Commande et Identification Floue

Inclus :

J.--S. Roger Jang, "ANFIS: Adaptive--network-- based fuzzy inference systems," IEEE Trans. on System, Man and Cybernetics, Vol. 23, No. 3, pp. 665-685, May/June, 1993. http://citeseer.ist.psu.edu/jang93anfis.html
Jang, J. R. and Sun, C. Neuro-Fuzzy Modeling and Control, Proceedings of the IEEE, 83, 378-406. http://citeseer.ist.psu.edu/jang95neurofuzzy.html

Gonzalo Cabodevila
gonzalo.cabodevila@femto-st.fr

3ème année

École Nationale Supérieure de
Mécanique et des Microtechniques
26, chemin de l'Épitaphe
25030 Besançon cedex – FRANCE
http://intranet-tice.ens2m.fr

Table des matières

Première partie

Commande floue

Chapitre 1

Introduction

La conception et la mise en oeuvre de systèmes d'informations avancées, de logiciels d'aide à la décision, de postes de supervisions de procédés industriels (avec les problèmes de communication homme machine qui les accompagnent) se caractérisent par la constante manipulation d'informations dont beaucoup sont subjectives, imprécises, vagues, incertaines.

Mener à bien cette intégration de l'homme dans son environnement informatique, exploiter correctement les savoirs, automatiser certaines tâches, tout cela requiert la formalisation et la mécanisation de méthodes humaines pour des raisonnements empiriques ou naturels, la rationalisation de procédures de choix. De façon générale, on est confronté à la nécessité de modéliser la connaissance, problématique qui se trouve en rupture avec la tradition des sciences dites objectives, lesquelles se préoccupent essentiellement de la modélisation de l'univers physique.

Les problèmes de représentation et d'utilisation des connaissances sont au centre d'une discipline scientifique relativement nouvelle et en tout cas controversée, qu'on appelle l'intelligence artificielle. Cette discipline a eu un impact limité, jusqu'à une date récente, sur les applications industrielles, parce qu'elle a mis l'accent, de façon exclusive, sur le traitement symbolique de la connaissance, par opposition à la modélisation numérique utilisée traditionnellement dans les sciences de l'ingénieur. Plus récemment, on a assisté à un retour du numérique dans ces problèmes d'intelligence artificielle, avec les réseaux neuro-mimétiques et la logique floue. Alors que les réseaux neuro-mimétiques proposent une approche implicite de type <boite noire> de la représentation des connaissances, très analogue à la démarche de l'identification des systèmes en automatique, la logique floue est plus conforme à l'intelligence artificielle symbolique, qui met en avant la notion de raisonnement, et où les connaissances sont codées explicitement. Néanmoins, la logique floue permet de faire le lien entre modélisation numérique et modélisation symbolique, ce qui a permis des développements industriels spectaculaires à partir d'algorithmes très simple de traduction de connaissances symboliques en entité numérique et inversement. La théorie des ensembles flous a également donné naissance à un traitement original de l'incertitude, fondée sur l'idée d'ordre, et qui permet de formaliser le traitement de l'ignorance partielle et de l'inconsistance dans les systèmes d'informations avancés. Les ensembles flous ont également eu un impact sur les techniques de classification automatique, et ont contribué à un certains renouvellement des approches existantes de l'aide à la décision.

1.1 Exemple introductif

Afin de mettre en évidence le principe fondamental de la logique floue, on présente un exemple simple, celui de la classification des personnes en trois ensembles <jeune>, <entre deux âges >, <âgé>.

Selon la logique classique (logique de Boole), qui n'admet pour les variables que les deux valeurs 0 et 1, une telle classification pourrait se faire comme la figure 1.1 Toutes les personnes âgées de mois de 25 ans sont alors considérées jeunes et toutes les personnes âgées de plus de 50 ans comme des vieux.

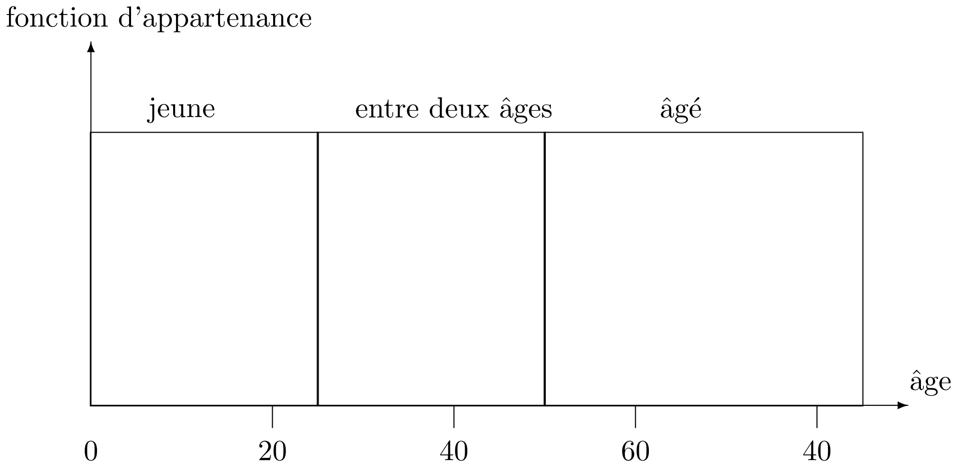


FIGURE 1.1 – Classification des personnes en trois ensembles selon la logique classique.

Cependant, une telle logique de classification n'est même pas logique. Pourquoi une personne, lorsqu'elle a eu 50 ans, doit-elle être considérée comme appartenant à l'ensemble <âgé> ? En réalité, un tel passage se fait progressivement et individuellement. La logique floue, dont les variables peuvent prendre n'importe quelle valeur entre 0 et 1, permet de tenir compte de cette réalité. Les limites ne varient pas soudainement, mais progressivement. La figure 1.2 montre une classification possible pour l'exemple précédent, cette fois-ci à l'aide de la logique floue. Ainsi une personne de 25 ans appartient à l'ensemble <jeune> avec une valeur μ =0,75 de la fonction d'appartenance et à l'ensemble <entre deux âges> avec μ =0,25. Par contre une personne âgée de 65 ans appartient avec une valeur μ =1 de la fonction d'appartenance à l'ensemble <âgé>.

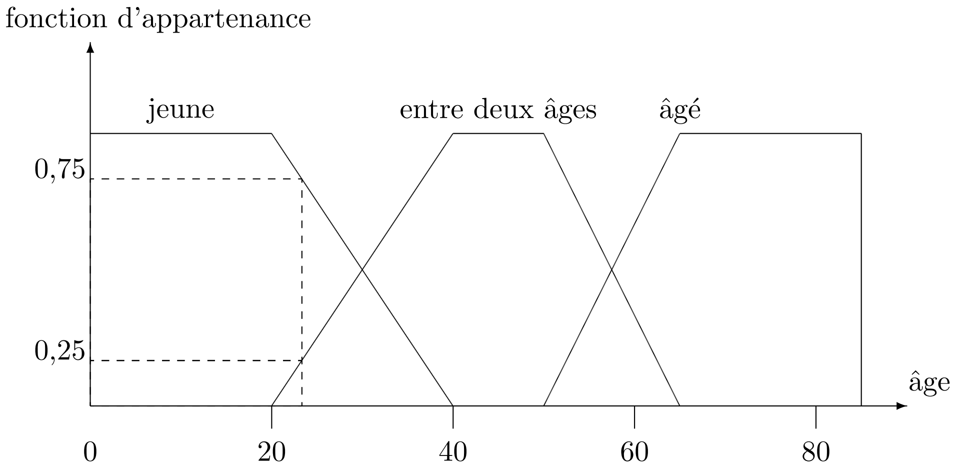


FIGURE 1.2 – Classification des personnes en trois ensembles selon la logique floue.

1.2 Historique

Nous venons de voir que la logique floue sert à représenter des connaissances incertaines et imprécises. La commande floue sert à prendre une décision même si l'on ne peut estimer les entrées/sorties qu'à partir de prédicats vagues ou lorsque ses entrée/sorties sont entachées d'erreurs que l'on ne peut évaluer que grossièrement.

On conçoit l'intérêt de faire entrer l'approche floue dans la régulation ou l'asservissement des processus industriels, pour lesquels les informations disponibles sont souvent imprécises, incertaines et

parfois qualitatives, dans des boucles de régulation parfois incomplètes. Le savoir faire de l'opérateur, constitué entre autres souvent de règles simples, lui permet de conduire chaque machine plus correctement parfois qu'un algorithme classique. Les prémisses de la logique floue sont apparues avant les années 1940, avec les premières approches, par des chercheurs américains, du concept d'incertitude. Il a fallu attendre 1965, pour que le concept de sous-ensemble flou soit proposé par L. A. Zadeh, automaticien de réputation internationale, professeur à l'université de Berkeley en Californie, qui a contribué à la modélisation de phénomènes sous forme floue, en vue de pallier les limitations dues aux incertitudes des modèles classiques à équation différentielle. En 1974, M. Mamdani expérimentait la théorie énoncée par Zadeh sur une chaudière à vapeur, matériel dont on connaît la complexité, introduisant ainsi la commande floue dans la régulation d'un processus industriel. Plusieurs applications ont alors vu le jour en Europe, pour des systèmes parfois très complexes, telle la régulation de fours de cimenterie réalisée par la société F. L. Smidt-Fuller. Grâce au chercheur japonais M. Sugeno, la logique floue était introduite au Japon dès 1985. Les sociétés japonaises comprirent l'avantage à la fois technique et commercial de la logique floue :
Avantages
- facilité d'implantation ;
- solution de problèmes multivariables complexes ;
- robustesse vis à vis des incertitudes ;
- possibilité d'intégration du savoir de l'expert.
- Possibilités de commande auto-adaptative aux variations du procédé

Inconvénients
- Réglage empirique.
- Performances en boucle fermée dépendent de l'expertise.
- Il n'existe pas de théorie générale qui caractérise rigoureusement la stabilité, la robustesse..(Difficultés de certification dans le transport, espace)

1.3 Les différents domaines d'applications

Si les premières applications de la commande floue ont été développées en Europe par Mamdani, Van Lauta Lemke, Willayes......dans les années 1975, ce type de commande a été largement utilisé au Japon à la fin des années 80 et au début des années 90, sur le métro de Sendai (Juillet 1987), dans la conception du lave-linge Aïsaïgo Day Fuzzy de Matsushita (Février 1990), par exemple.
Tirant profit d'une évolution de la technologie permettant d'aborder des problèmes en temps réel, à dynamique rapide (plus rapide que dans les premières applications de type conduite de processus lents : chaudière, fours,......), l'industrie japonaise a développé des produits grand public à base de logique floue :
- appareils électroménagers (lave-linge, aspirateurs, autocuiseurs,...) ,
- systèmes audio-visuels (appareils photos autofocus, caméscope à stabilisateur d'images, ...)
- systèmes automobiles embarqués (BVA, ABS, suspension, climatisation,...etc.),
- systèmes autonomes mobiles,
- systèmes de transport (train, métro, ascenseur,...),
- systèmes de conditionnement d'ambiance,
- systèmes de décision, diagnostic, reconnaissance.

1.4 Le concept de la logique floue

Ce rapport permet de considérer des classes d'objets dont les frontières ne sont pas clairement déterminées, par l'introduction d'une fonction caractéristique (fonctions d'appartenance des objets à la classe) prenant des valeurs courantes entre 0 et 1, contrairement aux ensembles <booléens>, dont la fonction caractéristique ne prend que deux valeurs possibles 0 et 1.

Ces ensembles flous ont le grand avantage de constituer une représentation mathématique de labels linguistiques largement utilisés dans l'expression de connaissances expertes, qualitatives et manipulées dans le raisonnement approché qui sera fait à partir de cette connaissance.

Ils apparaissent donc comme un moyen de réaliser l'interface entre l'information numérique (quantitative) et l'information symbolique (linguistique, qualitative).

Chapitre 2

Commande

2.1 Exemple : Contrôle de température d'une douche

Lorsque vous vous retrouvez pour la première fois devant une douche possédant deux robinet distincts, un pour l'eau chaude et un pour l'eau froide, vous ouvrez les deux robinets. Puis, en fonction de la température de l'eau vous appliquez des règles de ce type :

- Si l'eau est trop chaude ALORS ouvrir plus le robinet d'eau froide.
- Si l'eau est trop froide ALORS ouvrir plus le robinet d'eau chaude.
- Si l'eau est trop froide ET le robinet d'eau chaude est ouvert au maximum ALORS fermer plus le robinet d'eau froide.

Voilà, vous faisiez donc de la commande floue de température sans le savoir. Il s'agit maintenant de formaliser tout cela afin de le faire comprendre à un microprocesseur.
Cette partie va nous permettre de formaliser mathématiquement ces concepts. On verra dans l'ordre

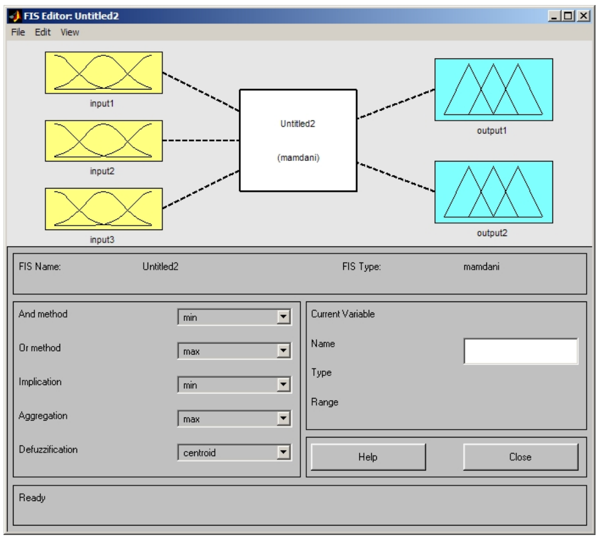
1. Comment fuzzifier,
   c'est-à-dire passer d'une variable réelle à une variable floue.
2. Comment définir les règles d'inférence.
3. Comment appliquer ces règles,
   c'est-à-dire définir les opérateurs logiques (ET, OU, NON),
   puis définir une méthode pour prendre en compte simultanément toutes ses règles.
4. Comment deffuzifier,
   c'est-à-dire générer une commande pour le système à contrôler.

2.2 La fuzzification

2.2.1 Les fonctions d'appartenance

La première étape d'une commande floue est donc la fuzzification des variables de la commande c'est-à-dire des entrées de la commande (capteurs) mais aussi des sorties (commande des actionneurs).
Nous avons vu précédemment que l'on peut donner un coefficient de confiance à l'affirmation <x appartient à un ensemble A>, par exemple : le coefficient d'appréciation de <l'air à la température égale à 30°C est chaud> vaut 0,6, ce qui signifie que cette température correspond à <plutôt chaud>. On peut pour toute température, donc pour tout x, définir ce coefficient directement de x. Cette propriété se présente facilement par une fonction dite d'appartenance $\mu_A(x)$ à valeurs dans [0,1], la notation signifie <coefficient d'appartenance de x à l'ensemble caractérisé par A>, l'argument x se rattache à la variable linguistique et l'indice A désigne l'ensemble concerné.

De la même manière, une variable y appartiendra à un ensemble B avec une fonction d'appartenance notée $\mu_B(y)$, par exemple <le vent est fort>. On peut associer x et y dans une même phrase, par



FIS Name: Untitled2
FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name
Type
Range

Ready

FIGURE 2.1 – Principe général d'une commande floue au sens de Mandani.

exemple l'ensemble C : <l'air est chaud ET le vent est fort>. La variable z définie par : <air chaud ET vent fort> correspond à l'intersection de <air est chaud> et de <vent est fort>. L'ensemble C correspond à l'intersection des ensembles A et B. La valeur de $\mu_C(z)$ se déduit des valeurs de $\mu_A(x)$ et $\mu_B(y)$. Il existe diverses solutions pour traduire mathématiquement le problème, nous les verrons au paragraphe 2.4.1.

2.2.2 Les différentes formes des fonctions d'appartenance

Comme exemple pour les fonctions d'appartenance, on présente l'exemple de la température d'un réservoir d'eau . Dans le cas le plus simple, on peut distinguer trois valeurs <froid>, <tiède>, et <chaud> de la variable linguistique <température T>. Elles forment trois ensembles flous (voir fig. 2.2). Ainsi, une température de 12°C appartient avec un facteur d'appartenance $\mu = 0,7$ à l'ensemble <froid> et avec $\mu = 0,3$ à l'ensemble <tiède>. Evidemment, le choix caractérisant l'allure trapézoïdale de la fonction d'appartenance est assez arbitraire et doit tenir compte des circonstances particulières.

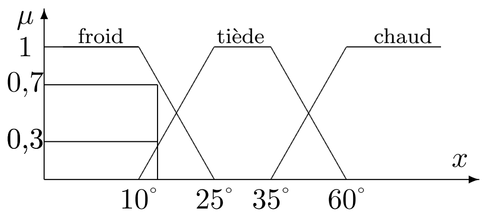


FIGURE 2.2 – Fonctions d'appartenance avec trois ensembles pour la variable <température>.

Souvent,il s'avère nécessaire d'introduire une subdivision plus fine, par exemple par 5 valeurs <très froid>, <froid>, <tiède>, <chaud> et <très chaud> pour la variable linguistique <Température T>, formant ainsi 5 ensembles comme le montre la figure 2.3. La température de 12° appartient, avec μ =0,3, à l'ensemble <tiède>. Evidemment, cette valeur de 12° n'appartient ni à l'ensemble <chaud> ni à celui <très chaud>.

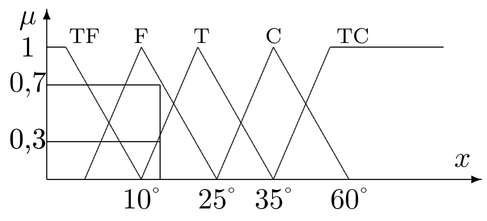


FIGURE 2.3 – Fonctions d'appartenance avec cinq ensembles pour la variable <température>.

2.2.3 Autres formes de fonctions d'appartenance

Le plus souvent, on utilise pour les fonctions d'appartenance des formes trapézoïdales ou triangulaires. Ils s'agit des formes les plus simples, composées par morceaux de droites. L'allure est complètement définie par les points A, B et C pour la forme triangulaire, ou les 4 points A, B, C et D pour la forme trapézoïdale (figure 2.4). Le triangle peut être considéré comme un cas particulier du trapèze lorsque deux points coïncident (B=D). Même la forme rectangulaire (pour représenter la logique classique) est comprise dans le trapèze si les deux premiers points (A,B) et les deux derniers points (C,D) se trouvent sur une verticale. Dans la plupart des cas, en particulier pour le réglage par logique floue, ces deux formes sont suffisantes pour délimiter des ensembles flous.

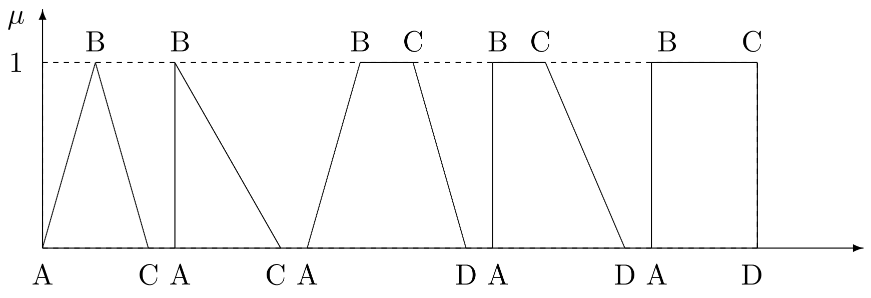


FIGURE 2.4 – Fonctions d'appartenance de formes triangulaires et trapézoïdales.

Il existe d'autres formes de fonctions d'appartenance, moins utilisées à cause du temps de calcul nécessaire, par exemple la forme donnée par :

$$\mu(x) = \frac{1}{2}\left[1 + \cos\frac{\pi(x - x_0)}{2a}\right]; -2a < x < x_0 + 2a \tag{2.1}$$

ou

$$\mu(x) = \frac{1}{1 + \left(\frac{x-x_0}{a}\right)^2}; -\infty < x < +\infty \tag{2.2}$$

Dans ce cas, x_0 détermine la position du sommet μ =1, tandis que le paramètre a impose la largeur du domaine. A noter que cette fonction d'appartenance s'annule seulement pour x tend vers plus ou moins l'infini. Elle est représentée à la figure 2.5.

Enfin, il est possible de composer la fonction d'appartenance par des morceaux de droites. Ainsi, on peut réaliser des formes concaves et convexes comme le montre la figure 2.6.

Les fonctions d'appartenance peuvent être symétriques et distribuées de manière équidistante, comme le montre la figure 2.7(a). Une forme est définie symétrique lorsque les fonctions d'appartenance sont symétriques par rapport $x = 0$. Par contre, la forme est définie équidistante lorsque les maxima des fonctions d'appartenance des différents ensembles sont écartés de manière équidistante.

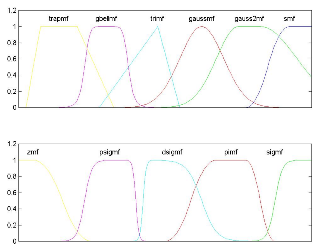


FIGURE 2.5 – Fonctions d'appartenance implantées sous MATLAB.

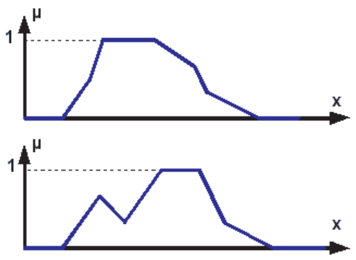


FIGURE 2.6 – Fonctions d'appartenance composées par des morceaux de droites (fonctions convexes et concaves).

2.2.4 Conseils pour le choix des fonctions d'appartenance

La notion d'"univers de discours" se conçoit aisément : reprenons notre concept de température : l'utilisateur pourra décrire la variable <température> par un certain nombre de mots : par exemple <chaud>, <froid>, <tiède>, ou <très chaud>, <assez chaud>, <tiède>, <assez froid>, <très froid>. Pour chacun de ces prédicats, on pourra donner une fonction d'appartenance. L'univers de discours sera considéré comme le domaine de fonctionnement du processus.

Les problèmes qui peuvent se poser sont :
– Combien de prédicats sont nécessaires à la commande et comment les choisir ?
– Est-il nécessaire de choisir chaque prédicat ?

En général, on peut donner un nombre de règles de commande important, mais l'intérêt de la commande floue est que seul un petit nombre de règles est nécessaire. Le nombre de prédicats dépend essentiellement de la manière dont l'expert peut décrire le processus et de la précision souhaitée. Par exemple, en commande floue classique, 5 prédicats (grand, très grand, moyen, petit, très petit) est un bon compromis ; parfois trois peuvent suffire (dans la cas extrêmes, on peut aller jusqu'à 7). Les trois prédicats <petit>, <moyen>, <grand> ou <négatif>, <positif>, et <proche de zéro> servent à la régulation ; les autres prédicats servent à la poursuite. La commande floue permet d'assurer une transition harmonieuse entre les deux modes de fonctionnement.

L'univers de discours d'une variable couvrira l'ensemble des valeurs prises par cette variable. En pratique, l'intersection entre deux prédicats consécutifs est non nulle, de façon à pouvoir exercer une pondération sur la commande. Il en résulte un chevauchement des variables qui doit être suffisant pour permettre une description continue des variables mais pas trop important pour limiter l'imprécision. Il est notamment préférable d'éviter que les fonctions d'appartenance de deux prédicats voisins soient simultanément égales à 1.

En premier lieu, je vous conseille de démarrer avec une fuzzification utilisant 3 à 5 ensembles donc 3 à 5 fonctions d'appartenance pour chaque variable. Les fonctions d'appartenance seront choisies triangulaires et/ou trapézoïdales, symétriques et équidistantes (comme sur la figure 2.7(a)).

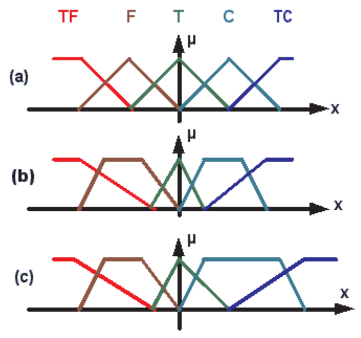


FIGURE 2.7 – (a) Fonction d'appartenance symétrique et équidistante. (b) Fonction d'appartenance symétrique et non équidistante. (c) Fonction d'appartenance non symétrique et non équidistante.

Puis en fonction du comportement obtenu en boucle fermée, vous pouvez :
- augmenter le nombre d'ensembles d'une ou plusieurs variables
  $\Longrightarrow$ plus de souplesse dans les règles.
- désymétriser les fonctions d'appartenance
  $\Longrightarrow$ prise en compte de non-linéarités du système.
- changer la forme des fonctions d'appartenance
  $\Longrightarrow$ variations de commande plus souples.

En tout état de cause, il faut éviter des lacunes ou un chevauchement insuffisant entre les fonctions d'appartenance de deux ensembles voisins (figure 2.11(a)). En effet, cela provoque des zones de non-intervention du régulateur, ce qui conduit à une instabilité de réglage. De même, on doit éviter un chevauchement trop important, surtout avec μ =1 entre deux ensembles voisins (figure 2.11(b)).

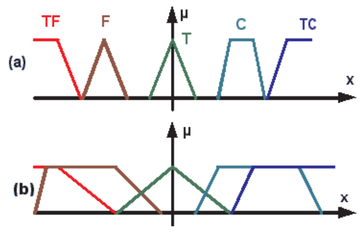


FIGURE 2.8 – (a) Fonction d'appartenance avec lacunes ou chevauchement insuffisant (b) Fonction d'appartenance avec chevauchement trop important.

2.3 La définition des règles d'inférence

C'est là le rôle de l'"expert". En effet l'une des applications de la logique floue et la stabilisation du savoir de l'expert. En effet pour certaines machines particulièrement complexes, seul l'expert (le technicien qui s'en occupe) sait réellement la piloter. Lui seul sait, après des années d'expérience, comment régler la machine.

Le plus souvent, l'établissement des règles relève du bon sens et dans ce cas c'est vous l'expert.

Il est impossible de donner des directives précises concernant l'établissement des règles, cependant les conseils peuvent être utiles :

1. Etablir un minimum de règles au départ, ce n'est que si le comportement du système en boucle fermée n'est pas satisfaisant qu'il faut augmenter le nombre de règles et parfois le nombre de fonctions d'appartenance
2. Méfiez-vous de l'"évidence première" lors de l'établissement des règles
3. Utilisez une matrice d'inférence comme indiqué sur le tableau 2.3, cela évite les règles contradictoires et permet d'imaginer plus facilement la surface de commande (voir § 2.8)

2.3.1 Exemple : un asservissement de moteur

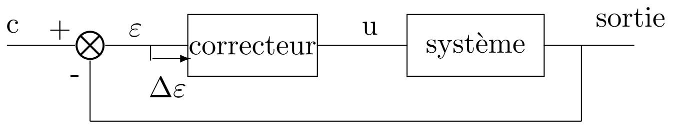


FIGURE 2.9 – Schéma bloc de l'asservissement de la position du moteur

avec :	ε	:	erreur de position
	$\Delta\varepsilon$	:	variation de l'erreur $= \varepsilon(kT_e) - \varepsilon((k-1)T_e)$
	u	:	commande en tension du moteur

La première étape consiste à fuzzifier les variables, cette fuzzification est donnée sur la figure A.3.

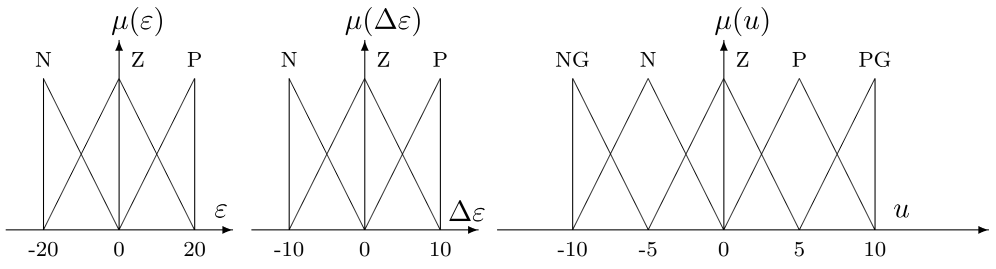


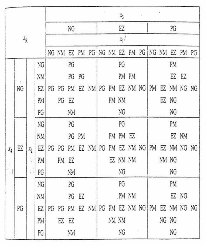
FIGURE 2.10 – Fuzzification des variables

Puis, on définit les règles à appliquer

$$\left.\begin{array}{l}
\text{Si } \varepsilon \text{ est N ET } \Delta\varepsilon \text{ est N ALORS } u \text{ est NG} \\
\text{Si } \varepsilon \text{ est N ET } \Delta\varepsilon \text{ est Z ALORS } u \text{ est N} \\
\text{Si } \varepsilon \text{ est N ET } \Delta\varepsilon \text{ est P ALORS } u \text{ est Z} \\
\text{Si } \varepsilon \text{ est Z ET } \Delta\varepsilon \text{ est N ALORS } u \text{ est N} \\
\text{Si } \varepsilon \text{ est Z ET } \Delta\varepsilon \text{ est Z ALORS } u \text{ est Z} \\
\text{Si } \varepsilon \text{ est Z ET } \Delta\varepsilon \text{ est P ALORS } u \text{ est P} \\
\text{Si } \varepsilon \text{ est P ET } \Delta\varepsilon \text{ est N ALORS } u \text{ est Z} \\
\text{Si } \varepsilon \text{ est P ET } \Delta\varepsilon \text{ est Z ALORS } u \text{ est P} \\
\text{Si } \varepsilon \text{ est P ET } \Delta\varepsilon \text{ est P ALORS } u \text{ est PG}
\end{array}\right\} \underbrace{\begin{array}{|c|c|c|c|}\hline \varepsilon \setminus \Delta\varepsilon & \text{N} & \text{Z} & \text{P} \\ \hline \text{N} & \text{NG} & \text{N} & \text{Z} \\ \hline \text{Z} & \text{N} & \text{Z} & \text{P} \\ \hline \text{P} & \text{Z} & \text{P} & \text{PG} \\ \hline \end{array}}_{\text{Matrice d'inférence}} \qquad (2.3)$$

Nous aurions pu définir la matrice d'inférence suivante : l'asservissement est alors moins rapide.

$\varepsilon \setminus \Delta\varepsilon$	N	Z	P
N	N	N	Z
Z	N	Z	P
P	Z	P	P



x_R				x_3														
				NG					EZ					PG				
				x_1														
				NG	NM	EZ	PM	PG	NG	NM	EZ	PM	PG	NG	NM	EZ	PM	PG
x_4	NG	x_2	NG			PG					PG					PM		
			NM			PG	PG				PM	PM				EZ	EZ	
			EZ	PG	PG	PM	EZ	NM	PG	PM	EZ	NM	NG	PM	EZ	NM	NG	NG
			PM		PG	EZ				PM	NM				EZ	NG		
			PG			NM					NG					NG		
	EZ		NG			PG					PG					PM		
			NM			PG	PM			PM	PM	EZ				EZ	NM	
			EZ	PG	PG	PM	EZ	NM	PG	PM	EZ	NM	NG	PM	EZ	NM	NG	NG
			PM		PM	EZ				EZ	NM	NM			NM	NG		
			PG			NM					NG					NG		
	PG		NG			PG					PG					PM		
			NM			PG	EZ				PM	NM				EZ	NG	
			EZ	PG	PG	PM	EZ	NM	PG	PM	EZ	NM	NG	PM	EZ	NM	NG	NG
			PM		EZ	EZ				NM	NM				NG	NG		
			PG			NM					NG					NG		

FIGURE 2.11 – Exemple de matrice d'inférence pour 4 entrées.

Note : On peut tout aussi bien imaginer des règles de la forme :

Si (ε est NON N ET $\Delta\varepsilon$ est N) OU ε est P ALORS u est NG

Cela devient alors difficile voire impossible à représenter sous forme matricielle

2.4 Les opérateurs de la logique floue

Au paragraphe précédent nous avons défini les règles d'inférence en utilisant les opérations logiques "ET, OU, NON". Il s'agit dans cette partie de définir ces opération logiques dans le cadre de la logique floue.
En effet, il est important de savoir composer entre les divers prédicats et leurs fonctions d'appartenance comme dans l'exemple <l'air est froid et le vent est fort> ou dans <si l'air est froid ou si le vent est fort> il faut fermer la porte. Il apparaît deux types de compositions, ET et OU auxquels il faut ajouter la négation. Notons x et y les variables linguistiques caractérisant la température de l'air et la force du vent et $\mu_A(x)$, $\mu_B(y)$, $\mu_E(z)$, $\mu_0(z)$, $\mu_C(z)$ avec $z = \{x, y\}$, les fonctions d'appartenance associées aux propriétés respectives <l'air est froid>, <le vent est fort>, <l'air est froid et le vent est fort>, <l'air est froid ou le vent est fort>, <l'air n'est pas froid>. <

2.4.1 Les opérateurs min-max

Opérateur NON

La propriété <l'air n'est pas froid>peut être caractérisée de façon évidente par la fonction d'appartenance

$$\mu_C(x) = 1 - \mu_A(x) \tag{2.4}$$

La figure 2.12 met en évidence cette relation.
A noter qu'il s'agit de l'opérateur NON, appelé aussi <complément>, <négation >ou <inverse>.

Opérateur ET

La solution la plus simple et la plus utilisée pour caractériser la satisfaction simultanée de deux propriétés est de poser

$$\mu_E(x) = \mu_{AetB}(x) = \min\{\mu_A(x), \mu_B(x)\}. \tag{2.5}$$

On parle alors d'opérateur minimum.
Cette opération est représentée à la figure 2.13. Comme on le voit, il est possible que la fonction d'appartenance résultante $\mu_E(x)$ n'atteigne pas la valeur 1.
On peut facilement vérifier que l'opérateur minimum est commutatif, c'est-à-dire qu'il est possible d'invertir $\mu_A(x)$ et $\mu_B(x)$ sans que le résultat change.

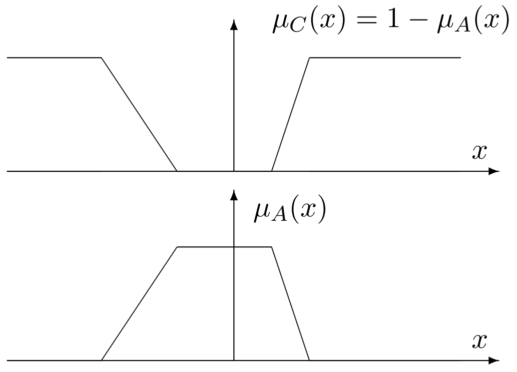


FIGURE 2.12 – Opérateur NON

Cet opérateur peut être appliqué à plus de deux ensembles. Dans ce cas s'applique le théorème d'associativité.

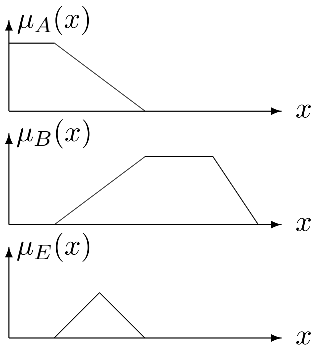


FIGURE 2.13 – Opérateur ET : $\mu_E(x) = \mu_A(x)$ ET $\mu_B(x)$

A noter qu'il s'agit de l'opérateur ET, aussi appelé <intersection>.

Opérateur OU

La réalisation de l'opérateur OU au niveau de la logique floue se fait en général par la formation du maximum, appliquée aux fonctions d'appartenance $\mu_A(x)$ et $\mu_B(x)$ des deux ensembles A et B. On a donc l'opérateur maximum.

$$\mu_O(x) = \mu_{AouB}(x) = \max\{\mu_A(x), \mu_B(x)\} \tag{2.6}$$

La figure 2.14 montre cette opération. A noter : l'opérateur maximum est aussi commutatif et associatif.
Dans ce paragraphe, on a introduit la formation du minimum et du maximum pour réaliser les opérateurs ET et OU. Dans la plupart des cas, ces opérateurs donnent des résultats convenables, surtout pour le réglage et la commande par logique floue.
Cependant, dans certaines circonstances, il peut être judicieux d'utiliser d'autres opérateurs, soit pour simplifier le traitement numérique, soit pour mieux tenir compte des opérations floues.

2.4.2 Opérateurs ET et OU, réalisés par opérateurs arithmétiques

Souvent, l'opérateur ET est réalisé par la formation du produit appliqué aux fonctions d'appartenance, selon la relation

$$\mu_E(x) = \mu_{AetB}(x) = \mu_A(x) \cdot \mu_B(x) \tag{2.7}$$

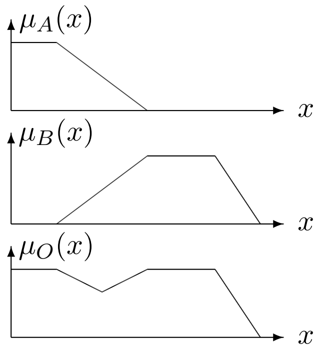


FIGURE 2.14 – Opérateur OU : $\mu_O(x) = \mu_A(x)$ OU $\mu_B(x)$

Il s'agit de l'opérateur produit. Le résultat de cette opération est représenté à la figure 2.15. La fonction d'appartenance résultante est toujours inférieure ou égale à 1. Elle reste donc à l'intérieur de l'intervalle défini par $\mu \in [0, 1]$. La règle de calcul 2.7 peut être étendue à plus de deux termes dans le produit lorsqu'il faut combiner trois ou plusieurs ensembles. L'opérateur produit est souvent utilisé dans le domaine de réglage et de commande par logique floue comme alternative à l'opérateur minimum.

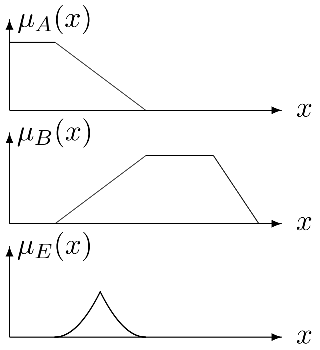


FIGURE 2.15 – Opérateur ET réalisé par la fonction produit

Par analogie, on peut réaliser l'opérateur OU par la formation de la somme des fonctions d'appartenance ou plus précisément par la valeur moyenne, à savoir :

$$\mu_O(x) = \mu_{AouB}(x) = 1/2[\mu_A(x) + \mu_B(x)] \tag{2.8}$$

On parle alors de l'opérateur somme.
La figure 2.16 montre l'effet de cet opérateur. La somme est divisée par 2. En effet, il est fort possible que la somme $[\mu A(x)+\mu B(x)]$ dépasse le domaine admissible [0,1]. Afin que cette somme reste dans le domaine défini, on peut l'écrêter ou la normaliser, comme effectué dans la définition 2.8. Dans ce cas aussi, il est possible d'étendre la règle de calcul 2.8 à plusieurs termes. Il faut alors diviser la somme par le nombre de termes, afin d'obtenir une normalisation simple.
Il est à noter que le logiciel MATLAB ne propose que le OU probabiliste qui est défini par la relation 2.9 :

$$\mu_O(x) = \mu_{AouB}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x) \tag{2.9}$$

Vous remarquerez sur la figure 2.17 que le résultat de ce OU arithmétique est assez proche du OU max.

2.4.3 Opérateurs ET flou et OU flou

Les opérateurs ET flou et OU flou sont des opérateurs combinés entre l'opérateur minimum et la moyenne arithmétique.

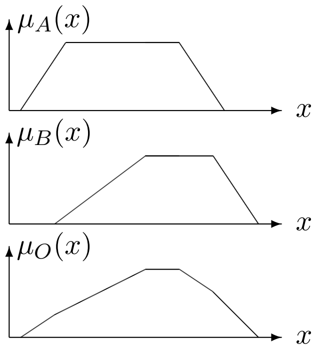


FIGURE 2.16 – Opérateur OU réalisé par la fonction somme

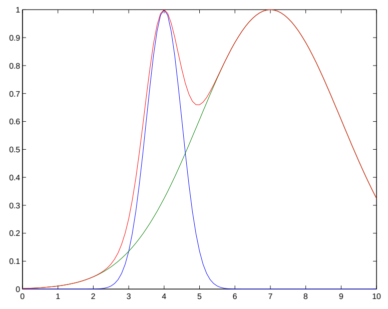


FIGURE 2.17 – Opérateur OU probabiliste

L'opérateur ET flou est défini par :

$$\mu_E(x) = \mu_{AetB}(x) = \beta \min[\mu_A(x), \mu_B(x)] + [(1-\beta)/2][\mu_A(x) + \mu_B(x)] \tag{2.10}$$

et l'opérateur OU flou par :

$$\mu_O(x) = \mu_{AouB}(x) = \beta \max[\mu_A(x), \mu_B(x)] + [(1-\beta)/2][\mu_A(x) + \mu_B(x)] \tag{2.11}$$

Avec le facteur $\beta \in [0,1]$, il est possible de pondérer l'influence des deux termes. Pour $\beta = 1$, on aboutit respectivement à l'opérateur minimal ou maximal. Par contre, pour $\beta = 0$, on obtient pour les deux opérateurs la moyenne arithmétique correspondant à l'opérateur somme selon 2.8. Dans ce cas, le ET flou et le OU flou se confondent. On peut étendre les deux opérateurs ET flou et le OU flou à trois ou à plusieurs termes. La somme qui apparaît entre crochets doit alors être divisée par le nombre de termes de la somme. La figure 2.18 représente l'opérateur ET flou et montre l'influence du facteur β sur l'allure de la fonction d'appartenance $\mu_E(x)$
L'influence du facteur β sur la fonction d'appartenance résultante pour l'opérateur OU flou est mise en évidence par la figure 2.19.

2.4.4 Opérateurs min-max et opérateur β

L'opérateur min-max est défini par la combinaison des opérateurs minimum et maximum, selon

$$\mu(x) = \beta \min[\mu_A(x), \mu_B(x)] + (1-\beta) \max[\mu_A(x), \mu_B(x)] \tag{2.12}$$

Le facteur $\beta \in [0,1]$, permet de pondérer les deux opérateurs. Pour $\beta = 1$, on obtient l'opérateur ET, réalisé par la formation du minimum, tandis que pour $\beta = 0$, on aboutit à l'opérateur OU, réalisé par la formation du maximum. Par contre, $\beta = 0,5$ conduit à l'opérateur OU, réalisé par la formation de la somme.

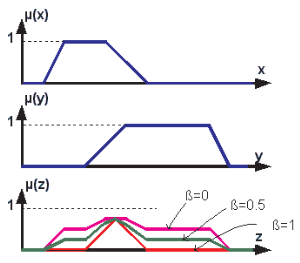


FIGURE 2.18 – Opérateur ET flou réalisé par la relation 2.10

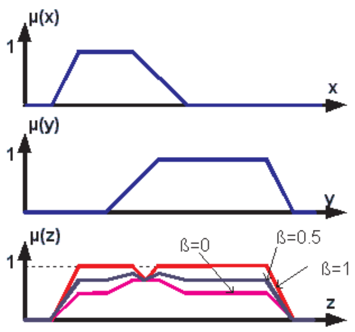


FIGURE 2.19 – Opérateur OU flou réalisé par la relation 2.11

La figure 2.20 montre l'effet de l'opérateur min-max en fonction du facteur β. On constate bien la grande variation de l'allure de la fonction d'appartenance résultante.

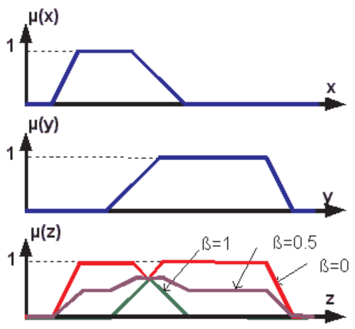


FIGURE 2.20 – Opérateur min-max réalisé par la relation 2.12

2.5 Méthodes d'inférence

Après la partie précédente, nous savons calculer les relations de type

– Si ε est N ET $\Delta\varepsilon$ est N
– Si (ε est NON N ET $\Delta\varepsilon$ est N) OU ε est P

Il reste donc à définir une opération pour traduire le terme ALORS ainsi qu'une opération pour définir l'interaction entre les règles.

2.5.1 Méthode d'inférence max-min

A ne pas confondre avec les opérateurs logiques min-max !
Avec la méthode d'inférence max-min le mot ALORS est représenté par l'opération minimum de la valeur de la règle et de la fonction d'appartenance considérée de la sortie. Ainsi la règle : Si ε est P ET $\Delta\varepsilon$ est Z ALORS u est P est traduite par la surface hachurée de la figure 2.21 qui représente la fonction d'appartenance partielle.

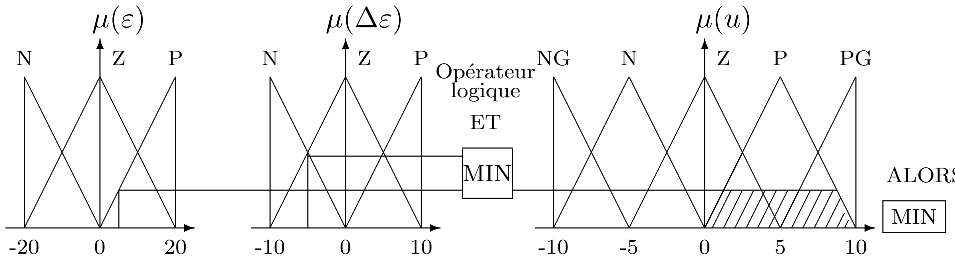


FIGURE 2.21 – Définition de ALORS dans la méthode d'inférence max-min

Par ailleurs l'interaction entre les règles sera nommée OU ainsi, on considère que les deux règles suivantes

– Si ε est Z ET $\Delta\varepsilon$ est N ALORS u est N
– Si ε est P ET $\Delta\varepsilon$ est Z ALORS u est P

se lisent :
– Si ε est Z ET $\Delta\varepsilon$ est N ALORS u est N OU
– Si ε est P ET $\Delta\varepsilon$ est Z ALORS u est P

et ce OU se traduit par l'opération max. Nous obtenons ainsi la fonction d'appartenance résultante, la partie 2.6 traitera de la transformation de cette fonction d'appartenance résultante en une commande.

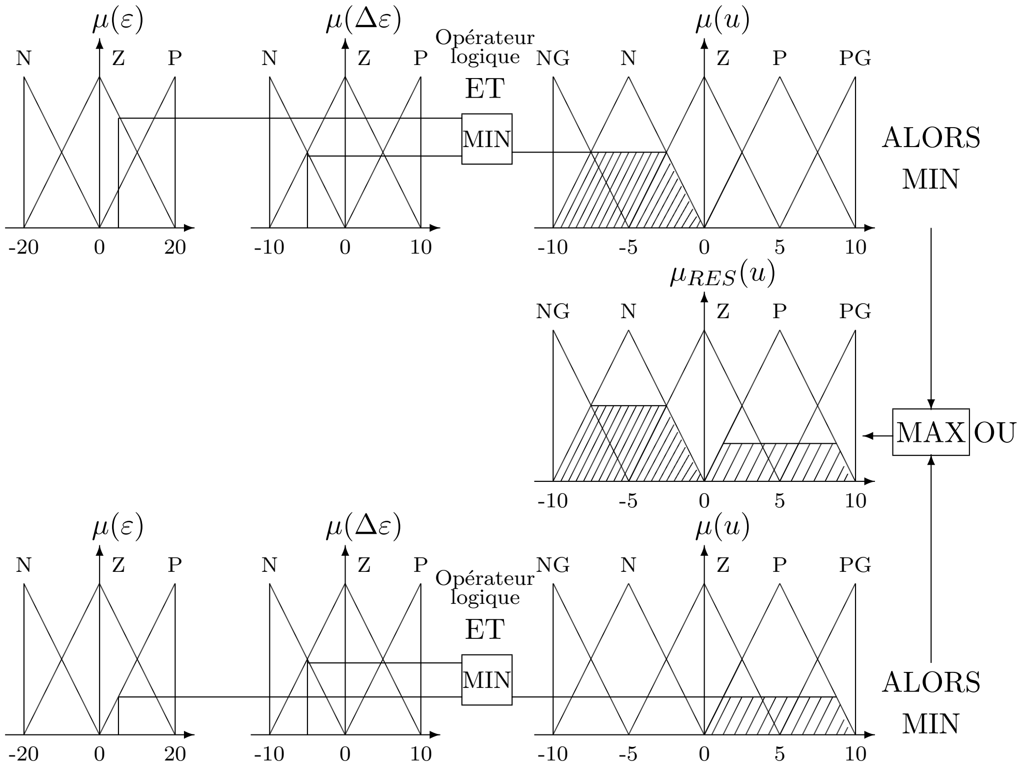


FIGURE 2.22 – Définition de OU dans la méthode d'inférence max-min

L'application de l'ensemble des règles donne la fonction d'appartenance partielle représentée sur la figure 2.23

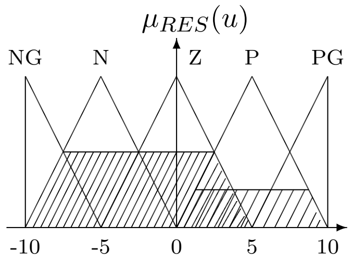


FIGURE 2.23 – Fonction d'appartenance résultante

2.5.2 Méthode d'inférence max-prod

La différence avec la méthode précédente est la réalisation de ALORS qui se traduit par la multiplication de la fonction d'appartenance considérée par la valeur de la règle. Ce résultat est illustré sur la figure 2.24.

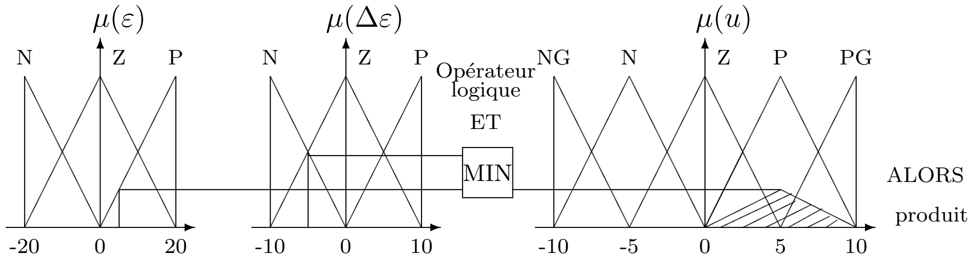


FIGURE 2.24 – Définition de ALORS dans la méthode d'inférence max-prod

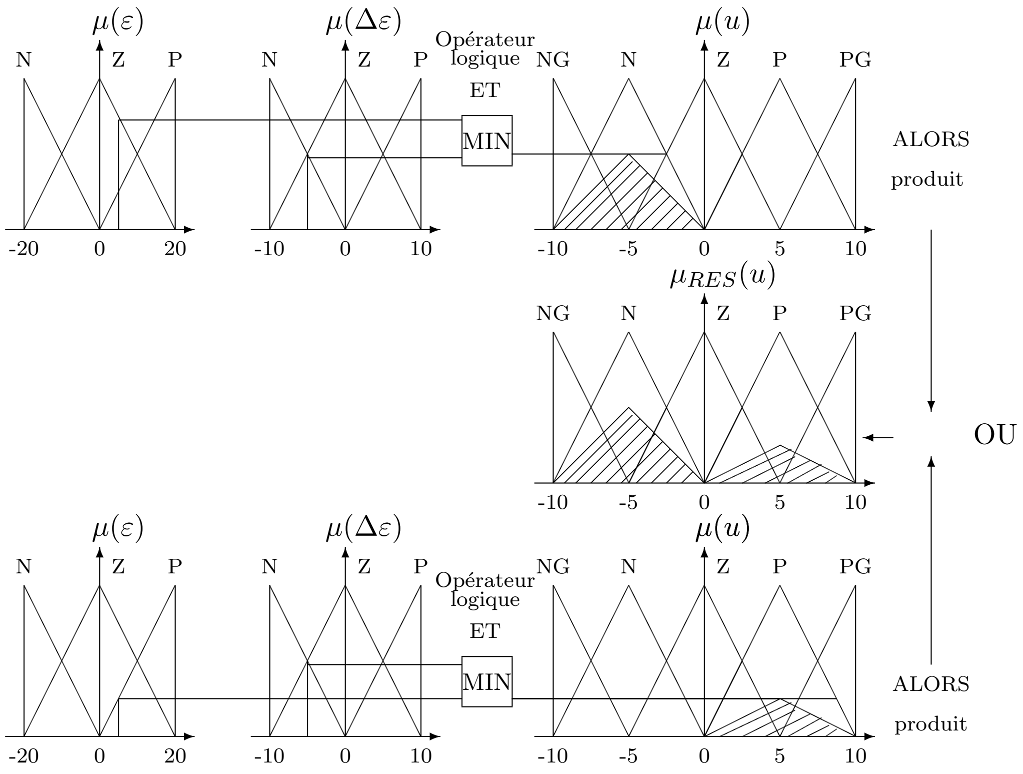


FIGURE 2.25 – Définition de OU dans la méthode d'inférence max-prod

2.5.3 Méthode d'inférence somme-prod

Proche de la méthode précédente, il suffit de remplacer la traduction de ALORS par la somme des fonctions d'appartenance partielles.

2.5.4 Critères de choix d'une méthode d'inférence

On utilisera en général la méthode max-min, celle-ci provoque une commande avec des sauts mais ce n'est en général pas ou peu contraignant. Il est à noter que la méthode somme-prod permet de développer un calcul analytique de la sortie permettant, si l'on possède un modèle du système, de calculer les caractéristiques temporelles de la sortie.

2.6 Méthodes de défuzzification

Comme on l'a vu à la section précédente, les méthodes d'inférence fournissent une fonction d'appartenance résultante $\mu_{RES}(x)$ pour la variable de sortie x. Il s'agit donc d'une information floue. Les actionneurs actuels, utilisés dans les boucles de commande ne s'accompagnent pas de ce genre de décision, il convient de la transformer en une grandeur de commande précise : c'est le but de l'étape de défuzzification. Les méthodes les plus couramment utilisées sont :

– La méthode du centre de gravité
– La méthode de la moyenne des maximums

2.6.1 Méthode du centre de gravité

La méthode la plus utilisée est de loin la méthode dite du centre de gravité. Il s'agit de calculer la position du centre de gravité de la fonction d'appartenance résultante, l'abscisse de ce centre de gravité devient la sortie du régulateur et donc la commande du système.

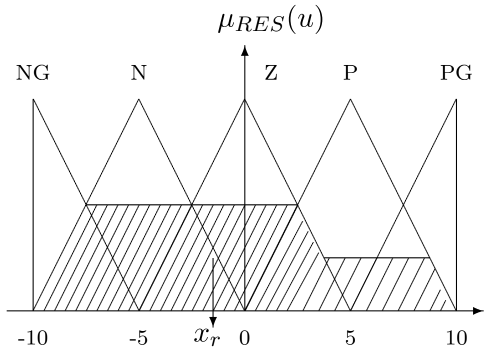


FIGURE 2.26 – Défuzzification par la méthode du centre de gravité.

Ce centre de gravité est relativement facile à calculer lorsque la fonction d'appartenance résultante est constituée de morceaux de droites. Dans les autres cas, ce calcul peut être gourmand en temps de calcul et donc pénalisant lors de l'implantation sur un microprocesseur. C'est l'une des raison qui font préférer les fonction d'appartenance de formes trapézoïdales (triangulaires) et les opérateurs logiques de type min-max.
Note : les valeurs extrêmes des sorties atteignables sont les centres de gravité des fonctions d'appartenance extrêmes des sorties. Aussi, et afin de minimiser le temps d'établissement, il faut choisir les fonctions d'appartenance des sorties maximales tels que leur centre de gravité corresponde aux valeurs maximales de commande.

2.6.2 Méthode de la moyenne des maximums

Celle-ci dérive d'une autre méthode encore plus simple, la méthode du maximum. Pour cette dernière, la valeur de la sortie est simplement l'abscisse du maximum de la fonction d'appartenance résultante. Si cette fonction présente plusieurs fois le même maximum, il y a indécision.
La méthode de la moyenne des maximums lève cette ambiguïté en prenant la moyenne de ces maximums (voir fig. 2.27(a)). Rapide à calculer, elle présente néanmoins un inconvénient majeur en effet, la valeur de la sortie peut présenter des sauts très importants pour des variations d'entrées très faibles comme illustré sur la figure 2.27(b). D'autres méthodes existent, elles sont illustrées sur la figure 2.28.

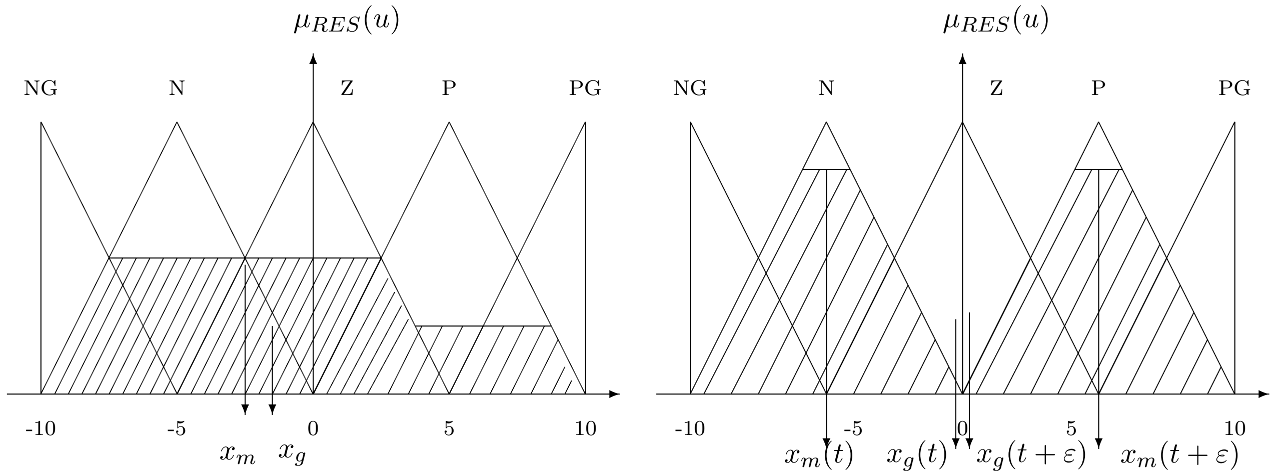


FIGURE 2.27 – Comparaison des méthodes de défuzzification et sauts de commande par la méthode moyenne des maximums

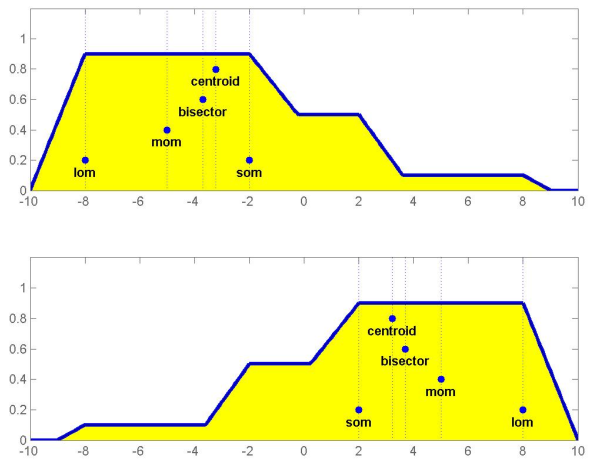


FIGURE 2.28 – Comparaison de différentes méthodes de défuzification.

2.7 Implémentation

Les différents concepts de la commande floue étant acquis, il s'agit maintenant d'appliquer la commande floue. En premier lieu, la figure 2.31 montre un "algorithme" de synthèse d'une commande floue.
Notez qu'il n'existe pas de méthode générale de synthèse.

2.7.1 Mise en œuvre

La mise en œuvre de la commande floue s'effectue à l'aide d'un calculateur, le plus souvent un microcontrôleur. Vous trouverez sans difficulté des bibliothèques de programmes en C qui vous aiderons à l'implémenter, par ailleurs, certains microcontrôleurs, comme le 68HC12 de Motorola possèdent des instructions permettant de faire de la commande floue directement en assembleur.
La commande floue est une commande quasi-continue, cela veut dire que la fréquence d'échantillonnage doit être très grande vis-à-vis de la fréquence maximum du système à contrôler.

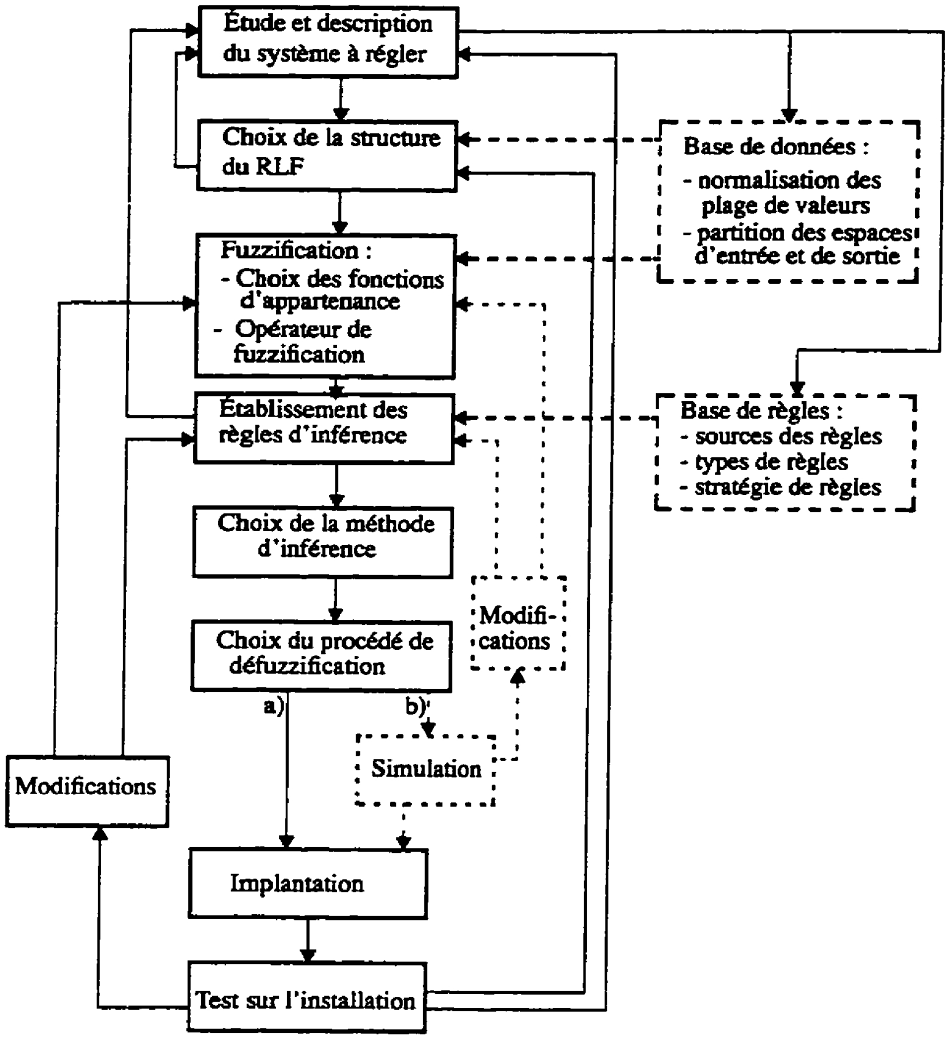

Étude et description du système à régler
Choix de la structure du RLF
Fuzzification :
- Choix des fonctions d'appartenance
- Opérateur de fuzzification
Établissement des règles d'inférence
Choix de la méthode d'inférence
Choix du procédé de défuzzification
a)
b)
Modifi-cations
Simulation
Modifications
Implantation
Test sur l'installation
Base de données :
- normalisation des plage de valeurs
- partition des espaces d'entrée et de sortie
Base de règles :
- sources des règles
- types de règles
- stratégie de règles

FIGURE 2.29 – Methodologie de la synthèse d'une commande floue.

2.8 Conclusions

Tous comptes faits, qu'avons nous réalisé jusqu'à présent, un régulateur PD non linéaire ! La figure 2.32 montre cette surface de commande en fonctions des entrées.
Ce polycopié ne se veut être qu'un brève introduction à la commande par un régulateur flou. Il existe pour chaque partie de ce cours une abondante bibliographie pour approfondir chaque détail de la logique floue et son application à la commande des procédés. Dans un premier temps il serai bon de lire les références suivantes : [2] [1] [7] [4] [3]. Vous remarquerez d'ailleurs une étrange ressemblance entre ce polycopié et le livre de professeur Bühler ! ? !

Sous son apparente facilité de mise en oeuvre, vous remarquerez rapidement que le nombre de paramètres à régler est très important, aussi, pour peu que vous n'ayez pas de modèle et/ou que le procédé soit lent, le temps d'implantation peut être très long. En comparaison un PID demande 3 paramètres et un modèle ! ! ! Une approche de la commande floue est l'extension non-linéaire d'un correcteur existant. Dans ce cas, une première identification du correcteur existant sera nécessaire (voir partie Identification).

Aussi, ayez à l'esprit que ce type de régulateur ne se justifie que si le système présente de fortes

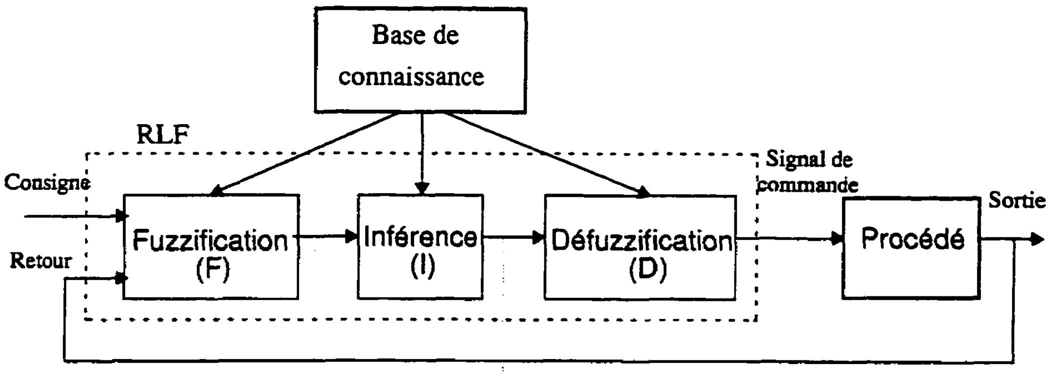


FIGURE 2.30 – Principe général d'implémentation d'une commande floue.

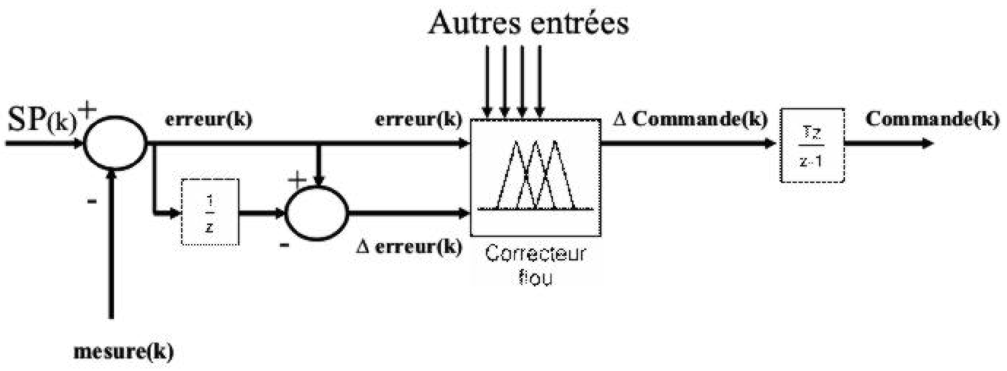


FIGURE 2.31 – Implémentation classique d'une commande floue de type PD non linéaire. Notez la présence de l'intégrateur pur en sortie.

non-linéarités et/ou des paramètres variant fortement. En tout état de cause, si vous possédez un modèle la comparaison avec d'autres approches de commande robuste s'impose.

2.9 Un peu de vocabulaire

Ensembles flous "jeune", "entre deux âges" et "âgé" sont les valeurs de la variable linguistique "âge"

Fonction d'appartenance c'est $\mu_{\text{jeune}}(\text{âge})$

Fonction d'appartenance résultante ?

Inférences ce sont les déductions floues c'est à dire une ou plusieurs règles.

Méthode d'inférence comment prendre en compte toutes les règles simultanément

Prédicats ou classes d'appartenance : grand, petit moyen chaud ...

Prémisses c'est la première partie d'une règle.

Règles ou règles d'inférence par exemple

$$\underbrace{\text{Si}\varepsilon \text{ est Z ET } \Delta\varepsilon \text{ est N}}_{\text{prémisse}}\underbrace{\text{ALORS u est N}}_{\text{conséquence}}$$

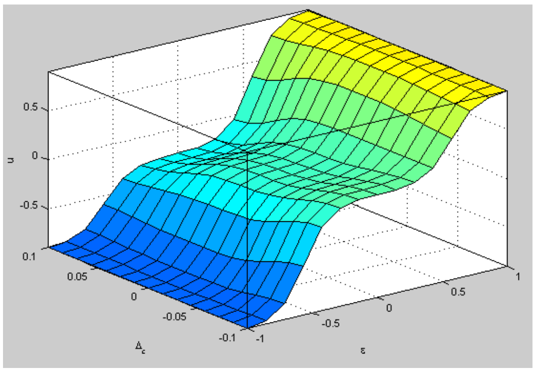


FIGURE 2.32 – Surface de commande d'un contrôleur à deux entrées.

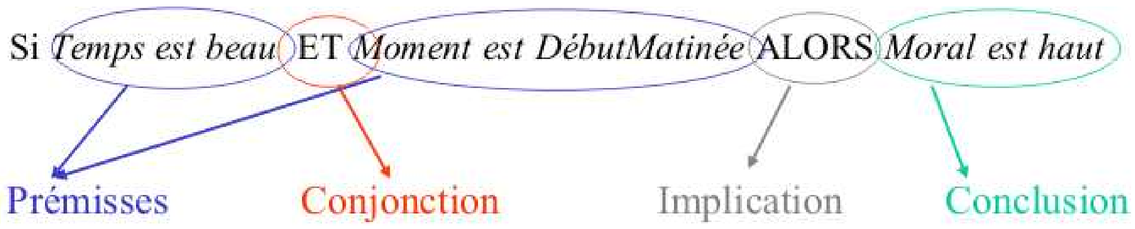

Si *Temps est beau* ET *Moment est DébutMatinée* ALORS *Moral est haut*
Prémisses
Conjonction
Implication
Conclusion

FIGURE 2.33 – Inférence floue

Univers de discours domaine de variation de la variable considérée : par exemple "jeune", "entre deux âges" et "âgé"

Variables linguistiques c'est "âge"

Deuxième partie

Identification floue

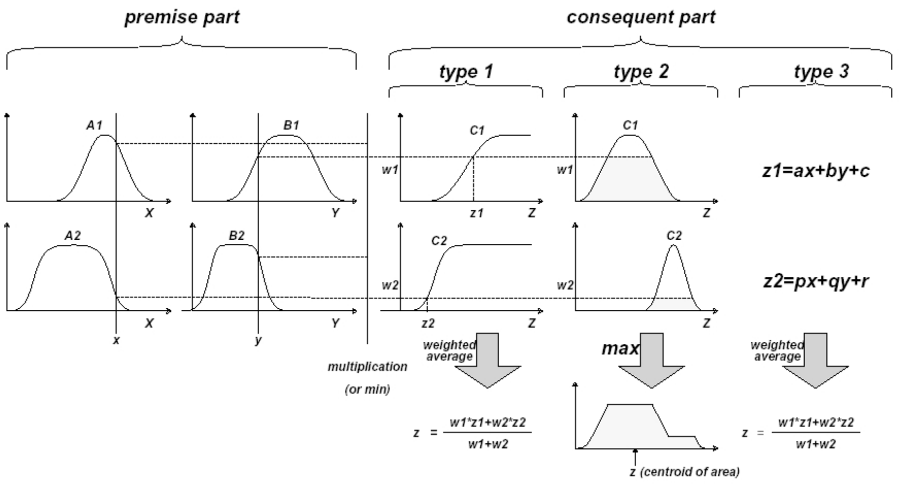


FIGURE 2.34 – Le type 1 a été initialement proposé par Tsukamoto, le type 2 est typiquement la forme de Mandani, le type 3 est l'approche de Takagi-Sugeno.

2.10 Résumé de l'identification floue

L'identification floue est un domaine extrêmement vaste, où chaque jour ou presque, une nouvelle approche est proposée. Certaines approches confinent à la théorie des réseaux de neurones, celle présentée ci-après en fait partie. Ce polycopié n'a pour ambition que de vous présenter l'approche générale de l'identification floue et de vous fournir un outil de base, ANFIS. Cet outil, gratuit, est disponible sur Matlab [1], sur Scilab et vous trouverez sans peine des codes sources en C++.

2.10.1 La commande floue au sens de Sugeno

La commande floue au sens de Sugeno diffère de la commande au sens de Mandani sur la partie défuzzification. En effet, Sugeno propose de remplacer la défuzzification des variables de sortie par une combinaison linéaire des variables d'entrée [2].
Une interprétation de la commande floue au sens de Sugeno est la suivante : sur la figure 2.35 imaginez que x représente l'erreur et y l'intégrale de l'erreur. Alors si $r_1 = 0$, f_1 représente un correcteur de type PI avec $k_p = p_1$ et $k_i = q_1$. Avec la même interprétation pour f_2, la commande f est alors un compromis entre deux correcteurs PI, agissant plus ou moins en fonction de l'erreur et de l'intégrale de l'erreur. Typiquement on pourrai choisir :
- f_1 régulateur PI en régulation agissant lorsque l'erreur x est petite.
- f_2 correcteur PI en poursuite agissant lorsque l'erreur x est grande.

L'idée n'est pas nouvelle, mais la commutation entre plusieurs correcteurs est très difficile, en particulier la prise en compte de conditions initiales non nulles. La commande floue est alors une façon propre de changer de correcteur en fonction de l'état du système. Notez bien que rien ne vous empêche d'étendre le raisonnement à plus de deux correcteurs, pas forcément de type PID ...
L'avantage de la structure de Sugeno est avant tout d'être assez intuitive pour l'automaticien, par la facilité d'implanter des correcteur linéaires de type PID. Par ailleurs, cette structure permet une mise en équation nettement plus simple que la structure de Mandani, il est en particulier plus facile de déterminer les relations entrées-sorties. Enfin dernier point, le plus important pour la suite de ce cours, la relation entre les variations de paramètres et les variations des sorties est continue, ce qui dans l'optique d'une optimisation des paramètres de la structure est un énorme avantage puisqu'il est nettement plus facile de déterminer un optimum de fonction continue que celui d'une fonction

1. Dans ce cas, il devient subrepticement payant !
2. Le plus souvent, cette combinaison linéaire se réduit à une constante ($Z_1 = k$), mais combinaisons plus complexes (parabolique, non linéaire ...) ont aussi été testées.

présentant des discontinuités.

2.10.2 L'identification floue

En observant la surface de commande 2.32, il vient à l'esprit de procéder à l'envers. Est-il possible, à partir de la surface de commande voulue, d'obtenir les fonctions d'appartenance, les règles, ... ?
Le problème est un problème d'optimisation non-linéaire. En effet, il "suffit" de déterminer les paramètres du régulateur (règles, forme des fonctions d'appartenance, position des fonctions d'appartenance, nombre de prémisses, coefficients p_i, q_i et r_i, ... ce qui, en pratique, est difficile à obtenir. Quelques auteurs s'y essaient notamment à l'aide d'algorithmes génétiques, puissants mais dont la convergence, lente, n'est pas garantie.
En imposant la forme générale du système, le nombre d'entrées, le nombre de fonctions d'appartenance et leurs formes, il est possible d'obtenir une convergence très rapide notamment par rétropropagation du gradient[3] vers un système reproduisant correctement la surface voulue. Bien entendu, le mot surface s'entend dans un cadre pluri-dimensionnel.

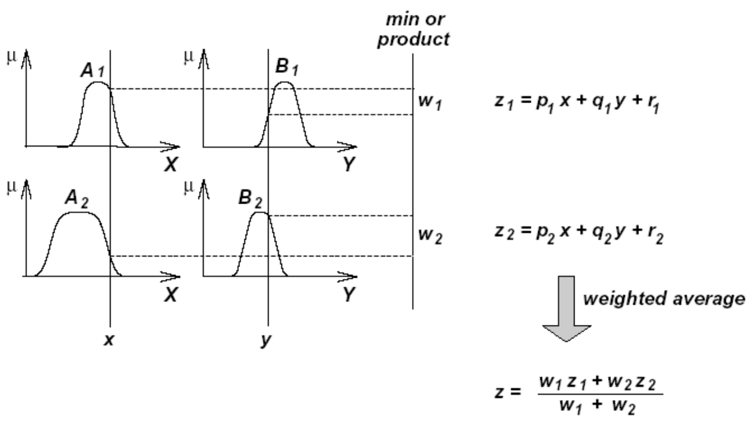


FIGURE 2.35 – La commande floue au sens de Takagi-Sugeno

.

$$\begin{array}{lll} IF(xisA_1)AND(yisB_1) & THEN & f_1 = p_1x + q_1y + r_1 \\ IF(xisA_2)AND(yisB_2) & THEN & f_2 = p_2x + q_2y + r_2 \end{array}$$

Les figures 2.36 et 2.37 montrent graphiquement la similitude entre réseaux de neurones et identification floue.
L'idée générale d'ANFIS est d'intégrer dans un seul concept les avantages des deux domaines : Logique floue et Réseaux de neurones.

- De la logique floue :
  Introduction de connaissances *a priori* afin de réduire l'espace des paramètres à optimiser
- Des réseaux de neurones :
  Utilisation de la méthode d'optimisation "back propagation"

Comme expliqué précédemment, l'identification de systèmes à l'aide de ANFIS (Adaptative Network Based Fuzzy Inference System) de Matlab s'effectue en suivant l'organigramme présenté en figure 2.38.
La première étape (*genfis*) consiste à imposer la forme du régulateur, la seconde, à optimiser les paramètres de ce régulateur vis-à-vis de l'erreur de prédiction.

3. voir cours réseaux de neurones.

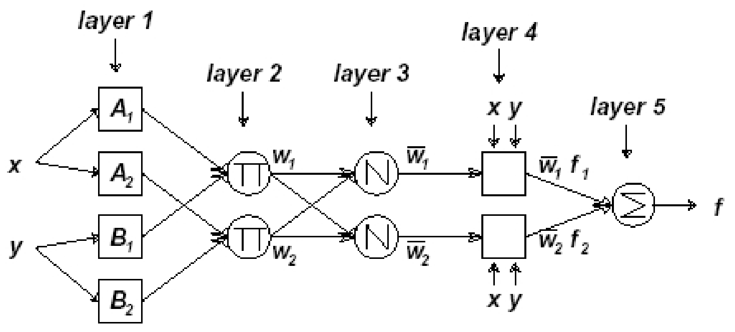


FIGURE 2.36 – Equivalent ANFIS de la figure 2.35

.

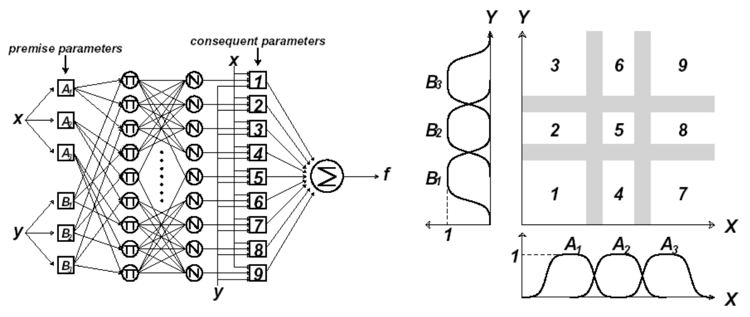


FIGURE 2.37 – Approche neuro-floue de la commande au sens de Takagi-Sugeno.

Utilité de l'identification floue

La première utilisation est bien entendu d'obtenir un modèle implantable dans un calculateur d'un système non linéaire. L'interprétation de ce modèle est encore soumise à caution.
La synthèse d'un régulateur, flou ou pas, directement à partir d'une identification floue, fait partie des thèmes de la recherche scientifique actuelle. Pour l'instant aucune méthodologie claire ne se dégage.
Une autre application est assez courante, il s'agit de l'identification de correcteurs plus classiques. En effet, la synthèse de correcteurs flous est assez "floue". Aussi, certains auteurs préconisent d'identifier un correcteur classique par un système flou puis d'étendre ce correcteur flou par de nouvelles fonctions d'appartenance et de nouvelles règles.
La synthèse d'un correcteur flou suit alors l'algorithme suivant

1. Synthèse d'un correcteur linéaire autour du point de fonctionnement,
2. identification floue de ce correcteur,
3. ajout de règles et/ou de prémisses pour étendre le domaine de "bon" fonctionnement.

Une autre méthode donnant de bons résultats, consiste à simuler le processus à commander précédé d'un correcteur flou, puis à optimiser les paramètres de ce correcteur flou en fonction de performances attendues en boucle fermée.

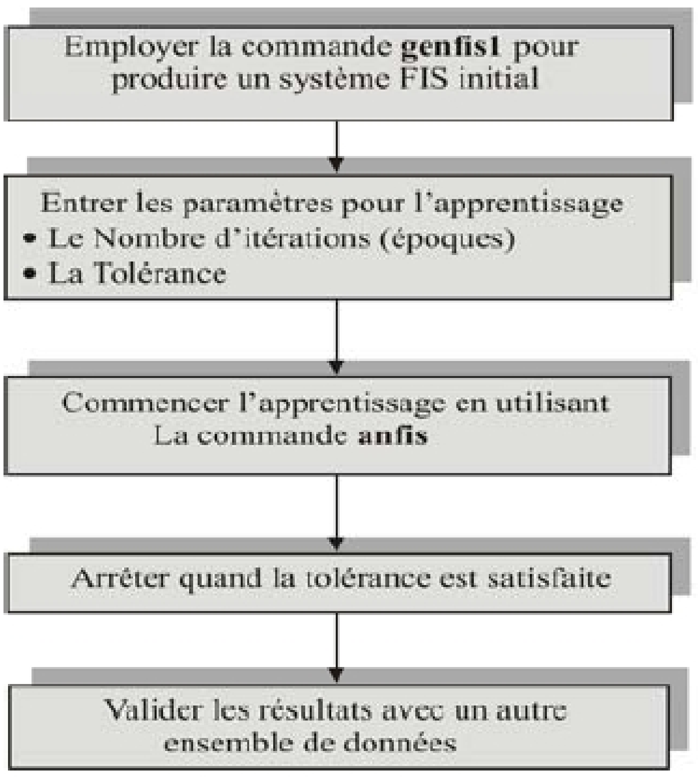

Employer la commande **genfis1** pour produire un système FIS initial
Entrer les paramètres pour l'apprentissage
• Le Nombre d'itérations (époques)
• La Tolérance
Commencer l'apprentissage en utilisant La commande **anfis**
Arrêter quand la tolérance est satisfaite
Valider les résultats avec un autre ensemble de données

FIGURE 2.38 – Etapes de synthèse d'une identification floue sous ANFIS de Matlab.

Chapitre 3

Applications - TDAO

3.1 Contrôle de niveau

L'objectif dans cette partie est de contrôler le niveau d'eau dans le réservoir représenté ci-dessous. Celui-ci est alimenté par une vanne réglable et il se vide continûment par un tuyau . On peut changer le débit de l'eau entrant dans le réservoir, mais l'écoulement dépend du diamètre du tuyaux de sortie (lequel est constant) et de la pression dans le réservoir (qui varie avec le niveau d'eau !). On remarque que le système a des caractéristiques non linéaires.

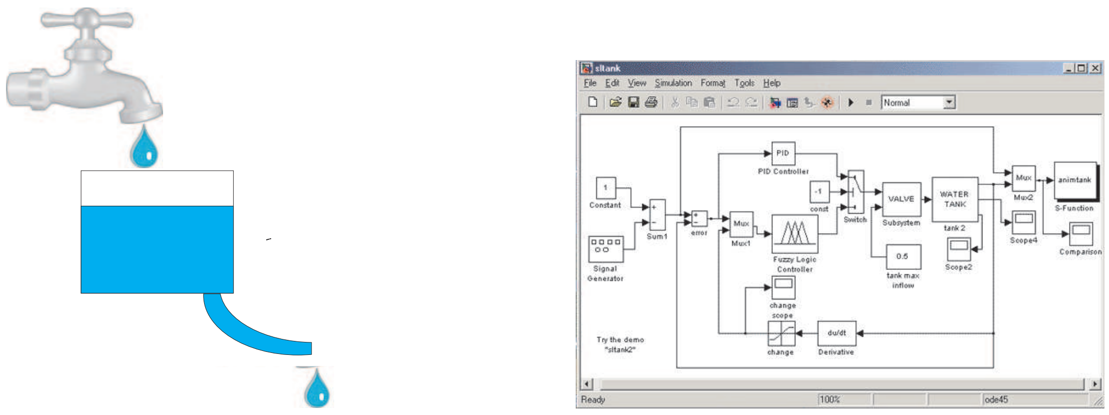


FIGURE 3.1 – Exemple : contrôle de niveau

L'entrée de notre contrôleur sera l'erreur sur le niveau (consigne - sortie) et sa sortie est un signal précisant le degré d'ouverture ou de fermeture de la vanne. Une première réflexion aboutit à un régulateur flou fonctionnant avec les règles suivantes :

– si (le niveau est bon) ALORS (Ne pas changer la vanne)
– si (le niveau est bas) ALORS (Ouvrir la vanne)
– si (le niveau est haut) ALORS (Fermer la vanne)

La question que l'on se pose est : ces trois règles sont-elles suffisantes pour réguler un tel système ? La solution est de simuler ce système ainsi que le régulateur flou associé.

En franglais cela donne les règles suivantes :

– Si (le niveau est bon (okay)) ALORS (la vanne n'aura pas de changement (no_change)).
– Si (le niveau est bas (low)) ALORS (on ouvrirait la vanne rapidement (open_fast)).
– Si (le niveau est haut (high)) ALORS (on fermerait la vanne rapidement (close_fast)).

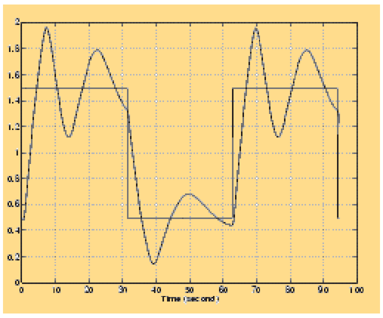


FIGURE 3.2 – Niveau en fonction du temps avec trois règles.

La simulation montre que trois règles ne suffisent pas pour la régulation ; le niveau d'eau tend à osciller autour de la consigne ce qui est illustré par la figure 3.2.
On a donc besoin d'une autre entrée pour notre régulateur, c'est la variation du niveau de l'eau (exprimée par sa dérivée) et ceci pour ralentir le mouvement de la vanne quand on est proche du niveau imposé par la consigne. On ajoutera alors à notre système d'inférence les règles suivantes :

– Si (le niveau est bon (okay)) ET (sa dérivée est positive (positive))
  ALORS (la vanne serait fermée lentement (close_slow)).
– Si (le niveau est bon (okay)) ET (sa dérivée est négative (negative))
  ALORS (la vanne serait ouverte lentement (open_slow)).

Le résultat est alors : le système atteint la consigne sans dépassement comme illustré sur la figure 3.3.

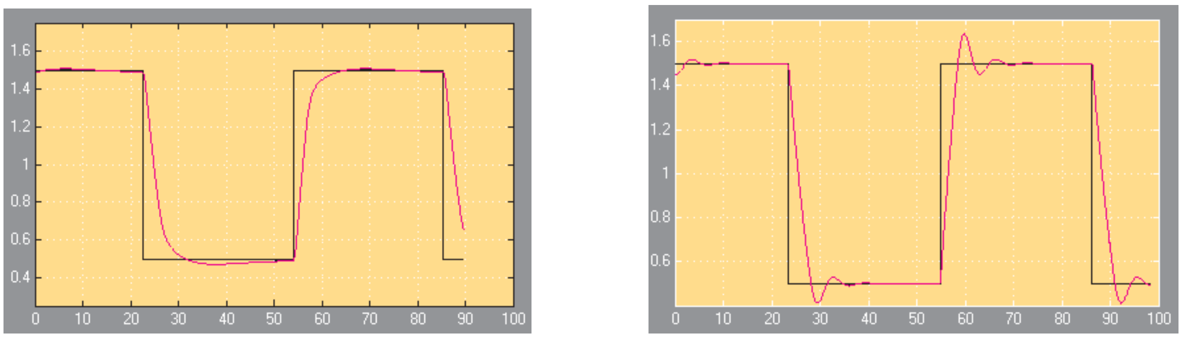
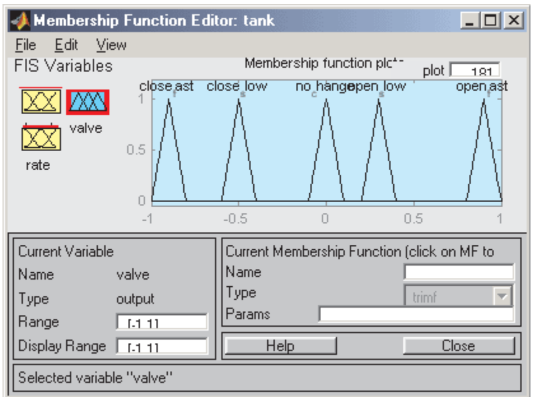


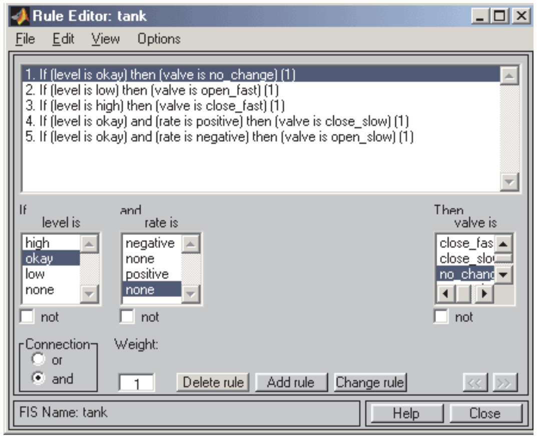
FIGURE 3.3 – Niveau en fonction du temps avec deux entrées (à droite : PID, à gauche : Régulateur flou).

Une caractéristique importante de ce système est le fait que ce réservoir met beaucoup plus de temps à se vider qu'à se remplir et ceci à cause du petit diamètre de la conduite de sortie. On peut y remédier en choisissant une fonction d'appartenance close_slow légèrement différente de celle open_slow $\Longrightarrow$ asymétrie de la sortie vanne ; un régulateur PID n'offre pas cette possibilité.

3.1.1 Le fichier de démonstration "water tank"

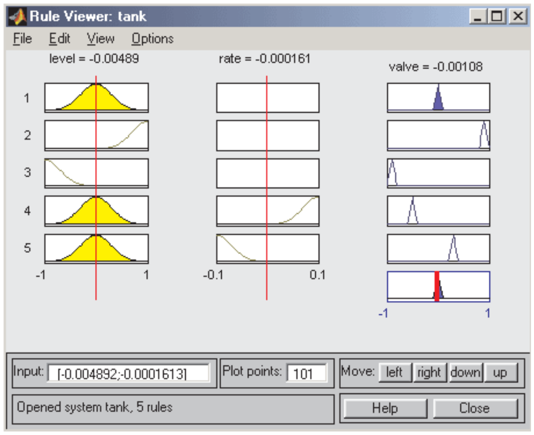
Ouvrez-le dans :
help $\rightarrow$ Examples and demo $\rightarrow$ Toolboxes $\rightarrow$ Fuzzy logic $\rightarrow$ water tank with rules viewer.
et sauvegardez-le dans un autre répertoire.
Le menu "view" de "Rules viewer" permet de passer d'une interface graphique à une autre.
IMPORTANT : dans le "Rules viewer" cliquez sur fichier $\rightarrow$ "save to disk as" et mettez le fichier tank.fis dans votre répertoire. Ensuite cliquez sur "save to "workspace". Ce dernier point sera a refaire à chaque changement de règles, de fonction d'appartenance ...


Membership Function Editor: tank
File Edit View
FIS Variables
level
valve
rate
Membership function plots
plot points: 181
close_fast close_low no_change open_low open_fast
Current Variable
Name valve
Type output
Range [-1 1]
Display Range [-1 1]
Current Membership Function (click on MF to select)
Name
Type trimf
Params
Help
Close
Selected variable "valve"



1. If (level is okay) then (valve is no_change) (1)
2. If (level is low) then (valve is open_fast) (1)
3. If (level is high) then (valve is close_fast) (1)
4. If (level is okay) and (rate is positive) then (valve is close_slow) (1)
5. If (level is okay) and (rate is negative) then (valve is open_slow) (1)

FIGURE 3.4 – Interface graphique : à gauche : éditeur des fonction d'appartenance, à droite : éditeur de règles.



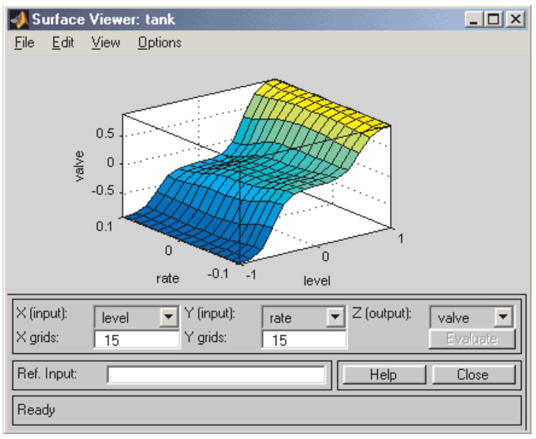


FIGURE 3.5 – Interface graphique : à gauche : effet des règles, à droite : surface de commande.

3.1.2 A faire

1. Simulez les correcteurs précédents.
2. Observez les variations de comportement en fonction de la définition des ensembles flous, notamment la forme choisie et la mise en œuvre des méthodes d'inférence .
3. En augmentant le débit d'entrée, essayez d'obtenir un comportement plus rapide quitte à avoir un léger dépassement.
4. en changeant significativement les paramètres du modèle, observez la robustesse de cette méthode de correction
5. Transformez le correcteur flou afin d'obtenir une erreur statique nulle.

3.2 Identification floue

Plus qu'un TD, il s'agit avant tout d'observer la puissance de l'identification non-linéaire par la logique floue.

1. Ouvrez le fichier "`init_identification_floue.mdl`"
2. Exécutez le fichier "`ident_floue.m`"
3. Vérifiez le fruit de votre identification à l'aide du fichier "`verif_identification_floue.mdl`", en cas de besoin pensez à changer les entrées du modèle flou.

4. Observez les différents paramètres du modèle, en particulier sa ”surface”.
5. Comprenez le fichier et changer les paramètres d’identification, en particulier la forme des fonctions d’appartenance (voir figure 2.5) et leur nombre.

1

ANFIS:

Adaptive-Network-Based Fuzzy Inference System

Jyh-Shing Roger Jang
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

This paper presents the architecture and learning procedure underlying ANFIS (Adaptive-Network-based Fuzzy Inference System), a fuzzy inference system implemented in the framework of adaptive networks. By using a hybrid learning procedure, the proposed ANFIS can construct an input-output mapping based on both human knowledge (in the form of fuzzy if-then rules) and stipulated input-output data pairs. In our simulation, we employ the ANFIS architecture to model nonlinear functions, identify nonlinear components on-linely in a control system, and predict a chaotic time series, all yielding remarkable results. Comparisons with artificail neural networks and earlier work on fuzzy modeling are listed and discussed. Other extensions of the proposed ANFIS and promising applications to automatic control and signal processing are also suggested.

1 Introduction

System modeling based on conventional mathematical tools (e.g., differential equations) is not well suited for dealing with ill-defined and uncertain systems. By contrast, a fuzzy inference system employing fuzzy if-then rules can model the qualitative aspects of human knowledge and

[*]Research supported in part by NASA Grant NCC-2-275; LLNL Grant No. ISCR 89-12; MICRO State Program Award No. 90-191; and OMRON Corporation.

reasoning processes without employing precise quantitative analyses. This *fuzzy modeling* or *fuzzy identification*, first explored systematically by Takagi and Sugeno [50], has found numerous practical applications in control [42, 34], prediction and inference [16, 17]. However, there are some basic aspects of this approach which are in need of better understanding. More specifically:

1. No standard methods exist for transforming human knowledge or experience into the rule base and database of a fuzzy inference system.

2. There is a need for effective methods for tuning the membership functions (MF's) so as to minimize the output error measure or maximize performance index.

In this perspective, the aim of this paper is to suggest a novel architecture called *Adaptive-Network-Based Fuzzy Inference System*, or simply *ANFIS*, which can serve as a basis for constructing a set of fuzzy if-then rules with appropriate membership functions to generate the stipulated input-output pairs. The next section introduces the basics of fuzzy if-then rules and fuzzy inference systems. Section 3 describes the structures and learning rules of adaptive networks. By embedding the fuzzy inference system into the framework of adaptive networks, we obtain the ANFIS architecture which is the backbone of this paper and it is covered in section 4. Application examples such as nonlinear function modeling and chaotic time series prediction are given in section 5. Section 6 concludes this paper by giving important extensions and future directions of this work.

2 Fuzzy If-Then Rules and Fuzzy Inference Systems

A. Fuzzy If-Then Rules

Fuzzy if-then rules or *fuzzy conditional statements* are expressions of the form *IF A THEN B*, where A and B are labels of *fuzzy sets* [62] characterized by appropriate membership functions. Due to their concise form, fuzzy if-then rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision. An example that describes a simple fact is

If pressure is high, then volume is small.

where *pressure* and *volume* are *linguistic variables* [63], *high* and *small* are *linguistic values* or *labels* that are characterized by membership functions.

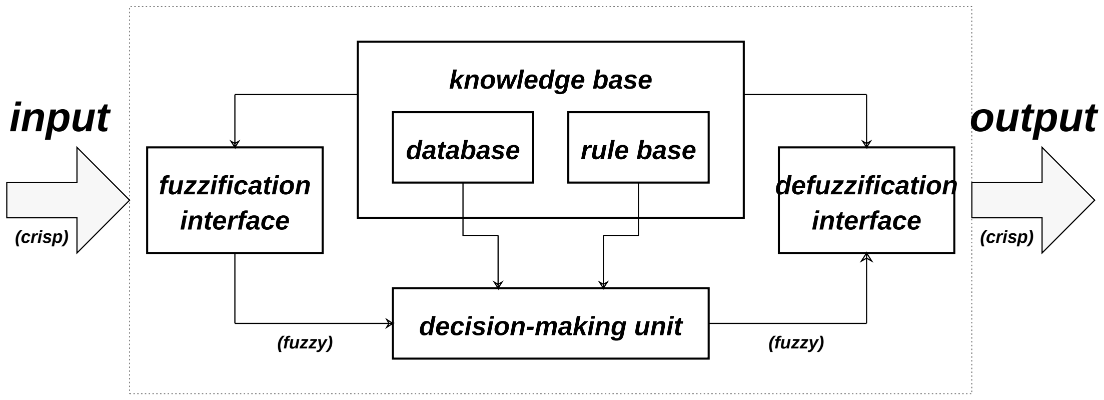


Figure 1: *Fuzzy inference system.*

Another form of fuzzy if-then rule, proposed by Takagi and Sugeno [49], has fuzzy sets involved only in the premise part. By using Takagi and Sugeno's fuzzy if-then rule, we can describe the resistant force on a moving object as follows:

$$\textit{If velocity is high, then } force \ = \ k * (velocity)^2.$$

where, again, *high* in the premise part is a linguistic label characterized by an appropriate membership function. However, the consequent part is described by a nonfuzzy equation of the input variable, velocity.

Both types of fuzzy if-then rules have been used extensively in both modeling and control. Through the use of linguistic labels and membership functions, a fuzzy if-then rule can easily capture the spirit of a "rule of thumb" used by humans. From another angle, due to the qualifiers on the premise parts, each fuzzy if-then rule can be viewed as a local description of the system under consideration. Fuzzy if-then rules form a core part of the fuzzy inference system to be introduced below.

A. Fuzzy Inference Systems

Fuzzy inference systems are also known as *fuzzy-rule-based systems*, *fuzzy models*, *fuzzy associative memories (FAM)*, or *fuzzy controllers* when used as controllers. Basically a fuzzy inference system is composed of five functional blocks (Figure 1):

- a **rule base** containing a number of fuzzy if-then rules;
- a **database** which defines the membership functions of the fuzzy sets used in the fuzzy rules;

- a **decision-making unit** which performs the inference operations on the rules;
- a **fuzzification interface** which transforms the crisp inputs into degrees of match with linguistic values;
- a **defuzzification interface** which transform the fuzzy results of the inference into a crisp output.

Usually, the rule base and the database are jointly referred to as the *knowledge base*.

The steps of *fuzzy reasoning* (inference operations upon fuzzy if-then rules) performed by fuzzy inference systems are:

1. Compare the input variables with the membership functions on the premise part to obtain the membership values (or compatibility measures) of each linguistic label. (This step is often called *fuzzification*).
2. Combine (through a specific T-norm operator, usually multiplication or min.) the membership values on the premise part to get *firing strength* (*weight*) of each rule.
3. Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.
4. Aggregate the qualified consequents to produce a crisp output. (This step is called *defuzzification*.)

Several types of fuzzy reasoning [22, 23] have been proposed in the literature. Depending on the types of fuzzy reasoning and fuzzy if-then rules employed, most fuzzy inference systems can be classified into three types (Figure 2):

Type 1: The overall output is the weighted average of each rule's crisp output induced by the rule's firing strength (the product or minimum of the degrees of match with the premise part) and output membership functions. The output membership functions used in this scheme must be monotonically non-decreasing [51].

Type 2: The overall fuzzy output is derived by applying "max" operation to the qualified fuzzy outputs (each of which is equal to the minimum of firing strength and the output membership function of each rule). Various schemes have been proposed to choose the final crisp output based on the overall fuzzy output; some of them are center of area, bisector of area, mean of maxima, maximum criterion, etc [22, 23].

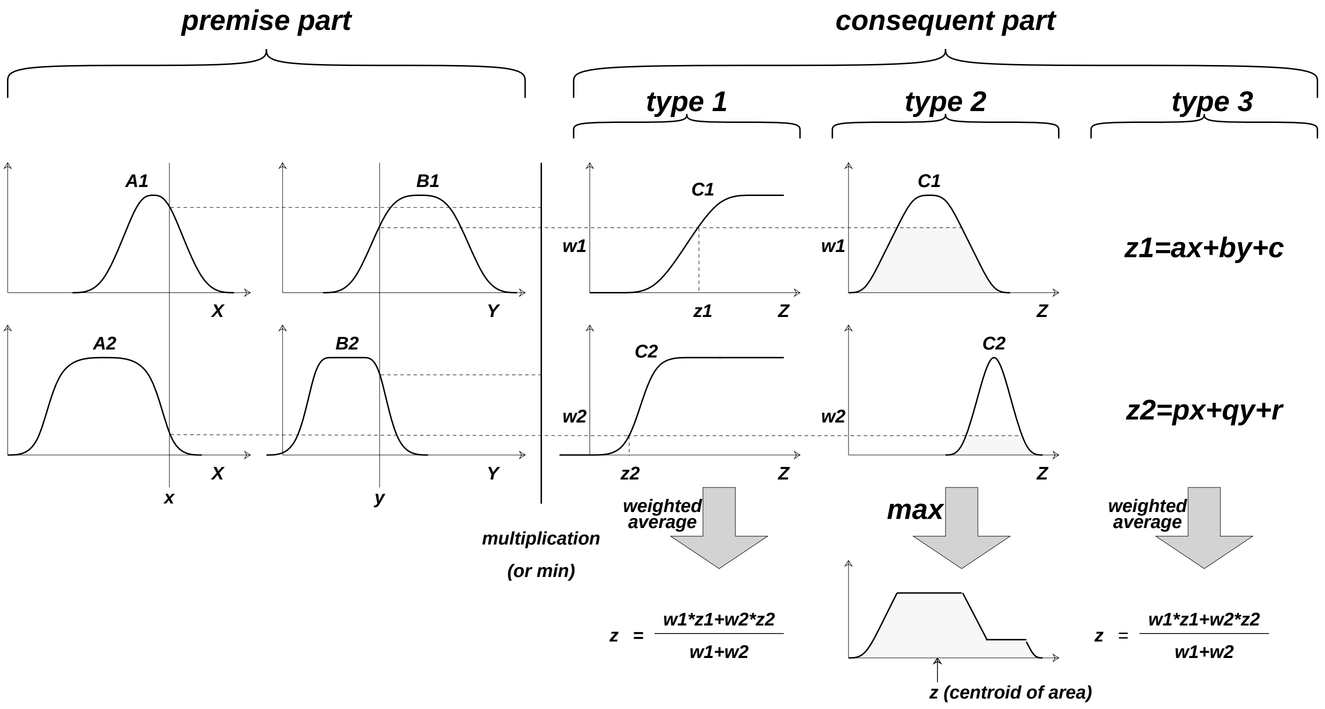


Figure 2: *Commonly used fuzzy if-then rules and fuzzy reasoning mechanisms*

Type 3: Takagi and Sugeno's fuzzy if-then rules are used [49]. The output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rule's output.

Figure 2 utilizes a two-rule two-input fuzzy inference system to show different types of fuzzy rules and fuzzy reasoning mentioned above. Be aware that most of the differences come from the specification of the consequent part (monotonically non-decreasing or bell-shaped membership functions, or crisp function) and thus the defuzzification schemes (weighted average, centroid of area, etc) are also different.

3 Adaptive Networks: Architectures and Learning Algorithms

This section introduces the architecture and learning procedure of the adaptive network which is in fact a superset of all kinds of feedforward neural networks with supervised learning capability. An adaptive network, as its name implies, is a network structure consisting of nodes and directional

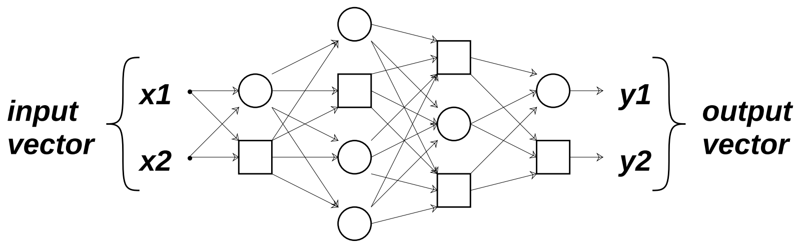


Figure 3: *An adaptive network.*

links through which the nodes are connected. Moreover, part or all of the nodes are adaptive, which means each output of these nodes depends on the parameter(s) pertaining to this node, and the learning rule specifies how these parameters should be changed to minimize a prescribed error measure.

The basic learning rule of adaptive networks is based on the gradient descent and the chain rule, which was proposed by by Werbos [57] in the 1970's. However, due to the state of artificial neural network research at that time, Werbos' early work failed to receive the attention it deserved. In the following presentation, the derivation is based on the author's work [11, 10] which generalizes the formulas in [37].

Since the basic learning rule is based the gradient method which is notorious for its slowness and tendency to become trapped in local minima, here we propose a hybrid learning rule which can speed up the learning process substantially Both the batch learning and the pattern learning of the proposed hybrid learning rule is discussed below, though our simulations are mostly based on the batch learning.

A. Architecture and Basic Learning Rule

An adaptive network (Figure 3) is a multi-layer feedforward network in which each node performs a particular function (*node function*) on incoming signals as well as a set of parameters pertaining to this node. The nature of the node functions may vary from node to node, and the choice of each node function depends on the overall input-output function which the adaptive network is required to carry out. Note that the links in an adaptive network only indicate the flow direction of signals between nodes; no weights are associated with the links.

To reflect different adaptive capabilities, we use both circle and square nodes in an adaptive network. A square node (adaptive node) has parameters while a circle node (fixed node) has none. The parameter set of an adaptive network is the union of the parameter sets of each adaptive node.

In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a gradient-based learning procedure described below.

Suppose that a given adaptive network has L layers and the k-th layer has $\#(k)$ nodes. We can denote the node in the i-th position of the k-th layer by (k, i), and its node function (or node output) by O_i^k. Since a node output depends on its incoming signals and its parameter set, we have

$$O_i^k = O_i^k(O_1^{k-1}, \ldots O_{\#(k-1)}^{k-1}, a, b, c, \ldots), \tag{1}$$

where a, b, c, etc. are the parameters pertaining to this node. (Note that we use O_i^k as both the node output and node function.)

Assuming the given training data set has P entries, we can define the *error measure* (or *energy function*) for the p-th $(1 \leq p \leq P)$ entry of training data entry as the sum of squared errors:

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2, \tag{2}$$

where $T_{m,p}$ is the m-th component of p-th target output vector, and $O_{m,p}^L$ is the m-th component of actual output vector produced by the presentation of the p-th input vector. Hence the overall error measure is $E = \sum_{p=1}^{P} E_p$.

In order to develop a learning procedure that implements gradient descent in E over the parameter space, first we have to calculate the *error rate* $\frac{\partial E_p}{\partial O}$ for p-th training data and for each node output O. The error rate for the output node at (L, i) can be calculated readily from equation (2):

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L). \tag{3}$$

For the internal node at (k, i), the error rate can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}, \tag{4}$$

where $1 \leq k \leq L-1$. That is, the error rate of an internal node can be expressed as a linear combination of the error rates of the nodes in the next layer. Therefore for all $1 \leq k \leq L$ and $1 \leq i \leq \#(k)$, we can find $\frac{\partial E_p}{\partial O_{i,p}^k}$ by equation (3) and (4).

Now if α is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}, \tag{5}$$

where S is the set of nodes whose outputs depend on α. Then the derivative of the overall error measure E with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha}. \tag{6}$$

Accordingly, the update formula for the generic parameter α is

$$\triangle \alpha = -\eta \frac{\partial E}{\partial \alpha}, \tag{7}$$

in which η is a learning rate which can be further expressed as

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} (\frac{\partial E}{\partial \alpha})^2}}, \tag{8}$$

where k is the *step size*, the length of each gradient transition in the parameter space. Usually, we can change the value of k to vary the speed of convergence. The heuristic rules for changing k are discussed in the section 5 where we report simulation results.

Actually, there are two learning paradigms for adaptive networks. With the *batch learning* (or *off-line learning*), the update formula for parameter α is based on equation (6) and the update action takes place only after the whole training data set has been presented, i.e., only after each *epoch* or *sweep*. On the other hand, if we want the parameters to be updated immediately after each input-output pair has been presented, then the update formula is based on equation (5) and it is referred to as the *pattern learning* (or *on-line learning*). In the following we will derive a faster hybrid learning rule and both of its learning paradigms.

B. Hybrid Learning Rule: Batch (Off-Line) Learning

Though we can apply the gradient method to identify the parameters in an adaptive network, the method is generally slow and likely to become trapped in local minima. Here we propose a hybrid learning rule [10] which combines the gradient method and the least squares estimate (LSE) to identify parameters.

For simplicity, assume that the adaptive network under consideration has only one output

$$output = F(\vec{I}, S), \tag{9}$$

where $\vec{I}$ is the set of input variables and S is the set of parameters. If there exists a function H such that the composite function $H \circ F$ is linear in some of the elements of S, then these elements can be identified by the least squares method. More formally, if the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2, \tag{10}$$

(where $\oplus$ represents direct sum) such that $H \circ F$ is linear in the elements of S_2, then upon applying H to equation (9), we have

$$H(output) = H \circ F(\vec{I}, S), \tag{11}$$

which is linear in the elements of S_2. Now given values of elements of S_1, we can plug P training data into equation (11) and obtain a matrix equation:

$$AX = B \tag{12}$$

where X is an unknown vector whose elements are parameters in S_2. Let $|S_2| = M$, then the dimensions of A, X and B are $P \times M$, $M \times 1$ and $P \times 1$, respectively. Since P (number of training data pairs) is usually greater than M (number of linear parameters), this is an overdetermined problem and generally there is no exact solution to equation (12). Instead, a *least squares estimate* (*LSE*) of X, X^*, is sought to minimize the squared error $||AX - B||^2$. This is a standard problem that forms the grounds for linear regression, adaptive filtering and signal processing. The most well-known formula for X^* uses the pseudo-inverse of X:

$$X^* = (A^T A)^{-1} A^T B, \tag{13}$$

where A^T is the transpose of A, and $(A^T A)^{-1} A^T$ is the pseudo-inverse of A if $A^T A$ is non-singular. While equation (13) is concise in notation, it is expensive in computation when dealing with the matrix inverse and, moreover, it becomes ill-defined if $A^T A$ is singular. As a result, we employ sequential formulas to compute the LSE of X. This sequential method of LSE is more efficient (especially when M is small) and can be easily modified to an on-line version (see below) for systems with changing characteristics. Specifically, let the ith row vector of matrix A defined in equation (12) be a_i^T and the ith element of B be b_i^T, then X can be calculated iteratively using the sequential formulas widely adopted in the literature [1, 7, 25, 41]:

$$\left.\begin{array}{rcl} X_{i+1} & = & X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} & = & S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0, 1, \cdots, P-1 \end{array}\right\}, \tag{14}$$

where S_i is often called the *covariance matrix* and the least squares estimate X^* is equal to X_P. The initial conditions to bootstrap equation (14) are $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$. When dealing with multi-output adaptive networks (*output* in equation (9) is a column vector), equation (14) still applies except that b_i^T is the i-th rows of matrix B.

Now we can combine the gradient method and the least squares estimate to update the parameters in an adaptive network. Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. In the forward pass, we supply input data and functional signals go forward to calculate each node output until the matrices A and B in equation (12) are obtained, and the parameters in S_2 are identified by the sequential least squares formulas in equation (14). After identifying parameters in S_2, the functional signals keep going forward till the error measure is calculated. In the backward pass, the error rates (the derivative of the error measure w.r.t. each node output, see equation(3) and (4)) propagate from the output end toward the input end, and the parameters in S_1 are updated by the gradient method in equation (7).

For given fixed values of parameters in S_1, the parameters in S_2 thus found are guaranteed to be the global optimum point in the S_2 parameter space due to the choice of the squared error measure. Not only can this hybrid learning rule decrease the dimension of the search space in the gradient method, but, in general, it will also cut down substantially the convergence time.

Take for example an one-hidden-layer back-propagation neural network with sigmoid activation functions. If this neural network has p output units, then the *output* in equation (9) is a column vector. Let $H(\cdot)$ be the inverse sigmoid function

$$H(x) = ln(\frac{x}{1-x}), \tag{15}$$

then equation (11) becomes a linear (vector) function such that each element of $H(output)$ is a linear combination of the parameters (weights and thresholds) pertaining to layer 2. In other words,

S_1 = weights and thresholds of hidden layer,

S_2 = weights and thresholds of output layer.

Therefore we can apply the back-propagation learning rule to tune the parameters in the hidden layer, and the parameters in the output layer can be identified by the least squares method. However, it should be keep in mind that by using the least squares method on the data transformed by $H(\cdot)$, the obtained parameters are optimal in terms of the transformed squared error measure instead of the original one. Usually this will not cause practical problem as long as $H(\cdot)$ is monotonically increasing.

C. Hybrid Learning Rule: Pattern (On-Line) Learning

If the parameters are updated after each data presentation, we have the *pattern learning* or *on-line learning* paradigm. This learning paradigm is vital to the on-line parameter identification for systems with changing characteristics. To modify the batch learning rule to its on-line version, it is obvious that the gradient descent should be based on E_p (see equation (5)) instead of E. Strictly speaking, this is not a truly gradient search procedure to minimize E, yet it will approximate to one if the learning rate is small.

For the sequential least squares formulas to account for the time-varying characteristics of the incoming data, we need to decay the effects of old data pairs as new data pairs become available. Again, this problem is well studied in the adaptive control and system identification literature and a number of solutions are available [7]. One simple method is to formulate the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. This amounts to the addition of a *forgetting factor* λ to the original sequential formula:

$$\left.\begin{array}{rcl} X_{i+1} & = & X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} & = & \frac{1}{\lambda}[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}] \end{array}\right\}, \tag{16}$$

where the value of λ is between 0 and 1. The smaller *lambda* is, the faster the effects of old data decay. But a small *lambda* sometimes causes numerical unstability and should be avoided.

4 ANFIS: Adaptive-Network-Based Fuzzy Inference System

The architecture and learning rules of adaptive networks have been described in the previous section. Functionally, there are almost no constraints on the node functions of an adaptive network except piecewise differentiability. Structurally, the only limitation of network configuration is that it should be of feedforward type. Due to these minimal restrictions, the adaptive network's applications are immediate and immense in various areas. In this section, we propose a class of adaptive networks which are functionally equivalent to fuzzy inference systems. The proposed architecture is referred to as *ANFIS*, standing for *Adaptive-Network-based Fuzzy Inference System*. We describe how to decompose the parameter set in order to apply the hybrid learning rule. Besides, we demonstrate how to apply the Stone-Weierstrass theorem to ANFIS with simplified

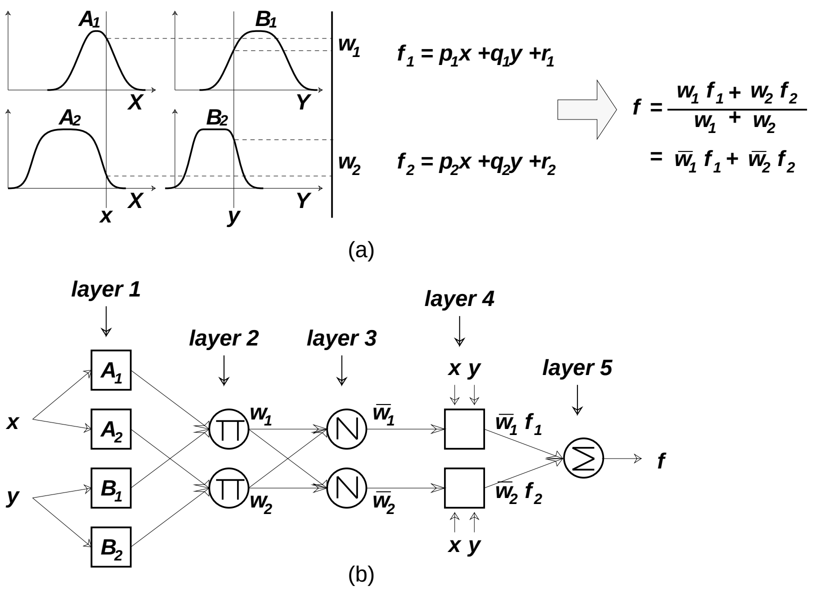


Figure 4: *(a) Type-3 fuzzy reasoning; (b) equivalent ANFIS (type-3 ANFIS).*

fuzzy if-then rules and how the radial basis function network relate to this kind of simplified ANFIS.

A. ANFIS architecture

For simplicity, we assume the fuzzy inference system under consideration has two inputs x and y and one output z. Suppose that the rule base contains two fuzzy if-then rules of Takagi and Sugeno's type [49]:

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1x + q_1y + r_1$,
Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2x + q_2y + r_2$.

then the type-3 fuzzy reasoning is illustrated in Figure 4(a), and the corresponding equivalent ANFIS architecture (*type-3 ANFIS*) is shown in Figure 4(b). The node functions in the same layer are of the same function family as described below:

Layer 1 Every node i in this layer is a square node with a node function

$$O_i^1 = \mu_{A_i}(x), \tag{17}$$

where x is the input to node i, and A_i is the linguistic label (*small* , *large*, etc.) associated with this node function. In other words, O_i^1 is the membership function of A_i and it specifies the degree to which the given x satisfies the quantifier A_i. Usually we choose $\mu_{A_i}(x)$ to be bell-shaped with maximum equal to 1 and minimum equal to 0, such as

$$\mu_{A_i}(x) = \frac{1}{1 + [(\frac{x-c_i}{a_i})^2]^{b_i}}, \tag{18}$$

or

$$\mu_{A_i}(x) = exp\{-[(\frac{x - c_i}{a_i})^2]^{b_i}\}, \tag{19}$$

where $\{a_i,\, b_i,\, c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped functions vary accordingly, thus exhibiting various forms of membership functions on linguistic label A_i. In fact, any continuous and piecewise differentiable functions, such as commonly used trapezoidal or triangular-shaped membership functions, are also qualified candidates for node functions in this layer. Parameters in this layer are referred to as *premise parameters*.

Layer 2 Every node in this layer is a circle node labeled Π which multiplies the incoming signals and sends the product out. For instance,

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y),\ i = 1, 2. \tag{20}$$

Each node output represents the firing strength of a rule. (In fact, other *T-norm* operators that perform generalized AND can be used as the node function in this layer.)

Layer 3 Every node in this layer is a circle node labeled N. The i-th node calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths:

$$\overline{w}_i = \frac{w_i}{w_1 + w_2},\ i = 1, 2. \tag{21}$$

For convenience, outputs of this layer will be called called *normalized firing strengths*.

Layer 4 Every node i in this layer is a square node with a node function

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i(p_i x + q_i y + r_i), \tag{22}$$

where $\overline{w}_i$ is the output of layer 3, and $\{p_i,\, q_i,\, r_i\}$ is the parameter set. Parameters in this layer will be referred to as *consequent parameters*.

Layer 5 The single node in this layer is a circle node labeled Σ that computes the overall output as the summation of all incoming signals, i.e.,

$$O_1^5 = overall\ output = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{23}$$

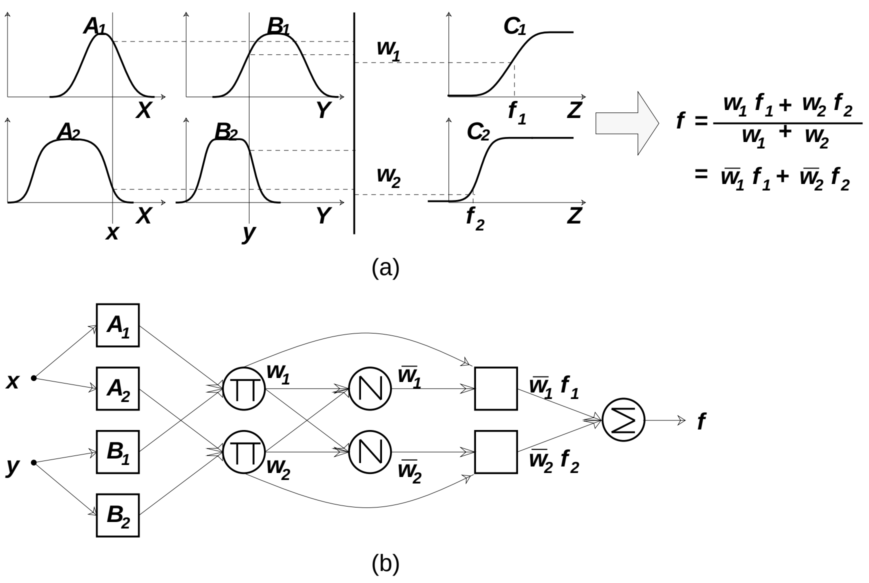


Figure 5: *(1) Type-1 fuzzy reasoning; (b) equivalent ANFIS (type-1 ANFIS).*

Thus we have constructed an adaptive network which is functionally equivalent to a type-3 fuzzy inference system. For type-1 fuzzy inference systems, the extension is quite straightforward and the type-1 ANFIS is shown in Figure 5 where the output of each rule is induced jointly by the output membership funcion and the firing strength. For type-2 fuzzy inference systems, if we replace the centroid defuzzification operator with a discrete version which calculates the approximate centroid of area, then type-3 ANFIS can still be constructed accordingly. However, it will be more complicated than its type-3 and type-1 versions and thus not worth the efforts to do so.

Figure 6 shows a 2-input, type-3 ANFIS with 9 rules. Three membership functions are associated with each input, so the input space is partitioned into 9 fuzzy subspaces, each of which is governed by a fuzzy if-then rules. The premise part of a rule delineates a fuzzy subspace, while the consequent part specifies the output within this fuzzy subspace.

B. Hybrid Learning Algorithm

From the proposed type-3 ANFIS architecture (Figure 4), it is observed that given the values of premise parameters, the overall output can be expressed as a linear combinations of the consequent parameters. More precisely, the output f in Figure 4 can be rewritten as

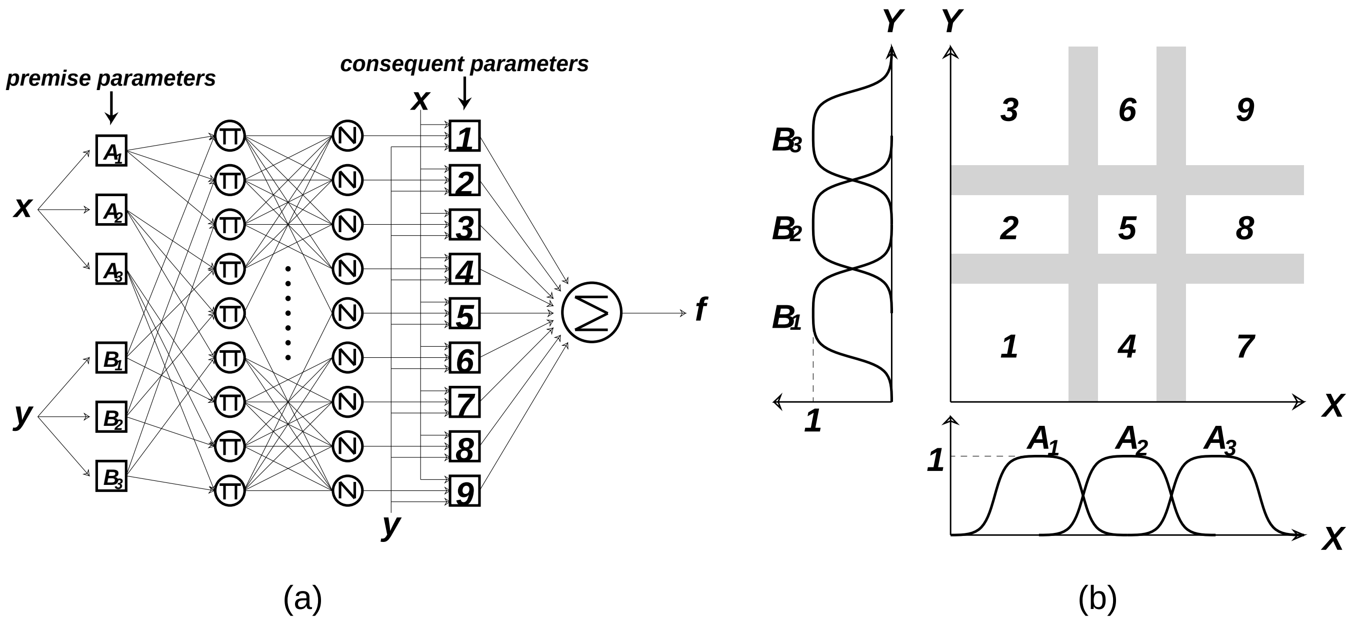


Figure 6: *(a) 2-input type-3 ANFIS with 9 rules; (b) corresponding fuzzy subspaces* .

$$
\begin{aligned}
f &= \tfrac{w_1}{w_1+w_2} f_1 + \tfrac{w_2}{w_1+w_2} f_2 \\
&= \overline{w}_1 f_1 + \overline{w}_2 f_2 \\
&= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + (\overline{w}_1)r_1 + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + (\overline{w}_2)r_2,
\end{aligned} \tag{24}
$$

which is linear in the consequent parameters (p_1, q_1, r_1, p_2, q_2 and r_2). As a result, we have

S = set of total parameters,

S_1 = set of premise parameters,

S_2 = set of consequent parameters,

in equation (10); $H(\cdot)$ and $F(\cdot,\cdot)$ are the identity function and the function of the fuzzy inference system, respectively. Therefore the hybrid learning algorithm developed in the previous chapter can be applied directly. More specifically, in the forward pass of the hybrid learning algorithm, functional signals go forward till layer 4 and the consequent parameters are identified by the least squares estimate. In the backward pass, the error rates propagate backward and the premise parameters are updated by the gradient descent. Table 1 summarizes the activities in each pass.

As mentioned earlier, the consequent parameters thus identified are optimal (in the consequent parameter space) under the condition that the premise parameters are fixed. Accordingly the hybrid approach is much faster than the strict gradient descent and it is worthwhile to look for the possibility of decomposing the parameter set in the manner of equation (10). For type-1

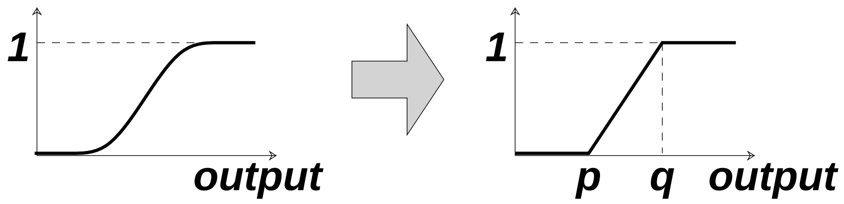


Figure 7: *Piecewise linear approximation of membership functions on the consequent part of type-1 ANFIS.*

ANFIS, this can be achieved if the membership function on the consequent part of each rule is replaced by a piecewise linear approximation with two consequent parameters (Figure 7). In this case, again, the consequent parameters constitute set S_2 and the hybrid learning rule can be employed directly.

However, it should be noted that the computation complexity of the least squares estimate is higher than that of the gradient descent. In fact, there are four methods to update the parameters, as listed below according to their computation complexities:

1. *Gradient descent only* : all parameters are updated by the gradient descent.
2. *Gradient descent and one pass of LSE* : the LSE is applied only once at the very beginning to get the initial values of the consequent parameters and then the gradient descent takes over to update all parameters.
3. *Gradient descent and LSE* : this is the proposed hybrid learning rule.
4. *Sequential LSE only* : the ANFIS is linearized w.r.t. all parameters and the extended kalman filter algorithm is employed to update all parameters. This has been proposed in the neural network literature [40, 39].

The choice of above methods should be based on the trade-off between computation complexity and resulting performance. Our simulations presented in the next section are performed by the

-	forward pass	backward pass
premise parameters	fixed	gradient descent
consequent parameters	least squares estimate	fixed
signals	node outputs	error rates

Table 1: *Two passes in the hybrid learning procedure for ANFIS.*

third method.

Because the update formulas of the premise and consequent parameters are decoupled in the hybrid learning rule (see Table 1), further speedup of learning is possible by using other versions of the gradient method on the premise parameters, such as conjugate gradient descent, second-order back-propagation [33], quick-propagation [5], nonlinear optimization [54] and many others.

C. Fuzzy Inference Systems with Simplified Fuzzy If-Then Rules

Though the reasoning mechanisms (Figure 2) introduced earlier are commonly used in the literature, each of them has inherent drawbacks. For type-1 reasoning (Figure 2 or 5), the membership functions on the consequence part are restricted to monotonically non-decreasing functions which are not compatible with linguistic terms such as "medium" whose membership function should be bell-shaped. For type-2 reasoning (Figure 2), the defuzzification process is time-consuming and systematic fine-tuning of the parameters are not easy. For type-3 reasoning (Figure 2 or 4), it is just hard to assign any appropriate linguistic terms to the consequence part which is bilinear in the input variables. To cope with these disadvantages, simplified fuzzy if-then rules of the following form are introduced:

If x is big and y is small, then z is d.

where d is a *crisp* value. Due to the fact that the output z is described by a crisp value (or equivalently, a singular membership function), this class of simplified fuzzy if-then rules can employ all three types of reasoning mechanisms. More specifically, the consequent part of this simplified fuzzy if-then rule is represented by a step function (centered at $z = d$) in type 1, a singular membership function (at $z = d$) in type 2, and a constant output function in type 3, respectively. The three reasoning mechanisms are unified under this simplified fuzzy if-then rules.

Most of all, with this simplified fuzzy if-then rule, it is possible to prove that under certain circumstance, the resulting fuzzy inference system has unlimited approximation power to match any nonlinear functions arbitrarily well on a compact set. We will proceed this in a descriptive way by applying the Stone-Weierstrass theorem [18, 36] stated below.

Theorem 4.1 *Let domain D be a compact space of N dimensions, and let $\mathcal{F}$ be a set of continuous real-valued functions on D, satisfying the following criteria:*

1. **Identity function**: *The constant $f(x) = 1$ is in $\mathcal{F}$.*

2. **Separability**: *For any two points $x_1 \neq x_2$ in D, there is an f in $\mathcal{F}$ such that $f(x_1) \neq f(x_2)$.*

3. **Algebraic closure**: *If f and g are any two functions in $\mathcal{F}$, then fg and $af + bg$ are in F for any two real numbers a and b.*

Then $\mathcal{F}$ is dense in $C(D)$, the set of continuous real-valued functions on D. In other words, for any $\epsilon > 0$, and any function g in $C(D)$, there is a function f in $\mathcal{F}$ such that $|g(x) - f(x)| < \epsilon$ for all $x \in D$.

In application of fuzzy inference systems, the domain in which we operate is almost always closed and bounded and therefore it is compact. For the first and second criteria, it is trivial to find simplified fuzzy inference systems that satisfy them. Now all we need to do is examine the algebraic closure under addition and multiplication. Suppose we have two fuzzy inference systems S and $\tilde{S}$; each has two rules and the output of each system can be expressed as

$$S: \; z = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}, \tag{25}$$

$$\tilde{S}: \; \tilde{z} = \frac{\tilde{w}_1 \tilde{f}_1 + \tilde{w}_2 \tilde{f}_2}{\tilde{w}_1 + \tilde{w}_2}, \tag{26}$$

where f_1, f_2, $\tilde{f}_1$ and $\tilde{f}_2$ are constant output of each rule. Then $az + b\tilde{z}$ and $z\tilde{z}$ can be calculated as follows:

$$\begin{aligned} az + b\tilde{z} &= a\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} + b\frac{\tilde{w}_1 \tilde{f}_1 + \tilde{w}_2 \tilde{f}_2}{\tilde{w}_1 + \tilde{w}_2} \\ &= \frac{w_1 \tilde{w}_1 (af_1 + b\tilde{f}_1) + w_1 \tilde{w}_2 (af_1 + b\tilde{f}_2) + w_2 \tilde{w}_1 (af_2 + b\tilde{f}_1) + w_2 \tilde{w}_2 (af_2 + b\tilde{f}_2)}{w_1 \tilde{w}_1 + w_1 \tilde{w}_2 + w_2 \tilde{w}_1 + w_2 \tilde{w}_2}, \end{aligned}$$

$$z\tilde{z} = \frac{w_1 \tilde{w}_1 f_1 \tilde{f}_1 + w_1 \tilde{w}_2 f_1 \tilde{f}_2 + w_2 \tilde{w}_1 f_2 \tilde{f}_1 + w_2 \tilde{w}_2 f_2 \tilde{f}_2}{w_1 \tilde{w}_1 + w_1 \tilde{w}_2 + w_2 \tilde{w}_1 + w_2 \tilde{w}_2}, \tag{27}$$

which are of the same form as equation (25) and (26). Apparently the ANFIS architectures that compute $az + b\tilde{z}$ and $z\tilde{z}$ are of the same class of S and $\tilde{S}$ if and only if the class of membership functions is invariant under multiplication. This is loosely true if the class of membership functions is the set of all bell-shaped functions, since the multiplication of two bell-shaped function is almost always still bell-shaped. Another more tightly defined class of membership functions satisfying this criteria, as pointed out by Wang [52, 53], is the scaled Gaussian membership function:

$$\mu_{A_i}(x) = a_i exp[-(\frac{x - c_i}{a_i})^2], \tag{28}$$

Therefore by choosing an appropriate class of membership functions, we can conclude that the ANFIS with simplified fuzzy if-then rules satisfy the four criteria of the Stone-Weierstrass theorem. Consequently, for any given $\epsilon > 0$, and any real-valued function g, there is a fuzzy inference system S such that $|g(\vec{x}) - S(\vec{x})| < \epsilon$ for all $\vec{x}$ in the underlying compact set. Moreover, since the simplified ANFIS is a proper subset of all three types of ANFIS in Figure 2, we can draw the conclusion that all the three types of ANFIS have unlimited approximation power to match any given data set. However, caution has to be taken in accepting this claim since there is no mention about how to construct the ANFIS according to the given data set. That is why learning plays a role in this context.

Another interesting aspect of the simplified ANFIS architecture is its functional equivalence to the radial basis function network (RBFN). This functional equivalence is established when the Gaussian membership function is used in the simplified ANFIS. A detailed treatment can be found in [13]. This functional equivalence provides us with a shortcut for better understanding of ANFIS and RBFN and advances in either literatures apply to both directly. For instance, the hybrid learning rule of ANFIS can be apply to RBFN directly and, vice versa, the approaches used to identify RBFN parameters, such as clustering preprocess [28, 29], orthogonal least squares learning [3], generalization properties [2], sequential adaptation [15], among others [14, 30], are all applicable techniques for ANFIS.

5 Application Examples

This section presents the simulation results of the proposed type-3 ANFIS with both batch (off-line) and pattern (on-line) learning. In the first two examples, ANFIS is used to model highly nonlinear functions and the results are compared with earlier work. In the third example, ANFIS is used as an identifier to identify a nonlinear component on-linely in a discrete control system. Lastly, we use ANFIS to predict a chaotic time series and compare the results with various statistical and connectionist approaches.

A. Practical Considerations

In a conventional fuzzy inference system, the number of rules is decided by an expert who is familiar with the system to be modeled. In our simulation, however, no expert is available and the number of membership functions assigned to each input variable is chosen empirically, i.e., by

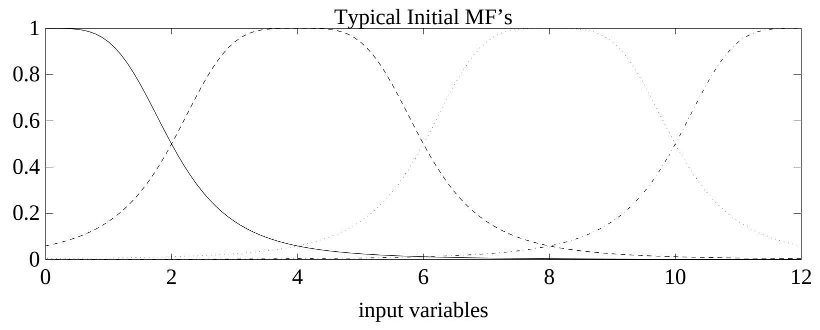


Figure 8: *A typical initial membership function setting in our simulation. (The operating range is assumed to be* $[0, 12]$*.)*

examining the desired input-output data and/or by trial and error. This situation is much the same as that of neural networks; there are no simple ways to determine in advance the minimal number of hidden nodes necessary to achieve a desired performance level.

After the number of membership functions associated with each inputs are fixed, the initial values of premise parameters are set in such a way that the membership functions are equally spaced along the operating range of each input variable. Moreover, they satisfy ϵ*-completeness* [22, 23] with $\epsilon = 0.5$, which means that given a value x of one of the inputs in the operating range, we can always find a linguistic label A such that $\mu_A(x) \geq \epsilon$. In this manner, the fuzzy inference system can provide smooth transition and sufficient overlapping from one linguistic label to another. Though we did not attempt to keep the *epsilon*-completeness during the learning in our simulation, it can be easily achieved by using the constrained gradient method [61]. Figure 8 shows a typical initial membership function setting when the number of membership function is 4 and the operating range is $[0, 12]$.

We mentioned that the step size k in equation (8) may influence the speed of convergence. It is observed that if k is small, the gradient method will closely approximate the gradient path, but convergence will be slow since the gradient must be calculated many times. On the other hand, if k is large, convergence will initially be very fast, but the algorithm will oscillate about the optimum. Based on these observations, we update k according to the following two heuristic rules (see Figure 9):

1. If the error measure undergoes 4 consecutive reductions, increase k by 10%.

2. If the error measure undergoes 2 consecutive combinations of 1 increase and 1 reduction,

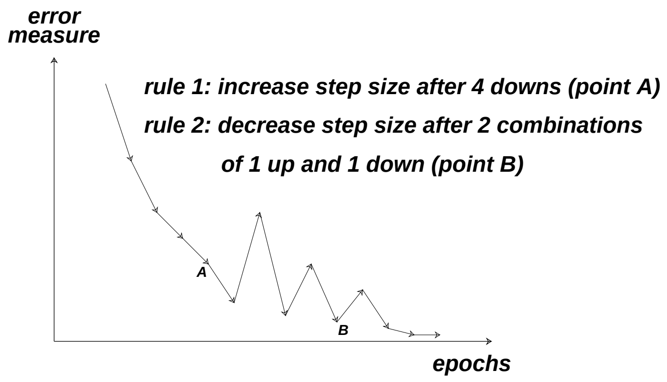

error measure
rule 1: increase step size after 4 downs (point A)
rule 2: decrease step size after 2 combinations of 1 up and 1 down (point B)
A
B
epochs

Figure 9: *Two heuristic rules for updating step size k.*

decrease k by 10%.

Though the numbers 10%, 4 and 2 are chosen more or less arbitrarily, the results shown in our simulation appear to be satisfactory. Furthermore, due to this dynamical update strategy, the initial value of k is usually not critical as long as it is not too big.

B. Simulation Results

Example 1: Modeling 2-input Nonlinear Function

In this example, we consider using ANFIS to model a nonlinear *sinc* equation

$$z = sinc(x, y) = \frac{sin(x)}{x} \times \frac{sin(y)}{y}. \tag{29}$$

From the grid points of the range $[-10, 10] \times [-10, 10]$ within the input space of the above equation, 121 training data pairs were obtained first. The ANFIS used here contains 16 rules, with four membership functions being assigned to each input variable and the total number of fitting parameters is 72. (We also tried ANFIS with 4 rules and 9 rules, but obviously they are too simple to describe the highly nonlinear sinc function.) Note that throughout the simulation examples in this section, all the membership functions used are defined as in equation (18) which contains three fitting parameters. Thus the 72 parameters in this example are composed of 24 premise parameters and 48 consequent parameters.

Figure 10 shows the RMSE (root mean squared error) curves for both the 2-18-1 neural network and the ANFIS. Each curve is the average of ten runs: for the neural network, this ten runs were started from 10 different set of initial random weights; for the ANFIS, 10 different initial step

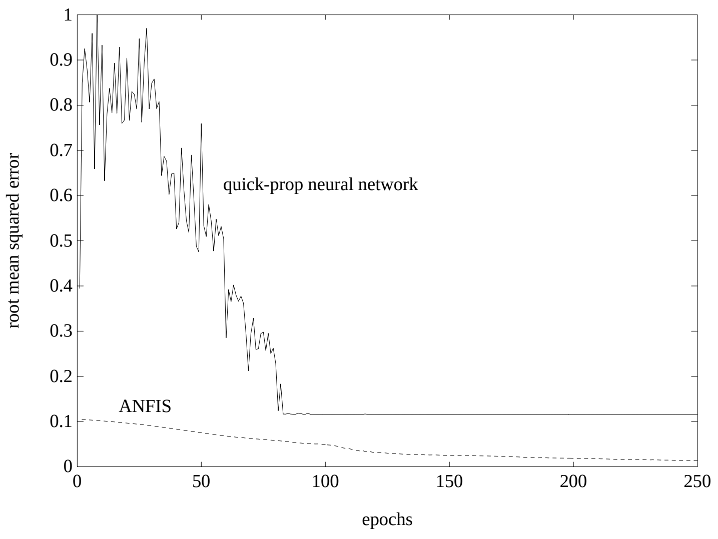


Figure 10: *RMSE curves for the quick-propagation neural networks and the ANFIS.*

size (= $0.01, 0.02, \ldots, 0.10$) were used. The neural network, containing 73 fitting parameters (connection weights and thresholds), was trained with quick propagation [5] which is considered one of the best learning algorithms for connectionist models. Figure 10 demonstrate how ANFIS can effectively model a highly nonlinear surface as compared to neural networks. However, this comparison cannot taken to be universal since we did not attempt an exhaustive search to find the optimal settings for the quick-propagation learning rule of the neural networks.

The training data and other reconstructed surfaces at different epoch numbers are shown in Figure 11. (Since the error measure is always computed after the forward pass is over, the epoch numbers shown in Figure 11 always end with ".5".) Note that the reconstructed surface after 0.5 epoch is due to the identification of consequent parameters only and it already looks similar to the training data surface.

Figure 12 lists the initial and final membership functions. It is interesting to observe that the sharp changes of the training data surface around the origin is accounted for by the moving of the membership functions toward the origin. Theoretically, the final memberships on both x and y should be symmetric with respect to the origin. However, they are not symmetric due to the computer truncation errors and the approximate initial conditions for bootstrapping the calculation of the sequential least squares estimate 14.

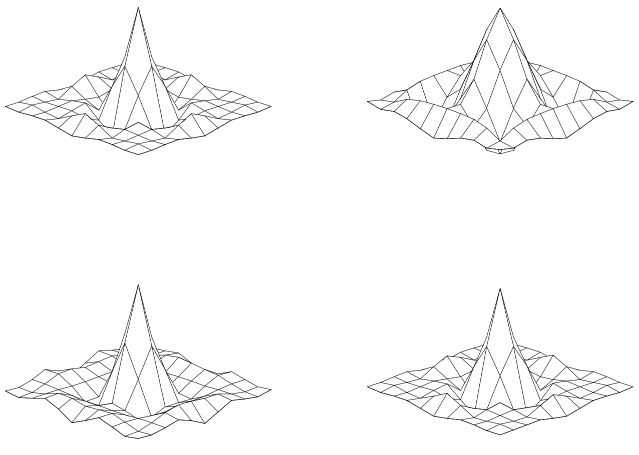


Figure 11: *Training data (upper left) and reconstructed surfaces at 0.5 (upper right), 99.5 (lower left) and 249.5 (lower right) epochs. (Example 1).*

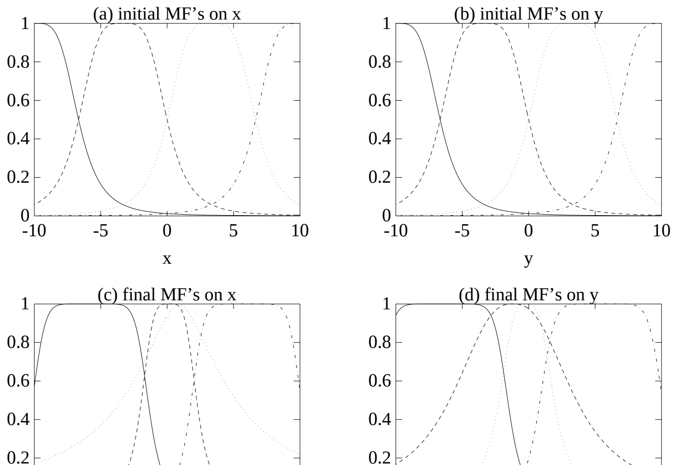


Figure 12: *Initial and final membership functions of example 1.*

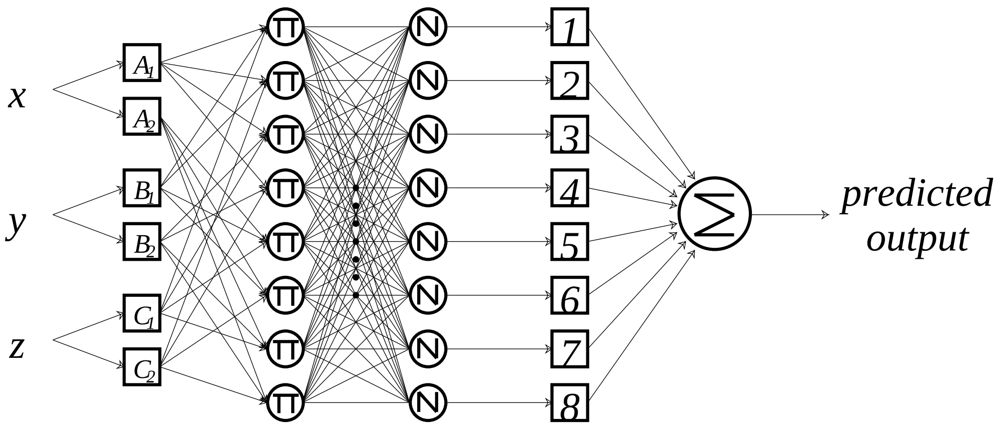


Figure 13: *The ANFIS architecture for example 2. (Note that the connections from inputs to layer 4 are not shown.)*

Example 2: Modeling a 3-input Nonlinear Function

The training data in this example are obtained from

$$output = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2, \tag{30}$$

which was also used by Takagi et al. [48], Sugeno et al. [43] and Kondo [19] to verify their approaches. The ANFIS (see Figure 13) used here contains 8 rules, with 2 membership functions being assigned to each input variable. 216 training data and 125 checking data were sampled uniformly from the input ranges $[1, 6] \times [1, 6] \times [1, 6]$ and $[1.5, 5.5] \times [1.5, 5.5] \times [1.5, 5.5]$, respectively. The training data was used for the training of ANFIS, while the checking data was used for verifying the identified ANFIS only. To allow comparison, we use the same performance index adopted in [43, 19]:

$$APE = average\ percentage\ error = \frac{1}{P}\sum_{i=1}^{P} \frac{|\, T(i) - O(i)\, |}{|\, T(i)\, |} * 100\%. \tag{31}$$

where P is the number of data pairs; $T(i)$ and $O(i)$ are i-th desired output and calculated output, respectively.

Figure 14 illustrates the membership functions before and after training. The training error curves with different initial step sizes (from 0.01 to 0.09) are shown in Figure 15(a), which demonstrates that the initial step size is not too critical on the final performance as long as it is not too big. Figure 15(b) is the training and checking error curves with initial step size equal to 0.1. After 199.5 epochs, the final results are $APE_{trn} = 0.043\%$ and $APE_{chk} = 1.066\%$, which is listed

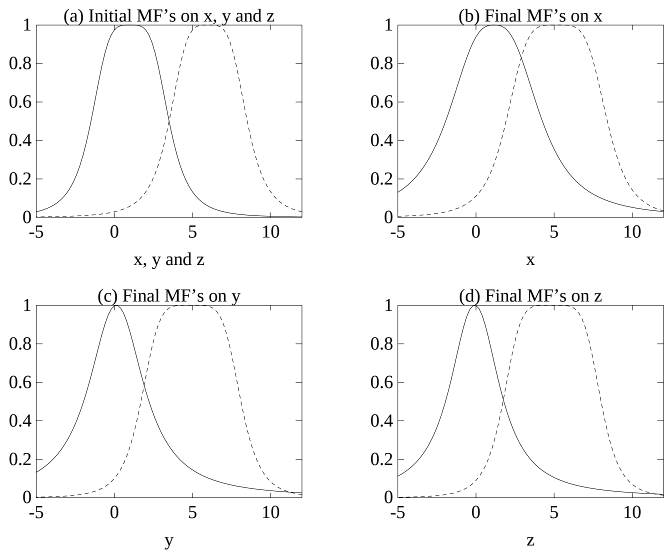


Figure 14: *Example 2, (a) membership functions before learning; (b)(c) (d) membership functions after learning.*

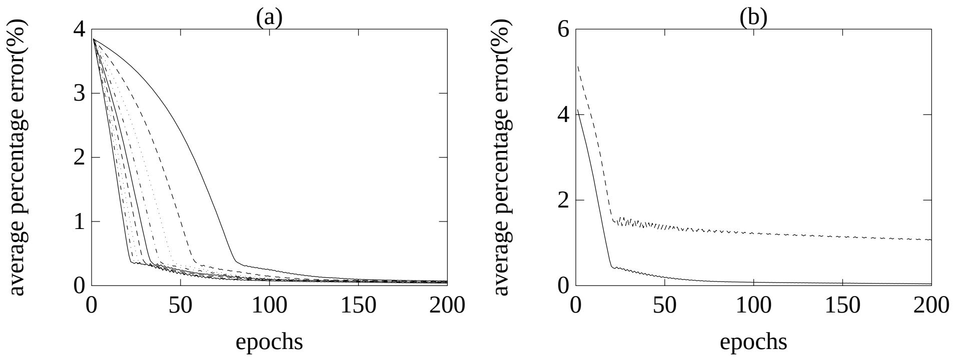


Figure 15: Error curves of example 2: (a) 9 training error curves for 9 initial step size from 0.01 (solid line) to 0.09; (b) training (solid line) and checking (dashed line) error curves with initial step size equal to 0.1
.

Model	APE_{trn} (%)	APE_{chk} (%)	Parameter no.	Training Set Size	Checking Set Size
ANFIS	0.043	1.066	50	216	125
GMDH model [19]	4.7	5.7	-	20	20
Fuzzy model 1 [43]	1.5	2.1	22	20	20
Fuzzy model 2 [43]	0.59	3.4	32	20	20

Table 2: *Example 2: comparisons with earlier work. (The last three rows are from [43].)*

in Table 2 along with other earlier work [43, 19]. Since each simulation cited here was performed under different assumptions and with different training and checking data sets, we cannot make conclusive comments here.

Example 3: On-line Identification in Control Systems

Here we repeat the simulation example 1 of [31] where a 1-20-10-1 neural network is employed to identify a nonlinear component in a control system, except that we use ANFIS to replace the neural network. The plant under consideration is governed by the following difference equation:

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + f(u(k)), \tag{32}$$

where y(k) and u(k) are the output and input, respectively, at time index k, and the unknown function $f(\cdot)$ has the form

$$f(u) = 0.3sin(\pi u) + 0.6sin(3\pi u) + 0.5sin(5\pi u). \tag{33}$$

In order to identify the plant, a series-parallel model governed by the difference equation

$$\hat{y}(k+1) = 0.3\hat{y}(k) + 0.6\hat{y}(k-1) + F(u(k)) \tag{34}$$

was used where $F(\cdot)$ is the function implemented by ANFIS and its parameters are updated at each time index. Here the ANFIS has 7 membership functions on its input (thus 7 rules, and 35 fitting parameters) and the pattern (on-line) learning paradigm was adopted with a learning rate $\eta = 0.1$ and a forgetting factor $\lambda = 0.99$. The input to the plant and the model was a sinusoid $u(k) = sin(2\pi k/250)$ and the adaptation started at $k = 1$ and stopped at $k = 250$. As shown in Figure 16, the output of the model follows the output of the plant almost immediately even after the adaptation stopped at $k = 250$ and the $u(k)$ is changed to $0.5sin(2\pi k/250) + 0.5sin(2\pi k/25)$ after $k = 500$. As a comparison, the neural network in [31] fails to follow the plant when the

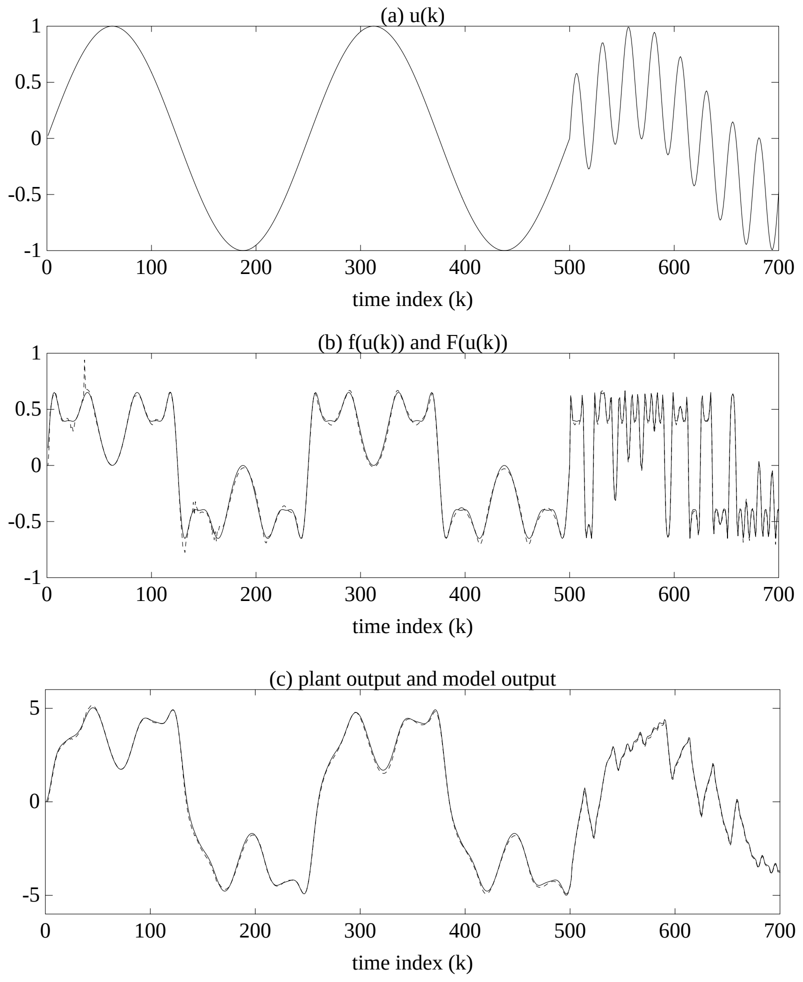


Figure 16: *Example 3: (a) $u(k)$; (a) $f(u(k))$ and $F(u(k))$; (b) plant output and model output.*

Method	Parameter Number	Time Steps of Adaptation
NN	261	50000
ANFIS	35	250

Table 3: *Example 3: comparison with NN identifier [31].)*

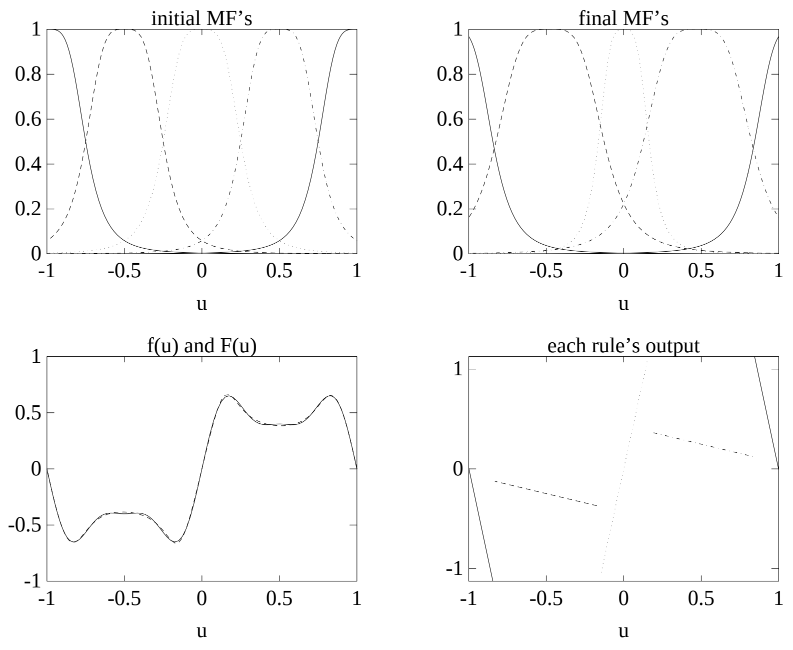


Figure 17: *Example 3: batch learning with 5 MF's.*

adaptation stopped at $k = 500$ and the identification procedure had to continue for $50,000$ time steps using a random input. Table 3 summarizes the comparison.

In the above, the MF number is determined by trial and errors. If the MF number is below 7 then the model output will not follow the plant output satisfactorily after 250 adaptations. But can we decrease the parameter numbers by using batch learning which is supposed to be more effective? Figure 17, 18 and 19 show the results after 49.5 epochs of batch learning when the MF numbers are 5, 4 and 3, respectively. As can be seen, the ANFIS is a good model even when the MF is as small as 3. However, as the MF number is getting smaller, the correlation between $F(u)$ and each rule's output is getting less obvious in the sense that it is harder to sketch $F(u)$ from each rule's consequent part. In other words, when the parameter number is reduced mildly, usually the ANFIS can still do the job but at the cost of sacrificing its semantics in terms of the local-description nature of fuzzy if-then rules; it is less of a structured knowledge representation and more of a black-box model (like neural networks).

Example 4: Predicting Chaotic Dynamics

Example 1, 2 and 3 show that the ANFIS can be used to model highly nonlinear functions

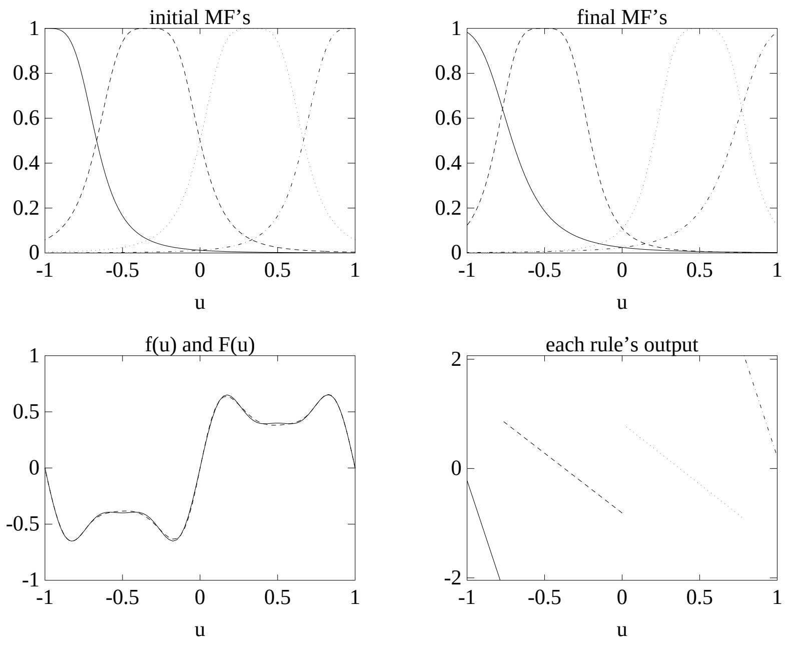


Figure 18: *Example 3: batch learning with 4 MF's.*

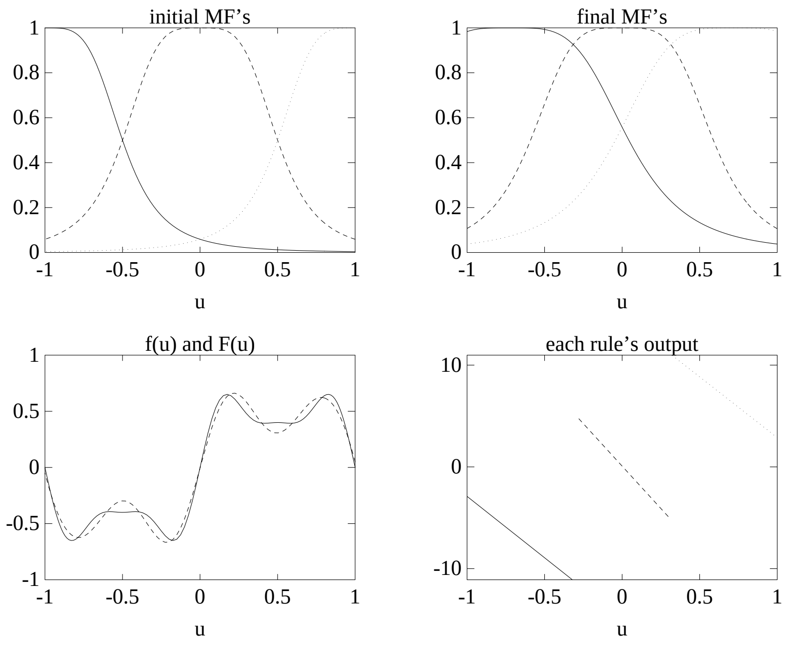


Figure 19: *Example 3: batch learning with 3 MF's.*

effectively. In this example, we will demonstrate how the proposed ANFIS can be employed to predict future values of a chaotic time series. The performance obtained in this example will be compared with the results of a cascade-correlation neural network approach reported in [35] and a simple conventional statistical approach, the auto-regressive (AR) model.

The time series used in our simulation is generated by the chaotic Mackey-Glass differential delay equation [26] defined below:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \tag{35}$$

The prediction of future values of this time series is a benchmark problem which has been considered by a number of connectionist researchers (Lapedes and Farber [21], Moody [29, 27], Jones et al. [14], Crower [35] and Sanger [38]).

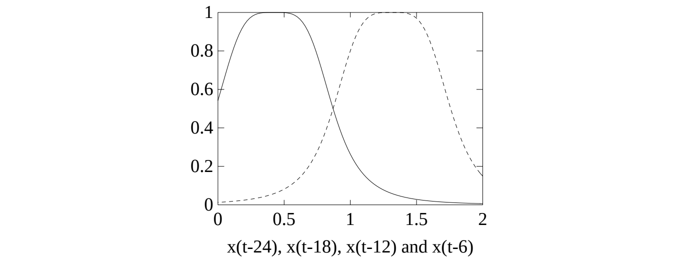
The goal of the task is to use known values of the time series up to the point $x = t$ to predict the value at some point in the future $x = t + P$. The standard method for this type of prediction is to create a mapping from D points of the time series spaced $\triangle$ apart, that is, $(x(t-(D-1)\triangle)$, ..., $x(t-\triangle)$, $x(t))$, to a predicted future value $x(t+P)$. To allow comparison with earlier work (Lapedes and Farber [21], Moody [29, 27], Crower [35]), the values $D = 4$ and $\triangle = P = 6$ were used. All other simulation settings in this example were purposedly arranged to be as close as possible to those reported in [35].

To obtain the time series value at each integer point, we applied the fourth-order Runge-Kutta method to find the numerical solution to equation (35). The time step used in the method is 0.1, initial condition $x(0) = 1.2$, $\tau = 17$, and $x(t)$ is thus derived for $0 \leq t \leq 2000$. (We assume $x(t) = 0$ for $t < 0$ in the integration.) From the Mackey-Glass time series $x(t)$, we extracted 1000 input-output data pairs of the following format:

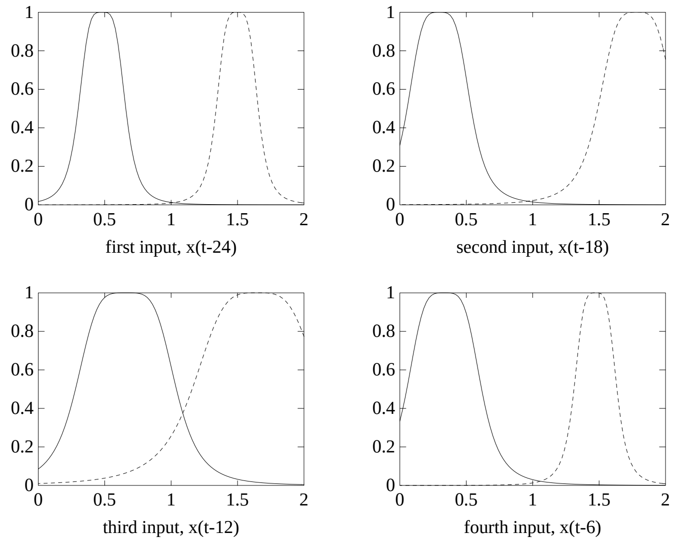
$$[x(t-24), x(t-18), x(t-12), x(t-6); x(t)], \tag{36}$$

where $t = 124$ to 1123. The first 500 pairs (training data set) was used for training the ANFIS while the remaining 500 pairs (checking data set) were used for validating the identified model. The number of membership functions assigned to each input of the ANFIS was arbitrarily set to 2, so the rule number is 16. Figure 20 (a) is the initial membership functions for each input variable. Note that the ANFIS used here contains a total of 104 fitting parameters, of which 24 are premise parameters and 80 are consequent parameters

After 499.5 epochs, we had $RMSE_{trn} = 0.0016$ and $RMSE_{chk} = 0.0015$, which are much



(a) Initial MF's for all four input variables.



(b) Final MF's for each input variable.

Figure 20: *Membership functions of example 3, (a) before learning; (b) after learning.*

better when compared with other approaches explained below. The desired and predicted values for both training data and checking data are essentially the same in Figure 21(a); their differences (Figure 21(b)) can only be seen on a finer scale. Figure 20 (b) is the final membership functions; Figure 22 shows the RMSE curves which indicate most of the learning was done in the first 100 epochs. It is interesting to note the unusual phenomenon that $RMSE_{trn} < RMSE_{chk}$ during the learning process. Considering both the RMSE's are vary small, we conclude that: (1) the ANFIS has captured the essential components of the underlying dynamics; (2) the training data contains the effects of the initial conditions (remember that we set $x(t) = 0$ for $t \leq 0$ in the integration) which might not be easily accounted for by the essential components identified by the ANFIS.

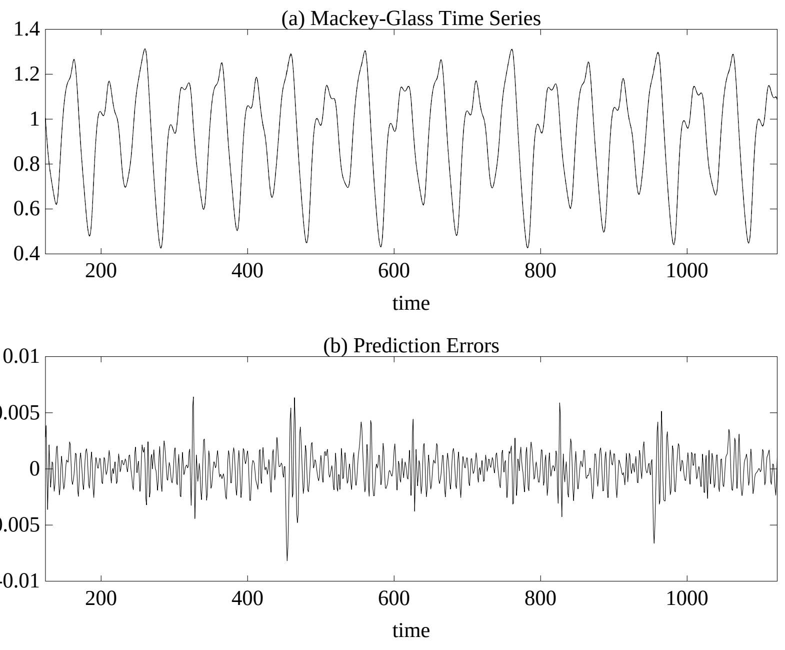


Figure 21: Example 3, (a) Mackey-Glass time series from $t = 124$ to 1123 and six-step ahead prediction (which is indistinguishable from the time series here); (b) prediction error.

As a comparison, we performed the same prediction by using the auto-regressive (AR) model with the same number of parameters:

$$x(t) = a_0 + a_1 x(t-6) + a_2 x(t-2*6) + ... + a_{103} x(t-103*6), \tag{37}$$

where there are 104 fitting parameters a_k, $k = 0$ to 103. From $t = 718$ to 1717, we extracted 1000 data pairs, of which the first 500 were used to identify a_k and the remaining were used for checking. The results obtained through the standard least squares estimate are $RMSE_{trn} = 0.005$ and $RMSE_{chk} = 0.078$ which is much worse than those of ANFIS. Figure 23 shows the predicted values and the prediction errors. Obviously, the over-parameterization of the AR model causes over-fitting in the training data and large errors in the checking data. To search for the best AR model in terms of generalization capability, we tried out different AR models with parameter number being varied from 2 to 104; Figure 24 shows the results where the AR model with the best generalization capability is obtained when the parameter number is 45. Based on this best AR model, we repeat the generalization test and Figure 25 shows the results where there is no over-fitting at the price of larger training errors.

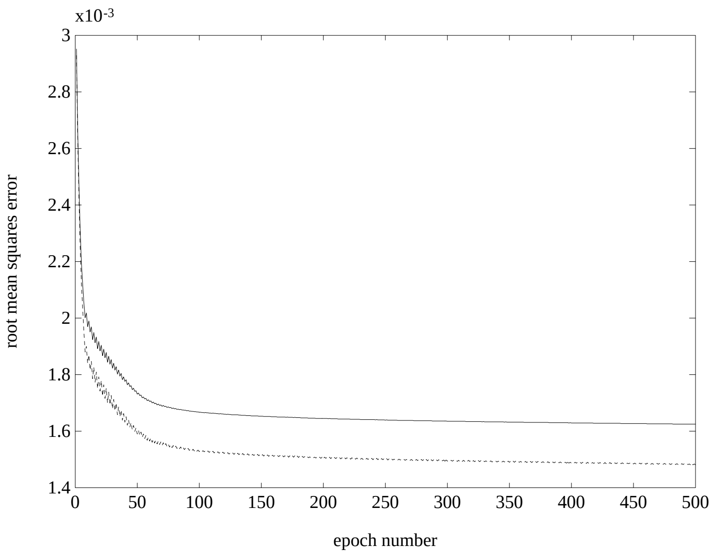


Figure 22: *Training and checking RMSE curves for ANFIS modeling.*

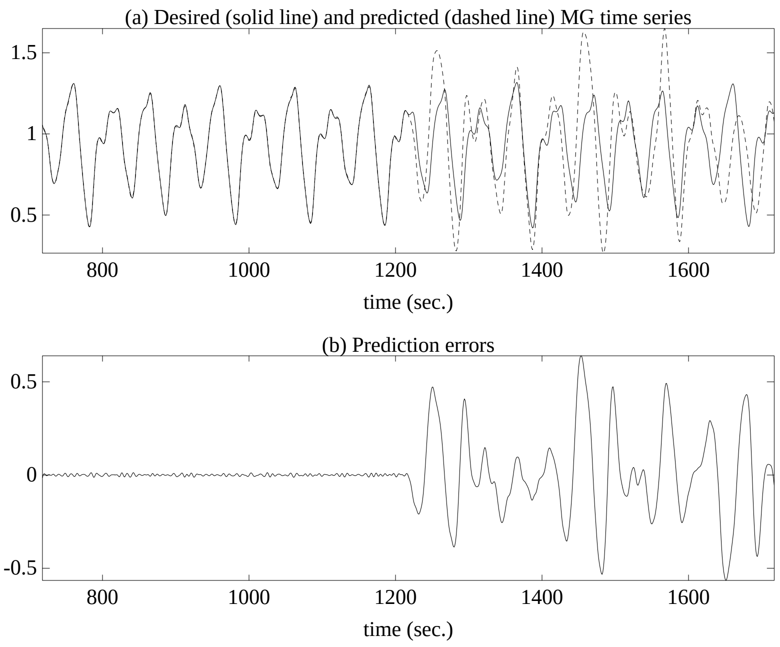


Figure 23: (a) Mackey-Glass time series (solid line) from $t = 718$ to 1717 and six-step ahead prediction (dashed line) by AR model with parameter = 104; (b) prediction errors.

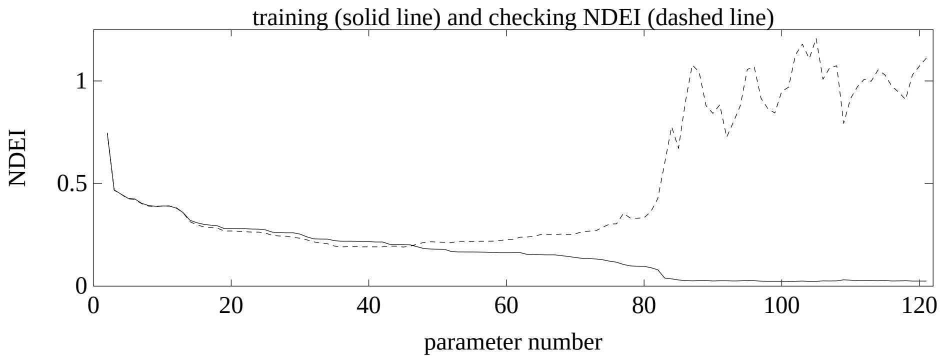


Figure 24: *Training (solid line) and checking (dashed line) errors of AR models with different parameter numbers.*

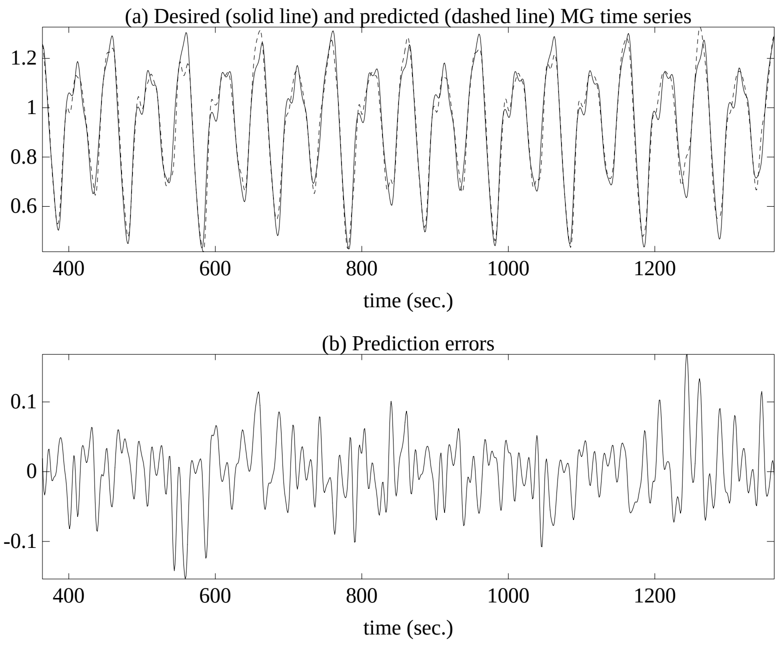


Figure 25: Example 3, (a) Mackey-Glass time series (solid line) from $t = 364$ to 1363 and six-step ahead prediction (dashed line) by the best AR model (parameter number = 45); (b) prediction errors.

It goes with saying that the nonlinear ANFIS outperforms the linear AR model. However, it should be noted that the identification of the AR model took only a few seconds, while the ANFIS simulation took about 2.5 hours on a HP Apollo 700 Series workstation. (We did not pay special attention on the optimization of the codes, though.)

Table 4 lists other methods' generalization capabilities which are measured by using each method to predict 500 points immediately following the training set. Here the *non-dimensional error index* (NDEI) [21, 35] is defined as the root mean square error divided by the standard deviation of the target series. (Note that the *average relative variance* used in [55, 56] is equal to the square of NDEI.) The remarkable generalization capability of the ANFIS, we believe, comes from the following facts:

1. The ANFIS can achieve a highly nonlinear mapping as shown in Example 1, 2 and 3, therefore it is superior to common linear methods in reproducing nonlinear time series.
2. The ANFIS used here has 104 adjustable parameters, much less than those of the cascade-correlation NN (693, the median size) and back-prop NN (about 540) listed in Table 4.
3. Though without *a priori* knowledge, the initial parameter settings of ANFIS are intuitively reasonable and it leads to fast learning that captures the underlying dynamics.

Method	Training Cases	Non-Dimensional Error Index
ANFIS	500	0.007
AR Model	500	0.19
Cascaded-Correlation NN	500	0.06
Back-Prop NN	500	0.02
6th-order Polynomial	500	0.04
Linear Predictive Method	2000	0.55

Table 4: *Generalization result comparisons for $P = 6$. (The last four rows are from [35].)*

Table 5 lists the results of the more challenging generalization test when $P = 84$ (the first six rows) and $P = 85$ (the last four rows). The results of the first six rows were obtained by iterating the prediction of $P = 6$ till $P = 84$. ANFIS still outperforms these statistical and connectionist approaches unless a substantially large amount of training data (i.e., the last row of Table 5) were used instead. Figure 26 illustrates the generalization test for the ANFIS where the first 500 points were used for the desired outputs while the last 500 are the predicted outputs for $P = 84$.

Method	Training Cases	Non-Dimensional Error Index
ANFIS	500	0.036
AR Model	500	0.39
Cascaded-Correlation NN	500	0.32
Back-Prop NN	500	0.05
6th-order Polynomial	500	0.85
Linear Predictive Method	2000	0.60
LRF	500	0.10-0.25
LRF	10000	0.025-0.05
MRH	500	0.05
MRH	10000	0.02

Table 5: *Generalization result comparisons for $P = 84$ (the first six rows) and 85 (the last four rows). Results for the first six methods are generated by iterating the solution at $P = 6$. Results for localized receptive fields (LRF) are multi-resolution hierarchies (MRH) are for networks trained for $P = 85$. (The last eight rows are from [35].)*

6 Concluding Remarks

A. Summary and Extensions of Current work

We have described the architecture of adaptive-network-based fuzzy inference systems (ANFIS) with type-1 and type-3 reasoning mechanisms. By employing a hybrid learning procedure, the proposed architecture can refine fuzzy if-then rules obtained from human experts to describe the input-output behavior of a complex system. However, if human expertise is not available, we can still set up intuitively reasonable initial membership functions and start the learning process to generate a set of fuzzy if-then rules to approximate a desired data set, as shown in the simulation examples of nonlinear function modeling and chaotic time series prediction.

Due to the high flexibility of adaptive networks, the ANFIS can have a number of variants from what we have proposed here. For instance, the membership functions can be changed to *L-R representation* [4] which could be asymmetric, Furthermore, we can replace Π nodes in layer 2 with the parameterized *T-norm* [4] and let the learning rule to decide the best T-norm operator for a specific application. By employing the adaptive network as a common framework, we have also proposed other adaptive fuzzy models tailored for data classification [45, 46] and feature extraction [47] purposes.

Another important issue in the training of ANFIS is how to preserve the human-plausible features such as bell-shaped membership functions, ϵ-completeness [22, 23] or sufficient overlapping

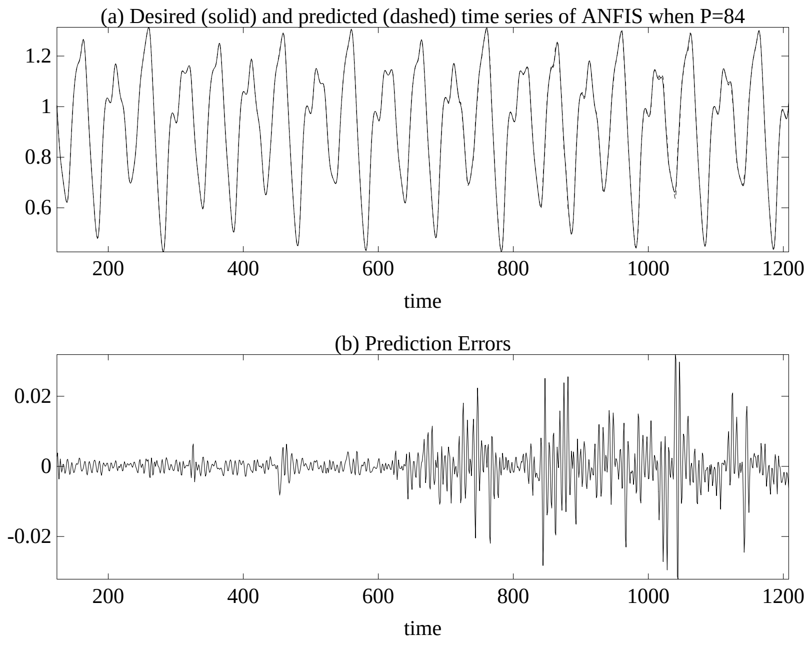


Figure 26: *Generalization test of ANFIS for* $P = 84$.

between adjacent membership functions, minimal uncertainty, etc. Though we did not pursue along this direction in this paper, mostly it can be achieved by maintaining certain constraints and/or modifying the original error measure as explained below.

- To keep bell-shaped membership functions, we need the membership functions to be bell-shaped regardless of the parameter values. In particular, equation (18) and equation (19) become up-side-down bell-shaped if $b_i < 0$; one easy way to correct this is to replace b_i with b_i^2 in both equations.

- The ϵ-completeness can be maintained by the constrained gradient descent [61]. For instance, suppose that $\epsilon = 0.5$ and the adjacent membership functions are of the form of equation (18) with parameter sets $\{a_i, b_i, c_i\}$ and $\{a_{i+1}, b_{i+1}, c_{i+1}\}$. Then the ϵ-completeness is satisfied if $c_i + a_i = c_{i+1} - a_{i+1}$ and this can be ensured throughout the training if the constrained gradient descent is employed.

- Minimal uncertainty refers to the situation that within most region of the input space, there should be a dominant fuzzy if-then rule to account for the final output, instead of multiple rules with similar firing strengths. This minimizes the uncertainty and make the rule set more informative. One way to do this is to use a modified error measure

$$E' = E + \sum_{i=1}^{P} [-\overline{w}_i \times ln(\overline{w}_i)], \tag{38}$$

where E is the original squared error ;P is the size of training data set; $\overline{w}_i$ is the normalized firing strength of the i-th rule (see equation (21)) and $\sum_{i=1}^{P}[-\overline{w}_i \times ln(\overline{w}_i)]$ is the *information entropy*. Since this modified error measure is not based on data fitting along, the ANFIS thus trained can also have a potentially better generalization capability. The improvement of generalization by using an error measure based both data fitting and weight elimination has been reported in the neural network literature [55, 56].

It is conjectured that this modified error measure could even decrease the possibility of over-fitting since the error measure is not based on data fitting alone. The effects of a similar error measure on neural networks was reported in [55].

In this paper, we assume the structure of the ANFIS is fixed and the *parameter identification* is solved through the hybrid learning rule. However, to make the whole approach more complete, the *structure identification* [43, 44] (which concerns with the selection of an appropriate input-space partition style and the number of membership functions on each input, etc.) is equally important to the successful applications of ANFIS. Effective partition of the input space can decrease the rule number and thus increase the speed in both learning and application phases. Advances on neural networks' structure identification [6, 24] can shed some lights on this aspect.

B. Applications to Automatic Control and Signal Processing

Fuzzy control is by far the most successful applications of the fuzzy set theory and fuzzy inference systems. Due to the adaptive capability of ANFIS, its applications to adaptive control and learning control are immediate. Most of all, it can replace almost any neural networks in control systems to serve the same purposes. For instance, Narendra's pioneering work of using neural networks in adaptive control [31] can be all achieved similarly by ANFIS. Moreover, four of the generic designs (i.e., *supervised control*, *direct inverse control*, *neural adaptive control* and *back-propagation of utility*) of neural networks in control, as proposed by Werbos [58, 9], are also directly applicable schemes for ANFIS. Particularly we have employed a similar method of the *back-propagation through time* [32] or *unfolding in time* to achieve a self-learning fuzzy controller with four rules that can balance an inverted pendulum in an near-optimal manner [12]. It is expected that the advances of neural network techniques in control can promote those of ANFIS as well, and vice versa.

The active role of neural networks in signal processing [60, 20] also suggests similar applications of ANFIS. The nonlinearity and structured knowledge representation of ANFIS are the primary advantages over classical linear approaches in adaptive filtering [8] and adaptive signal processing [59], such as identification, inverse modeling, predictive coding, adaptive channel equalization, adaptive interference (noise or echo) canceling, etc.

Acknowledgement

The guidance and help of Professor Lotfi A. Zadeh and other members of the "fuzzy group" at UC Berkeley is gratefully acknowledged.

References

[1] K. J. Aström and B. Wittenmark. *Computer Controller Systems: Theory and Design.* Prentice-Hall, Inc., 1984.

[2] S. M. Botros and C. G. Atkeson. Generalization properties of radial basis functions. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 707–713. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[3] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. on Neural Networks*, 2(2):302–309, March 1991.

[4] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic press, New York, 1980.

[5] S. E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 38–51, Carnegic Mellon University, 1988.

[6] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, G. Hinton, and T. Sejnowski, editors, *Advances in Neural Information Processing Systems II.* Morgan Kaufmann, 1990.

[7] G. C. Goodwin and K. S. Sin. *Adaptive filtering prediction and control.* Prentice-Hall, ENglewood Cliffs, N.J., 1984.

[8] S. S. Haykin. *Adaptive filter theory.* Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1991.

[9] W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors. *Neural Networks for Control.* Massachusetts Institute of Technology, 1990.

[10] J.-S. Roger Jang. Fuzzy modeling using generalized neural networks and Kalman filter algorithm. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 762–767, July 1991.

[11] J.-S. Roger Jang. Rule extraction using generalized neural networks. In *Proc. of the 4th IFSA World Congress*, July 1991.

[12] J.-S. Roger Jang. Self-learning fuzzy controller based on temporal back-propagation. *IEEE Trans. on Neural Networks*, September 1992.

[13] J.-S. Roger Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. on Neural Networks*, 1992. (Forthcoming).

[14] R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, and P. S. Lewis. Function approximation and time series prediction with neural networks. In *Proc. of IEEE International Joint Conference on Neural Networks*, pages I–649–665, 1990.

[15] V. Kadirkamanathan, M. Niranjan, and F. Fallside. Sequential adaptation of radial basis function neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 721–727. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[16] A. Kandel. *Fuzzy expert systems.* Addison-Wesley, 1988.

[17] A. Kandel. *Fuzzy expert systems.* CRC Press, Boca Raton, FL, 1992.

[18] L. V. Kantorovich and G. P. Akilov. *Functional analysis.* Pergamon, Oxford, 2 edition, 1982.

[19] T. Kondo. Revised GMDH algorithm estimating degree of the complete polynomial. *Tran. of the Society of Instrument and Controrl Engineers*, 22(9):928–934, 1986. (Japanese).

[20] B. Kosko. *Neural networks for signal processing.* Prentice Hall, Englewood Ciffs, NJ, 1991.

[21] A. S. Lapedes and R. Farber. Nonlinear signal processing using neural networks: prediction and system modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, 1987.

[22] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 1. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):404–418, 1990.

[23] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 2. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):419–435, 1990.

[24] T.-C. Lee. *Structure level adaptation for artificial neural networks.* Kluwer Academic Publishers, 1991.

[25] L. Ljung. *System identification: theory for the user.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[26] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, July 1977.

[27] J. Moody. Fast learning in multi-resolution hierarchies. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, chapter 1, pages 29–39. Morgan Kaufmann, San Mateo, CA, 1989.

[28] J. Moody and C. Darken. Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School.* Carnegie Mellon University, Morgan Kaufmann Publishers, 1988.

[29] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[30] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, 1992.

[31] K. S. Narendra and K. Parthsarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4–27, 1990.

[32] D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, pages 18–23, April 1990.

[33] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order Hebbian learning. In *Proc. of IEEE International Conference on Neural Networks*, pages 593–600, 1987.

[34] W. Pedrycz. *Fuzzy control and fuzzy systems.* Wiley, New York, 1989.

[35] III R. S. Crowder 'Predicting the Mackey-Glass timeseries with cascade-correlation learning. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 117–123, Carnegic Mellon University, 1990.

[36] H. L. Royden. *Real analysis.* Macmillan, New York, 2 edition, 1968.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volum 1*, chapter 8, pages 318–362. The MIT Press, 1986.

[38] T. D. Sanger. A tree-structured adaptive network for function approximate in high-dimensional spaces. *IEEE Trans. on Neural Networks*, 2(2):285–293, March 1991.

[39] S. Shar and F. Palmieri. MEKA-a fast, local algorithm for training feedforward neural networks. In *Proc. of International Joint Conference on Neural Networks*, pages III 41–46, 1990.

[40] S. Singhal and L. Wu. Training multilayer perceptrons with the extended kalman algorithm. In David S. Touretzky, editor, *Advances in neural information processing systems I*, pages 133–140. Morgan Kaufmann Publishers, 1989.

[41] P. Strobach. *Linear prediction theory: a mathematical basis for adaptive systems.* Springer-Verlag, 1990.

[42] M. Sugeno, editor. *Industrial applications of fuzzy control.* Elsevier Science Pub. Co., 1985.

[43] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15–33, 1988.

[44] C.-T. Sun. Rulebase structure identificaiton in an adaptive network based fuzzy inference system. *IEEE Trans. on Neural Networks*, September 1992. (Forthcoming).

[45] C.-T Sun and J.-S. Roger Jang. Adaptive network based fuzzy classification. In *Proc. of the Japan-U.S.A. Symposium on Flexible Automation*, July 1992.

[46] C.-T. Sun and J.-S. Roger Jang. Fuzzy classification based on adaptive networks and genetic algorithms. *IEEE Trans. on Neural Networks*, 1992. (Submitted).

[47] C.-T. Sun and J.-S. Roger Jang. Plasma modeling: an application of fuzzy filters and neural networks. In *to be submitted to World Congress on Neural Networks*, 1993.

[48] H. Takagi and I. Hayashi. NN-driven fuzzy reasoning. *International Journal of Approximate Reasoning*, 5(3):191–212, 1991.

[49] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operator's control actions. *Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 55–60, July 1983.

[50] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Trans. on Systems, Man, and Cybernetics*, 15:116–132, 1985.

[51] Y. Tsukamoto. An approach to fuzzy reasoning method. In Madan M. Gupta, Rammohan K. Ragade, and Ronald R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*, pages 137–149. North-Holland, Amsterdam, 1979.

[52] L.-X. Wang. Fuzzy systems are universal approximators. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Diego, March 1992.

[53] L.-X. Wang and J. M. Mendel. Fuzzy basis function, universal approximation, and orthogonal least squares learning. *IEEE Trans. on Neural Networks*, September 1992. (Forthcoming).

[54] R. L. Watrous. Learning algorithms for connectionist network: applied gradient methods of nonlinear optimization. In *Proc. of IEEE International Conference on Neural Networks*, pages 619–627, 1991.

[55] A. A. Weigend, D. E. Rumelhart, and B. A. Huberman. Back-propagation, weight-elimination and time series prediction. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 105–116, Carnegic Mellon University, 1990.

[56] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems III*, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.

[57] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[58] P. J. Werbos. An overview of neural networks for control. *IEEE Control Systems Magazine*, 11(1):40–41, January 1991.

[59] B. Widrow and D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1985.

[60] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, pages 25–39, March 1988.

[61] D. A. Wismer and R. Chattergy. *Introduction to nonlinear optimization: a problem solving approach*, chapter 6, pages 139–162. North-Holland Publishing Company, 1978.

[62] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[63] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. on Systems, Man, and Cybernetics*, 3(1):28–44, January 1973.

1

1

Neuro-Fuzzy Modeling and Control

Jyh-Shing Roger Jang, Chuen-Tsai Sun

Abstract— **Fundamental and advanced developments in neuro-fuzzy synergisms for modeling and control are reviewed. The essential part of neuro-fuzzy synergisms comes from a common framework called adaptive networks, which unifies both neural networks and fuzzy models. The fuzzy models under the framework of adaptive networks is called ANFIS (Adaptive-Network-based Fuzzy Inference System), which possess certain advantages over neural networks. We introduce the design methods for ANFIS in both modeling and control applications. Current problems and future directions for neuro-fuzzy approaches are also addressed.**

Keywords— **Fuzzy logic, neural networks, fuzzy modeling, neuro-fuzzy modeling, neuro-fuzzy control, ANFIS.**

I. Introduction

In 1965, Zadeh published the first paper on a novel way of characterizing non-probabilistic uncertainties, which he called **fuzzy sets** [118]. This year marks the 30th anniversary of fuzzy logic and fuzzy set theory, which has now evolved into a fruitful area containing various disciplines, such as calculus of fuzzy if-then rules, fuzzy graphs, fuzzy interpolation, fuzzy topology, fuzzy reasoning, fuzzy inferences systems, and fuzzy modeling. The applications, which are multi-disciplinary in nature, includes automatic control, consumer electronics, signal processing, time-series prediction, information retrieval, database management, computer vision, data classification, decision-making, and so on.

Recently, the resurgence of interest in the field of artificial neural networks has injected a new driving force into the "fuzzy" literature. The back-propagation learning rule, which drew little attention till its applications to artificial neural networks was discovered, is actually an universal learning paradigm for any smooth parameterized models, including fuzzy inference systems (or fuzzy models). As a result, a fuzzy inference system can now not only take linguistic information (linguistic rules) from human experts, but also adapt itself using numerical data (input/output pairs) to achieve better performance. This gives fuzzy inference systems an edge over neural networks, which cannot take linguistic information directly.

In this paper, we formalize the adaptive networks as a universal representation for any parameterized models. Under this common framework, we reexamine back-propagation algorithm and propose speedup schemes utilizing the least-squared method. We explain why neural networks and fuzzy inference systems are all special instances of adaptive networks when proper node functions are assigned, and all learning schemes applicable to adaptive networks are also qualified methods for neural networks and fuzzy inference systems.

When represented as an adaptive network, a fuzzy inference system is called ANFIS (Adaptive-Networks-based Fuzzy Inference Systems). For three of the most commonly used fuzzy inference systems, the equivalent ANFIS can be derived directly. Moreover, the training of ANFIS follows the spirit of the **minimum disturbance principle** [111] and is thus more efficient than sigmoidal neural networks.

Once a fuzzy inference system is equipped with learning capability, all the design methodologies for neural network controllers become directly applicable to fuzzy controllers. We briefly review these design techniques and give related references for further studies.

The arrangement of this article is as follows. In Section 2, an in-depth introduction to the basic concepts of fuzzy sets, fuzzy reasoning, fuzzy if-then rules, and fuzzy inference systems are given. Section 3 is devoted to the formalization of adaptive networks and their learning rules, where the back-propagation neural network and radial basis function network are included as special cases. Section 4 explains the ANFIS architecture and demonstrates its superiority over back-propagation neural networks. A number of design techniques for fuzzy and neural controllers is described in Section 5. Section 6 concludes this paper by pointing out current problems and future directions.

II. Fuzzy Sets, Fuzzy Rules, Fuzzy Reasoning, and Fuzzy Models

This section provides a concise introduction to and a summary of the basic concepts central to the study of fuzzy sets. Detailed treatments of specific subjects can be found in the reference list.

A. Fuzzy Sets

A classical set is a set with a crisp boundary. For example, a classical set A can be expressed as

$$A = \{x \mid x > 6\}, \tag{1}$$

where there is a clear, unambiguous boundary point 6 such that if x is greater than this number, then x belongs to the set A, otherwise x does not belong to this set. In contrast to a classical set, a **fuzzy set**, as the name implies, is a set without a crisp boundary. That is, the transition from "belonging to a set" to "not belonging to a set" is gradual, and this smooth transition is characterized by **membership functions** that give fuzzy sets flexibility in modeling commonly used linguistic expressions, such as "the water is hot" or "the temperature is high." As Zadeh pointed out in 1965 in his seminal paper entitled "Fuzzy Sets" [118],

This paper is to appear in the Proceedings of the IEEE, March 1995

Jyh-Shing Roger Jang is with the Control and Simulation Group, The MathWorks, Inc., Natick, Massachusetts. Email: jang@mathworks.com.

Chuen-Tsai Sun is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. Email: ctsun@cis.nctu.edu.tw.

such imprecisely defined sets or classes "play an important role in human thinking, particularly in the domains of pattern recognition, communication of information, and abstraction." Note that the fuzziness does not come from the randomness of the constituent members of the sets, but from the uncertain and imprecise nature of abstract thoughts and concepts.

Definition 1: Fuzzy sets and membership functions
If X is a collection of objects denoted generically by x, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{2}$$

$\mu_A(x)$ is called the membership function (MF for short) of x in A. The MF maps each element of X to a continuous membership value (or membership grade) between 0 and 1.

□

Obviously the definition of a fuzzy set is a simple extension of the definition of a classical set in which the characteristic function is permitted to have continuous values between 0 and 1. If the value of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classical set and $\mu_A(x)$ is the characteristic function of A.

Usually X is referred to as the **universe of discourse**, or simply the **universe**, and it may contain either discrete objects or continuous values. Two examples are given below.

Example 1: Fuzzy sets with discrete X
Let $X = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\}$ be the set of numbers of courses a student may take in a semester. Then the fuzzy set A = "appropriate number of courses taken" may be described as follows:

$$A = \{(1, 0.1), (2, 0.3), (3, 0.8), (4, 1), (5, 0.9), (6, 0.5), (7, 0.2), (8, 0.1)\}.$$

This fuzzy set is shown in Figure 1 (a).

□

Example 2: Fuzzy sets with continuous X
Let $X = R^+$ be the set of possibles ages for human beings. Then the fuzzy set B = "about 50 years old" may be expressed as

$$B = \{(x, \mu_B(x) | x \in X\},$$

where

$$\mu_B(x) = \frac{1}{1 + \left(\dfrac{x-50}{5}\right)^4}.$$

This is illustrated in Figure 1 (b).

□

An alternative way of denoting a fuzzy set A is

$$A = \begin{cases} \sum_{x_i \in X} \mu_A(x_i)/x_i, & \text{if } X \text{ is discrete.} \\ \int_X \mu_A(x)/x, & \text{if } X \text{ is continuous.} \end{cases} \tag{3}$$

The summation and integration signs in equation (3) stand for the union of $(x, \mu_A(x))$ pairs; they do not indicate summation or integration. Similarly, "/" is only a marker and does not imply division. Using this notation, we can rewrite the fuzzy sets in examples 1 and 2 as

$$A = 0.1/1 + 0.3/2 + 0.8/3 + 1.0/4 + 0.9/5 + 0.5/6 + 0.2/7 + 0.1/8,$$

and

$$B = \int_{R^+} \frac{1}{1 + (\frac{x-50}{5})^4} \Big/ x,$$

respectively.

From example 1 and 2, we see that the construction of a fuzzy set depends on two things: the identification of a suitable universe of discourse and the specification of an appropriate membership function. It should be noted that the specification of membership functions is quite *subjective*, which means the membership functions specified for the same concept (say, "cold") by different persons may vary considerably. This subjectivity comes from the indefinite nature of abstract concepts and has nothing to do with randomness. Therefore the *subjectivity* and *non-randomness* of fuzzy sets is the primary difference between the study of fuzzy sets and probability theory, which deals with objective treatment of random phenomena.

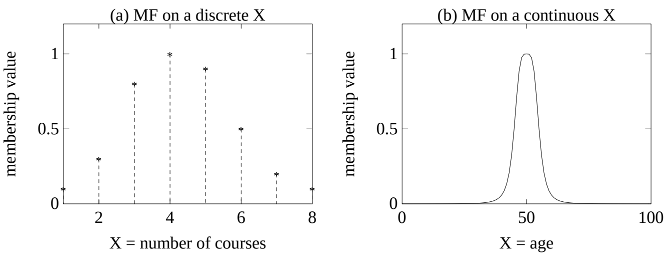


Fig. 1. *(a) A = "appropriate number of courses taken"; (b) B = "about 50 years old".*

Corresponding to the ordinary set operations of union, intersection, and complement, fuzzy sets have similar operations, which were initially defined in Zadeh's paper [118]. Before introducing these three fuzzy set operations, first we will define the notion of containment, which plays a central role in both ordinary and fuzzy sets. This definition of containment is, of course, a natural extension of the case for ordinary sets.

Definition 2: Containment or subset
Fuzzy set A is **contained** in fuzzy set B (or, equivalently, A is a **subset** of B, or A is smaller than or equal to B) if and only if $\mu_A(x) \le \mu_B(x)$ for all x. In symbols,

$$A \subseteq B \Longleftrightarrow \mu_A(x) \le \mu_B(x). \tag{4}$$

□

Definition 3: Union (disjunction)
The **union** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cup B$ or $C = A \quad OR \quad B$, whose MF is related to those of A and B by

$$\mu_C(x) = max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x). \tag{5}$$

□

As pointed out by Zadeh [118], a more intuitive and appealing definition of union is the smallest fuzzy set containing both A and B. Alternatively, if D is any fuzzy set that contains both A and B, then it also contains $A \cup B$. The intersection of fuzzy sets can be defined analogously.

Definition 4: Intersection (conjunction)
The **intersection** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cap B$ or $C = A \;\; AND \;\; B$, whose MF is related to those of A and B by

$$\mu_C(x) = min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x). \quad (6)$$

□

As in the case of the union, it is obvious that the intersection of A and B is the largest fuzzy set which is contained in both A and B. This reduces to the ordinary intersection operation if both A and B are nonfuzzy.

Definition 5: Complement (negation)
The **complement** of fuzzy set A, denoted by $\overline{A}$ ($\neg A$, $NOT\ A$), is defined as

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x). \quad (7)$$

□

Figure 2 demonstrates these three basic operations: (a) illustrates two fuzzy sets A and B, (b) is the complement of A, (c) is the union of A and B, and (d) is the intersection of A and B.

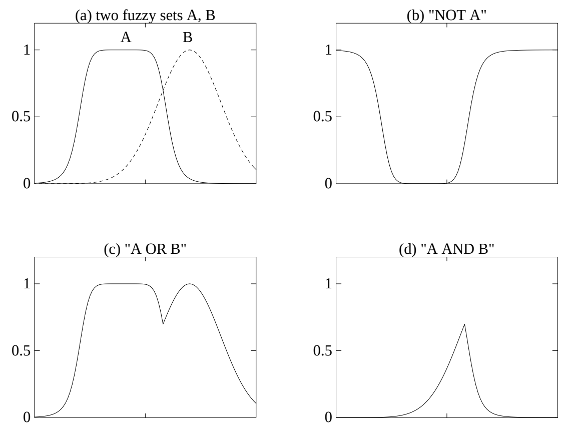


Fig. 2. *Operations on fuzzy sets: (a) two fuzzy sets A and B; (b) $\overline{A}$; (c) $A \cup B$; (d) $A \cap B$.*

Note that other consistent definitions for fuzzy AND and OR have been proposed in the literature under the names **T-norm** and **T-conorm** operators [16], respectively. Except for *min* and *max*, none of these operators satisfy the law of distributivity:

$$\begin{array}{rcl} \mu_{A\cup(B\cap C)}(x) & = & \mu_{(A\cup B)\cap(A\cup C)}(x), \\ \mu_{A\cap(B\cup C)}(x) & = & \mu_{(A\cap B)\cup(A\cap C)}(x). \end{array}$$

However, *min* and *max* do incur some difficulties in analyzing fuzzy inference systems. A popular alternative is to use the probabilistic AND and OR:

$$\begin{array}{rcl} \mu_{A\cap B}(x) & = & \mu_A(x)\mu_B(x). \\ \mu_{A\cup B}(x) & = & \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x). \end{array}$$

In the following, we shall give several classes of parameterized functions commonly used to define MF's. These parameterized MF's play an important role in adaptive fuzzy inference systems.

Definition 6: Triangular MF's
A **triangular MF** is specified by three parameters $\{a, b, c\}$, which determine the x coordinates of three corners:

$$triangle(x; a, b, c) = max\left(min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right). \quad (8)$$

Figure 3 (a) illustrates an example of the triangular MF defined by $triangle(x; 20, 60, 80)$.

□

Definition 7: Trapezoidal MF's
A **trapezoidal MF** is specified by four parameters $\{a, b, c, d\}$ as follows:

$$trapezoid(x; a, b, c, d) = max\left(min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right) \quad (9)$$

Figure 3 (b) illustrates an example of a trapezoidal MF defined by $trapezoid(x; 10, 20, 60, 95)$. Obviously, the triangular MF is a special case of the trapezoidal MF.

□

Due to their simple formulas and computational efficiency, both triangular MF's and trapezoidal MF's have been used extensively, especially in real-time implementations. However, since the MF's are composed of straight line segments, they are not smooth at the switching points specified by the parameters. In the following we introduce other types of MF's defined by smooth and nonlinear functions.

Definition 8: Gaussian MF's
A **Gaussian MF** is specified by two parameters $\{\sigma, c\}$:

$$gaussian(x; \sigma, c) = e^{-\left(\frac{x-c}{\sigma}\right)^2}, \quad (10)$$

where c represents the MF's center and σ determines the MF's width. Figure 3 (c) plots a Gaussian MF defined by $gaussian(x; 20, 50)$.

□

Definition 9: Generalized bell MF's
A **generalized bell MF** (or **bell MF**) is specified by three parameters $\{a, b, c\}$:

$$bell(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}, \quad (11)$$

where the parameter b is usually positive. Note that this MF is a direct generalization of the Cauchy distribution

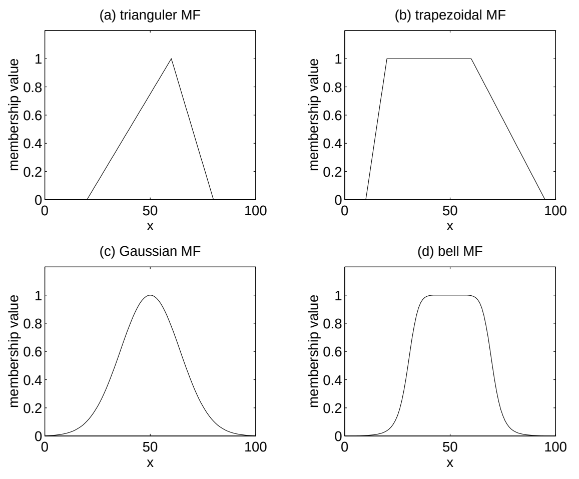


Fig. 3. *Examples of various classes of MF's: (a) triangle*$(x; 20, 60, 80)$*; (b) trapezoid*$(x; 10, 20, 60, 95)$*; (c) gaussian*$(x; 20, 50)$*; (d) bell*$(x; 20, 4, 50)$.

used in probability theory. Figure 3 illustrates a generalized bell MF defined by $bell(x; 20, 4, 50)$.

□

A desired generalized bell MF can be obtained by a proper selection of the parameter set $\{a, b, c\}$. Specifically, we can adjust c and a to vary the center and width of the MF, and then use b to control the slopes at the crossover points. Figure 4 shows the physical meanings of each parameter in a bell MF.

Because of their smoothness and concise notation, Gaussian MF's and bell MF's are becoming increasingly popular methods for specifying fuzzy sets. Gaussian functions are well known in the fields of probability and statistics, and they possess useful properties such as invariance under multiplication and Fourier transform. The bell MF has one more parameter than the Gaussian MF, so it can approach a nonfuzzy set if $b \to \infty$.

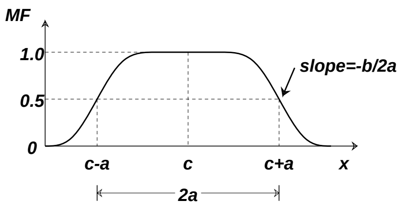


Fig. 4. *Physical meaning of parameters in a generalized bell function.*

Definition 10: Sigmoidal MF's
A **sigmoidal MF** is defined by

$$sigmoid(x; a, c) = \frac{1}{1 + exp[-a(x - c)]}, \tag{12}$$

where a controls the slope at the crossover point $x = c$.

□

Depending on the sign of the parameter a, a sigmoidal MF is inherently open right or left and thus is appropriate for representing concepts such as "very large" or "very negative." Sigmoidal functions of this kind are employed widely as the activation function of artificial neural networks. Therefore, for a neural network to simulate the behavior of a fuzzy inference system, the first problem we face is how to synthesize a close MF through a sigmoidal function. There are two simple ways to achieve this: one is to take the product of two sigmoidal MF's; the other is to take the absolute difference of two sigmoidal MF's.

It should be noted that the list of MF's introduced in this section is by no means exhaustive; other specialized MF's can be created for specific applications if necessary. In particular, any types of continuous probability distribution functions can be used as an MF here, provided that a set of parameters are given to specify the appropriate meanings of the MF.

B. Fuzzy If-Then Rules

A **fuzzy if-then rule** (**fuzzy rule**, **fuzzy implication** or **fuzzy conditional statement**) assumes the form

$$\textit{if } x \textit{ is } A \textit{ then } y \textit{ is } B, \tag{13}$$

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y, respectively. Often "*x is A*" is called the **antecedent** or **premise** while "*y is B*" is called the **consequence** or **conclusion**. Examples of fuzzy if-then rules are widespread in our daily linguistic expressions, such as the following:

- If pressure is high then volume is small.
- If the road is slippery then driving is dangerous.
- If a tomato is red then it is ripe.
- If the speed is high then apply the brake a little.

Before we can employ fuzzy if-then rules to model and analyze a system, we first have to formalize what is meant by the expression "*if x is A then y is B*", which is sometimes abbreviated as $A \to B$. In essence, the expression describes a relation between two variables x and y; this suggests that a fuzzy if-then rule be defined as a **binary fuzzy relation** R on the product space $X \times Y$. Note that a binary fuzzy relation R is an extension of the classical Cartesian product, where each element $(x, y) \in X \times Y$ is associated with a membership grade denoted by $\mu_R(x, y)$. Alternatively, a binary fuzzy relation R can be viewed as a fuzzy set with universe $X \times Y$, and this fuzzy set is characterized by a **two-dimensional MF** $\mu_R(x, y)$.

Generally speaking, there are two ways to interpret the fuzzy rule $A \to B$. If we interpret $A \to B$ as **A coupled with B**, then

$$R = A \to B = A \times B = \int_{X \times Y} \mu_A(x) \tilde{*} \mu_B(y)/(x, y),$$

where $\tilde{*}$ is a fuzzy AND (or more generally, T-norm) operator and $A \to B$ is used again to represent the fuzzy relation R. On the other hand, if $A \to B$ is interpreted as **A entails B**, then it can be written as four different formulas:

- Material implication: $R = A \to B = \neg A \cup B$.
- Propositional calculus: $R = A \to B = \neg A \cup (A \cap B)$.

- Extended propositional calculus: $R = A \to B = (\neg A \cap \neg B) \cup B$.
- Generalization of modus ponens: $\mu_R(x,y) = sup\{c \mid \mu_A(x) \tilde{*} c \le \mu_B(y)$ *and* $0 \le c \le 1\}$, where $R = A \to B$ and $\tilde{*}$ is a T-norm operator.

Though these four formulas are different in appearance, they all reduce to the familiar identity $A \to B \equiv \neg A \cup B$ when A and B are propositions in the sense of two-valued logic. Figure 5 illustrates these two interpretations of a fuzzy rule $A \to B$. Here we shall adopt the first interpretation, where $A \to B$ implies **A coupled with B**. The treatment of the second interpretation can be found in [34], [49], [50].

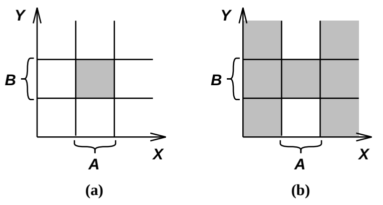


Fig. 5. *Two interpretations of fuzzy implication: (a) A coupled with B; (b) A entails B..*

C. Fuzzy Reasoning (Approximate Reasoning)

Fuzzy reasoning (also known as **approximate reasoning**) is an inference procedure used to derive conclusions from a set of fuzzy if-then rules and one or more conditions. Before introducing fuzzy reasoning, we shall discuss the **compositional rule of inference** [119], which is the essential rationale behind fuzzy reasoning.

The compositional rule of inference is a generalization of the following familiar notion. Suppose that we have a curve $y = f(x)$ that regulates the relation between x and y. When we are given $x = a$, then from $y = f(x)$ we can infer that $y = b = f(a)$; see Figure 6 (a). A generalization of the above process would allow a to be an interval and $f(x)$ to be an interval-valued function, as shown in Figure 6 (b). To find the resulting interval $y = b$ corresponding to the interval $x = a$, we first construct a cylindrical extension of a (that is, extend the domain of a from X to $X \times Y$) and then find its intersection I with the interval-valued curve. The projection of I onto the y-axis yields the interval $y = b$.

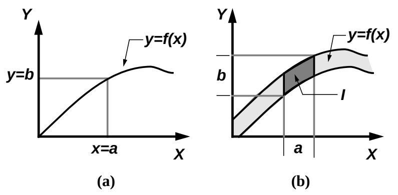


Fig. 6. *Derivation of $y = b$ from $x = a$ and $y = f(x)$: (a) a and b are points, $y = f(x)$ is a curve; (b) a and b are intervals, $y = f(x)$ is an interval-valued function.*

Going one step further in our generalization, we assume that A is a fuzzy set of X and F is a fuzzy relation on $X \times Y$, as shown in Figure 7 (a) and (b). To find the resulting fuzzy set B, again, we construct a cylindrical extension $c(A)$ with base A (that is, we expand the domain of A from X to $X \times Y$ to get $c(A)$). The intersection of $c(A)$ and F (Figure 7 (c)) forms the analog of the region of intersection I in Figure 6 (b). By projecting $c(A) \cap F$ onto the y-axis, we infer y as a fuzzy set B on the y-axis, as shown in Figure 7 (d).

Specifically, let μ_A, $\mu_{c(A)}$, μ_B, and μ_F be the MF's of A, $c(A)$, B, and F, respectively, where $\mu_{c(A)}$ is related to μ_A through

$$\mu_{c(A)}(x,y) = \mu_A(x).$$

Then

$$\begin{aligned}\mu_{c(A)\cap F}(x,y) &= \min[\mu_{c(A)}(x,y), \mu_F(x,y)] \\ &= \min[\mu_A(x), \mu_F(x,y)].\end{aligned}$$

By projecting $c(A) \cap F$ onto the y-axis, we have

$$\begin{aligned}\mu_B(y) &= \max_x \min[\mu_A(x), \mu_F(x,y)] \\ &= \vee_x[\mu_A(x) \wedge \mu_F(x,y)].\end{aligned}$$

This formula is referred to as **max-min composition** and B is represented as

$$B = A \circ F,$$

where $\circ$ denotes the composition operator. If we choose product for fuzzy AND and *max* for fuzzy OR, then we have **max-product composition** and $\mu_B(y)$ is equal $\vee_x[\mu_A(x)\mu_F(x,y)]$.

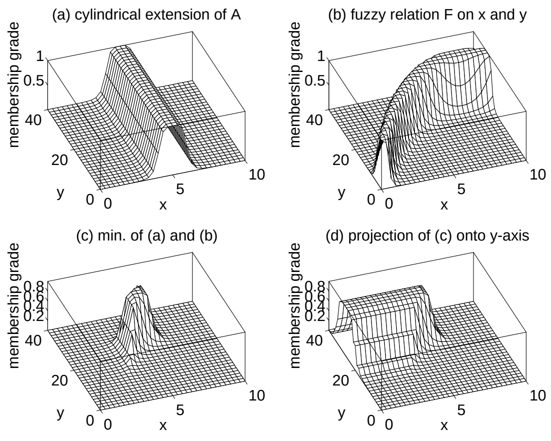


Fig. 7. *Compositional rule of inference.*

Using the compositional rule of inference, we can formalize an inference procedure, called fuzzy reasoning, upon a set of fuzzy if-then rules. The basic rule of inference in traditional two-valued logic is **modus ponens**, according to which we can infer the truth of a proposition B from the truth of A and the implication $A \to B$. For instance, if A is identified with "the tomato is red" and B with "the tomato is ripe," then if it is true that "the tomato is red," it is also true that "the tomato is ripe." This concept is illustrated below.

premise 1 (fact):	x is A,
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B.

However, in much of human reasoning, modus ponens is employed in an approximate manner. For example, if we have the same implication rule "if the tomato is red then it is ripe" and we know that "the tomato is more or less red," then we may infer that "the tomato is more or less ripe." This is written as

premise 1 (fact):	x is A',
premise 2 (rule):	if x is A then y is B,
consequence (conclusion):	y is B',

where A' is close to A and B' is close to B. When A, B, A', and B' are fuzzy sets of appropriate universes, the above inference procedure is called fuzzy reasoning or approximate reasoning; it is also called **generalized modus ponens**, since it has modus ponens as a special case.

Using the composition rule of inference introduced earlier, we can formulate the inference procedure of fuzzy reasoning as the following definition.

Definition 11: Fuzzy Reasoning Based On Max-Min Composition.

Let A, A', and B be fuzzy sets of X, X, and Y, respectively. Assume that the fuzzy implication $A \rightarrow B$ is expressed as a fuzzy relation R on $X \times Y$. Then the fuzzy set B' induced by "*x is A'*" and the fuzzy rule "*if x is A then y is B*" is defined by

$$\begin{aligned} \mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \mu_R(x,y)] \\ &= \vee_x[\mu_{A'}(x) \wedge \mu_R(x,y)], \end{aligned} \tag{14}$$

or, equivalently,

$$B' = A' \circ R = A' \circ (A \rightarrow B). \tag{15}$$

□

Remember that equation (15) is a general expression for fuzzy reasoning, while equation (14) is an instance of fuzzy reasoning where *min* and *max* are the operators for fuzzy AND and OR, respectively.

Now we can use the inference procedure of the generalized modus ponens to derive conclusions, provided that the fuzzy implication $A \rightarrow B$ is defined as an appropriate binary fuzzy relation.

C.1 Single rule with single antecedent

For a single rule with a single antecedent, the formula is available in equation (14). A further simplification of the equation yields

$$\begin{aligned} \mu_{B'}(y) &= [\vee_x(\mu_{A'}(x) \wedge \mu_A(x)] \wedge \mu_B(y) \\ &= w \wedge \mu_B(y). \end{aligned}$$

In other words, first we find the degree of match w as the maximum of $\mu_{A'}(x) \wedge \mu_A(x)$ (the shaded area in the antecedent part of Figure 8); then the MF of the resulting B' is equal to the MF of B clipped by w, shown as the shaded area in the consequent part of Figure 8.

A fuzzy if-then rule with two antecedents is usually written as "*if x is A and y is B then z is C*." The corresponding problem for approximate reasoning is expressed as

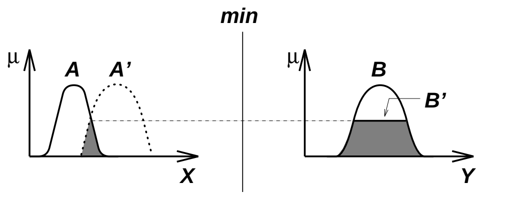


Fig. 8. *Fuzzy reasoning for a single rule with a single antecedent.*

premise 1 (fact):	x is A' and y is B',
premise 2 (rule):	if x is A and y is B then z is C,
consequence (conclusion):	z is C'.

The fuzzy rule in premise 2 above can be put into the simpler form "$A \times B \rightarrow C$." Intuitively, this fuzzy rule can be transformed into a ternary fuzzy relation R, which is specified by the following MF:

$$\mu_R(x,y,z) = \mu_{(A\times B)\times C}(x,y,z) = \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z).$$

And the resulting C' is expressed as

$$C' = (A' \times B') \circ (A \times B \rightarrow C).$$

Thus

$$\begin{aligned} \mu_{C'}(z) &= \vee_{x,y}[\mu_{A'}(x) \wedge \mu_{B'}(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)] \\ &= \vee_{x,y}\{[\mu_{A'}(x) \wedge \mu_{B'}(y) \wedge \mu_A(x) \wedge \mu_B(y)]\} \wedge \mu_C(z) \\ &= \underbrace{\{\vee_x[\mu_{A'}(x) \wedge \mu_A(x)]\}}_{w_1} \wedge \underbrace{\{\vee_y[\mu_{B'}(y) \wedge \mu_B(y)]\}}_{w_2} \wedge \mu_C(z) \\ &= \underbrace{w_1 \wedge w_2}_{\text{firing strength}} \wedge \mu_C(z), \end{aligned} \tag{16}$$

where w_1 is the degree of match between A and A'; w_2 is the degree of match between B and B'; and $w_1 \wedge w_2$ is called the **firing strength** or **degree of fulfillment** of this fuzzy rule. A graphic interpretation is shown in Figure 9, where the MF of the resulting C' is equal to the MF of C clipped by the firing strength w, $w = w_1 \wedge w_2$. The generalization to more than two antecedents is straightforward.

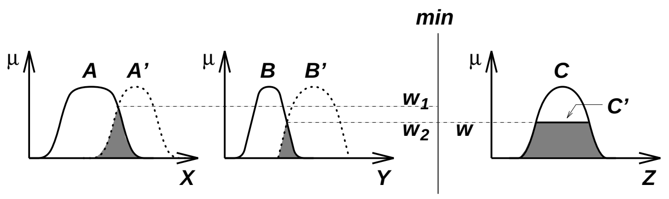


Fig. 9. *Approximate reasoning for multiple antecedents.*

C.2 Multiple rules with multiple antecedents

The interpretation of multiple rules is usually taken as the union of the fuzzy relations corresponding to the fuzzy rules. For instance, given the following fact and rules:

premise 1 (fact):	x is A' and y is B',
premise 2 (rule 1):	if x is A_1 and y is B_1 then z is C_1,
premise 3 (rule 2):	if x is A_2 and y is B_2 then z is C_2,
consequence (conclusion):	z is C',

we can employ the fuzzy reasoning shown in Figure 10 as an inference procedure to derive the resulting output fuzzy set C'.

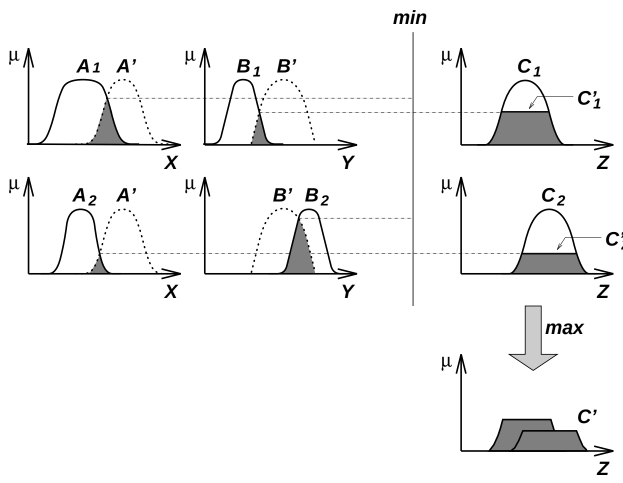


Fig. 10. *Fuzzy reasoning for multiple rules with multiple antecedents.*

To verify this inference procedure, let $R_1 = A_1 \times B_1 \rightarrow C_1$ and $R_2 = A_2 \times B_2 \rightarrow C_2$. Since the max-min composition operator $\circ$ is distributive over the $\cup$ operator, it follows that

$$\begin{aligned} C' &= (A' \times B') \circ (R_1 \cup R_2) \\ &= [(A' \times B') \circ R_1] \cup [(A' \times B') \circ R_2] \\ &= C_1' \cup C_2', \end{aligned} \tag{17}$$

where C_1' and C_2' are the inferred fuzzy sets for rule 1 and 2, respectively. Figure 10 shows graphically the operation of fuzzy reasoning for multiple rules with multiple antecedents.

When a given fuzzy rule assumes the form "if x is A or y is B then z is C," then firing strength is given as the maximum of degree of match on the antecedent part for a given condition. This fuzzy rule is equivalent to the union of the two fuzzy rules "if x is A then z is C" and "if y is B then z is C" if and only if the max-min composition is adopted.

D. Fuzzy Models (Fuzzy Inference Systems)

The **Fuzzy inference system** is a popular computing framework based on the concepts of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning. It has been successfully applied in fields such as automatic control, data classification, decision analysis, expert systems, and computer vision. Because of its multi-disciplinary nature, the fuzzy inference system is known by a number of names, such as **fuzzy-rule-based system**, **fuzzy expert system** [37], **fuzzy model** [98], [91], **fuzzy associative memory** [47], **fuzzy logic controller** [60], [49], [50], and simply (and ambiguously) **fuzzy system**.

The basic structure of a fuzzy inference system consists of three conceptual components: a **rule base**, which contains a selection of fuzzy rules, a **database** or **dictionary**, which defines the membership functions used in the fuzzy rules, and a **reasoning mechanism**, which performs the inference procedure (usually the fuzzy reasoning introduced earlier) upon the rules and a given condition to derive a reasonable output or conclusion.

Note that the basic fuzzy inference system can take either fuzzy inputs or crisp inputs (which can be viewed as fuzzy singletons that have zero membership grade everywhere except at certain points where the membership grades achieve unity), but the outputs it produces are almost always fuzzy sets. Often it is necessary to have a crisp output, especially in a situation where a fuzzy inference system is used as a controller. Therefore we need a **defuzzification strategy** to extract a crisp value that best summarize a fuzzy set. A fuzzy inference system with a crisp output is shown in Figure 11, where the dashed line indicates a basic fuzzy inference system with fuzzy output and the defuzzification block serves the purpose of transforming a fuzzy output into a crisp one. An example of a basic fuzzy inference system is the two-rule two-input system of Figure 10. The function of the defuzzification block will be explained at a later point.

With crisp inputs and outputs, a fuzzy inference system implements a nonlinear mapping from its input space to output space. This mapping is accomplished by a number of fuzzy if-then rules, each of which describes the local behavior of the mapping. In particular, the antecedent of each rule defines a fuzzy region of the input space, and the consequent specifies the corresponding outputs.

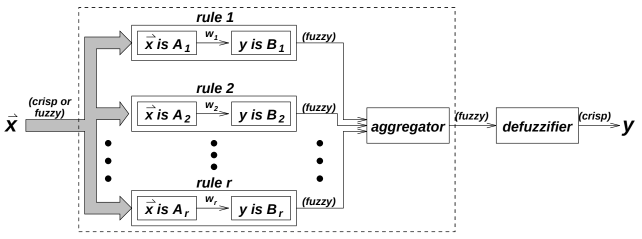


Fig. 11. *Block diagram for a fuzzy inference system.*

In what follows, we will first introduce three types of fuzzy inference systems that have been widely employed in various applications. The differences between these three fuzzy inference systems lie in the consequents of their fuzzy rules, and thus their aggregation and defuzzification procedures differ accordingly. Then we will introduce three ways of partitioning the input space for any type of fuzzy inference system. Last, we will address briefly the features and the problems of fuzzy modeling, which is concerned with the construction of a fuzzy inference system for modeling a specific target system.

D.1 Mamdani Fuzzy Model

The **Mamdani fuzzy model** [60] was proposed as the very first attempt to control a steam engine and boiler combination by a set of linguistic control rules obtained from experienced human operators. Figure 12 is an illustration of how a two-rule fuzzy inference system of the Mamdani type derives the overall output z when subjected to two crisp inputs x and y.

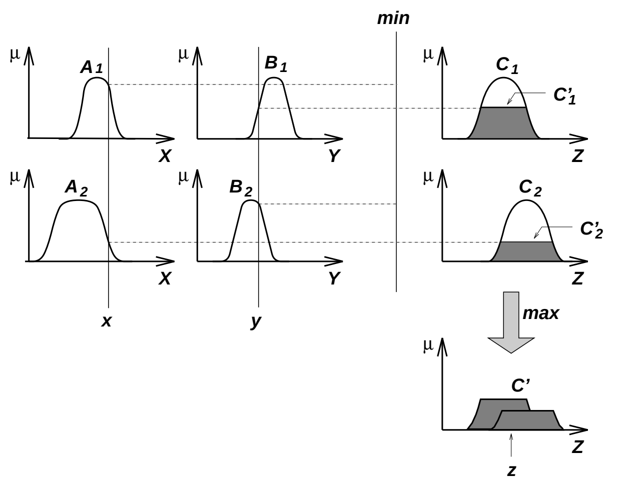


Fig. 12. *The Mamdani fuzzy inference system using min and max for fuzzy AND and OR operators, respectively.*

If we adopt product and *max* as our choice for the fuzzy AND and OR operators, respectively, and use max-product composition instead of the original max-min composition, then the resulting fuzzy reasoning is shown in Figure 13, where the inferred output of each rule is a fuzzy set scaled down by its firing strength via the algebraic product. Though this type of fuzzy reasoning was not employed in Mamdani's original paper, it has often been used in the literature. Other variations are possible if we have different choices of fuzzy AND (T-norm) and OR (T-conorm) operators.

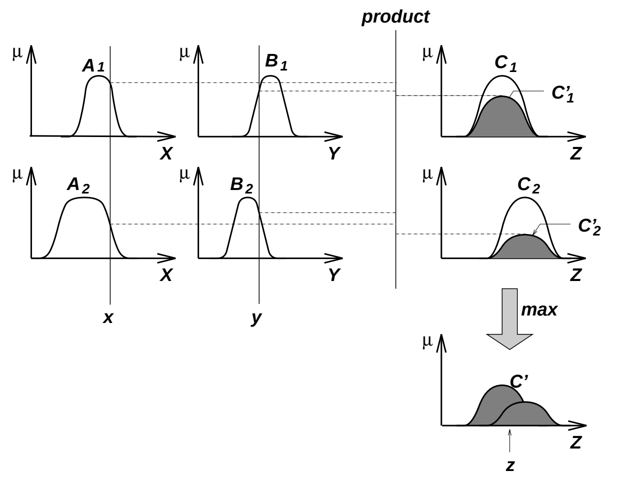


Fig. 13. *The Mamdani fuzzy inference system using product and max for fuzzy AND and OR operators, respectively.*

In Mamdani's application [60], two fuzzy inference systems were used as two controllers to generate the heat input to the boiler and throttle opening of the engine cylinder, respectively, in order to regulate the steam pressure in the boiler and the speed of the engine. Since the plant takes only crisp values as inputs, we have to use a defuzzifier to convert a fuzzy set to a crisp value. **Defuzzification** refers to the way a crisp value is extracted from a fuzzy set as a representative value. The most frequently used defuzzification strategy is the centroid of area, which is defined as

$$z_{COA} = \frac{\int_Z \mu_{C'}(z) z \, dz}{\int_Z \mu_{C'}(z) \, dz}, \tag{18}$$

where $\mu_{C'}(z)$ is the aggregated output MF. This formula is reminiscent of the calculation of expected values in probability distributions. Other defuzzification strategies arise for specific applications, which includes bisector of area, mean of maximum, largest of maximum, and smallest of maximum, and so on. Figure 14 demonstrate these defuzzification strategies. Generally speaking, these defuzzification methods are computation intensive and there is no rigorous way to analyze them except through experiment-based studies. Other more flexible defuzzification methods can be found in [73], [115], [80].

Both Figure 12 and 13 conform to the fuzzy reasoning defined previously. In practice, however, a fuzzy inference system may have certain reasoning mechanisms that do not follow the strict definition of the compositional rule of inference. For instance, one might use either *min* or product for computing firing strengths and/or qualified rule outputs. Another variation is to use pointwise summation (*sum*) instead of *max* in the standard fuzzy reasoning, though *sum* is not really a fuzzy OR operators. An advantage of this **sum-product composition** [47] is that the final crisp output via centroid defuzzification is equal to the weighted average of each rule's crisp output, where the weighting factor for a rule is equal to its firing strength multiplied by the area of the rule's output MF, and the crisp output of a rule is equal to the the centroid defuzzified value of its output MF. This reduces the computation burden if we can obtain the area and the centroid of each output MF in advance.

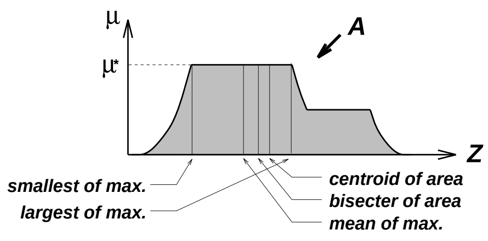


Fig. 14. *Various defuzzification schemes for obtaining a crisp output.*

D.2 Sugeno Fuzzy Model

The **Sugeno fuzzy model** (also known as the **TSK fuzzy model**) was proposed by Takagi, Sugeno, and

Kang [98], [91] in an effort to develop a systematic approach to generating fuzzy rules from a given input-output data set. A typical fuzzy rule in a Sugeno fuzzy model has the form

if x is A and y is B then z = f(x,y),

where A and B are fuzzy sets in the antecedent, while $z = f(x, y)$ is a crisp function in the consequent. Usually $f(x, y)$ is a polynomial in the input variables x and y, but it can be any function as long as it can appropriately describe the output of the system within the fuzzy region specified by the antecedent of the rule. When $f(x, y)$ is a first-order polynomial, the resulting fuzzy inference system is called a **first-order Sugeno fuzzy model**, which was originally proposed in [98], [91]. When f is a constant, we then have a **zero-order Sugeno fuzzy model**, which can be viewed either as a special case of the Mamdani fuzzy inference system, in which each rule's consequent is specified by a fuzzy singleton (or a pre-defuzzified consequent), or a special case of the Tsukamoto fuzzy model (to be introduce later), in which each rule's consequent is specified by an MF of a step function crossing at the constant. Moreover, a zero-order Sugeno fuzzy model is functionally equivalent to a radial basis function network under certain minor constraints [32].

It should be pointed out that the output of a zero-order Sugeno model is a smooth function of its input variables as long as the neighboring MF's in the premise have enough overlap. In other words, the overlap of MF's in the consequent does not have a decisive effect on the smoothness of the interpolation; it is the overlap of the MF's in the premise that determines the smoothness of the resulting input-output behavior.

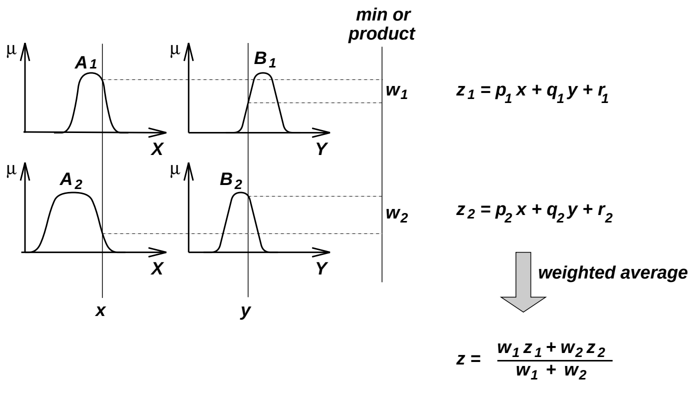


Fig. 15. *The Sugeno fuzzy model.*

Figure 15 shows the fuzzy reasoning procedure for a first-order Sugeno fuzzy model. Note that the aggregator and defuzzifier blocks in Figure 11 are replaced by the operation of **weighted average**, thus avoiding the time-consuming procedure of defuzzification. In practice, sometimes the weighted average operator is replaced with the **weighted sum** operator (that is, $z = w_1z_1 + w_2z_2$ in Figure 15) in order to further reduce computation load, especially in training a fuzzy inference system. However, this simplification could lead to the loss of MF linguistic meanings unless the sum of firing strengths (that is, $\sum_i w_i$) is close to unity.

D.3 Tsukamoto Fuzzy Model

In the **Tsukamoto fuzzy models** [101], the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonical MF, as shown in Figure 16. As a result, the inferred output of each rule is defined as a crisp value induced by the rule's firing strength. The overall output is taken as the weighted average of each rule's output. Figure 16 illustrates the whole reasoning procedure for a two-input two-rule system.

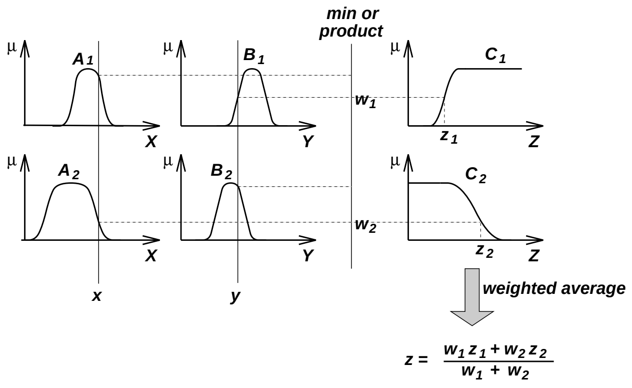


Fig. 16. *The Tsukamoto fuzzy model.*

Since each rule infers a crisp output, the Tsukamoto fuzzy model aggregates each rule's output by the method of weighted average and thus also avoids the time-consuming process of defuzzification.

D.4 Partition Styles for Fuzzy Models

By now it should be clear that the spirit of fuzzy inference systems resembles that of "divide and conquer" – the antecedents of fuzzy rules partition the input space into a number of local fuzzy regions, while the consequents describe the behavior within a given region via various constituents. The consequent constituent could be an output MF (Mamdani and Tsukamoto fuzzy models), a constant (zero-order Sugeno model), or a linear equation (first-order Sugeno model). Different consequent constituents result in different fuzzy inference systems, but their antecedents are always the same. Therefore the following discussion of methods of partitioning input spaces to form the antecedents of fuzzy rules is applicable to all three types of fuzzy inference systems.

- **Grid partition**: Figure 17 (a) illustrates a typical grid partition in a two-dimensional input space. This partition method is often chosen in designing a fuzzy controller, which usually involves only several state variables as the inputs to the controller. This partition strategy needs only a small number of MF's for each input. However, it encounters problems when we have a moderately large number of inputs. For instance, a fuzzy model with 10 inputs and two MF's on each input would result in $2^{10} = 1024$ fuzzy if-then rules, which is prohibitively large. This problem, usually referred to as the **curse of dimensionality**, can be alleviated by the other partition strategies introduced below.

- **Tree partition**: Figure 17 (b) shows a typical tree partition, in which each region can be uniquely specified along a corresponding decision tree. The tree partition relieves the problem of an exponential increase in the number of rules. However, more MF's for each input are needed to define these fuzzy regions, and these MF's do not usually bear clear linguistic meanings such as "small," "big," and so on.
- **Scatter partition**: As shown in Figure 17 (c), by covering a subset of the whole input space that characterizes a region of possible occurrence of the input vectors, the scatter partition can also limit the number of rules to a reasonable amount.

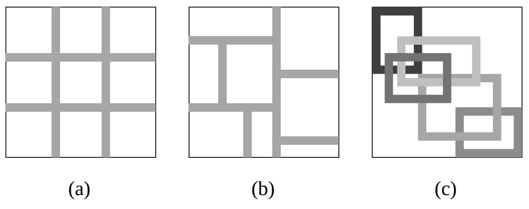


Fig. 17. *Various methods for partitioning the input space: (a) grid partition; (b) tree partition; (b) scatter partition.*

D.5 Neuro-Fuzzy Modeling

The process for constructing a fuzzy inference system is usually called **fuzzy modeling**, which has the following features:

- Due to the rule structure of a fuzzy inference system, it is easy to incorporate human expertise about the target system directly into the modeling process. Namely, fuzzy modeling takes advantage of **domain knowledge** that might not be easily or directly employed in other modeling approaches.
- When the input-output data of a system to be modeled is available, conventional system identification techniques can be used for fuzzy modeling. In other words, the use of **numerical data** also plays an important role in fuzzy modeling, just as in other mathematical modeling methods.

A common practice is to use domain knowledge for **structure determination** (that is, determine relevant inputs, number of MF's for each input, number of rules, types of fuzzy models, and so on) and numerical data for **parameter identification** (that is, identify the values of parameters that can generate best the performance). In particular, the term **neuro-fuzzy modeling** refers to the way of applying various learning techniques developed in the neural network literature to fuzzy inference systems. In the subsequent sections, we will apply the concept of the adaptive network, which is a generalization of the common back-propagation neural network, to tackle the parameter identification problem in a fuzzy inference system.

III. Adaptive Networks

This section describes the architectures and learning procedures of adaptive networks, which are a superset of all kinds of neural network paradigms with supervised learning capability. In particular, we shall address two of the most popular network paradigms adopted in the neural network literature: the back-propagation neural network (BPNN) and the radial basis function network (RBFN). Other network paradigms that can be interpreted as a set of fuzzy if-then rules are described in the next section.

A. Architecture

As the name implies, an **adaptive network** (Figure 18) is a network structure whose overall input-output behavior is determined by the values of a collection of modifiable parameters. More specifically, the configuration of an adaptive network is composed of a set of nodes connected through directed links, where each node is a process unit that performs a static **node function** on its incoming signals to generate a single **node output** and each link specifies the direction of signal flow from one node to another. Usually a node function is a parameterized function with modifiable parameters; by changing these parameters, we are actually changing the node function as well as the overall behavior of the adaptive network.

In the most general case, an adaptive network is heterogeneous and each node may have a different node function. Also remember that each link in an adaptive network are merely used to specify the propagation direction of a node's output; generally there are no weights or parameters associated with links. Figure 18 shows a typical adaptive network with two inputs and two outputs.

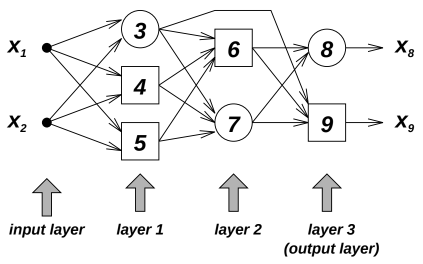


Fig. 18. *A feedforward adaptive network in layered representation.*

The parameters of an adaptive network are distributed into the network's nodes, so each node has a local parameter set. The union of these local parameter sets is the network's overall parameter set. If a node's parameter set is non-empty, then its node function depends on the parameter values; we use a square to represent this kind of **adaptive node**. On the other hand, if a node has an empty parameter set, then its function is fixed; we use a circle to denote this type of **fixed node**.

Adaptive networks are generally classified into two categories on the basis of the type of connections they have: **feedforward** and **recurrent** types. The adaptive network shown in Figure 18 is a feedforward network, since the output of each node propagates from the input side (left) to the

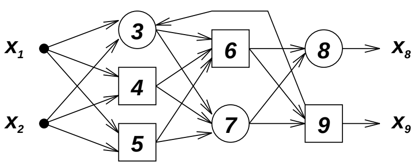


Fig. 19. *A recurrent adaptive network.*

output side (right) unanimously. If there is a feedback link that forms a circular path in a network, then the network is a recurrent network; Figure 19 is an example. (From the viewpoint of graph theory, a feedforward network is represented by an *acyclic* directed graph which contains no directed cycles, while a recurrent network always contains at least one directed cycle.)

In the **layered representation** of the feedforward adaptive network in Figure 18, there are no links between nodes in the same layer and outputs of nodes in a specific layer are always connected to nodes in succeeding layers. This representation is usually preferred because of its modularity, in that nodes in the same layer have the same functionality or generate the same level of abstraction about input vectors.

Another representation of feedforward networks is the **topological ordering representation**, which labels the nodes in an ordered sequence 1, 2, 3, ..., such that there are no links from node i to node j whenever $i \geq j$. Figure 20 is the topological ordering representation of the network in Figure 18. This representation is less modular than the layer representation, but it facilitates the formulation of the learning rule, as will be seen in the next section. (Note that the topological ordering representation is in fact a special case of the layered representation, with one node per layer.)

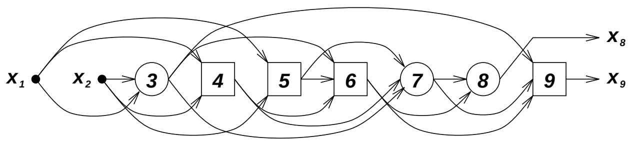


Fig. 20. *A feedforward adaptive network in topological ordering representation.*

Conceptually, a feedforward adaptive network is actually a static mapping between its input and output spaces; this mapping may be either a simple linear relationship or a highly nonlinear one, depending on the structure (node arrangement and connections, and so on) for the network and the function for each node. Here our aim is to construct a network for achieving a desired nonlinear mapping that is regulated by a data set consisting of a number of desired input-output pairs of a target system. This data set is usually called the **training data set** and the procedure we follow in adjusting the parameters to improve the performance of the network are often referred to as the **learning rule** or **learning algorithm** . Usually an adaptive network's performance is measured as the discrepancy between the desired output and the network's output under the same input conditions. This discrepancy is called the **error measure** and it can assume different forms for different applications. Generally speaking, a learning rule is derived by applying a specific optimization technique to a given error measure.

Before introducing a basic learning algorithm for adaptive networks, we shall present several examples of adaptive networks.

Example 3: An adaptive network with a single linear node.
Figure 21 is an adaptive network with a single node specified by

$$x_3 = f_3(x_1, x_2; a_1, a_2, a_3) = a_1x_1 + a_2x_2 + a_3,$$

where x_1 and x_2 are inputs and a_1, a_2, and a_3 are modifiable parameters. Obviously this function defines a plane in $x_1 - x_2 - x_3$ space, and by setting appropriate values for the parameters, we can place this plane arbitrarily. By adopting the squared error as the error measure for this network, we can identify the optimal parameters via the linear least-squares estimation method.

□

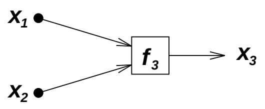


Fig. 21. *A linear single-node adaptive network.*

Example 4: A building block for the perceptron or the back-propagation neural network.
If we add another node to let the output of the adaptive network in Figure 21 have only two values 0 and 1, then the nonlinear network shown in Figure 22 is obtained. Specifically, the node outputs are expressed as

$$x_3 = f_3(x_1, x_2; a_1, a_2, a_3) = a_1x_1 + a_2x_2 + a_3,$$

and

$$x_4 = f_4(x_3) = \begin{cases} 1 & \text{if } x_3 \geq 0 \\ 0 & \text{if } x_3 < 0 \end{cases},$$

where f_3 is a linearly parameterized function and f_4 is a step function which maps x_3 to either 0 or 1. The overall function of this network can be viewed as a **linear classifier**: the first node forms a decision boundary as a straight line in $x_1 - x_2$ space, and the second node indicates which half plane the input vector (x_1, x_2) resides in. Obviously we can form an equivalent network with a single node whose function is the composition of f_3 and f_4; the resulting node is the building block of the classical **perceptron**.

Since the step function is discontinuous at one point and flat at all the other points, it is not suitable for learning procedures based on gradient descent. One way to get around this difficulty is to use the sigmoid function:

$$x_4 = f_4(x_3) = \frac{1}{1 + e^{-x_3}},$$

which is a continuous and differentiable approximation to the step function. The composition of f_3 and this differentiable f_4 is the building block for the back-propagation neural network in the following example.

□

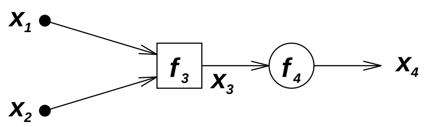


Fig. 22. *A nonlinear single-node adaptive network.*

Example 5: A back-propagation neural network. Figure 23 is a typical architecture for a back-propagation neural network with three inputs, two outputs, and three hidden nodes that do not connect directly to either inputs or outputs. (The term back-propagation refers to the way the learning procedure is performed, that is, by propagating gradient information from the network's outputs to its inputs; details on this are to be introduced next.) Each node in a network of this kind has the same node function, which is the composition of a linear f_3 and a sigmoidal f_4 in example 4. For instance, the node function of node 7 in Figure 23 is

$$x_7 = \frac{1}{1 + exp[-(w_{4,7}x_4 + w_{5,7}x_5 + w_{6,7}x_6 + t_7)]},$$

where x_4, x_5, and x_6 are outputs from nodes 4, 5, and 6, respectively, and $\{w_{4,7}, w_{5,7}, w_{6,7}, t_7\}$ is the parameter set. Usually we view $w_{i,j}$ as the **weight** associated with the link connecting node i and j and t_j as the **threshold** associated with node j. However, it should be noted that this weight-link association is only valid in this type of network. In general, a link only indicates the signal flow direction and the causal relationship between connected nodes, as will be shown in other types of adaptive networks in the subsequent development. A more detailed discussion about the structure and learning rules of the artificial neural network will be presented later.

□

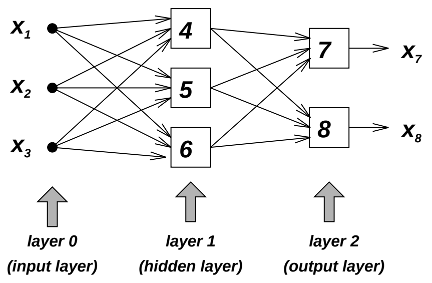


Fig. 23. *A 3-3-2 neural network.*

B. Back-Propagation Learning Rule for Feedforward Networks

The central part of a learning rule for an adaptive network concerns how to recursively obtain a gradient vector in which each element is defined as the derivative of an error measure with respect to a parameter. This is done by means of the chain rule, and the method is generally referred to as the **back-propagation learning rule** because the gradient vector is calculated in the direction opposite to the flow of the output of each node. Details follow below.

Suppose that a given feedforward adaptive network in the layered representation has L layers and layer l ($l = 0, 1, \ldots, L$; $l = 0$ represents the input layer) has $N(l)$ nodes. Then the output and function of node i ($i = 1, \ldots, N(l)$) of layer l can be represented as $x_{l,i}$ and $f_{l,i}$, respectively, as shown in Figure 24 (a). Without loss of generality, we assume there are no jumping links, that is, links connecting non-consecutive layers. Since the output of a node depends on the incoming signals and the parameter set of the node, we have the following general expression for the node function $f_{l,i}$:

$$x_{l,i} = f_{l,i}(x_{l-1,1}, \ldots x_{l-1,N(l-1)}, \alpha, \beta, \gamma, \ldots), \tag{19}$$

where α, β, γ, etc. are the parameters pertaining to this node.

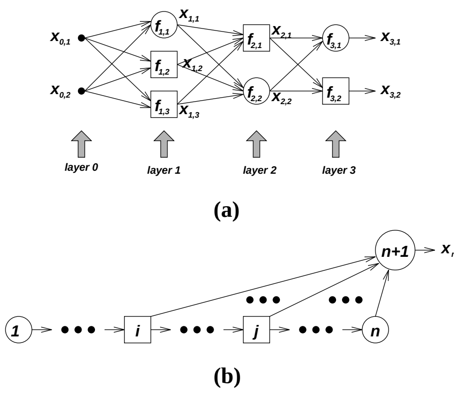


Fig. 24. *Our notational conventions: (a) layered representation; (b) topological ordering representation.*

Assuming the given training data set has P entries, we can define an **error measure** for the p-th ($1 \leq p \leq P$) entry of the training data as the sum of squared errors:

$$E_p = \sum_{k=1}^{N(L)} (d_k - x_{L,k})^2, \tag{20}$$

where d_k is the k-th component of the p-th desired output vector and $x_{L,k}$ is the k-th component of the actual output vector produced by presenting the p-th input vector to the network. (For notational simplicity, we omit the subscript p for both d_k and $x_{L,k}$.) Obviously, when E_p is equal to zero, the network is able to reproduce exactly the desired output vector in the p-th training data pair. Thus our task here is to minimize an overall error measure, which is defined as $E = \sum_{p=1}^{P} E_p$.

Remember that the definition of E_p in equation (20) is not universal; other definitions of E_p are possible for specific situations or applications. Therefore we shall avoid using an explicit expression for the error measure E_p in order to emphasize the generality. In addition, we assume that E_p depends on the output nodes only; more general situations will be discussed below.

To use the gradient method to minimize the error measure, first we have to obtain the gradient vector. Before calculating the gradient vector, we should observe that

$$\boxed{\begin{array}{c}\text{change in}\\ \text{parameter } \alpha\end{array}} \Rightarrow \boxed{\begin{array}{c}\text{change in the output}\\ \text{of node containing } \alpha\end{array}} \Rightarrow$$
$$\boxed{\begin{array}{c}\text{change in the output}\\ \text{of the final layer}\end{array}} \Rightarrow \boxed{\begin{array}{c}\text{change in the}\\ \text{error measure}\end{array}}$$

where the arrows $\Rightarrow$ indicate causal relationships. In other words, a small change in a parameter α will affect the output of the node containing α; this in turn will affect the output of the final layer and thus the error measure. Therefore the basic concept in calculating the gradient vector of the parameters is to pass a form of derivative information starting from the output layer and going backward layer by layer until the input layer is reached.

To facilitate the discussion, we define the **error signal** $\epsilon_{l,i}$ as the derivative of the error measure E_p with respect to the output of node i in layer l, taking both direct and indirect paths into consideration. In symbols,

$$\epsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}}. \tag{21}$$

This expression was called the **ordered derivative** by Werbos [109]. The difference between the ordered derivative and the ordinary partial derivative lies in the way we view the function to be differentiated. For an internal node output $x_{l,i}$ (where $l \neq L$), the partial derivative $\frac{\partial E_p}{\partial x_{l,i}}$ is equal to zero, since E_p does not depend on $x_{l,i}$ directly. However, it is obvious that E_p does depend on $x_{l,i}$ indirectly, since a change in $x_{l,i}$ will propagate through indirect paths to the output layer and thus produce a corresponding change in the value of E_p. Therefore $\epsilon_{l,i}$ can be viewed as the ratio of these two changes when they are made infinitesimal. The following example demonstrates the difference between the ordered derivative and the ordinary partial derivative.

Example 6: Ordered derivatives and ordinary partial derivatives

Consider the simple adaptive network shown in Figure 25, where z is a function of x and y, and y is in turn a function of x:

$$\begin{cases} y &= f(x), \\ z &= g(x, y). \end{cases}$$

For the ordinary partial derivative $\frac{\partial z}{\partial x}$, we assume that all the other input variables (in this case, y) are constant:

$$\frac{\partial z}{\partial x} = \frac{\partial g(x, y)}{\partial x}.$$

In other words, we assume the direct inputs x and y are independent, without paying attention to the fact that y is actually a function of x. For the ordered derivative, we take this indirect causal relationship into consideration:

$$\begin{aligned}\frac{\partial^+ z}{\partial x} &= \frac{\partial g(x, f(x))}{\partial x} \\ &= \left.\frac{\partial g(x, y)}{\partial x}\right|_{y=f(x)} + \left.\frac{\partial g(x, y)}{\partial y}\right|_{y=f(x)} \cdot \frac{\partial f(x)}{\partial x}.\end{aligned}$$

Therefore the ordered derivative takes into consideration both the direct and indirect paths that lead to the causal relationship.

□

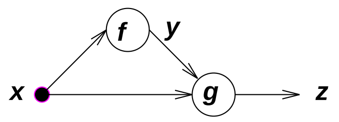


Fig. 25. *Ordered derivatives and ordinary partial derivatives (see text for details).*

The error signal for the i-th output node (at layer L) can be calculated directly:

$$\epsilon_{L,i} = \frac{\partial^+ E_p}{\partial x_{L,i}} = \frac{\partial E_p}{\partial x_{L,i}}. \tag{22}$$

This is equal to $\epsilon_{L,i} = -2(d_i - x_{L,i})$ if E_p is defined as in equation (20). For the internal (non-output) node at the i-th position of layer l, the error signal can be derived by the chain rule:

$$\begin{aligned}\epsilon_{l,i} &= \underbrace{\frac{\partial^+ E_p}{\partial x_{l,i}}}_{\substack{\text{error signal}\\ \text{at layer } l}} = \sum_{m=1}^{N(l+1)} \underbrace{\frac{\partial^+ E_p}{\partial x_{l+1,m}}}_{\substack{\text{error signal}\\ \text{at layer } l+1}} \times \frac{\partial f_{l+1,m}}{\partial x_{l,i}} \\ &= \sum_{m=1}^{N(l+1)} \epsilon_{l+1,m} \frac{\partial f_{l+1,m}}{\partial x_{l,i}},\end{aligned} \tag{23}$$

where $0 \le l \le L-1$. That is, the error signal of an internal node at layer l can be expressed as a linear combination of the error signal of the nodes at layer $l+1$. Therefore for any l and i ($0 \le l \le L$ and $1 \le i \le N(l)$), we can find $\epsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}}$ by first applying equation (22) once to get error signals at the output layer, and then applying equation (23) iteratively until we reach the desired layer l. Since the error signals are obtained sequentially from the output layer back to the input layer, this learning paradigm is called the **back-propagation** learning rule by Rumelhart, Hinton and Williams [79].

The gradient vector is defined as the derivative of the error measure with respect to each parameter, so we have to apply the chain rule again to find the gradient vector. If α is a parameter of the i-th node at layer l, we have

$$\frac{\partial^+ E_p}{\partial \alpha} = \frac{\partial^+ E_p}{\partial x_{l,i}} \frac{\partial f_{l,i}}{\partial \alpha} = \epsilon_{l,i} \frac{\partial f_{l,i}}{\partial \alpha}. \tag{24}$$

Note that if we allow the parameter α to be shared between different nodes, then equation (24) should be changed to a more general form:

$$\frac{\partial^+ E_p}{\partial \alpha} = \sum_{x^* \in S} \frac{\partial^+ E_p}{\partial x^*} \frac{\partial f^*}{\partial \alpha}, \tag{25}$$

where S is the set of nodes containing α as a parameter and f^* is the node function for calculating x^*.

The derivative of the overall error measure E with respect to α is

$$\frac{\partial^+ E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial^+ E_p}{\partial \alpha}. \tag{26}$$

Accordingly, the update formula for the generic parameter α is

$$\triangle\alpha = -\eta \frac{\partial^+ E}{\partial \alpha}, \tag{27}$$

in which η is the **learning rate**, which can be further expressed as

$$\eta = \frac{\kappa}{\sqrt{\sum_\alpha (\frac{\partial E}{\partial \alpha})^2}}, \tag{28}$$

where κ is the **step size**, the length of each transition along the gradient direction in the parameter space. Usually we can change the step size to vary the speed of convergence; two heuristic rules for updating the value of κ are described in [29].

When an n-node feedforward network is represented in its topological order, we can envision the error measure E_p as the output of an additional node with index $n+1$, whose node function f_{n+1} can be defined on the outputs of any nodes with smaller indices; see Figure 24 (b). (Therefore E_p may depend directly on any internal nodes.) Applying the chain rule again, we have the following concise formula for calculating the error signal $\epsilon_i = \partial E_p / \partial x_i$:

$$\frac{\partial^+ E_p}{\partial x_i} = \frac{\partial f_{n+1}}{\partial x_i} + \sum_{i<j\le n} \frac{\partial^+ E_p}{\partial x_j}\frac{\partial f_j}{\partial x_i}, \tag{29}$$

or

$$\epsilon_i = \frac{\partial f_{n+1}}{\partial x_i} + \sum_{i<j\le n} \epsilon_j \frac{\partial f_j}{\partial x_i}, \tag{30}$$

where the first term shows the direct effect of x_i on E_p via the direct path from node i to node $n+1$ and each product term in the summation indicates the indirect effect of x_i on E_p. Once we find the error signal for each node, then the gradient vector for the parameters is derived as before.

Another simple and systematic way to calculate the error signals is through the representation of the **error-propagation network** (or **sensitivity model**), which is obtained from the original adaptive network by reversing the links and supplying the error signals at the output layer as inputs. The following example illustrates this idea.

Example 7: Adaptive network and its error-propagation model
Figure 26 (a) is an adaptive network, where each node is indexed by a unique number. Again, we use f_i and x_i to denote the function and output of node i. In order to calculate the error signals at internal nodes, an error-propagation network is constructed in Figure 26 (b), where the output of node i is the error signal of this node in the original adaptive network. In symbols, if we choose the squared error measure for E_p, then we have the following:

$$\epsilon_9 = \frac{\partial^+ E_p}{\partial x_9} = \frac{\partial E_p}{\partial x_9} = -2\,(d_9 - x_9),$$

$$\epsilon_8 = \frac{\partial^+ E_p}{\partial x_8} = \frac{\partial E_p}{\partial x_8} = -2\,(d_8 - x_8),$$

(Thus nodes 9 and 8 in the error-propagation network are only buffer nodes.)

$$\epsilon_7 = \frac{\partial^+ E_p}{\partial x_7} = \frac{\partial^+ E_p}{\partial x_8}\frac{\partial f_8}{\partial x_7} + \frac{\partial^+ E_p}{\partial x_9}\frac{\partial f_9}{\partial x_7} = \epsilon_8 \frac{\partial f_8}{\partial x_7} + \epsilon_9 \frac{\partial f_9}{\partial x_7},$$

$$\epsilon_6 = \frac{\partial^+ E_p}{\partial x_6} = \frac{\partial^+ E_p}{\partial x_8}\frac{\partial f_8}{\partial x_6} + \frac{\partial^+ E_p}{\partial x_9}\frac{\partial f_9}{\partial x_6} = \epsilon_8 \frac{\partial f_8}{\partial x_6} + \epsilon_9 \frac{\partial f_9}{\partial x_6}.$$

Similar expressions can be written for the error signals of node 1, 2, 3, 4 and 5. It is interesting to observe that in the error-propagation net, if we associate each link connecting nodes i and j ($i < j$) with a weight $w_{ij} = \frac{\partial f_j}{\partial x_i}$, then each node performs a linear function and the error-propagation net is actually a linear network. The error-propagation network is helpful in correctly formulating the expressions for error signals. The same concept applies to recurrent networks with either synchronous or continuous operations [34].

□

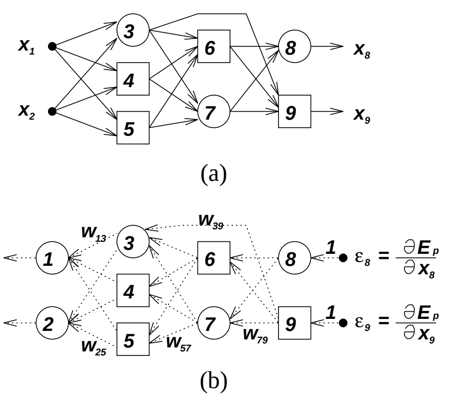


Fig. 26. *(a) An adaptive network and (b) its error-propagation model.*

Depending on the applications we are interested in, two types of learning paradigms for adaptive networks are available to suit our needs. In **off-line learning** (or **batch learning**), the update formula for parameter α is based on equation (26) and the update action takes place only after the whole training data set has been presented, that is, only after each **epoch** or **sweep**. On the other hand, in **on-line learning** (or **pattern learning**), the parameters are updated immediately after each input-output pair has been presented, and the update formula is based on equation (24). In practice, it is possible to combine these two learning modes and update the parameter after k training data entries have been presented, where k is between 1 and P and it is sometimes referred to as the **epoch size**.

C. Back-Propagation Learning Rule for Recurrent Networks

For recurrent adaptive networks, the back-propagation learning rule is still applicable if we can transform the network configurations to be of the feedforward type. To simplify our notation, we shall use the network in Figure 27 for our discussion, where x_1 and x_2 are inputs and x_5 and x_6 are outputs. Because it has directional loops 3-4-5, 3-4-6-5, and 6 (a self loop), this is a typical recurrent network with node functions denoted as follows:

$$\begin{cases} x_3 &= f_3(x_1, x_5) \\ x_4 &= f_4(x_2, x_3) \\ x_5 &= f_5(x_4, x_6) \\ x_6 &= f_6(x_4, x_6) \end{cases} \tag{31}$$

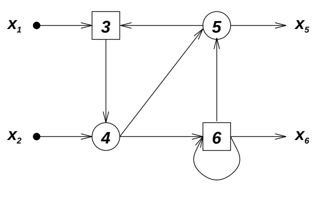


Fig. 27. *A simple recurrent network.*

In order to correctly derive the back-propagation learning rule for the recurrent net in Figure 27, we have to distinguish two operating modes through which the network may satisfy equation (31). These two modes are **synchronous operation** and **continuous operation**.

For continuously operated networks, all nodes continuously change their outputs until equation (31) is satisfied. This operating mode is of particular interest for analog circuit implementations, where a certain kind of dynamical evolution rule is imposed on the network. For instance, the dynamical formula for node 3 can be written as

$$\tau_3 \frac{dx_3}{dt} + x_3 = f_3(x_1, x_5). \tag{32}$$

Similar formulas can be devised for other nodes. It is obvious that when $x_3(t)$ stops changing (i.e., $\frac{dx_3}{dt} = 0$), equation (32) leads to the correct fixed points satisfying equation (31). However, this kind of recurrent networks do pose some problems in software simulation, as the stable fixed point satisfying equation (31) may be hard to find. Here we shall not go into details about continuously operated networks. A detailed treatment of continuously operated networks which use the Mason gain formula [62] as a learning rule can be found in [34].

On the other hand, if a network is operated synchronously, all nodes change their outputs simultaneously according to a global clock signal and there is a time delay associated with each link. This synchronization is reflected by adding the time t as an argument to the output of each node in equation (31) (assuming there is a unit time delay associated with each link):

$$\begin{cases} x_3(t+1) &= f_3(x_1(t), x_5(t)) \\ x_4(t+1) &= f_4(x_2(t), x_3(t)) \\ x_5(t+1) &= f_5(x_4(t), x_6(t)) \\ x_6(t+1) &= f_6(x_4(t), x_6(t)) \end{cases} \tag{33}$$

C.1 Back-Propagation Through Time (BPTT)

When using synchronously operated networks, we usually are interested in identifying a set of parameters that will make the output of a node (or several nodes) follow a given trajectory (or trajectories) in a discrete time domain. This problem of **tracking** or **trajectory following** is usually solved by using a method called **unfolding of time** to transform a recurrent network into a feedforward one, as long as the time t does not exceed a reasonable maximum T. This idea was originally introduced by Minsky and Papert [64] and combined with back-propagation by Rumelhart, Hinton, and Williams [79]. Consider the recurrent net in Figure 27, which is redrawn in Figure 28 (a) with the same configuration except that the input variables x_1 and x_2 are omitted for simplicity. The same network in a feedforward architecture is shown in Figure 28 (b) with the time index t running from 1 to 4. In other words, for a recurrent net that synchronously evaluates each of its node functions from $t = 1, 2, ..., T$, we can simply duplicate all units T times and arrange the resulting network in a layered feedforward manner.

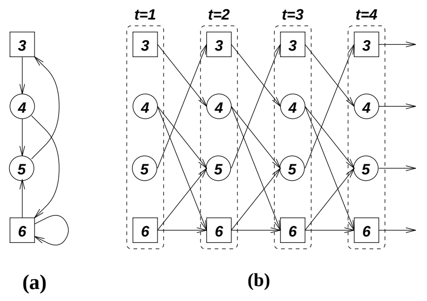


Fig. 28. *(a) A synchronously operated recurrent network and (b) its feedforward equivalent obtained via unfolding of time.*

It is obvious that the two networks in Figure 28 (a) and (b) will behave identically for $t = 1$ to T, provided that all copies of each parameter at different time steps remain identical. For instance, the parameter in node 3 of Figure 28 (a) must be the same at all time instants. This is the problem of **parameter sharing**; a quick solution is to move the parameters from node 3 and 6 into the so-called **parameter nodes**, which are independent of the time step, as shown in Figure 29. (Without loss of generality, we assume nodes 3 and 6 both have only one parameter, denoted by a and b, respectively.) After setting up the parameter nodes in this way, we can apply the back-propagation learning rule as usual to the network in Figure 29 (which is still feedforward in nature) without the slightest concern about the parameter sharing constraint. Note that the error signals of parameter nodes come from the error signals of nodes located at layers of different time instants; thus the BP for this kind of unfolded network is often called **back-propagation through time** (**BPTT**).

C.2 Real Time Recurrent Learning (RTRL)

BPTT generally works well for most problems; the only complication is that it requires extensive computing resources when the sequence length T is large, because the duplication of nodes makes both memory requirements and simulation time proportional to T. Therefore for long sequences or sequences of unknown length, **real time recur-**

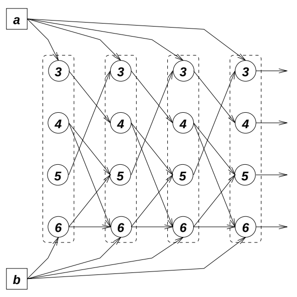


Fig. 29. *An alternative representation of Figure 28 (b) that automatically satisfies the parameter-sharing requirement.*

rent learning (RTRL) [114] is employed instead to perform on-line learning, that is, to update parameters while the network is running rather than at the end of the presented sequences.

To explain the rationale behind the RTRL algorithm, we take as an example the simple recurrent network in Figure 30 (a), where there is only one node with one parameter a. After moving the parameter out of the unfolded architecture, we obtain the feedforward network shown in Figure 30 (b). Figure 30 (c) is the corresponding error-propagation network. Here we assume $E = \sum_i^T E_i = \sum_i^T (d_i - x_i)^2$, where i is the index for time and d_i and x_i are the desired and the actual node output, respectively, at time instant i.

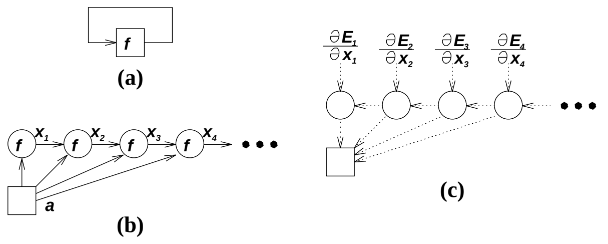


Fig. 30. *A simple recurrent adaptive network to illustrate RTRL: (a) a recurrent net with single node and single parameter; (b) unfolding-of-time architecture; (c) error-propagation network.*

To save computation and memory requirements, a sensible choice is to minimize E_i at each time step instead of trying to minimize E at the end of a sequences. To achieve this, we need to calculate $\partial^+ E/\partial a$ recursively at each time step i. For $i = 1$, the error-propagation network is as shown in Figure 31 (a) and we have

$$\frac{\partial^+ x_1}{\partial a} = \frac{\partial x_1}{\partial a} \text{ and } \frac{\partial^+ E_1}{\partial a} = \frac{\partial E_1}{\partial x_1}\frac{\partial^+ x_1}{\partial a}. \tag{34}$$

For $i = 2$, the error-propagation network is as shown in Figure 31 (b) and we have

$$\frac{\partial^+ x_2}{\partial a} = \frac{\partial x_2}{\partial a} + \frac{\partial x_2}{\partial x_1}\frac{\partial^+ x_1}{\partial a} \text{ and } \frac{\partial^+ E_2}{\partial a} = \frac{\partial E_2}{\partial x_2}\frac{\partial^+ x_2}{\partial a}. \tag{35}$$

For $i = 3$, the error-propagation network is as shown in Figure 31 (c) and we have

$$\frac{\partial^+ x_3}{\partial a} = \frac{\partial x_3}{\partial a} + \frac{\partial x_3}{\partial x_2}\frac{\partial^+ x_2}{\partial a} \text{ and } \frac{\partial^+ E_3}{\partial a} = \frac{\partial E_3}{\partial x_3}\frac{\partial^+ x_3}{\partial a}. \tag{36}$$

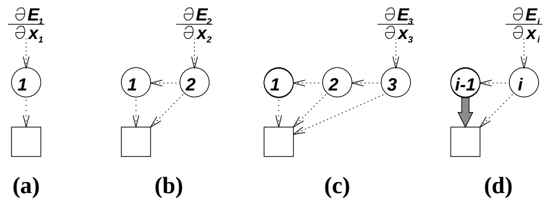


Fig. 31. *Error-propagation networks at different time steps: (a) $i = 1$; (b) $i = 2$; (c) $i = 3$; (d) a general situation, where the thick arrow represents $\frac{\partial^+ x_{i-1}}{\partial a}$.*

In general, for the error-propagation at time instant i, we have

$$\frac{\partial^+ x_i}{\partial a} = \frac{\partial x_i}{\partial a} + \frac{\partial x_i}{\partial x_{i-1}}\frac{\partial^+ x_{i-1}}{\partial a} \text{ and } \frac{\partial^+ E_i}{\partial a} = \frac{\partial E_i}{\partial x_i}\frac{\partial^+ x_3}{\partial a}, \tag{37}$$

where $\frac{\partial^+ x_{i-1}}{\partial a}$ is already available from the calculation at the previous time instant. Figure 31 shows this general situation, where the thick arrow represents $\frac{\partial^+ x_{i-1}}{\partial a}$, which is already available at the time instant $i - 1$.

Therefore, by trying to minimize each individual E_i, we can recursively find the gradient $\frac{\partial^+ E_i}{\partial a}$ at each time instant; there is no need to wait until the end of the presented sequence. Since this is an approximation of the original BPTT, the learning rate η in the update formula

$$\Delta a = -\eta \frac{\partial^+ E_i}{\partial a}$$

should be kept small and, as a result, the learning process usually takes longer.

D. Hybrid Learning Rule: Combining BP and LSE

It is observed that if an adaptive network's output (assuming only one) or its transformation is linear in some of the network's parameters, then we can identify these linear parameters by the well-known linear least-squares method. This observation leads to a hybrid learning rule [24], [29] which combines the gradient method and the least-squares estimator (LSE) for fast identification of parameters.

D.1 Off-Line Learning (Batch Learning)

For simplicity, assume that the adaptive network under consideration has only one output

$$output = F(\vec{I}, S), \tag{38}$$

where $\vec{I}$ is the vector of input variables and S is the set of parameters. If there exists a function H such that the composite function $H \circ F$ is linear in some of the elements of S, then these elements can be identified by the least-squares method. More formally, if the parameter set S can be decomposed into two sets

$$S = S_1 \oplus S_2, \tag{39}$$

(where $\oplus$ represents direct sum) such that $H \circ F$ is linear in the elements of S_2, then upon applying H to equation

(38), we have

$$H(output) = H \circ F(\vec{I}, S), \tag{40}$$

which is linear in the elements of S_2. Now given values of elements of S_1, we can plug P training data into equation (40) and obtain a matrix equation:

$$A\boldsymbol{\theta} = B \tag{41}$$

where $\boldsymbol{\theta}$ is an unknown vector whose elements are parameters in S_2. This equation represents the standard linear least-squares problem and the best solution for $\boldsymbol{\theta}$, which minimizes $||A\boldsymbol{\theta} - B||^2$, is the least-squares estimator (LSE) $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = (A^T A)^{-1} A^T B, \tag{42}$$

where A^T is the transpose of A and $(A^T A)^{-1} A^T$ is the pseudo-inverse of A if $A^T A$ is non-singular. Of course, we can also employ the recursive LSE formula [23], [1], [58]. Specifically, let the i-th row vector of matrix A defined in equation (41) be a_i^T and the i-th element of B be b_i^T; then $\boldsymbol{\theta}$ can be calculated iteratively as follows:

$$\left.\begin{array}{rcl} \boldsymbol{\theta}_{i+1} & = & \boldsymbol{\theta}_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T \boldsymbol{\theta}_i) \\ S_{i+1} & = & S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0, 1, \cdots, P-1 \end{array}\right\}, \tag{43}$$

where the least-squares estimator $\boldsymbol{\theta}^*$ is equal to $\boldsymbol{\theta}_P$. The initial conditions needed to bootstrap equation (43) are $\boldsymbol{\theta}_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the identity matrix of dimension $M \times M$. When we are dealing with multi-output adaptive networks (*output* in equation (38) is a column vector), equation (43) still applies except that b_i^T is the i-th row of matrix B.

Now we can combine the gradient method and the least-squares estimator to update the parameters in an adaptive network. For hybrid learning to be applied in a batch mode, each epoch is composed of a forward pass and a backward pass. In the forward pass, after an input vector is presented, we calculate the node outputs in the network layer by layer until a corresponding row in the matrices A and B in equation (41) are obtained. This process is repreated for all the training data entries to form the complete A and B; then parameters in S_2 are identified by either the pseudo-inverse formula in equation (42) or the recursive least-squares formulas in equation (43). After the parameters in S_2 are identified, we can compute the error measure for each training data entry. In the backward pass, the error signals (the derivative of the error measure w.r.t. each node output, see equations (22) and (23)) propagate from the output end toward the input end; the gradient vector is accumulated for each training data entry. At the end of the backward pass for all training data, the parameters in S_1 are updated by the gradient method in equation (27).

For given fixed values of the parameters in S_1, the parameters in S_2 thus found are guaranteed to be the global optimum point in the S_2 parameter space because of the choice of the squared error measure. Not only can this hybrid learning rule decrease the dimension of the search space in the gradient method, but, in general, it will also substantially reduce the time needed to reach convergence.

It should be kept in mind that by using the least-squares method on the data transformed by $H(\cdot)$, the obtained parameters are optimal in terms of the transformed squared error measure instead of the original one. In practice, this usually will not cause a problem as long as $H(\cdot)$ is monotonically increasing and the training data are not too noisy. A more detailed treatment of this transformation method can be found in [34].

D.2 On-Line Learning (Pattern Learning)

If the parameters are updated after each data presentation, we have a **on-line learning** or **pattern learning** scheme. This learning strategy is vital to on-line parameter identification for systems with changing characteristics. To modify the batch learning rule to obtain an on-line version, it is obvious that the gradient descent should be based on E_p (see equation (24)) instead of E. Strictly speaking, this is not a truly gradient search procedure for minimizing E, yet it will approximate one if the learning rate is small.

For the recursive least-squares formula to account for the time-varying characteristics of the incoming data, the effects of old data pairs must decay as new data pairs become available. Again, this problem is well studied in the adaptive control and system identification literature and a number of solutions are available [20]. One simple method is to formulate the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. This amounts to the addition of a **forgetting factor** λ to the original recursive formula:

$$\left.\begin{array}{rcl} \boldsymbol{\theta}_{i+1} & = & \boldsymbol{\theta}_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T \boldsymbol{\theta}_i) \\ S_{i+1} & = & \frac{1}{\lambda}[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}] \end{array}\right\}, \tag{44}$$

where the typical value of λ in practice is between 0.9 and 1. The smaller λ is, the faster the effects of old data decay. A small λ sometimes causes numerical instability, however, and thus should be avoided. For a complete discussion and derivation of equation (44), the reader is referred to [34], [58], [20].

D.3 Different Ways of Combining GD and LSE

The computational complexity of the least-squares estimator (LSE) is usually higher than that of the gradient descent (GD) method for one-step adaptation. However, for achieving a prescribed performance level, the LSE is usually much faster. Consequently, depending on the available computing resources and required level of performance, we can choose from among at least five types of hybrid learning rules combining GD and LSE in different degrees, as follows.

1. One pass of LSE only: Nonlinear parameters are fixed while linear parameters are identified by one-time application of LSE.
2. GD only: All parameters are updated by GD iteratively.

3. One pass of LSE followed by GD: LSE is employed only once at the very beginning to obtain the initial values of linear parameters and then GD takes over to update all parameters iteratively.
4. GD and LSE: This is the proposed hybrid learning rule, where each iteration (epoch) of GD used to update the nonlinear parameters is followed by LSE to identify the linear parameters.
5. Sequential (approximate) LSE only: The outputs of an adaptive network are linearized with respect to its parameters, and then the extended Kalman filter algorithm [21] is employed to update all parameters. This method has been proposed in the neural network literature [85], [84], [83].

The choice of one of the above methods should be based on a trade-off between computational complexity and performance. Moreover, the whole concept of fitting data to parameterized models is called **regression** in statistics literature, and there are a number of other techniques for either linear or nonlinear regression, such as the Guass-Newton method (linearization method) and the Marquardt procedure [61]. These methods can be found in advanced textbooks on regression and they are also viable techniques for finding optimal parameters in adaptive networks.

E. Neural Networks as Special Cases of Adaptive Networks

Some special cases of adaptive networks have been explored extensively in the neural network literature. In particular, we will introduce two types of neural networks: the back-propagation neural network (BPNN) and the radial basis function network (RBFN). Other types of adaptive networks that can be interpreted as a set of fuzzy if-then rules are investigated in the next section.

E.1 Back Propagation Neural Networks (BPNN's)

A back-propagation neural network (BPNN), as already mentioned in examples 4 and 5, is an adaptive network whose nodes (called **neurons**) perform the same function on incoming signals; this node function is usually a composite function of the weighted sum and a nonlinear function called the **activation function** or **transfer function**. Usually the activation functions are of either a sigmoidal or a hyper-tangent type which approximates the **step function** (or **hard limiter**) and yet provides differentiability with respect to input signals. Figure 32 depicts the four different types of activation functions $f(x)$ defined below.

$$\begin{array}{ll}
\text{Step function:} & f(x) = \left\{ \begin{array}{l} 1 \text{ if x} \geq 0. \\ 0 \text{ if x} < 0. \end{array} \right. \\
\text{Sigmoid function:} & f(x) = \dfrac{1}{1+e^{-x}}. \\
\text{Hyper-tangent function:} & f(x) = \tanh(x/2) = \dfrac{1-e^{-x}}{1+e^{-x}}. \\
\text{Identity function:} & f(x) = x.
\end{array}$$

When the step function (hard-limiter) is used as the activation function for a layered network, the network is often called a **perceptron** [78], [70], as explained in example 4. For a neural network to approximate a continuous-valued function not necessarily limited to the interval [0, 1] or $[1, -1]$, we usually let the node function for the output layer be a weighted sum with no limiting-type activation functions. This is equivalent to the situation where the activation function is an identity function, and output nodes of this type are often called **linear nodes**.

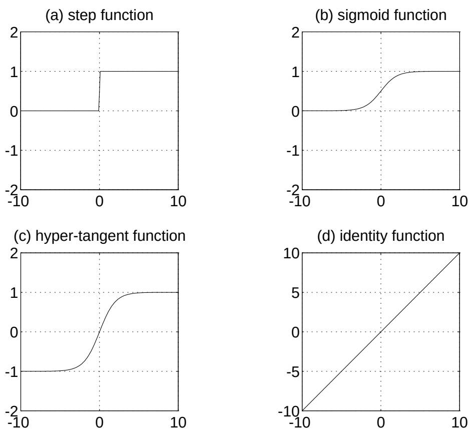


Fig. 32. *Activation functions for BPNN's: (a) step function; (b) sigmoid function; (c) hyper-tangent function; (d) identity function.*

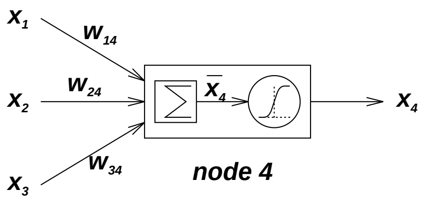


Fig. 33. *A BPNN node.*

For simplicity, we assume the BPNN in question uses the sigmoidal function as its activation function. The *net input* $\bar{x}$ of a node is defined as the weighted sum of the incoming signals plus a threshold. For instance, the net input and output of node j in Figure 33 (where $j = 4$) are

$$\begin{aligned} \bar{x}_j &= \textstyle\sum_i w_{ij} x_i + t_j, \\ x_j &= f(\bar{x}_j) = \frac{1}{1+e^{-\bar{x}_j}}, \end{aligned} \tag{45}$$

where x_i is the output of node i located in the previous layer, w_{ij} is the weight associated with the link connecting nodes i and j, and t_j is the threshold of node j. Since the weights w_{ij} are actually internal parameters associated with each node j, changing the weights of a node will alter the behavior of the node and in turn alter the behavior of the whole BPNN. Figure 23 shows a two-layer BPNN with 3 inputs in the input layer, 3 neurons in the hidden layer, and 2 output neurons in the output layer. For simplicity, this BPNN will be referred to as a 3-3-2 structure, corresponding to the number of nodes in each layer. (Note that the input layer is composed of three buffer nodes for distributing the input signals; therefore this layer is conventionally not counted as a physical layer of the BPNN.)

BPNN's are by far the most commonly used NN struc-

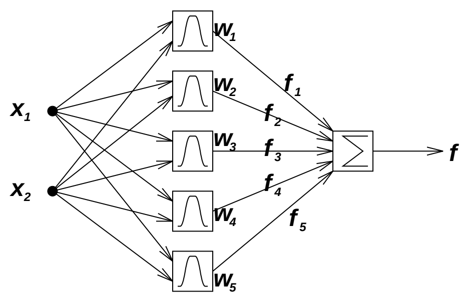


Fig. 34. *A radial basis function network (RBFN) .*

ture for applications in a wide range of areas, such as speech recognition, optical character recognition (OCR), signal processing, data compression, and automatic control.

E.2 Radial Basis Function Networks (RBFN's)

The locally-tuned and overlapping receptive field is a well-known structure that has been studied in the regions of the cerebral cortex, the visual cortex, and so forth. Drawing on the knowledge of biological receptive fields, Moody and Darken [66], [67] proposed a network structure that employs local receptive fields to perform function mappings. Similar schemes have been proposed by Powell [74], Broomhead and Lowe [7], and many others in the areas of interpolation and approximation theory; these schemes are collectively called radial basis function approximations. Here we shall call this network structure the **radial basis function network** or **RBFN**. Figure 34 shows a schematic diagram of an RBFN with five receptive field units; the activation level of the i-th receptive field unit (or hidden unit) is

$$w_i = R_i(\vec{x}) = R_i(||\vec{x} - \vec{c_i}||/\sigma_i),\ i = 1, 2, ..., H \qquad (46)$$

where $\vec{x}$ is a multi-dimensional input vector, $\vec{c_i}$ is a vector with the same dimension as $\vec{x}$, H is the number of radial basis functions (or equivalently, receptive field units), and $R_i(\cdot)$ is the i-th radial basis function with a single maximum at the origin. Typically, $R_i(\cdot)$ is chosen as a Gaussian function

$$R_i(\vec{x}) = exp[-\frac{||\vec{x} - \vec{c_i}||^2}{\sigma_i^2}] \qquad (47)$$

or as a logistic function

$$R_i(\vec{x}) = \frac{1}{1 + exp[||\vec{x} - \vec{c_i}||^2/\sigma_i^2]} \qquad (48)$$

Thus the activation level of the radial basis function w_i computed by the i-th hidden unit is maximum when the input vector $\vec{x}$ is at the center $\vec{c_i}$ of that unit.

The output of a radial basis function network can be computed in two ways. In the simpler method, as shown in Figure 34, the final output is the weighted sum of the output value associated with each receptive field:

$$f(\vec{x}) = \sum_{i=1}^{H} f_i w_i = \sum_{i=1}^{H} f_i R_i(\vec{x}), \qquad (49)$$

where f_i is the output value associated with the i-th receptive field. A more complicated method for calculating the overall output is to take the weighted average of the output associated with each receptive field:

$$f(\vec{x}) = \frac{\sum_{i=1}^{H} f_i w_i}{\sum_{i=1}^{H} w_i} = \frac{\sum_{i=1}^{H} f_i R_i(\vec{x})}{\sum_{i=1}^{H} R_i(\vec{x})}. \qquad (50)$$

This mode of calculation, though has a higher degree of computational complexity, possesses the advantage that points in the overlapping area of two receptive fields will have a well interpolated output value between the output values of the two receptive fields. For representation purposes, if we change the radial basis function $R_i(\vec{x})$ in each node of layer 2 in Figure 34 by its **normalized** counterpart $R_i(\vec{x})/\sum_i R_i(\vec{x})$, then the overall output is specified by equation (50).

Several learning algorithms have been proposed to identify the parameters ($\vec{c_i}$, σ_i and f_i) of an RBFN. Note that the RBFN is an ideal example of the hybrid learning described in the previous section, where the linear parameters are f_i and the nonlinear parameters are c_i and σ_i. In practice, the $\vec{c_i}$ are usually found by means of vector quantization or clustering techniques (which assume similar input vectors produce similar outputs) and the σ_i are obtained heuristically (such as by taking the average distance to the first several nearest neighbors of $\vec{c_i}$'s). Once these nonlinear parameters are fixed, the linear parameters can be found by either the least-squares method or the gradient method. Chen et al. [8] used an alternative method that employs the orthogonal least-squares algorithm to determine the c_i's and f_i's while keeping the σ_i's at a predetermined constant.

An extension of Moody-Darken's RBFN is to assign a linear function as the output function of each receptive field; that is, f_i is a linear function of the input variables instead of a constant:

$$f_i = \vec{a}_i \cdot \vec{x} + b_i, \qquad (51)$$

where $\vec{a}_i$ is a parameter vector and b_i is a scalar parameter. Stokbro et al. [89] used this structure to model the Mackey-Glass chaotic time series [59] and found that this extended version performed better than the original RBFN with the same number of fitting parameters.

It was pointed out by the authors that under certain constraints, the RBFN is functionally equivalent to the the zero-order Sugeno fuzzy model. See [32] or [34] for details.

IV. ANFIS: Adaptive Neuro-Fuzzy Inference Systems

A class of adaptive networks that act as a fundamental framework for adaptive fuzzy inference systems is introduced in this section. This type of networks is referred to as **ANFIS** [25], [24], [29], which stands for **Adaptive-Network-based Fuzzy Inference System**, or semantically equivalently, **Adaptive Neuro-Fuzzy Inference System**. We will describe primarily the ANFIS architecture and its learning algorithm for the Sugeno fuzzy model,

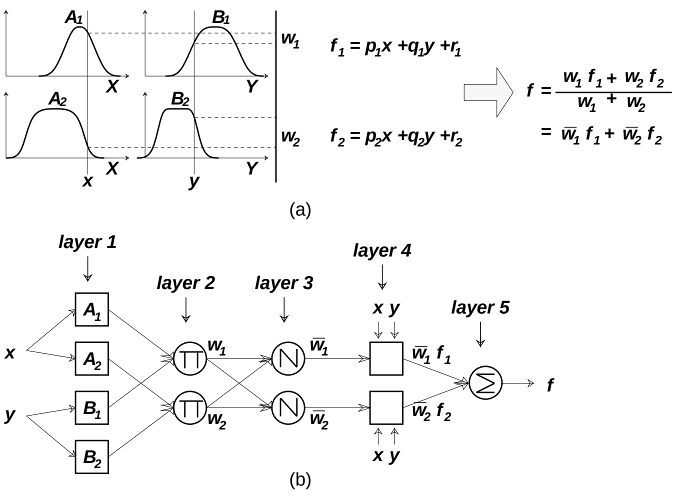


Fig. 35. *(a) A two-input first-order Sugeno fuzzy model with two rules; (b) equivalent ANFIS architecture.*

with an application example of chaotic time series prediction.

Note that similar network structures were also proposed independently by Lin and Lee [55] and Wang and Mendel [106]).

A. ANFIS Architecture

For simplicity, we assume the fuzzy inference system under consideration has two inputs x and y and one output z. For a first-order Sugeno fuzzy model [98], [91], a typical rule set with two fuzzy if-then rules can be expressed as

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1x + q_1y + r_1$,
Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2x + q_2y + r_2$.

Figure 35 (a) illustrates the reasoning mechanism for this Sugeno model. The corresponding equivalent ANFIS architecture is as shown in Figure 35(b), where nodes of the same layer have similar functions, as described below. (Here we denote the output node i in layer l as $O_{l,i}$.)

Layer 1: Every node i in this layer is an adaptive node with a node output defined by

$$\begin{array}{ll} O_{1,i} = \mu_{A_i}(x), & \text{for } i = 1,\ 2, \text{ or} \\ O_{1,i} = \mu_{B_{i-2}}(y), & \text{for } i = 3,\ 4, \end{array} \qquad (52)$$

where x (or y) is the input to the node and A_i (or B_{i-2}) is a fuzzy set associated with this node. In other words, outputs of this layer are the membership values of the premise part. Here the membership functions for A_i and B_i can be any appropriate parameterized membership functions introduced in Section II. For example, A_i can be characterized by the generalized bell function:

$$\mu_A(x) = \frac{1}{1 + [(\frac{x-c_i}{a_i})^2]^{b_i}}, \qquad (53)$$

where $\{a_i,\ b_i,\ c_i\}$ is the parameter set. Parameters in this layer are referred to as *premise parameters*.

Layer 2: Every node in this layer is a fixed node labeled $\mathbf{\Pi}$, which multiplies the incoming signals and outputs the product. For instance,

$$O_{2,i} = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y),\ i = 1, 2. \qquad (54)$$

Each node output represents the firing strength of a rule. (In fact, any other T-norm operators that perform fuzzy AND can be used as the node function in this layer.)

Layer 3: Every node in this layer is a fixed node labeled $\mathbf{N}$. The i-th node calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths:

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2},\ i = 1, 2. \qquad (55)$$

For convenience, outputs of this layer will be called *normalized firing strengths*.

Layer 4: Every node i in this layer is an adaptive node with a node function

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i(p_ix + q_iy + r_i), \qquad (56)$$

where $\overline{w}_i$ is the output of layer 3 and $\{p_i,\ q_i,\ r_i\}$ is the parameter set. Parameters in this layer will be referred to as *consequent parameters*.

Layer 5: The single node in this layer is a fixed node labeled Σ, which computes the overall output as the summation of all incoming signals:

$$O_{5,1} = \mathit{overall\ output} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \qquad (57)$$

Thus we have constructed an adaptive network that has exactly the same function as a Sugeno fuzzy model. Note that the structure of this adaptive network is not unique; we can easily combine layers 3 and 4 to obtain an equivalent network with only four layers. Similarly, we can perform weight normalization at the last layer; Figure 36 illustrates an ANFIS of this type.

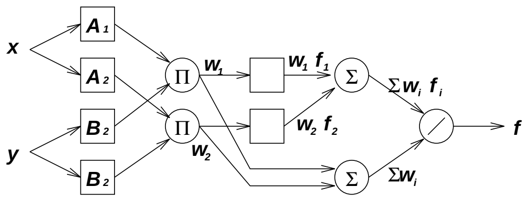


Fig. 36. *Another ANFIS architecture for the two-input two-rule Sugeno fuzzy model.*

Figure 37 (a) is an ANFIS architecture that is equivalent to a two-input first-order Sugeno fuzzy model with nine rules, where each input is assumed to have three associated MF's. Figure 37 (b) illustrates how the 2-D input space is partitioned into nine overlapping fuzzy regions, each of which is governed by fuzzy if-then rules. In other words, the premise part of a rule defines a fuzzy region, while the consequent part specifies the output within this region.

For ANFIS architectures for the Mamdani and Tsukamoto fuzzy models, the reader is referred to [29] and [34] for more details.

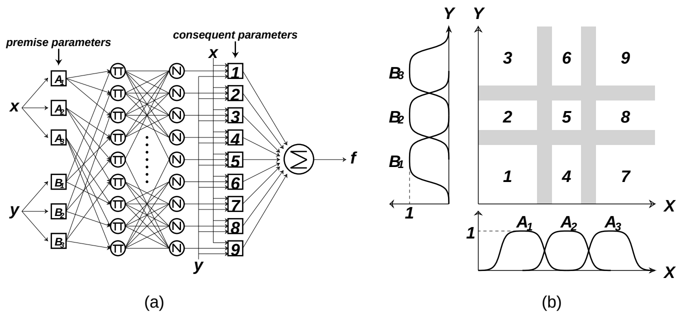


Fig. 37. *(a) ANFIS architecture for a two-input first-order Sugeno fuzzy model with nine rules; (b) partition of the input space into nine fuzzy regions.*

B. Hybrid Learning Algorithm

From the ANFIS architecture shown in Figure 35 (b), we observe that when the values of the premise parameters are fixed, the overall output can be expressed as a linear combination of the consequent parameters. In symbols, the output f in Figure 35 (b) can be rewritten as

$$\begin{aligned} f &= \frac{w_1}{w_1+w_2} f_1 + \frac{w_2}{w_1+w_2} f_2 \\ &= \overline{w}_1 f_1 + \overline{w}_2 f_2 \\ &= (\overline{w}_1 x) p_1 + (\overline{w}_1 y) q_1 + (\overline{w}_1) r_1 + (\overline{w}_2 x) p_2 + (\overline{w}_2 y) q_2 + (\overline{w}_2) r_2, \end{aligned} \quad (58)$$

which is linear in the consequent parameters p_1, q_1, r_1, p_2, q_2, and r_2. Therefore the hybrid learning algorithm developed in the previous section can be applied directly. More specifically, in the forward pass of the hybrid learning algorithm, node outputs go forward until layer 4 and the consequent parameters are identified by the least-squares method. In the backward pass, the error signals propagate backward and the premise parameters are updated by gradient descent. Table I summarizes the activities in each pass.

TABLE I
Two passes in the hybrid learning procedure for ANFIS.

	Forward Pass	Backward Pass
Premise Parameters	Fixed	Gradient Descent
Consequent Parameters	Least-Squares Estimate	Fixed
Signals	Node Outputs	Error Signals

As mentioned earlier, the consequent parameters thus identified are optimal under the condition that the premise parameters are fixed. Accordingly, the hybrid approach converges much faster since it reduces the dimension of the search space of the original back-propagation method.

If we fix the membership functions and adapt only the consequent part, then ANFIS can be viewed as a functional-link network [46], [71] where the "enhanced representations" of the input variables are obtained via the membership functions. These "enhanced representations", which take advantage of human knowledge, apparently express more insight than the functional expansion and the tensor (outer product) models [71]. By fine-tuning the membership functions, we actually make this "enhanced representation" also adaptive.

From equations (49), (50), and equation (57), it is not too hard to see the resemblance between the radial basis function network (RBFN) and the ANFIS for the Sugeno model. Actually these two computing framework are functionally equivalent under certain minor conditions [32]; this cross-fertilize both disciplines in many respects.

C. Application to Chaotic Time Series Prediction

ANFIS can be applied to a wide range of areas, such as nonlinear function modeling [24], [29], time series prediction [33], [29], on-line parameter identification for control systems [29], and fuzzy controller design [26], [28]. In particular, GE has been using ANFIS for modeling correction factors in steel rolling mills [6]. Here we will briefly report the application of ANFIS to chaotic time series prediction [33], [29].

The time series used in our simulation is generated by the Mackey-Glass differential delay equation [59]:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t). \quad (59)$$

The prediction of future values of this time series is a benchmark problem that has been used and reported by a number of connectionist researchers, such as Lapedes and Farber [48], Moody [67], [65], Jones et al. [35], Crower [77], and Sanger [81]. The simulation results presented here were reported in [33], [29]; more details can be found therein.

The goal of the task is to use past values of the time series up to the point $x = t$ to predict the value at some point in the future $x = t + P$. The standard method for this type of prediction is to create a mapping from D points of the time series spaced $\triangle$ apart, that is, $(x(t-(D-1)\triangle)$, ..., $x(t-\triangle)$, $x(t))$, to a predicted future value $x(t+P)$. To allow comparison with earlier work (Lapedes and Farber [48], Moody [67], [65], Crower [77]), the values $D = 4$ and $\triangle = P = 6$ were used. All other simulation settings were arranged to be as similar as possible to those reported in [77].

From the Mackey-Glass time series $x(t)$, we extracted 1000 input-output data pairs of the following format:

$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)], \quad (60)$$

where $t = 118$ to 1117. The first 500 pairs (training data set) were used for training ANFIS, while the remaining 500 pairs (checking data set) were used for validating the model identified. The number of membership functions assigned to each input of the ANFIS was set to two, so the number of rules is 16. The ANFIS used here contains a total of 104 fitting parameters, of which 24 are premise parameters and 80 are consequent parameters

Figure 38 shows the results after about 500 epochs of learning. The desired and predicted values for both training data and checking data are essentially the same in Fig-

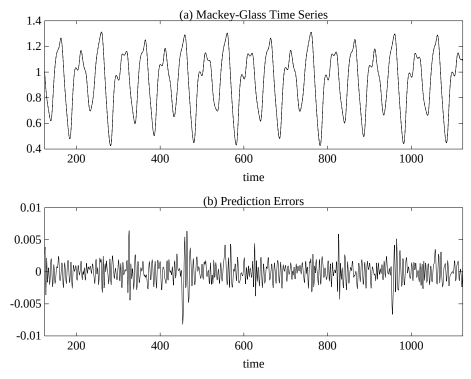


Fig. 38. (a) Mackey-Glass time series from $t = 124$ to 1123 and six-step ahead prediction (which is indistinguishable from the time series here); (b) prediction error. (Note that the first 500 data points are training data, while the remaining are for validation.)

ure 38 (a); the differences between them can only be seen on a much finer scale, such as that in Figure 38 (b).

TABLE II

Generalization result comparisons for $P = 6$.

Methods	Training Data	NDEI
ANFIS	500	0.007
AR Model	500	0.19
Cascade-Correlation NN	500	0.06
Back-Prop NN	500	0.02
6th-order Polynomial	500	0.04
Linear Predictive Method	2000	0.55

Table II lists the generalization capabilities of other methods, which were measured by using each method to predict 500 points immediately following the training set. The last four row of Table II are from [77] directly. The *non-dimensional error index* (NDEI) [48], [77] is defined as the root mean square error divided by the standard deviation of the target series. The remarkable generalization capability of ANFIS is attributed to the following facts:

- ANFIS can achieve a highly nonlinear mapping, therefore it is well-suited for predicting nonlinear time series.
- The ANFIS used here has 104 adjustable parameters, far fewer than those used in the cascade-correlation NN (693, the median) and back-prop NN (about 540) listed in Table II.
- Though not based on *a priori* knowledge, the initial parameter settings of ANFIS are intuitively reasonable and results in fast convergence to good parameter values that captures the underlying dynamics.
- ANFIS consists of fuzzy rules which are actually local mappings (which are called local experts in [36]) instead of global ones. These local mappings facilitate the **minimal disturbance principle** [111], which states that the adaptation should not only reduce the output error for the current training pattern but also minimize disturbance to response already learned. This is particularly important in on-line learning. We also found the use of least-squares method to determine the output of each local mapping is of particular importance. Without using LSE, the learning time would be ten times longer.

Other generalization tests and comparisons with neural network approaches can be found in [29].

The original ANFIS C codes and several examples (including this one) can be retrieved via anonymous ftp in `user/ai/areas/fuzzy/systems/anfis` at `ftp.cs.cmu.edu` (CMU Artificial Intelligence Repository).

V. Neuro-Fuzzy Control

Once a fuzzy controller is transformed into an adaptive network, the resulting ANFIS can take advantage of all the NN controller design techniques proposed in the literature. In this section we shall introduce common design techniques for ANFIS controllers. Most of these methodologies are derived directly from their counterparts for NN controllers. However, certain design techniques apply exclusively to ANFIS, which will be pointed out explicitly.

As shown in Figure 39, the block diagram of a typical feedback control system consists of a plant block and a controller block. The plant block is usually represented by a set of differential equations that describe the physical system to be controlled. These equations govern the behavior of the plant state $\mathbf{x}(t)$, which is assumed to be accessible in our discussion. In contrast, the controller block is usually a static function denoted by $\mathbf{g}$; it maps the the plant state $\mathbf{x}(t)$ into a control action $\mathbf{u}(t)$ that can hopefully achieve a given control objective. Thus for a general time-invariant control system, we have the following equations:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) && \text{(plant dynamics)}, \\ \mathbf{u}(t) &= \mathbf{g}(\mathbf{x}(t)) && \text{(controller)}. \end{aligned}$$

The control objective here is to design a controller function $\mathbf{g}(\cdot)$ such that the plant state $\mathbf{x}(t)$ can follow a desired trajectory $\mathbf{x}_d(t)$ as closely as possible.

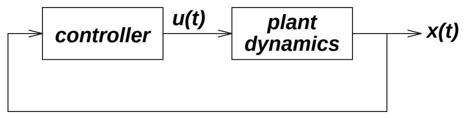


Fig. 39. *Block diagram for a continuous time feedback control system.*

A simple example of a feedback control system is the inverted pendulum system (Figure 40) where a rigid pole is hinged to a cart through a free joint with only one degree of freedom, and the cart moves on the rail tracks to its right or left depending on the force exerted on it. The control goal is to find the applied force u as a function of the state variable $\mathbf{x} = [\theta, \dot{\theta}, z, \dot{z}]$ (where θ is the pole angle and z is the cart position) such that the pole can be balanced from a given non-zero initial condition.

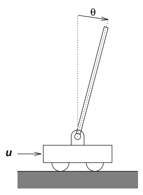


Fig. 40. *The inverted pendulum system.*

For a feedback control system in a discrete time domain, a general block diagram representation is as shown in Figure 41. Note that the inputs to the plant block include the control action $\mathbf{u}(k)$ and the previous plant output $\mathbf{x}(k)$, so the plant block now represents a static mapping. In symbols, we have

$$\begin{array}{lcll} \mathbf{x}(k+1) & = & \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)) & \text{(plant)}, \\ \mathbf{u}(k) & = & \mathbf{g}(\mathbf{x}(k)) & \text{(controller)}. \end{array}$$

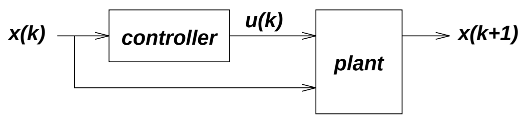


Fig. 41. *Block diagram for a discrete-time feedback control system.*

A central problem in control engineering is that of finding the control action $\mathbf{u}$ as a function of the plant output $\mathbf{x}$ in order to achieve a given control goal. Each design method for neuro-fuzzy controllers corresponds to a way of obtaining the control action; these methods are discussed next.

A. Mimicking Another Working Controller

Most of the time, the controller being mimicked is an experienced human operator who can control the plant satisfactorily. In fact, the whole concept of mimicking a human expert is the original intention of fuzzy controllers whose ultimate goal is to replace human operators who can control complex systems such as chemical reaction processes, subway trains, and traffic systems. An experienced human operator usually can summarize his or her control actions as a set of fuzzy if-then rules with roughly correct membership functions; this corresponds to the linguistic information. Prior to the emergence of neuro-fuzzy approaches, refining membership function is usually obtained via a lengthy trial-and-error process. Now with learning algorithms, we can further take advantage of the numerical information (input/output data pairs) and refine the membership functions in a systematic way. Note that the capability to utilize linguistic information is specific to fuzzy inference systems; it is not always available in neural networks. Successful applications of fuzzy controller based on linguistic information plus trial-and-error tuning includes steam engine and boiler control [60], Sendai subway systems [117], container ship crane control [116], elevator control [54], nuclear reaction control [5], automobile transmission control [40], aircraft control [14], and many others [90]. With the availability of learning algorithms, a wider range of applications is expected.

Note that this approach is not only for control applications. If the target system to be emulated is a human physician or a credit analyst, then the resulting fuzzy inference systems become a fuzzy expert system for diagnosis and credit analysis, respectively.

B. Inverse Control

Another scheme for obtaining desired control action is the inverse control method shown in Figure 42. For simplicity, we assume that the plant has only one state $x(k)$ and one input $u(k)$. In the learning phase, a training set is obtained by generating inputs $u(k)$ at random, and observing the corresponding outputs $x(k)$ produced by the plant. The ANFIS in Figure 42 (a) is then used to learn the inverse model of the plant by fitting the data pairs $(x(k), x(k+1); u(k))$. In the application phase, the ANFIS identifier is copied to the ANFIS controller in Figure 42 for generating the desired output. The input to the ANFIS controller is $(x(k), x_d(k))$; if the inverse model (ANFIS identifier) that maps $(x(k), x(k+1))$ to $u(k)$ is accurate, then the generated $u(k)$ should result in $x(k+1)$ that is close to $x_d(k)$. That is, the whole system in Figure 42 will behave like a pure unit-delay system.

This method seems straightforward and only one learning task is needed to find the inverse model of the plant. However, it assumes existence of the inverse of a plant, which is not valid in general. Moreover, minimization of the network error $||e_u(k)||^2$ does not guarantee minimization of the overall system error $||x_d(k) - x(k)||^2$.

Using ANFIS for adaptive inverse control can be found in [42].

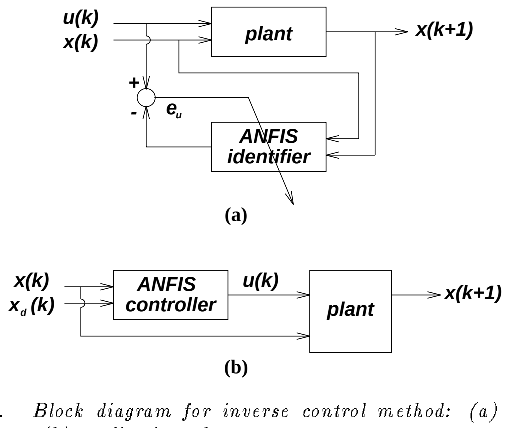
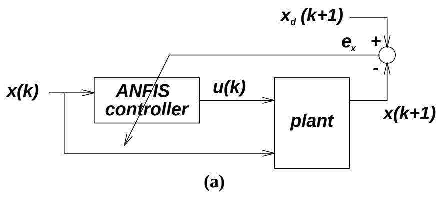


Fig. 42. *Block diagram for inverse control method: (a) learning phase; (b) application phase.*

C. Specialized Learning

The major problem with the inverse control scheme is that we are minimizing the network error instead of the overall system error. An alternative is to minimize the system error directly; this is called **specialized learning** [76]. In order to back-propagate error signals through the plant block in Figure 43, we need to find a model representing the behavior of the plant. In fact, in order to apply back-propagation learning, all we need to know is the **Jacobian matrix** of the plant, where the element at row i and column j is equal to the derivative of the plant's i-th output with respect to its j-th input.

If the Jacobian matrix is not easy to find, an alternative is to estimate it on-line from the changes of the plant's inputs and outputs during two consecutive time instants. Other similar methods that aim at using an approximate Jacobian matrix to achieve the same learning effects can be found in [41], [11], [103]. Applying specialized learning to find an ANFIS controller for the inverted pendulum was reported in [27].



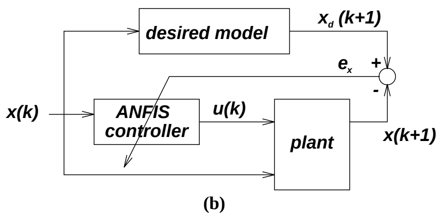


Fig. 43. *Block diagram for (a) specialized learning; (b) specialized learning with model reference.*

It is not always convenient to specify the desired plant output $x_d(k)$ at every time instant k. As a standard approach in model reference adaptive control, the desired behavior of the overall system can be implicitly specified by a (usually linear) model that is able to achieve the control goal satisfactorily. This alternative approach is shown in Figure 43 (b), where the desired output $x_d(k+1)$ is generated through a desired model.

D. Back-Propagation Through Time and Real Time Recurrent Learning

If we replace the controller and the plant block in Figure 39 with two adaptive networks, the feedback control system becomes a recurrent adaptive network discussed in Section III. Assuming the synchronous operation is adopted here (which virtually convert the system into the discrete time domain), we can apply the same scheme of unfolding of time to obtain a feedforward network, and then use the same back-propagation learning algorithm to identify the optimal parameters.

In terms of the inverted pendulum system (pole only), Figure 41 becomes Figure 44 if the controller block is replaced with a four-rule ANFIS and the plant block is replaced with a two-node adaptive network. To obtain the state trajectory, we cascade the network in Figure 44 to obtain the **trajectory network** shown in Figure 45. In particular, the inputs to the trajectory network are initial

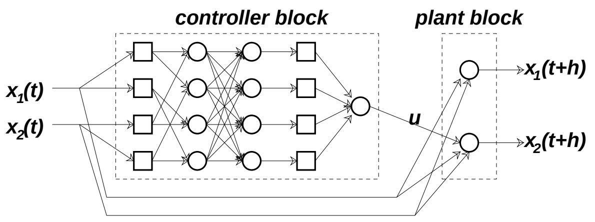


Fig. 44. *Network implementation of Figure 41 .*

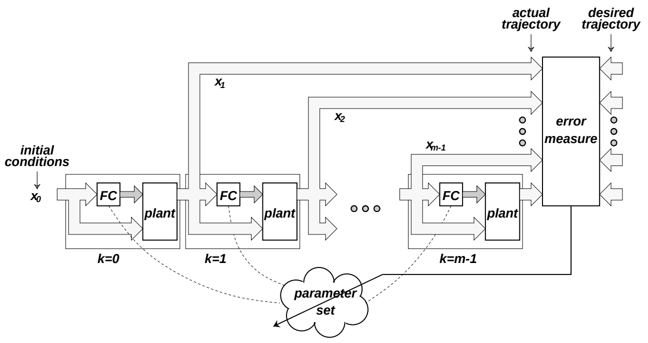


Fig. 45. *A trajectory network for control application (FC stands for "fuzzy controller").*

conditions of the plant; the outputs are the state trajectory from $k = 1$ to $k = m$. The adjustable parameters are all pertaining to the FC (fuzzy controller) block implemented as an four-rule ANFIS. Though there are m FC blocks, all of them refer to the same parameter set. For clarity, this parameter set is shown explicitly in Figure 45 and it is updated according to the output of the error measure block.

Each entry of the training data is of the following format:

(initial conditions; desired trajectory),

and the corresponding error measure to be minimized is

$$E = \sum_{k=1}^{m} \|\mathbf{x}(k) - \mathbf{x_d}(k)\|^2,$$

where $\mathbf{x}_d(k)$ is a desired state vector at $t = k * T$ (T is the sampling period). If we take control efforts into consideration, a revised error measure would be

$$E = \sum_{k=1}^{m} \|\mathbf{x}(k) - \mathbf{x_d}(k)\|^2 + \lambda \sum_{k=0}^{m-1} \|\mathbf{u}(k)\|^2,$$

where $\mathbf{u}(k)$ is the control action at time step k. By a proper selection of λ, a compromise between trajectory error and control efforts can be obtained.

Use of back-propagation through time to train a neural network for backing up a tractor-trailer system is reported in [69]. The same technique was used to design an ANFIS controller for balancing an inverted pendulum [28]. Note that back-propagation through time is usually an off-line learning algorithms in the sense that the parameters will not be updated till the sequence ($k = 1$ to m) is over. If the sequence is too long or if we want to update the parameters in the middle of the sequence, we can always apply RTRL (real time recurrent learning) introduced earlier.

E. Feedback Linearization and Sliding Control

The equations of motion of a class of dynamic systems in continuous time domain can be expressed in the canonical form:

$$x^{(n)}(t) = f(x(t), \dot{x}(t), \cdots x^{(n-1)}(t)) + bu(t), \qquad (61)$$

where f is an unknown continuous function, b is the control gain, and $u \in R$ and $y \in R$ are the input and output of the system, respectively. The control objective is to force the state vector $\boldsymbol{x} = [x, \dot{x}, \ldots, x^{(n-1)}]^T$ to follow a specified desired trajectory $\boldsymbol{x}_d = [x_d, \dot{x}_d, \ldots, x_d^{(n-1)}]^T$. If we define the tracking error vector as $\mathbf{e} = \mathbf{x} - \mathbf{x}_d$, then the control objective is to design a control law $u(t)$ which ensures $\boldsymbol{e} \rightarrow 0$ as $t \rightarrow \infty$. (For simplicity, we assume $b = 1$ in the following discussion.)

Equation (61) is a typical **feedback linearizable** system since it can be reduced to a linear system if f is known exactly. Specifically, the following control law

$$u(t) = -f(\boldsymbol{x}(t)) + x_d^{(n)} + \boldsymbol{k}^T \boldsymbol{e} \qquad (62)$$

would transform the original nonlinear dynamics into a linear one:

$$e^{(n)}(t) + k_1 e^{(n-1)} + \cdots + k_n e = 0, \qquad (63)$$

where $\boldsymbol{k}=[k_n, \ldots, k_1]^T$ is an appropriately chosen vector that ensures satisfactory behavior of the close-loop linear system in equation (63).

Since f is unknown, an intuitive candidate of u would be

$$u = -F(\boldsymbol{x}, \boldsymbol{p}) + x_d^{(n)} + \boldsymbol{k}^T \boldsymbol{e} + v, \qquad (64)$$

where v is an additional control input to be determined later, F is an parameterized function (such as ANFIS, neural networks, or any other types of adaptive networks) that is rich enough to approximate f. Using this control law, the close-loop system becomes

$$e^{(n)} + k_1 e^{(n-1)} + \cdots + k_n e = (f - F) + v. \qquad (65)$$

Now the problem is divided into two tasks:

- How to update the parameter vector $\boldsymbol{p}$ incrementally so that $F(\boldsymbol{x}, \boldsymbol{p}) \approx f(\boldsymbol{x})$ for all $\boldsymbol{x}$.
- How to apply v to guarantee global stability while F is approximating f during the whole process.

The first task is not too difficult as long as F, which could be a neural network or a fuzzy inference system, is equipped with enough parameters to approximate f. For the second task, we need to apply the concept of a branch of nonlinear control theory called **sliding control** [102], [86]. The standard approach is to define an error metrics as

$$s(t) = (\frac{d}{dt} + \lambda)^{n-1} e(t), \text{ with } \lambda > 0. \qquad (66)$$

The equation $s(t) = 0$ defines a time varying hyperplane in R^n on which the tracking error vector $\boldsymbol{e}(t)$ $= [e(t), \dot{e}(t), \ldots, e^{n-1}(t)]^T$ decays exponentially to zero, so that perfect tracking can be obtained asymptotically. Moreover, if we can maintain the following condition:

$$\frac{d|s(t)|}{dt} \leq -\eta, \qquad (67)$$

then $|s(t)|$ will approach the hyperplane $|s(t)| = 0$ in a finite time less than or equal to $|s(0)|/\eta$. In other words, by maintain the condition in equation (67), $s(t)$ will approaches the sliding surface $s(t) = 0$ in a finite time, and then the error vector $\boldsymbol{e}(t)$ will converge to the origin exponentially with a time constant $(n-1)/\lambda$.

From equation (66), s can be rearranged as follows:

$$s = (\lambda + \frac{d}{dt})^{n-1} e = [\lambda^{n-1}, (n-1)\lambda^{n-2}, \ldots, 1]\mathbf{e}. \qquad (68)$$

Differentiate the above equation and plug in $e^{(n)}$ from equation (65), we obtain

$$\begin{array}{rcl} \frac{ds}{dt} & = & e^{(n)} + [0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]\mathbf{e} \\ & = & f - F + v - [k_n, k_{n-1}, \cdots, k_1]\mathbf{e} \\ & & +[0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]\mathbf{e} \end{array} \qquad (69)$$

By setting $[k_n, k_{n-1}, \cdots, k_1] = [0, \lambda^{n-1}, (n-1)\lambda^{n-2}, \cdots, \lambda]$, we have

$$\frac{ds}{dt} = f - F + v,$$

and

$$\begin{array}{rcl} \frac{d|s|}{dt} & = & \frac{ds}{dt} sgn(s) \\ & = & (f - F + v) sgn(s). \end{array}$$

That is, equation (67) is satisfied if and only if

$$(f - F + v) sgn(s) \leq -\eta.$$

If we assume the approximation error $|f - F|$ is bounded by a positive number A, then the above equation is always satisfied if

$$v = -(A + \eta) sgn(s).$$

To sum up, if we choose the control law as

$$u(t) = -F(\mathbf{x}, \mathbf{p}) + x_d^{(n)} + [0, \lambda^{n-1}, (n-1)\lambda^{n-2} \ldots \lambda]e - (A+\eta) sgn(s),$$

where $F(\mathbf{x}, \mathbf{p})$ is an adaptive network that approximates $f(\mathrm{x})$ and A is the error bound, then the close-loop system can achieve perfect tracking asymptotically with global stability.

This approach uses a number of nonlinear control design techniques and possesses rigorous proofs for global stability. However, its applicability is restricted to feedback linearizable systems. The reader is referred to [86] for a more detailed treatment of this subject. Applications of this technique to neural network and fuzzy control can be found in [82] and [104], respectively.

F. Gain Scheduling

Under certain arrangements, the first-order Sugeno fuzzy model becomes a gain scheduler that switches between several sets of feedback gains. For instance, a first-order Sugeno fuzzy controller for an hypothetical inverted pendulum system with varying pole length may have the following fuzzy if-then rules:

$$\begin{cases} \text{If pole is short, then } f_1 = k_{11}\theta + k_{12}\dot{\theta} + k_{13}z + k_{14}\dot{z}, \\ \text{If pole is medium, then } f_2 = k_{21}\theta + k_{22}\dot{\theta} + k_{23}z + k_{24}\dot{z}, \\ \text{If pole is long, then } f_3 = k_{31}\theta + k_{32}\dot{\theta} + k_{33}z + k_{34}\dot{z}. \end{cases} \tag{70}$$

This is in fact a gain scheduling controller, where the scheduling variable is the pole length and the control action is switching smoothly between three sets of feedback gains depending on the value of the scheduling variable. In general, the scheduling variables only appear in the premise part while the state variables only appear in the consequent part. The design method here is standard in gain scheduling: find several nominal points in the space formed by scheduling variables and employ any of the linear control design techniques to find appropriate feedback gains. If the number of nominal points is small, we can construct the fuzzy rules directly. On the other hand, if the number of nominal points is large, we can always use ANFIS to fit desired control actions to a fuzzy controller.

Examples of applying this method to both one-pole and two-pole inverted pendulum systems with varying pole lengths can be found in the demo programs in [31].

G. Others

Other design techniques that do not use the learning algorithm in neuro-fuzzy modeling are summarized here.

For complex control problems with perfect plant models, we can always use gradient-free optimization schemes, such as genetic algorithms [22], [19], simulated annealing [44], [45], downhill Simplex method [68], and random method [63], [88]. In particular, use of genetic algorithms for neural network controllers can be found in [113]; for fuzzy logic controllers, see [39], [52], [38].

If the plant model is not available, we can apply reinforcement learning [2] to find a working controller directly. The close relationship between reinforcement learning and dynamic programming was addressed in [3], [110]. Other variants of reinforcement learning includes temporal difference methods (TD(λ) algorithms) and Q-learning [108]. Representative applications of reinforcement learning to fuzzy control can be found in [4], [51], [12], [56].

Some other design and analysis approaches for fuzzy controllers include cell-to-cell mapping techniques [13], [87], model-based design method [99], self-organizing controllers [75], [100], and so on. As more and more people are working in this field, new design methods are coming out sooner than before.

VI. Concluding Remarks

A. A. Current Problems and Possible Solutions

A typical modeling problem includes **structure determination** and **parameter identification**. We address the parameter identification problem for ANFIS in this paper, which is solved via the back-propagation gradient descent and the least-squares method. The structure determination problem, which deals with the partition style, the number of MF's for each input, and the number of fuzzy if-then rules, and so on, is now an active research topic in the field. Work along this direction includes Jang's fuzzy CART approach [30], Lin's reinforcement learning method [57], Sun's fuzzy k-d trees [93], Sugeno's iterative method [92] and various clustering algorithms proposed by Chiu [15], Khedkar [43] and Wang [105]. Moreover, advances on the constructive and destructive learning of neural networks [18], [53] can also shed some lights on this problem.

Though we can speed up the parameter identification problem by introducing the least-squares estimator into the learning cycle, gradient descent still slows down the training process and the training time could be prohibitively long for a complicated task. Therefore the need to search for better learning algorithms hold equally true for both neural networks and fuzzy models. Variants of gradient descent proposed in the neural network literature; including second-order back-propagation [72], quick-propagation [17], and so on, can be used to speed up training. A number of techniques used in nonlinear regression can also contribute in this regard, such as the Guass-Newton method (linearization method) and the Marquardt procedure [61]. Another important resource is the rich literature of optimization, which offers many better gradient-based optimization routines, such as quadratic programming and conjugate gradient descent.

B. Future Directions

Due to the extreme flexibility of adaptive networks, ANFIS can have a number of variants that are different from what we have proposed here. For instance, we can replace the Π nodes in layer 2 of ANFIS with the parameterized T-norm operator [16] and let the learning algorithm decide the best T-norm function for a specific application. By employing the adaptive network as a common framework, we have also proposed other adaptive fuzzy models tailored for different purposes, such as the neuro-fuzzy classifier [94], [95] for data classification and the fuzzy filter scheme [96], [97] for feature extraction. There are a number of possible extensions and applications and they are currently under investigation.

During the past years, we have witnessed the rapid growth of the application of fuzzy logic and fuzzy set theory to consumer electronic products, automotive industry and process control. With the advent of fuzzy hardware with possibly on-chip learning capability, the applications to adaptive signal processing and control are expected. Potential applications within adaptive signal processing includes adaptive filtering [21], channel equalization [9], [10], [107], noise or echo cancelling [112], predictive coding [53], and so on.

Acknowledgments

The authors wish to thank Steve Chiu for providing numerous helpful comments. Most of this paper was finished while the first author was a research associate at UC Berkeley, so the authors would like to acknowledge the guidance and help of Professor Lotfi A. Zadeh and other members of the "fuzzy group" at UC Berkeley. Research supported in part by the BISC Program, NASA Grant NCC 2-275, EPRI Agreement RP 8010-34 and MICRO State Program No. 92-180.

References

[1] K. J. Aström and B. Wittenmark. *Computer Controller Systems: Theory and Design.* Prentice-Hall, 1984.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[3] A. G. Barto, R. S. Sutton, and C.J.C.H Watkins. Learning and sequential decision making. In M. Gabriel and J. W. Moore, editors, *Learning and Computational Neuroscience.* MIT Press, Cambridge, 1991.

[4] H. R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Trans. on Neural Networks*, 3(5):724–740, 1992.

[5] J. A. Bernard. Use of rule-based system for process control. *IEEE Control Systems Magazine*, 8(5):3–13, 1988.

[6] P. Bonissone, V. Badami, K. Chiang, P. Khedkar, K. Marcelle, and M. Schutten. Industrial applications of fuzzy logic at general electric. *The Proceedings of the IEEE*, March 1995.

[7] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[8] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. on Neural Networks*, 2(2):302–309, March 1991.

[9] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant. Adaptive equalization of finite nonlinear channels using multilayer perceptrons. *Signal Processing*, 20:107–119, 1990.

[10] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant. Reconstruction of binary signals using an adaptive radial-basis-function equalizer. *Signal Processing*, 22:77–93, 1991.

[11] V. C. Chen and Y. H. Pao. Learning control with neural networks. In *Proc. of International Conference on Robotics and Automation*, pages 1448–1453, 1989.

[12] Y.-Y. Chen. A self-learning fuzzy controller. In *Proc. of IEEE international conference on fuzzy systems*, March 1992.

[13] Y.-Y. Chen and T.-C. Tsao. A description of the dynamic behavior of fuzzy systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(4):745–755, July 1989.

[14] S. Chiu, S. Chand, D. Moore, and A. Chaudhary. Fuzzy logic for control of roll and moment for a flexible wing aircraft. *IEEE Control Systems Magazine*, 11(4):42–48, 1991. 1991.

[15] S. L. Chiu. Fuzzy model identification based on cluster estimation. *Journal of Intelligent and Fuzzy Systems*, 2(3), 1994.

[16] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic press, New York, 1980.

[17] S. E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 38–51, Carnegic Mellon University, 1988.

[18] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, G. Hinton, and T. Sejnowski, editors, *Advances in Neural Information Processing Systems II.* Morgan Kaufmann, 1990.

[19] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[20] G. C. Goodwin and K. S. Sin. *Adaptive filtering prediction and control.* Prentice-Hall, Englewood Cliffs, N.J., 1984.

[21] S. S. Haykin. *Adaptive filter theory.* Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1991.

[22] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, 1975.

[23] T. C. Hsia. *System Identification: Least-Squares Methods.* D.C. Heath and Company, 1977.

[24] J.-S. Roger Jang. Fuzzy modeling using generalized neural networks and Kalman filter algorithm. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 762–767, July 1991.

[25] J.-S. Roger Jang. Rule extraction using generalized neural networks. In *Proc. of the 4th IFSA World Congress*, pages 82–86 (in the Volume for Artificial Intelligence), July 1991.

[26] J.-S Roger Jang. A self-learning fuzzy controller with application to automobile tracking problem. In *Proc. of IEEE Roundtable Discussion on Fuzzy and Neural Systems, and Vehicle Application*, page paper no. 10, Tokyo, Japan, November 1991. Institute of Industrial Science, Univ. of Tokyo.

[27] J.-S. Roger Jang. Fuzzy controller design without domain experts. In *Proc. of IEEE international conference on fuzzy systems*, March 1992.

[28] J.-S. Roger Jang. Self-learning fuzzy controller based on temporal back-propagation. *IEEE Trans. on Neural Networks*, 3(5):714–723, September 1992.

[29] J.-S. Roger Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(03):665–685, May 1993.

[30] J.-S. Roger Jang. Structure determination in fuzzy modeling: a fuzzy CART approach. In *Proc. of IEEE international conference on fuzzy systems*, Orlando, Florida, June 1994.

[31] J.-S. Roger Jang and N. Gulley. The Fuzzy Logic Toolbox for use with MATLAB. The MathWorks, Inc., Natick, Massachusetts, 1995.

[32] J.-S. Roger Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. on Neural Networks*, 4(1):156–159, January 1993.

[33] J.-S. Roger Jang and C.-T. Sun. Predicting chaotic time series with fuzzy if-then rules. In *Proc. of IEEE international conference on fuzzy systems*, San Francisco, March 1993.

[34] J.-S. Roger Jang and C.-T. Sun. Neuro-fuzzy modeling: an computational approach to intelligence, 1995. Submitted for publication.

[35] R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, and P. S. Lewis. Function approximation and time series prediction with neural networks. In *Proc. of IEEE International Joint Conference on Neural Networks*, pages I–649–665, 1990.

[36] M. I. Jordan and R. A. Jacobs. Hierarchical mextures of experts and the EM algorithm. Technical report, M.I.T., 1993.

[37] A. Kandel, editor. *Fuzzy expert systems.* CRC Press, Boca Raton, FL, 1992.

[38] C. L. Karr. GAs for fuzzy controllers. *AI Expert*, 6(2):26–33, February 1991.

[39] C. L. Karr and E. J. Gentry. Fuzzy control of pH using genetic algorithms. *IEEE Trans. on Fuzzy Systems*, 1(1):46–53, February 1993.

[40] Y. Kasai and Y. Morimoto. Electronically controlled continuously variable transmission. In *Proc. of International Congress Transportation Electronics*, Dearborn, Michigan, 1988.

[41] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[42] D. J. Kelly, P. D. Burton, and M.A. Rahman. The application of a neural fuzzy controller to process control. In *Proc. of the International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference, the Industrial Fuzzy Control and Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic*, San Antonio, Texas, December 1994.

[43] P. S. Khedkar. *Learning as Adaptive Interpolation in Neural Fuzzy Systems.* PhD thesis, Computer Science Division, Department of EECS, University of California at Berkeley, 1993.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Research Report 9335 IBM T. J. Watson Center*, 1983.

[45] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[46] M. S. Klassen and Y.-H. Pao. Characteristics of the functional-link net: A higher order delta rule net. In *IEEE Proc. of the International Conference on Neural Networks*, San Diego, June 1988.

[47] B. Kosko. *Neural networks and fuzzy systems: a dynamical systems approach.* Prentice Hall, Englewood Ciffs, NJ, 1991.

[48] A. S. Lapedes and R. Farber. Nonlinear signal processing using neural networks: prediction and system modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, 1987.

[49] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 1. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):404–418, 1990.

[50] C.-C. Lee. Fuzzy logic in control systems: fuzzy logic controller-part 2. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):419–435, 1990.

[51] C.-C. Lee. A self-learning rule-based controller employing approximate reasoning and neural net concepts. *International Journal of Intelligent Systems*, 5(3):71–93, 1991.

[52] M. A. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proc. of the second IEEE International Conference on Fuzzy Systems*, pages 612–617, San Francisco, 1993.

[53] T.-C. Lee. *Structure level adaptation for artificial neural networks.* Kluwer Academic Publishers, 1991.

[54] Fujitec Company Limited. FLEX-8800 series elevator group control system, 1988. Osaka, Japan.

[55] C.-T. Lin and C. S. G. Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Trans. on Computers*, 40(12):1320–1336, December 1991.

[56] C.-T. Lin and C.-S. G. Lee. Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. In *Proc. of IEEE International Conference on Fuzzy Systems*, pages 88–93, San Francisco, March 1993.

[57] C.-T. Lin and C.-S. G. Lee. Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. *IEEE Trans. on Fuzzy Systems*, 2(1):46–63, 1994.

[58] L. Ljung. *System identification: theory for the user.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[59] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, July 1977.

[60] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[61] D. W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 2:431–441, 1963.

[62] S. J. Mason. Feedback theory – further properties of signal flow graphs. *Proc. IRE*, 44(7):920–926, July 1956.

[63] J. Matyas. Random optimization. *Automation and Remote Control*, 26:246–253, 1965.

[64] M. Minsky and S. Papert. *Perceptrons.* MIT Press, MA, 1969.

[65] J. Moody. Fast learning in multi-resolution hierarchies. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, chapter 1, pages 29–39. Morgan Kaufmann, San Mateo, CA, 1989.

[66] J. Moody and C. Darken. Learning with localized receptive fields. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School.* Carnegie Mellon University, Morgan Kaufmann Publishers, 1988.

[67] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[68] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1964.

[69] D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, pages 18–23, April 1990.

[70] N.J. Nilsson. *Learning machines: foundations of trainable pattern classifying systems.* McGraw-Hill, New York, 1965.

[71] Y.-H. Pao. *Adaptive Pattern Recognition and Neural Networks*, chapter 8, pages 197–222. Addison-Wesley Publishing Company, Inc., 1989.

[72] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order Hebbian learning. In *Proc. of IEEE International Conference on Neural Networks*, pages 593–600, 1987.

[73] N. Pfluger, J. Yen, and R. Langari. A defuzzification strategy for a fuzzy logic controller employing prohibitive information in command formulation. In *Proc. of IEEE international conference on fuzzy systems*, pages 717–723, San Diego, March 1992.

[74] M.J.D. Powell. Radial basis functions for multivariable interpolation: a review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Oxford University Press, 1987.

[75] T. J. Procyk and E. H. Mamdani. A linguistic self-organizing process controller. *Automatica*, 15:15–30, 1978.

[76] D. Psaltis, A. Sideris, and A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, 8(4):17–21, April 1988.

[77] III R. S. Crowder 'Predicting the Mackey-Glass timeseries with cascade-correlation learning. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 117–123, Carnegic Mellon University, 1990.

[78] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms.* Spartan, New York, 1962.

[79] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volum 1*, chapter 8, pages 318–362. The MIT Press, 1986.

[80] T. A. Runkler and M. Glesner. Defuzzification and ranking in the context of membership value semantics, rule modality, and measurement theory. In *European Congress on Fuzzy and Intelligent Technologies*, Aachen, September 1994.

[81] T. D. Sanger. A tree-structured adaptive network for function approximate in high-dimensional spaces. *IEEE Trans. on Neural Networks*, 2(2):285–293, March 1991.

[82] R. M. Sanner and J. J. E. Slotine. Gaussian networks for direct adaptive control. *IEEE Trans. on Neural Networks*, 3:837–862, 1992.

[83] S. Shah, F. Palmieri, and M. Datum. Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, 5(5):779–787, 1992.

[84] S. Shar and F. Palmieri. MEKA-a fast, local algorithm for training feedforward neural networks. In *Proc. of International Joint Conference on Neural Networks*, pages III 41–46, 1990.

[85] S. Singhal and L. Wu. Training multilayer perceptrons with the extended kalman algorithm. In David S. Touretzky, editor, *Advances in neural information processing systems I*, pages 133–140. Morgan Kaufmann Publishers, 1989.

[86] J.-J. E. Slotine and W. Li. *Applied nonlinear control.* Prentice Hall, 1991.

[87] S. M. Smith and D. J. Comer. Automated calibration of a fuzzy logic controller using a cell state space algorithm. *IEEE Control Systems Magazine*, 11(5):18–28, August 1991.

[88] F. J. Solis and J. B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[89] K. Stokbro, D. K. Umberger, and J. A. Hertz. Exploiting neurons with localized receptive fields to learn chaos. *Complex Systems*, 4:603–622, 1990.

[90] M. Sugeno, editor. *Industrial applications of fuzzy control.* Elsevier Science Pub. Co., 1985.

[91] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15–33, 1988.

[92] M. Sugeno and T. Yasukawa. A fuzzy-logic-based approach to qualitative modeling. *IEEE Trans. on Fuzzy Systems*, 1(1):7–31, February 1993.

[93] C.-T. Sun. Rulebase structure identification in an adaptive network based fuzzy inference system. *IEEE Trans. on Fuzzy Systems*, 2(1):64–73, 1994.

[94] C.-T Sun and J.-S. Roger Jang. Adaptive network based fuzzy classification. In *Proc. of the Japan-U.S.A. Symposium on Flexible Automation*, July 1992.

[95] C.-T. Sun and J.-S. Roger Jang. A neuro-fuzzy classifier and its applications. In *Proc. of IEEE international conference on fuzzy systems*, San Francisco, March 1993.

[96] C.-T. Sun, J.-S. Roger Jang, and C.-Y. Fu. Neural network analysis of plasma spectra. In *Proc. of the International Conference on Artificial Neural Networks*, Amsterdam, September 1993.

[97] C.-T. Sun, T.-Y. Shuai, and G.-L. Dai. Using fuzzy filters as feature detectors. In *Proc. of IEEE international conference on fuzzy systems*, pages 406–410 (Vol I), Orlando, Florida, June 1994.

[98] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Trans. on Systems, Man, and Cybernetics*, 15:116–132, 1985.

[99] K. Tanaka and M. Sugeno. Stability analysis and design of fuzzy control systems. *Fuzzy Sets and Systems*, 45:135–156, 1992.

[100] R. Tanscheit and E. M. Scharf. Experiments with the use of a rule-based self-organizing controller for robotics applications. *Fuzzy Sets and Systems*, 26:195–214, 1988.

[101] Y. Tsukamoto. An approach to fuzzy reasoning method. In Madan M. Gupta, Rammohan K. Ragade, and Ronald R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*, pages 137–149. North-Holland, Amsterdam, 1979.

[102] V. I. Utkin. Variable structure systems with sliding mode: a survey. *IEEE Trans. on Automatic Control*, 22:212, 1977.

[103] K. P. Venugopal, R. Sudhakar, and A. S. Pandya. An improved scheme for direct adaptive control of dynamical systems using backpropagation neural networks. *Journal of Circuits, Systems and Signal Processing*, 1994. (Forthcoming).

[104] L.-X. Wang. Stable adaptive fuzzy control of nonlinear systems. *IEEE Trans. on Fuzzy Systems*, 1(1):146–155, 1993.

[105] L.-X. Wang. Traning fuzzy logic systems using nearest neighborhood clustering. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Francisco, March 1993.

[106] L.-X. Wang and J. M. Mendel. Back-propagation fuzzy systems as nonlinear dynamic system identifiers. In *Proc. of the IEEE International Conference on Fuzzy Systems*, San Diego, March 1992.

[107] L.-X. Wang and J. M. Mendel. Fuzzy adaptive filters, with application to nonlinear channel equalization. *IEEE Trans. on Fuzzy Systems*, 1(3):161–170, 1993.

[108] C.J.C.H Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[109] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[110] Paul J. Werbos. A menu of designs for reinforcement learning over time. In III W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, chapter 3. The MIT Press, Bradford, 1990.

[111] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceptron, madline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

[112] B. Widrow and D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1985.

[113] Alexis P. Wieland. Evolving controls for unstable systems. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 91–102, Carnegie Mellon University, 1990.

[114] R. J. William and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[115] R. R. Yager and D. P. Filev. SLIDE: A Simple Adaptive Defuzzification Method. *IEEE Transactions on Fuzzy Systems*, 1(1):69–78, February 1993.

[116] S. Yasunobu and G. Hasegawa. Evaluation of an automatic container crane operation system based on predictive fuzzy control. *Control Theory and Advanced Technology*, 2(2):419–432, 1986. 1986.

[117] S. Yasunobu and S. Miyamoto. Automatic train operation by predictive fuzzy control. In M. Sugeno, editor, *Industrial Applications of Fuzzy Control*, pages 1–18. North-Holland, Amsterdam, 1985.

[118] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[119] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. on Systems, Man, and Cybernetics*, 3(1):28–44, January 1973.

Jyh-Shing Roger Jang was born in Taipei, Taiwan in 1962. He received the B.S. degree in electrical engineering from National Taiwan University in 1984, and the Ph.D. degree in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, in 1992.

During the summer of 1989, he was a summer student in NASA Ames Research Center, working on the design and implementation of fuzzy controllers. Between 1991 and 1992 he was a research scientist in the Lawrence Livermore National Laboratory, working on spectrum modeling and analysis using neural networks and fuzzy logic. After obtaining his Ph.D. degree, he was a research associate in in the same department, working on machine learning techniques using fuzzy logic. Since September 1993, he has been with The MathWorks, Inc., working on the Fuzzy Logic Toolbox used with MATLAB.

His interests lie in the area of neuro-fuzzy modeling, system identification, machine learning, nonlinear regression, optimization, and computer aided control system design.

Dr. Jang is a member of IEEE.

Chuen-Tsai Sun received his B.S. degree in Electrical Engineering in 1979 and his M.A. degree in History in 1984, both from National Taiwan University, Taiwan. He received his Ph.D degree in Computer Science from the University of California at Berkeley in 1992. His Ph.D. research advisor was Professor Lotfi A. Zadeh, the initiator of fuzzy set theory.

During the period of 1989 and 1990 he worked as a consultant with the Pacific Gas and Electric Company, San Francisco, in charge of designing and implementing an expert system for protective device coordination in electric distribution circuits. Between 1991 and 1992 he was a research scientist in the Lawrence Livermore National Laboratory, working on plasma analysis using neural networks and fuzzy logic. Since August 1992, he has been on the faculty of the Department of Computer and Information Science at National Chiao Tung University. His current research interests include computational intelligence, system modeling, and computer assisted learning.

Dr. Sun is a member of IEEE. He was the Arthur Gould Tasheira Scholarship winner in 1986. He was also honored with the Phi Hua Scholar Award in 1985 for his publications in history.

Liens

http://www.mathworks.com/products/fuzzylogic.html

http://www.mathtools.net/MATLAB/Fuzzy_Logic/index.html

http://www.dbai.tuwien.ac.at/marchives/fuzzy-mail/

http://europa.eu.int/comm/research/rtdinfo/fr/24/02.html

http://www.gala.univ-perp.fr/~polit/chap0.html

http://www.eru.ulaval.ca/ptt15225

http://vcampus.u-strasbg.fr/uticeweb/mhiri/projet_V2

http://www.mathworks.fr/products/controldesign/modanal.shtml

Bibliographie

[1] H. Bandemer and S. Gottwald. *Fuzzy sets fuzzy logic fuzzy Methods with applications.* Edition Wiley, England, 1995.

[2] H. Bühler. *Reglage par logique floue.* Presses polythechniques et universitaires romandes, Lausane, Suisse, 1994.

[3] P. Borne, J. Rozeinoer, J.R. Dieulot, and L. Dubois. *Introduction à la commande floue.* Edition Technip, Paris, 1998.

[4] L. Foulloy. *Logique floue.* Masson, Paris, 1994.

[5] Jyh-Shing Roger Jang and Chuen-Tsai Sun. Neuro-fuzzy modeling and control. *Proceedings of the IEEE*, 1995.

[6] Jyh-Sing Roger Jang. ANFIS : Adaptive-Network-Based Fuzzy Inference System. *IEEE Transactions on Systems, Man, and Cybernetics*, 23 :665–684, 1993.

[7] H. T. Nguyen, M. Sugeno, R. Tong, and R. R. Yager. *The theoretical aspects of fuzzy control.* Edition Wiley, England, 1995.

Troisième partie

Annexes

Annexe A

Examens et corrigés

$3^{\text{ème}}$ année
Option automatique

Examen d'Identification et Commande Floue

Cet examen consiste à modéliser et à commander le système de commande d'altitude d'un avion. Il s'agit dans un premier temps de déterminer un modèle simple du système de façon graphique, puis de commander le système par un régulateur flou. Enfin, le système étant corrigé par ce régulateur flou on déterminera un modèle paramétrique plus précis.

A.1 Méthodes graphiques

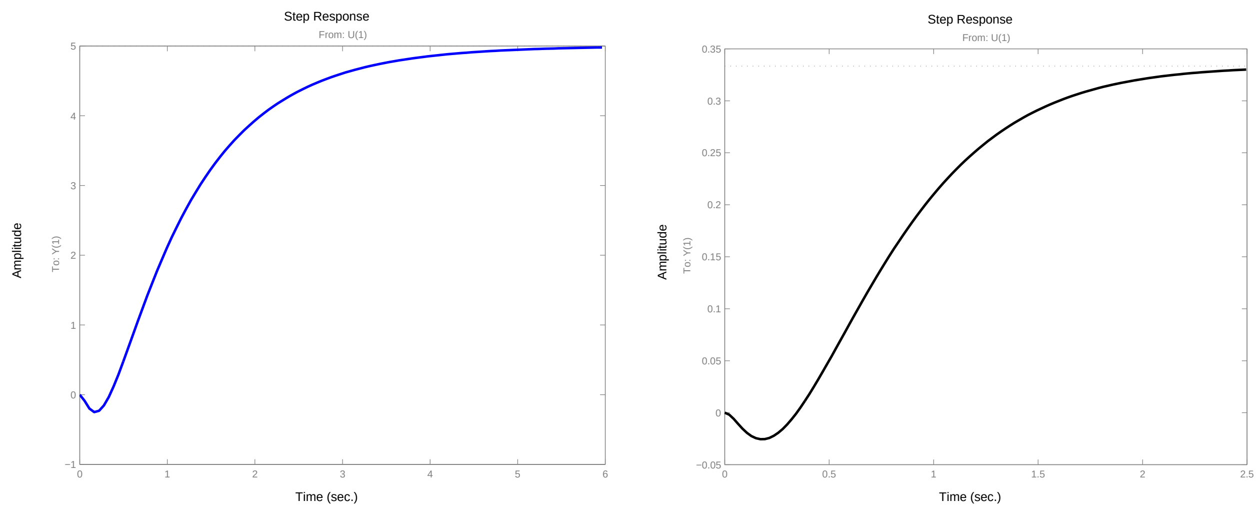


FIGURE A.1 – Réponses du système à un échelon : à gauche en boucle ouverte, à droite en boucle fermée.

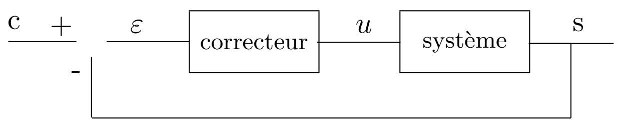


FIGURE A.2 – Identification en boucle fermée

1. La réponse à un échelon du système en boucle ouverte est donnée sur la figure A.4(gauche). En faisant l'hypothèse que le système est un système du premier ordre avec un retard pur, donnez les valeurs numériques d'un tel modèle.
2. Toujours sur cette même figure appliquez la méthode de Strejc afin d'obtenir un modèle plus précis.
3. Ce même système est bouclé par un retour unitaire et un gain de 0,1 en guise de correcteur comme indiqué sur la figure A.2. Sa réponse à l'échelon unité est donnée sur la figure A.4(droite).
   (a) Déterminer un modèle du système en boucle fermée.
   (b) En négligeant le retard pur, déduisez-en un modèle en boucle ouverte.
4. Finalement, on insère un gain variable dans la boucle et il s'avère que le système oscille pour un gain $k = 1$.
   (a) Ce résultat est-il concordant avec vos modèles (raisonnement qualitatif !) ?
   (b) Au vu des résultats précédents, proposez un modèle à trois constantes de temps
5. Ce système présente de façon évidente un zéro dit instable !
   (a) Proposez alors un modèle du système incluant ce zéro instable.
   (b) Donnez une valeur numérique très approchée (pratiquement sans calcul) de ce zéro agrémenté d'une ligne de commentaire sur ce choix.

A.2 Commande par contrôleur flou

Finalement, le système précédent est commandé par un contrôleur flou.

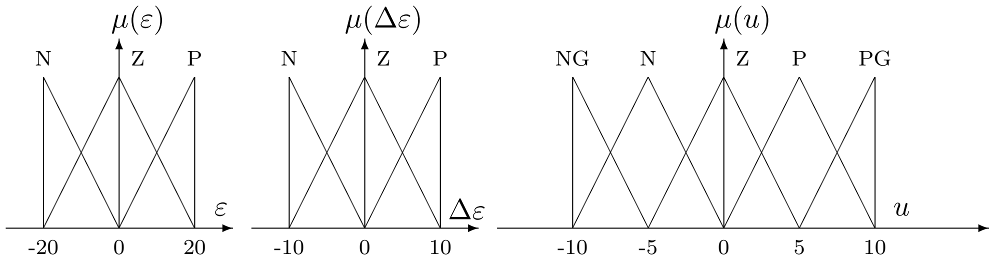


FIGURE A.3 – Fuzzification des variables

$\varepsilon \setminus \Delta\varepsilon$	N	Z	P
N	NG	N	Z
Z	N	Z	P
P	Z	P	PG

avec opérateurs logiques : ET = min OU = max
méthode d'inférence : somme-produit
défuzzification : moyenne des maximums.

1. En supposant $\varepsilon = 10$ et $\Delta\varepsilon = 0$, calculez la sortie u du contrôleur.
2. Toujours en supposant $\Delta\varepsilon = 0$, calculez la sortie u du contrôleur en fonction de la valeur de ε. Tracez la courbe $u = f(\varepsilon)$ en précisant les points importants et la forme de la courbe entre ses points (droites, paraboles, ...)
3. Même question mais en utilisant les hypothèses suivantes :
   – opérateurs logiques : ET = min OU = max
   – méthode d'inférence : max-min
   – défuzzification : centre de gravité

A.3 Modèles paramétriques

Le système étant corrigé par le régulateur flou précédent, on décide de déterminer un modèle paramétrique du système corrigé. Dans ce qui suit, et quels que soient vos résultats précédents, on supposera que le système présente :

– 3 constantes de temps de l'ordre de : 0,1, 0,05 et 0,02

1. Calculer la fréquence d'échantillonnage nécessaire pour pouvoir identifier correctement ces trois constantes de temps.
2. Calculer la longueur de la séquence binaire pseudo aléatoire en fonction de cette fréquence d'échantillonnage pour bien identifier le gain du système.
3. Quelle est alors le temps d'excitation du système ?
4. On opte pour une méthode d'estimation des paramètres dite de la matrice instrumentale, définie par :

$$\overset{\wedge}{\theta}= (Z^T X)^{-1} Z^T y \text{ avec } \begin{cases} \overset{\wedge}{\theta}, & \text{Paramètres estimés ;} \\ Z, & \text{Matrice instrumentale ;} \\ y, & \text{Vecteur des sorties mesurées ;} \\ X, & \text{Construite à partir du jeu d'E/S } (y = X\theta + e) \text{ ;} \\ e, & \text{Résidus.} \end{cases}$$

Montrez que pour que cet estimateur soit sans biais, il faut que

$$\begin{aligned} E[Z^T X] &\neq 0 \\ E[Z^T e] &= 0 \end{aligned}$$

3$^{\text{ème}}$ année
Option automatique

Corrigé de l'Examen d'Identification et Commande Floue

A.4 Méthodes graphiques

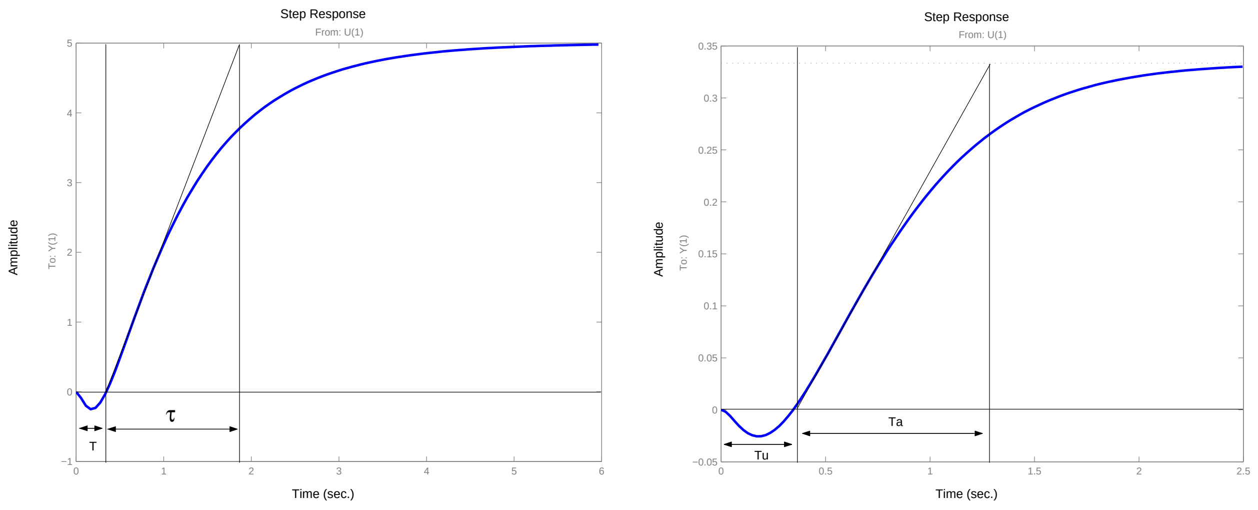


FIGURE A.4 – Réponses du système à un échelon : à gauche en boucle ouverte, à droite en boucle fermée.

1. Modèle du premier ordre
   On mesure $T = 0,3s$ et $\tau = 1,9 - 0,3 = 1,6s$ le modèle est donc :

$$G(p) = \frac{Ae^{-Tp}}{1+\tau p} = \frac{5e^{-0,3p}}{1+1,6p}$$

2. Modèle Strejc en boucle ouverte
   On mesure $Tu = 0,3s$ et $Ta = 1,6s$. En appliquant la méthode :

$$\begin{aligned} Tu/Ta = 0,1875 &\quad\Rightarrow\quad n = 2 \\ Ta/\tau = 2,718 &\quad\Rightarrow\quad \tau = Ta/2,718 = 0,5887 \\ Tu/\tau = 0,282 &\quad\Rightarrow\quad Tu' = 0.1660 \\ Tr = Tu - Tu' &\quad\Rightarrow\quad Tr = 0.166 \end{aligned}$$

   d'où le modèle :

$$G_s(p) = \frac{Ae^{-Trp}}{(1+\tau p)^n} = \frac{5e^{-0,166p}}{(1+0,5887p)^2}$$

3. En boucle fermée
   (a) Modèle du système en boucle fermée,
       On mesure $Tu = 0,33s$ et $Ta = 1s$. En appliquant la méthode :

$$\begin{aligned} Tu/Ta = 0,330 &\quad\Rightarrow\quad n = 4 \\ Ta/\tau = 4,463 &\quad\Rightarrow\quad \tau = Ta/4,463 = 0,2241 \\ Tu/\tau = 1,425 &\quad\Rightarrow\quad Tu' = 0,319 \\ Tr = Tu - Tu' &\quad\Rightarrow\quad Tr = 0.01 \end{aligned}$$

d'où le modèle :

$$G_s(p) = \frac{Ae^{-Trp}}{(1+\tau p)^n} = \frac{0,34e^{-0,01p}}{(1+0,224p)^4}$$

(b) Modèle en boucle ouverte déduit :

$$G_s(p) = \frac{A/k}{\tau^4p^4 + 4\tau^3p^3 + 6\tau^2p^2 + 4\tau p + 1 - A}$$

4. Finalement, on insère un gain variable dans la boucle et il s'avère que le système oscille pour un gain $k = 1$.
   (a) A priori oui, les deux modèles obtenus sont susceptibles d'être instables dans une boucle fermée si le gain est suffisamment grand.
   (b) le plus simple est de reprendre le modèle obtenu par la méthode de Strejc sur la boucle ouverte :

$$G_s(p) = \frac{Ae^{-Trp}}{(1+\tau p)^n} \simeq \frac{A}{(1+\tau p)^2(1+Trp)}$$

5. Introduction du zéro instable
   (a) Encore une fois, en reprenant le modèle précédent et en développant au premier ordre l'exponentielle, on obtient :

$$G_s(p) = \frac{Ae^{-Trp}}{(1+\tau p)^n} \simeq \frac{A(1-Trp)}{(1+\tau p)^2}$$

   (b) $Tr = 0.166$. En fait pour tenir compte des termes que l'on néglige on augmente "un peu" la valeur de Tr. Ainsi $Tr' = 1,5Tr$ donne de bien meilleurs résultats.

A.5 Commande par contrôleur flou

1. Si $\varepsilon = 10$ et $\Delta\varepsilon = 0$, la sortie vaut 2,5.
2. En supposant $\Delta\varepsilon = 0$, calculez la sortie u du correcteur en fonction de ε

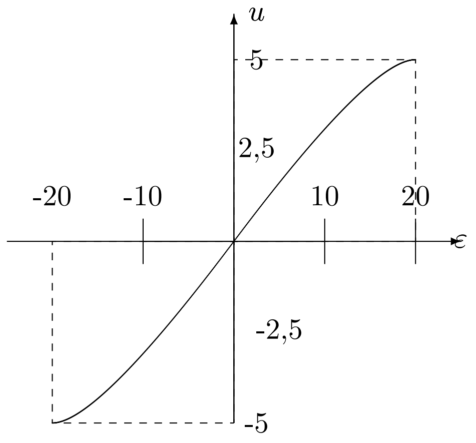


FIGURE A.5 – $u = f(\varepsilon)$(courbe exagérée !)

3. avec :
   – opérateurs logiques : ET = min OU = max
   – méthode d'inférence : max-min
   – défuzzification : centre de gravité

A.6 Modèles paramétriques

Le système présente 3 constantes de temps de l'ordre de : 0,1, 0,05 et 0,02.

1. Une SBPA est un bruit blanc sur $0,3f_e$ donc :

$$0,3f_e \geq \frac{1}{2\pi\tau_{min}} \Rightarrow f_e \geq \frac{1}{0,3\times 2\pi\tau_{min}} = \frac{1}{0,3\times 2\pi\times 0.02} = 26,5Hz$$

2. Le plateau maximum est de NT_e avec $T_e = 1/f_e$ et $2^N - 1$ la longueur de la SBPA.
   Il faut donc que :

$$NT_e \geq 3 \text{ à } 5\tau_{max} \Rightarrow N \geq 3 \text{ à } 5f_e\tau_{max} = 3 \text{ à } 5\times 26,5\times 0,1 = 8 \text{ à } 13$$

   $N = 10$ semble un bon compromis.

3. Le temps d'excitation est :

$$T_{excitation} = (2^N - 1)T_e = 1023T_e = 38,6s$$

   mais en général on envoie deux séquences consécutives pour éviter de prendre en compte le transitoire au début de la séquence, donc $T_{excitation} \simeq 77,2s$

4. Méthode de la matrice instrumentale :

$$\overset{\wedge}{\theta} = (Z^TX)^{-1}Z^Ty \text{ avec } \begin{cases} \overset{\wedge}{\theta}, & \text{Paramètres estimés;} \\ Z, & \text{Matrice instrumentale;} \\ y, & \text{Vecteur des sorties mesurées;} \\ X, & \text{Construite à partir du jeu d'E/S } (y = X\theta + e)\text{;} \\ e, & \text{Résidus.} \end{cases}$$

$$\begin{aligned} \overset{\wedge}{\theta} &= (Z^TX)^{-1}Z^T(X\theta + e) \\ &= \theta + (Z^TX)^{-1}Z^Te \end{aligned}$$

   pour que $E[\overset{\wedge}{\theta}] = \theta$, il faut que :

$$E[(Z^TX)^{-1}Z^Te] = 0$$

   donc que :

$$\begin{aligned} E[Z^TX] &\neq 0 \\ E[Z^Te] &= 0 \end{aligned}$$

$3^{\text{ème}}$ année
Option mécatronique

Examen d'Identification

Cet examen consiste à modéliser un bras de robot présentant deux types de dérives lentes. En premier lieu, la constante de temps principale varie dans le temps puis le système présente une dérive lente de la valeur moyenne de sortie.

A.7 Méthodes graphiques

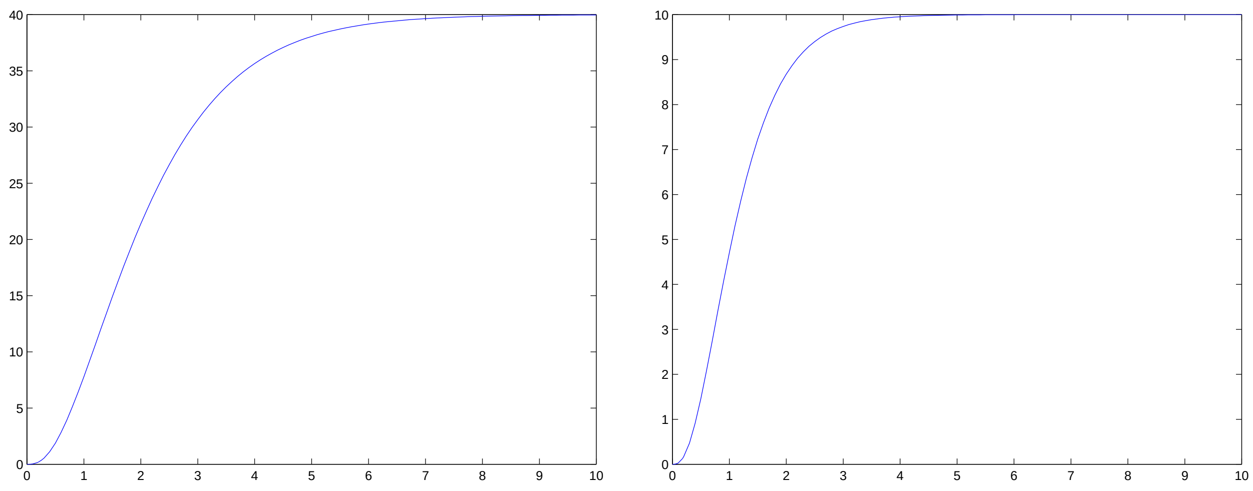


FIGURE A.6 – Réponse du système à un échelon au début de l'identification (gauche) et après une heure (droite).

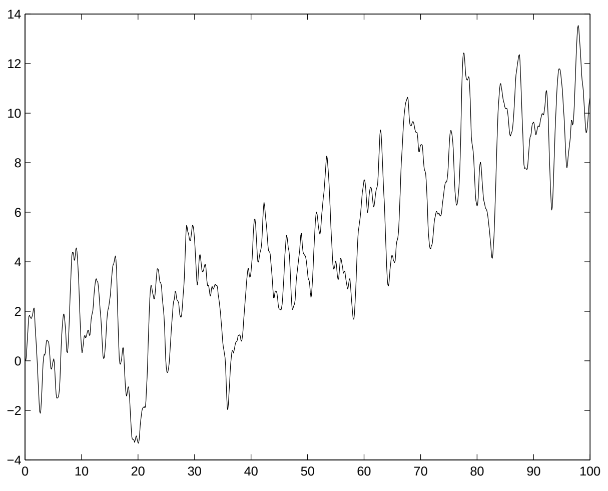


FIGURE A.7 – Réponse du système à une excitation de type SBPA

A.8 Modèles de Strecj

1. La réponse à un échelon du système en boucle ouverte est donnée sur la figure A.6(gauche). En faisant l'hypothèse que le système est un système du premier ordre avec un retard pur, donnez les valeurs numériques d'un tel modèle.
2. Toujours sur cette même figure appliquez la méthode de Strejc afin d'obtenir un modèle plus précis.
3. Appliquez Strecj sur la figure A.6(droite) et déduisez-en un nouveau modèle du système.
4. En faisant l'hypothèse d'une variation linéaire de la constante de temps dominante et à l'aide des deux modèles précédents, proposez un modèle du système $H(p, t)$

A.9 Modèles ARX

Afin d'étudier plus précisément les autres constantes de temps du système, vous décidez de soumettre le système à une séquence binaire pseudo-aléatoire. Le résultat est donné fig. A.7

1. Rappeler le principe de calcul des modèles ARX et la formule appliquée pour obtenir une estimation des paramètres du système.
2. La première tentative de modélisation ARX est très mauvaise ! Quels sont les deux phénomènes qui "empêchent" d'avoir un modèle ARX correct.
3. Afin de compenser la dérive lente de la valeur moyenne, vous envisagez deux méthodes
   (a) Boucler le système, et faire une identification en boucle fermée,
   (b) Enlever cette dérive par un traitement numérique hors ligne (un peu comme on enlève la valeur moyenne).
4. Proposez un correcteur simple pour le bouclage du système
5. Donnez la relation entre $y(t)$ valeurs de la figure A.7 et $y_1(t)$ qui représente $y(t)$ sans la dérive. Les valeurs numériques approchées seront prises sur la figure A.7.
6. Après calcul des modèles ARX, les modèles restent passablement mauvais avec un léger avantage pour le modèle calculé à partir des valeurs en boucle fermée, pourquoi ?
7. Les méthodes précédentes ne donnant pas de modèles suffisants pour l'application envisagée, proposez une méthode qui permet d'identifier ce système malgré toutes ses imperfections.

3$^{\text{ème}}$ année
Option mécatronique

Examen de commande floue

Cet examen est destiné à déterminer un estimateur flou de pluie en fonction de la pression atmosphérique, du taux d'hygrométrie et de la température. Les valeurs numériques seront estimées grossièrement à partir des graphiques donnés où les abscisses sont en jours.

A.10 Fuzzification des variables

1. Après avoir observé les relations entre quantité de pluie et les variables d'entrée en figure A.9(pression atmosphérique, taux d'hygrométrie, température et leurs dérivées) peu de temps avant le début des précipitations, proposez un choix d'ensembles flous pertinents. justifiez brièvement le choix du nombre d'ensembles flous par variable et la forme de ses ensembles.
2. La forme de la variable de sortie est donnée figure A.8. Etablissez les règles d'inférence pour chaque précipitation.

L'estimateur s'avère être trop cher (trop de capteurs) et votre cahier des charges devient donc "estimateur de pluie uniquement fonction de la pression atmosphérique et de ses variations".

1. proposez, dans ce cas, les règles d'inférence sous la forme d'un tableau :

$p \setminus \Delta p$	N	Z	P
N			
Z			
P			

2. Proposez une méthode pour chaque point suivant :
   – opérateurs logiques : ET = ? OU = ?,
   – méthode d'inférence,
   – défuzzification.
3. Après un premier test en simulation il s'avère que votre estimateur prédit correctement dans 70% des cas mais le service marketing vous explique qu'il serait bon de donner outre le fait qu'il pleuve ou pas une estimation de la probabilité d'orage. L'orage est probable s'il y a une baisse de 1 à 2 hPa de la pression atmosphérique en une heure. Proposez un deuxième estimateur flou donnant cette indication.
4. Question subsidiaire pour ceux qui ont suivit le cours d'identification. D'après vous, un modèle de type ARX serait-il meilleur ou moins bon pour ce type de prévisions ?

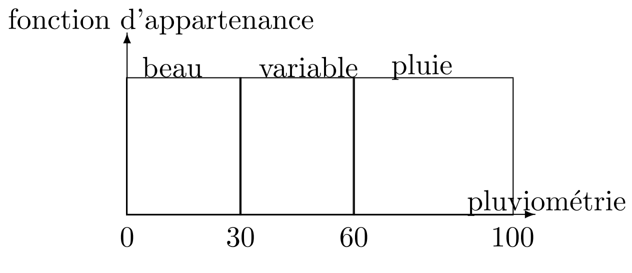


FIGURE A.8 – Fuzzification de la variable de sortie

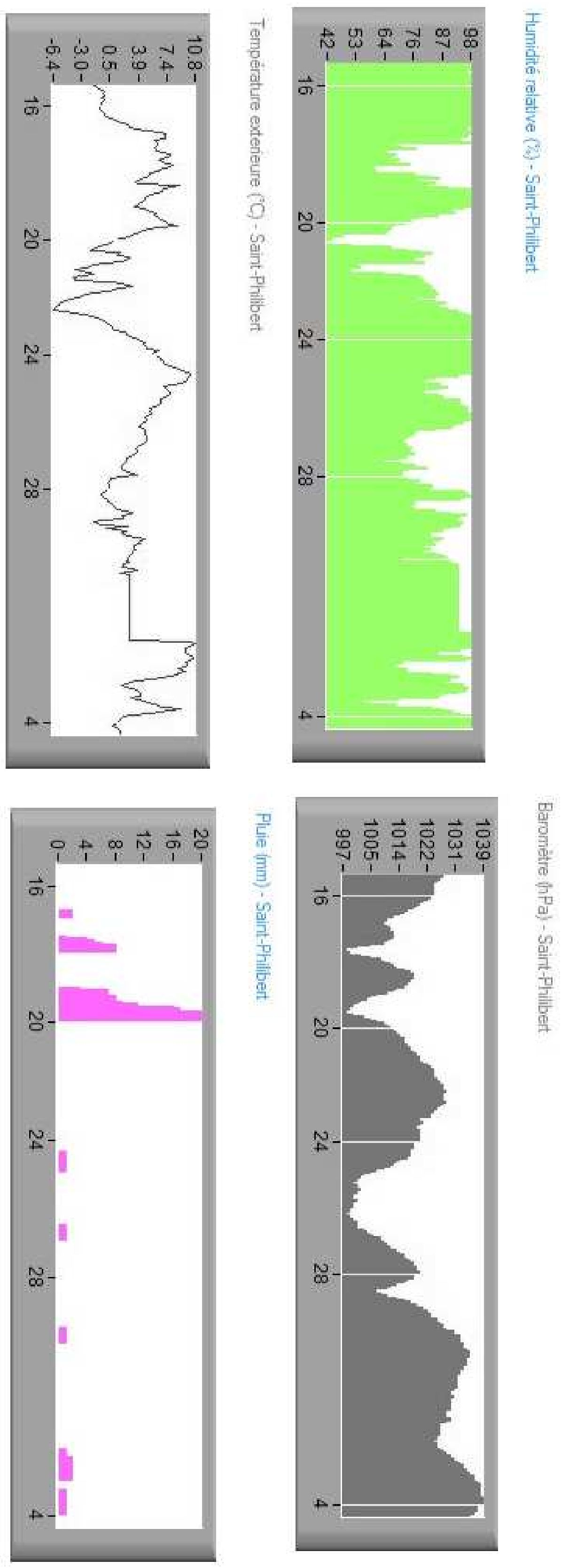


FIGURE A.9 – Données météorologiques à St-Philibert (abscisses en jours)

3$^{\text{ème}}$ année
Janvier 2007
Option mécatronique

Examen de Commande et Identification floue
Durée : 2h.

Tous documents autorisés.

A.11 Principe de l'examen

1. Officiellement vous travaillez seul, je vous fait confiance.
2. Compte rendu : un document Word de 1 page maximum (document fournit `CRexamen.rtf`), expliquant clairement la méthode choisie et les résultats obtenus.
3. Compte rendu envoyé à mon adresse mail : gonzalo.cabodevila@femto-st.fr avant 12h. Dans l'objet de votre mail inscrivez `Examen commande floue`.

A.12 Sujet

Le fichier `exam2007_2.mdl` contient un modèle non-linéaire asservis par un PID classique. Vous noterez qu'en fonction de la valeur finale de l'échelon d'entrée, le comportement dynamique est très différent (voir fig. A.10 et A.11).

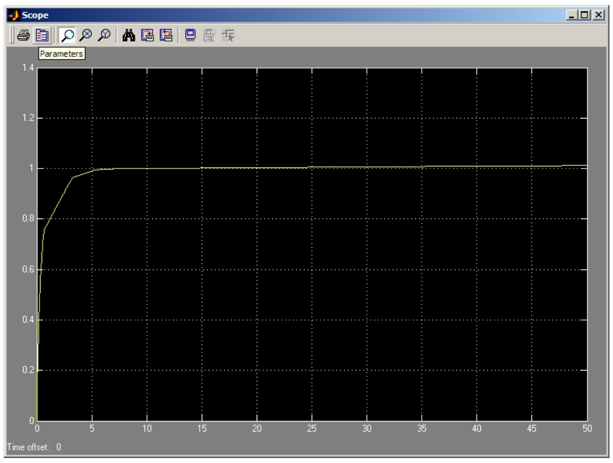


FIGURE A.10 – Réponse du système à un échelon d'amplitude 1.

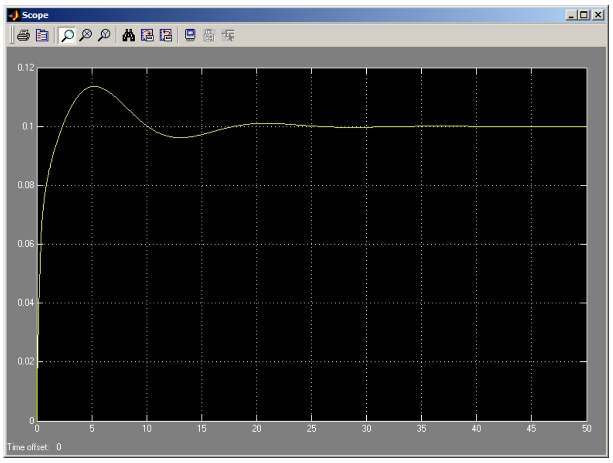


FIGURE A.11 – Réponse du système à un échelon d'amplitude 0,1.

1. Observez la réponse et la commande du correcteur flou déjà implanté. Quelle amélioration simple peut on apporter pour obtenir une erreur statique nettement plus faible.
2. Déterminer un correcteur flou donnant approximativement les mêmes caractéristiques de la réponse indicielle quelle que soit l'amplitude de l'échelon d'entrée. Pour autant que faire ce peut, on essayera d'obtenir un dépassement de 5%.

Notes :
– Aucune méthode n'est réellement meilleure que les autres pour ce sujet.
– Vous avez le choix complet du correcteur :
  – Type de régulateur : Sugeno ou Mandani

- Entrées : nombre, fonction d'appartenance...
- Règles : nombre, formes...
- Principe de défuzzification....

A.13 Encore quelques indications

- Utilisez Matlab 7
- Pour un échelon de 0,1, la simulation semble ne pas fonctionner ! Elle fonctionne, observez dans la fenêtre `rule viewver` la commande appliquée.
- Dans le compte rendu, veuillez copié-coller la fenêtre `rule viewver`
  (fenêtre `rule viewver` active, appuyez simultanément sur "Alt" et "Impr écran", puis fenêtre `Word` active tapez "Ctrl" et "v")
- Pensez à exporter votre correcteur flou vers le "workspace", sinon il n'est pas pris en compte !