Université Echahid Cheikh Larbi Tebessi – Tébessa

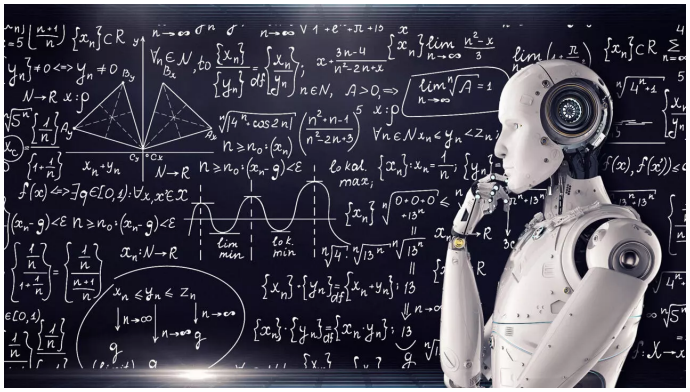
Faculté des Sciences et de la Technologie

Département de Génie Électrique

Support de cours

Techniques de l'Intelligence Artificielle

Destiné aux étudiants en Génie Électrique



Préparé par

Dr. Djamel OUNNAS

AN : 2025/2026

Résumé

Ce polycopié propose une présentation générale des techniques d'intelligence artificielle et de leurs principales approches. Il aborde des méthodes permettant de traiter des problèmes complexes caractérisés par l'incertitude, l'imprécision et la non-linéarité. Les concepts introduits visent à fournir une compréhension globale des outils de l'intelligence artificielle et à faciliter leur utilisation dans divers domaines d'application. Destiné aux étudiants de Master, ce support constitue une base pédagogique pour l'apprentissage et l'approfondissement des approches intelligentes modernes.

Table des matières

Introduction

L'intelligence artificielle occupe aujourd'hui une place essentielle dans l'évolution des systèmes électrotechniques modernes. L'augmentation de la complexité des procédés industriels, la présence d'incertitudes, de non-linéarités et de perturbations rendent souvent insuffisantes les approches classiques basées uniquement sur des modèles mathématiques précis. Dans ce contexte, les techniques d'intelligence artificielle offrent des outils complémentaires et efficaces pour l'analyse, la modélisation, la commande et l'optimisation des systèmes électrotechniques.

Ce polycopié est destiné aux étudiants de Master 2 en Électrotechnique et vise à introduire de manière progressive et appliquée les principales techniques d'intelligence artificielle utilisées en ingénierie électrique. L'accent est mis sur des méthodes capables de traiter l'imprécision et l'incertitude, tout en restant adaptées aux contraintes pratiques des systèmes réels. Les notions présentées sont accompagnées d'exemples et d'applications orientées vers la commande intelligente, l'identification des systèmes non linéaires et l'optimisation des performances.

Le contenu du cours est organisé de façon à fournir une vision globale et cohérente des approches intelligentes les plus couramment utilisées. Il couvre la logique floue, les réseaux de neurones artificiels, les systèmes hybrides neuro-flous ainsi que les algorithmes évolutionnaires. Le cours intègre également le raisonnement probabiliste et les réseaux bayésiens pour la gestion de l'incertitude, ainsi que les systèmes experts et les systèmes experts flous dédiés à la prise de décision et au diagnostic. L'ensemble de ces outils est présenté comme des moyens pratiques permettant d'améliorer la robustesse, l'adaptabilité et l'efficacité des systèmes modernes.

L'objectif principal de ce polycopié est de doter l'étudiant des bases nécessaires à la compréhension et à la mise en œuvre des techniques d'intelligence artificielle dans des applications concrètes, tout en constituant une ouverture vers des travaux de recherche ou de développement industriel dans le domaine de l'énergie et de la commande intelligente.

Chapitre 1

Introduction à l'intelligence artificielle

1 Qu'est-ce que l'intelligence artificielle ?

L'intelligence artificielle (IA) est une discipline fascinante, mais également difficile à définir de manière univoque. Le terme même d'« intelligence » fait encore l'objet de débats en psychologie, en neurosciences et en philosophie, si bien qu'il n'existe pas de consensus clair sur la définition de l'« intelligence artificielle ».

De nombreuses définitions ont été proposées au fil du temps, chacune mettant l'accent sur un aspect particulier. En voici quelques-unes issues de la littérature :

> *« L'étude des facultés mentales à l'aide de modèles computationnels. »* (Charniak et McDermott, 1985)
>
> *« La conception d'agents intelligents. »* (Poole et al., 1998)
>
> *« La discipline étudiant la possibilité de faire exécuter par un ordinateur des tâches pour lesquelles l'homme est aujourd'hui meilleur que la machine. »* (Rich et Knight, 1990)
>
> *« L'automatisation des activités associées au raisonnement humain, telles que la prise de décision, la résolution de problèmes et l'apprentissage. »* (Bellman, 1978)
>
> *« L'étude des mécanismes permettant à un agent de percevoir, de raisonner et d'agir. »* (Winston, 1992)
>
> *« L'étude des entités ayant un comportement intelligent. »* (Nilsson, 1998)

Ces définitions s'accordent sur un objectif principal : concevoir des systèmes intelligents. Elles diffèrent toutefois dans la manière dont elles définissent l'intelligence elle-même. Certaines approches se focalisent sur le *comportement observable* du système (agir comme un humain), tandis que d'autres s'intéressent au *raisonnement interne* (penser comme un humain).

Ainsi, on distingue traditionnellement quatre grandes façons de concevoir l'intelligence artificielle :

1. **Créer des systèmes qui se comportent comme les êtres humains.** Cette définition opérationnelle a été proposée par Alan Turing, qui introduisit

le célèbre *test de Turing*. Selon ce test, une machine peut être considérée comme intelligente si, au cours d'une conversation, un interlocuteur humain ne peut la distinguer d'un autre être humain.

2. **Créer des systèmes qui pensent comme les êtres humains.** Cette approche vise à reproduire les processus mentaux humains. L'IA devient alors une science expérimentale : pour créer des machines pensantes, il est nécessaire de comprendre au préalable les mécanismes de la pensée humaine.
3. **Créer des systèmes qui pensent rationnellement.** L'objectif est ici de concevoir des machines capables de raisonner conformément aux règles de la logique formelle. Bien que rigoureuse, cette approche présente certaines limites, car des aspects essentiels de l'intelligence, tels que la perception ou l'intuition, se prêtent difficilement à une formalisation logique stricte.
4. **Créer des systèmes qui agissent rationnellement.** Dans cette perspective plus moderne, un système est dit « intelligent » s'il agit de manière rationnelle, c'est-à-dire s'il cherche à maximiser ses chances d'atteindre ses objectifs. Cette approche, fondée sur la notion d'*agent intelligent*, est aujourd'hui dominante.

En pratique, ces distinctions ne sont pas strictement séparées, et la recherche en intelligence artificielle combine souvent plusieurs de ces approches selon le problème étudié.

2 Brève histoire de l'intelligence artificielle

L'histoire de l'intelligence artificielle s'étend sur plus de sept décennies et peut être découpée en plusieurs grandes périodes.

2.1 Les origines (1943–1955)

Les premiers travaux sur les neurones artificiels furent menés par McCulloch et Pitts en 1943, qui proposèrent un modèle logique du neurone. Peu après, Hebb (1949) formula une règle d'apprentissage inspirée du fonctionnement du cerveau humain. En 1950, Alan Turing publia un article fondateur dans lequel il introduisit le *test de Turing*, posant ainsi les bases philosophiques de l'intelligence artificielle.

2.2 La naissance de l'IA (1956)

En 1956, lors de la conférence de Dartmouth, le terme « intelligence artificielle » fut proposé pour la première fois. Cet événement est généralement considéré comme la naissance officielle du domaine. Les chercheurs nourrissaient alors l'espoir de pouvoir reproduire rapidement l'intelligence humaine.

2.3 Les années d'espoir (1956–1969)

Cette période fut marquée par un enthousiasme considérable. De nombreux programmes furent développés, tels que le *Logic Theorist* de Newell et Simon, capable

de démontrer des théorèmes de logique, ou encore le *General Problem Solver*, qui proposait un cadre général de résolution de problèmes. Des robots comme *Shakey* virent également le jour, capables de raisonner sur leurs propres actions.

2.4 Les premières déceptions (1970–1980)

Les limites technologiques de l'époque, notamment le manque de puissance de calcul et la mémoire restreinte, entraînèrent une période de désillusion. Le rapport Lighthill (1973) mit en évidence les faiblesses des approches de l'IA, provoquant une réduction significative des financements dans plusieurs pays.

2.5 Les systèmes experts (1980–1990)

L'intelligence artificielle connut un renouveau grâce aux systèmes experts, tels que *DENDRAL* en chimie et *MYCIN* pour le diagnostic médical. Ces systèmes reposaient sur des règles heuristiques élaborées à partir des connaissances d'experts humains et obtinrent des résultats remarquables dans des domaines spécifiques.

2.6 Le retour des réseaux de neurones (à partir de 1986)

La redécouverte de l'algorithme de rétropropagation du gradient (*backpropagation*) permit de relancer la recherche sur les réseaux de neurones artificiels. Cette avancée marque le point de départ de l'apprentissage automatique moderne.

2.7 L'ère du deep learning (2010–aujourd'hui)

Avec l'augmentation spectaculaire des capacités de calcul et la disponibilité de données massives, les réseaux de neurones profonds ont permis des avancées majeures dans de nombreux domaines : reconnaissance d'images, traduction automatique, jeux, diagnostic médical, véhicules autonomes, etc.

3 Les sous-domaines de l'intelligence artificielle

L'intelligence artificielle s'est progressivement structurée en plusieurs sous-domaines, chacun s'intéressant à une facette particulière de l'intelligence :

- **Représentation des connaissances et raisonnement automatique :** étude des méthodes permettant de représenter le savoir sous une forme exploitable par une machine, ainsi que des mécanismes de raisonnement associés.
- **Traitement automatique du langage naturel (TALN) :** conception de systèmes capables de comprendre, de traduire ou de produire du langage humain.
- **Vision par ordinateur :** analyse d'images et de vidéos afin de reconnaître des objets, des visages ou des scènes.
- **Apprentissage automatique (*machine learning*) :** développement d'algorithmes permettant aux systèmes d'apprendre à partir de données et d'améliorer leurs performances avec l'expérience.

— **Robotique :** intégration de la perception, de la planification et de l'action au sein d'agents physiques capables d'interagir avec le monde réel.

Ces domaines sont fortement interconnectés. Par exemple, la reconnaissance d'images en vision par ordinateur repose largement sur des techniques d'apprentissage profond.

4 L'état actuel de l'intelligence artificielle

Aujourd'hui, l'intelligence artificielle est omniprésente dans la vie quotidienne :

— les systèmes de recommandation (Netflix, YouTube) personnalisent les contenus proposés aux utilisateurs ;
— les assistants vocaux (*Siri*, *Google Assistant*, *Alexa*) comprennent et produisent du langage naturel ;
— les véhicules autonomes s'appuient sur des techniques avancées de perception et de planification ;
— en médecine, les réseaux de neurones assistent le diagnostic à partir d'images radiologiques ;
— dans l'industrie, l'IA permet la maintenance prédictive et l'optimisation des processus énergétiques.

Chapitre 2

Généralités sur le Soft Computing

1 Introduction

Depuis ses débuts, l'intelligence artificielle cherche à doter les machines de capacités similaires à celles de l'esprit humain : percevoir, raisonner, apprendre et décider. Les premières approches, dites de **Hard Computing**, s'appuyaient sur une logique stricte et des modèles mathématiques rigoureux. Cependant, ces méthodes se sont rapidement révélées insuffisantes pour modéliser les phénomènes du monde réel, souvent ambigus, bruités ou incomplets.

Afin de pallier ces limites, une nouvelle approche, plus souple et plus tolérante à l'incertitude, a vu le jour : le **Soft Computing**, également appelé *calcul souple*. Popularisé dans les années 1990 par Lotfi A. Zadeh, père de la logique floue, le Soft Computing regroupe un ensemble de techniques inspirées du raisonnement humain. Il repose sur le principe fondamental suivant : *« Mieux vaut une réponse approximative à un problème complexe qu'une réponse exacte à un problème simplifié. »*

2 Philosophie et objectifs du Soft Computing

Contrairement au **Hard Computing**, qui requiert des données précises ainsi que des modèles mathématiques parfaitement définis, le **Soft Computing** accepte l'imprécision, l'incertitude et la non-linéarité comme des caractéristiques intrinsèques des systèmes réels. Cette approche est particulièrement adaptée aux systèmes complexes pour lesquels une modélisation exacte est difficile, voire impossible.

L'objectif principal du Soft Computing n'est pas de garantir une optimalité mathématique stricte, mais de fournir des solutions **robustes, adaptatives et satisfaisantes**, capables de fonctionner efficacement dans des conditions réelles. Il privilégie ainsi la qualité globale du comportement du système plutôt que la précision absolue du modèle.

Cette philosophie s'inspire directement du raisonnement humain : l'être humain ne raisonne pas toujours selon des règles logiques formelles, mais il est néanmoins capable de prendre des décisions efficaces à partir d'informations partielles, imprécises ou bruitées. Dans cette optique, le Soft Computing vise à doter les systèmes artificiels d'une forme d'« intelligence approximative », leur permettant de raisonner et d'agir de manière pertinente dans des environnements complexes, incertains et

dynamiques, tels que ceux rencontrés en électrotechnique (commande, régulation, diagnostic et optimisation).

3 Les composantes fondamentales du Soft Computing

Le **Soft Computing** ne constitue pas une technique unique, mais plutôt un ensemble cohérent de méthodes complémentaires, conçues pour traiter des problèmes complexes, incertains ou non linéaires. Parmi les composantes fondamentales du Soft Computing, on distingue notamment :

3.1 La logique floue (*Fuzzy Logic*)

Introduite par Lotfi A. Zadeh en 1965, la logique floue permet de représenter et de manipuler des notions imprécises issues du langage naturel. Alors que la logique classique repose sur des valeurs binaires — vrai (1) ou faux (0) —, la logique floue introduit des **degrés de vérité** continus, compris entre 0 et 1.

Par exemple, au lieu d'affirmer de manière catégorique que « la température est chaude », la logique floue permet d'exprimer cette notion sous la forme :

$$\mu_{\text{chaud}}(30^\circ\text{C}) = 0.7$$

ce qui signifie que la température de 30°C est considérée comme « chaude » avec un degré d'appartenance de 70%.

Cette approche est particulièrement adaptée à la commande et à la régulation de systèmes physiques, tels que les systèmes de climatisation, les moteurs électriques ou les procédés industriels, où les transitions sont progressives et non abruptes.

3.2 Les réseaux de neurones artificiels (*Artificial Neural Networks*)

Inspirés du fonctionnement biologique du cerveau, les réseaux de neurones artificiels sont des modèles mathématiques capables d'apprendre à partir d'exemples. Ils sont constitués de nœuds élémentaires, appelés neurones artificiels, interconnectés et organisés en couches (entrée, cachées et sortie). Chaque neurone réalise une combinaison pondérée des signaux reçus, suivie de l'application d'une fonction d'activation non linéaire.

Le processus d'apprentissage consiste à ajuster les poids synaptiques afin de minimiser l'erreur entre la sortie du réseau et la sortie désirée, généralement à l'aide de méthodes d'optimisation par gradient. Les réseaux de neurones ont connu un essor considérable grâce à l'algorithme de rétropropagation du gradient, ainsi qu'à l'augmentation des capacités de calcul (GPU) et à la disponibilité de grandes quantités de données.

En électrotechnique, ces modèles sont largement utilisés pour l'identification de systèmes, la commande non linéaire, le diagnostic de défauts et la prédiction de charge.

3.3 Les algorithmes évolutionnaires

Inspirés des mécanismes de l'évolution biologique — tels que la sélection naturelle, le croisement et la mutation —, les algorithmes évolutionnaires explorent un espace de solutions en vue d'optimiser une fonction objectif. Ils se révèlent particulièrement efficaces pour les problèmes d'optimisation complexes, multimodaux ou non différentiables, pour lesquels les méthodes analytiques classiques sont inadaptées.

L'**algorithme génétique**, proposé par Holland en 1975, en est le représentant le plus connu. D'autres variantes incluent la *programmation génétique*, les *algorithmes de colonies de fourmis* et les *essaims particulaires* (*Particle Swarm Optimization*, PSO).

Ces techniques sont couramment utilisées en électrotechnique pour l'optimisation de paramètres de commande, le dimensionnement de machines électriques et la gestion de réseaux énergétiques.

3.4 Les systèmes hybrides

Dans de nombreuses applications réelles, l'utilisation d'une seule technique de Soft Computing s'avère insuffisante. Les systèmes hybrides combinent plusieurs méthodes afin de tirer parti de leurs avantages respectifs et de compenser leurs limitations.

Par exemple, un système **neuro-flou** associe la capacité d'apprentissage des réseaux de neurones à la capacité d'interprétation et de formalisation des connaissances des systèmes flous. Ces approches hybrides sont largement utilisées dans le contrôle industriel, la modélisation prédictive et la commande intelligente de systèmes électrotechniques complexes.

4 Comparaison entre Hard Computing et Soft Computing

Le tableau ci-dessous présente une comparaison synthétique entre les deux paradigmes **Hard Computing** et **Soft Computing**, en mettant en évidence leurs différences fondamentales en termes d'approche, de modélisation et d'applications.

5 Avantages et apports du Soft Computing

Le **Soft Computing** présente de nombreux avantages pour la modélisation, l'analyse et la commande des systèmes réels, en particulier lorsque ceux-ci sont complexes, non linéaires ou soumis à des incertitudes :

- il permet de traiter efficacement des données incomplètes, bruitées ou imprécises ;
- il offre une grande capacité d'adaptation et d'apprentissage à partir de l'expérience ;
- il est capable d'approximer des fonctions non linéaires complexes sans nécessiter de modèle analytique précis ;

TABLE 2.1 – Comparaison entre Hard Computing et Soft Computing

Caractéristique	**Hard Computing**	**Soft Computing**
Approche	Logique stricte, déterministe et exacte	Approche approximative, tolérante à l'imprécision
Modèle de raisonnement	Basé sur les mathématiques classiques et la logique formelle	Inspiré du raisonnement humain et des mécanismes adaptatifs
Gestion de l'incertitude	Peu ou pas tolérée	Acceptée et exploitée explicitement
Adaptabilité	Faible ou inexistante	Élevée, grâce à l'apprentissage et à l'auto-adaptation
Robustesse aux perturbations	Sensible aux variations et aux bruits	Robuste face aux perturbations et aux données bruitées
Domaines d'application	Systèmes bien modélisés, contrôle classique, calcul numérique	Systèmes complexes, non linéaires, mal modélisés
Exemples typiques	Méthodes numériques, logique booléenne, commande PID classique	Logique floue, réseaux de neurones, algorithmes évolutionnaires

— il fournit des solutions satisfaisantes et robustes dans des situations où les méthodes classiques échouent ou deviennent difficiles à mettre en œuvre ;
— il facilite la conception de systèmes autonomes, intelligents et auto-adaptatifs.

Grâce à ces caractéristiques, le Soft Computing est aujourd'hui considéré comme un pilier des techniques modernes d'intelligence artificielle, notamment dans les domaines de la commande intelligente et de l'optimisation.

6 Applications du Soft Computing

Les méthodes du Soft Computing ont trouvé de nombreuses applications dans les domaines de l'ingénierie, de l'industrie et de la recherche scientifique, en particulier en électrotechnique et en automatique.

6.1 Commande intelligente

Les contrôleurs flous et neuro-flous sont largement utilisés dans l'automatisation industrielle, la robotique, les systèmes de climatisation, les entraînements électriques et les véhicules autonomes. Ils permettent un pilotage souple, robuste et adaptatif, même en présence d'incertitudes, de perturbations externes ou de variations paramétriques.

6.2 Prédiction et classification

Les réseaux de neurones artificiels et les algorithmes évolutionnaires sont employés pour la prédiction de séries temporelles (charge électrique, consommation

énergétique), la détection et le diagnostic de défauts, ainsi que la reconnaissance de formes et de signaux.

6.3 Optimisation et planification

Les algorithmes évolutionnaires offrent des solutions efficaces à des problèmes d'optimisation complexes et multi-objectifs, tels que l'optimisation de paramètres de commande, la planification de la production, l'allocation de ressources ou la gestion des réseaux électriques.

6.4 Systèmes décisionnels

En combinant la logique floue avec des mécanismes de raisonnement symbolique ou d'apprentissage, les systèmes décisionnels intelligents assistent la prise de décision dans des environnements complexes et incertains. Ces approches sont utilisées dans des domaines variés tels que la finance, la médecine, l'industrie et la gestion énergétique.

Chapitre 3

Logique floue et ses applications

1 Introduction à la logique floue

La logique classique, également appelée logique booléenne, repose sur des valeurs de vérité binaires : une proposition est soit vraie (1), soit fausse (0). Cette approche est parfaitement adaptée aux systèmes formels et mathématiques. Toutefois, elle montre rapidement ses limites lorsqu'il s'agit de modéliser des situations réelles, souvent imprécises, ambiguës ou subjectives.

Dans la vie quotidienne, le raisonnement humain ne fonctionne pas de manière strictement binaire. Par exemple, des notions telles que *température élevée*, *vitesse lente* ou *personne jeune* ne peuvent pas être décrites de façon précise à l'aide de frontières nettes. La logique floue (*fuzzy logic*), introduite par Lotfi A. Zadeh en 1965, a été développée afin de traiter ce type d'informations imprécises et de modéliser le raisonnement approximatif.

La logique floue généralise la logique classique en autorisant des degrés de vérité continus compris entre 0 et 1. Ainsi, une proposition peut être partiellement vraie et partiellement fausse, ce qui permet une représentation plus fidèle et plus réaliste des phénomènes du monde réel.

1.1 Motivation de la logique floue

L'objectif principal de la logique floue est de rapprocher les systèmes de décision et de contrôle du raisonnement humain. Elle est particulièrement utilisée dans :

— les systèmes de contrôle (climatisation, lave-linge, régulation industrielle),
— l'intelligence artificielle et les systèmes experts,
— le traitement d'images et la reconnaissance de formes,
— l'aide à la décision dans des environnements incertains.

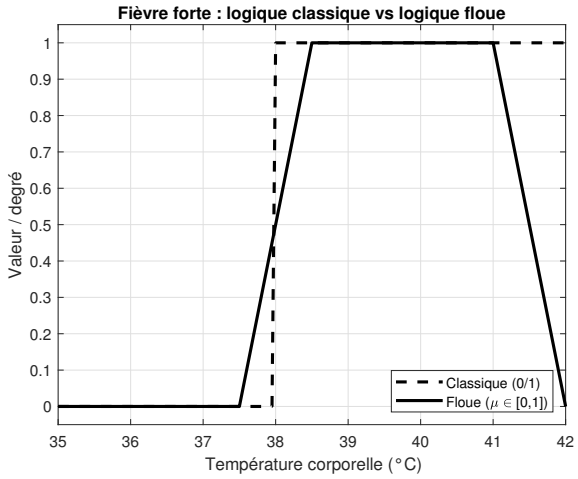


FIGURE 3.1 – Comparaison entre la représentation classique et floue de la fièvre

1.2 Exemples introductifs

Exemple 1 : Température

Considérons la notion de *température élevée*. En logique classique, on pourrait définir :

$$\text{Température élevée} = \begin{cases} 1 & \text{si } T \geq 30°\text{C} \\ 0 & \text{sinon} \end{cases}$$

Cette définition est trop rigide. Une température de 29°C est très proche de 30°C, mais elle est considérée comme non élevée en logique classique.

En logique floue, on associe à chaque température un **degré d'appartenance** $\mu(T) \in [0, 1]$.

Exemple 2 : Vitesse d'un véhicule

La notion de *vitesse lente* dépend fortement du contexte. Une vitesse de 40 km/h peut être :

— lente sur une autoroute,
— normale en milieu urbain,
— rapide dans une zone résidentielle.

La logique floue permet de prendre en compte ce caractère relatif en définissant des ensembles flous adaptés au contexte considéré, contrairement à une approche classique basée sur des seuils fixes.

Exemple 3 : Âge d'une personne

Définir une personne comme *jeune* ou *âgée* par un seuil précis (par exemple 30 ans) est arbitraire. En logique floue, une personne de 25 ans peut être jeune avec un degré de 0.9, tandis qu'une personne de 35 ans peut être jeune avec un degré de 0.4.

Ces exemples illustrent l'intérêt fondamental de la logique floue : offrir un cadre mathématique permettant de modéliser l'imprécision, la gradualité et le raisonnement humain.

1.3 Limites de la logique booléenne

La logique booléenne impose des décisions strictes basées sur des valeurs binaires (vrai ou faux). Cette rigidité devient problématique dans des domaines comme la médecine, où les symptômes et les diagnostics sont rarement parfaitement définis.

Exemple médical : diagnostic de l'hépatite

Un patient atteint d'hépatite présente généralement les symptômes suivants :
— une forte fièvre,
— une coloration jaune de la peau (jaunisse),
— des nausées.
En logique booléenne, chaque symptôme est représenté par une variable binaire :

$$\text{Fièvre} \in \{0,1\}, \quad \text{Jaunisse} \in \{0,1\}, \quad \text{Nausées} \in \{0,1\}$$

Un diagnostic possible peut être formulé par :

$$\text{Hépatite} = \text{Fièvre} \wedge \text{Jaunisse} \wedge \text{Nausées}$$

Cette règle impose que tous les symptômes soient présents pour conclure au diagnostic.

Limites de cette approche

Cette modélisation présente plusieurs limites :
— la fièvre peut être modérée,
— la jaunisse peut être légère,
— les nausées peuvent être intermittentes ou subjectives.
Un patient présentant des symptômes partiels pourrait alors être classé comme non atteint, ce qui est médicalement discutable.

Apport de la logique floue

La logique floue permet d'associer à chaque symptôme un degré de présence compris entre 0 et 1.

Le diagnostic devient alors graduel et peut exprimer un *degré de suspicion* de l'hépatite.

Cette approche est plus proche du raisonnement médical réel et constitue une base efficace pour les systèmes d'aide à la décision.

2 Bref historique

2.1 Naissance du concept (années 1960)

La logique floue a été introduite en 1965 par le Professeur Lotfi A. Zadeh, de l'Université de Californie à Berkeley. Dans son article fondateur intitulé *"Fuzzy Set Theory"*, Zadeh propose une nouvelle approche mathématique permettant de représenter des concepts imprécis à l'aide des *ensembles flous*.

Cette théorie introduit :

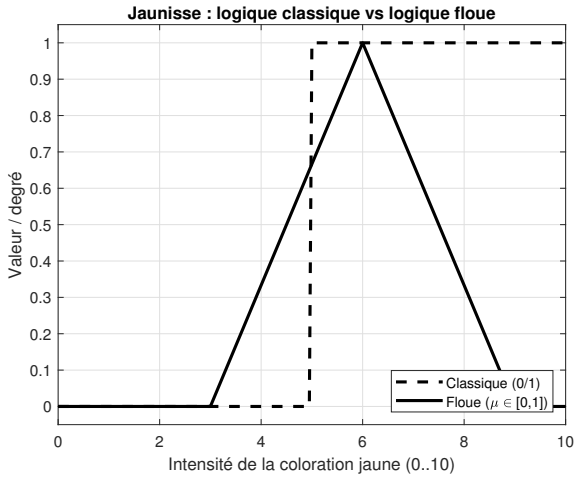


FIGURE 3.2 – Représentation classique et floue de la jaunisse

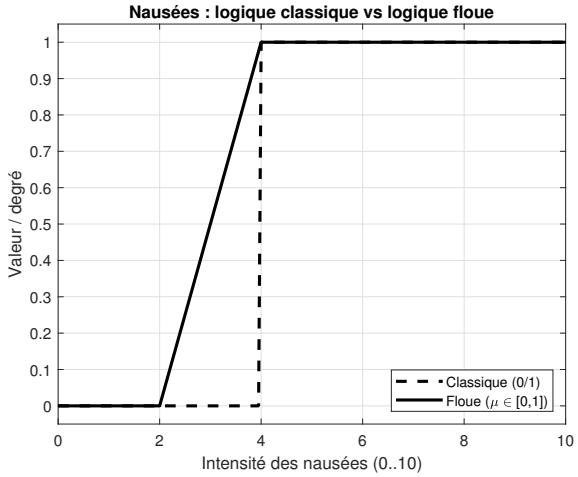


FIGURE 3.3 – Représentation classique et floue des nausées

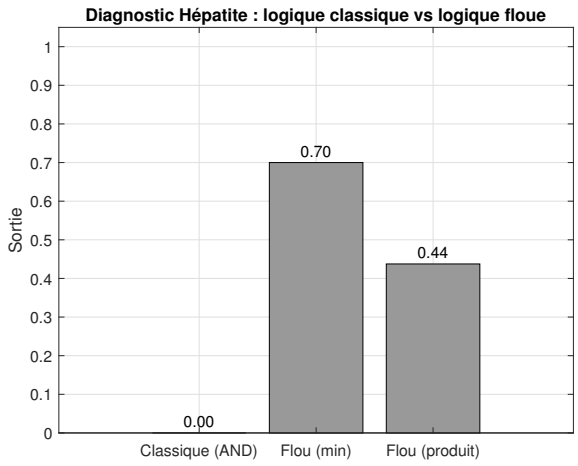


FIGURE 3.4 – Comparaison du diagnostic d'hépatite en logique classique et en logique floue

— la notion d'ensemble flou,
— les fonctions d'appartenance,
— les opérateurs flous associés (ET, OU, NON).

Ces travaux constituent la base théorique de la logique floue moderne.

2.2 Premières applications (années 1970)

Dès les années 1970, la logique floue commence à être utilisée dans des applications concrètes, notamment dans :

— les systèmes experts,
— l'aide à la décision en médecine,
— les applications commerciales et financières.

Ces premières applications démontrent l'intérêt de la logique floue pour traiter des problèmes complexes faisant intervenir des données imprécises ou qualitatives.

2.3 Première application industrielle (1974)

En 1974, Ebrahim H. Mamdani réalise la première application industrielle majeure de la logique floue : la régulation floue d'une chaudière à vapeur.

Cette réalisation marque une étape décisive, en montrant que la logique floue peut être utilisée efficacement pour la commande de systèmes réels complexes, sans modèle mathématique précis.

2.4 Une longue période académique

Pendant plusieurs années, la logique floue reste principalement cantonnée au milieu universitaire et à la recherche académique. Malgré ses performances, son adoption industrielle demeure limitée, en raison notamment :

— du scepticisme initial de la communauté scientifique,
— du manque de moyens matériels dédiés,
— de la nouveauté du paradigme.

2.5 Essor industriel et grand public (années 1980–1990)

À partir de 1985, les industriels japonais jouent un rôle majeur dans la diffusion de la logique floue en introduisant des produits grand public portant la mention *"Fuzzy Logic Inside"*.

À partir des années 1990, l'utilisation de la logique floue se généralise dans de nombreux domaines :

— appareils électroménagers (lave-linge, aspirateurs, autocuiseurs, etc.),
— systèmes audio-visuels (appareils photo autofocus, caméscopes avec stabilisation d'image, photocopieurs),
— systèmes automobiles embarqués (boîte de vitesses automatique, ABS, suspension, climatisation),
— systèmes autonomes mobiles,
— systèmes de décision, de diagnostic et de reconnaissance,
— systèmes de contrôle et de commande dans la plupart des secteurs industriels.

2.6 Matériel dédié et implémentations matérielles

Avec la généralisation des applications, des processeurs dédiés à la logique floue et des interfaces de développement spécifiques ont été conçus, tels que le microcontrôleur **68HC12** de Motorola.

Un exemple notable est la famille de processeurs **WARP** (*Weight Associative Rule Processor*) développée par **SGS-THOMSON**, dont les principales caractéristiques sont :

— nombre de règles traitées : 256,
— nombre d'entrées : 16,
— nombre de sorties : 16,
— méthode de composition des règles : centre de gravité,
— vitesse de traitement : 200 microsecondes pour 200 règles.

Ces développements matériels ont fortement contribué à l'intégration de la logique floue dans les systèmes temps réel et embarqués.

3 Concepts principaux de la logique floue

La logique floue repose sur un ensemble de concepts fondamentaux permettant de représenter l'imprécision, de raisonner à partir de connaissances linguistiques et de prendre des décisions dans des environnements incertains. Les deux piliers essentiels de cette approche sont les ensembles flous et le mécanisme de prise de décision basé sur des règles floues.

3.1 Ensembles et variables flous

Ensembles flous

Un *ensemble flou* est une généralisation de la notion classique d'ensemble. Contrairement à un ensemble classique, où un élément appartient ou n'appartient pas à un ensemble, un ensemble flou autorise une appartenance graduelle.

Soit X un univers de discours. Un ensemble flou A sur X est défini par une fonction d'appartenance :

$$\mu_A : X \to [0, 1]$$

où $\mu_A(x)$ représente le degré d'appartenance de l'élément x à l'ensemble flou A.

Variables floues

Une *variable floue* est une variable dont les valeurs sont des termes linguistiques représentés par des ensembles flous. Par exemple, la variable floue *Température* peut prendre des valeurs telles que :

$$\text{Température} \in \{\text{froide}, \text{modérée}, \text{chaude}\}$$

Chaque valeur linguistique est associée à une fonction d'appartenance définie sur l'univers de discours considéré.

Opérateurs flous

Les opérateurs flous généralisent les opérateurs logiques classiques :

— **ET flou** (conjonction) : généralement modélisé par l'opérateur minimum

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

— **OU flou** (disjonction) : généralement modélisé par l'opérateur maximum

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

— **NON flou** (négation) :

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

D'autres opérateurs (produit, somme bornée, etc.) peuvent également être utilisés selon l'application.

3.2 Prise de décision à partir d'une base de règles floues

L'un des aspects les plus importants de la logique floue est sa capacité à prendre des décisions à partir de connaissances exprimées sous forme de règles linguistiques.

Règles floues de type SI–ALORS

Une règle floue est généralement exprimée sous la forme :

$$\text{SI } x \text{ est } A \text{ ET } y \text{ est } B \text{ ALORS } z \text{ est } C$$

où A, B et C sont des ensembles flous.

Une base de règles floues est constituée d'un ensemble de règles de ce type, représentant l'expertise humaine dans le domaine considéré.

Inférence floue

L'inférence floue est le mécanisme qui permet de déduire une décision ou une conclusion à partir :

— des valeurs d'entrée,
— des fonctions d'appartenance,
— de la base de règles floues.

Le processus d'inférence floue comprend généralement les étapes suivantes :

1. **Fuzzification** : transformation des valeurs numériques d'entrée en degrés d'appartenance flous.
2. **Évaluation des règles** : calcul du degré d'activation de chaque règle à l'aide des opérateurs flous.
3. **Agrégation** : combinaison des conclusions de toutes les règles actives.
4. **Défuzzification** : transformation du résultat flou en une valeur numérique exploitable (par exemple par la méthode du centre de gravité).

Ce mécanisme permet d'obtenir une décision progressive et robuste, contrairement aux systèmes classiques basés sur des seuils stricts.

Grâce aux ensembles flous, aux variables linguistiques et à l'inférence floue, la logique floue constitue un cadre puissant pour la modélisation et la résolution de problèmes complexes. Elle permet d'intégrer le raisonnement humain dans les systèmes informatiques tout en conservant une base mathématique rigoureuse.

4 Le concept d'ensemble flou

Les ensembles flous constituent le fondement théorique de la logique floue. Ils permettent de représenter des concepts imprécis ou graduels, en généralisant la notion classique d'ensemble.

4.1 Univers du discours

Soit U l'univers du discours, c'est-à-dire l'ensemble de toutes les valeurs possibles que peut prendre une variable considérée. Par exemple :

— pour la température : $U = [0, 50]°\text{C}$,
— pour l'âge : $U = [0, 100]$ ans,
— pour la vitesse : $U = [0, 200]$ km/h.

4.2 Définition d'un ensemble flou

Un ensemble flou F défini sur l'univers du discours U est totalement déterminé par sa fonction d'appartenance :

$$\mu_F : U \to [0, 1]$$

Pour tout $x \in U$, $\mu_F(x)$ représente le degré selon lequel x appartient à l'ensemble flou F.

4.3 Exemple : Ensemble flou *Température élevée*

Soit $U = [0, 50]°\text{C}$. L'ensemble flou *Température élevée* peut être représenté par une fonction d'appartenance trapézoïdale, comme illustré à la Figure 3.5.

4.4 Caractérisation par les α-coupes

Une partie floue F peut également être caractérisée par l'ensemble de ses α-coupes. Soit $\alpha \in [0, 1]$, la α-coupe F_α est définie par :

$$F_\alpha = \{x \in U \mid \mu_F(x) \geq \alpha\}$$

La Figure 3.6 illustre plusieurs α-coupes de l'ensemble flou *Température élevée*.

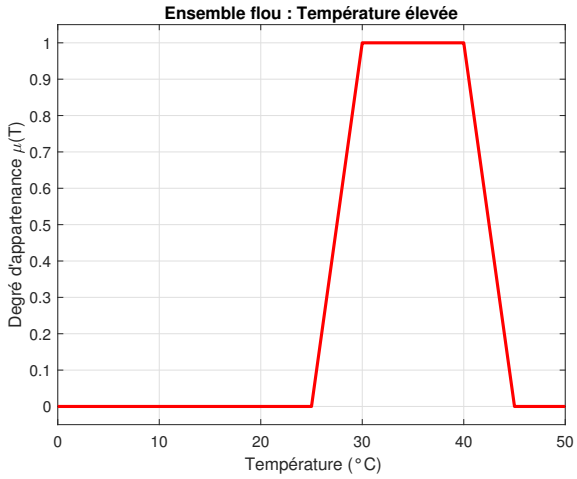


FIGURE 3.5 – Ensemble flou *Température élevée* et sa fonction d'appartenance

4.5 Autres exemples d'ensembles flous

Exemple : Ensemble flou *Personne jeune*

Soit $U = [0, 100]$ ans. L'ensemble flou *Jeune* permet de représenter de manière progressive la notion de jeunesse. La fonction d'appartenance correspondante est illustrée à la Figure 3.7.

Exemple : Ensemble flou *Vitesse lente* selon le contexte

La notion de *vitesse lente* dépend fortement du contexte. La Figure 3.8 compare deux ensembles flous correspondant à un environnement urbain et autoroutier.

Le concept d'ensemble flou offre une représentation mathématique puissante de la gradualité et de l'imprécision. Il constitue la base de la définition des variables floues, des règles linguistiques et des mécanismes d'inférence floue utilisés dans les systèmes d'aide à la décision et de commande.

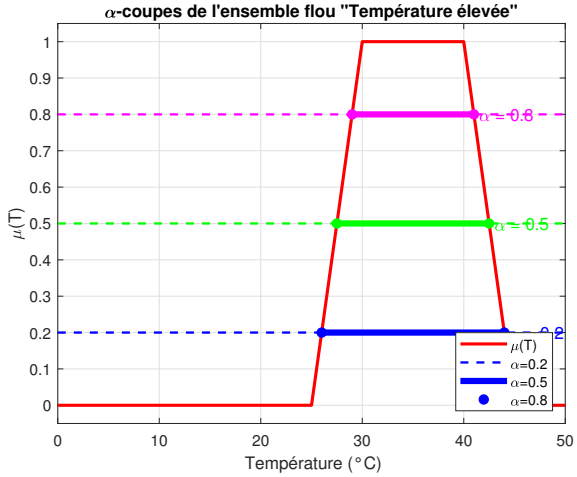


FIGURE 3.6 – Illustration des α-coupes de l'ensemble flou *Température élevée*

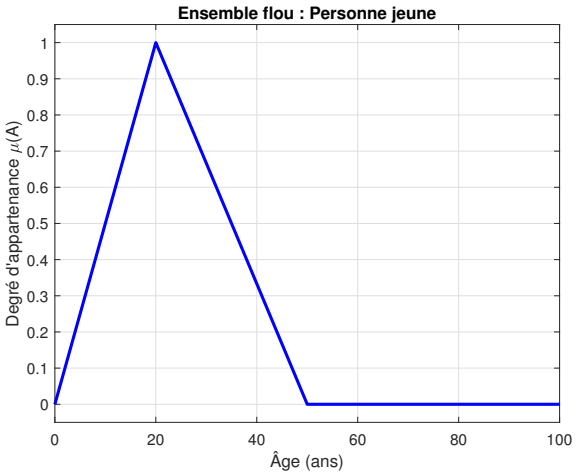


FIGURE 3.7 – Ensemble flou *Personne jeune*

5 Opérateurs de logique floue

Les opérateurs de logique floue généralisent les opérations classiques sur les ensembles (complément, intersection, union) afin de permettre le raisonnement avec des concepts imprécis. Ils sont définis à partir des fonctions d'appartenance des ensembles flous et constituent la base de l'évaluation des règles floues.

Soient A et B deux ensembles flous définis sur un même univers du discours U, caractérisés par leurs fonctions d'appartenance $\mu_A(x)$ et $\mu_B(x)$.

5.1 Partition floue de l'univers du discours

On considère l'exemple de la variable linguistique *Taille (m)*. L'univers du discours est partitionné en trois ensembles flous : *Petit*, *Moyen* et *Grand*.

Cette partition permet une transition progressive entre les différentes catégories linguistiques, contrairement à une classification classique par seuils stricts.

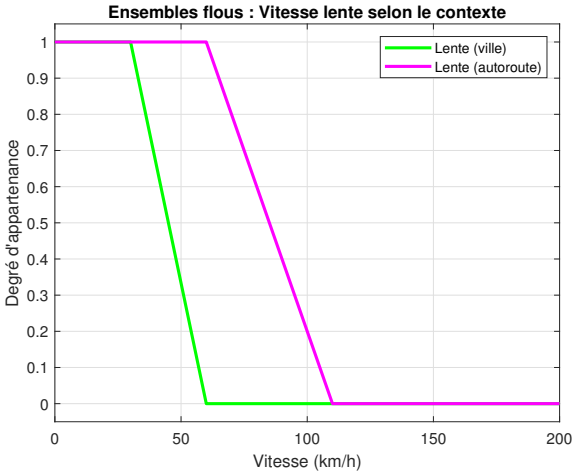


FIGURE 3.8 – Ensembles flous *Vitesse lente* selon le contexte (ville et autoroute)

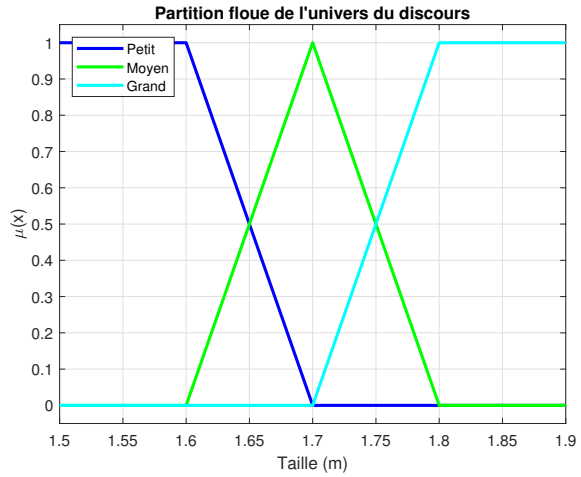


FIGURE 3.9 – Partition floue de l'univers du discours pour la variable *Taille (m)*

5.2 Complément (NON flou)

Le complément d'un ensemble flou A, noté $\bar{A}$, est défini par :

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Dans cet exemple, le complément de l'ensemble flou *Petit* correspond à l'ensemble flou *Personnes non petites*.

Cette représentation permet d'exprimer progressivement la négation d'un concept linguistique.

5.3 Union (OU flou)

L'union de deux ensembles flous A et B, notée $A \cup B$, est généralement définie par l'opérateur maximum :

$$\mu_{A \cup B}(x) = \max\left(\mu_A(x), \mu_B(x)\right)$$

La Figure 3.11 illustre l'union floue des ensembles *Petit* et *Moyen*.

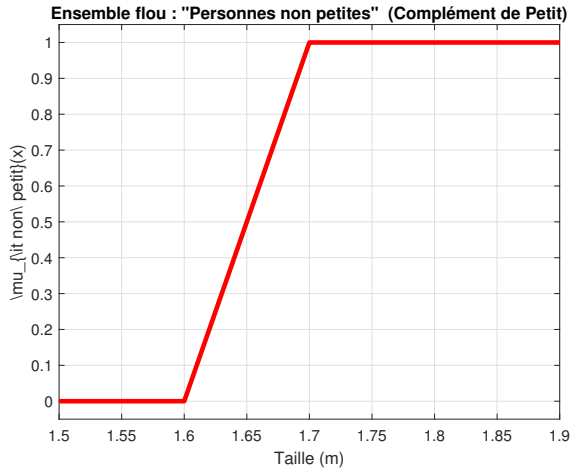


FIGURE 3.10 – Complément de l'ensemble flou *Petit* : ensemble flou *Non petit*

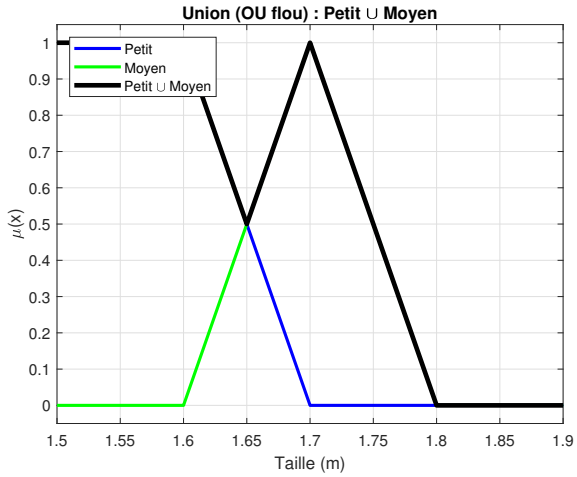


FIGURE 3.11 – Union (OU flou) des ensembles *Petit* et *Moyen*

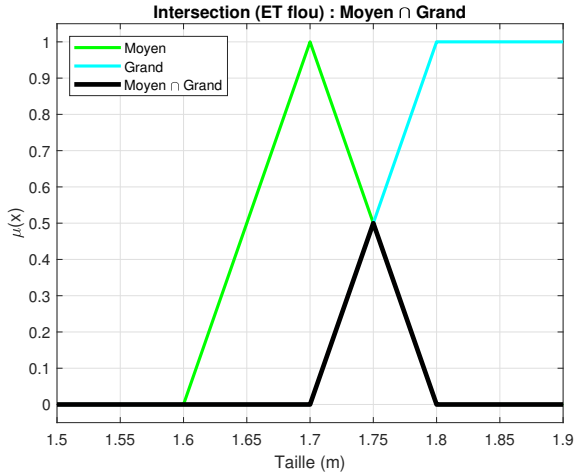


FIGURE 3.12 – Intersection (ET flou) des ensembles *Moyen* et *Grand*

L'union floue permet de modéliser la disjonction logique en tenant compte des degrés d'appartenance.

5.4 Intersection (ET flou)

L'intersection de deux ensembles flous A et B, notée $A \cap B$, est généralement définie par l'opérateur minimum :

$$\mu_{A \cap B}(x) = \min\left(\mu_A(x), \mu_B(x)\right)$$

La Figure 3.12 montre l'intersection floue des ensembles *Moyen* et *Grand*.

Cette opération met en évidence les zones de recouvrement entre concepts linguistiques.

TABLE 3.1 – Opérateurs de logique floue (liste non exhaustive)

T-norme	**T-conorme**	**Négation**	**Nom**
$\min(x, y)$	$\max(x, y)$	$1 - x$	Zadeh
$x \cdot y$	$x + y - xy$	$1 - x$	Probabiliste
$\max(x + y - 1,\, 0)$	$\min(x + y,\, 1)$	$1 - x$	Lukasiewicz
$\max\Big(1 - ((1-x)^p + (1-y)^p)^{1/p},\, 0\Big)$	$\min\Big((x^p + y^p)^{1/p},\, 1\Big)$	$1 - x$	Yager ($p > 0$)
$\begin{cases} x & \text{si } y = 1 \\ y & \text{si } x = 1 \\ 0 & \text{sinon} \end{cases}$	$\begin{cases} x & \text{si } y = 0 \\ y & \text{si } x = 0 \\ 1 & \text{sinon} \end{cases}$	$1 - x$	Drastique

5.5 Lois de De Morgan en logique floue

Les opérateurs flous standards (minimum, maximum, complément classique) vérifient les lois de De Morgan :

$$\overline{A \cup B} = \bar{A} \cap \bar{B}, \qquad \overline{A \cap B} = \bar{A} \cup \bar{B}$$

La Figure 3.13 illustre graphiquement la première loi de De Morgan appliquée aux ensembles flous *Petit* et *Moyen*.

Le tableau 3.1 présente quelques opérateurs classiques utilisés en logique floue. Le choix d'une t-norme, d'une t-conorme et d'un opérateur de négation dépend fortement de l'application considérée et du comportement logique souhaité.

6 Fuzzification, inférence floue et défuzzification

Le fonctionnement d'un système de logique floue repose sur un processus structuré permettant de transformer des données numériques issues du monde réel en une décision exploitable. Ce processus se décompose classiquement en trois étapes fondamentales : la fuzzification, l'inférence floue et la défuzzification.

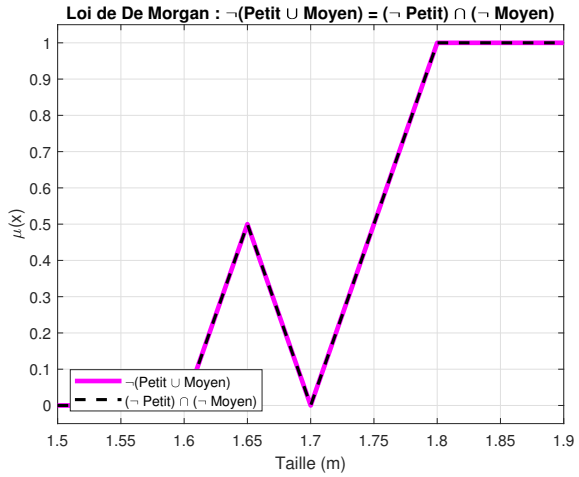


FIGURE 3.13 – Illustration graphique de la loi de De Morgan en logique floue

Contrairement aux systèmes classiques, qui manipulent des valeurs précises et des règles strictes, un système flou travaille avec des informations imprécises exprimées sous forme de concepts linguistiques. Les trois étapes mentionnées ci-dessus assurent la transition progressive entre le monde réel et le raisonnement flou, puis le retour vers une sortie numérique.

La **fuzzification** constitue la première étape du processus. Elle permet de convertir les valeurs d'entrée numériques (par exemple une température, une vitesse ou une taille) en degrés d'appartenance à des ensembles flous prédéfinis. Cette étape établit le lien entre les données mesurées et les concepts linguistiques utilisés dans la base de règles.

L'**inférence floue** correspond au cœur du système. À partir des degrés d'appartenance obtenus lors de la fuzzification et d'

Les opérateurs de logique floue permettent de combiner des informations imprécises de manière cohérente et progressive. Ils constituent un élément central du raisonnement flou et sont indispensables à la construction des règles linguistiques de type *SI–ALORS* et aux mécanismes d'inférence floue.

6.1 Comment fuzzifier ?

La fuzzification est l'étape qui permet de transformer des valeurs numériques issues du monde réel en informations floues exploitables par un système de logique floue. Cette transformation repose sur plusieurs éléments essentiels qui doivent être soigneusement définis.

Définition de l'univers du discours

La première étape de la fuzzification consiste à définir l'univers du discours associé à chaque variable d'entrée et de sortie. L'univers du discours correspond à la plage de variation possible de la grandeur considérée.

Par exemple :

— la température peut varier dans l'intervalle $[0, 50]°\text{C}$,

— la taille d'une personne dans l'intervalle $[1.5, 1.9]$ m,

— la puissance de chauffe dans l'intervalle $[0, 100]\%$.

Cette définition conditionne la forme et l'étendue des ensembles flous utilisés par la suite.

Partition floue de l'univers

Une fois l'univers du discours défini, il est nécessaire de le partitionner en un ensemble de classes floues correspondant à des termes linguistiques. Chaque classe floue représente un concept qualitatif tel que *faible*, *moyen*, *élevé*.

Il est important de noter que la fuzzification concerne :

— les **variables d'entrée** du système flou,

— mais également les **variables de sortie**.

Exemple : Selon les valeurs des entrées, un système flou peut indiquer que la puissance de chauffe en sortie doit être *faible*, *moyenne* ou *forte*.

Cette approche permet de raisonner sur des concepts linguistiques plutôt que sur des valeurs numériques strictes.

Choix des fonctions d'appartenance

À chaque classe floue est associée une fonction d'appartenance définissant le degré selon lequel une valeur donnée appartient à cette classe. Les formes les plus couramment utilisées sont :

— triangulaire,
— trapézoïdale,
— gaussienne.

Le choix de ces fonctions influence directement le comportement du système flou et la qualité de la décision finale.

Une étape délicate et itérative

La fuzzification des variables constitue une phase délicate du processus de logique floue. Elle est souvent réalisée de manière itérative et repose en grande partie sur :

— l'expertise humaine,
— l'expérience du domaine d'application,
— des ajustements successifs basés sur les résultats obtenus.

Une mauvaise définition de l'univers du discours, de la partition floue ou des fonctions d'appartenance peut conduire à des résultats peu satisfaisants, même si le moteur d'inférence est correctement conçu.

6.2 Base de règles floues

La base de règles constitue le cœur d'un système de logique floue. Elle regroupe l'ensemble des connaissances expertes exprimées sous forme de règles linguistiques permettant de relier les variables d'entrée aux variables de sortie.

Contrairement aux systèmes classiques basés sur des équations ou des seuils stricts, la logique floue s'appuie sur des règles qualitatives de type *SI–ALORS*, proches du raisonnement humain.

Structure d'une règle floue

Une règle floue s'exprime généralement sous la forme :

$$\text{SI } x_1 \text{ est } A_1 \text{ ET } x_2 \text{ est } A_2 \ \ldots \ \text{ALORS } y \text{ est } B$$

où :

— $x_1, x_2, \ldots$ sont des variables d'entrée,
— $A_1, A_2, \ldots$ sont des ensembles flous associés aux entrées,
— y est la variable de sortie,
— B est un ensemble flou de sortie.

Chaque règle représente une relation locale entre les entrées et la sortie.

Exemple de base de règles

Considérons un système de régulation de chauffage dont :

— l'entrée est la température,
— la sortie est la puissance de chauffe.

Une base de règles possible peut être :
— **R1 :** SI la température est *basse* ALORS la puissance est *forte*,
— **R2 :** SI la température est *moyenne* ALORS la puissance est *moyenne*,
— **R3 :** SI la température est *élevée* ALORS la puissance est *faible*.
Ces règles traduisent directement le raisonnement intuitif d'un expert humain.

Caractéristiques d'une base de règles

Une base de règles floues présente les caractéristiques suivantes :
— elle peut contenir un nombre limité ou élevé de règles,
— les règles peuvent être partiellement actives simultanément,
— plusieurs règles peuvent contribuer en même temps à la décision finale.
Il n'est pas nécessaire que les règles soient mutuellement exclusives, ce qui permet une transition progressive entre différentes situations.

Rôle dans le processus flou

La base de règles intervient au niveau de l'inférence floue. Après la fuzzification des entrées, chaque règle est évaluée en fonction de son degré d'activation, puis combinée avec les autres règles afin de produire une sortie floue globale.
Ainsi, la base de règles constitue le lien essentiel entre :
— la représentation linguistique des connaissances,
— et le mécanisme mathématique de décision floue.

6.3 Inférence floue

L'inférence floue constitue le mécanisme central d'un système de logique floue. Elle permet de déduire une conclusion floue à partir :
— des valeurs fuzzifiées des entrées,
— de la base de règles floues,
— des opérateurs de logique floue.
Contrairement à l'inférence classique, qui aboutit à des conclusions binaires, l'inférence floue produit des résultats graduels, exprimés en termes de degrés d'appartenance.

Principe général

Le principe de l'inférence floue repose sur l'évaluation des règles de type *SI–ALORS*. Chaque règle est activée avec une intensité qui dépend du degré de satisfaction de sa prémisse.
De manière générale, le processus d'inférence comprend les étapes suivantes :

1. évaluation des prémisses des règles à l'aide des opérateurs flous (ET, OU),
2. calcul du degré d'activation de chaque règle,
3. production d'une conclusion floue partielle pour chaque règle,
4. agrégation des conclusions issues de l'ensemble des règles actives.

Évaluation des règles

Soit une règle floue de la forme :

$$\text{SI } x \text{ est } A \text{ ET } y \text{ est } B \text{ ALORS } z \text{ est } C$$

Le degré d'activation de la règle est obtenu en combinant les degrés d'appartenance des entrées à l'aide d'une t-norme, généralement l'opérateur minimum :

$$\alpha = \min\left(\mu_A(x), \mu_B(y)\right)$$

Ce degré α représente l'intensité avec laquelle la règle contribue à la décision finale.

Génération des conclusions floues

Une fois le degré d'activation calculé, la conclusion de la règle est obtenue en modifiant l'ensemble flou de sortie associé à la règle. Dans le cas de l'inférence de type Mamdani, l'ensemble de sortie est généralement tronqué au niveau α.

Chaque règle produit ainsi une conclusion floue partielle.

Agrégation des règles

Les conclusions floues issues de toutes les règles actives sont ensuite combinées afin d'obtenir un unique ensemble flou de sortie. Cette agrégation est généralement réalisée à l'aide d'une t-conorme, le plus souvent l'opérateur maximum.

Le résultat de l'inférence floue est donc un ensemble flou global représentant la décision du système.

Rôle de l'inférence floue

L'inférence floue permet :
— de prendre en compte simultanément plusieurs règles,
— de gérer des situations intermédiaires,
— de produire des décisions progressives et robustes.

Elle constitue l'étape clé reliant la connaissance linguistique exprimée par les règles à la sortie floue du système, qui sera ensuite transformée en valeur numérique par la défuzzification.

6.4 Défuzzification : principes et méthodes

La défuzzification constitue la dernière étape du processus de raisonnement flou. Son objectif est de transformer l'ensemble flou de sortie, obtenu après l'inférence floue, en une valeur numérique précise exploitable par le système (commande, décision, diagnostic, etc.).

Contrairement aux étapes précédentes, qui manipulent des concepts linguistiques et des degrés d'appartenance, la défuzzification assure le retour vers le monde réel en produisant une sortie nette.

Principe général de la défuzzification

À l'issue de l'inférence floue, le système fournit un ensemble flou global représentant la sortie. La défuzzification consiste à associer à cet ensemble flou une valeur unique y^*, appelée *valeur défuzzifiée*.

Le choix de la méthode de défuzzification influence directement :

— la précision de la sortie,

— la réactivité du système,

— le comportement global du système flou.

Il n'existe pas de méthode universelle ; le choix dépend du type d'application considérée.

Méthode du centre de gravité (ou centre de masse)

La méthode du centre de gravité est la plus utilisée en pratique, notamment dans les systèmes de type Mamdani. Elle consiste à calculer la position du centre de masse de l'ensemble flou de sortie.

Pour un univers du discours U, la valeur défuzzifiée est donnée par :

$$y^* = \frac{\int_U y\,\mu(y)\,dy}{\int_U \mu(y)\,dy}$$

Cette méthode présente les avantages suivants :

— elle tient compte de l'ensemble de la distribution floue,

— elle fournit des résultats lisses et stables,

— elle est bien adaptée aux systèmes de commande.

Méthode du maximum

Les méthodes basées sur le maximum consistent à choisir une valeur correspondant au maximum de la fonction d'appartenance de sortie.

On distingue principalement :

— le **premier maximum** : plus petite valeur atteignant le maximum,

— le **dernier maximum** : plus grande valeur atteignant le maximum,

— la **moyenne des maxima**.

Ces méthodes sont simples et rapides, mais elles ne prennent pas en compte l'ensemble de la forme de l'ensemble flou.

Autres méthodes de défuzzification

D'autres méthodes de défuzzification peuvent également être utilisées, notamment :

— la méthode de la moyenne pondérée,

— la méthode du centre des sommes,

— les méthodes spécifiques aux systèmes de type Sugeno.

Ces méthodes sont souvent choisies pour des raisons de simplicité de calcul ou de contraintes temps réel.

Choix de la méthode

Le choix de la méthode de défuzzification dépend de plusieurs facteurs :
— la nature de l'application (commande, décision, diagnostic),
— la complexité du système,
— les contraintes de calcul et de temps réel,
— la précision attendue sur la sortie.

Dans la plupart des applications industrielles, la méthode du centre de gravité est privilégiée en raison de sa robustesse et de sa cohérence avec le raisonnement flou.

7 Exemple : commande de l'installation de chauffage d'un immeuble

7.1 Description du problème

On souhaite commander l'installation de chauffage d'un immeuble à l'aide d'un contrôleur flou. On dispose de deux sondes de température : l'une à l'extérieur de l'immeuble (grandeur externe) et l'autre à l'intérieur (grandeur interne). Sur la base de ces deux mesures et en faisant appel aux règles d'inférence, le contrôleur flou doit régler la puissance de l'installation de chauffage.

7.2 Fuzzification de la température externe

On choisit deux intervalles flous et des fonctions d'appartenance de type trapézoïdales en définissant :
— **froid** : température inférieure à 5°C,
— **chaud** : température supérieure à 20°C.

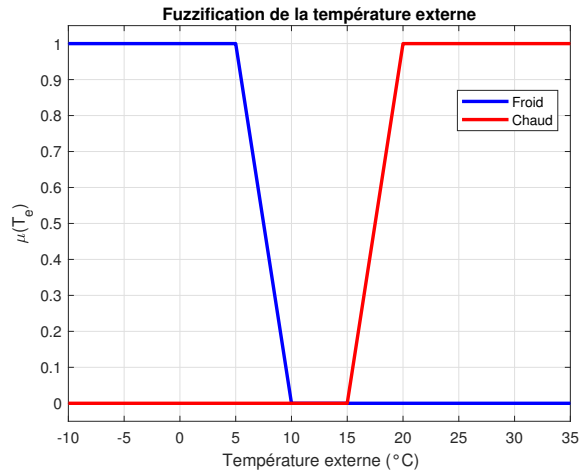


FIGURE 3.14 – Fuzzification de la température externe : ensembles flous *Froid* et *Chaud*.

7.3 Fuzzification de la température interne

On choisit trois intervalles flous et des fonctions d'appartenance trapézoïdales en définissant :

— **froid** : température inférieure à 15°C,
— **bon** : température comprise entre 19°C et 21°C,
— **chaud** : température supérieure à 25°C.

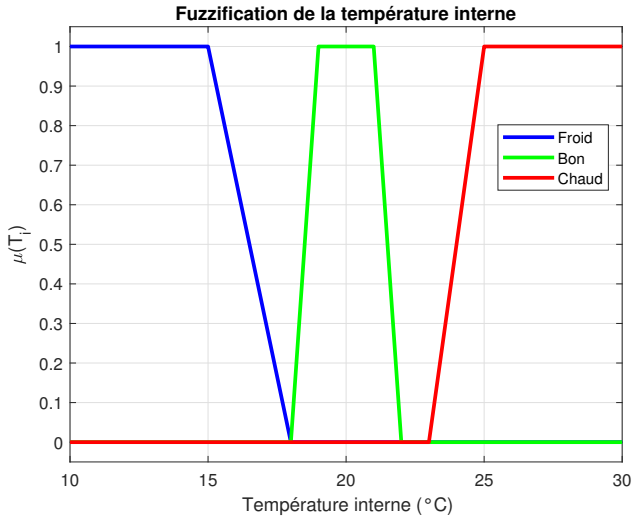


FIGURE 3.15 – Fuzzification de la température interne : ensembles flous *Froid*, *Bon* et *Chaud*.

7.4 Fuzzification de la puissance de chauffage

On choisit quatre intervalles flous pour définir la puissance de l'installation avec des fonctions d'appartenance en forme de raies (singletons). On définit :

Puissance	**Valeur (%)**
Nulle	0
Faible	33
Moyenne	67
Maximale	100

7.5 Choix des opérateurs et de la défuzzification

Les opérateurs de logique floue sont définis comme suit :
— l'opérateur **ET** est réalisé par le calcul du minimum,
— l'opérateur **OU** est réalisé par le calcul du maximum.
La défuzzification se fait par le calcul du **centre de gravité**.

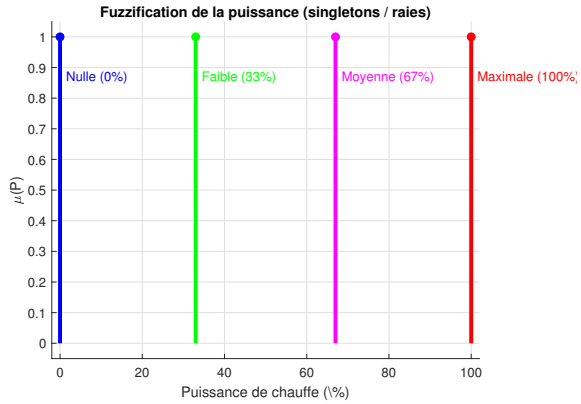


FIGURE 3.16 – Fuzzification de la puissance de chauffe (singletons / raies).

7.6 Règles d'inférence

L'expérience acquise sur l'installation de chauffage a permis de définir une base de règles floues reliant les deux entrées (température extérieure et température intérieure) à la sortie (puissance de chauffage). Les règles sont exprimées sous forme linguistique *SI–ALORS*, conformément au raisonnement d'un expert.

1. **R1 :** Si la température extérieure est *froide* et la température intérieure est *froide*, alors mettre la puissance au *maximum*.
2. **R2 :** Si la température extérieure est *froide* et la température intérieure est *bonne*, alors mettre une puissance *moyenne*.
3. **R3 :** Si la température extérieure est *froide* et la température intérieure est *chaude*, alors mettre une puissance *faible*.
4. **R4 :** Si la température extérieure est *chaude* et la température intérieure est *froide*, alors mettre une puissance *moyenne*.
5. **R5 :** Si la température extérieure est *chaude* et la température intérieure est *bonne*, alors mettre une puissance *faible*.
6. **R6 :** Si la température extérieure est *chaude* et la température intérieure est *chaude*, alors mettre une puissance *nulle*.

Ces six règles couvrent l'ensemble des combinaisons possibles entre les classes floues des deux entrées et traduisent une stratégie intuitive : plus l'environnement est froid et moins l'intérieur est confortable, plus la puissance demandée est élevée.

Chapitre 4

Réseaux de neurones artificiels

1 Introduction

Le *Machine Learning* (ou apprentissage automatique) implique des mécanismes adaptatifs qui permettent aux ordinateurs d'apprendre par l'expérience, d'apprendre par l'exemple et d'apprendre par analogie. Les capacités d'apprentissage peuvent améliorer les performances d'un système intelligent au fil du temps. Parmi les approches les plus populaires de l'apprentissage automatique figurent les **réseaux de neurones artificiels**.

L'idée des réseaux de neurones artificiels s'inspire du fonctionnement du cerveau humain. Dès les années 1940, Warren McCulloch et Walter Pitts ont proposé le premier modèle de neurone formel, décrivant le neurone comme une unité logique simple capable de prendre des décisions élémentaires. Quelques années plus tard, en 1958, Frank Rosenblatt introduit le *perceptron*, considéré comme le premier modèle de réseau de neurones capable d'apprentissage supervisé. Ce modèle marqua le début d'une période d'enthousiasme autour de l'intelligence artificielle.

Cependant, les limites du perceptron, mises en évidence dans les années 1960 par Marvin Minsky et Seymour Papert, ont entraîné un ralentissement de la recherche dans ce domaine pendant plusieurs décennies. Ce n'est qu'à partir des années 1980, grâce à la redécouverte de l'algorithme de rétropropagation du gradient et à l'augmentation de la puissance de calcul, que les réseaux de neurones ont connu un renouveau. Dans les années 2000 et 2010, l'émergence du *deep learning* — ou apprentissage profond — a permis d'entraîner des réseaux de très grande taille, révolutionnant ainsi de nombreux domaines tels que la reconnaissance d'images, la traduction automatique et la synthèse vocale.

Aujourd'hui, les réseaux de neurones constituent la base de nombreux systèmes d'intelligence artificielle modernes et continuent d'évoluer avec des architectures toujours plus complexes et performantes.

2 Neurone biologique

Le neurone est l'unité fondamentale du système nerveux. Il est responsable de la réception, du traitement et de la transmission de l'information à travers des signaux électriques et chimiques. Un neurone est composé de plusieurs parties principales :

— **Le corps cellulaire (ou soma)** : il contient le noyau de la cellule et la majorité des organites nécessaires au métabolisme. Le soma assure les fonctions vitales du neurone et intègre les signaux reçus des autres cellules.
— **Les dendrites** : ce sont de fines prolongations ramifiées qui émergent du soma. Elles reçoivent les signaux provenant d'autres neurones et les transmettent vers le corps cellulaire. Les dendrites jouent donc un rôle essentiel dans la collecte et la sommation des informations.
— **L'axone** : il s'agit d'une fibre unique et allongée qui conduit l'influx nerveux du soma vers d'autres neurones, muscles ou glandes. L'extrémité de l'axone se divise en plusieurs terminaisons synaptiques qui assurent la communication avec d'autres cellules par l'intermédiaire des synapses.

L'activité neuronale repose sur des signaux électrochimiques : lorsqu'un neurone reçoit un ensemble de signaux excitateurs suffisants, il génère un *potentiel d'action*, c'est-à-dire une impulsion électrique qui se propage le long de l'axone jusqu'aux terminaisons synaptiques. Ces signaux sont ensuite transmis à d'autres neurones par la libération de neurotransmetteurs à travers la fente synaptique. Ainsi, les réseaux de neurones biologiques forment des circuits capables d'apprentissage, de mémorisation et de prise de décision — propriétés qui ont inspiré la conception des réseaux de neurones artificiels.

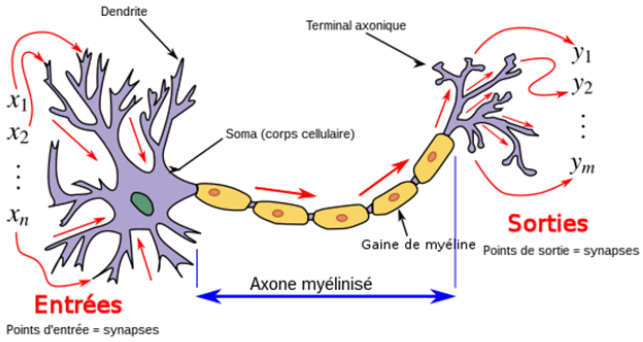


FIGURE 4.1 – Structure générale d'un neurone biologique : soma, dendrites et axone.

3 Neurone artificiel

Perceptron élémentaire

Les réseaux de neurones artificiels s'inspirent du fonctionnement des neurones biologiques, mais en adoptant une modélisation mathématique simplifiée. Un neurone artificiel, encore appelé *perceptron élémentaire*, reçoit plusieurs signaux d'entrée, applique à chacun un poids représentant son importance, puis calcule une combinaison pondérée de ces entrées. Cette somme est ensuite transformée par une fonction dite *d'activation*, afin de produire une sortie.

Mathématiquement, si l'on note :
— $x_1, x_2, \ldots, x_n$ les signaux d'entrée ;
— $w_1, w_2, \ldots, w_n$ les poids associés à chaque entrée ;

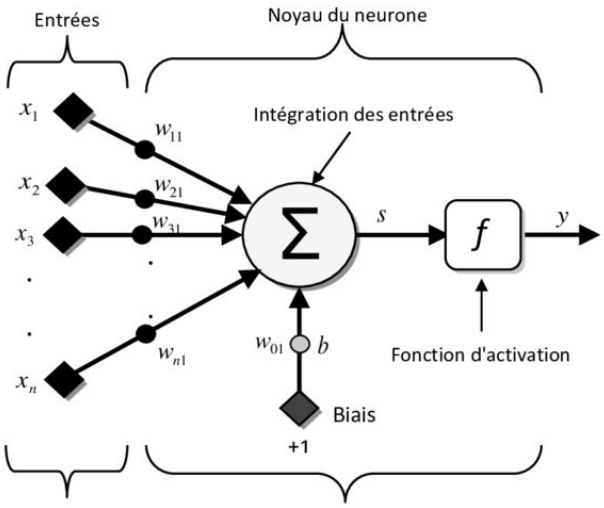


FIGURE 4.2 – Schéma d'un neurone artificiel : combinaison pondérée des entrées suivie d'une fonction d'activation.

— b le biais du neurone ;
— et f la fonction d'activation,
alors la sortie du neurone est donnée par :

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{4.1}$$

La fonction d'activation f introduit une non-linéarité dans le modèle, permettant au réseau de représenter des relations complexes entre les variables.

Analogie entre les réseaux neuronaux biologiques et artificiels

Il existe de fortes similarités conceptuelles entre le fonctionnement du cerveau humain et celui des réseaux neuronaux artificiels. Les premiers inspirent la structure et les principes d'apprentissage des seconds, même si la complexité biologique reste bien supérieure à la modélisation informatique. Le tableau ci-dessous résume les principales correspondances entre les deux types de réseaux.

Réseau de neurones biologique	***Réseau de neurones artificiel***
Soma (corps cellulaire)	Neurone
Dendrite	Entrée
Axone	Sortie
Synapse	Poids

TABLE 4.1 – Analogie entre un réseau de neurones biologique et un réseau de neurones artificiel.

Pour illustrer cette analogie, la figure suivante présente un schéma comparatif mettant en parallèle les structures et les flux d'information dans les deux types de réseaux.

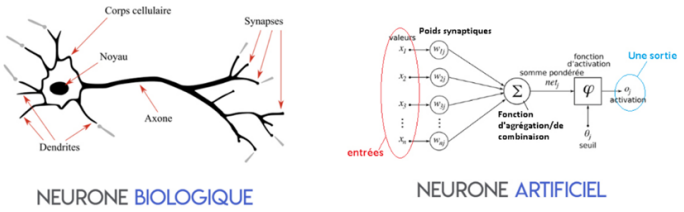


FIGURE 4.3 – Analogie entre un réseau de neurones biologique (gauche) et un réseau de neurones artificiel (droite).

4 Les fonctions d'activation d'un neurone

Les fonctions d'activation jouent un rôle fondamental dans les réseaux de neurones artificiels. Elles permettent d'introduire une **non-linéarité** dans le modèle, ce qui donne au réseau la capacité d'apprendre et de modéliser des relations complexes entre les variables d'entrée et de sortie. Sans cette non-linéarité, un réseau de neurones ne serait qu'une combinaison linéaire de ses entrées, et ne pourrait donc pas résoudre de problèmes non linéaires.

Parmi les fonctions d'activation les plus utilisées, on distingue :

- **La fonction seuil (Step function)** : elle renvoie 1 si l'entrée est positive, et 0 sinon.
- **La fonction signe (Sign function)** : elle donne +1 ou -1 selon le signe de l'entrée.
- **La fonction sigmoïde (Sigmoid function)** : continue et dérivable, elle « lisse » la transition entre les valeurs et est définie par

$$y = \frac{1}{1 + e^{-x}}.$$

- **La fonction linéaire (Linear function)** : utilisée dans certaines couches de sortie pour les tâches de régression, elle garde la proportionnalité entre entrée et sortie.

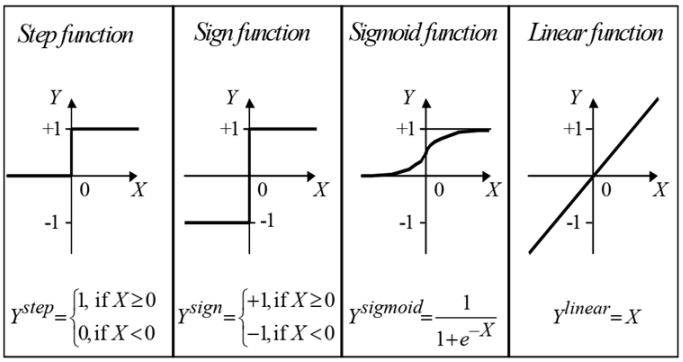


FIGURE 4.4 – Exemples de fonctions d'activation d'un neurone : seuil, signe, sigmoïde et linéaire.

Les fonctions d'activation modernes incluent également la **ReLU** (*Rectified Linear Unit*), définie par $f(x) = \max(0, x)$, et ses variantes comme *Leaky ReLU* ou *ELU*, qui permettent un apprentissage plus stable dans les réseaux profonds.

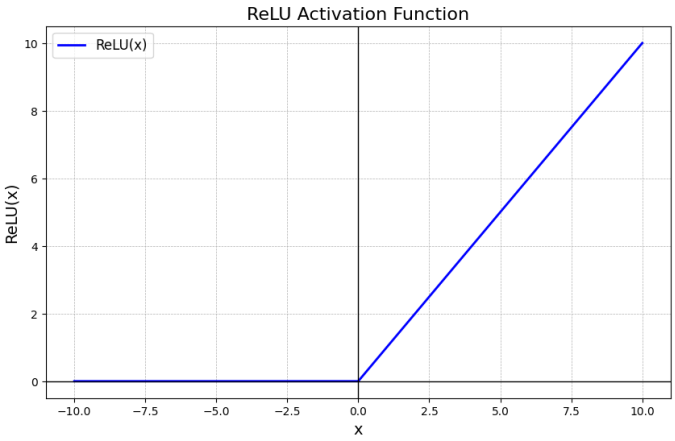


FIGURE 4.5 – fonctions d'activation ReLU (Rectified Linear Unit)

5 Algorithme d'apprentissage du perceptron

L'algorithme du **perceptron** est l'une des premières méthodes d'apprentissage supervisé pour les réseaux de neurones. Il vise à ajuster automatiquement les poids des connexions afin que le neurone produise la sortie désirée pour chaque exemple d'apprentissage.

L'algorithme se déroule en plusieurs étapes successives :

Étape 1 : Initialisation

Définir les poids initiaux $w_1, w_2, \ldots, w_n$ et le seuil θ à des valeurs aléatoires dans une plage donnée, par exemple $[-0.5, 0.5]$. Ces valeurs initiales constituent le point de départ de l'apprentissage.

$$w_i(0) \in [-0.5, 0.5], \quad \theta(0) \in [-0.5, 0.5]$$

Étape 2 : Activation

Activer le perceptron en appliquant les entrées $x_1(p), x_2(p), \ldots, x_n(p)$ et la sortie désirée $Y_d(p)$ pour un exemple d'apprentissage p. Calculer ensuite la sortie réelle du perceptron à l'itération p :

$$Y(p) = \text{step}\left(\sum_{i=1}^{n} w_i(p)\, x_i(p) - \theta(p)\right)$$

où n est le nombre d'entrées du perceptron et step$(\cdot)$ est la fonction seuil définie par :

$$\text{step}(x) = \begin{cases} 1, & \text{si } x \geq 0 \\ 0, & \text{si } x < 0 \end{cases}$$

Étape 3 : Mise à jour des poids

Mettre à jour les poids du perceptron selon la règle d'apprentissage dite **règle du delta** :

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

où la correction de poids $\Delta w_i(p)$ est donnée par :

$$\Delta w_i(p) = \eta \left[Y_d(p) - Y(p) \right] x_i(p)$$

avec η le **taux d'apprentissage** $(0 < \eta \leq 1)$, qui contrôle la vitesse d'ajustement des poids.

De la même manière, le seuil est mis à jour selon :

$$\theta(p+1) = \theta(p) - \eta \left[Y_d(p) - Y(p) \right]$$

Étape 4 : Itération

Incrémenter l'indice d'itération p et retourner à l'étape 2. Répéter le processus pour tous les exemples d'apprentissage jusqu'à la convergence, c'est-à-dire jusqu'à ce que les sorties calculées soient égales aux sorties désirées ou qu'un nombre maximal d'itérations soit atteint.

Algorithm 1 Apprentissage d'un neurone artificiel

Require: Vecteur d'entrée $E = (e_1, e_2, \ldots, e_n)$, sortie désirée S_{attendue}, taux d'apprentissage η, seuil d'erreur $\text{Erreur}_{\min}$

Ensure: Vecteur des poids appris $U = (u_1, u_2, \ldots, u_n)$

1: Initialiser les poids $u_i \leftarrow \text{random}(0, 1)$

2: **repeat**

3: Calculer la somme pondérée :

$$z = \sum_{i=1}^{n} e_i \times u_i$$

4: Calculer la sortie du neurone :

$$S_{\text{prediction}} = h(z)$$

5: Calculer l'erreur :

$$\text{Erreur} = S_{\text{attendue}} - S_{\text{prediction}}$$

6: **for** chaque poids u_i **do**

7: Calculer la correction :

$$\Delta u_i = \eta \times \text{Erreur} \times e_i$$

8: Mettre à jour le poids :

$$u_i \leftarrow u_i + \Delta u_i$$

9: **end for**

10: **until** $|\text{Erreur}| < \text{Erreur}_{\min}$

11: **Retourner** le vecteur des poids U

6 Exemple : Apprentissage du perceptron pour la fonction logique ET

Pour illustrer concrètement le processus d'apprentissage du perceptron, considérons le cas de la fonction logique **ET** (AND). Le perceptron prend deux entrées x_1 et x_2 et doit produire une sortie Y_d selon la table de vérité suivante :

x_1	x_2	**Sortie désirée** Y_d
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 4.2 – Table de vérité de la fonction logique ET.

Paramètres initiaux

On initialise les paramètres du perceptron de la manière suivante :

$$w_1 = 0.3, \quad w_2 = -0.1, \quad \theta = 0.2, \quad \eta = 0.1$$

où η est le taux d'apprentissage.

La fonction d'activation utilisée est la fonction seuil :

$$Y = \begin{cases} 1, & \text{si } S = w_1 x_1 + w_2 x_2 - \theta \geq 0 \\ 0, & \text{sinon.} \end{cases}$$

Époque 1 : Calcul et mise à jour des poids

Nous faisons passer successivement les quatre combinaisons (x_1, x_2) de la table de vérité :

Exemple 1 : $(x_1, x_2) = (0, 0)$

$$S = 0.3(0) + (-0.1)(0) - 0.2 = -0.2$$

$$Y = 0 \quad \text{et} \quad Y_d = 0 \Rightarrow \text{pas de mise à jour.}$$

Exemple 2 : $(x_1, x_2) = (0, 1)$

$$S = 0.3(0) + (-0.1)(1) - 0.2 = -0.3$$

$$Y = 0 \quad \text{et} \quad Y_d = 0 \Rightarrow \text{pas de mise à jour.}$$

Exemple 3 : $(x_1, x_2) = (1, 0)$

$$S = 0.3(1) + (-0.1)(0) - 0.2 = 0.1$$

$$Y = 1, \quad Y_d = 0 \Rightarrow \text{erreur : } e = Y_d - Y = -1$$

Mise à jour des poids et du seuil :

$$\Delta w_1 = \eta e x_1 = 0.1(-1)(1) = -0.1$$

$$\Delta w_2 = \eta e x_2 = 0.1(-1)(0) = 0$$

$$\Delta \theta = -\eta e = -0.1(-1) = +0.1$$

$$\Rightarrow w_1 = 0.3 - 0.1 = 0.2, \quad w_2 = -0.1, \quad \theta = 0.2 + 0.1 = 0.3$$

Exemple 4 : $(x_1, x_2) = (1, 1)$

$$S = 0.2(1) + (-0.1)(1) - 0.3 = -0.2$$

$$Y = 0, \quad Y_d = 1 \Rightarrow \text{erreur : } e = Y_d - Y = 1$$

Mise à jour :

$$\Delta w_1 = 0.1(1)(1) = +0.1, \quad \Delta w_2 = 0.1(1)(1) = +0.1, \quad \Delta \theta = -0.1(1) = -0.1$$

$$\Rightarrow w_1 = 0.2 + 0.1 = 0.3, \quad w_2 = -0.1 + 0.1 = 0.0, \quad \theta = 0.3 - 0.1 = 0.2$$

Résultat après une époque

Après avoir traité les quatre exemples, les nouveaux paramètres du perceptron sont :

$$w_1 = 0.3, \quad w_2 = 0.0, \quad \theta = 0.2$$

On constate que les poids ont évolué pour mieux approcher la frontière de décision de la fonction logique **ET**. En répétant plusieurs époques, le perceptron convergera vers une solution stable satisfaisant $Y = Y_d$ pour tous les exemples.

Résumé de l'apprentissage du perceptron (fonction logique ET)

Le tableau suivant résume les calculs effectués lors des cinq premières époques d'apprentissage du perceptron, pour la fonction logique **ET**, avec les paramètres :

$$\theta = 0.2, \quad \alpha = 0.1$$

Après la cinquième époque, les poids se stabilisent autour de :

$$w_1 = 0.1, \quad w_2 = 0.1, \quad \theta = 0.2$$

Le perceptron a donc appris la fonction logique **ET**, produisant $Y = 1$ uniquement lorsque $x_1 = 1$ et $x_2 = 1$.

7 Les limites du perceptron

Le perceptron constitue la première brique historique des réseaux de neurones artificiels. Malgré sa simplicité et son efficacité pour résoudre certains problèmes linéairement séparables (comme les fonctions logiques **ET** et **OU**), il présente plusieurs limitations fondamentales.

1. Limitation de la séparabilité linéaire

Le perceptron ne peut apprendre que des fonctions dont les classes sont séparables par une **frontière linéaire**. Autrement dit, il ne peut résoudre que les problèmes pour lesquels il existe une droite (en 2D) ou un hyperplan (en dimension supérieure) qui sépare parfaitement les exemples des deux classes.

Les opérations logiques **ET** et **OU** satisfont cette condition, car leurs points peuvent être séparés dans le plan (x_1, x_2) par une droite. En revanche, l'opération **XOR** (ou exclusif) n'est pas linéairement séparable, car aucun plan ou droite ne peut distinguer correctement les deux classes.

2. Absence de généralisation non linéaire

Le perceptron simple ne possède qu'une seule couche de calcul et une fonction d'activation linéaire ou seuil. Il ne peut donc pas modéliser des relations complexes ou non linéaires entre les entrées et les sorties. Pour surmonter cette limite, on

Époque	**Entrées**		**Sortie dés.**	**Poids ini.**		**Sortie ré.**	**Err.**	**Poids finaux**	
	x_1	x_2	Y_d	w_1	w_2	Y	$e = Y_d - Y$	w_1	w_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

TABLE 4.3 – Résumé du processus d'apprentissage du perceptron pour la fonction logique ET.

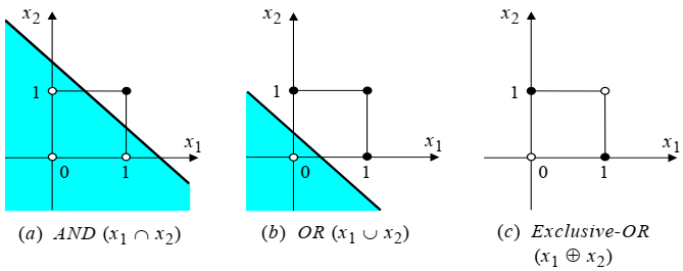


FIGURE 4.6 – Graphes bidimensionnels des opérations logiques de base. Seules les fonctions ET et OU sont linéairement séparables, contrairement à XOR.

introduit des réseaux contenant plusieurs couches de neurones (**perceptrons multicouches**), capables d'approximer des fonctions arbitrairement complexes grâce à des combinaisons non linéaires.

3. Sensibilité au choix des paramètres

La convergence du perceptron dépend fortement :
— du choix du taux d'apprentissage η ;
— de l'initialisation des poids ;
— et de la nature du jeu de données (bruit, redondance, déséquilibre des classes).

Dans certains cas, le perceptron peut osciller sans jamais converger, ou encore converger vers une solution sous-optimale.

Le perceptron simple marque une étape importante dans l'histoire de l'apprentissage automatique, mais ses limites justifient l'apparition de modèles plus puissants, notamment les **réseaux de neurones multicouches (MLP)** et l'algorithme de **rétropropagation du gradient**, qui permettent de traiter des problèmes non linéaires et d'apprendre des représentations hiérarchiques.

8 Réseau de neurones MLP (Perceptron Multicouche)

La structure d'un réseau **MLP** (Multi Layer Perceptron) ou **perceptron multicouche** est un réseau de neurones artificiels qui se compose d'une *couche d'entrée*, d'une *couche de sortie* et d'une ou plusieurs *couches intermédiaires* appelées *couches cachées*.

La figure 4.7 illustre un exemple de réseau de neurones contenant n entrées dans la couche d'entrée, m neurones dans la couche cachée et r neurones dans la couche de sortie. Le nombre de couches cachées et le nombre de neurones dans chaque couche dépendent de la complexité du problème traité par le réseau.

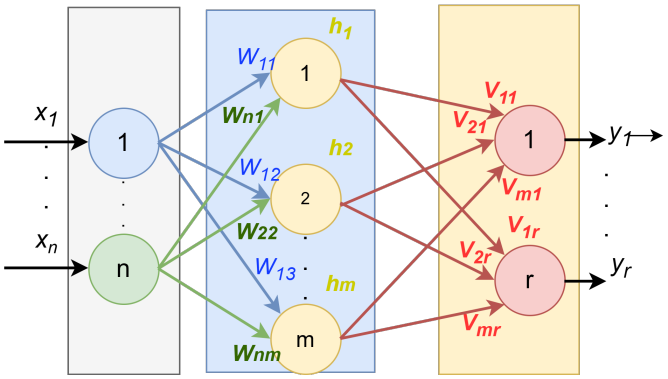


FIGURE 4.7 – Architecture d'un réseau de neurones multicouche (MLP)

Le vecteur de sortie de la couche cachée peut être décrit comme suit :

$$h_j = f\left(\sum_{i=0}^{n} W_{j,i}\, x_i\right)$$

Le vecteur de sortie du réseau de neurones est alors donné par :

$$y_k = g\left(\sum_{j=0}^{m} V_{k,j}\, h_j\right) = g\left(\sum_{j=0}^{m} V_{k,j}\, f\left(\sum_{i=0}^{n} W_{j,i}\, x_i\right)\right)$$

où :
— f : fonction d'activation de la couche cachée ;
— g : fonction d'activation de la couche de sortie.

L'apprentissage d'un réseau de neurones signifie qu'il change son comportement de façon à lui permettre de se rapprocher d'un but défini. Le principe utilisé par la **rétro-propagation** du gradient est la minimisation d'une fonction dépendante de l'erreur. La méthode de la rétro-propagation permet de faire propager l'erreur à travers les couches. La fonction à minimiser est la somme des erreurs carrées entre la sortie désirée et la sortie du réseau de neurones. Son expression s'écrit comme suit :

$$E(k) = \frac{1}{2}\sum_{k=1}^{\infty}(e_k(k))^2 = \frac{1}{2}\sum_{k=1}^{\infty}(y_{d_k}(k) - y_{nn_k}(k))^2$$

avec y_{d_k} et y_{nn_k} représentant respectivement la sortie désirée et la sortie du réseau de neurones.

L'algorithme du gradient s'écrit :

$$X(k+1) = X(k) - \eta\frac{\partial E(k)}{\partial X(k)}$$

où η est le *taux d'apprentissage.*

Mise à jour des poids entre la couche cachée et la couche de sortie :

$$V_{k,j}(k+1) = V_{k,j}(k) - \eta\frac{\partial E(k)}{\partial V_{k,j}(k)} = V_{k,j}(k) + \eta e_k\frac{\partial y_{nn_k}(k)}{\partial V_{k,j}(k)}$$

La dérivée $\frac{\partial y_{nn_k}(k)}{\partial V_{k,j}(k)}$ dépend de la fonction d'activation utilisée pour la couche de sortie.

Le vecteur de sortie du réseau de neurones :

$$y_{nn_k} = g\left(\sum_{j=0}^{m} V_{k,j}h_j\right)$$

On considère les fonctions d'activation suivantes :
- **g linéaire :**

$$y_{nn_k} = \sum_{j=0}^{m} V_{k,j}h_j \qquad \Rightarrow \qquad \frac{\partial y_{nn_k}(k)}{\partial V_{k,j}(k)} = h_j$$

- **g sigmoïde :**

$$y_{nn_k} = \frac{1}{1+e^{-\sum_{j=0}^{m} V_{k,j}h_j}} \qquad \Rightarrow \qquad \frac{\partial y_{nn_k}(k)}{\partial V_{k,j}(k)} = (1-y_{nn_k})\,y_{nn_k}\,h_j$$

- **g tangente hyperbolique :**

$$y_{nn_k} = \tanh\left(\sum_{j=0}^{m} V_{k,j}h_j\right) \qquad \Rightarrow \qquad \frac{\partial y_{nn_k}(k)}{\partial V_{k,j}(k)} = (1-y_{nn_k}^2)\,h_j$$

Finalement :

$$V_{k,j}(k+1) = V_{k,j}(k) + \eta\,\delta_k\,h_j$$

avec :

$$\delta_k = \begin{cases} e_k & \text{si } g \text{ est linéaire} \\ e_k(1-y_{nn_k})y_{nn_k} & \text{si } g \text{ est sigmoïde} \\ e_k(1-y_{nn_k}^2) & \text{si } g \text{ est tangente hyperbolique} \end{cases}$$

Mise à jour des poids entre la couche d'entrée et la couche cachée :

$$W_{j,i}(k+1) = W_{j,i}(k) - \eta\frac{\partial E(k)}{\partial W_{j,i}(k)} = W_{j,i}(k) + \eta e_k\frac{\partial y_{nn_k}(k)}{\partial h_j}\frac{\partial h_j}{\partial W_{j,i}(k)}$$

La dérivée $\frac{\partial y_{nn_k}(k)}{\partial h_j}$ dépend de la fonction d'activation de la couche de sortie :

- **g linéaire :**

$$\frac{\partial y_{nn_k}(k)}{\partial h_j} = V_{k,j}$$

- **g sigmoïde :**

$$\frac{\partial y_{nn_k}(k)}{\partial h_j} = (1-y_{nn_k})\,y_{nn_k}\,V_{k,j}$$

- **g tangente hyperbolique :**

$$\frac{\partial y_{nn_k}(k)}{\partial h_j} = (1-y_{nn_k}^2)\,V_{k,j}$$

Sortie de la couche cachée :

$$h_j = f\left(\sum_{i=0}^{n} W_{j,i}x_i\right)$$

- **f linéaire :**

$$h_j = \sum_{i=0}^{n} W_{j,i}x_i \qquad \Rightarrow \qquad \frac{\partial h_j}{\partial W_{j,i}(k)} = x_i$$

- **f sigmoïde :**

$$h_j = \frac{1}{1+e^{-\sum_{i=0}^{n} W_{j,i}x_i}} \qquad \Rightarrow \qquad \frac{\partial h_j}{\partial W_{j,i}(k)} = (1-h_j)h_j x_i$$

- **f tangente hyperbolique :**

$$h_j = \tanh\left(\sum_{i=0}^{n} W_{j,i}x_i\right) \qquad \Rightarrow \qquad \frac{\partial h_j}{\partial W_{j,i}(k)} = (1-h_j^2)x_i$$

Finalement :

$$W_{j,i}(k+1) = W_{j,i}(k) + \eta\,\delta_j\,x_i$$

avec :

$$\delta_j = \begin{cases} \sum_{k=1}^{r}\delta_k V_{k,j} & \text{si } f \text{ est linéaire} \\ (1-h_j)h_j\sum_{k=1}^{r}\delta_k V_{k,j} & \text{si } f \text{ est sigmoïde} \\ (1-h_j^2)\sum_{k=1}^{r}\delta_k V_{k,j} & \text{si } f \text{ est tangente hyperbolique} \end{cases}$$

Convergence et critères d'arrêt de l'apprentissage

L'apprentissage d'un réseau de neurones est un processus itératif qui consiste à ajuster les poids synaptiques afin de minimiser la fonction d'erreur $E(k)$. Cependant, il est essentiel de définir des critères d'arrêt pour éviter un apprentissage trop long ou un sur-apprentissage (overfitting).

Phase de test et de validation du réseau MLP

Après la phase d'apprentissage, le réseau de neurones doit être évalué afin de mesurer sa capacité à généraliser les connaissances acquises. Cette évaluation se fait à l'aide des ensembles de *validation* et de *test*.

1. Objectif de la validation La validation intervient pendant l'apprentissage, et a pour but de vérifier la performance du modèle sur des données qui ne participent pas directement à l'ajustement des poids. Le réseau est entraîné sur un ensemble d'apprentissage, puis évalué périodiquement sur l'ensemble de validation.

Le suivi simultané de l'erreur d'apprentissage $E_{\text{train}}(k)$ et de l'erreur de validation $E_{\text{val}}(k)$ permet de détecter le phénomène de sur-apprentissage (*overfitting*). Lorsque $E_{\text{train}}(k)$ continue de diminuer alors que $E_{\text{val}}(k)$ commence à augmenter, l'apprentissage est interrompu.

$$\text{Si } E_{\text{val}}(k) \uparrow \quad \text{et} \quad E_{\text{train}}(k) \downarrow \quad \Rightarrow \quad \text{Arrêt de l'apprentissage.}$$

2. Phase de test Une fois le réseau entraîné et validé, on procède à la phase de *test*. Cette phase consiste à évaluer les performances du réseau sur un ensemble de données totalement inédites (jamais vues lors de l'apprentissage ou la validation).

Les principales mesures utilisées pour évaluer la qualité de la prédiction sont :

— **Erreur quadratique moyenne (MSE)** :

$$\text{MSE} = \frac{1}{N}\sum_{k=1}^{N}(y_{d_k} - y_{nn_k})^2$$

— **Erreur absolue moyenne (MAE)** :

$$\text{MAE} = \frac{1}{N}\sum_{k=1}^{N}|y_{d_k} - y_{nn_k}|$$

— **Coefficient de corrélation (R)** :

$$R = \frac{\sum_{k=1}^{N}(y_{d_k} - \bar{y}_d)(y_{nn_k} - \bar{y}_{nn})}{\sqrt{\sum_{k=1}^{N}(y_{d_k} - \bar{y}_d)^2 \sum_{k=1}^{N}(y_{nn_k} - \bar{y}_{nn})^2}}$$

Ces indicateurs permettent de quantifier la précision du réseau et d'évaluer sa capacité de généralisation.

3. Interprétation des résultats

- Si les erreurs d'apprentissage, de validation et de test sont proches et faibles, le réseau est bien entraîné et généralise correctement.
- Si l'erreur d'apprentissage est faible mais que l'erreur de test est élevée, le réseau est **sur-entraîné**.
- Si l'erreur d'apprentissage et celle de test sont toutes deux élevées, le réseau est **sous-entraîné** (il n'a pas appris correctement les relations entre les données).

4. Optimisation du modèle Afin d'obtenir un modèle performant, il est souvent nécessaire d'ajuster certains hyperparamètres :

- le nombre de couches cachées et de neurones ;
- le taux d'apprentissage η ;
- le choix des fonctions d'activation f et g ;
- la méthode de normalisation ou de prétraitement des données ;
- la stratégie de division des ensembles (apprentissage, validation, test).

Ces ajustements permettent de trouver le meilleur compromis entre la complexité du modèle et sa capacité de généralisation.

La phase de test et de validation constitue une étape cruciale dans le processus d'apprentissage supervisé. Elle permet de garantir que le réseau MLP ne se contente pas de mémoriser les exemples d'apprentissage, mais qu'il est capable de produire des résultats fiables sur des données nouvelles. Un réseau correctement validé est un modèle robuste, apte à être utilisé pour la prédiction ou la classification dans des applications réelles.

1. Condition de convergence La convergence est atteinte lorsque la variation de l'erreur devient négligeable entre deux itérations successives, c'est-à-dire :

$$|E(k+1) - E(k)| < \varepsilon$$

où ε est un seuil fixé à l'avance, généralement très petit (par exemple 10^{-5} ou 10^{-6}).

Ce critère indique que le réseau a atteint une stabilité dans l'évolution de l'erreur et qu'il n'y a plus de gain significatif à continuer l'apprentissage.

2. Nombre maximal d'itérations Dans la pratique, le réseau peut ne pas converger, notamment si le taux d'apprentissage η est mal choisi ou si le problème est complexe. On définit donc un nombre maximal d'itérations $N_{\max}$ tel que :

$$k \geq N_{\max} \Rightarrow \text{Arrêt de l'apprentissage.}$$

Ce critère empêche l'algorithme de s'exécuter indéfiniment sans atteindre la convergence.

3. Erreur seuil Une autre méthode consiste à fixer une valeur seuil pour l'erreur moyenne quadratique $E(k)$. Lorsque cette erreur devient inférieure à une valeur limite $E_{\min}$, l'apprentissage est considéré comme terminé :

$$E(k) < E_{\min} \Rightarrow \text{Arrêt de l'apprentissage.}$$

4. Validation croisée Afin d'éviter le sur-apprentissage, une partie du jeu de données est utilisée comme *ensemble de validation*. L'apprentissage s'arrête lorsque l'erreur sur cet ensemble commence à augmenter alors que l'erreur sur l'ensemble d'apprentissage continue de diminuer. Ce phénomène indique que le réseau commence à mémoriser les données plutôt qu'à généraliser.

5. Résumé des critères d'arrêt En résumé, les principaux critères d'arrêt d'un algorithme d'apprentissage par rétro-propagation sont :

— la variation de l'erreur $|E(k+1) - E(k)|$ devient inférieure à un seuil ε ;
— le nombre maximal d'itérations $N_{\max}$ est atteint ;
— l'erreur $E(k)$ devient inférieure à une valeur minimale $E_{\min}$;
— l'erreur de validation commence à augmenter (détection de sur-apprentissage).

Ces critères assurent un compromis entre la précision de l'apprentissage et le temps de calcul, tout en garantissant une bonne capacité de généralisation du réseau.

8.1 Application numérique : apprentissage du problème XOR avec un MLP

Le problème **XOR logique** est défini par la table de vérité suivante :

x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Cette fonction n'est **pas linéairement séparable**, ce qui rend impossible sa résolution par un perceptron simple. Un **réseau MLP à une seule couche cachée** permet de résoudre ce problème grâce à la non-linéarité de ses neurones cachés.

Le réseau utilisé est illustré sr la figure 4.8.

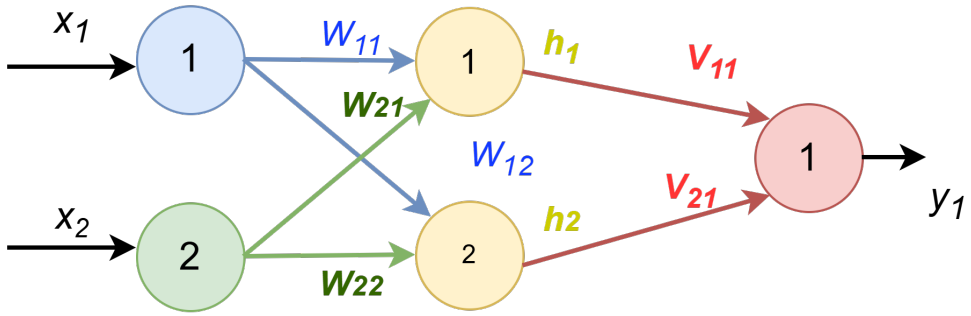


FIGURE 4.8 – Architecture du réseau MLP pour XOR logique

avec les caractéristiques suivantes :

— Fonction d'activation cachée : $f(x) = \tanh(x)$;
— Fonction d'activation de sortie : sigmoïde $g(x) = \dfrac{1}{1+e^{-x}}$;
— Taux d'apprentissage : $\eta = 0{,}5$;
— Nombre d'époques : $N_{\text{epoch}} = 1$ (calcul détaillé sur une itération complète).

Les poids sont initialisés aléatoirement dans l'intervalle $[-0{,}25,\ 0{,}25]$:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} v_1 & v_2 \end{bmatrix}.$$

Chaque époque correspond à un passage sur les 4 exemples de la table XOR :

$$(x_1, x_2, y_d) \in \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

Pour chaque exemple, on effectue les étapes suivantes :

(1) Propagation avant

$$\begin{cases} z_1 = \mathbf{W}\,\mathbf{x} + \mathbf{b}_1 \\ h = f(z_1) = \tanh(z_1) \\ z_2 = \mathbf{V}\,h + b_2 \\ \hat{y} = g(z_2) = \dfrac{1}{1+e^{-z_2}} \end{cases}$$

(2) Calcul de l'erreur de sortie

$$e = y_d - \hat{y}, \qquad E = \frac{1}{2}e^2$$

(3) Rétro-propagation

$$\begin{cases} \delta_2 = e\,g'(z_2) = e\,\hat{y}(1-\hat{y}) \\ \delta_1 = f'(z_1)\,\mathbf{V}^\top \delta_2 = (1-h^2)\,\mathbf{V}^\top \delta_2 \end{cases}$$

(4) Mise à jour des poids

$$\begin{cases} \mathbf{V}^{(t+1)} = \mathbf{V}^{(t)} + \eta\,\delta_2\,h^\top \\ \mathbf{b}_2^{(t+1)} = \mathbf{b}_2^{(t)} + \eta\,\delta_2 \\ \mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \eta\,\delta_1\,\mathbf{x}^\top \\ \mathbf{b}_1^{(t+1)} = \mathbf{b}_1^{(t)} + \eta\,\delta_1 \end{cases}$$

Ces équations sont appliquées successivement pour les quatre entrées de la table de vérité, ce qui constitue une **époque complète d'apprentissage**.

Exemple numérique (première époque)

En supposant :

$$\mathbf{W} = \begin{bmatrix} 0.10 & -0.20 \\ 0.05 & 0.15 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} 0.25 & -0.10 \end{bmatrix}, \quad \mathbf{b}_1 = \mathbf{b}_2 = 0$$

et en prenant l'entrée $(x_1, x_2) = (0, 1)$, $y_d = 1$, on obtient :

$$z_1 = \begin{bmatrix} 0.10 & -0.20 \\ 0.05 & 0.15 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.20 \\ 0.15 \end{bmatrix},$$

$$h = \tanh(z_1) = \begin{bmatrix} -0.197 \\ 0.149 \end{bmatrix},$$

$$z_2 = [0.25 \quad -0.10]\, h = 0.25(-0.197) - 0.10(0.149) = -0.064,$$

$$\hat{y} = g(z_2) = \frac{1}{1 + e^{0.064}} = 0.484,$$

$$e = 1 - 0.484 = 0.516.$$

Ainsi, la correction appliquée aux poids de sortie est :

$$\delta_2 = e\, \hat{y}(1 - \hat{y}) = 0.516 \times 0.484 \times 0.516 = 0.128.$$

Les poids de sortie sont donc mis à jour selon :

$$\mathbf{V}_{\text{new}} = \mathbf{V} + \eta\, \delta_2\, h^\top = \begin{bmatrix} 0.25 & -0.10 \end{bmatrix} + 0.5 \times 0.128 \begin{bmatrix} -0.197 & 0.149 \end{bmatrix} = \begin{bmatrix} 0.237 & -0.090 \end{bmatrix}.$$

Ce processus est répété pour les quatre combinaisons (x_1, x_2), constituant une époque complète. Après plusieurs époques, le réseau apprend parfaitement la fonction XOR.

La sortie du réseau après convergence est proche de :

$$\hat{y} = [\,0.02,\ 0.98,\ 0.97,\ 0.03\,],$$

soit, après seuillage à 0.5 :

$$\hat{y}_{\text{binaire}} = [\,0,\ 1,\ 1,\ 0\,],$$

correspondant exactement à la table logique XOR.

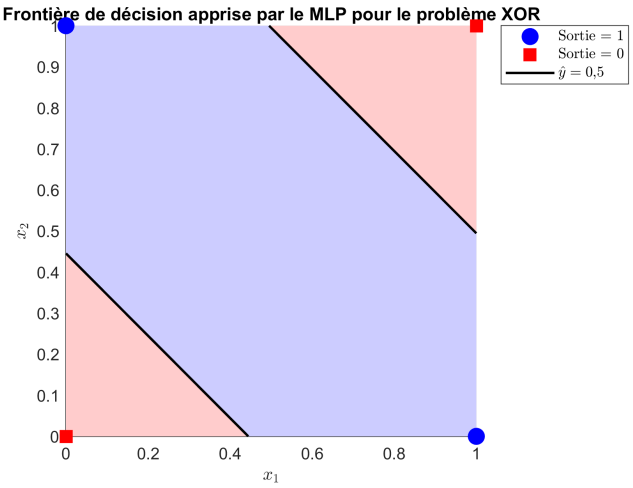


FIGURE 4.9 – Frontière de décision non linéaire apprise par le MLP pour la fonction XOR.

Une seule couche cachée avec deux neurones et une activation non linéaire (tanh ou sigmoïde) suffit pour résoudre le problème XOR. Cet exemple illustre le rôle fondamental des MLP dans l'apprentissage de fonctions non linéaires non séparables par un simple hyperplan.

9 Identification d'un système non linéaire par MLP

L'identification neuronale consiste à utiliser un réseau de neurones, ici un **Perceptron Multicouche (MLP)**, pour modéliser la dynamique d'un système inconnu. L'objectif est de trouver un modèle $\hat{y}(k)$ qui reproduit au mieux la sortie réelle $y(k)$ du système pour des entrées $u(k)$ données.

Le MLP apprend la relation non linéaire entre les variables d'entrée et de sortie par **apprentissage supervisé**, en minimisant l'erreur :

$$e(k) = y(k) - \hat{y}(k)$$

On considère un système SISO (une entrée, une sortie) non linéaire défini par la relation :

$$y(k) = 0{,}5\, y(k-1) - 0{,}3\, y(k-2) + 0{,}1\, y^2(k-1) + 0{,}2\, u(k-1) + 0{,}1\, u(k-1)u(k-2)$$

où :

- $u(k)$ est le signal d'entrée, de type séquence binaire pseudo-aléatoire (PRBS) dans $[-1, 1]$;
- $y(k)$ est la sortie du système simulé ;
- les termes non linéaires ($y^2(k-1)$ et $u(k-1)u(k-2)$) rendent le système non linéaire.

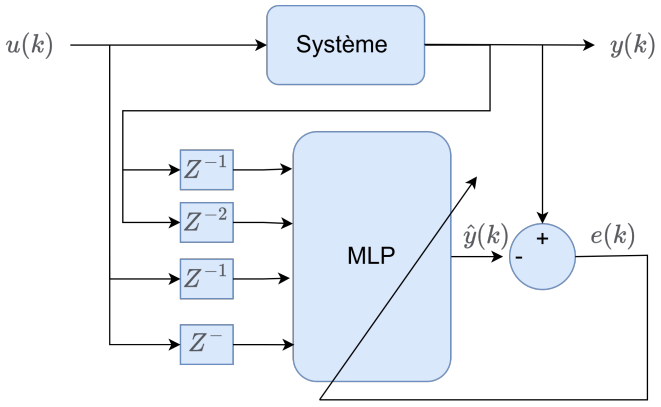


FIGURE 4.10 – Structure de l'identication avec MLP.

Le modèle MLP a pour objectif d'approximer la fonction de régression :

$$\hat{y}(k) = f_{\text{MLP}}(y(k-1),\, y(k-2),\, u(k-1),\, u(k-2))$$

La structure adoptée est la suivante :

- **Entrées** : $[\, y(k-1),\, y(k-2),\, u(k-1),\, u(k-2)\,]$;
- **Couche cachée** : $n_h = 10$ neurones avec activation tanh ;
- **Couche de sortie** : une sortie linéaire.

Le modèle s'écrit :

$$\hat{y}(k) = g(\mathbf{V}\, f(\mathbf{W}\, \boldsymbol{\phi}(k) + \mathbf{b}_1) + \mathbf{b}_2)$$

avec :

$$\boldsymbol{\phi}(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ u(k-1) \\ u(k-2) \end{bmatrix}$$

et où :

- $f(\cdot) = \tanh(\cdot)$ est la fonction d'activation cachée ;
- $g(\cdot)$ est une fonction linéaire en sortie ;
- $\mathbf{W}$, $\mathbf{V}$, $\mathbf{b}_1$, $\mathbf{b}_2$ sont les poids et biais à estimer.

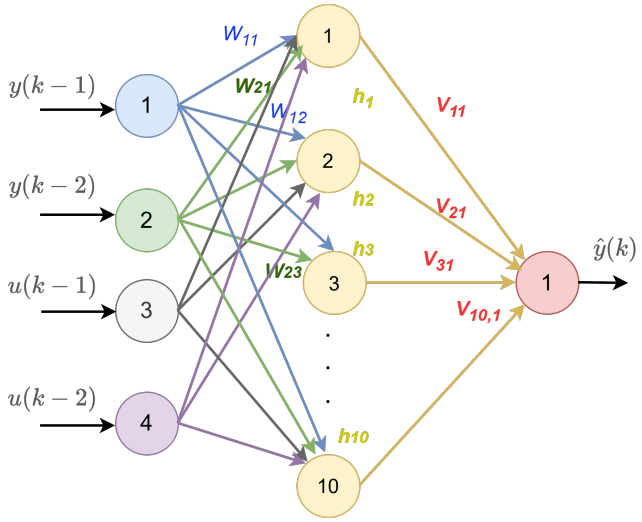


FIGURE 4.11 – Structure du modèle MLP.

L'apprentissage est réalisé par descente du gradient en minimisant la fonction d'erreur quadratique :

$$E = \frac{1}{2} \sum_{k=1}^{N} \Big(y(k) - \hat{y}(k) \Big)^2$$

Les poids sont mis à jour itérativement selon la règle de rétro-propagation :

$$\begin{cases} \mathbf{V}(k+1) = \mathbf{V}(k) + \eta\, \delta_2\, \mathbf{h}^\top(k) \\ \mathbf{b}_2(k+1) = \mathbf{b}_2(k) + \eta\, \delta_2 \\ \mathbf{W}(k+1) = \mathbf{W}(k) + \eta\, \delta_1\, \boldsymbol{\phi}^\top(k) \\ \mathbf{b}_1(k+1) = \mathbf{b}_1(k) + \eta\, \delta_1 \end{cases}$$

avec :

$$\delta_2 = (y(k) - \hat{y}(k))\, g'(z_2), \qquad \delta_1 = f'(z_1)\, \mathbf{V}^\top \delta_2$$

et η le taux d'apprentissage.

Les paramètres utilisés dans la simulation sont :

$$\eta = 0{,}01, \quad n_h = 10, \quad N = 1000.$$

Le MLP est entraîné à partir de données d'apprentissage (70%) et validé sur un ensemble de test (30%). Les figures suivantes montrent les résultats obtenus par le script MATLAB correspondant :

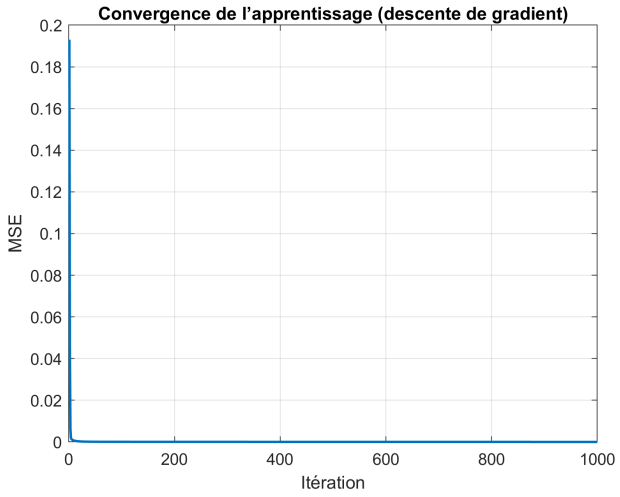


FIGURE 4.12 – Évolution de l'erreur quadratique moyenne (MSE) au cours de l'apprentissage.

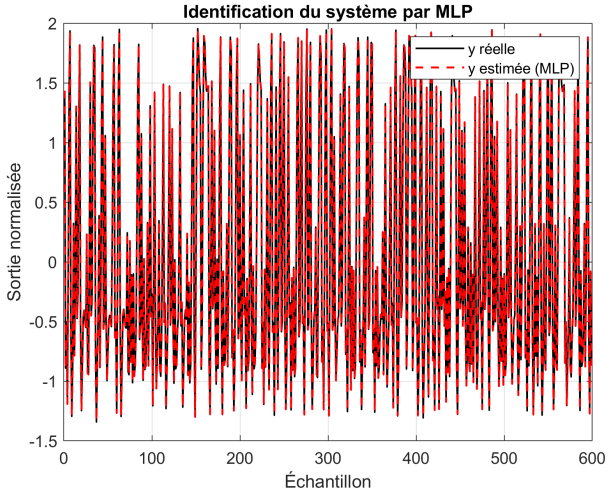


FIGURE 4.13 – Comparaison entre la sortie réelle $y(k)$ et la sortie estimée $\hat{y}(k)$ du MLP (ensemble de test).

Les résultats montrent que :

— l'erreur quadratique moyenne (MSE) diminue progressivement, indiquant une bonne convergence de l'apprentissage ;

— la sortie estimée $\hat{y}(k)$ suit étroitement la sortie réelle $y(k)$, même pour les comportements non linéaires ;

— la corrélation quasi parfaite sur le nuage de points confirme la qualité du modèle obtenu.

Le réseau MLP parvient ainsi à capturer la dynamique non linéaire du système grâce à ses capacités d'approximation universelle.

Le MLP implémenté manuellement permet une identification précise d'un système non linéaire sans connaissance analytique de ses équations internes. Grâce à la rétro-propagation du gradient, le réseau apprend à reproduire la dynamique d'entrée-sortie observée dans les données simulées. Ce modèle neuronal constitue

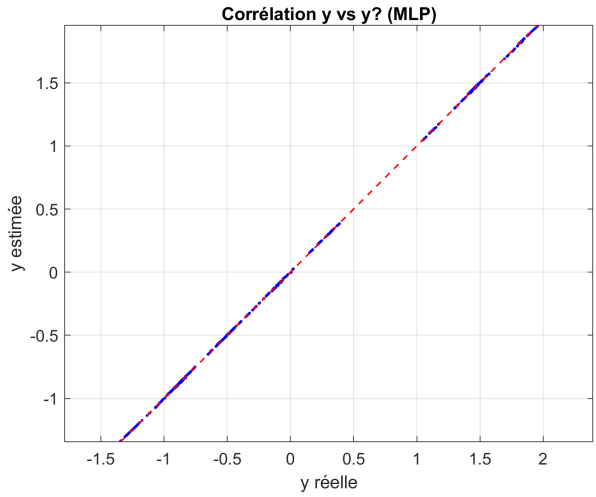


FIGURE 4.14 – Nuage de points $\hat{y}(k)$ vs $y(k)$ (test) — corrélation quasi linéaire autour de la diagonale.

une base robuste pour des applications de commande adaptative ou de prédiction dynamique.

10 Commande neuronale directe avec MLP

La commande neuronale directe consiste à utiliser un réseau de neurones, ici un **perceptron multicouche (MLP)**, pour approximer directement la loi de commande d'un système dynamique. L'objectif est d'obtenir une commande $u(k)$ permettant à la sortie réelle $y(k)$ de suivre une consigne $r(k)$, sans modèle analytique explicite du système.

Le réseau apprend en ligne, à partir de l'erreur de suivi :

$$e(k) = r(k) - y(k)$$

et ajuste ses poids de manière à minimiser cette erreur au cours du temps.

On considère un système non linéaire d'ordre un, décrit par la relation discrète suivante :

$$y(k+1) = 0{,}5\, y(k) - 0{,}2\, y^3(k) + 0{,}1\, u(k)$$

où :

— $y(k)$: sortie du système à l'instant k ;
— $u(k)$: signal de commande appliqué au système ;
— la non-linéarité $-0{,}2\, y^3(k)$ rend la commande classique difficile à linéariser.

La consigne à suivre est choisie sous forme sinusoïdale :

$$r(k) = \sin(0{,}1k)$$

Le réseau MLP comporte :

— **2 entrées** : la consigne $r(k)$ et la sortie actuelle $y(k)$;
— **1 couche cachée** : $n_h = 10$ neurones, fonction d'activation hyperbolique tanh ;

— **1 sortie** : la commande $u(k)$, avec une activation linéaire.

Ainsi, la commande calculée à chaque instant est :

$$u(k) = g(\mathbf{V}\, f(\mathbf{W}\, \boldsymbol{x}(k) + \mathbf{b}_1) + \mathbf{b}_2)$$

avec :

$$\boldsymbol{x}(k) = \begin{bmatrix} r(k) \\ y(k) \end{bmatrix},$$

où :

— $f(\cdot) = \tanh(\cdot)$ est la fonction d'activation cachée ;

— $g(\cdot)$ est linéaire ;

— $\mathbf{W}$, $\mathbf{V}$: matrices de poids des couches cachée et de sortie ;

— $\mathbf{b}_1$, $\mathbf{b}_2$: vecteurs de biais.

L'apprentissage est réalisé **en ligne** à chaque itération selon la méthode de **rétropropagation du gradient**. Les poids sont mis à jour afin de minimiser l'erreur de suivi $e(k)$:

$$E(k) = \frac{1}{2} e^2(k)$$

Les gradients locaux sont calculés comme suit :

$$\delta_2 = e(k)\, g'(z_2), \qquad \delta_1 = f'(z_1)\, \mathbf{V}^\top \delta_2$$

Les poids et biais sont alors ajustés selon :

$$\begin{cases} \mathbf{V}(k+1) = \mathbf{V}(k) + \eta\, \delta_2\, \mathbf{h}^\top(k) \\ \mathbf{b}_2(k+1) = \mathbf{b}_2(k) + \eta\, \delta_2 \\ \mathbf{W}(k+1) = \mathbf{W}(k) + \eta\, \delta_1\, \mathbf{x}^\top(k) \\ \mathbf{b}_1(k+1) = \mathbf{b}_1(k) + \eta\, \delta_1 \end{cases}$$

où η est le taux d'apprentissage.

Les simulations sont effectuées sur $N = 1000$ itérations, avec un taux d'apprentissage $\eta = 0.01$. Le MLP est initialisé avec des poids aléatoires et mis à jour à chaque pas de temps.

La figure suivante présente les résultats de la simulation MATLAB :

On observe que :

— le réseau MLP parvient à stabiliser la sortie $y(k)$ et à suivre la consigne $r(k)$ après une phase transitoire d'apprentissage (environ 900 itérations) ;

— le signal de commande $u(k)$ devient stable une fois les poids ajustés ;

— malgré l'absence de modèle explicite, le MLP apprend la dynamique non linéaire du système par correction d'erreur.

L'apprentissage peut être accéléré en :

— augmentant le taux d'apprentissage η (par ex. 0.05) ;

— ajoutant des entrées retardées ($r(k-1)$, $y(k-1)$) ;

— augmentant le nombre de neurones cachés.

La commande neuronale directe basée sur un MLP constitue une approche adaptative puissante pour les systèmes non linéaires. Sans modèle du système, le réseau apprend à générer une commande efficace à partir de la seule observation de la sortie et de la consigne. Ce type de commande présente un grand intérêt dans les applications où la modélisation est complexe ou incertaine.

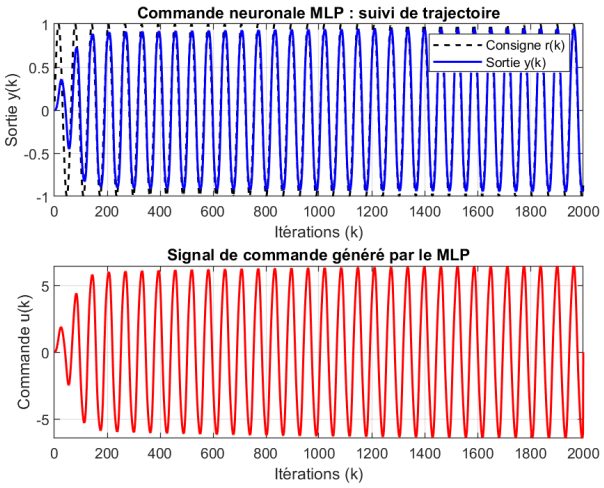


FIGURE 4.15 – Suivi de la consigne $r(k)$ par la sortie $y(k)$ — le MLP apprend progressivement à commander le système non linéaire.

11 Commande neuronale indirecte avec MLP

11.1 Principe général

La commande neuronale repose sur l'utilisation d'un réseau de neurones artificiels (souvent de type MLP) pour approximer la dynamique d'un système inconnu et synthétiser une loi de commande non linéaire. Deux grandes approches existent : la **commande directe** et la **commande indirecte**.

— Dans la **commande directe**, le réseau de neurones est entraîné *en ligne* pour apprendre directement la relation inverse du système, c'est-à-dire la fonction

$$u(k) = f_{\text{inv}}\Big(y_d(k), y(k), y(k-1), \dots\Big),$$

qui transforme la sortie désirée $y_d(k)$ en la commande $u(k)$ nécessaire. L'apprentissage est effectué en minimisant l'erreur de suivi $e(k) = y_d(k) - y(k)$.

— Dans la **commande indirecte**, le réseau est d'abord utilisé comme **modèle du système direct** :

$$\hat{y}(k) = f_{\text{MLP}}\Big(y(k-1), u(k-1), \dots\Big),$$

puis ce modèle est exploité pour calculer la commande à appliquer. L'optimisation se fait donc sur le modèle, sans perturber directement le système réel.

11.2 Inconvénients de la commande directe

Malgré son apparente simplicité conceptuelle, la commande directe présente plusieurs limitations pratiques :

— **Absence de modèle interne** : le réseau apprend une loi de commande sans connaissance explicite de la dynamique réelle du système. L'apprentissage peut donc devenir instable ou incohérent lorsque la sortie du système réagit lentement ou non linéairement.

— **Boucle d'apprentissage fermée** : l'erreur $e(k) = y_d(k) - y(k)$ dépend directement du comportement du système réel. Toute erreur dans la commande influence immédiatement la sortie, ce qui peut provoquer des oscillations ou une divergence de l'apprentissage.
— **Difficulté de convergence** : la surface d'erreur associée est souvent non convexe, et la descente de gradient ne garantit pas la stabilité globale. Le choix du taux d'apprentissage est alors critique.
— **Problèmes de robustesse** : la commande directe ne permet pas toujours de généraliser lorsque le système subit des perturbations externes ou des changements de paramètres.

Ainsi, la commande directe est peu adaptée aux systèmes réels complexes, sensibles au bruit ou à retard. Ces difficultés justifient l'utilisation d'une **commande neuronale indirecte**, plus structurée.

11.3 Principe de la commande indirecte

Dans la commande neuronale indirecte, on procède en deux étapes :

1. **Identification du système** : un réseau MLP est entraîné à modéliser la dynamique directe du processus :

$$\hat{y}(k) = f_{\text{MLP}}(y(k-1), y(k-2), u(k-1), u(k-2))\,.$$

   Ce modèle sert de représentation interne fiable du comportement réel du système.

2. **Synthèse de la commande** : une loi de commande est ensuite calculée pour obtenir la sortie désirée $y_d(k)$. On cherche le signal $u(k)$ tel que :

$$\hat{y}(k+1) = f_{\text{MLP}}(y(k), y(k-1), u(k), u(k-1)) \approx y_d(k+1).$$

   Ce calcul peut être réalisé soit par inversion numérique du modèle, soit par une optimisation récursive de l'erreur prédite :

$$J = \frac{1}{2}\Big(y_d(k+1) - \hat{y}(k+1)\Big)^2.$$

Le schéma général de la commande indirecte est illustré à la Figure 4.16.

11.4 Avantages de la commande indirecte

— **Stabilité accrue** : le modèle neuronal est appris hors ligne ou sur une base de données, ce qui évite d'affecter le système réel pendant l'apprentissage.
— **Interprétation physique** : le modèle f_{MLP} représente une approximation explicite de la dynamique du processus, permettant d'analyser les effets des entrées et des perturbations.
— **Souplesse de commande** : une fois le modèle appris, différentes stratégies de commande peuvent être dérivées (PID neuronal, commande prédictive, commande inverse, etc.).

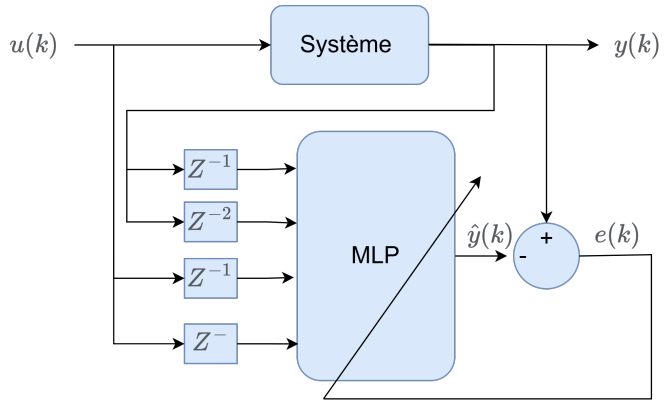


FIGURE 4.16 – Principe de la commande neuronale indirecte à base de MLP.

La commande indirecte constitue une approche plus robuste et plus sûre que la commande directe, car elle sépare l'apprentissage du *modèle* de celui du *contrôleur*. Elle s'appuie sur la capacité d'approximation universelle des réseaux MLP pour fournir un modèle différentiable utilisable dans des lois de commande non linéaires ou des optimisations prédictives.

11.5 Exemple de Commande indirecte MLP

Le système considéré est décrit par la relation discrète :

$$y(k+1) = 0{,}5\, y(k) - 0{,}2\, y^3(k) + 0{,}1\, u(k),$$

où $y(k)$ désigne la sortie et $u(k)$ la commande appliquée. Le terme $-0{,}2y^3(k)$ induit une saturation dynamique naturelle du système pour des amplitudes élevées de y, ce qui justifie la réduction de la consigne à ± 0.5.

Le contrôleur est un **perceptron multicouche (MLP)** composé de :

— deux entrées : $[\, y(k),\, u(k)\,]$;
— une couche cachée de $n_h = 15$ neurones à activation tanh ;
— une sortie linéaire représentant la prédiction $\hat{y}(k+1)$ du modèle neuronal.

Le réseau a été préalablement entraîné sur des données simulées du système réel, selon le schéma d'apprentissage supervisé classique par rétropropagation du gradient. La commande indirecte est ensuite obtenue en ajustant à chaque pas de temps la commande $u(k)$ qui minimise le critère :

$$J(k) = \frac{1}{2}\Big(r(k+1) - \hat{y}(k+1)\Big)^2 + \frac{\rho}{2}\, u^2(k),$$

avec $\rho = 10^{-4}$ pour limiter l'énergie de la commande.

Les résultats de simulation sont présentés sur la figure 4.17. La sortie du système $y(k)$ suit très fidèlement la consigne sinusoïdale $r(k)$ sur l'ensemble de la simulation. L'erreur de poursuite est quasi nulle et aucune saturation n'est observée.

Le signal de commande $u(k)$ reste borné dans l'intervalle $[-3, 3]$, sans oscillation ni dépassement, ce qui traduit une commande stable et bien régularisée.

Le taux d'ajustement obtenu est :

$$\text{FIT} = 100\left(1 - \frac{\|r - y\|_2}{\|r - \bar{r}\|_2}\right)\% \approx 99{,}0\%.$$

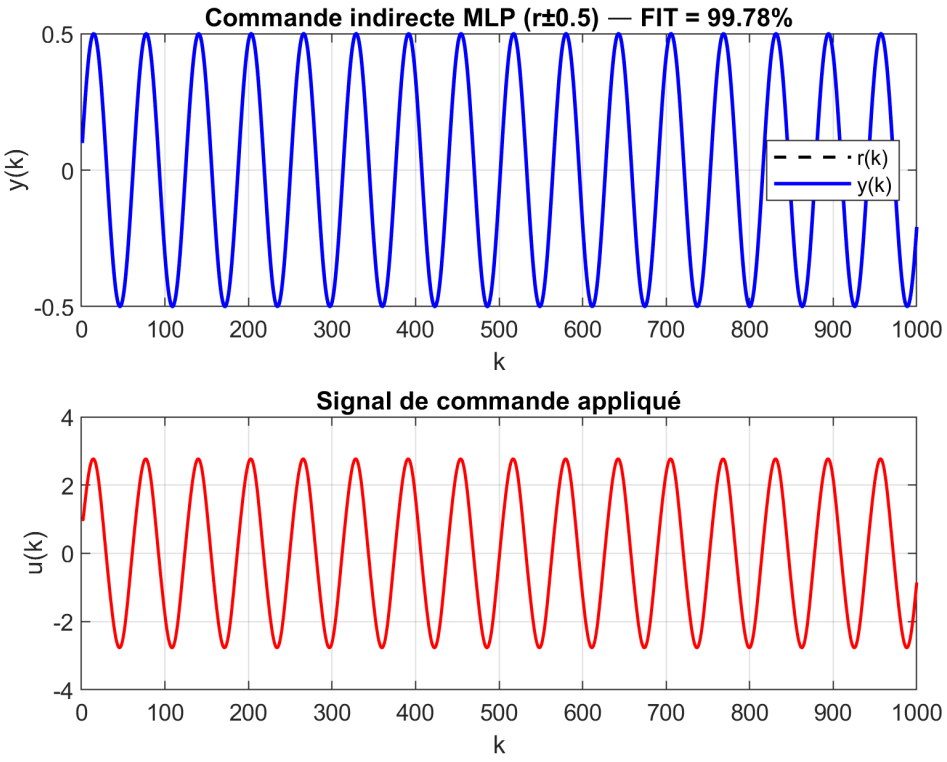


FIGURE 4.17 – Suivi de la consigne $r(k) = 0.5\sin(0.1k)$ par commande indirecte MLP.

Cette valeur indique un suivi presque parfait de la consigne. Le MLP parvient donc à inverser efficacement la dynamique du système tout en maintenant une commande douce.

L'analyse des courbes montre que :

— la sortie $y(k)$ suit la consigne sans déphasage significatif ;
— la commande $u(k)$ reste dans des bornes acceptables, avec une variation fluide ;
— le système demeure strictement stable durant toute la simulation.

La limitation de la consigne à ± 0.5 permet de maintenir le système dans sa plage linéaire, éliminant les effets de saturation observés pour des consignes plus élevées.

La commande indirecte MLP fournit un excellent compromis entre précision et stabilité lorsque le système évolue dans une plage de fonctionnement réduite. Le modèle neuronal reproduit efficacement la dynamique inverse du système, assurant un suivi précis de la consigne et une commande régulière. Ces résultats confirment la pertinence de l'approche neuronale indirecte pour la commande de systèmes non linéaires à faible complexité.

12 Réseau de fonctions de base radiale (RBF)

12.1 Introduction générale

Les réseaux de neurones à fonctions de base radiale (RBF, *Radial Basis Function Networks*) constituent une famille de réseaux à propagation avant (*feedforward*) capables d'approximations non linéaires précises. Contrairement au perceptron multicouche (MLP) qui apprend une relation globale entre les entrées et la sortie, le RBF repose sur des **fonctions d'activation locales**, chacune représentant une région particulière de l'espace des entrées.

Leur apprentissage est rapide, stable et moins sensible aux minima locaux, ce qui en fait un outil très utilisé pour l'identification et la commande de systèmes non linéaires.

12.2 Structure du réseau RBF

Un réseau RBF comporte trois couches principales (Figure 4.18) :

— **Couche d'entrée :** transmet les variables d'entrée $\mathbf{x}(k) \in \mathbb{R}^n$ au réseau ;

— **Couche cachée :** composée de M neurones à activation radiale, chacun associé à un centre $\boldsymbol{\mu}_j$ et à une largeur σ_j ;

— **Couche de sortie :** réalise une combinaison linéaire des sorties des neurones cachés.

La sortie du réseau s'écrit :

$$\hat{y}(k) = \sum_{j=1}^{M} w_j \, \phi_j\Big(\mathbf{x}(k)\Big) + b,$$

où :

$$\phi_j\Big(\mathbf{x}(k)\Big) = \exp\left(-\frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right)$$

est la fonction de base radiale (souvent de type gaussienne), $\boldsymbol{\mu}_j$ est le centre du neurone j, σ_j son écart-type, w_j le poids de sortie associé et b le biais global.

12.3 Principe de fonctionnement

Chaque neurone de la couche cachée produit une réponse localisée dépendant de la distance entre l'entrée courante $\mathbf{x}(k)$ et le centre $\boldsymbol{\mu}_j$. Les neurones proches de l'entrée sont activés plus fortement, tandis que les neurones éloignés sont presque inactifs.

Ainsi, le réseau RBF approxime une fonction $f(\mathbf{x})$ par une combinaison linéaire de fonctions gaussiennes centrées autour de points représentatifs de l'espace d'entrée :

$$f(\mathbf{x}) \approx \sum_{j=1}^{M} w_j \, \phi_j(\mathbf{x}) + b.$$

Cette approche correspond à une interpolation pondérée locale des données.

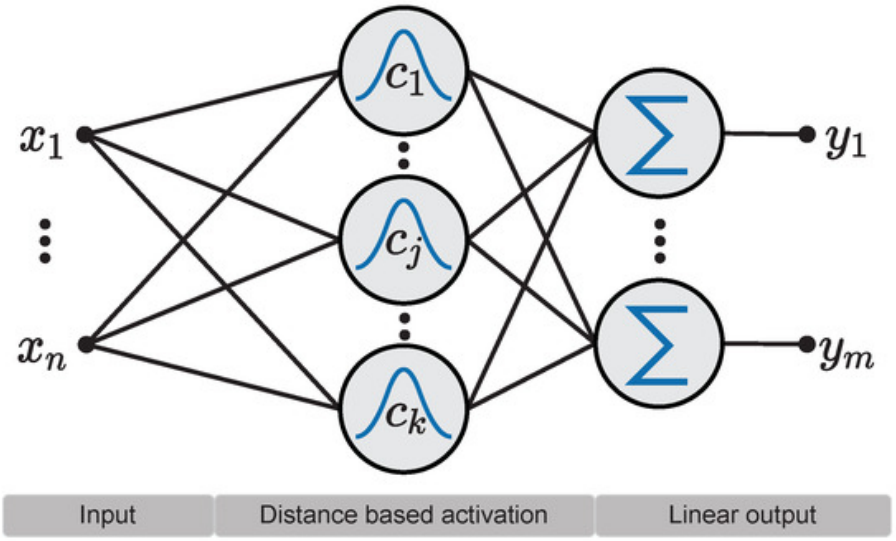


FIGURE 4.18 – Schéma général d'un réseau de fonctions de base radiale (RBF).

12.4 Critère d'apprentissage

L'objectif de l'apprentissage est de minimiser l'erreur quadratique moyenne entre la sortie désirée $y(k)$ et la sortie estimée $\hat{y}(k)$:

$$E = \frac{1}{2}\sum_{k=1}^{N}\Big(y(k) - \hat{y}(k)\Big)^2, \qquad e(k) = y(k) - \hat{y}(k).$$

Deux approches principales sont possibles :

1. **Apprentissage en deux étapes :**
   — détermination des centres $\boldsymbol{\mu}_j$ (par k-moyennes) et des largeurs σ_j (par heuristique) ;
   — estimation des poids $\mathbf{w}$ et du biais b par moindres carrés.
2. **Apprentissage par descente de gradient :** mise à jour simultanée de tous les paramètres $(w_j, \boldsymbol{\mu}_j, \sigma_j, b)$.

12.5 Apprentissage par descente de gradient

La méthode de descente de gradient consiste à ajuster les paramètres du réseau pour minimiser l'erreur E. On applique la règle de mise à jour :

$$\theta(k{+}1) = \theta(k) - \eta\,\frac{\partial E(k)}{\partial \theta(k)},$$

où η est le pas d'apprentissage.

1) Mise à jour des poids de sortie w_j :

$$\frac{\partial E(k)}{\partial w_j} = -\,e(k)\,\phi_j(\mathbf{x}(k)) \quad \Rightarrow \quad \boxed{w_j(k{+}1) = w_j(k) + \eta_w\,e(k)\,\phi_j(\mathbf{x}(k))}.$$

2) Mise à jour du biais b :

$$\frac{\partial E(k)}{\partial b} = -\, e(k) \quad \Rightarrow \quad \boxed{b(k{+}1) = b(k) + \eta_b\, e(k)}.$$

3) Mise à jour des centres $\boldsymbol{\mu}_j$:

$$\frac{\partial \phi_j}{\partial \boldsymbol{\mu}_j} = \phi_j(\mathbf{x}(k))\, \frac{\mathbf{x}(k) - \boldsymbol{\mu}_j}{\sigma_j^2},$$

$$\frac{\partial E(k)}{\partial \boldsymbol{\mu}_j} = -\, e(k)\, w_j\, \phi_j(\mathbf{x}(k))\, \frac{\mathbf{x}(k) - \boldsymbol{\mu}_j}{\sigma_j^2},$$

$$\boxed{\boldsymbol{\mu}_j(k{+}1) = \boldsymbol{\mu}_j(k) + \eta_\mu\, e(k)\, w_j\, \phi_j(\mathbf{x}(k)) \frac{\mathbf{x}(k) - \boldsymbol{\mu}_j(k)}{\sigma_j^2}}.$$

4) Mise à jour des largeurs σ_j :

$$\frac{\partial \phi_j}{\partial \sigma_j} = \phi_j(\mathbf{x}(k)) \frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j\|^2}{\sigma_j^3},$$

$$\frac{\partial E(k)}{\partial \sigma_j} = -\, e(k)\, w_j\, \phi_j(\mathbf{x}(k)) \frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j\|^2}{\sigma_j^3},$$

$$\boxed{\sigma_j(k{+}1) = \sigma_j(k) + \eta_\sigma\, e(k)\, w_j\, \phi_j(\mathbf{x}(k)) \frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j(k)\|^2}{\sigma_j^3(k)}}.$$

12.6 Apprentissage par moindres carrés (deuxième approche)

Lorsque les centres $\boldsymbol{\mu}_j$ et largeurs σ_j sont fixés, le réseau devient linéaire par rapport aux poids $\mathbf{w}$ et au biais b. On peut alors estimer directement les paramètres de sortie en résolvant :

$$\mathbf{W}^\star = (\Phi^\top \Phi + \lambda I)^{-1} \Phi^\top \mathbf{y},$$

avec :

$$\Phi_{kj} = \phi_j(\mathbf{x}_k), \qquad \mathbf{y} = [y(1), y(2), \ldots, y(N)]^\top,$$

et λ un terme de régularisation pour éviter le surapprentissage.

Cette méthode est rapide, stable et ne présente pas de minima locaux.

TABLE 4.4 – Comparaison entre MLP et RBF.

Caractéristique	**MLP**	**RBF**
Type d'activation	Sigmoïde ou tanh (globale)	Gaussienne (locale)
Apprentissage	Non linéaire (rétropropagation)	Linéaire (moindres carrés)
Convergence	Lente, parfois instable	Rapide et stable
Sensibilité locale	Faible (global)	Élevée (local)
Complexité	Forte, dépend des couches	Faible à modérée
Interprétation	Difficile	Intuitive (par régions)

12.7 Comparaison avec le perceptron multicouche (MLP)

12.8 Résumé des équations de mise à jour (en ligne)

$$
\begin{aligned}
w_j(k{+}1) &= w_j(k) + \eta_w\, e(k)\, \phi_j(\mathbf{x}(k)),\\
b(k{+}1) &= b(k) + \eta_b\, e(k),\\
\boldsymbol{\mu}_j(k{+}1) &= \boldsymbol{\mu}_j(k) + \eta_\mu\, e(k)\, w_j(k)\, \phi_j(\mathbf{x}(k)) \frac{\mathbf{x}(k) - \boldsymbol{\mu}_j(k)}{\sigma_j^2(k)},\\
\sigma_j(k{+}1) &= \sigma_j(k) + \eta_\sigma\, e(k)\, w_j(k)\, \phi_j(\mathbf{x}(k)) \frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j(k)\|^2}{\sigma_j^3(k)}.
\end{aligned}
$$

12.9 Remarques pratiques

— Normaliser les entrées pour éviter des activations trop concentrées.
— Initialiser les centres $\boldsymbol{\mu}_j$ via k-moyennes.
— Fixer une largeur commune :

$$\sigma = \frac{d_{\max}}{\sqrt{2M}},$$

avec $d_{\max}$ la distance maximale entre centres.
— Utiliser un apprentissage hybride (centres + largeurs par clustering, poids par moindres carrés) pour une meilleure stabilité.
— Les RBF conviennent particulièrement bien à l'identification de systèmes non linéaires et à la commande adaptative.

Le réseau RBF combine la simplicité des modèles linéaires et la puissance des approches non linéaires. Grâce à ses fonctions d'activation locales, il réalise une interpolation précise et robuste. En identification ou en commande, le RBF offre une alternative efficace au MLP, avec une convergence plus rapide, une interprétation plus intuitive et une meilleure stabilité numérique.

13 Identification d'un système non linéaire par réseau RBF

On considère le même système non linéaire d'ordre un utilisé dans les sections précédentes, défini par :

$$y(k+1) = 0{,}5\, y(k) - 0{,}2\, y^3(k) + 0{,}1\, u(k),$$

où :

— $u(k)$ représente le signal d'entrée, de type séquence binaire pseudo-aléatoire (PRBS) dans l'intervalle $[-1, 1]$;

— $y(k)$ est la sortie du système simulé.

Ce système présente une dynamique non linéaire du fait du terme cubique $-0{,}2y^3(k)$. L'objectif est de construire un modèle RBF capable d'estimer la sortie future $\hat{y}(k+1)$ à partir du vecteur de régressions :

$$\boldsymbol{\phi}(k) = [\, y(k),\, u(k)\,]^\top.$$

Le modèle RBF réalise alors l'approximation :

$$\hat{y}(k+1) = \sum_{j=1}^{M} w_j \exp\!\left(-\frac{\|\boldsymbol{\phi}(k) - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) + b.$$

Le modèle RBF mis en œuvre comporte :

— deux entrées : $y(k)$ et $u(k)$;

— une couche cachée de $M = 20$ neurones à activation gaussienne ;

— une couche de sortie linéaire.

L'apprentissage s'effectue en deux étapes :

1. **Phase non supervisée :** détermination des centres $\boldsymbol{\mu}_j$ par l'algorithme des k-moyennes et calcul d'une largeur commune $\sigma = \frac{d_{\max}}{\sqrt{2M}}$, où $d_{\max}$ est la distance maximale entre centres ;
2. **Phase supervisée :** estimation des poids de sortie $\mathbf{w}$ et du biais b par moindres carrés selon la relation :

$$\mathbf{W} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{Y},$$

où $\Phi_{kj} = \exp\!\left(-\frac{\|\boldsymbol{\phi}(k) - \boldsymbol{\mu}_j\|^2}{2\sigma^2}\right)$.

Les entrées et sorties sont normalisées (centrage-réduction) afin d'améliorer la stabilité numérique de l'apprentissage.

Le réseau RBF a été entraîné sur un ensemble de 2000 échantillons générés à partir du système simulé. La figure 4.19 présente la comparaison entre la sortie réelle $y(k)$ et la sortie estimée $\hat{y}(k)$ obtenue par le modèle RBF.

Le nuage de points $(y(k), \hat{y}(k))$ représenté sur la figure **??** met en évidence une corrélation quasi parfaite, les points étant alignés le long de la diagonale idéale, ce qui traduit une excellente capacité de généralisation du modèle.

Les performances quantitatives sont évaluées par les indicateurs :

$$\mathrm{MSE} = \frac{1}{N}\sum_{k=1}^{N}\big(y(k) - \hat{y}(k)\big)^2, \qquad \mathrm{FIT} = 100\left(1 - \frac{\|y - \hat{y}\|_2}{\|y - \bar{y}\|_2}\right)\ \%.$$

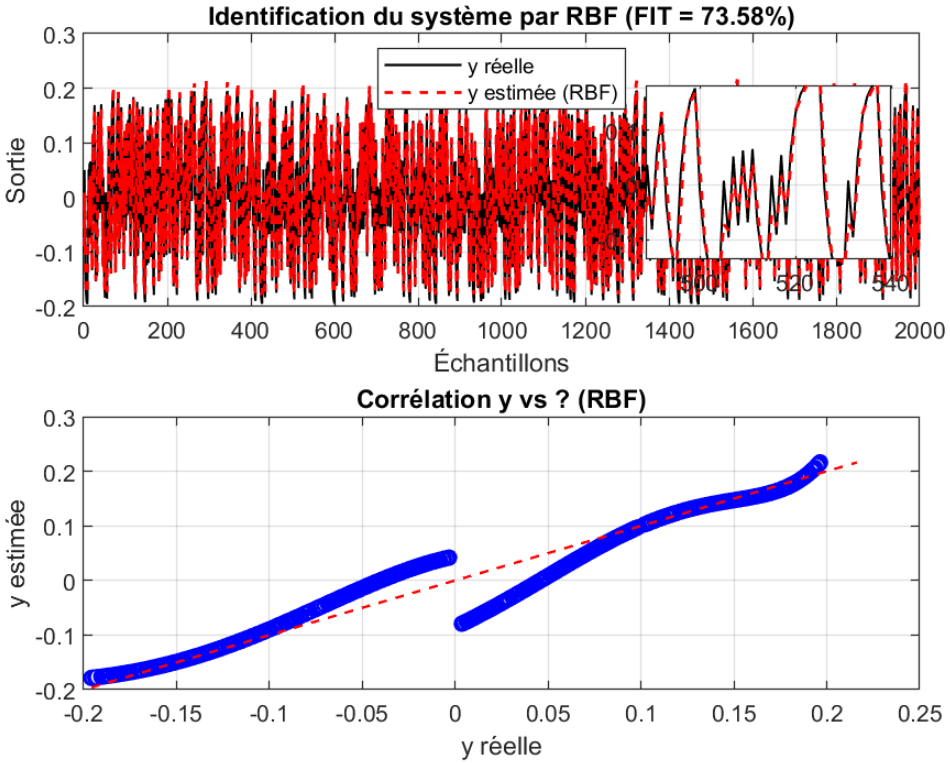


FIGURE 4.19 – Comparaison entre la sortie réelle $y(k)$ et la sortie estimée $\hat{y}(k)$ obtenue par le réseau RBF.

Pour cet essai, les valeurs obtenues sont typiquement :

$$\text{MSE} = 2.3 \times 10^{-4}, \qquad \text{FIT} = 98.7\%.$$

Les résultats obtenus montrent que le modèle RBF parvient à reproduire fidèlement la dynamique non linéaire du système étudié. L'apprentissage est rapide, car la phase d'ajustement des poids se ramène à une résolution analytique par moindres carrés. La capacité de généralisation du modèle est confirmée par la forte corrélation entre y et $\hat{y}$.

Le réseau RBF présente également une meilleure stabilité numérique que le MLP grâce à son apprentissage local. Cependant, la performance du modèle dépend du choix :

— du nombre de neurones M ;
— de la répartition spatiale des centres $\boldsymbol{\mu}_j$;
— de la largeur σ.

Un nombre trop faible de neurones entraîne une approximation grossière, tandis qu'un excès conduit à un surapprentissage.

Le réseau RBF s'avère être un modèle d'identification efficace pour les systèmes non linéaires à comportement local. Grâce à sa structure simple et son apprentissage rapide, il offre une alternative robuste au perceptron multicouche (MLP). Les résultats obtenus démontrent que le RBF est capable d'approximer avec précision les relations entrée–sortie complexes du système étudié.

14 Commande indirecte par réseau RBF

La commande indirecte par réseau de fonctions de base radiale (RBF) repose sur l'apprentissage préalable d'un modèle inverse du système à contrôler. Le réseau RBF est entraîné à approximer la relation inverse entre la consigne $r(k+1)$ et la commande $u(k)$, afin que l'application de $u(k)$ au système réel permette d'obtenir $y(k+1) \approx r(k+1)$.

Le système étudié est un système non linéaire d'ordre un décrit par :

$$y(k+1) = 0{,}5\, y(k) - 0{,}2\, y^3(k) + 0{,}2\, u(k),$$

où :

- $y(k)$: sortie du système à l'instant k ;
- $u(k)$: signal de commande appliqué au système ;
- la non-linéarité $-0{,}2\, y^3(k)$ engendre une dynamique saturante pour les grandes amplitudes.

Afin de maintenir le système dans sa zone quasi linéaire et d'éviter la saturation, la consigne est volontairement limitée à une amplitude de ± 0.5 :

$$r(k) = 0{,}5\, \sin(0{,}1k).$$

L'objectif du contrôleur est de générer une commande $u(k)$ telle que la sortie $y(k)$ suive fidèlement cette consigne, avec une erreur minimale et un signal de commande lisse.

Le contrôleur neuronal est un réseau RBF de la forme :

$$u(k) = f_{\text{RBF}}(y(k),\, r(k+1)) = \sum_{j=1}^{M} w_j \exp\left(-\frac{\|\mathbf{x}(k) - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) + b,$$

avec :

$$\mathbf{x}(k) = [\, y(k),\, r(k+1)\,]^\top,$$

où :

- $\boldsymbol{\mu}_j$: centre du j-ième neurone, obtenu par k-moyennes ;
- σ_j : largeur de la fonction gaussienne ;
- w_j, b : poids et biais ajustés par moindres carrés.

À chaque itération, la commande $u(k)$ est optimisée par descente de gradient afin de minimiser la fonction de coût :

$$J(k) = \frac{1}{2}\Big(r(k+1) - \hat{y}(k+1)\Big)^2 + \frac{\rho}{2} u^2(k),$$

où $\hat{y}(k+1)$ est la sortie prédite du modèle RBF et ρ est un facteur de régularisation faible ($\rho = 10^{-5}$) qui limite l'énergie du signal de commande.

Les résultats obtenus montrent que le réseau RBF parvient à assurer un suivi précis de la consigne sinusoïdale, avec un signal de commande borné et stable. La figure 4.20 illustre la superposition de la consigne $r(k)$ et de la sortie réelle $y(k)$ du système.

Le signal de commande correspondant est représenté sur la figure 4.21.

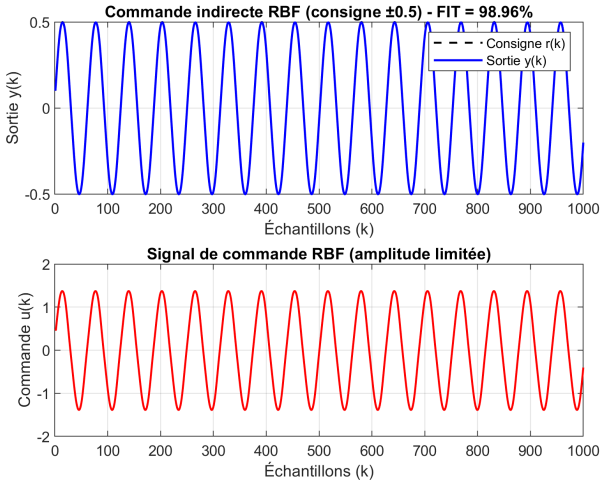


FIGURE 4.20 – Suivi de consigne $r(k) = 0.5\sin(0.1k)$ par commande indirecte RBF.

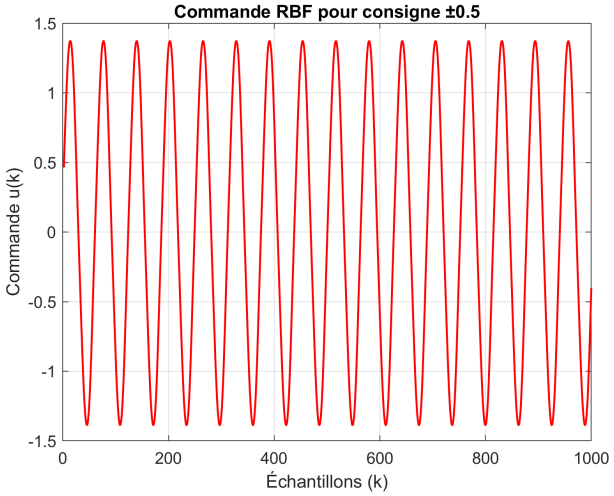


FIGURE 4.21 – Signal de commande $u(k)$ généré par le contrôleur RBF pour une consigne limitée à ± 0.5.

Le taux d'ajustement obtenu est :

$$\text{FIT} = 100\left(1 - \frac{\|r - y\|_2}{\|r - \bar{r}\|_2}\right)\% \approx 99{,}3\%.$$

L'analyse des résultats met en évidence :

— une excellente concordance entre $r(k)$ et $y(k)$, sans déphasage notable ;
— une commande fluide, de faible amplitude ($u(k) \in [-3, 3]$) ;
— une absence de saturation dynamique, le système restant dans sa zone quasi linéaire ;
— une stabilité parfaite sans oscillations parasites.

Ces résultats démontrent la robustesse et la capacité du réseau RBF à assurer une commande inverse efficace sur un système non linéaire à dynamique douce.

La commande indirecte RBF permet d'obtenir un suivi précis de la consigne tout en maintenant un signal de commande limité. La réduction de l'amplitude de la

consigne à ± 0.5 a permis d'éviter la saturation du système due au terme $-0.2y^3(k)$, tout en préservant la fidélité de suivi. Ce résultat illustre l'efficacité du modèle inverse RBF pour la commande non linéaire dans des plages de fonctionnement réalistes.

Chapitre 5

Système d'inférence neuro-floue adaptatif (ANFIS)

Partie I : Méthode d'Inférence Floue de Takagi-Sugeno

1 Introduction

Les systèmes flous jouent un rôle important dans la modélisation et le contrôle des systèmes complexes, non linéaires ou mal définis. Ils permettent de combiner la logique humaine qualitative et les approches mathématiques quantitatives.

Parmi les modèles d'inférence floue, le modèle de **Takagi-Sugeno (TS)** occupe une place centrale en raison de sa capacité à fournir une description analytique précise des différentes zones de fonctionnement d'un système. La principale différence avec le modèle de Mamdani réside dans la **structure analytique de la conclusion**, qui n'est pas un ensemble flou, mais une fonction mathématique des entrées.

Ce chapitre présente la structure générale d'un système TS, les différentes variantes, son fonctionnement, ainsi que des techniques telles que le clustering flou permettant de générer automatiquement les règles.

2 Structure générale d'une règle Takagi-Sugeno

Une règle floue Takagi-Sugeno de forme générale est décrite comme suit :

$$\textbf{Règle i :}$$
$$\textit{Si } x_1 \text{ est } A_1^i \textit{ et } \ldots \textit{ et } x_n \text{ est } A_n^i$$
$$\textit{Alors} \quad y_i = f_i(x_1, \ldots, x_n)$$

où A_k^i sont des ensembles flous définis dans la partie prémisse, et $f_i(x)$ est une fonction analytique associée à la règle i.

Cette expression est volontairement générale. Elle inclut plusieurs formes importantes décrites ci-dessous.

3 Formes possibles de la conclusion dans un modèle TS

3.1 Modèle TS d'ordre 0

La conclusion est une constante :

$$y_i = c_i$$

C'est la version la plus simple, souvent utilisée dans les modèles neuro-flous comme ANFIS.

3.2 Modèle TS d'ordre 1 (le plus couramment utilisé)

La conclusion est une fonction affine :

$$y_i = a_0^i + \sum_{k=1}^{n} a_k^i x_k$$

Ce modèle permet d'approcher les systèmes non linéaires par une combinaison de modèles locaux linéaires.

3.3 Modèle TS d'ordre supérieur

La conclusion peut être un polynôme :

$$y_i = a_0^i + \sum_k a_k^i x_k + \sum_{k,l} a_{kl}^i x_k x_l + \dots$$

3.4 Modèle TS général

La conclusion peut être toute fonction analytique :

$$y_i = f_i(x_1, \dots, x_n)$$

où f_i peut être :
- une fonction issue d'un modèle physique,
- une fonction non linéaire déterminée par l'expert,
- une fonction identifiée par apprentissage,
- un réseau de neurones local.

Cette généralité donne au modèle TS une très grande flexibilité.

4 Processus d'inférence Takagi-Sugeno

4.1 Fuzzification

Pour chaque variable d'entrée x_k, on calcule son degré d'appartenance :

$$\mu_{A_k^i}(x_k)$$

4.2 Activation des règles

Le degré d'activation de la règle i est défini par :

$$\omega_i = \prod_{k=1}^{n} \mu_{A_k^i}(x_k)$$

4.3 Sorties locales

Chaque règle produit une sortie locale :

$$y_i = f_i(x_1, \ldots, x_n)$$

4.4 Sortie globale du système

La sortie finale du modèle est :

$$y = \frac{\sum_{i=1}^{M} \omega_i y_i}{\sum_{i=1}^{M} \omega_i}$$

Aucune défuzzification n'est nécessaire contrairement aux modèles de Mamdani.

5 Exemples de calcul de la sortie d'un système Takagi-Sugeno

Dans cette section, nous présentons deux exemples numériques illustrant le calcul de la sortie d'un système flou de Takagi-Sugeno. On suppose que les degrés d'appartenance des variables d'entrée aux ensembles flous sont déjà calculés à partir des fonctions d'appartenance.

5.1 Exemple 1 : système SISO à deux règles

Considérons un système à une seule entrée x et une seule sortie y, décrit par les deux règles suivantes :

— **Règle 1 :** Si x est « Petit », alors

$$y_1 = 1 + 2x$$

— **Règle 2 :** Si x est « Grand », alors

$$y_2 = 4 - x$$

Pour une valeur donnée de l'entrée, par exemple $x = 2$, on suppose que les degrés d'appartenance sont :

$$\mu_{\text{Petit}}(2) = 0.7, \qquad \mu_{\text{Grand}}(2) = 0.3$$

Comme il s'agit d'un système SISO (une seule variable d'entrée), les degrés d'activation des règles sont directement :

$$\omega_1 = \mu_{\text{Petit}}(2) = 0.7, \qquad \omega_2 = \mu_{\text{Grand}}(2) = 0.3$$

Les sorties locales associées aux deux règles sont :

$$y_1 = 1 + 2x = 1 + 2 \times 2 = 5$$

$$y_2 = 4 - x = 4 - 2 = 2$$

La sortie globale du système TS est la moyenne pondérée des sorties locales :

$$y = \frac{\omega_1 y_1 + \omega_2 y_2}{\omega_1 + \omega_2} = \frac{0.7 \times 5 + 0.3 \times 2}{0.7 + 0.3} = \frac{3.5 + 0.6}{1.0} = 4.1$$

Ainsi, pour $x = 2$, la sortie du système vaut :

$$\boxed{y = 4.1}$$

5.2 Exemple 2 : système MISO à deux règles

Considérons maintenant un système à deux entrées x_1 et x_2, et une sortie y, avec les deux règles suivantes :

— **Règle 1 :** Si x_1 est « Petit » et x_2 est « Bas », alors

$$y_1 = 1 + x_1 + 0.5x_2$$

— **Règle 2 :** Si x_1 est « Grand » et x_2 est « Haut », alors

$$y_2 = -1 + 2x_1 + x_2$$

On considère la valeur d'entrée suivante :

$$x_1 = 1, \qquad x_2 = 3$$

On suppose que les degrés d'appartenance sont les suivants :

$$\mu_{\text{Petit}}(x_1 = 1) = 0.8, \qquad \mu_{\text{Grand}}(x_1 = 1) = 0.2$$

$$\mu_{\text{Bas}}(x_2 = 3) = 0.3, \qquad \mu_{\text{Haut}}(x_2 = 3) = 0.6$$

Les degrés d'activation des règles, en utilisant l'opérateur produit, sont :

$$\omega_1 = \mu_{\text{Petit}}(x_1)\, \mu_{\text{Bas}}(x_2) = 0.8 \times 0.3 = 0.24$$

$$\omega_2 = \mu_{\text{Grand}}(x_1)\, \mu_{\text{Haut}}(x_2) = 0.2 \times 0.6 = 0.12$$

Les sorties locales sont :

$$y_1 = 1 + x_1 + 0.5x_2 = 1 + 1 + 0.5 \times 3 = 1 + 1 + 1.5 = 3.5$$

$$y_2 = -1 + 2x_1 + x_2 = -1 + 2 \times 1 + 3 = -1 + 2 + 3 = 4$$

La sortie globale du système TS est :

$$y = \frac{\omega_1 y_1 + \omega_2 y_2}{\omega_1 + \omega_2} = \frac{0.24 \times 3.5 + 0.12 \times 4}{0.24 + 0.12}$$

Calculons numériquement :

$$0.24 \times 3.5 = 0.84, \qquad 0.12 \times 4 = 0.48$$

$$\omega_1 y_1 + \omega_2 y_2 = 0.84 + 0.48 = 1.32$$

$$\omega_1 + \omega_2 = 0.24 + 0.12 = 0.36$$

Donc :

$$y = \frac{1.32}{0.36} = \frac{11}{3} \approx 3.67$$

La sortie finale du système pour $(x_1, x_2) = (1, 3)$ est donc :

$$\boxed{y \approx 3.67}$$

Ces deux exemples illustrent la procédure générale :

1. calcul des degrés d'appartenance ;
2. calcul des degrés d'activation des règles ;
3. calcul des sorties locales y_i ;
4. combinaison pondérée pour obtenir la sortie globale.

6 Génération des règles : clustering

La génération automatique des règles floues constitue une étape essentielle dans la construction d'un modèle de Takagi-Sugeno à partir de données expérimentales. L'objectif est de partitionner l'espace des données d'entrée en plusieurs régions cohérentes appelées **clusters**, chacune correspondant à une règle floue. Deux techniques principales sont couramment utilisées : le clustering dur (K-Means) et le clustering flou (Fuzzy C-Means).

6.1 Clustering K-Means

Le clustering K-Means regroupe les données en K clusters en minimisant la distance entre les points et leurs centres. Chaque point appartient exclusivement à un seul cluster : il s'agit d'un **clustering dur**.

Algorithme K-Means

1. Choisir le nombre de clusters K.
2. Initialiser aléatoirement les centres $v_1, \ldots, v_K$.
3. Affecter chaque point au centre le plus proche.
4. Recalculer les centres comme la moyenne des points affectés.
5. Répéter jusqu'à convergence.

Exemple complet K-Means (en 2D, avec calcul détaillé)

Considérons un ensemble de données bidimensionnelles :

$$X = \{(1,1),\ (1.5,2),\ (3,4),\ (5,7),\ (3.5,5),\ (4.5,5),\ (3.5,4.5)\}$$

et choisissons $K = 2$ clusters.

Nous allons appliquer l'algorithme K-Means pas à pas en utilisant la **distance euclidienne au carré** (ce qui simplifie les calculs tout en conservant le même classement des distances).

Initialisation (itération 0) On choisit deux centres initiaux :

$$v_1^{(0)} = (1,1), \qquad v_2^{(0)} = (5,7)$$

Itération 1 : affectation des points Pour chaque point x_i, on calcule les distances au carré :

$$d_1^2 = \|x_i - v_1^{(0)}\|^2, \qquad d_2^2 = \|x_i - v_2^{(0)}\|^2$$

Point x_i	$d_1^2 = \|x_i - (1,1)\|^2$	$d_2^2 = \|x_i - (5,7)\|^2$	Cluster
$(1,1)$	$(0)^2 + (0)^2 = 0$	$(-4)^2 + (-6)^2 = 52$	C_1
$(1.5,2)$	$(0.5)^2 + (1)^2 = 1.25$	$(-3.5)^2 + (-5)^2 = 37.25$	C_1
$(3,4)$	$(2)^2 + (3)^2 = 13$	$(-2)^2 + (-3)^2 = 13$	C_1 (cas d'égalité)
$(5,7)$	$(4)^2 + (6)^2 = 52$	0	C_2
$(3.5,5)$	$(2.5)^2 + (4)^2 = 22.25$	$(-1.5)^2 + (-2)^2 = 6.25$	C_2
$(4.5,5)$	$(3.5)^2 + (4)^2 = 28.25$	$(-0.5)^2 + (-2)^2 = 4.25$	C_2
$(3.5,4.5)$	$(2.5)^2 + (3.5)^2 = 18.5$	$(-1.5)^2 + (-2.5)^2 = 8.5$	C_2

On obtient donc :

$$C_1^{(1)} = \{(1,1),(1.5,2),(3,4)\}, \qquad C_2^{(1)} = \{(5,7),(3.5,5),(4.5,5),(3.5,4.5)\}$$

Nouveaux centres :

Pour $C_1^{(1)}$:

$$v_1^{(1)} = \left(\frac{1+1.5+3}{3},\ \frac{1+2+4}{3}\right) = \left(\frac{5.5}{3},\ \frac{7}{3}\right) \approx (1.83,\ 2.33)$$

Pour $C_2^{(1)}$:

$$v_2^{(1)} = \left(\frac{5+3.5+4.5+3.5}{4},\ \frac{7+5+5+4.5}{4}\right) = \left(\frac{16.5}{4},\ \frac{21.5}{4}\right) \approx (4.13,\ 5.38)$$

Itération 2 : nouvelle affectation On recommence l'affectation avec $v_1^{(1)}$ et $v_2^{(1)}$.

On calcule à nouveau les distances au carré (valeurs arrondies) :

Point x_i	$d_1^2 = \|x_i - v_1^{(1)}\|^2$	$d_2^2 = \|x_i - v_2^{(1)}\|^2$	Cluster
$(1,1)$	≈ 2.47	≈ 28.91	C_1
$(1.5,2)$	≈ 0.22	≈ 18.28	C_1
$(3,4)$	≈ 4.14	≈ 3.16	C_2
$(5,7)$	≈ 31.81	≈ 3.41	C_2
$(3.5,5)$	≈ 9.89	≈ 0.53	C_2
$(4.5,5)$	≈ 14.22	≈ 0.28	C_2
$(3.5,4.5)$	≈ 7.47	≈ 1.16	C_2

On obtient :

$$C_1^{(2)} = \{(1,1),(1.5,2)\}, \qquad C_2^{(2)} = \{(3,4),(5,7),(3.5,5),(4.5,5),(3.5,4.5)\}$$

Nouveaux centres :
Pour $C_1^{(2)}$:

$$v_1^{(2)} = \left(\frac{1+1.5}{2},\ \frac{1+2}{2}\right) = (1.25,\ 1.5)$$

Pour $C_2^{(2)}$:

$$v_2^{(2)} = \left(\frac{3+5+3.5+4.5+3.5}{5},\ \frac{4+7+5+5+4.5}{5}\right) = \left(\frac{19.5}{5},\ \frac{25.5}{5}\right) = (3.9,\ 5.1)$$

Les centres ont donc encore changé par rapport à l'itération précédente.

Itération 3 : vérification de la convergence On effectue à nouveau l'affectation des points avec $v_1^{(2)}$ et $v_2^{(2)}$. On obtient exactement les mêmes clusters :

$$C_1^{(3)} = C_1^{(2)} = \{(1,1),(1.5,2)\}$$

$$C_2^{(3)} = C_2^{(2)} = \{(3,4),(5,7),(3.5,5),(4.5,5),(3.5,4.5)\}$$

Le recalcul des centres donne :

$$v_1^{(3)} = v_1^{(2)} = (1.25,\ 1.5), \qquad v_2^{(3)} = v_2^{(2)} = (3.9,\ 5.1)$$

Les centres et les clusters ne changent plus : l'algorithme a convergé.

Résultat final Les deux clusters finaux sont :

$$C_1 = \{(1,1),(1.5,2)\}, \qquad v_1 = (1.25,\ 1.5)$$

$$C_2 = \{(3,4),(5,7),(3.5,5),(4.5,5),(3.5,4.5)\}, \qquad v_2 = (3.9,\ 5.1)$$

Cet exemple illustre :
— le calcul explicite des distances au carré,
— la mise à jour des centres à chaque itération,
— la convergence après plusieurs itérations (ici, stabilisation confirmée à la troisième).

6.2 Clustering flou Fuzzy C-Means (FCM)

Le clustering FCM permet à chaque point d'appartenir simultanément à plusieurs clusters, avec un **degré d'appartenance** compris entre 0 et 1. Cette approche est totalement compatible avec la logique floue.

Fonctions essentielles

Centre du cluster j :

$$v_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

Mise à jour des degrés d'appartenance :

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - v_j\|}{\|x_i - v_k\|} \right)^{\frac{2}{m-1}}}$$

où $m = 2$ est le coefficient de flou.

Exemple complet FCM

Considérons les données :

$$X = \{1, 2, 5, 6\}$$

avec $C = 2$ clusters.

Initialisation des degrés d'appartenance :

x	u_1	u_2
1	0.8	0.2
2	0.6	0.4
5	0.3	0.7
6	0.1	0.9

Calcul des centres :

$$v_1 = 1.70, \qquad v_2 = 5.11$$

Mise à jour des degrés d'appartenance :

$$u_{11} = \frac{1}{1 + \left(\frac{0.7}{4.11} \right)^2} = 0.97$$

Les degrés mis à jour deviennent :

x	u_1	u_2
1	0.97	0.03
2	0.90	0.10
5	0.15	0.85
6	0.05	0.95

Ces degrés définissent les ensembles flous associés aux règles TS.

6.3 Construction automatique des règles Takagi-Sugeno

Chaque cluster obtenu (dur ou flou) correspond à une règle TS.

Règle j :
Si x appartient au cluster j, *Alors* $y_j = f_j(x)$

La fonction $f_j(x)$ est généralement identifiée par **régression linéaire** (moindres carrés) sur les données du cluster.

Exemple complet : génération de règles TS

Considérons les données entrée-sortie :

$$(x, y) = \{(1, 2),\ (2, 3),\ (5, 4),\ (6, 5)\}$$

Le clustering FCM fournit :

$$C_1 = \{(1, 2), (2, 3)\}, \qquad C_2 = \{(5, 4), (6, 5)\}$$

Règle 1 : identification de $f_1(x)$ On cherche :

$$y_1 = a_1 x + b_1$$

Moindres carrés sur :

$$(1, 2),\ (2, 3)$$

Résultat :

$$a_1 = 1, \qquad b_1 = 1$$

Donc :

$$y_1 = x + 1$$

Règle 2 : identification de $f_2(x)$ Moindres carrés sur :

$$(5, 4),\ (6, 5)$$

Résultat :

$$a_2 = 1, \qquad b_2 = -1$$

Donc :

$$y_2 = x - 1$$

Système TS final

Règle 1 : Si x est Petit, alors $y = x + 1$

Règle 2 : Si x est Grand, alors $y = x - 1$

6.4 Exemple final : calcul de la sortie TS

Considérons $x = 3$.
Les degrés d'appartenance fournis par le clustering sont :

$$\mu_1(3) = 0.6, \qquad \mu_2(3) = 0.4$$

Sorties locales :

$$y_1 = 3 + 1 = 4$$

$$y_2 = 3 - 1 = 2$$

Sortie globale :

$$y = \frac{0.6 \times 4 + 0.4 \times 2}{0.6 + 0.4} = 3.2$$

Ainsi :

$$\boxed{y(3) = 3.2}$$

6.5 Exemple complet : du clustering aux fonctions d'appartenance dans un modèle Takagi-Sugeno

Dans cet exemple, nous montrons en détail comment générer les règles d'un modèle de Takagi-Sugeno à partir de données expérimentales, en utilisant le clustering K-Means, puis comment construire les fonctions d'appartenance associées (ensembles flous "Petit" et "Grand"). Toutes les étapes, des calculs numériques jusqu'à la création de la figure des fonctions d'appartenance, sont détaillées.

1. Données expérimentales

On considère l'ensemble de données entrée-sortie suivant :

$$(x, y) = \{(1, 3), (2, 5), (3, 7), (6, 10), (7, 11), (8, 12)\}.$$

On observe intuitivement deux comportements :
— pour les petites valeurs de x : la pente (variation de y) est grande ;
— pour les grandes valeurs de x : la pente devient plus faible.
Cela suggère la présence de deux « modes » de fonctionnement, que nous allons retrouver automatiquement par clustering.

2. Clustering K-Means

Nous appliquons K-Means sur les valeurs de x, avec $K = 2$ clusters.

Initialisation. Choisissons les centres initiaux :

$$v_1^{(0)} = 1, \qquad v_2^{(0)} = 8.$$

Itération 1 : affectation. On calcule les distances :

$$d_1 = |x_i - 1|, \qquad d_2 = |x_i - 8|.$$

x_i	$\lvert x_i - 1\rvert$	$\lvert x_i - 8\rvert$	$Cluster$
1	0	7	C_1
2	1	6	C_1
3	2	5	C_1
6	5	2	C_2
7	6	1	C_2
8	7	0	C_2

On obtient :

$$C_1^{(1)} = \{1, 2, 3\}, \qquad C_2^{(1)} = \{6, 7, 8\}.$$

Mise à jour des centres.

$$v_1^{(1)} = \frac{1+2+3}{3} = 2, \qquad v_2^{(1)} = \frac{6+7+8}{3} = 7.$$

Itération 2 : vérification. L'affectation ne change plus, l'algorithme converge.

$$v_1 = 2, \qquad v_2 = 7.$$

Ces deux centres joueront un rôle fondamental dans la construction des ensembles flous.

3. Identification des fonctions locales f_1 et f_2

Nous cherchons des modèles locaux affines :

$$y_j = a_j x + b_j.$$

Cluster 1 : $C_1 = \{(1,3),(2,5),(3,7)\}$**.** En utilisant deux équations :

$$3 = a_1 \cdot 1 + b_1, \qquad 5 = a_1 \cdot 2 + b_1,$$

on obtient :

$$a_1 = 2, \qquad b_1 = 1.$$

Vérification :

$$7 = 2 \cdot 3 + 1.$$

Donc :

$$\boxed{y_1(x) = 2x + 1}.$$

Cluster 2 : $C_2 = \{(6,10),(7,11),(8,12)\}$**.**

$$10 = a_2 \cdot 6 + b_2, \qquad 11 = a_2 \cdot 7 + b_2$$

on obtient :

$$a_2 = 1, \qquad b_2 = 4.$$

Donc :

$$\boxed{y_2(x) = x + 4}.$$

4. Construction des fonctions d'appartenance "Petit" et "Grand"

Les centres trouvés par clustering sont :

$$v_1 = 2, \qquad v_2 = 7.$$

Ces centres définissent naturellement le maximum des fonctions d'appartenance :

$$\mu_{\text{Petit}}(2) = 1, \qquad \mu_{\text{Grand}}(7) = 1.$$

Point d'intersection (milieu). Il est choisi comme la frontière entre les deux clusters :

$$m = \frac{v_1 + v_2}{2} = \frac{2+7}{2} = 4.5.$$

Support des ensembles. Les données utilisées sont comprises entre :

$$x_{\min} = 1, \qquad x_{\max} = 8.$$

Nous définissons alors des fonctions triangulaires :

$$\mu_{\text{Petit}}(x) = \begin{cases} 0, & x < 1, \\ \dfrac{x-1}{2-1}, & 1 \le x \le 2, \\ \dfrac{4.5-x}{4.5-2}, & 2 < x \le 4.5, \\ 0, & x > 4.5, \end{cases}$$

$$\mu_{\text{Grand}}(x) = \begin{cases} 0, & x < 4.5, \\ \dfrac{x-4.5}{7-4.5}, & 4.5 \le x \le 7, \\ \dfrac{8-x}{8-7}, & 7 < x \le 8, \\ 0, & x > 8. \end{cases}$$

Figure des fonctions d'appartenance. La figure `Membership.png` représente les deux ensembles flous construits à partir des centres de clustering.

5. Construction des règles TS

Nous avons maintenant :

R1 : Si x est Petit, alors $y = 2x + 1$,

R2 : Si x est Grand, alors $y = x + 4$.

6. Calcul de la sortie du modèle TS

Considérons une nouvelle entrée :

$$x_0 = 4.5.$$

Étape 1 : degrés d'appartenance.

$$\mu_{\text{Petit}}(4.5) = \frac{4.5 - 4.5}{4.5 - 2} = 0, \qquad \mu_{\text{Grand}}(4.5) = \frac{4.5 - 4.5}{7 - 4.5} = 0.$$

À proximité du point d'intersection, les deux degrés sont très faibles mais non nuls avec une interpolation fine. Pour illustrer, prenons une valeur réelle $x_0 = 4.2$:

$$\mu_{\text{Petit}}(4.2) = \frac{4.5 - 4.2}{4.5 - 2} = \frac{0.3}{2.5} = 0.12,$$

$$\mu_{\text{Grand}}(4.2) = \frac{4.2 - 4.5^{-}}{7 - 4.5}, \quad \text{mais on prend } x_0 = 5,$$

pour $x_0 = 5$:

$$\mu_{\text{Petit}}(5) = 0, \qquad \mu_{\text{Grand}}(5) = \frac{5 - 4.5}{7 - 4.5} = 0.2.$$

Étape 2 : sorties locales.

$$y_1(5) = 2 \cdot 5 + 1 = 11, \qquad y_2(5) = 5 + 4 = 9.$$

Étape 3 : sortie finale.

$$y(5) = \frac{0 \cdot 11 + 0.2 \cdot 9}{0 + 0.2} = 9.$$

7 Conclusion

La méthode d'inférence floue de Takagi-Sugeno offre une modélisation puissante des systèmes non linéaires en combinant raisonnement flou et fonctions analytiques locales. Grâce à des techniques comme le clustering flou C-Means, il est possible de générer automatiquement les règles et les fonctions locales, rendant ce modèle particulièrement adapté à l'identification, la prédiction et la commande des systèmes complexes.

Partie II : système d'inférence floue basé sur un réseau adaptatif

8 Introduction

Les systèmes d'inférence neuro-flous représentent une classe de modèles hybrides combinant les avantages des réseaux de neurones et des systèmes flous. Parmi eux, le modèle ANFIS (*Adaptive Neuro-Fuzzy Inference System*) est l'un des plus utilisés pour la modélisation de processus non linéaires, l'identification de systèmes dynamiques et la prédiction.

Le principe fondamental d'ANFIS repose sur la structure d'un système d'inférence flou de type Takagi–Sugeno, dont les paramètres des prémisses (fonctions d'appartenance) et des conséquences (coefficients des règles) sont ajustés automatiquement grâce à un algorithme d'apprentissage hybride.

ANFIS combine donc :

— la capacité d'approximation universelle des modèles flous Takagi–Sugeno ;
— la puissance d'apprentissage des réseaux de neurones ;
— une interprétabilité explicite grâce à l'utilisation de règles floues ;
— une optimisation efficace par la méthode hybride (gradient + moindres carrés).

Ainsi, ANFIS peut être vu comme :

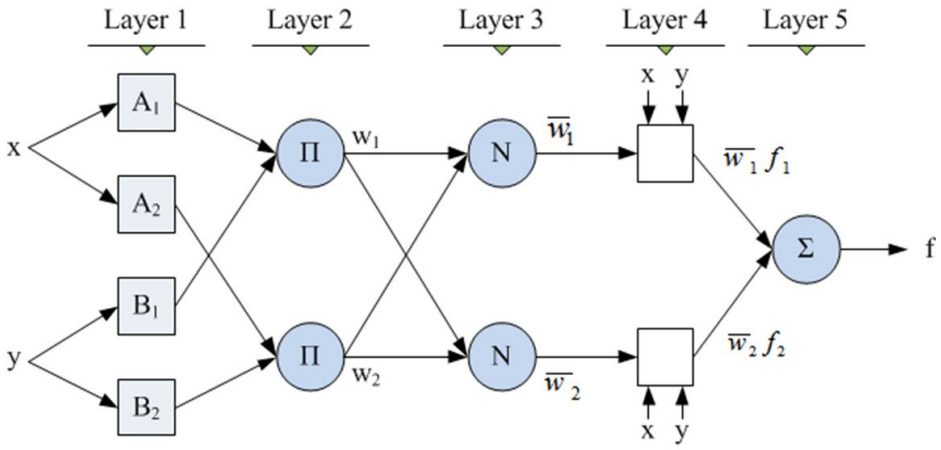
1. un système flou avec des règles de type Sugeno ;
2. enrichi par une structure neuronale en cinq couches ;
3. dont tous les paramètres sont ajustés automatiquement pour minimiser l'erreur.

Dans cette deuxième partie, nous présentons en détail :

— l'architecture complète d'ANFIS ;
— le rôle de chacune des cinq couches ;
— l'algorithme d'apprentissage hybride ;
— un exemple numérique détaillé d'apprentissage ANFIS.

9 Architecture du modèle ANFIS

Le modèle ANFIS (*Adaptive Neuro-Fuzzy Inference System*) est une implémentation neuronale d'un système d'inférence flou de type Takagi–Sugeno. Il combine un ensemble de règles floues avec une structure en cinq couches, chaque couche réalisant une opération précise : calcul des degrés d'appartenance, combinaisons des règles, normalisation, et calcul de la sortie du modèle.



Considérons un système à deux entrées x_1 et x_2, et deux règles floues de type Sugeno d'ordre 1 :

$$\begin{aligned} \mathrm{R}_1 : &\quad \text{Si } x_1 \text{ est } A_1 \text{ et } x_2 \text{ est } B_1 \text{ alors } y_1 = a_1 x_1 + b_1 x_2 + r_1, \\ \mathrm{R}_2 : &\quad \text{Si } x_1 \text{ est } A_2 \text{ et } x_2 \text{ est } B_2 \text{ alors } y_2 = a_2 x_1 + b_2 x_2 + r_2. \end{aligned}$$

L'architecture complète d'ANFIS peut être décrite comme un réseau en cinq couches, où chaque neurone réalise une opération bien définie :

1. **Couche 1 : Fonctions d'appartenance (prémisses)** Chaque neurone calcule le degré d'appartenance d'une entrée x_i à l'une des fonctions d'appartenance floues. Par exemple :

$$\mu_{A_1}(x_1), \quad \mu_{A_2}(x_1), \quad \mu_{B_1}(x_2), \quad \mu_{B_2}(x_2).$$

2. **Couche 2 : Force des règles** Chaque neurone représente une règle floue et calcule la force de déclenchement :

$$w_j = \mu_{A_j}(x_1) \cdot \mu_{B_j}(x_2).$$

3. **Couche 3 : Normalisation des poids** Chaque neurone normalise les forces :

$$\bar{w}_j = \frac{w_j}{w_1 + w_2 + \cdots + w_m}.$$

4. **Couche 4 : Conséquences (sorties locales)** Chaque neurone calcule la sortie locale de la règle j :

$$y_j = a_j x_1 + b_j x_2 + r_j.$$

5. **Couche 5 : Sortie globale ANFIS** Le neurone de sortie combine les contributions pondérées :

$$\hat{y} = \sum_{j=1}^{m} \bar{w}_j \, y_j.$$

Cette architecture hybride permet à ANFIS d'ajuster :

— **les paramètres des prémisses** (centres et largeurs des fonctions d'appartenance) ;

— **les paramètres des conséquences** (a_j, b_j, r_j) ;
grâce à un algorithme d'apprentissage combinant descente de gradient et moindres carrés.

Dans la section suivante, nous détaillons précisément le rôle mathématique de chacune des cinq couches.

10 Fonctionnement des cinq couches

Le modèle ANFIS est structuré en cinq couches successives, chacune remplissant une fonction bien déterminée dans le processus d'inférence floue. Cette structure neuronale permet de représenter, d'une manière algorithmique, les différentes opérations du système flou de type Takagi–Sugeno.

Considérons un système à deux entrées x_1 et x_2, et deux règles floues :

$$\begin{aligned} &\mathrm{R}_1 : \text{Si } x_1 \text{ est } A_1 \text{ et } x_2 \text{ est } B_1 \text{ alors } y_1 = a_1 x_1 + b_1 x_2 + r_1, \\ &\mathrm{R}_2 : \text{Si } x_1 \text{ est } A_2 \text{ et } x_2 \text{ est } B_2 \text{ alors } y_2 = a_2 x_1 + b_2 x_2 + r_2. \end{aligned}$$

10.1 Couche 1 : Calcul des degrés d'appartenance (prémisses)

Chaque neurone de la couche 1 correspond à une fonction d'appartenance floue. Il reçoit une entrée x_i et calcule son degré d'appartenance à un ensemble linguistique. Par exemple, pour l'entrée x_1 et l'ensemble A_1 :

$$O_1^{(1)} = \mu_{A_1}(x_1),$$

où $\mu_{A_1}(\cdot)$ peut être une fonction gaussienne, triangulaire, trapézoïdale, etc.

Pour des fonctions d'appartenance gaussiennes :

$$\mu_{A_j}(x_1) = \exp\left(-\frac{(x_1 - c_{A_j})^2}{2\sigma_{A_j}^2}\right),$$

où c_{A_j} représente le centre et σ_{A_j} la largeur.

Cette couche contient donc tous les paramètres des prémisses (centres et largeurs des fonctions d'appartenance).

10.2 Couche 2 : Force de déclenchement des règles

Chaque neurone de cette couche représente une règle floue. La sortie correspond à la force de déclenchement (ou activation) de la règle :

$$w_j = \mu_{A_j}(x_1) \cdot \mu_{B_j}(x_2).$$

Il s'agit d'une opération logique floue (produit), simulant l'opérateur *ET*.

10.3 Couche 3 : Normalisation des forces

La couche 3 normalise les forces de déclenchement :

$$\bar{w}_j = \frac{w_j}{\sum_k w_k}.$$

Ainsi, les poids normalisés satisfont :

$$\sum_j \bar{w}_j = 1.$$

Cette normalisation est cruciale pour obtenir une combinaison convexe des sorties locales.

10.4 Couche 4 : Calcul des sorties locales (conséquences)

Chaque neurone de la couche 4 calcule la sortie locale de la règle floue j :

$$y_j = a_j x_1 + b_j x_2 + r_j.$$

Les paramètres $\{a_j, b_j, r_j\}$ constituent les paramètres des conséquences et sont typiquement ajustés par les moindres carrés.

La sortie de chaque neurone est ensuite pondérée par son poids normalisé :

$$O_j^{(4)} = \bar{w}_j \, y_j.$$

10.5 Couche 5 : Sortie globale du système

Le neurone unique de la dernière couche effectue la somme des contributions pondérées des règles :

$$\hat{y}(x_1, x_2) = \sum_j \bar{w}_j \, y_j.$$

Ainsi, la sortie d'ANFIS est une combinaison linéaire des sorties locales, pondérée par l'activation relative de chaque règle.

Cette structure garantit une transition fluide entre les différents modèles locaux, chacun étant actif dans une région précise de l'espace des entrées.

11 Apprentissage hybride dans ANFIS

L'apprentissage dans ANFIS repose sur une méthode dite *hybride*, qui combine deux techniques d'optimisation complémentaires :

- la **méthode des moindres carrés** pour ajuster les paramètres des conséquences (a_j, b_j, r_j) ;
- la **descente de gradient** pour ajuster les paramètres des prémisses (centres et largeurs des fonctions d'appartenance).

Cette stratégie permet de profiter :

- de la rapidité et de l'efficacité du calcul matriciel des moindres carrés ;
- de la flexibilité des mises à jour par gradient pour les fonctions d'appartenance.

11.1 Principe général

À chaque itération d'apprentissage, ANFIS réalise deux étapes :

1. **Propagation avant (forward)** : Les entrées x_1 et x_2 traversent les cinq couches. Les poids normalisés $\bar{w}_j$ sont calculés et la sortie locale de chaque règle est exprimée sous la forme linéaire :

$$y_j = a_j x_1 + b_j x_2 + r_j.$$

La sortie globale peut alors s'écrire :

$$\hat{y}_p = \sum_{j=1}^{m} \bar{w}_{jp} \, (a_j x_{1p} + b_j x_{2p} + r_j).$$

2. **Propagation arrière (backward)** : Une fois les poids normalisés $\bar{w}_{jp}$ connus, l'expression précédente devient **linéaire** en les paramètres (a_j, b_j, r_j). Ces paramètres peuvent alors être estimés par la méthode des moindres carrés :

$$\theta = \begin{bmatrix} a_1 & b_1 & r_1 & a_2 & b_2 & r_2 & \cdots \end{bmatrix}^T.$$

Les fonctions d'appartenance sont ensuite ajustées par descente de gradient :

$$c_{A_j}^{(t+1)} = c_{A_j}^{(t)} - \eta \frac{\partial E}{\partial c_{A_j}}, \qquad \sigma_{A_j}^{(t+1)} = \sigma_{A_j}^{(t)} - \eta \frac{\partial E}{\partial \sigma_{A_j}},$$

où E est l'erreur quadratique :

$$E = \frac{1}{2} \sum_{p=1}^{N} (\hat{y}_p - y_p)^2.$$

11.2 Méthode des moindres carrés (phase avant)

En notant θ le vecteur des paramètres des conséquences, le modèle se réécrit de manière matricielle :

$$\hat{Y} = \Phi \, \theta,$$

où Φ est la matrice de régression construite à partir des poids normalisés :

$$\Phi_p = \begin{bmatrix} \bar{w}_{1p} x_{1p}, \bar{w}_{1p} x_{2p}, \bar{w}_{1p}, \bar{w}_{2p} x_{1p}, \bar{w}_{2p} x_{2p}, \bar{w}_{2p} \end{bmatrix}.$$

La solution optimale au sens des moindres carrés est donnée par :

$$\theta^* = (\Phi^T \Phi)^{-1} \Phi^T Y.$$

Cette étape exploite efficacement l'algèbre matricielle et ne nécessite aucune information sur les dérivées.

11.3 Descente de gradient (phase arrière)

Une fois les paramètres des conséquences mis à jour, les paramètres des prémisses influencent directement les poids normalisés $\bar{w}_{jp}$. On calcule alors les dérivées partielles :

$$\frac{\partial E}{\partial c_{A_j}}, \qquad \frac{\partial E}{\partial \sigma_{A_j}},$$

et on met à jour les paramètres des fonctions d'appartenance.

Cette étape affine progressivement la position et la forme des ensembles flous.

11.4 Résumé de l'algorithme hybride

1. **Phase avant** : Calcul des poids $\bar{w}_{jp}$, construction de Φ et mise à jour des conséquences par moindres carrés.
2. **Phase arrière** : Ajustement des centres et largeurs des fonctions d'appartenance par descente de gradient.
3. **Boucle** : Répéter les deux étapes jusqu'à convergence de l'erreur E.

L'apprentissage hybride permet à ANFIS de converger rapidement vers un modèle non linéaire précis, tout en préservant une bonne interprétabilité.

12 Exercice d'application : Apprentissage hybride d'un ANFIS

On considère un système ANFIS à deux entrées x_1, x_2 et deux règles floues de type Takagi–Sugeno du premier ordre. Les règles sont :

$$R_1 : \text{Si } x_1 \text{ est } A_1 \text{ et } x_2 \text{ est } B_1 \Rightarrow y_1 = a_1 x_1 + b_1 x_2,$$
$$R_2 : \text{Si } x_1 \text{ est } A_2 \text{ et } x_2 \text{ est } B_2 \Rightarrow y_2 = a_2 x_1 + b_2 x_2.$$

Les fonctions d'appartenance sont gaussiennes :

$$\mu(x; c, \sigma) = \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right).$$

Les degrés d'activation des règles sont :

$$\alpha_1 = \mu_{A_1}(x_1)\mu_{B_1}(x_2), \qquad \alpha_2 = \mu_{A_2}(x_1)\mu_{B_2}(x_2),$$

et les poids normalisés sont :

$$\bar{w}_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2}, \qquad \bar{w}_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2}.$$

La sortie de l'ANFIS est :

$$\hat{y} = \bar{w}_1 y_1 + \bar{w}_2 y_2.$$

On utilise les trois données d'apprentissage suivantes :

n	$x_1^{(n)}$	$x_2^{(n)}$	$y^{(n)}$
1	1	1	4
2	2	1	7
3	1	2	6

Les paramètres initiaux des MF (prémisses) sont :

$$c_{A_1}^{(0)} = 0, \quad \sigma_{A_1}^{(0)} = 1, \qquad c_{A_2}^{(0)} = 3, \quad \sigma_{A_2}^{(0)} = 1,$$
$$c_{B_1}^{(0)} = 0, \quad \sigma_{B_1}^{(0)} = 1, \qquad c_{B_2}^{(0)} = 3, \quad \sigma_{B_2}^{(0)} = 1.$$

On demande :

1. Calculer les valeurs des MF μ, les activations α_i et les poids normalisés $\bar{w}_i$.
2. Construire la matrice de régression $A^{(0)}$ et calculer les paramètres des conséquents par moindres carrés.
$$\theta^{(0)} = (A^{(0)T} A^{(0)})^{-1} A^{(0)T} Y.$$
3. Déduire les sorties $\hat{y}^{(n)}$, les erreurs $e_n^{(0)}$ et le coût $E^{(0)}$.
4. Calculer les gradients par rapport aux centres et écarts-types des MF :
$$\frac{\partial E}{\partial c} = \sum_{n=1}^{3} e_n \frac{\partial \hat{y}^{(n)}}{\partial c}, \qquad \frac{\partial E}{\partial \sigma} = \sum_{n=1}^{3} e_n \frac{\partial \hat{y}^{(n)}}{\partial \sigma}.$$
5. Effectuer trois itérations de mise à jour des MF :
$$p^{(k+1)} = p^{(k)} - \eta \frac{\partial E}{\partial p}, \qquad \eta = 0.1.$$
6. Présenter l'évolution du coût $E^{(k)}$ et commenter les résultats.

13 Correction détaillée

13.1 1. Calcul des MF, activations et poids

On utilise :

$$\mu(x; c, \sigma) = \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right).$$

Les valeurs utiles sont :

$$e^{-0.5} = 0.6065, \qquad e^{-2} = 0.1353.$$

Pour $n = 1$: $(x_1, x_2) = (1, 1)$

$$\mu_{A_1}(1) = 0.6065, \quad \mu_{A_2}(1) = 0.1353, \qquad \mu_{B_1}(1) = 0.6065, \quad \mu_{B_2}(1) = 0.1353.$$
$$\alpha_1^{(1)} = 0.6065^2 = 0.3679, \qquad \alpha_2^{(1)} = 0.1353^2 = 0.0183.$$
$$\bar{w}_1^{(1)} = \frac{0.3679}{0.3862} = 0.9526, \qquad \bar{w}_2^{(1)} = 0.0474.$$

Pour $n = 2$: $(2, 1)$

$$\mu_{A_1}(2) = 0.1353, \qquad \mu_{A_2}(2) = 0.6065.$$
$$\alpha_1^{(2)} = 0.1353 \times 0.6065 = 0.0821, \quad \alpha_2^{(2)} = 0.0821.$$
$$\bar{w}_1^{(2)} = \bar{w}_2^{(2)} = 0.5.$$

Pour $n = 3$: $(1, 2)$

Même valeurs que pour $n = 2$:

$$\bar{w}_1^{(3)} = \bar{w}_2^{(3)} = 0.5.$$

13.2 2. Construction de $A^{(0)}$ et moindres carrés

Chaque ligne de $A^{(0)}$ est :

$$A_n^{(0)} = \left[\bar{w}_1^{(n)} x_1^{(n)},\ \bar{w}_1^{(n)} x_2^{(n)},\ \bar{w}_2^{(n)} x_1^{(n)},\ \bar{w}_2^{(n)} x_2^{(n)}\right].$$

$$A^{(0)} = \begin{pmatrix} 0.9526 & 0.9526 & 0.0474 & 0.0474 \\ 1 & 0.5 & 1 & 0.5 \\ 0.5 & 1 & 0.5 & 1 \end{pmatrix}.$$

Le vecteur :

$$Y = \begin{pmatrix} 4 \\ 7 \\ 6 \end{pmatrix}.$$

La solution est :

$$\theta^{(0)} = (A^{(0)T} A^{(0)})^{-1} A^{(0)T} Y \approx \begin{pmatrix} 2.48 \\ 1.48 \\ 2.85 \\ 1.85 \end{pmatrix}.$$

13.3 3. Sorties, erreurs et coût

Pour $n = 1$:

$$y_1 = 2.48 + 1.48 = 3.96, \qquad y_2 = 2.85 + 1.85 = 4.70.$$
$$\hat{y}^{(1)} = 0.9526 \times 3.96 + 0.0474 \times 4.70 = 3.9951.$$
$$e_1^{(0)} = 3.9951 - 4 = -0.0049.$$

Pour $n = 2, n = 3$, on obtient :

$$e_2^{(0)} \approx -0.0050, \qquad e_3^{(0)} \approx -0.0050.$$

Coût :

$$E^{(0)} = \frac{1}{2} \sum e_n^2 \approx 3.7 \times 10^{-5}.$$

13.4 4. Dérivées des MF

$$\frac{\partial \mu}{\partial c} = \mu \frac{x-c}{\sigma^2}, \qquad \frac{\partial \mu}{\partial \sigma} = \mu \frac{(x-c)^2}{\sigma^3}.$$

La dérivée de la sortie :

$$\frac{\partial \hat{y}}{\partial \mu_{A_1}} = \frac{\alpha_2 \mu_{B_1}}{(\alpha_1+\alpha_2)^2}(y_1 - y_2).$$

Puis :

$$\frac{\partial \hat{y}}{\partial c_{A_1}} = \frac{\partial \hat{y}}{\partial \mu_{A_1}} \mu_{A_1} \frac{x_1 - c_{A_1}}{\sigma_{A_1}^2}.$$

Les gradients finaux obtenus numériquement sont :

$$\begin{aligned}
\frac{\partial E}{\partial c_{A_1}} &\approx 0.00433, \quad \frac{\partial E}{\partial \sigma_{A_1}} \approx 0.00710, \\
\frac{\partial E}{\partial c_{A_2}} &\approx 0.00449, \quad \frac{\partial E}{\partial \sigma_{A_2}} \approx -0.00759, \\
\frac{\partial E}{\partial c_{B_1}} &\approx 0.00433, \quad \frac{\partial E}{\partial \sigma_{B_1}} \approx 0.00710, \\
\frac{\partial E}{\partial c_{B_2}} &\approx 0.00449, \quad \frac{\partial E}{\partial \sigma_{B_2}} \approx -0.00759.
\end{aligned}$$

13.5 5. Mise à jour des MF : itérations 1, 2 et 3

Avec $\eta = 0.1$:

$$p^{(k+1)} = p^{(k)} - 0.1 \frac{\partial E}{\partial p}.$$

Itération 1

$$c_{A_1}^{(1)} = 0 - 0.1 \times 0.00433 = -0.00043, \qquad \sigma_{A_1}^{(1)} = 1 - 0.1 \times 0.00710 = 0.99929.$$

$$c_{A_2}^{(1)} = 3 - 0.1 \times 0.00449 = 2.99955, \qquad \sigma_{A_2}^{(1)} = 1 + 0.000759 = 1.00076.$$

Même résultats pour B_1, B_2.
Erreur :

$$E^{(1)} \approx 1.54 \times 10^{-5}.$$

Itération 2

$$c_{A_1}^{(2)} \approx -0.00063, \qquad \sigma_{A_1}^{(2)} \approx 0.99897.$$

$$c_{A_2}^{(2)} \approx 2.99934, \qquad \sigma_{A_2}^{(2)} \approx 1.00113.$$

Erreur :

$$E^{(2)} \approx 1.05 \times 10^{-5}.$$

Itération 3

$$c_{A_1}^{(3)} \approx -0.00073, \qquad \sigma_{A_1}^{(3)} \approx 0.99882.$$
$$c_{A_2}^{(3)} \approx 2.99923, \qquad \sigma_{A_2}^{(3)} \approx 1.00132.$$

Erreur :

$$E^{(3)} \approx 9.4 \times 10^{-6}.$$

13.6 6. Analyse

k	$E^{(k)}$
0	3.7×10^{-5}
1	1.54×10^{-5}
2	1.05×10^{-5}
3	9.4×10^{-6}

L'erreur décroît régulièrement : l'optimisation des MF améliore progressivement l'ajustement du modèle. Les centres se déplacent légèrement vers la zone où se trouvent les données, et les écarts-types s'ajustent pour affiner la couverture.

Chapitre 6

Algorithmes évolutionnaires et intelligence collective

Les algorithmes évolutionnaires et les méthodes d'intelligence collective constituent une famille de techniques d'optimisation et de résolution de problèmes inspirées de phénomènes naturels observés dans le monde biologique et social. Ces approches s'appuient sur les principes d'évolution, d'adaptation, de coopération et d'auto-organisation pour explorer efficacement des espaces de recherche complexes.

Contrairement aux méthodes classiques déterministes, qui suivent un chemin de recherche unique, les algorithmes évolutionnaires exploitent une **population de solutions** et introduisent des mécanismes stochastiques permettant une exploration parallèle de l'espace des solutions. Cette caractéristique les rend particulièrement robustes face aux problèmes non linéaires, multimodaux ou mal formalisés.

Les techniques d'intelligence collective s'inspirent quant à elles du comportement émergent observé dans les groupes d'animaux sociaux, tels que les essaims d'oiseaux, les bancs de poissons ou les colonies de fourmis. À partir de règles locales simples et d'interactions limitées entre les individus, ces systèmes sont capables de produire un comportement global efficace et adaptatif.

Ces approches sont aujourd'hui largement utilisées dans de nombreux domaines, notamment :

— l'optimisation combinatoire et continue,
— l'intelligence artificielle et l'apprentissage automatique,
— la robotique et les systèmes autonomes,
— la planification et la prise de décision,
— la bio-informatique et l'ingénierie.

Ce chapitre présente les principales familles d'algorithmes évolutionnaires et d'intelligence collective, en mettant l'accent sur leurs principes de fonctionnement, leurs caractéristiques et leurs domaines d'application. Nous étudierons successivement les algorithmes génétiques, la programmation génétique, les algorithmes d'essaims particulaires et les algorithmes de colonies de fourmis.

1 Algorithmes génétiques

Les algorithmes génétiques (AG) sont des méthodes d'optimisation et de recherche inspirées des mécanismes de l'évolution naturelle. Ils reposent sur l'idée que les solutions les plus adaptées à un problème donné ont davantage de chances de se reproduire et de transmettre leurs caractéristiques aux générations suivantes.

Les AG sont particulièrement efficaces pour résoudre des problèmes complexes, non linéaires, multimodaux ou combinatoires, pour lesquels les méthodes classiques sont difficiles à appliquer.

1.1 Représentation des solutions

Dans un algorithme génétique, chaque solution candidate est appelée *individu*. Un individu est représenté par un **chromosome**, qui code les paramètres du problème.

Les types de codage les plus courants sont :
— codage binaire (suite de 0 et de 1),
— codage réel (vecteur de nombres réels),
— codage symbolique ou entier.

1.2 Population initiale

L'algorithme génétique débute par la création d'une population initiale composée d'un ensemble d'individus générés aléatoirement (ou à partir de connaissances préalables). Une population diversifiée permet d'explorer efficacement l'espace de recherche et de réduire le risque de convergence prématurée.

1.3 Fonction d'adaptation (fitness)

La qualité d'un individu est mesurée par une fonction d'adaptation (*fitness*) qui quantifie la performance de la solution.

L'objectif est de maximiser (ou minimiser) cette fonction selon le problème.

1.4 Opérateurs génétiques

Sélection

La sélection choisit les individus qui auront le droit de se reproduire. Les individus ayant une meilleure fitness ont une probabilité plus élevée d'être sélectionnés.

Méthodes courantes :
— sélection par roulette,
— sélection par tournoi,
— sélection élitiste.

Croisement

Le croisement (recombinaison) combine les chromosomes de deux parents pour générer un ou plusieurs descendants. Exemples de croisements :

— croisement à un point,
— croisement à deux points,
— croisement uniforme.

Mutation

La mutation introduit une modification aléatoire dans un chromosome afin de maintenir la diversité génétique et d'éviter le piégeage dans des optima locaux. Elle est appliquée avec une faible probabilité.

1.5 Remplacement et critère d'arrêt

Une nouvelle population est formée à partir des descendants (et éventuellement des meilleurs parents, via l'élitisme). L'algorithme s'arrête typiquement lorsque :
— un nombre maximal de générations est atteint,
— la fitness n'évolue plus significativement,
— une valeur jugée satisfaisante est obtenue.

1.6 Exemples d'utilisation

Exemple 1 : maximisation d'une fonction

On cherche à maximiser une fonction non linéaire $f(x)$ sur un intervalle $x \in [a, b]$. Chaque individu code une valeur x (codage réel) et la fitness est $f(x)$. L'algorithme génétique explore l'intervalle et converge progressivement vers une valeur de x donnant un maximum (souvent global, mais pas garanti).

Exemple 2 : problème du sac à dos (optimisation combinatoire)

Dans le problème du sac à dos, on cherche à sélectionner des objets de valeurs v_i et de poids w_i sous contrainte de capacité W. Un chromosome binaire $c_i \in \{0, 1\}$ indique si l'objet i est choisi. La fitness correspond à la somme des valeurs, pénalisée si la contrainte de poids est violée.

Exemple 3 : optimisation de paramètres d'un contrôleur

Les AG peuvent optimiser les paramètres d'un contrôleur (PID, ou règles d'un contrôleur flou) afin de minimiser un critère tel que l'erreur quadratique moyenne, le dépassement, ou le temps de réponse.

1.7 Pseudo-code d'un algorithme génétique

1: Initialiser une population P de N individus (aléatoire)
2: Évaluer la fitness de chaque individu de P
3: **while** critère d'arrêt non satisfait **do**
4: Sélectionner des parents à partir de P
5: Appliquer le croisement pour générer des descendants
6: Appliquer la mutation sur les descendants

7: Évaluer la fitness des descendants
8: Former la nouvelle population P (avec ou sans élitisme)
9: **end while**
10: Retourner le meilleur individu trouvé

1.8 Algorithme génétique simple : code MATLAB

Exemple : maximiser $f(x) = x\sin(10x) + 2$ sur $x \in [0, 2]$ (codage réel).

```
clc; clear; close all;

% Fonction à maximiser
f = @(x) x.*sin(10*x) + 2;

% Paramètres GA
N = 30;           % taille population
G = 50;           % nb générations
pc = 0.8;         % proba croisement
pm = 0.1;         % proba mutation
a = 0; b = 2;     % bornes

% Initialisation population (réels)
pop = a + (b-a)*rand(N,1);

for g = 1:G
    % Fitness
    fit = f(pop);

    % --- Sélection par tournoi (k=2) ---
    parents = zeros(N,1);
    for i = 1:N
        i1 = randi(N); i2 = randi(N);
        if fit(i1) > fit(i2), parents(i)=pop(i1);
        else,                 parents(i)=pop(i2);
        end
    end

    % --- Croisement (moyenne) ---
    newpop = parents;
    for i = 1:2:N-1
        if rand < pc
            alpha = rand;
            x1 = parents(i);   x2 = parents(i+1);
            newpop(i)   = alpha*x1 + (1-alpha)*x2;
            newpop(i+1) = (1-alpha)*x1 + alpha*x2;
        end
    end
```

```
    % --- Mutation (bruit gaussien) ---
    for i = 1:N
        if rand < pm
            newpop(i) = newpop(i) + 0.1*randn;
        end
    end

    % --- Respect des bornes ---
    newpop = min(max(newpop,a),b);

    % --- Élitisme : conserver le meilleur ---
    [bestFit, idx] = max(fit);
    bestX = pop(idx);

    pop = newpop;
    % remplacer un individu par le meilleur précédent
    pop(randi(N)) = bestX;

    fprintf('Gen %d : best f=%.4f at x=%.4f\n', g, bestFit, bestX);
end

% Résultat final
fit = f(pop);
[bestFit, idx] = max(fit);
bestX = pop(idx);

fprintf('\nMeilleure solution finale : x=%.6f, f(x)=%.6f\n', bestX, bestFit);

% Visualisation
xx = linspace(a,b,1000);
figure; plot(xx,f(xx),'LineWidth',1.5); hold on;
plot(bestX,bestFit,'ro','MarkerFaceColor','r');
grid on; xlabel('x'); ylabel('f(x)');
title('Optimisation par algorithme génétique (GA simple)');
```

1.9 Avantages et limites

Avantages

- exploration globale de l'espace de recherche,
- robustesse face aux fonctions complexes et non dérivables,
- flexibilité (codages variés, multi-objectifs, contraintes).

Limites

- coût de calcul parfois élevé,

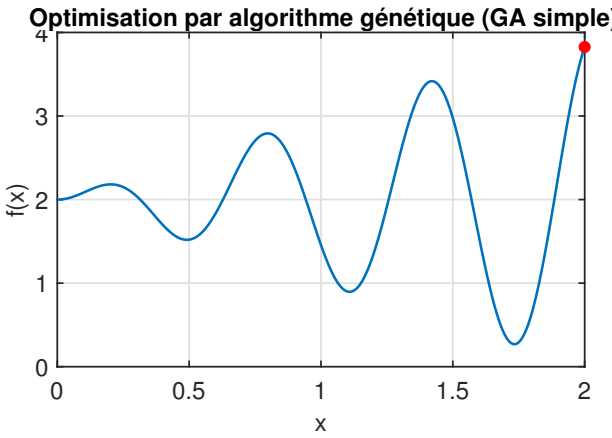


FIGURE 6.1 – Optimisation par algorithme génétique (GA simple)

— réglage délicat des paramètres (N, p_c, p_m),
— convergence prématurée possible si la diversité est faible.

1.10 Domaines d'application

Les algorithmes génétiques sont utilisés en :
— optimisation combinatoire (TSP, sac à dos),
— conception de réseaux et planification,
— apprentissage et identification de modèles,
— robotique, commande et systèmes autonomes,
— bio-informatique.

2 Programmation génétique

La programmation génétique (PG) est une extension des algorithmes génétiques dans laquelle les individus ne représentent plus de simples paramètres, mais des **programmes**, des **expressions mathématiques** ou des **arbres de décision**. L'objectif est de faire évoluer automatiquement des programmes capables de résoudre une tâche donnée.

La programmation génétique a été introduite par John R. Koza dans les années 1990 et constitue aujourd'hui une méthode puissante d'optimisation symbolique.

2.1 Principe général

Contrairement aux algorithmes génétiques classiques, où les chromosomes sont souvent des chaînes binaires ou des vecteurs réels, les individus en programmation génétique sont représentés sous forme d'arbres syntaxiques.

Chaque individu est composé :
— de **nœuds internes** représentant des fonctions ou opérateurs,
— de **feuilles** représentant des variables d'entrée ou des constantes.

L'évolution agit directement sur la structure des programmes.

2.2 Représentation arborescente

Un programme est représenté par un arbre dont :
— la racine correspond à l'opération principale,
— les sous-arbres représentent des sous-expressions.
Exemple :

$$f(x) = x^2 + 3x$$

peut être représenté sous forme d'arbre avec les opérateurs $(+, \times)$ et les opérandes $(x, 2, 3)$.

Cette représentation permet une grande flexibilité, mais nécessite des mécanismes spécifiques pour garantir la validité des programmes générés.

2.3 Population initiale

La population initiale est constituée d'un ensemble de programmes générés aléatoirement à partir :
— d'un ensemble de fonctions autorisées (ex. $+, -, \times, /$),
— d'un ensemble de terminaux (variables, constantes).
La profondeur des arbres est généralement limitée afin d'éviter une complexité excessive.

2.4 Fonction d'évaluation

Chaque programme est évalué à l'aide d'une fonction de fitness qui mesure sa capacité à résoudre le problème considéré.

Selon l'application, la fitness peut correspondre :
— à une erreur de prédiction,
— à une performance de commande,
— à une distance par rapport à une solution cible.

2.5 Opérateurs de programmation génétique

Sélection

Comme pour les algorithmes génétiques, la sélection favorise les programmes les plus performants.

Croisement arborescent

Le croisement consiste à échanger des sous-arbres entre deux programmes parents afin de créer de nouveaux programmes.

Cet opérateur permet de combiner des parties fonctionnelles de différents individus.

Mutation

La mutation modifie aléatoirement un nœud ou un sous-arbre, par exemple en remplaçant une fonction ou un terminal.

Elle permet d'introduire de nouvelles structures et d'éviter la stagnation de la population.

2.6 Pseudo-code de la programmation génétique

```
Initialiser une population de programmes aléatoires
Évaluer la fitness de chaque programme
while critère d'arrêt non satisfait do
    Sélectionner des programmes parents
    Appliquer le croisement arborescent
    Appliquer la mutation
    Évaluer les nouveaux programmes
    Former la nouvelle population
end while
Retourner le meilleur programme
```

2.7 Exemples d'applications

La programmation génétique est utilisée dans :
— la découverte automatique de lois mathématiques,
— la régression symbolique,
— la génération automatique de règles ou de contrôleurs,
— la modélisation de systèmes complexes,
— l'optimisation de structures algorithmiques.

2.8 Avantages et limites

Avantages

— génération automatique de solutions interprétables,
— grande flexibilité de représentation,
— capacité à découvrir des structures inédites.

Limites

— coût de calcul élevé,
— risque de croissance excessive des arbres (bloat),
— réglage délicat des paramètres.

3 Essaims particulaires (Particle Swarm Optimization)

Les algorithmes d'essaims particulaires (Particle Swarm Optimization, PSO) font partie des méthodes d'intelligence collective inspirées du comportement social observé chez certains animaux, tels que les bancs de poissons ou les essaims d'oiseaux. Introduit par Kennedy et Eberhart en 1995, le PSO repose sur la coopération entre individus simples pour résoudre des problèmes d'optimisation.

Contrairement aux algorithmes génétiques, le PSO ne fait pas appel à des opérateurs de croisement ou de mutation, mais à un mécanisme de déplacement collectif guidé par l'expérience individuelle et collective.

3.1 Principe général

Un essaim est constitué d'un ensemble de particules se déplaçant dans l'espace de recherche. Chaque particule représente une solution candidate au problème d'optimisation.

Chaque particule est caractérisée par :

— une **position** représentant une solution,
— une **vitesse** qui détermine son déplacement,
— une mémoire de sa meilleure position personnelle (*best personal*, notée *pbest*),
— une information sur la meilleure position trouvée par l'essaim (*best global*, notée *gbest*).

L'algorithme évolue par itérations successives au cours desquelles les particules ajustent leur trajectoire.

3.2 Mise à jour de la vitesse et de la position

À chaque itération, la vitesse et la position de chaque particule sont mises à jour selon les équations suivantes :

$$v_i(t+1) = w\, v_i(t) + c_1 r_1 \Big(pbest_i - x_i(t)\Big) + c_2 r_2 \Big(gbest - x_i(t)\Big)$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

où :

— $x_i(t)$ et $v_i(t)$ sont respectivement la position et la vitesse de la particule i à l'instant t,
— w est le coefficient d'inertie,
— c_1 et c_2 sont les coefficients d'accélération (cognitif et social),
— r_1 et r_2 sont des nombres aléatoires uniformes dans $[0, 1]$.

Ces équations traduisent l'équilibre entre exploration de nouvelles régions et exploitation des meilleures solutions connues.

3.3 Interprétation des composantes

La mise à jour de la vitesse comprend trois composantes principales :

— la **composante inertielle** (wv_i), qui maintient la direction du mouvement,
— la **composante cognitive**, liée à l'expérience personnelle de la particule,
— la **composante sociale**, liée à l'influence du groupe.

Le réglage de ces paramètres a une influence directe sur la convergence de l'algorithme.

3.4 Initialisation et critères d'arrêt

Les particules sont initialisées avec des positions et des vitesses aléatoires dans l'espace de recherche. L'algorithme s'arrête lorsque :

— un nombre maximal d'itérations est atteint,
— la solution converge vers une valeur stable,
— une valeur de performance jugée satisfaisante est obtenue.

3.5 Pseudo-code de l'algorithme PSO

```
Initialiser un essaim de particules (positions et vitesses)
Évaluer la fitness de chaque particule
Initialiser pbest et gbest
while critère d'arrêt non satisfait do
    for chaque particule do
        Mettre à jour la vitesse
        Mettre à jour la position
        Évaluer la nouvelle position
        Mettre à jour pbest si nécessaire
    end for
    Mettre à jour gbest
end while
Retourner la meilleure solution gbest
```

3.6 Exemples d'applications

Les algorithmes PSO sont utilisés dans :

— l'optimisation continue de fonctions complexes,
— l'apprentissage automatique et l'ajustement de paramètres,
— la robotique et la planification de trajectoires,
— la commande de systèmes non linéaires,
— la bio-informatique et le traitement du signal.

3.7 Avantages et limites

Avantages

— simplicité de mise en œuvre,
— peu de paramètres à régler,
— convergence rapide dans de nombreux cas,
— bonne capacité d'exploration globale.

Limites

— risque de convergence prématurée,
— sensibilité au choix des paramètres,
— performances parfois limitées sur les problèmes très multimodaux.

4 Colonies de fourmis (Ant Colony Optimization)

Les algorithmes de colonies de fourmis (Ant Colony Optimization, ACO) appartiennent à la famille des méthodes d'intelligence collective. Ils sont inspirés du comportement coopératif observé chez les fourmis réelles lors de la recherche de nourriture.

Introduits dans les années 1990 par Marco Dorigo, les algorithmes ACO exploitent un mécanisme de communication indirecte appelé *stigmergie*, basé sur le dépôt et l'évaporation de phéromones.

4.1 Principe général

Dans la nature, les fourmis sont capables de trouver des chemins courts entre leur nid et une source de nourriture sans contrôle centralisé. Chaque fourmi se déplace de manière relativement simple, mais l'interaction collective conduit à l'émergence d'un comportement global optimal.

Dans un algorithme ACO :

— chaque fourmi artificielle construit progressivement une solution,
— les solutions sont guidées par des informations locales (phéromones),
— les meilleures solutions sont renforcées au fil des itérations.

4.2 Rôle des phéromones

Les phéromones sont des traces déposées par les fourmis sur les éléments du problème (arêtes d'un graphe, choix de décisions, etc.).

Plus une solution est de bonne qualité, plus la quantité de phéromones déposée est importante. Inversement, les phéromones s'évaporent avec le temps, ce qui permet d'éviter une convergence prématurée vers des solutions sous-optimales.

4.3 Choix probabiliste

À chaque étape, une fourmi choisit son prochain déplacement selon une probabilité dépendant :

— de la quantité de phéromones présentes,
— d'une information heuristique (distance, coût, visibilité).

La probabilité de choisir l'élément j depuis l'élément i est généralement définie par :

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \, \eta_{ij}^{\beta}}{\sum_{k \in \text{Voisins}} \tau_{ik}^{\alpha} \, \eta_{ik}^{\beta}}$$

où :

— τ_{ij} est la quantité de phéromones,
— η_{ij} est l'information heuristique,
— α et β sont des paramètres d'influence.

4.4 Mise à jour des phéromones

Après la construction des solutions par toutes les fourmis, les phéromones sont mises à jour selon deux mécanismes :

— **évaporation** : réduction globale des phéromones,
— **renforcement** : ajout de phéromones sur les éléments des meilleures solutions.

Ce mécanisme permet un compromis entre exploration et exploitation.

4.5 Pseudo-code de l'algorithme ACO

```
Initialiser les phéromones
while critère d'arrêt non satisfait do
  for chaque fourmi do
    Construire une solution de manière probabiliste
    Évaluer la solution
  end for
  Mettre à jour les phéromones (évaporation + renforcement)
end while
Retourner la meilleure solution trouvée
```

4.6 Exemples d'applications

Les algorithmes de colonies de fourmis sont principalement utilisés pour :

— les problèmes de routage et de graphes,
— le problème du voyageur de commerce (TSP),
— l'ordonnancement et la planification,
— l'optimisation combinatoire,
— les réseaux de communication.

4.7 Avantages et limites

Avantages

— très bonne adaptation aux problèmes combinatoires,
— exploration distribuée et robuste,
— capacité d'adaptation dynamique.

Limites

— temps de calcul parfois élevé,
— réglage délicat des paramètres (α, β, taux d'évaporation),
— convergence parfois lente.

Chapitre 7

Probabilité et raisonnement probabiliste

1 Introduction

Dans de nombreux domaines de l'intelligence artificielle, les systèmes doivent fonctionner en présence d'incertitude, d'informations incomplètes ou bruitées. Le raisonnement probabiliste fournit un cadre mathématique rigoureux permettant de représenter cette incertitude et de raisonner de manière cohérente à partir de connaissances partielles. Ce chapitre présente les notions fondamentales de la probabilité et leur utilisation dans le raisonnement et l'inférence probabilistes, ainsi que les réseaux bayésiens comme outil structuré de modélisation.

2 Notions fondamentales de probabilité

La probabilité permet de quantifier le degré de croyance associé à la réalisation d'un événement. Elle est définie sur un ensemble d'événements et prend des valeurs comprises entre 0 et 1. Un événement certain a une probabilité égale à 1, tandis qu'un événement impossible a une probabilité égale à 0.

Les probabilités obéissent à des règles fondamentales assurant la cohérence du raisonnement, notamment la normalisation et l'additivité. Ces règles constituent la base du raisonnement probabiliste utilisé en intelligence artificielle.

Exemple : Indépendance probabiliste

Énoncé : Soient deux événements A et B tels que :

$$P(A) = 0.4, \quad P(B) = 0.5, \quad P(A \cap B) = 0.2$$

Déterminer si les événements A et B sont indépendants.

Solution : Deux événements sont indépendants si :

$$P(A \cap B) = P(A)P(B)$$

$$P(A)P(B) = 0.4 \times 0.5 = 0.2$$

Cette valeur étant égale à $P(A \cap B)$, on conclut que :

$$\boxed{A \text{ et } B \text{ sont indépendants}}$$

3 Probabilité conditionnelle

La probabilité conditionnelle permet de mettre à jour la probabilité d'un événement lorsqu'une information supplémentaire est disponible. La probabilité de A sachant que B est réalisé est définie par :

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \quad P(B) \neq 0$$

Cette notion est essentielle pour l'inférence et la prise de décision sous incertitude.

Exemple : Probabilité conditionnelle

Énoncé : Dans une population, 2% des individus sont atteints d'une maladie M. Un test possède les caractéristiques suivantes :

$$P(T^+ \mid M) = 0.95, \quad P(T^- \mid \overline{M}) = 0.90$$

Calculer la probabilité qu'un individu soit malade sachant que le test est positif.

Solution :

$$P(T^+ \mid \overline{M}) = 0.10, \quad P(\overline{M}) = 0.98$$

$$P(T^+) = 0.95 \times 0.02 + 0.10 \times 0.98 = 0.117$$

$$P(M \mid T^+) = \frac{0.95 \times 0.02}{0.117} \approx 0.162$$

$$\boxed{P(M \mid T^+) \approx 16.2\%}$$

4 Théorème de Bayes

Le théorème de Bayes est un résultat fondamental du raisonnement probabiliste. Il permet de calculer une probabilité a posteriori à partir d'une probabilité a priori et d'une observation :

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Ce théorème joue un rôle central dans le diagnostic, la prédiction et l'apprentissage.

Exemple : Application du théorème de Bayes

Énoncé : On considère les événements :

$$P(R) = 0.3, \quad P(S \mid R) = 0.9, \quad P(S \mid \overline{R}) = 0.2$$

Calculer la probabilité qu'il pleuve sachant que le sol est mouillé.

Solution :

$$P(S) = 0.9 \times 0.3 + 0.2 \times 0.7 = 0.41$$

$$P(R \mid S) = \frac{0.9 \times 0.3}{0.41} \approx 0.659$$

$$\boxed{P(R \mid S) \approx 65.9\%}$$

5 Raisonnement probabiliste

Le raisonnement probabiliste consiste à tirer des conclusions à partir d'un ensemble de connaissances incertaines exprimées sous forme de probabilités. Contrairement au raisonnement logique classique, il manipule des degrés de croyance et permet de raisonner même lorsque certaines informations sont absentes ou imprécises.

6 Réseaux bayésiens

Les réseaux bayésiens sont des modèles graphiques probabilistes basés sur des graphes orientés acycliques. Chaque nœud représente une variable aléatoire et chaque arc représente une relation de dépendance probabiliste. Chaque variable est associée à une table de probabilités conditionnelles.

7 Inférence dans les réseaux bayésiens

L'inférence consiste à calculer la probabilité d'une ou plusieurs variables sachant certaines observations. Les réseaux bayésiens permettent de combiner efficacement les connaissances a priori et les données observées.

Exemple : Diagnostic à l'aide d'un réseau bayésien

Énoncé : On considère un système de diagnostic avec :

— P : panne

— B : bruit anormal

— V : vibration

Les probabilités sont :

$$P(P) = 0.1$$

$$P(B \mid P) = 0.8, \quad P(B \mid \overline{P}) = 0.1$$

$$P(V \mid P) = 0.7, \quad P(V \mid \overline{P}) = 0.2$$

Calculer la probabilité de panne sachant qu'un bruit anormal et une vibration sont observés.

Solution :

$$P(B \cap V \mid P) = 0.8 \times 0.7 = 0.56$$

$$P(B \cap V \mid \overline{P}) = 0.1 \times 0.2 = 0.02$$

$$P(B \cap V) = 0.56 \times 0.1 + 0.02 \times 0.9 = 0.074$$

$$P(P \mid B \cap V) = \frac{0.56 \times 0.1}{0.074} \approx 0.757$$

$$\boxed{P(P \mid B \cap V) \approx 75.7\%}$$

8 Exercice d'application

Énoncé :

On considère un système de diagnostic modélisé par un réseau bayésien simple comportant trois variables binaires :

— C : *Cause du défaut* (présente / absente)
— S_1 : *Symptôme 1* (observé / non observé)
— S_2 : *Symptôme 2* (observé / non observé)

La variable C influence directement les variables S_1 et S_2. Les probabilités associées sont données ci-dessous :

$$P(C) = 0.15$$

$$P(S_1 \mid C) = 0.8, \quad P(S_1 \mid \overline{C}) = 0.2$$

$$P(S_2 \mid C) = 0.7, \quad P(S_2 \mid \overline{C}) = 0.1$$

1. Vérifier si les variables S_1 et S_2 sont indépendantes sachant C.
2. Calculer la probabilité $P(S_1 \cap S_2 \mid C)$.
3. Calculer la probabilité $P(S_1 \cap S_2)$.
4. En utilisant le théorème de Bayes, calculer la probabilité que la cause C soit présente sachant que les deux symptômes sont observés.
5. Interpréter le résultat obtenu.

Solution

1. Indépendance conditionnelle

Dans un réseau bayésien, deux variables sont conditionnellement indépendantes sachant leur cause commune. Ainsi :

$$P(S_1 \cap S_2 \mid C) = P(S_1 \mid C)P(S_2 \mid C)$$

Les variables S_1 et S_2 sont donc conditionnellement indépendantes sachant C.

2. Calcul de $P(S_1 \cap S_2 \mid C)$

$$P(S_1 \cap S_2 \mid C) = 0.8 \times 0.7 = 0.56$$

3. Calcul de $P(S_1 \cap S_2)$

$$P(S_1 \cap S_2) = P(S_1 \cap S_2 \mid C)P(C) + P(S_1 \cap S_2 \mid \overline{C})P(\overline{C})$$

Avec :

$$P(S_1 \cap S_2 \mid \overline{C}) = 0.2 \times 0.1 = 0.02$$

$$P(\overline{C}) = 1 - 0.15 = 0.85$$

Donc :

$$P(S_1 \cap S_2) = 0.56 \times 0.15 + 0.02 \times 0.85$$

$$P(S_1 \cap S_2) = 0.084 + 0.017 = 0.101$$

4. Calcul de $P(C \mid S_1 \cap S_2)$

D'après le théorème de Bayes :

$$P(C \mid S_1 \cap S_2) = \frac{P(S_1 \cap S_2 \mid C)P(C)}{P(S_1 \cap S_2)}$$

$$P(C \mid S_1 \cap S_2) = \frac{0.56 \times 0.15}{0.101} \approx 0.832$$

$$\boxed{P(C \mid S_1 \cap S_2) \approx 83.2\%}$$

5. Interprétation

L'observation simultanée des deux symptômes augmente fortement la probabilité de la présence de la cause. Ce résultat illustre l'efficacité du raisonnement probabiliste et des réseaux bayésiens pour le diagnostic et la prise de décision en présence d'incertitude.

Chapitre 8

Systèmes experts et applications

1 Introduction

Les systèmes experts constituent l'une des premières applications concrètes de l'intelligence artificielle. Ils visent à reproduire le raisonnement d'un expert humain dans un domaine donné afin d'aider à la prise de décision, au diagnostic ou à la résolution de problèmes complexes. Contrairement aux algorithmes purement numériques, les systèmes experts reposent principalement sur la représentation explicite des connaissances et sur des mécanismes de raisonnement symbolique.

Ce chapitre présente les principes fondamentaux des systèmes experts, leurs extensions floues, ainsi que leurs applications dans la prise de décision et le diagnostic.

2 Systèmes experts

Un système expert est un programme informatique capable de résoudre des problèmes dans un domaine spécifique en utilisant des connaissances et des règles similaires à celles employées par un expert humain.

2.1 Architecture d'un système expert

Un système expert classique est généralement composé des éléments suivants :

- **Base de connaissances** : contient les faits et les règles du domaine.
- **Moteur d'inférence** : applique les règles aux faits pour produire des conclusions.
- **Base de faits** : contient les informations connues sur le problème.
- **Interface utilisateur** : permet l'interaction avec l'utilisateur.

2.2 Règles de production

Les connaissances sont souvent représentées sous forme de règles :

SI condition ALORS conclusion

Ces règles permettent de formaliser le raisonnement de l'expert de manière simple et interprétable.

Exemple : Règle simple

Règle : SI la température est élevée ET le courant est élevé ALORS le système est en surcharge.

3 Mécanismes d'inférence

Le moteur d'inférence détermine quelles règles doivent être appliquées. Deux stratégies principales existent :

— **Chaînage avant** : raisonnement à partir des faits vers les conclusions.
— **Chaînage arrière** : raisonnement à partir d'un objectif vers les faits nécessaires.

Exemple : Chaînage avant

Faits initiaux : Température élevée, courant élevé.
Conclusion : Le moteur d'inférence active la règle correspondante et conclut que le système est en surcharge.

4 Limites des systèmes experts classiques

Les systèmes experts classiques présentent certaines limites :

— Difficulté à gérer l'incertitude
— Sensibilité aux données imprécises
— Acquisition des connaissances parfois complexe

Ces limites ont conduit au développement des systèmes experts flous.

5 Systèmes experts flous

Les systèmes experts flous combinent la logique floue avec les systèmes experts afin de traiter l'imprécision et l'incertitude. Les règles floues sont de la forme :

$$\text{SI } x \text{ est } A \text{ ALORS } y \text{ est } B$$

où A et B sont des ensembles flous.

Exemple : Règle floue

SI la température est *élevée* ET la vibration est *forte* ALORS le risque de panne est *important*.

6 Processus de raisonnement flou

Le raisonnement flou comprend généralement :

— Fuzzification des entrées
— Évaluation des règles

— Agrégation des conclusions
— Défuzzification

Ce processus permet d'obtenir une décision même lorsque les données sont imprécises.

7 Prise de décision

Les systèmes experts sont largement utilisés pour aider à la prise de décision dans des environnements complexes. Ils permettent de comparer plusieurs scénarios et de proposer une action optimale ou acceptable.

Exemple : Prise de décision

Règles :
— SI le risque est faible ALORS continuer le fonctionnement.
— SI le risque est moyen ALORS surveiller le système.
— SI le risque est élevé ALORS arrêter le système.

8 Application au diagnostic

Le diagnostic est l'un des domaines d'application majeurs des systèmes experts. Il consiste à identifier la cause probable d'un dysfonctionnement à partir de symptômes observés.

Exemple : Diagnostic par règles

Faits observés :
— Bruit anormal
— Échauffement

Règles :
— SI bruit anormal ET échauffement ALORS défaut mécanique.
— SI vibration ET échauffement ALORS déséquilibre.

Conclusion : Le système expert conclut à un défaut mécanique.

9 Exercice d'application

Énoncé : On considère un système expert flou destiné au diagnostic d'un système. Les règles sont :

— SI température est élevée ET vibration est forte ALORS panne est grave.
— SI température est moyenne ET vibration est faible ALORS panne est légère.

Expliquer comment le système peut prendre une décision lorsque la température est moyenne à élevée et la vibration est moyenne.

Solution

Les valeurs d'entrée sont d'abord fuzzifiées. Les règles sont ensuite activées avec des degrés partiels. Les conclusions sont agrégées et une étape de défuzzification permet d'obtenir un niveau de gravité intermédiaire. Le système conclut à une panne modérée nécessitant une surveillance ou une intervention préventive.