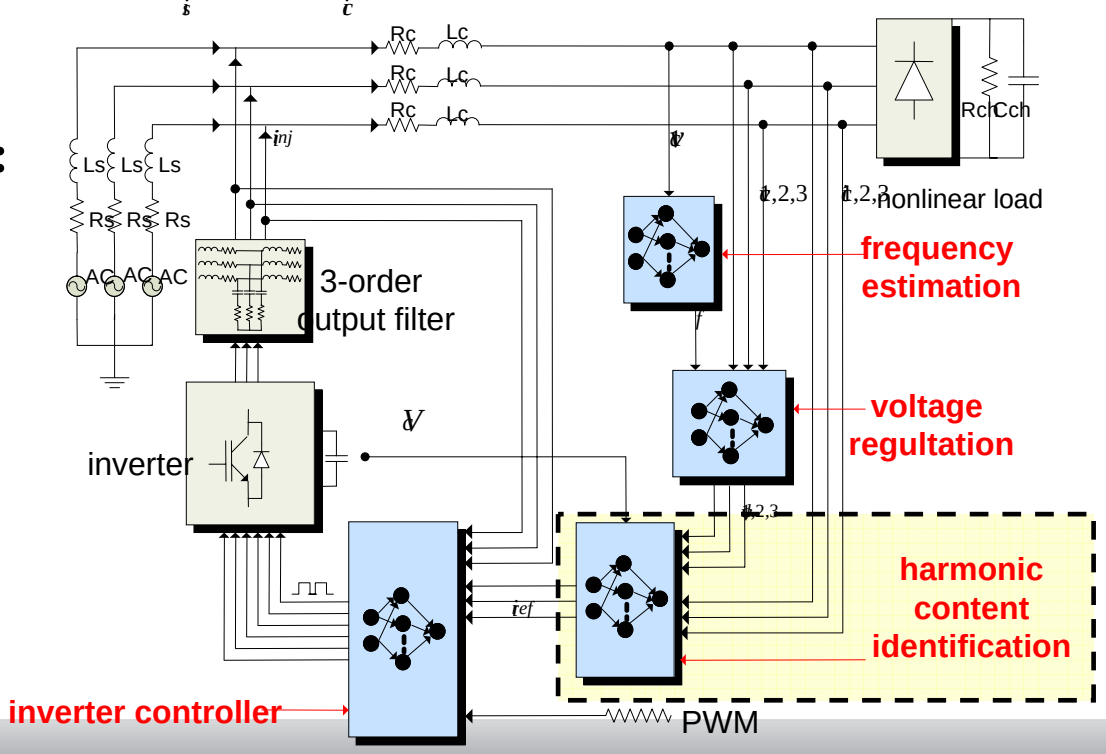


Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Identification : commande directe avec un MLP 4-20-1

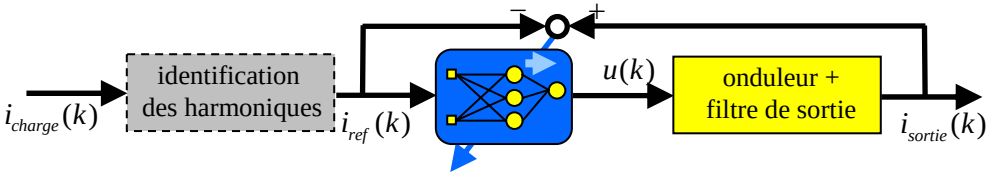
Schéma et principe d'un
filtre actif parallèle :

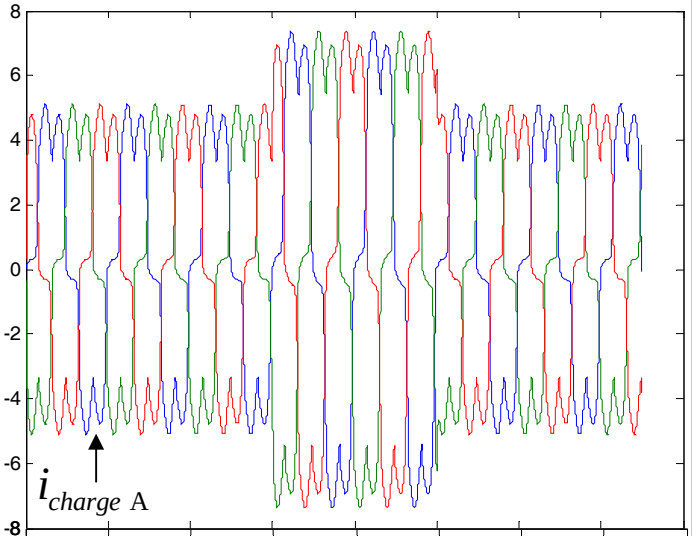


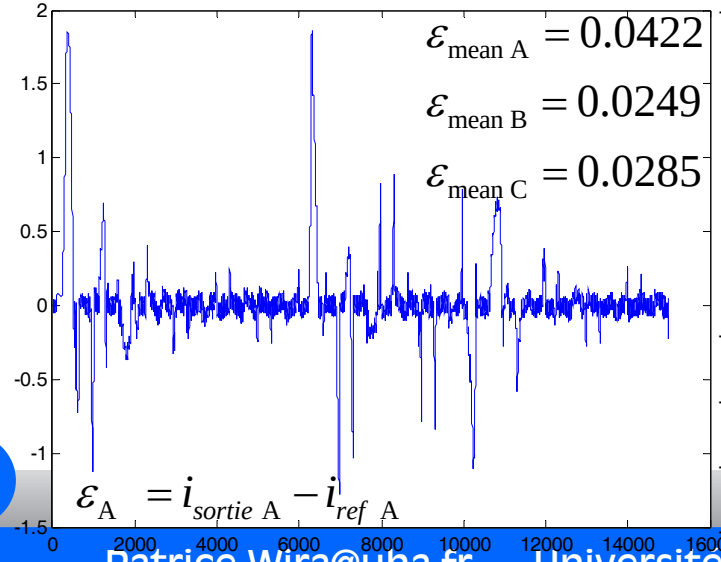
Application 4 : réseau multicouche

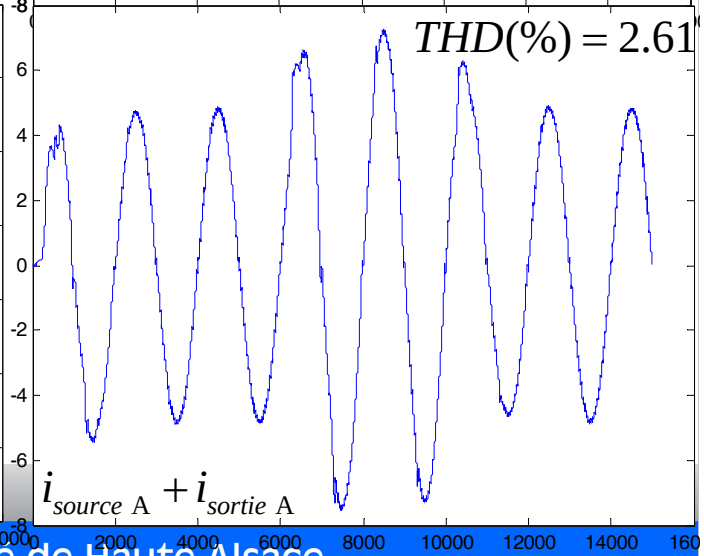
Objectif : identifier la fonction de transfert d'un onduleur.

Identification : commande directe avec un MLP 4-20-1









Application 4 : réseau multicouche

Résultat de simulation avec un nombre différent de neurones dans la couche cachée du MLP.

La charge non linéaire est modifiée 3 fois, THD et erreurs sont donnés après stabilisation (qq ms).

Cette approche a également été validée expérimentalement.

$$THD(\%) = \frac{\sqrt{\sum_{h=2}^{\infty} I_h}}{I_f} \qquad \varepsilon_{\mathrm{A}} = i_{sortie\ \mathrm{A}} - i_{ref\ \mathrm{A}}$$

méthode de commande	THD (%) après compensation	erreurs statiques
sans compensation	*[26.55 24.04 26.50]*	
PID classique	*[1.50 2.44 1.40]*	*[0.1382 0.2813 0.0324]*
Hystérésis	*[1.4 2.83 1.41]*	*[0.0221 0.0159 0.0253]*
PI Neuronale	*[1.68 2.95 1.64]*	*[0.0305 0.0257 0.0302]*
MLP 4:2 :1	*[2.63 3.07 1.74]*	*[0.033 0.0404 0.0370]*
MLP 4 :3 :1	*[2.25 3.04 1.65]*	*[0.0097 0.0486 0.0283]*
MLP 4 :4 :1	*[1.99 3.04 1.67]*	*[0.0214 0.0424 0.0244]*
MLP 4 :5 :1	*[1.66 2.96 1.54]*	*[0.0226 0.0285 0.0256]*
MLP 4 :10:1	*[1.76 2.98 1.64]*	*[0.0351 0.0245 0.0327]*
MLP 4 :15:1	*[1.79 3.02 1.66]*	*[0.0335 0.0208 0.0264]*
MLP 4 :20:1	*[2.61 3.24 1.71]*	*[0.0422 0.0249 0.0285]*
MLP 4 :25:1	*[1.87 2.91 1.75]*	*[0.0303 0.0224 0.0365]*

Application 4 : réseau multicouche

Objectif : identifier la fonction de transfert d'un onduleur.

Identification : apprentissage du système direct avec un MLP 4-20-1

Bilan :

• La commande directe n'est pas la commande la plus performante. On peut s'attendre à de meilleurs résultats avec des commandes plus évoluée (telles que direct-inverse, MRAC).

• Plus il y a des neurones dans un réseau, plus le jeu d'apprentissage doit être grand.

• La commande neuronale est adaptative (au contenu harmonique) et sa réponse est suffisamment rapide.

Réseaux de neurones artificiels : architectures et applications

Conclusion

Conclusion

Ce cours a illustrée de manière modeste le formalisme des réseaux de neurones. Nous avons développé en particulier :

- le réseau Adalines (ADAptive LINear Element),
- la carte auto-organisatrice de Kohonen, (apprentissage non supervisé)
- et le réseau multicouche avec rétropropagation de l'erreur. (apprentissage supervisé)

Conclusion

L'approche utilisée est une approche purement "traitement du signal".

L'objectif est d'étudier et de développer des méthodes d'identification et lois de commandes adaptatives et intelligentes. On parle alors de lois de commandes neuronales et de contrôleurs neuronaux.

Les applications directes qui ont permis d'illustrer nos propos concernaient :

- l'approximation de fonctions, linéaire et non linéaire,
- l'estimation de fonctions robotiques (non linéaires) complexes,
- l'identification de la fonction de transfert d'un système non linéaire.

Conclusion

Perspectives dans l'utilisation des réseaux neuromimétiques
dans les lois de commande intelligentes

Les systèmes de demain seront plus complexes. Il faudra les rendre plus « intelligents ».
Les lois de commande utilisées seront de fait des commandes intelligentes.

Le recensement des derniers progrès technologiques, permet également d'envisager que l'utilisation les réseaux de neurones seront démultipliées dans le futur.

Les investigations en cours recherchent à optimiser les réseaux par rapport aux supports matériels de calcul.

Aujourd'hui, les commandes neuronales sont spécialisées pour une application spécifique. Il faut à l'inverse chercher à les généraliser, en particulier dans le cas des commandes de moteurs.

Bibliographie (1/2)

[1] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4-27, 1990.

[2] B. Widrow and E. Walach, Adaptive Inverse Control. Upper Saddle River: Prentice Hall Press, 1996.

[3] S. Haykin and J. Principe, "Dynamic modeling with neural networks," IEEE Signal Processing Magazine, vol. 15, no. 3, 1998.

[4] S. Haykin, Neural networks : A comprehensive Foundation, 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 1999.

[5] P. Vas, Artificial-intelligence-Based Electrical Machines and Drives: Application of Fuzzy, Neural, Fuzzy-Neural and Genetic-Algorithm-Based Techniques. Oxford: Oxford University Press, 1999.

[6] M. Nørgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen, Neural Networks for Modelling and Control of Dynamic Systems. London: Springer-Verlag, 2000.

[7] B. K. Bose, "Neural Network Applications in Power Electronics and Motor Drives - An Introduction and Perspective," IEEE Transactions on Industrial Electronics, vol. 54, no. 1, pp. 14-33, 2007.

Bibliographie (2/2)

[8] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," Biological Cybernetics, vol. 43, pp. 59-69, 1982.

[9] T. Kohonen, Self-Organizing Maps vol. 30. Berlin: Springer-Verlag, 1995.

[10] T. Kohonen, Content-Addressable Memories, 2nd ed. vol. 1. Berlin: Springer-Verlag, 1987.

[11] G. W. Irwin, K. Warwick, and K. J. Hunt, "Neural Network Applications in Control," in Control Engineering Series. vol. 53 London: The Institution of Electrical Engineers, 1995.

[12] M. N. Cirstea, A. Dinu, M. McCormick, and J. G. Khor, Neural and Fuzzy Logic Control of Drives and Power Systems. Oxford (England): Newnes, 2002.

[13] L. Personnaz and I. Rivals, Réseaux de neurones formels pour la modélisation, la commande et la classification. Paris: CNRS Éditions, 2003.

[14] P. Borne, M. Benrejeb, and J. Haggège, Les réseaux de neurones - Présentation et applications. Paris: Technip, 2007.

[15] G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, and S. Thiria, Apprentissage statistique : Réseaux de neurones - Cartes topologiques - Machines à vecteurs supports. Paris: Eyrolles, 2008.

[16] J. Hérault and C. Jutten, Réseaux neuronaux et traitement du signal. Paris: Hermès, 1994.

Cours en ligne

Merci de votre attention.

Ce cours est disponible en ligne :

http://www.trop.mips.uha.fr/cours-ed/

(à partir du 1er mai 2009)