République Algérienne Démocratique et Populaire
Ministre de l'Enseignement Supérieur et de la Recherche Scientifique

Université Amar Telidji de Laghouat
Faculté de Technologie
Département de l'Électrotechnique

Polycopié de cours

Commande Intelligente

Niveau : 2éme Année Master
Option : Automatique et Systèmes

Présenté par :

Dr. BENMOUIZA Khalil
Maître de Conférences -A-

2022/2023

Sommaire

Partie I : Logique floue (7 semaines)

Partie II : Réseaux de neurones artificiels (RNA) (8 semaines)

Résumé :

Le cours de la commande intelligente est destiné aux étudiants de **Master en Automatique**, option **Automatique et Systèmes**. L'objectif de ce cours est de présenter les méthodes et outils nécessaires à l'intégration de la logique floue et des réseaux de neurones dans les schémas d'identification et de commandes de processus industriels. Il vise à donner une base théorique indispensable à la compréhension de ces approches et à leur utilisation dans les phases d'analyse, de synthèse et de mise en œuvre. Le contenu du cours peut se résumer dans les points suivants :

Partie I : **Logique floue**

- Introduction à la théorie des ensembles flous.
- Raisonnement flou.
- Modélisation floue et systèmes d'inférence floue .
- Commande floue.

Partie II : **Réseaux de neurones artificiels**

- Introduction sur les réseaux de neurones artificiels .
- Modélisations (modèle de Mac Culloch et Pitts, Modélisation générale, Le perceptron.
- Algorithmes/techniques d'apprentissage).
- Réseaux multicouches .
- Application des réseaux de neurones artificiels.

Connaissances préalables recommandées :

L'étudiant devra posséder les connaissances suivantes :

- Systèmes asservis linéaires ;
- Systèmes échantillonnés ;
- Connaissances sur Matlab .

Introduction générale

La commande intelligente est un domaine de l'ingénierie qui vise à concevoir des systèmes de commande automatique capables de s'adapter et de prendre des décisions de manière autonome. Elle utilise souvent des techniques de logique floue et de réseaux de neurones pour traiter les données imprécises et incomplètes qui peuvent être disponibles dans des environnements complexes [1].

La commande intelligente est importante car elle permet de concevoir des systèmes de commande automatique capables de s'adapter et de prendre des décisions de manière autonome dans des environnements complexes et incertains. Cela peut améliorer la performance et la fiabilité de ces systèmes, ainsi que leur capacité à s'adapter à des situations imprévues. Par exemple, dans le domaine de la robotique, la commande intelligente peut permettre à un robot de s'adapter à son environnement et de réaliser des tâches de manière autonome, sans intervention humaine. Dans le domaine de la conduite autonome, elle peut permettre à un véhicule de prendre des décisions en temps réel pour éviter les accidents et améliorer la sécurité routière. Dans le domaine du contrôle des processus industriels, elle peut permettre d'optimiser la performance et l'efficacité de ces processus en prenant en compte des données en temps réel et en ajustant les paramètres de commande en conséquence. En général, la commande intelligente peut apporter de nombreux bénéfices en termes de performance, de fiabilité et de flexibilité dans de nombreux domaines où elle est utilisée[1-2].

La logique floue est un formalisme de raisonnement qui permet de traiter des situations imprécises ou incertaines. Elle a été développée dans les années 1950 par le mathématicien Lotfi Zadeh [3] pour mieux modéliser les phénomènes complexes de la vie réelle. Elle est basée sur l'utilisation de variables floues, qui prennent des valeurs comprises entre 0 et 1, plutôt que des valeurs binaires (vrai/faux) comme dans la logique classique. Ainsi, la logique floue peut exprimer des degrés de vérité, par exemple, un objet peut être à la fois "grand" et "petit", mais dans des degrés différents [4].

Les réseaux de neurones sont des systèmes informatiques inspirés du fonctionnement du cerveau humain. Ils sont constitués de nombreux noeuds ou "neurones" reliés entre eux par des arcs ou "synapses", qui peuvent être activés ou désactivés. Un réseau de neurones peut apprendre à partir de données en ajustant ces poids de synapses de manière à minimiser l'erreur entre la sortie attendue et la sortie obtenue [5].

Semestre: 3
Unité d'enseignement: UEF 2.1.1
Matière: Commande intelligente
VHS: 45h00 (Cours: 1h30, TD: 1h30)
Crédits: 4
Coefficient: 2

Partie I : Logique Floue

I.1 Introduction

La logique floue est une extension de la logique booléenne basée sur la théorie mathématique des ensembles flous, qui est une généralisation de la théorie classique des ensembles. En introduisant la notion de degré dans la vérification d'une condition, permettant ainsi à une condition d'être dans un état autre que vrai ou faux, la logique floue apporte une souplesse de raisonnement très appréciable, qui permet de prendre en compte les imprécisions et les incertitudes. Cette technique est robuste où aucune entrée précise n'est requise. Les systèmes flous peuvent accepter plusieurs types d'entrées, y compris des données vagues, déformées ou imprécises.

I.2 Historique de la logique floue

Le terme logique floue a été introduit avec la proposition de théorie des ensembles flous de 1965 par le mathématicien azerbaïdjanais iranien Lotfi Zadeh. La logique floue avait cependant été étudiée depuis les années 1920, en tant que logique à valeurs infinies, notamment par Łukasiewicz et Tarski.

En 1973, Lotfi Zadeh à introduit la notion de variable linguistique, après en 1974 Mamdani a réalisé un contrôleur flou à moteur à vapeur. Dans les années 80, c'était l'explosion de la logique floue au Japon qui atteint son progrès en 1990. Aujourd'hui, il existe une variété des produits flous dans notre vie[6-8].

I.3 Importance de la logique floue

- Il résout le problème de l'incertitude dans le domaine de l'ingénierie.
- Lorsqu'un raisonnement précis n'est pas disponible, il fournit un niveau de raisonnement précis.

- La logique floue a une structure simple et facile à comprendre.
- C'est un moyen efficace de contrôler les machines.
- Il apporte des solutions à divers problèmes industriels (en particulier la prise de décision).
- Il nécessite peu de données pour être exécuté.

I.4 Comparaison entre la logique floue et la logique booléenne (classique)

La théorie classique des ensembles désigne simplement la branche des mathématiques qui étudie les ensembles. Par exemple, 5, 10, 7, 6, 9 sont un ensemble d'entiers. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 est l'ensemble des nombres entiers entre 0 et 10. 's' 'd'; 'z', 'a' est un ensemble de caractères. "Site", "de", "zéro" est un ensemble de mots. On peut aussi créer des ensembles de fonctions, des hypothèses, des définitions, des ensembles d'individus (c'est-à-dire une population), etc. Et même des ensembles d'ensembles. A noter que dans un ensemble, l'ordre n'a pas d'importance : 7, 6, 9 désignent le même ensemble que 9, 7, 6. Cependant, pour améliorer la lisibilité, il convient de classer les éléments par ordre croissant, soit 6, 7, 9. Les ensembles sont souvent représentés sous forme graphique, généralement par des cercles, comme l'illustre la figure I.1.

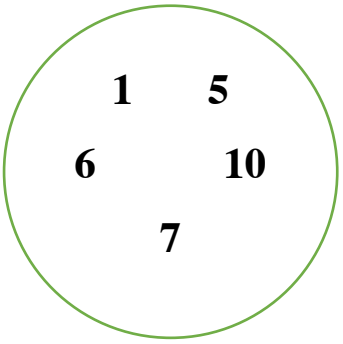


Figure I.1 Représentation graphique de l'ensemble [1, 5, 6, 7, 10].

Le concept d'appartenance est important en théorie des ensembles : il fait référence au fait qu'un élément fasse partie ou non d'un ensemble. Par exemple, l'entier 7 appartient à l'ensemble 6, 7, 9. En revanche, l'entier 5 n'appartient pas à l'ensemble 6, 7, 9. L'appartenance est symbolisée par le caractère dans la non-appartenance et par le même symbole, mais barré possible. Ainsi, nous avons $7 \in \{6, 7, 9\}$ et $5 \notin \{ / 6, 7, 9\}$.

Une fonction d'appartenance (aussi appelée fonction indicatrice ou fonction caractéristique) est une fonction qui explicite l'appartenance ou non à un ensemble E. Cette notion d'appartenance est très importante pour ce cours, car la logique floue est basée sur la notion d'appartenance floue. Cela signifie simplement que nous pouvons appartenir à un ensemble à 0,8, contrairement à la théorie des ensembles classique où comme nous venons de le voir, l'appartenance est soit 0 (non possédé), soit 1 (partie).

La théorie classique ne peut pas résoudre certains paradoxes tel que le problème de classification, alors en la logique floue un objet peut appartenir à un ensemble et en même temps à son complément avec un certain degré « d'appartenance ».

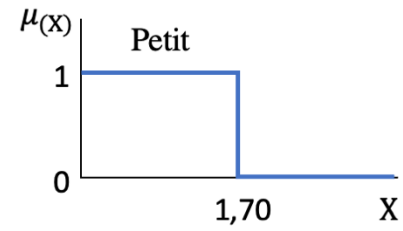
La logique floue est basée sur la théorie des ensembles flous, qui est une généralisation de la théorie classique des ensembles. Dire que la théorie des ensembles flous est une généralisation de la théorie des ensembles classique signifie que cette dernière est un cas particulier de la théorie des ensembles flous.

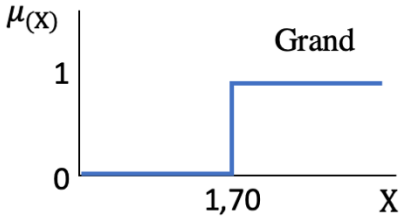
I.5 Théorie des sous-ensembles flous

La logique floue est basée sur la théorie des ensembles flous, qui est une généralisation de la théorie classique des ensembles. Par abus de langage, suivant les habitudes de la littérature, nous utiliserons les termes ensembles flous au lieu de sous-ensembles flous. Les ensembles classiques sont également appelés ensemble clairs, par opposition à vagues, et du même coup la logique classique est également connue sous le nom de logique booléenne ou binaire [7,9].

Example 1 : soit une variable x (la taille) et un univers de discours U (les individus), un ensemble floue A est définie par une fonction d'appartenance $\mu_{(x)}$ qui décrit le degré avec lequel l'élément x appartient à A. Voici un petit exemple d'explication :

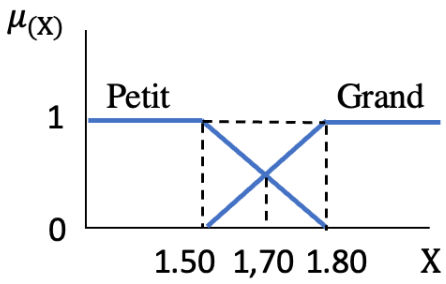
A- La logique classique





$$\mu_{(x)} = \begin{cases} 1, si\ x \in 1 \\ 0, si\ non \end{cases}$$

B- La logique floue



$\mu_{(x)} \in [0,1]$

On parle de variable linguistique « La taille » et de valeur linguistique « Petit et Grand » celle-ci associé avec la fonction d'appartenance $\mu_{(x)}$ constitue des ensembles flous.

I.5.1 Les fonctions d'appartenances

La fonction d'appartenance peut prendre différentes formes arbitraires, mais il est raisonnable de prendre une fonction convexe qui doit avoir au moins un point de degré d'appartenance maximal et que le degré décroît quand on s'éloigne de ce point. Les fonctions d'appartenances les plus utilisées sont sur la Figure I.2 [3,7-9].

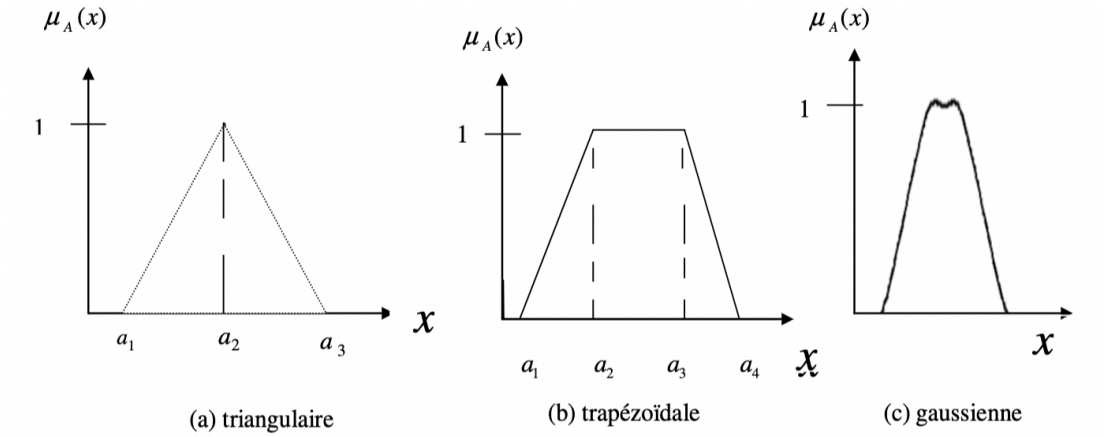


Figure I.2 Les fonctions d'appartenances.

Une fonction d'appartenance est constituée de trois parties :

- Une zone telle que $\mu_A(x) = 0$ → non-appartenance.
- Une zone telle que $\mu_A(x) = 1$ → appartenance totale.
- Une zone $\mu_A(x) \in [0,1]$ → appartenance partielle.

Son rôle est de préciser numériquement sur un support donné, la signification d'un ensemble flou, énoncée sous forme linguistique.

I.5.2 Les opérateurs de la logique floue

Il s'agit d'une généralisation des opérateurs négation (complément), intersection et union de la théorie classique des ensembles. Ces relations sont traduites par les opérateurs « NON », « OU » et « ET» [3].

- L'opérateur NON :

C'est défini par : $\bar{A} = \{x;\ x \notin A\}$.

En logique floue en peut réaliser cet opérateur par : $NON\big(\mu_A(x)\big) = 1 - \mu_A(x)$.

- L'opérateur ET :

C'est défini par : $A \cap B = \{x;\ x \in A\ \ \boldsymbol{et}\ \ x \in B\}$.

En logique floue en peut réaliser cet opérateur par :

- La fonction min : $\mu_{A\cap B}(x) = \min\big(\mu_A(x), \mu_B(x)\big)$.
- La fonction arithmétique produit : $\mu_{A\cap B}(x) = \mu_A(x).\mu_B(x)$.

- L'opérateur OU :

C'est défini par : $A \cup B = \{x;\ x \in A\ \ \boldsymbol{ou}\ x \in B\}$.

En logique floue en peut réaliser cet opérateur par :

- La fonction max : $\mu_{A\cap B}(x) = \max\big(\mu_A(x), \mu_B(x)\big)$.
- La fonction arithmétique somme : $\mu_{A\cap B}(x) = \mu_A(x) + \mu_B(x)$.

Note : dans la littérature spécialisée, plusieurs méthodes de calculs des opérateurs de la logique floue ont été proposées.

I.5.3 Les implications floues

L'implication floue est une déclaration de la forme suivante [7-9] :

SI x est A **ALORS** y est B

où x et y sont des variables linguistiques (variable floue), et A et B sont des valeurs linguistiques, déterminées par les ensembles flous sur les ensembles X et Y.

Exemple 1 : la tension est haute.

La variable linguistique tension prend la valeur linguistique *élevée*. La plage de valeurs linguistiques possibles d'une règle représente l'univers de cette variable.

Exemple 2 : SI vitesse est lente ALORS arrêt est court

La variable vitesse peut avoir une plage de valeurs entre 0 et 220 km/h. On peut inclure des sous-ensembles flous (très lent, lent, moyenne, rapide, très rapide) pour modifier cette règle. Chaque sous-ensemble flou représente une valeur linguistique pour la variable.

La logique classique (SI – ALORS) utilise la logique binaire. La logique floue permet d'associer une plage de valeurs (un ensemble flou) à des variables linguistiques. On peut réduire le nombre de règles jusqu'à 90% en utilisant la logique floue.

I.6 Réglage par la logique floue

Le réglage flou a une structure identique à un système muni d'un réglage classique avec feedback. La figure I.3 schématise la structure d'un réglage par logique floue [8] .

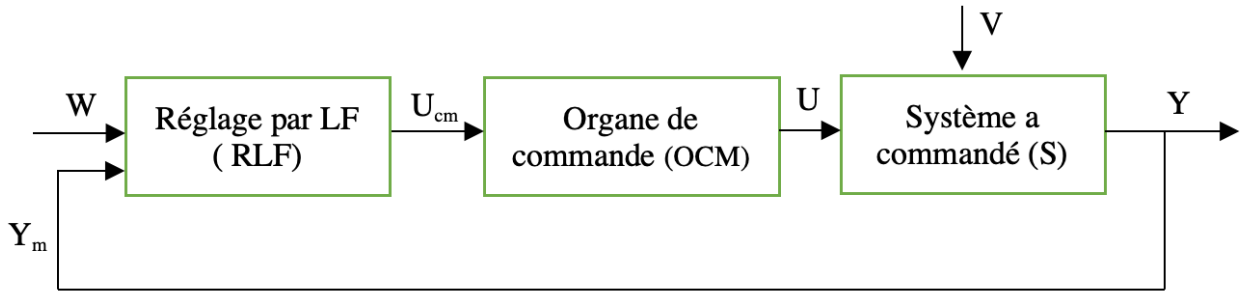


Figure I.3 Structure d'un réglage par LF.

Avec :

W : la grandeur de consigne.

U_{cm} : le signal de commande fourni par le régulateur flou (RLF).

U : le grandeur de commande fourni par l' OCM.

V : la perturbation.

Y : la sortie.

Y_m : le vecteur a réglé.

La configuration interne d'un régulateur flou est donnée par la figure I.4.

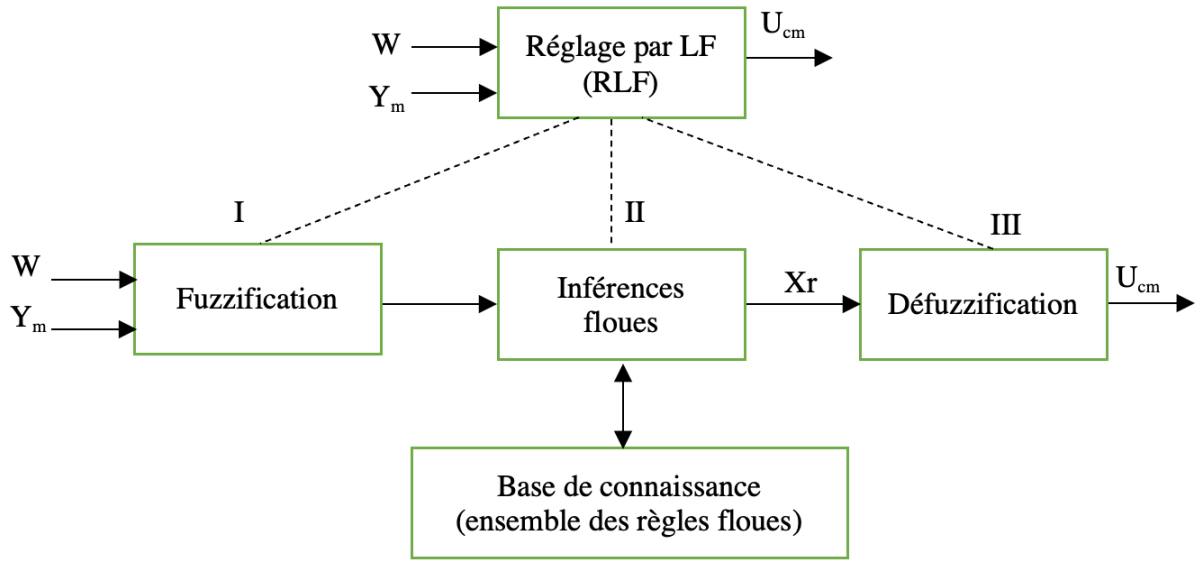


Figure I.4 Régulateur flou.

I.6.1 Fonctionnement d'un RLF

A) Fuzzification

La fuzzification est le processus de conversion d'une valeur d'entrée (grandeur physique) nette en une valeur floue (variable linguistique) qui est effectuée l'identification des fonctions d'appartenances de toutes les variables d'entrées. Bien que divers types de courbes puissent être vus dans la littérature, les formes gaussiens, triangulaires et trapézoïdaux sont les plus couramment utilisés dans le processus de fuzzification. Bien qu'il n'existe pas des règles précises sur ce choix, on prend l'exemple de fuzzification suivant (Figure I.5) [3,7-9] :

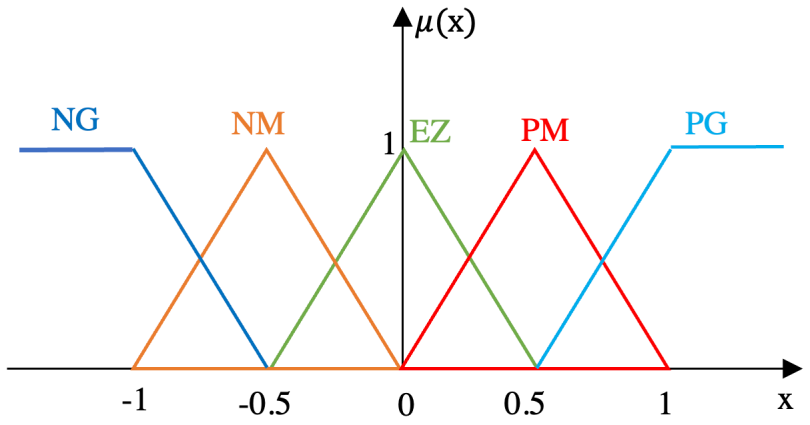


Figure I.5 Ensemble pour 5 règles floues.

Avec, x est une variable linguistique d'entrée (normalisé sur un univers de discours [-1,1]) à laquelle en fait correspondre 5 sous-ensembles flous caractérisés par des fonctions d'appartenances (variables linguistiques) comme suit :

- NG : négatif grand.
- NM : négatif moyen.
- EZ : environ zéro.
- PM : positif moyen.
- PG : positif grand.

Expriment maintenant ces fonctions d'appartenances, on trouve :

$$\mu_{NG}(x) = \begin{cases} 1 & si \quad x \leq -1 \\ -2x - 1 & si \quad -1 < x \leq -0.5 \\ 0 & , \quad ailleurs \end{cases}$$

$$\mu_{NM}(x) = \begin{cases} 2x + 2 & si \quad -1 < x \leq -0.5 \\ -2x & si \quad -0.5 < x \leq 0 \\ 0 & , \quad ailleurs \end{cases}$$

$$\mu_{EZ}(x) = \begin{cases} 2x + 1 & si \quad -0.5 < x \leq 0 \\ -2x + 1 & si \quad 0 < x \leq 0.5 \\ 0 & , \quad ailleurs \end{cases}$$

$$\mu_{PM}(x) = \begin{cases} 2x & si \quad 0 < x \leq 0.5 \\ -2x + 2 & si \quad 0.5 < x \leq 1 \\ 0 & , \quad ailleurs \end{cases}$$

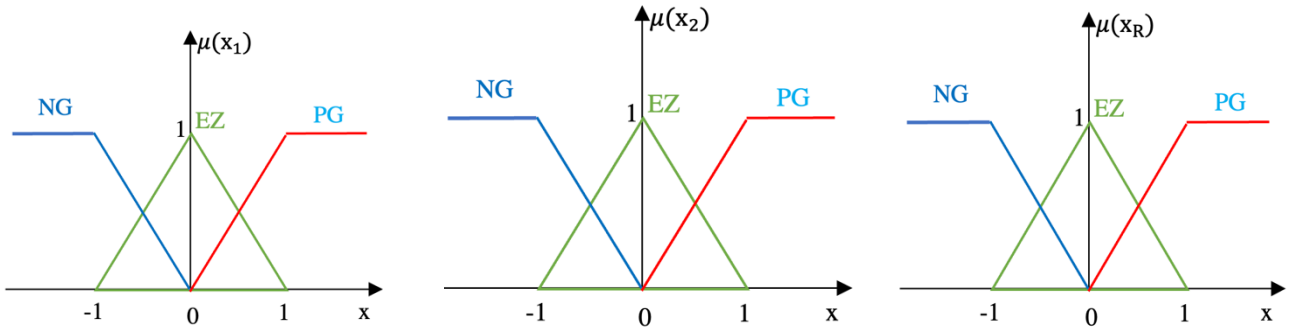
$$\mu_{PG}(x) = \begin{cases} 2x - 1 & si \quad 0.5 < x \leq 1 \\ 1 & si \quad x > 1 \\ 0 & , \quad ailleurs \end{cases}$$

B) Inférence floue et base de règles

L'inférence floue permit d'établir une relation « vague ou floue » qui reliait entre les variables d'entrées et la variable de sortie. Les règles floues sont données par un expert en sous basant de ses connaissances du problème.

Remarque : les règles doivent être exprimées linguistiquement et il est obligé de compléter toutes les règles. Il existe plusieurs méthodes de réalisation de l'étape de l'inférence floue. Pour mieux comprendre cela, on va voir l'exemple suivant.

Supposons qu'on a 2 entrées x1 et x2 et une sortie X_R, toutes fuzzifiées par les 3 sous-ensembles flous suivant :



Les fonctions d'appartenances sont exprimées par :

$$\mu_{NG}(x) = \begin{cases} 1 & si \quad x \leq -1 \\ -x & si \quad -1 < x \leq -0.5 \\ 0 & , \quad ailleurs \end{cases}$$

$$\mu_{EZ}(x) = \begin{cases} x+1 & si \quad -1 < x \leq 0 \\ -x+1 & si \quad 0 < x \leq 1 \\ 0 & , \quad ailleurs \end{cases}$$

$$\mu_{PG}(x) = \begin{cases} 1 & si \quad x > 1 \\ x & si \quad 0 \leq x \leq 1 \\ 0 & , \quad ailleurs \end{cases}$$

Supposons aussi que l'inférence comprend les 2 règles suivantes :

SI x1 est PG **ET** x2 est EZ **ALORS** X_R est EZ
OU
SI x1 est EZ **OU** x2 est NG **ALORS** X_R est NG

Prenons pour le calcul x1 = 0.44 et x2 = - 0.67. Ce qui donne :

$\mu_{PG}(x1 = 0.44) = 0.44$ et $\mu_{EZ}(x2 = -0.67) = 0.33$

$\mu_{EZ}(x1 = 0.44) = 0.56$ et $\mu_{NG}(x2 = -0.67) = 0.67$

Il faut maintenant traduire les opérateurs ET, OU et ALORS par l'un des méthodes suivantes.

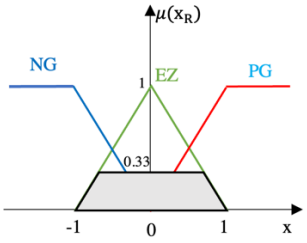
B1) Méthode d'inférence Min-Max

Dans ce cas et au niveau de la condition, les opérateurs flous ET et OU sont réalisé par les fonctions minimum (min) et maximum (max) ; respectivement. D'autre part, et au niveau de la conclusion on réalise les opérateurs OU et ALORS par les fonctions Max et Min d'où les désignations de la méthode.

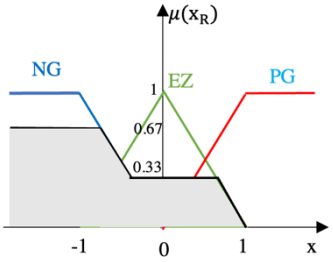
Méthode : Min-Max
ET : min
OU : max
ALORS : min

Traitons l'exemple précèdent :

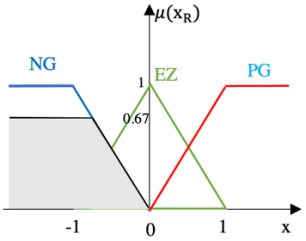
SI x1 est PG	**ET**	x2 est EZ	ALORS	XR est EZ
0.44	Min	0.33	Min	0.33



OU (max)



SI x1 est PG	**OU**	x2 est EZ	ALORS	XR est EZ
0.56	Max	0.67	Min	0.67



La surface en gris représente la sortie floue du RLF issue de l'ensemble des règles.

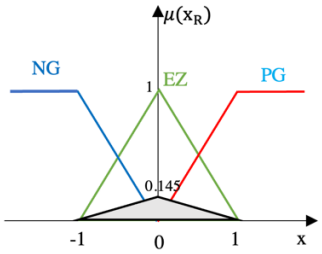
B2) Méthode d'inférence Somme-Produit

Dans ce cas et au niveau de la condition, les opérateurs flous ET et OU sont réalisé par les fonctions minimum (Produit) et maximum (somme (moyenne arithmétique)) ; respectivement. D'autre part, et au niveau de la conclusion on réalise les opérateurs OU et ALORS par les fonctions somme et produit d'où les désignations de la méthode.

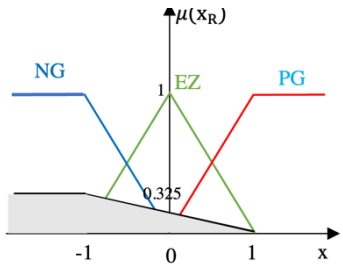
Méthode : Somme-Produit
ET : produit
OU : somme arithmétique
ALORS : produit

Traitons l'exemple précèdent :

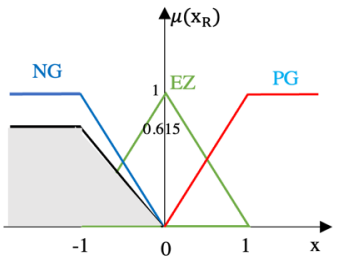
SI x1 est PG	**ET**	x2 est EZ	ALORS	XR est EZ
0.44	Produit	0.33	Produit	0.145



OU (somme)



SI x1 est PG	**OU**	x2 est EZ	ALORS	XR est EZ
0.56	Somme	0.67	Produit	0.615



B3) Méthode d'inférence Max-Produit

Dans ce cas et au niveau de la condition, les opérateurs flous ET et OU sont réalisé par les fonctions minimum (Min) et maximum (Max) ; respectivement. D'autre part, et au niveau de la conclusion on réalise les opérateurs OU et ALORS par les fonctions Max et produit d'où les désignations de la méthode.

Méthode : Max-Produit

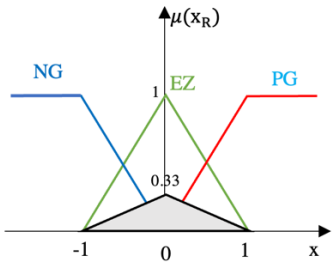
ET : min

OU : Max

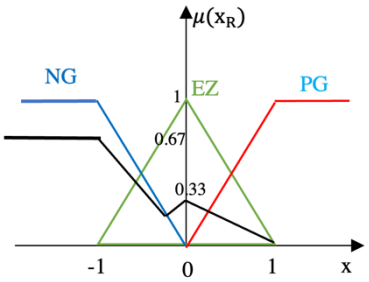
ALORS : produit

Traitons l'exemple précèdent :

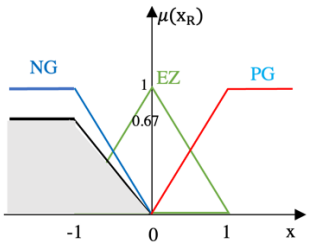
SI x1 est PG	**ET**	x2 est EZ	ALORS	XR est EZ
0.44	Min	0.33	Produit	0.33



OU (somme)



SI x1 est PG	**OU**	x2 est EZ	ALORS	XR est EZ
0.56	Somme	0.67	Produit	0.67

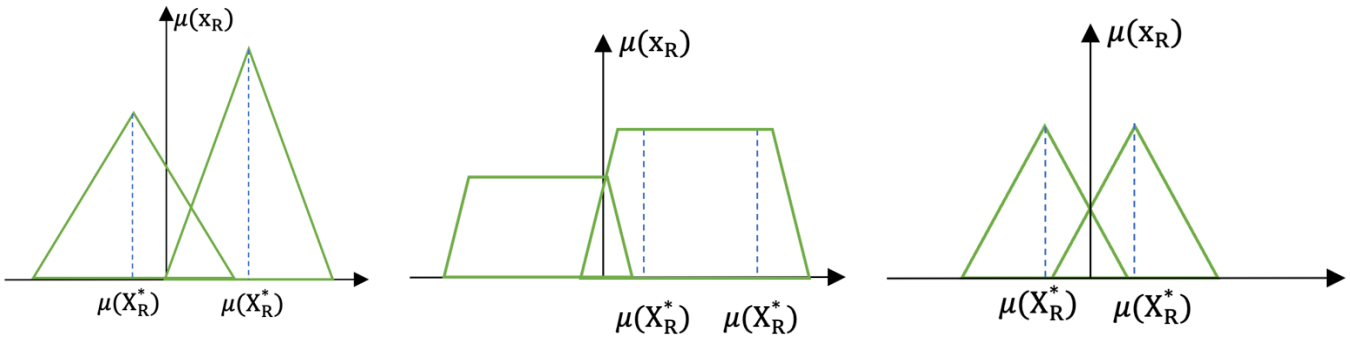


C) Défuzzification

Les méthodes d'inférences fournissent une fonction d'appartenance résultante pour la variable de la sortie, il s'agit donc d'une information floue qu'il faut transformer à une grandeur physique. Il existe plusieurs méthodes de défuzzification, parmi eux [3,7-9] :

C1) Méthode de maximum

Dans ce cas la sortie correspond à l'abscisse du maximum de la fonction d'appartenance résultante. On peut écrire $S = X_R^*$ tq. $\max\left(\mu_{res}(X_R)\right) = \mu_{res}(X_R^*)$, Où S : est la sortie de RLF. Trois cas peuvent se produire comme suit :



On conclusions ; cette méthode est simple, rapide et facile, mais elle introduit des ambiguïtés et une discontinué du sorite.

C2) Méthode du moyen des maximums

Dans le cas où plusieurs sous-ensembles flous auraient la même hauteur maximale on réalise leur moyenne arithmétique, par conséquent en écart la confusion de la méthode des maximums. Dans ce cas, on peut écrire :

$$S = X_R^* = \frac{\int_{U'} X_R \mu_{res}(X_R) dX_R}{\int_{U'} \mu_{res}(X_R) dX_R} \qquad \text{Eq. (I.1)}$$

Où : $U' \subset U : X_R \in U'$ $si \mu_{res}(X_R) = \mu_{max}\left(\mu_{res}(X_R)\right)$

C3) Méthode centroïde (centre de gravité)

La sortie correspond à l'abscisse du centre de gravité de la surface de la fonction résultante à la sortie du bloque d'inférence. Il existe deux manières d'évaluer cette abscisse :

- On prend l'union des sous-ensembles flous des sorties et on lire le centroïde globale.
- On prend chaque sous-ensemble flou séparément et en calcul son centroïde, puis on réalise la moyenne de tous les centroïdes.

La sortie peut être calculée par :

$$X_R^* = \frac{\int_{-1}^{1} X_R \, f(X_R) dX_R}{\int_{-1}^{1} f(X_R) dX_R} \qquad \text{Eq.(I.2)}$$

Ou

$$X_R^* = \frac{\sum_{i=1}^{n_r} \mu_{c_i} X_{G_i} S_i}{\sum_{i=1}^{n_r} \mu_{c_i} S_i} \qquad \text{Eq.(I.3)}$$

Avec :

X_{G_i} : l'abscisse du centre de gravité du sous-ensemble flou de la sorite correspondante à la i^{éme} règle.

S_i : est la surface du sous-ensemble flou de la sorite correspondante à la i^{éme} règle.

n_r : est le nombre des règles.

C4) Méthode de la somme pondérée

C'est un compromis entre les 2 méthodes précédentes. Calculant individuellement les sorties relatives à chaque règle selon le principe de la moyenne du maximum, puit on réalise leur moyenne pondérée. La sortie est calculée par :

$$X_R^* = \frac{\sum_{i=1}^{n_r} \mu_{c_i} X_{R_i}}{\sum_{i=1}^{n_r} \mu_{c_{i_{Max}}}} \qquad \text{Eq.(I.4)}$$

$\mu_{c_{i_{Max}}}$: le maximum de la fonction de sortie de la règle i et X_{R_i} est son abscisse.

I.7 Régulateur flou (PD, PI, PID)

Le schéma synaptique permettant un réglage par la logique floue est donné par la figure I.6.

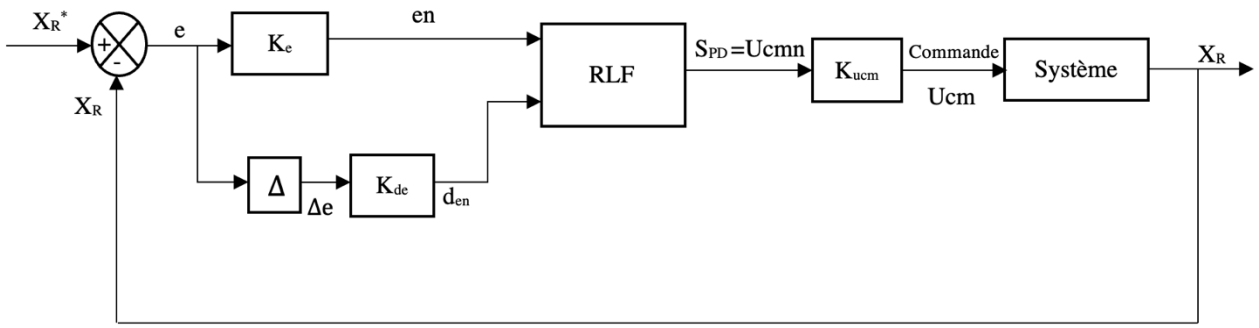


Figure 1.6 Régulateur flou à fonction PD.

Voici la table d'inférence assurant une action PD non-linéaire à la sortie de RLF, dans le cas d'une fuzzification par 5 sous-ensembles flous par variable pour x (tq. x = en , den, Ucmn).

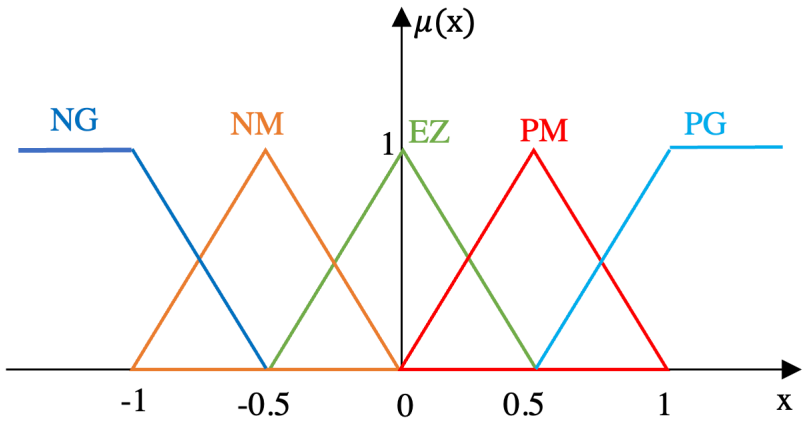


Tableau 1 : la table d'inférence pour l'action PD.

S_{PD}		en				
		NG	NM	EZ	PM	PG
den	NG	NG	NG	NM	NM	EZ
	NM	NG	NM	NM	EZ	PM
	EZ	NM	NM	EZ	PM	PM
	PM	NM	EZ	PM	PM	PG
	PG	EZ	PM	PM	PG	PG

Notons dans ce cas que la sortie de régulateur flou et une fonction non-linéaire de l'erreur et son dérivé d'où le caractère PD de ce régulateur. On peut donc écrire :

$$S_{PD} = U_{cm} = K_p(e, \Delta_e).e + K_D(e, \Delta_e).\Delta_e \qquad \text{Eq.(I.5)}$$

Où K_p, K_D sont des fonctions non linéaires qui changent avec les points de fonctionnement de RLF , cette action PD n'agit que dans le régime transitoire. Pour cela, on a besoin des actions PI ou PID . Pour réaliser ces deux actions on utilise des RLF avec PI et PID comme illustré par les figures I.7 et I.8.

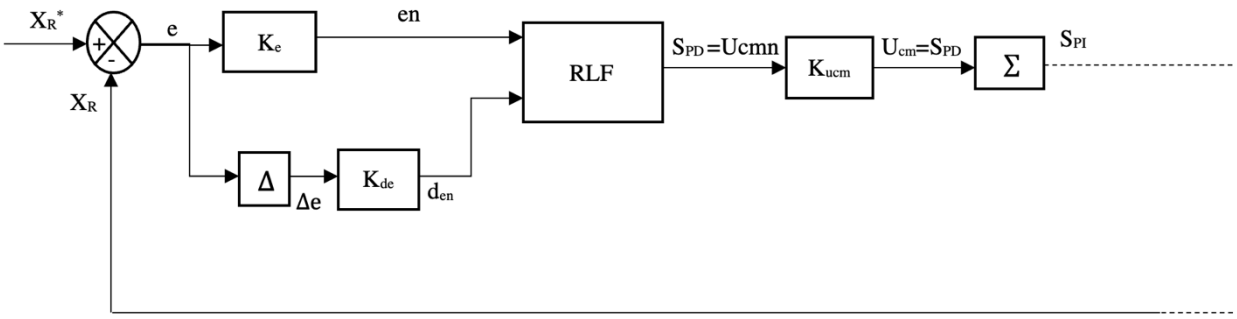


Figure 1.7 Régulateur flou à fonction PI.

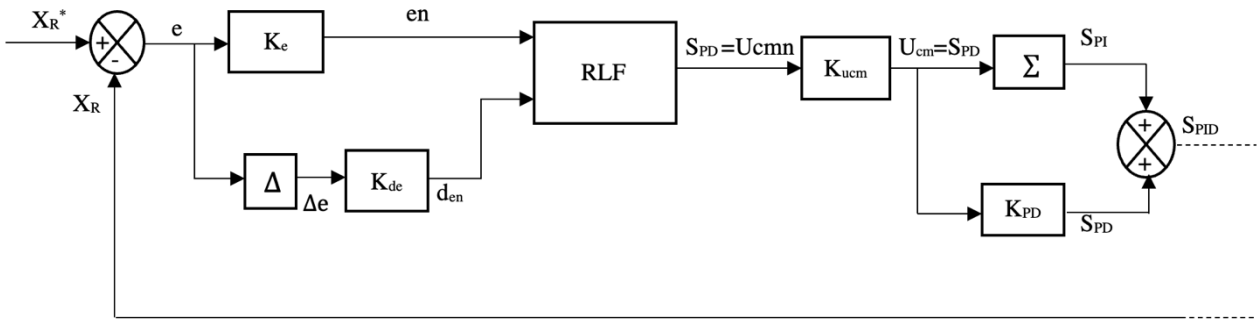


Figure 1.8 Régulateur flou à fonction PID.

I.8 Avantages et inconvénients de la logique floue

Plusieurs méthodes peuvent être utilisées pour la réalisation des différents réglages par logique floue. Le choix de telle ou telle méthode est conditionné par un compromis entre facilité et performance. D'autre part, voici un tableau qui résume les avantages et inconvénient principaux de la logique floue.

Tableau 2 : les avantages et inconvénients de la LF.

Avantages	Inconvénient
- Le RLF n'a pas besoin d'un modèle - Implémentation de la connaissance linguistique - Maitrise des systèmes à comportement complexe (non-linéaire)	- Manque de direction précise pendant l'étape de conception. - Approche artisanale et non systématique. - Impossibilité de démonstration de la stabilité du système de réglage.

I.9 Conclusion

La logique floue est un outil efficace pour résoudre divers problèmes informatiques dans le monde. Cette technique a été appliquée dans différentes machines et applications pour contrôler des actions en fonction de certaines conditions prédéfinies. À l'avenir, la logique floue sera appliquée à divers produits et systèmes. La portée des applications de logique floue augmentera en raison des progrès technologiques et de la transformation numérique.

Partie II : Réseaux de neurones artificiels (RNA)

II.1 Introduction

Chez les êtres vivants, la reconnaissance de forme, le traitement de signal, l'apprentissage, la mémorisation et la généralisation sont des tâches remplies quotidiennement d'une manière naturelle. C'est à partir de l'hypothèse que le comportement intelligent émergé de la structure et du comportement des éléments de base du cerveau que les réseaux de neurones artificiels (RNA) sont développés.

L'introduction des RNA dans le domaine de science de l'ingénieur a été une évolution les plus marquée. Tout à commencer en 1943 avec l'invention du premier neurone artificiel, qui n'était qu'un produit scalaire d'un vecteur d'entrée et un vecteur de poids, suivis d'un élément à seuil. Grace à leur propriété tel que le parallélisme, l'adaptation, la généralisation et l'approximation, les RNA constituent aujourd'hui un véritable outil pour la résolution de plusieurs problèmes où les méthodes classiques ont montré leurs limites. L'optimisation des systèmes non linéaires par exemple a été parmi les applications où les RNA ont été utilisés, notamment après l'apparition de la technique d'apprentissage par rétropropagation.

Cette partie de cours vise à introduire les RNA , on effectuant un surveille historique de l'évolution des RNA , leurs différentes architectures et applications. Ensuite, nous décrivons comment un RNA calcule des valeurs prédire du processus à modéliser ou à commander en fonction de ses entrée. Nous présentons ensuite les techniques d'apprentissages et la méthodologie liée aux choix de la complexité des RNA.

II.2 Historique des RNA

L'histoire des RNA remonte aux années 1940, lorsque Warren McCulloch et Walter Pitts ont publié un article sur les modèles informatiques de neurones avec la proposition du premier neurone artificiel.

Dans les années 1950 et 1960, des travaux ont été menés sur les RNA pour comprendre comment ils pouvaient être utilisés pour résoudre des problèmes de reconnaissance de formes et de traitement de signal. En 1957, F. Rosenblatt inventa le modèle de perceptron qui est un neurone formel, le plus petit réseau neuronal possible. En 1969, M. Minsky et S. Papert publièrent un ouvrage qui met en exergue les limitations théoriques du perceptron. Cependant, ces premiers modèles étaient très limités en termes de complexité et de capacités de traitement.

En 1985, c'était la présentation d'un perceptron multicouche avec l'apparaissions d'algorithme d'apprentissage adapté aux réseaux de neurones multicouches (rétropropagation de gradient). Au cours des années 1980 et 1990, les progrès en matière de traitement de l'information et de stockage de données ont permis le développement de RNA de plus en plus complexes, capables de résoudre des tâches de plus en plus difficiles. Ces avancées ont conduit à l'utilisation de RNA dans de nombreux domaines, notamment la vision par ordinateur, la reconnaissance de la parole et la robotique.

Aujourd'hui, les RNA sont largement utilisés dans de nombreuses applications, notamment la reconnaissance de la parole et de l'image, le traitement du langage naturel, la prédiction de séries chronologiques et l'utilisation de deep-learning (réseau de neurones profond). Ils continuent d'évoluer et de s'améliorer grâce aux progrès de la recherche et de l'informatique [10].

II.3 Définition des RNA

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau [5].

II.4 Neurone biologique

L'informatique neuronale est un paradigme de traitement de l'information, inspiré du système biologique, composé d'un grand nombre d'éléments de traitement hautement interconnectés (neurones) travaillant à l'unisson pour résoudre des problèmes spécifiques.

Le cerveau humain est constitué d'un grand nombre, plus d'un milliard de cellules neurales qui traitent l'information. Chaque cellule fonctionne comme un simple processeur. L'interaction massive entre toutes les cellules et leur traitement parallèle ne fait que rendre possibles les capacités du cerveau.

La figure II.1 représente une unité nerveuse biologique humaine. Diverses parties du réseau neuronal biologique (BNN) sont marquées dans la figure II.1 [5,10].

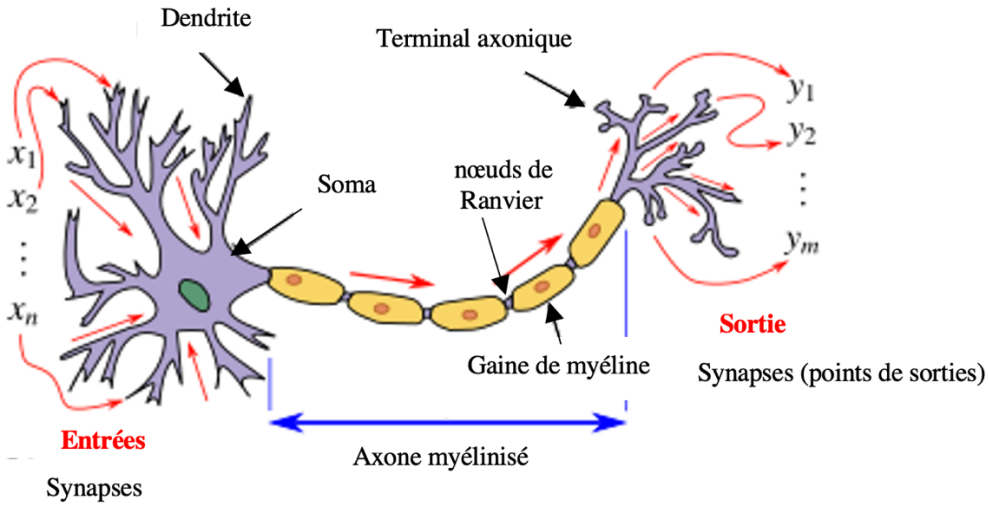


Figure II.1 Unité nerveuse biologique humaine.

- Les **dendrites** : sont des fibres ramifiées qui s'étendent du corps cellulaire ou du soma.

- Le **soma** ou **corps cellulaire** d'un neurone contient le noyau et d'autres structures, supporte le traitement chimique et la production de neurotransmetteurs.

- **L'axone** : est une fibre singulière qui transporte l'information du soma vers les sites synaptiques d'autres neurones (dendrites et somas), muscles ou glandes.

- **Cône axonique** : est le site de sommation des informations entrantes. À tout moment, l'influence collective de tous les neurones qui transmettent des impulsions à un neurone donné déterminera si oui ou non un potentiel d'action sera initié au niveau de la butte axonale et propagé le long de l'axone.

- La **gaine de myéline** est constituée de cellules contenant de la graisse qui isolent l'axone de l'activité électrique. Cette isolation agit pour augmenter le taux de transmission des signaux.

- Un espace existe entre chaque cellule de la gaine de myéline le long de l'axone. Puisque la graisse inhibe la propagation de l'électricité, les signaux sautent d'un espace à l'autre.

- Les **nœuds de Ranvier** : sont les espaces (environ 1 µm) entre les cellules de la gaine de myéline. Puisque la graisse sert de bon isolant, les gaines de myéline accélèrent le taux de transmission d'une impulsion électrique le long de l'axone.

- La **synapse** est le point de connexion entre deux neurones ou un neurone et un muscle ou une glande. La communication électrochimique entre les neurones à lieu à ces jonctions.

- Les **boutons** terminaux d'un neurone sont les petits boutons situés à l'extrémité d'un axone qui libèrent des substances chimiques appelées neurotransmetteurs.

II.5 Modèle mathématique d'une neurone artificiel formel

L'origine de RNA est issue de la modélisation du neurone biologique. McCulloch et Pitts supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée est née grâce à l'effet collectif d'un RN interconnecté, la figure II.2 représente un neurone artificiel formel [11].

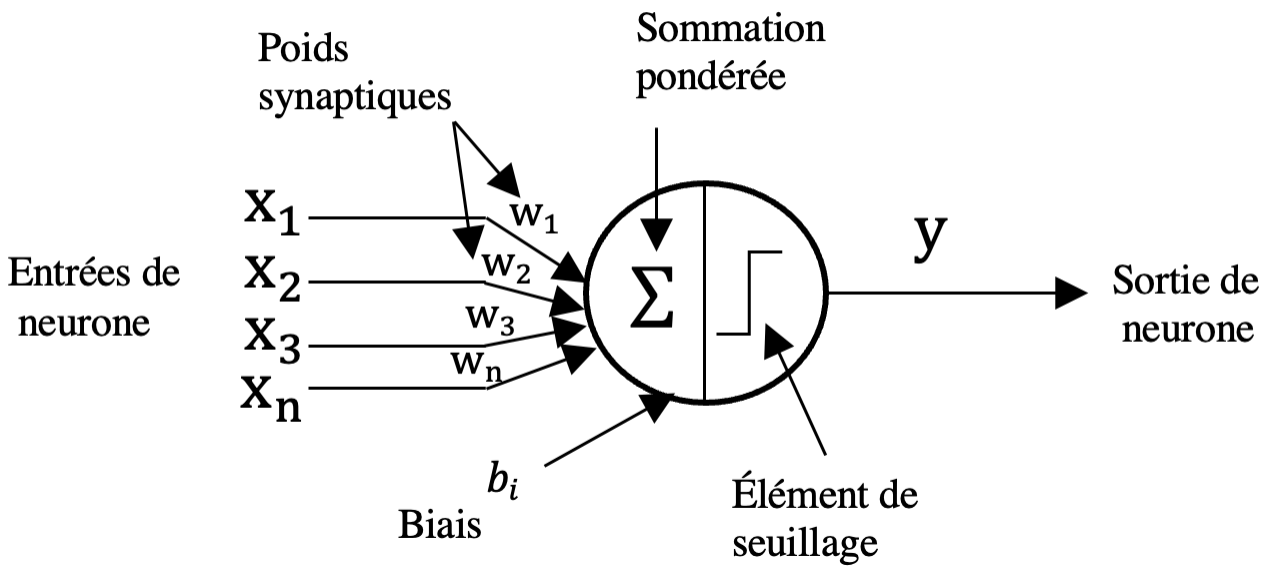


Figure II.2 Neurone artificiel formel.

Les entrées sont désignées par x_i (i = 1…n) et les paramètres w_i (i = 1…n) sont appelés poids synaptiques de neurone, b_i est le biais ; c'est un paramètre supplémentaire dans le réseau de neurones qui est utilisé pour ajuster la sortie avec la somme pondérée des entrées du neurone. La somme pondérée des signaux d'entrée constitue l'activation du neurone ; c'est une fonction qui définit l'activité du neurone, elle est appelée aussi « fonction de base » qui se transforme en sortie après son passage par une fonction de seuillage.

II.5.1 Fonction de base (activation)

Généralement, elle est notée *U (w,x)* tel que *w* et *x* sont les vecteurs des poids synaptiques et des entrées. Cette fonction possède plusieurs formes [10-11] :

- La fonction L.B.F. (Linear Basis Function)

$$U(w,x) = \sum_{i=1}^{n} w_i x_i$$

- La fonction R.B.F. (Radial Basis Function)

$$U(w,x) = \sqrt{\sum_{i=1}^{n} (w_i - x_i)^2}$$

- La fonction E.B.F. (Elleptic basis function)

$$U(w,x) = \sum_{i=1}^{n} \left[\alpha_i (w_i - x_i)^2 + \theta_j\right]$$

II.5.2 Fonction de seuillage (transfert)

L'objective de cette fonction dite aussi fonction d'activation est de rendre la sortie bornée. L'une des formes de cette fonction est la fonction « signe », or cette fonction qui délivre une sortie discret ou binaire n'est mathématiquement pas adoptée à certaine opérations notamment « la différentiation » qui, comme on le verra par la suite , est nécessaire pour l'apprentissage du RNA.

Pour cela , la fonction de seuillage doit être continue , dérivable et monotone. D'autres fonctions qui effectuent le seuillage, mais d'une manière moins abrupte sont utilisées, parmi ces fonctions, on trouve la fonction sigmoïde, la fonction tangente hyperboloïde et la fonction linéaire saturée. La figure II.3 regroupe les fonctions de seuillage les plus utilisées [5, 10-11].

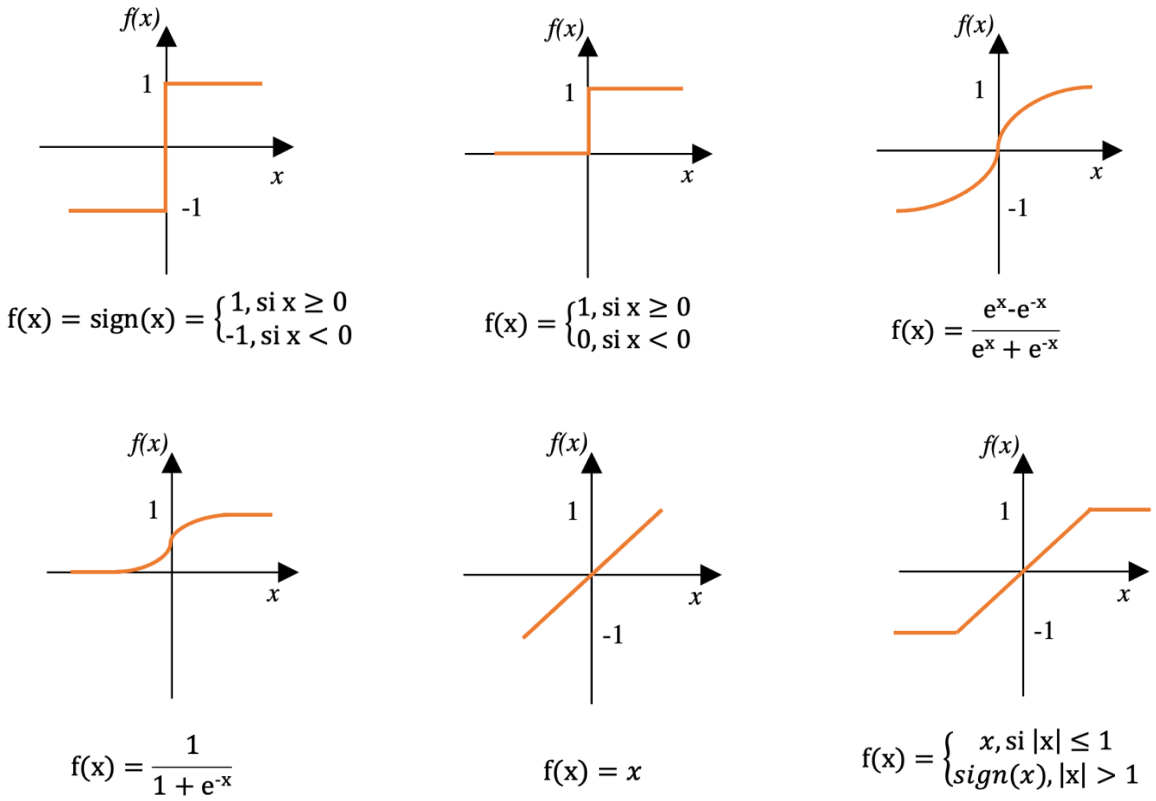


Figure II.3 Les fonctions de seuillage

Donc la sortie d'un neurone artificiel est donnée par :

$$y = f(x) = f(U(w, x), b) \qquad \text{Eq.(II.1)}$$

II.6 Le réseau de neurones artificiels (R.N.A.)

Un réseau de neurones est composé de nombreux neurones connectés entre eux. Chaque neurone peut recevoir des entrées de plusieurs autres neurones et produire une sortie qui est transmise à d'autres neurones. Les connexions entre les neurones sont pondérées, ce qui signifie qu'elles ont un poids qui peut être modifié pour influencer la sortie du neurone.

Les neurones du réseau sont organisés en couches, avec une couche d'entrée qui reçoit les données en entrée, une ou plusieurs couches cachées qui traitent les données et une couche de sortie qui produit le résultat final, cette organisation est appelée réseau de neurones multicouche. (Multi Layer Perception MLP). Les connexions entre les neurones de différentes couches sont généralement pondérées de manière à ce que certaines connexions aient plus d'influence sur la sortie que d'autres. La figure II.4 illustre le MLP [5, 10-11].

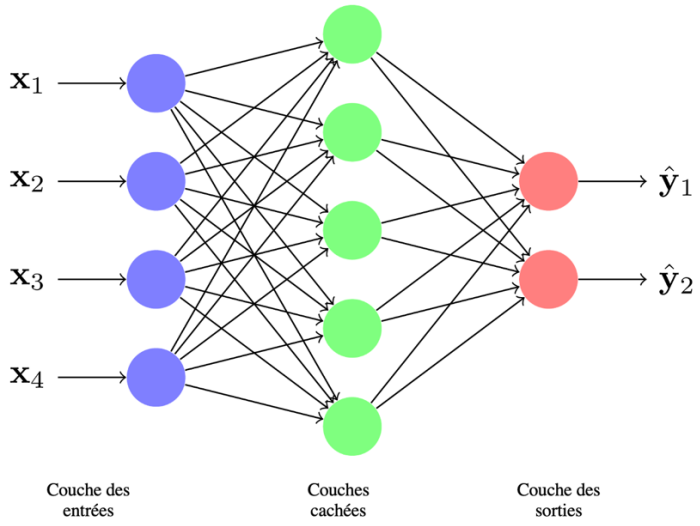


Figure II.4 L'architecture d'un réseau de neurones multicouches.

II.6.1 Architectures des RNA

Il existe trois grandes classes d'architecture des RNA ; les RNA statiques (non récurrentes) , les RNA dynamiques (récurrentes) et les RNA à architecture évolutive [5, 10-11].

A) RNA Statiques

Un réseau de neurones statique est un modèle de réseau de neurones qui ne peut pas être modifié une fois qu'il a été entraîné. Cela signifie que les poids des connexions entre les neurones du réseau ne peuvent pas être mis à jour, même si le réseau est présenté avec de nouvelles données. Les réseaux de neurones statiques sont utilisés principalement pour la reconnaissance de formes et la classification de données. Ils sont souvent utilisés comme modèles de base pour comparer les performances de différents algorithmes de traitement du signal et de l'image.

Les réseaux de neurones statiques sont généralement moins complexes que les réseaux de neurones dynamiques, qui peuvent être modifiés en fonction des données qu'ils reçoivent. Cependant, les réseaux de neurones statiques ont généralement une précision inférieure lorsqu'ils sont utilisés pour résoudre des tâches de classification ou de reconnaissance de formes, car ils ne peuvent pas s'adapter aux nouvelles données.

B) RNA dynamiques

Un réseau de neurones dynamique est un modèle de réseau de neurones qui peut être modifié en fonction des données qu'il reçoit. Cela signifie que les poids des connexions entre les neurones du réseau peuvent être mis à jour lorsque le réseau est présenté avec de nouvelles données.

Les réseaux de neurones dynamiques sont utilisés principalement pour la reconnaissance de formes et la classification de données. Ils sont souvent utilisés pour résoudre des tâches de classification ou de reconnaissance de formes qui nécessitent une adaptation continue aux nouvelles données.

Les réseaux de neurones dynamiques sont généralement plus complexes que les réseaux de neurones statiques, car ils peuvent s'adapter aux nouvelles données. Cependant, ils peuvent également être plus difficiles à entraîner et à mettre en œuvre que les réseaux de neurones statiques. Il existe de nombreuses variantes de réseaux de neurones dynamiques, notamment les réseaux de neurones à rétropropagation, les réseaux de neurones à base de données et les réseaux de neurones

en temps réel. Chacune de ces variantes est utilisée pour résoudre différents types de problèmes de traitement de l'information.

II.6.2 L'apprentissage des RNA

L'apprentissage des réseaux de neurones est le processus par lequel un réseau de neurones est capable de "apprendre" à partir de données. Cela se fait en ajustant les poids et les biais des différentes couches du réseau de neurones, de manière à ce qu'il soit capable de produire des résultats précis pour de nouvelles données [12-13].

Pour entraîner un réseau de neurones, on utilise généralement une grande quantité de données d'entraînement et on utilise un algorithme d'optimisation pour ajuster les poids et les biais du réseau de manière à minimiser l'erreur de prédiction. L'erreur de prédiction est mesurée en comparant les prédictions du réseau avec les valeurs réelles pour chaque exemple d'entraînement. Plus l'erreur est faible, plus le réseau de neurones est précis.

Une fois que le réseau de neurones a été entraîné, il peut être utilisé pour effectuer des prédictions sur de nouvelles données. Si le réseau a été entraîné de manière adéquate, il devrait être capable de produire des prédictions précises pour ces nouvelles données.

A) L'apprentissage supervisé

L'apprentissage supervisé est un type d'apprentissage automatique dans lequel un modèle est entraîné sur des données d'entraînement qui comprennent des exemples d'entrée et leur sortie correspondante attendue. Le modèle essaie de prédire la sortie attendue pour de nouvelles entrées en utilisant ce qu'il a appris pendant l'entraînement.

Pendant l'entraînement, le modèle est exposé à de nombreux exemples d'entrée et de leur sortie attendue, et il ajuste ses paramètres pour minimiser l'erreur de prédiction. L'erreur de prédiction est mesurée en comparant les prédictions du modèle avec les valeurs réelles de la sortie attendue pour chaque exemple d'entraînement.

L'apprentissage supervisé est couramment utilisé dans de nombreux domaines, notamment la reconnaissance de la parole, la reconnaissance d'images et la prédiction des prix des actions. Il est particulièrement utile lorsqu'il y a une grande quantité de données d'entraînement et que la relation entre les entrées et les sorties est clairement définie.

L'apprentissage supervisé peut être utilisé pour entraîner des réseaux de neurones de manière similaire à la façon dont il est utilisé pour entraîner d'autres modèles d'apprentissage automatique. Pour entraîner un réseau de neurones en utilisant l'apprentissage supervisé, vous devez disposer d'un jeu de données d'entraînement qui comprend des exemples d'entrée et leur sortie attendue correspondante. Vous utilisez ces données pour entraîner le réseau de neurones en ajustant les poids et les biais de manière à minimiser l'erreur de prédiction.

Voici un exemple de processus d'apprentissage supervisé pour un réseau de neurones [12-13] :

1- Préparez votre jeu de données d'entraînement, qui comprend des exemples d'entrée et leur sortie attendue.
2- Définissez la structure du réseau de neurones, c'est-à-dire le nombre de couches et le nombre de neurones dans chaque couche.
3- Initialisez les poids et les biais du réseau de manière aléatoire.
4- Pour chaque exemple d'entraînement, faites passer l'entrée à travers le réseau et obtenez la prédiction de sortie.
5- Calculez l'erreur de prédiction en comparant la prédiction avec la sortie attendue pour cet exemple.
6- Utilisez l'algorithme d'optimisation pour ajuster les poids et les biais du réseau de manière à minimiser l'erreur de prédiction.
7- Répétez les étapes 4 à 6 jusqu'à ce que l'erreur de prédiction soit acceptablement faible.

Une fois que le réseau de neurones a été entraîné, vous pouvez l'utiliser pour effectuer des prédictions sur de nouvelles données en faisant passer ces données à travers le réseau et en obtenant la prédiction de sortie. Si le réseau a été entraîné de manière adéquate, il devrait être capable de produire des prédictions précises pour ces nouvelles données.

B) L'apprentissage non-supervisé

L'apprentissage non supervisé est un type d'apprentissage automatique dans lequel un modèle est entraîné sur des données qui ne comprennent pas de sortie attendue. Au lieu de cela, le modèle doit apprendre à partir des données elles-mêmes et à découvrir des structures et des patterns dans les données.

L'apprentissage non supervisé peut être utilisé pour entraîner des réseaux de neurones de manière similaire à la façon dont il est utilisé pour entraîner d'autres modèles d'apprentissage automatique. Pour entraîner un réseau de neurones en utilisant l'apprentissage non supervisé, vous devez disposer d'un jeu de données qui ne comprend pas de sortie attendue. Vous utilisez ces données pour entraîner le réseau de neurones en ajustant les poids et les biais de manière à ce qu'il soit capable de découvrir des structures et des patterns dans les données.

Voici un exemple de processus d'apprentissage non supervisé pour un réseau de neurones [12-13]:

1- Préparez votre jeu de données d'entraînement, qui ne comprend pas de sortie attendue.
2- Définissez la structure du réseau de neurones, c'est-à-dire le nombre de couches et le nombre de neurones dans chaque couche.
3- Initialisez les poids et les biais du réseau de manière aléatoire.
4- Pour chaque exemple d'entraînement, faites passer l'entrée à travers le réseau et obtenez la prédiction de sortie.
5- Utilisez l'algorithme d'optimisation pour ajuster les poids et les biais du réseau de manière à ce qu'il soit capable de découvrir des structures et des patterns dans les données.
6- Répétez les étapes 4 et 5 jusqu'à ce que le réseau ait appris à découvrir les structures et les patterns dans les données de manière satisfaisante.

Une fois que le réseau de neurones a été entraîné, vous pouvez l'utiliser pour effectuer des prédictions sur de nouvelles données en faisant passer ces données à travers le réseau et en obtenant la prédiction de sortie. Cependant, étant donné que l'apprentissage non supervisé ne fournit pas de sortie attendue pour comparer les prédictions, il est souvent difficile de savoir si le réseau de neurones a été entraîné de manière adéquate ou non.

II.6.2.1 Les algorithmes d'apprentissage des RNA

A) L'apprentissage par rétropropagation

La rétropropagation est un algorithme utilisé dans les réseaux de neurones pour entraîner et mettre à jour les poids des différentes couches du réseau. Elle est également connue sous le nom d'algorithme de descente de gradient.

Lors de l'entraînement d'un réseau de neurones, on utilise une grande quantité de données d'entraînement qui sont envoyées à travers le réseau. Le réseau calcule alors des prédictions pour chaque entrée de données et la comparaison de ces prédictions avec les valeurs attendues (également appelées "étiquettes") permet de calculer une "erreur" pour chaque entrée de données. L'objectif de l'entraînement du réseau est de minimiser cette erreur en ajustant les poids du réseau de manière à ce qu'il produise des prédictions plus précises.

La rétropropagation est utilisée pour mettre à jour les poids du réseau de neurones. Elle fonctionne en effectuant une "propagation avant" des données d'entraînement à travers le réseau, puis en effectuant une "propagation arrière" de l'erreur calculée en comparant les prédictions du réseau aux étiquettes attendues. Lors de la propagation arrière, l'erreur est propagée à travers le réseau et utilisée pour mettre à jour les poids de chaque couche du réseau de manière à minimiser l'erreur globale.

La rétropropagation est largement utilisée dans l'apprentissage automatique en raison de sa simplicité et de sa robustesse. Elle est particulièrement utile pour entraîner des réseaux de neurones à plusieurs couches, qui sont couramment utilisés dans de nombreuses tâches d'apprentissage automatique.

Voici les équations de base utilisées pour mettre à jour les poids d'un réseau de neurones lors de l'entraînement par rétropropagation [12-13] :

1. Puisque l'erreur est la différence entre la sortie réelle $y(k)$ et la sortie souhaitée $y_d(k)$, l'erreur dépend des poids w_{ij} , et nous devons ajuster les poids afin de minimiser l'erreur quadratique globale ε. Nous pouvons définir la fonction d'erreur pour la sortie de chaque neurone :

$$\varepsilon = (y_d(k) - y(k))^2 \qquad \text{Eq.(II.2)}$$

2. Nous prenons le carré de la différence entre la sortie et la cible souhaitée parce qu'elle sera toujours positive, et parce qu'elle sera plus grande si la différence est grande, et moindre si la différence est petite. L'erreur du réseau sera simplement la somme des erreurs de tous les neurones de la couche de sortie

$$\varepsilon = \frac{1}{2}\sum_k (y_d(k) - y(k))^2 \qquad \text{Eq.(II.3)}$$

3. L'algorithme de rétropropagation calcule maintenant comment l'erreur dépend de la sortie, des entrées et des poids. Après avoir trouvé cela, nous pouvons ajuster les poids en utilisant la méthode de gradient descendant

$$\Delta w_{ij} = -\alpha \frac{\partial \varepsilon}{\partial w_{ij}} \qquad \text{Eq.(II.4)}$$

4. Cette formule peut être interprétée de la manière suivante : l'ajustement de chaque poids w_{ij} sera le négatif d'une constante a multiplié par la dépendance du i poids précédent à l'erreur du réseau, qui est la dérivée de *e* par rapport à w_{ij} . Nous l'utilisons jusqu'à ce que nous trouvions les poids appropriés (l'erreur est minime) dans la couche m.

$$\begin{aligned} \Delta w_{ij}^m = -\alpha \quad & \frac{1}{2}\frac{\partial}{\partial w_{ij}^m}\left(\sum_k (y_d(k) - y^{m+1}(k))(y_d(k) - y^{m+1}(k))\right) \\ & = -\alpha \sum_k (y_d(k) - y^{m+1}(k)) \frac{\partial y^{m+1}(k)}{\partial w_{ij}^m} \end{aligned} \qquad \text{Eq.(II.5)}$$

Avec

$$\frac{\partial y^{m+1}(k)}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^{m+1}} f^{m+1}(x^{m+1}(k)) \qquad \text{Eq.(II.6)}$$

$$x^{m+1}(k) = \sum_{ij} w_{ij}^{m+1} y_{ij}^m(k) \qquad \text{Eq.(II.7)}$$

On obtient

$$\Delta w_{ij}^{m} = \alpha \sum_k (y_d(k) - y^{m+1}(k)) w_{ij}^{m+1} f^{(m+1)'}(x^{m+1}(k)) f^{m'}\left(x_{ij}^{m}(k)\right) y_{ij}^{m-1}(k) \qquad \text{Eq.(II.8)}$$

Introduisons maintenant l'erreur élémentaire :

$$\begin{aligned} \delta^{m+1}(k) &= (y_d(k) - y^{m+1}(k)) f^{(m+1)'}(x^{m+1}(k)) \\ \Delta w_{ij}^{m} &= \alpha \sum_k \delta^{m+1} y_{ij}^{m} \end{aligned} \qquad \text{Eq.(II.9)}$$

Généralement nous avons :

$$\delta^{m}(k) = \left\{\sum_k \delta^{m+1}(k) w_{ij}^{m+1}(k)\right\} f^{m'}\left(x_{ij}^{m}(k)\right) \qquad \text{Eq.(II.10)}$$

Et :

$$\Delta w_{ij}^{m} = \alpha \sum_k (y_d(k) - y^{m}(k)) f^{m'}\left(x_{ij}^{m}(k)\right) y_{ij}^{m-1}(k) \qquad \text{Eq.(II.11)}$$

Remarque : On peut choisir la vitesse de mis-à-jour des poids lors de l'apprentissage en ajustant sur le coefficient α ce qui peut accélérer l'opération de convergence d'apprentissage .

Un résumé cette algorithme est donné ci-dessous:

- Alimenter le MLP avec un échantillon d'apprentissage ;
- Comparez la sortie à la sortie souhaitée de cet échantillon et calculez l'erreur dans chaque neurone (il s'agit de l'erreur locale) ;
- L'erreur locale est supposée être causée par les neurones du niveau précédent, proportionnellement à la valeur de poids de chaque connexion neuronale arrivant au niveau sous enquête;
- Ajuster les poids arrivant à chaque neurone pour minimiser l'erreur locale ;
- Répétez les étapes ci-dessus jusqu'au niveau précédent (en arrière) en utilisant comme erreur la contribution des neurones à l'erreur locale de l'étape précédente.

B) Algorithme quasi-Newton

L'algorithme quasi-Newton est une méthode utilisée dans l'apprentissage de réseaux de neurones pour mettre à jour les poids du réseau de manière à minimiser l'erreur. Il s'agit d'une variante de l'algorithme de descente de gradient, qui est largement utilisé dans l'apprentissage de réseaux de neurones.

L'algorithme quasi-Newton utilise une approximation de la matrice Hessienne (une matrice de seconde dérivée) pour calculer la direction de descente du gradient à chaque itération de l'entraînement. Cette approximation est calculée à partir de l'historique des itérations précédentes et permet de mieux capturer la structure de l'espace de perte (c'est-à-dire la fonction d'erreur du réseau) et de converger plus rapidement vers un minimum local.

En général, l'algorithme quasi-Newton peut être plus rapide et plus efficace que l'algorithme de descente de gradient standard, en particulier lorsque l'espace de perte est "plat" ou "creux" (c'est-à-dire lorsque la dérivée de la fonction d'erreur est très petite ou nulle). Cependant, il peut être plus difficile à implémenter et nécessite une plus grande quantité de mémoire pour stocker l'historique des itérations.

En général, l'algorithme quasi-Newton est utilisé en combinaison avec d'autres techniques d'optimisation, telles que l'algorithme de descente de gradient, pour améliorer l'efficacité de l'apprentissage de réseaux de neurones. Il est également couramment utilisé dans d'autres domaines de l'apprentissage automatique, tels que la régression et la classification, pour minimiser l'erreur et améliorer les performances du modèle.

Les nouveaux poids synaptiques de la couche *m* sont calculés par [12-13] :

$$\Delta w_{ij}^{m}(\mathrm{n}+1) = w_{ij}^{m}(\mathrm{n}) - H^{-1}(n)G(n) \qquad \text{Eq.(II.12)}$$

Avec :

n : est le nombre d'itérations.

$H(n)$: est la matrice Hessienne.

$G(n)$: est le gradient.

C) Algorithme de Levenberg-Marquardt

L'algorithme de Levenberg-Marquardt combine les avantages de l'algorithme de descente de gradient et de l'algorithme quasi-Newton en utilisant une approximation de la matrice Hessienne (une matrice de seconde dérivée) pour calculer la direction de descente du gradient à chaque itération de l'entraînement. Cette approximation est calculée à partir de l'historique des itérations précédentes et permet de mieux capturer la structure de l'espace de perte (c'est-à-dire la fonction d'erreur du réseau) et de converger plus rapidement vers un minimum local.

En général, l'algorithme de Levenberg-Marquardt peut être plus rapide et plus efficace que l'algorithme de descente de gradient standard .Cependant, il peut être plus difficile à implémenter et nécessite une plus grande quantité de mémoire pour stocker l'historique des itérations. En général, l'algorithme de Levenberg-Marquardt est utilisé en combinaison avec d'autres techniques d'optimisation, telles que l'algorithme de descente de gradient, pour améliorer l'efficacité de l'apprentissage de réseaux de neurones.

Les nouveaux poids synaptiques de la couche *m* sont calculés par [12-13] :

$$\Delta w_{ij}^{m}(\mathrm{n}+1) = w_{ij}^{m}(\mathrm{n}) + (H_a(n) + \alpha I)^{-1} G(n) \qquad \text{Eq.(II.13)}$$

$H_a(n) = J_c^T J_c$ (avec J_c : la matrice jacobienne).
$G(n) = J_c^T \varepsilon$: (avec ε : le vecteur d'erreur du RNA par rapport aux poids).

II.6.3 Quelques applications des RNA

On va se limiter dans cette section à montrer comment appliquer les RNA aux problèmes d'identification et de commande [14].

A) Identification par les RNA

L'identification par réseau de neurones est une technique de traitement de l'information utilisée dans l'apprentissage automatique. Elle consiste à utiliser un réseau de neurones artificiels pour effectuer des tâches d'identification, comme la reconnaissance de formes ou de mots.

Pour effectuer l'identification par réseau de neurones, on utilise généralement un réseau de neurones en réseau de type "feedforward". Cela signifie que l'information est introduite au début du réseau et transite à travers différentes couches de neurones avant d'atteindre la sortie. Chaque neurone reçoit de l'information des neurones de la couche précédente, la traite et envoie le résultat aux neurones de la couche suivante.

Pour entraîner un réseau de neurones à effectuer une tâche d'identification, on utilise une grande quantité de données d'entraînement et on ajuste les poids des connexions entre les neurones de manière à minimiser l'erreur de prédiction du réseau. Une fois entraîné, le réseau de neurones peut être utilisé pour identifier des objets ou des mots en analysant de nouvelles données d'entrée.

Il existe de nombreuses applications de l'identification par réseau de neurones, notamment la reconnaissance de formes et de mots, la traduction automatique, la reconnaissance de la parole et la détection de spam.

B) Commande par RNA

La commande par réseau de neurones est une technique utilisée dans le domaine de l'automatisation et de la robotique pour contrôler un système à l'aide d'un réseau de neurones artificiels. Elle permet de remplacer les algorithmes de commande traditionnels par un modèle de traitement de l'information inspiré de la structure du cerveau humain.

Le principe de la commande par réseau de neurones est similaire à celui de l'identification par réseau de neurones. Un réseau de neurones est entraîné à partir de données d'entraînement pour effectuer une tâche de commande en analysant des données d'entrée et en produisant des données de sortie. La différence principale est que, dans le cas de la commande, les données d'entrée correspondent aux grandeurs mesurées sur le système à commander (par exemple, la vitesse d'un moteur) et les données de sortie correspondent aux actions à effectuer sur le système (par exemple, le courant à fournir au moteur).

La commande par réseau de neurones à l'avantage de pouvoir traiter des données complexes et de s'adapter à des situations imprévues. Elle est souvent utilisée dans les systèmes de commande de robots mobiles, de véhicules autonomes et de systèmes de production industrielle. Toutefois, elle nécessite généralement un grand nombre de données d'entraînement et peut être difficile à mettre en œuvre et à déboguer.

Il existe deux types de structure pour la commande par RNA

- Architecture de contrôle autoréglable :

Le contrôle autoréglable (ou "feedback") est une technique de commande qui utilise une boucle de rétroaction pour maintenir un système à un état souhaité. Elle consiste à mesurer l'état actuel du système et à comparer cette mesure à une cible prédéfinie. En fonction de l'écart entre la mesure et la cible, une correction est apportée à l'action de commande pour ramener le système à l'état souhaité. Dans ce type le RNA est utilisés pour ajuster les paramètres d'un contrôleur conventionnel, comme illustrer par la figure II.5 ;

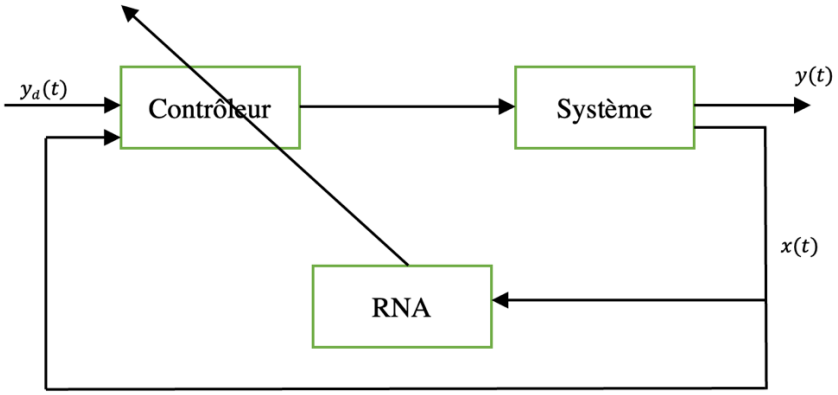


Figure II.5 Architecture de contrôle autoréglable par RNA.

- Architecture de contrôle direct :

Le contrôle direct par réseau de neurones, quant à lui, est une technique de commande qui utilise un réseau de neurones artificiels pour analyser les données d'entrée et produire des données de sortie qui commandent le système. Le réseau de neurones est entraîné à partir de données d'entraînement pour apprendre à effectuer la tâche de commande en fonction des données d'entrée. Dans ce type , les signaux de commande sont générés par le RNA lui-même, comme illustrer par la figure II.6 ;

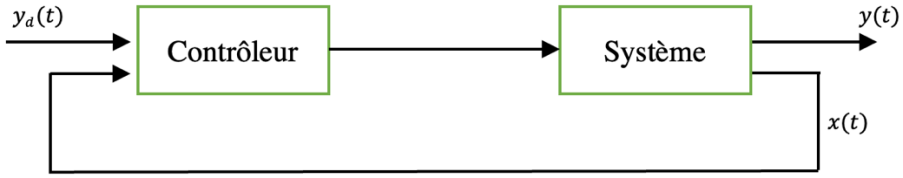


Figure II.6 Architecture de contrôle direct par RNA.

Le choix entre le contrôle autoréglable et le contrôle direct par réseau de neurones dépend de la nature du système à commander et de la tâche à effectuer. Le contrôle autoréglable est souvent utilisé lorsque le système est instable et nécessite un maintien à un état souhaité. Le contrôle direct par réseau de neurones est souvent utilisé lorsque le système est complexe et que l'on souhaite s'affranchir des modèles mathématiques traditionnels de commande.

B.1) Le contrôle supervisé

Le contrôle supervisé par réseau de neurones est une technique de commande utilisée dans le domaine de l'apprentissage automatique. Elle consiste à utiliser un réseau de neurones artificiels entraîné à partir de données étiquetées pour effectuer une tâche de commande en analysant des données d'entrée et en produisant des données de sortie.

Le contrôle supervisé par réseau de neurones est similaire au contrôle direct par réseau de neurones, mais il utilise des données d'entraînement étiquetées qui comprennent non seulement les données d'entrée, mais également les données de sortie souhaitées pour chaque cas d'entraînement. Cela permet au réseau de neurones de "savoir" quelle sortie est attendue pour chaque entrée et de s'adapter en conséquence.

Le contrôle supervisé par réseau de neurones est souvent utilisé dans les applications de commande où il est possible de disposer de données d'entraînement étiquetées, comme la commande de robots mobiles ou de véhicules autonomes. Il peut également être utilisé dans les applications de commande où il est difficile de définir explicitement les règles de commande, comme la commande de systèmes industriels complexes. Toutefois, il nécessite généralement un grand nombre de données d'entraînement et peut être difficile à mettre en œuvre et à déboguer.

II.6.4 Réseaux de neurones avec Matlab

Matlab est un logiciel de calcul scientifique et technique qui offre une variété d'outils pour le développement de réseaux de neurones. Vous pouvez utiliser Matlab pour entraîner, simuler et évaluer des réseaux de neurones pour différentes applications.

Voici les étapes générales pour créer et entraîner un réseau de neurones dans Matlab :

1- Préparez les données d'entraînement et de test : vous aurez besoin de données d'entrée et de sortie étiquetées pour entraîner le réseau de neurones. Assurez-vous de disposer d'un échantillon suffisant de données et de les diviser en ensembles d'entraînement et de test.
2- Créez le modèle de réseau de neurones : utilisez les outils de création de réseau de neurones de MATLAB pour définir la structure et les fonctions d'activation de votre réseau.
3- Entraînez le réseau de neurones : utilisez les outils d'apprentissage de réseau de neurones de MATLAB pour entraîner votre réseau sur les données d'entraînement. Vous pouvez utiliser différentes méthodes d'optimisation et de régularisation pour améliorer les performances du réseau.
4- Évaluez les performances du réseau de neurones : utilisez les données de test pour évaluer les performances du réseau de neurones entraîné et vérifiez sa capacité à généraliser sur de nouvelles données.
5- Utilisez le réseau de neurones pour effectuer des prédictions : une fois entraîné, vous pouvez utiliser le réseau de neurones pour effectuer des prédictions sur de nouvelles données d'entrée.

Il existe également plusieurs fonctions prédéfinies et outils de visualisation dans MATLAB qui vous permettent de faciliter le développement de réseaux de neurones. Vous pouvez également utiliser des modèles pré-entraînés disponibles dans la bibliothèque de réseaux de neurones de MATLAB (par ex. : **nnstart**)ou dans d'autres bibliothèques tierces.

Voici un exemple de programme de base pour créer et entraîner un réseau de neurones dans MATLAB :

```
% Chargement des données d'entraînement et de test
[xTrain, tTrain] = loadTrainingData;
[xTest, tTest] = loadTestData;

% Définition de la structure du réseau de neurones
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);

% Configuration de l'apprentissage du réseau
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Entraînement du réseau de neurones
[net,tr] = train(net,xTrain,tTrain);

% Évaluation des performances du réseau de neurones sur les données de test
y = net(xTest);
perf = perform(net,tTest,y);

% Affichage de la performance du réseau de neurones
fprintf('Performance du réseau de neurones sur les données de test :
%.2f%%\n', perf);

% Affichage de la courbe d'apprentissage du réseau de neurones
figure;
plotperform(tr);

% Affichage de la matrice de confusion du réseau de neurones
figure;
plotconfusion(tTest,y);

% Utilisation du réseau de neurones pour effectuer des prédictions sur de
nouvelles données
xNew = loadNewData;
yNew = net(xNew);
```

Ce programme effectue les étapes suivantes :

1. Chargement des données d'entraînement et de test à partir de fonctions de chargement de données personnalisées.
2. Création d'un réseau de neurones de type **feedforward** avec une couche cachée de 10 neurones à l'aide de la fonction **fitnet**.
3. Configuration de l'apprentissage du réseau en définissant le ratio d'utilisation des données d'entraînement, de validation et de test à l'aide des propriétés **divideParam**.
4. Entraînement du réseau de neurones à l'aide de la fonction train.
5. Évaluation des performances du réseau de neurones sur les données de test à l'aide de la fonction **perform**.
6. Affichage de la performance du réseau de neurones en pourcentage.
7. Affichage de la courbe d'apprentissage du réseau de neurones à l'aide de la fonction **plotperform**.
8. Affichage de la matrice de confusion du réseau de neurones à l'aide de la fonction **plotconfusion**.
9. Chargement de nouvelles données et utilisation du réseau de neurones entraîné pour effectuer des prédictions sur ces données à l'aide de l'opérateur ().

Vous pouvez adapter ce programme en modifiant la structure du réseau de neurones, en utilisant des fonctions d'activation différentes ou en modifiant les paramètres d'apprentissage.

II.7 Conclusion

Un réseau de neurones est un modèle de traitement de l'information inspiré de la structure du cerveau humain. Il est constitué de nombreux neurones artificiels reliés entre eux, qui sont capables de traiter de l'information de manière similaire à ceux du cerveau humain.

Les réseaux de neurones sont utilisés dans de nombreuses applications de l'apprentissage automatique, comme la reconnaissance de formes et de mots, la traduction automatique, la reconnaissance de la parole et la détection de spam. Ils sont également utilisés dans la commande de systèmes automatisés, comme les robots mobiles et les véhicules autonomes.

Il existe plusieurs types de réseaux de neurones, qui varient en fonction de la structure des neurones et de la manière dont ils sont reliés entre eux. Les réseaux de neurones sont généralement entraînés à partir de données d'entraînement en ajustant les poids des connexions entre les neurones de manière à minimiser l'erreur de prédiction du réseau.

Le développement de réseaux de neurones a été l'un des principaux moteurs de l'apprentissage automatique au cours des dernières décennies et il a conduit à de nombreuses avancées dans divers domaines, comme la vision par ordinateur, la reconnaissance de la parole, la traduction automatique et les applications AI (artificiel intelligence).

Conclusion générale

Les techniques d'intelligence artificielle basées sur les réseaux de neurones et la logique floue ont un impact considérable sur de nombreux domaines. Elles sont utilisées pour résoudre des problèmes complexes et effectuer des tâches qui nécessitent une grande quantité de traitement de données et de reconnaissance de motifs.

Les réseaux de neurones sont une technique d'apprentissage automatique qui peut être utilisée pour classer des données, prévoir des événements futurs et détecter des anomalies. Ils sont particulièrement utiles pour les tâches de reconnaissance de l'image et de l'audio, ainsi que pour la prédiction de séries chronologiques.

La logique floue est une technique utilisée pour modéliser et résoudre des problèmes qui sont difficiles à représenter de manière précise en utilisant des règles de logique classiques. Elle permet de traiter les incertitudes et les données imprécises de manière plus naturelle, en utilisant des concepts tels que "très grand" ou "un peu". La logique floue est souvent utilisée pour la commande et le contrôle de systèmes complexes, ainsi que pour la prise de décision dans des situations où il y a beaucoup de variables incertaines.

En résumé, les techniques d'intelligence artificielle basées sur les réseaux de neurones et la logique floue ont un grand potentiel pour résoudre des problèmes complexes et effectuer des tâches difficiles, et elles ont déjà été mises en œuvre avec succès dans de nombreux domaines allant de la reconnaissance de la parole et de l'image à la commande et au contrôle de systèmes industriels.

Références bibliographiques

[1] F. J. Leon and J. M. Corchado, "Intelligent Control Systems Using Soft Computing Methodologies," Wiley, 2002.

[2] B. De Schutter and J. P. Paci, "Intelligent Control Systems: An Introduction with Examples," Springer, 2014.

[3] L. Zadeh, "Fuzzy sets," Information and Control, vol. 8, pp. 338-353, 1965.

[4] J. F. Bonnefon, M. Benrejeb, and J. Haggège, "Les systèmes flous et leur rôle dans l'automatique," Automatique et Traitement du Signal, vol. 25, pp. 3-24, 2018.

[5] P. Borne, M. Benrejeb, and J. Haggège, Les réseaux de neurons: Présentation et applications, Technip, Collection : Méthodes et pratiques de l'ingénieur, 2007.

[6] G. J. Klir and B. Yuan, "Fuzzy Logic: The Foundations," Prentice Hall, 1995.

[7] D. Dubois and H. Prade, Logique floue et systèmes flous, Hermès, 1994.

[8] I. Borne, Introduction à la commande floue, Technip, 1998.

[9] G. J. Klir and B. Yuan, Fuzzy Sets and Fuzzy Logic: Theory and Applications, Prentice Hall; 1st edition, 1995.

[10] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," MIT Press, 2016.

[11] S. Raschka, Intelligence artificielle : comprendre les réseaux de neurones, Eyrolles, 2018.

[12] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning: comprendre les réseaux de neurones profonds," O'Reilly, 2017.

[13] M. Nielsen, Apprendre avec les réseaux de neurones, O'Reilly, 2015.

[14] G. Dreyfus, M. Samuelides, J.-M. Martinez, M. B. Gordon, F. Badran, S. Thiria, and L. Hérault, Réseaux de neurones : Méthodologie et applications, Eyrolles (2e édition), 2004.