ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

DESARROLLO DE UN MÓDULO PARA LA ENSEÑANZA DE CONTROL INTELIGENTE

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y CONTROL

ROMERO SUÁREZ ANA MARÍA

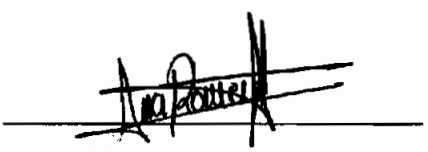
DIRECTOR: PROF. MSc. PATRICIO BURBANO R.

Quito, marzo 2002

DECLARACIÓN

Yo Ana María Romero Suárez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

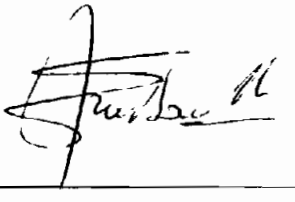
A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.



Ana María Romero Suárez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Ana María Romero Suárez, bajo mi supervisión.



Prof. MSc. Patricio Burbano
DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Al Profesor y amigo MSc. Patricio Burbano R. por su tiempo, esfuerzo y apoyo en la dirección de este trabajo. Sin su empuje, esta empresa no habría sido posible.

A mis amigos, cuya compañía logró hacer más llevaderos los problemas estudiantiles, por su apoyo diario.

A todas las personas que de una u otra forma han hecho posible la culminación de este trabajo.

DEDICATORIA

A Jehová Dios, por su bondad infinita, porque mediante Él todas las cosas son posibles.

A mis Padres, por su amor, dedicación, y apoyo brindados durante cada uno de mis años de estudio.

A mis Hermanos, parte fundamental de mi vida.

A Andrés, porque desde el inicio de la carrera ha permanecido a mi lado siendo amigo y compañero.

CONTENIDO

RESUMEN

PRESENTACIÓN

CAPITULO I: **ENFOQUE PROBLÉMICO PARA LA ENSEÑANZA DEL CONTROL INTELIGENTE**

1.1 ENSEÑANZA - APRENDIZAJE BASADAS EN UNA PEDAGOGÍA ACTIVA 1
1.1.1 EL CONSTRUCTIVISMO 2
1.1.2 ENSEÑANZA MODULAR 2
1.1.3 ENSEÑANZA BASADA EN TALLERES 3
1.1.4 SITUACIÓN PROBLÉMICA 4
1.1.5 PROBLEMA DOCENTE 4
1.1.6 PROBLEMA ALUMNOS 5
1.1.7 EL ELEMENTO DIDÁCTICO 6
1.2 NUEVO PARADIGMA EN LA SOLUCIÓN DE PROBLEMAS DE INGENIERÍA 6
1.3 ENFOQUE PROBLÉMICO (ABP) 7
1.3.1 CREACIÓN DE UNA SITUACIÓN PROBLÉMICA 8
1.3.2 PROCEDIMIENTO DOCENTE EDUCATIVO 9
1.1.2.1 Problemas Académicos 10
1.1.2.2 Problemas Profesionales 11
1.4 ESTRUCTURA MODULAR 11

CAPITULO II: **DESARROLLO DEL MÓDULO PARA CONTROL INTELIGENTE**

2.1 ESTRUCTURA DEL MÓDULO DIDÁCTICO 14
2.2 MÓDULO 16

NEURONAS ARTIFICIALES 15
REDES NEURONALES ARTIFICIALES Y SU ENTRENAMIENTO 33
APLICACIONES DE LAS REDES NEURONALES 56
CONJUNTOS DIFUSOS 85
LÓGICA DIFUSA 114
SISTEMAS DE LÓGICA DIFUSA 132
MANUAL DE USUARIO DEL TOOLBOX DE SISTEMAS LÓGICOS DIFUSOS DE MATLAB 159

CAPITULO III: **DESARROLLO DEL SOFTWARE DE SOPORTE PARA EL MÓDULO**

3.1 REQUERIMIENTOS PARA EL SOFTWARE 173
3.2 ESTRUCTURA DEL SOFTWARE 174
3.2.1 NEURONAS ARTIFICIALES 175
3.2.2 REDES NEURONALES Y SU ENTRENAMIENTO 176
3.2.2.1 Estructura 177
3.2.2.2 Entrenamiento 184
3.2.2.3 Simulación 186
3.2.3 APLICACIÓN 191

CAPITULO IV: **RESULTADOS**

4.1 LA ENSEÑANZA PROBLÉMICA Y EL CONTROL INTELIGENTE 200
4.2 RESULTADO DEL DISEÑO DEL SISTEMA LÓGICO DIFUSO: EL PÉNDULO INVERTIDO 201
RESULTADO DEL ENTRENAMIENTO DE UNA RED RECURRENTE (DISEÑO Y ENTRENAMIENTO DE UNA RED QUE DETECTA LA AMPLITUD DE SEÑALES SENOIDALES) 206

CAPITULO V: **CONCLUSIONES**

CONCLUSIONES 213

RESUMEN

Básicamente, la propuesta es ofrecer al alumno una herramienta de aprendizaje efectiva, con la que será capaz de auto instruirse o constituirse en una complemento para que el maestro preste la guía y el soporte si es necesario.

El capítulo I trae una breve introducción a la teoría del nuevo paradigma en el proceso enseñanza - aprendizaje basado en una pedagogía activa y la aplicación a este proceso de un enfoque problémico (ABP), el cual ha demostrado ser muy eficaz en cuanto a alcanzar objetivos académicos. Además, se trata sobre los beneficios que trae consigo la enseñanza por medio de módulos, los mismos que pueden ser utilizados en las modalidades semi presencial o a distancia, además de las varias técnicas pedagógicas utilizadas en la nueva escuela.

En el capítulo II se desarrolla el módulo propiamente dicho. Por tratarse de un módulo de enseñanza de Control Inteligente, se contemplará dos áreas diferentes, las redes neuronales y el control difuso.

En cuanto a las redes neuronales se estudiará desde una neurona artificial hasta la configuración de una red de neuronas artificiales, pasando por su entrenamiento y por supuesto, las aplicaciones prácticas donde el alumno visualizará la utilidad y los alcances de la nueva técnica de control aprendida.

De igual manera, en sistemas difusos, se introducirá al alumno en los temas teóricos básicos tales como conjuntos difusos y lógica difusa, para encaminarlo al final hacia las aplicaciones de los sistemas de lógica difusa en el área del Control Inteligente, donde verificará igualmente la potencialidad de esta técnica.

En el capítulo III, se implementará el software de soporte SINTEL, desarrollado en Matlab, el mismo que servirá al alumno para complementar la parte teórica con la práctica, pues esta herramienta permite simular mediante el software los sistemas de control basados tanto en redes neuronales como en lógica difusa.

PRESENTACIÓN

Por largo tiempo el diseño de los sistemas de control fue realizado dentro de un marco estrictamente matemático, la planta era modelada con ecuaciones diferenciales o de diferencia, por esta razón, el control se veía restringido a sistemas simples y sencillos; pero existen muchos sistemas que no pueden ser modelados de la manera tradicional, por tanto es necesario el uso de nuevas técnicas o metodologías.

El presente estudio pretende iniciar al alumno en el estudio de estas técnicas, fundamentales para el diseño y control de sistemas más variados y complejos: el control inteligente.

El control inteligente no se reduce únicamente a los controladores difusos y redes neurales, de hecho, de acuerdo a varias definiciones, no todos los controladores difusos o neurales son inteligentes.

Se puede decir entonces que el control inteligente se realiza mediante ciertos métodos que han sido desarrollados para describir y controlar un sistema de manera mas bien cualitativa.

El objetivo general de este trabajo es ofrecer a los estudiantes una herramienta de eficaz aprendizaje, pues al utilizar como base pedagógica el ABP (Aprendizaje Basado en Problemas), los conocimientos científicos adquiridos no serán perecederos, y lo que es mejor, desarrollarán hábitos de estudio e independencia cognoscitiva.

Cuando se trata de la enseñanza a distancia, la información que se da a los alumnos es básica, pero se los provee de elementos que permitirán ahondar en el tema, además de actividades con las que ejercitarán los nuevos conocimientos; esta es la orientación que se ha dado a este módulo didáctico, realizando para cada tema un resumen muy conciso, que contendrá la

información que el alumno requiere para iniciarse en el estudio de redes neurales y lógica difusa, todo esto mediante un enfoque problémico.

Hoy en día, la computación es pieza fundamental en el desarrollo de los controladores inteligentes, por esto, como elemento didáctico se ha desarrollado una herramienta informática implementada en Matlab, un paquete computacional muy versátil, que el alumno podrá utilizar a manera de "toolbox" entrenándose así para la modelación de su propio controlador inteligente.

CAPÍTULO I

ENFOQUE PROBLÉMICO PARA LA ENSEÑANZA DE CONTROL INTELIGENTE

1.1 ENSEÑANZA - APRENDIZAJE BASADOS EN UNA PEDAGOGÍA ACTIVA

El proceso enseñanza - aprendizaje actual, conocido como Escuela Tradicional, tiene sus orígenes a finales del siglo XVII cuando caía el régimen feudal y se daba inicio a otra forma de opresión: la burguesía. Para la sociedad de esta época era propicio crear un modelo de enseñanza en el cual se imponía la obediencia y el orden.

Con esta perspectiva, se observa que este modelo, no permite al alumno el derecho a razonar, a buscar soluciones propias a sus necesidades creando en él una conducta de inercia frente al proceso, acrítico e irreflexivo, pendiente únicamente de lo que el maestro diga y obligado a memorizar y repetir fielmente lo que se ha pronunciado en clase, como consecuencia, no se obtiene en el alumno una conciencia creadora y el conocimiento alcanzado es volátil pues no tuvo participación activa en el desarrollo de la clase.

En esto precisamente radica la diferencia entre la educación tradicional y la activa.

El proceso enseñanza - aprendizaje basado en una pedagogía activa implica permitir la naturalidad y espontaneidad del alumno en sus iniciativas dentro del proceso de aprendizaje, llevándolo a la comprensión y significación de lo aprendido, haciéndolo capaz de utilizar este conocimiento en situaciones futuras.

1.1.1 El constructivismo

El constructivismo es un modelo educativo que busca que el alumno integre los conceptos científicos a las vivencias y creencias que posee previamente generadas en lo cotidiano, es decir, es la reestructuración del esquema conceptual previo en otro distinto, para construir la realidad científica. Para esto, el maestro deberá crear en el alumno un conflicto cognitivo que produzca la duda sobre la validez y la universalidad de sus conocimientos previos permitiéndole lograr comparaciones y generalizaciones.

Los conceptos empíricos de los cuales es poseedor el alumno se deben considerar como una aproximación sucesiva a la conceptualización científica, pero tal aproximación a los conocimientos no es lineal como lo era tradicionalmente, sino que es una progresión que avanza dependiente de los acontecimientos cotidianos significativos y otros factores tales como: pensamiento del alumno (sus esquemas preexistentes, motivaciones e intereses), pensamiento del maestro (estrategias didácticas) y el medio en el que se desenvuelven.

Se puede resumir exponiendo lo siguiente:

- "El hombre no se forma desde afuera en base a estímulos externos, su personalidad se forma en base a una relación activa con su ambiente".
- "Todo conocimiento es construido por el individuo cuando interacciona con el medio y trata de comprenderlo."
- "Construir significados requiere de esfuerzos por parte del que aprende para generar relaciones entre los estímulos y la información acumulada".

1.1.2 Enseñanza modular

Según Walter Dick y Goldschmid el módulo es "una unidad didáctica de carácter autosuficiente e independiente diseñado para ayudar al alumno en el proceso de aprendizaje" [1].

El objetivo es lograr que el alumno aprenda avanzando a su ritmo, según sus propias necesidades e inquietudes; desarrollando en él las capacidades de autoaprendizaje y autoevaluación.

Un módulo puede ser autosuficiente, es decir, llevar por sí solo de la mano al alumno en la etapa del aprendizaje, de manera que no es necesaria la ayuda directa del profesor, o semi - suficiente cuando necesite de éste para lograr los objetivos inicialmente propuestos.

El módulo educativo, creado para ayudar al alumno en la educación a distancia ha demostrado ser un excelente elemento didáctico aún en las clases presenciales.

Puesto que el alumno estudia, investiga y realiza talleres, el módulo se convierte en parte importante del proceso activo de enseñanza - aprendizaje.

1.1.3 Enseñanza basada en talleres

El taller es una técnica de aprendizaje con base en el trabajo grupal, pues reúne de 6 a 8 personas para conseguir un fin común, producir conocimiento persiguiendo el objetivo planteado por el docente.

Cirigliano y Villaverde definen las técnicas grupales como "maneras, procedimientos o medios sistematizados de organizar y desarrollar la actividad del grupo, sobre la base de conocimientos suministrados por la teoría de la dinámica de grupos" [2].

El trabajo grupal trae consigo beneficios individuales:

- Crea en el alumno la capacidad del análisis crítico, además de fomentar el respeto al criterio de los demás.
- La heterogeneidad del grupo hace al trabajo más enriquecedor, pues cada individuo aportará con información dependiente de sus vivencias, experiencias

y conocimientos, lo que obliga al grupo a llegar a un consenso en las ideas, encontrando puntos de convergencia.

- Origina en el individuo una disposición para colaborar haciéndolo socialmente activo.
- Al sentirse parte de un conglomerado, busca soluciones para un problema común, es sensible a los problemas sociales.

1.1.4 Situación problémica

La situación problémica implica crear en el alumno una actitud de interés por el conocimiento, de apertura cognoscitiva. Con esta herramienta, el estudiante será capaz de resolver problemas no estereotipados, pues en lugar de repasar casos simples y comunes se creará en él la capacidad de analizar la situación para conseguir la respuesta que mejor se adapte a sus necesidades.

A diferencia de la enseñanza tradicional en la que la transferencia de conocimientos se realiza en forma unilateral con actividad única del docente y ésta es captada por el alumno casi en forma inconsciente, la enseñanza problémica propondrá situaciones para los cuales el alumno deba utilizar tanto conocimientos pasados como nuevos llevándolo a inmiscuirse en una investigación del tema y conseguir así su propia respuesta.

Este tipo de enseñanza tiene la ventaja de que el conocimiento adquirido no es transitorio, sino que perdura en el alumno pues fue producto del razonamiento consciente y de la investigación. Además, el objetivo educativo logrado es muy superior al del método tradicional pues en el alumno no solo se implantó el conocimiento científico sino que se creó en él una capacidad creadora e investigativa.

1.1.5 Problema docente

El problema docente constituye la búsqueda del maestro por crear un ambiente problémico y elementos didácticos que contribuyan al desarrollo del mismo. Sin

embargo muchas veces éstos surgirán por sí solos, entonces el maestro debe ser capaz de utilizarlos adecuadamente para lograr en el alumno un grado alto de activación mental e interés cognoscitivo y explotar todo el potencial que posee la enseñanza problémica; de modo que incluso el docente se enfrenta a una situación problémica de carácter pedagógico: ¿cómo aprovechar las preguntas, tareas y situaciones dándoles un enfoque problémico?.

La pregunta ha sido desde siempre la herramienta pedagógica óptima para llegar al alumno, pero no siempre logra producir en él la activación sistemática de la actividad cognoscitiva que es la razón de ser de la enseñanza problémica. El maestro entonces deberá tener la capacidad de realizar preguntas que encierren cierto grado de dificultad intelectual, es decir que para responderlas, no serán suficientes los conocimientos precedentes ni los aportados por el profesor, originando en el alumno la necesidad de investigar, con esto se ha logrado prepararlos para la búsqueda independiente de conocimientos.

La tarea de igual manera, incentivará el aprendizaje independiente en los alumnos cuando se la envía tomando en consideración lo siguiente:

- ¿Qué instrucciones les va a proporcionar?
- ¿Qué información previa es necesaria para que exista un aprendizaje significativo?
- ¿Que pistas de base se les proporcionará tales como bibliografía, fuentes, etc.?

1.1.6 Problema alumnos

Cada persona es poseedora de experiencias y por ende conocimientos propios de la edad con la que cuentan. El alumno será capaz por tanto de aprender y resolver situaciones por sí solo, se dice entonces que el conocimiento está dentro de una *"Zona de Desarrollo Efectivo"*, pero existirán otros a los que no podrá acceder si no es con la ayuda de un tutor, un texto o un audiovisual, a esta zona de conocimientos se la denomina *"Zona de Desarrollo Potencial"*. El límite entre estas dos es llamado *"Zona de Desarrollo Próximo"*; cuando el conocimiento

desarrollado en el alumno ha sido logrado mediante un proceso activo, sólo entonces éste pasará a formar parte de su memoria comprensiva, creando mayores posibilidades de utilizar lo aprendido en futuras situaciones, es decir, ha habido un crecimiento de las zonas de desarrollo próximo.

1.1.7 El elemento didáctico

El elemento didáctico es parte importante en el proceso enseñanza - aprendizaje, pues constituye no solamente un recurso ilustrativo, sino un mecanismo que lleva al alumno hacia una auto - motivación para que trabaje, investigue, descubra y construya el conocimiento sobre la base de experiencias propias, conviertiéndolo en parte activa del proceso.

El recurso didáctico deberá cumplir los siguientes requisitos:

- Ser adecuado al tema de clase
- Ser de fácil comprensión y manejo
- Estar en perfectas condiciones de funcionamiento

1.2 NUEVO PARADIGMA EN LA SOLUCIÓN DE PROBLEMAS DE INGENIERÍA

Kuhn define el paradigma como "logros científicos universalmente reconocidos que proporcionan, por algún tiempo, problemas y soluciones modelos a una comunidad de científicos. Son la fuente de métodos, problemas de campo y normas de solución aceptados por cualquier comunidad científica moderna en un momento dado".

Tradicionalmente, para la modelación de los problemas de ingeniería, se tenía como base el conocimiento objetivo. El punto de partida es pues, la obtención de un modelo matemático en ecuaciones diferenciales o de diferencias exacto y preciso, pero, ¿cuál es el inconveniente de este método?. Con el avance de la tecnología, los sistemas a controlar se vuelven más complejos, lo que dificulta el

análisis y peor aún la extracción del modelo a ecuaciones diferenciales, pues existen sistemas de control que no pueden ser adecuadamente descritos mediante este marco matemático, entonces el conocimiento objetivo se vuelve obsoleto.

El nuevo paradigma se basa en la incertidumbre, inexactitud y vaguedad. El control inteligente se origina entonces en la dificultad de plantear bajo un marco matemático riguroso el fenómeno a ser controlado. Se maneja técnicas de comunicación lingüística que representan la ambigüedad de las expresiones propias de los seres humanos, cuanto más inteligente se necesita sea el controlador, mayores serán las técnicas lingüísticas utilizadas.

El nuevo paradigma tiene una gran capacidad para simular el razonamiento, toma de decisiones y otros aspectos propios de los seres humanos, la capacidad de capturar y dar con el significado de oraciones expresadas en lenguaje natural.

Éste viene a ser el conocimiento subjetivo, pues es más bien un tipo de conocimiento cualitativo. Se define al subjetivismo como un "punto de vista filosófico según el cual lo decisivo para el valor del conocimiento no es el objeto, sino la naturaleza o estado del sujeto" [4].

1.3 ENFOQUE PROBLÉMICO (ABP)

El aprendizaje basado en problemas (ABP) es una nueva técnica enseñanza - aprendizaje en la que se prepara al estudiante para la solución adecuada de problemas profesionales, elevando el nivel de asimilación de conceptos, de generalización y abstracción, sin restringirse a la enseñanza de algunas operaciones mentales en orden casual, sino en base a un sistema de acciones intelectuales para solucionar tareas no estereotipadas.

El enfoque problémico proporciona al alumno herramientas para alcanzar el conocimiento, es decir, provoca en él una actitud productiva y creadora; los conocimientos asimilados de esta forma no serán perecederos pues se retiene

mejor aquello que se ha descubierto por sí solo o con esfuerzo; además, se logra una clase más activa animando a los alumnos a participar eficazmente. Adicionalmente se consigue un conocimiento útil, que se puede modificar y aplicar.

1.3.1 Creación de una situación problémica

Una situación problémica puede generarse de manera natural, es decir, inconscientemente durante el proceso enseñanza - aprendizaje, o puede ser creada por el maestro, pero sin duda, será mejor aprovechada aquella que es premeditada, pues se constituye en un sistema encaminado hacia un objetivo.

El maestro ciertamente deberá estar preparado en el área pedagógica, para explotar de igual manera tanto el surgimiento natural como la creación de situaciones problémicas, obteniendo de los alumnos una participación individual en el proceso y un alto grado de activación mental.

Una situación problémica puede generarse a partir de las siguientes condiciones del proceso:

1. Cuando al alumno se le plantea una tarea que no puede solucionar y explicar con los conocimientos que posee.

2. Se crea en la necesidad de los alumnos de aplicar en la práctica conocimientos anteriores.

3. En la contradicción, ante la imposibilidad de aplicar una solución teóricamente procedente a la práctica.

4. En la falta de conocimientos de los alumnos para argumentar teóricamente un resultado alcanzado en la práctica de la tarea didáctica.

El docente podrá pues, crear premeditadamente situaciones problémicas, formulando tareas que contribuyan con el desarrollo cognoscitivo del alumno. Como ejemplo se enumerarán las siguientes:

- Utilizar las contradicciones que surgen en la realización de observaciones de fenómenos de la naturaleza.

- Plantear tareas problémicas didácticas para explicar un fenómeno o buscar vías para su aplicación práctica.

- Estimular a los alumnos a realizar análisis de hechos y fenómenos de la vida cotidiana que ocasionen aparentes contradicciones entre la concepción popular y los conceptos científicos de estos hechos.

- Elaborar suposiciones, formular deducciones y comprobarlas en forma experimental.

- Estimular a los alumnos a comparar, confrontar y contraponer hechos, fenómenos, reglas y acciones a partir de las cuales puede surgir una situación problémica.

- Motivar al estudiante a establecer relaciones, generalizaciones y abstracciones a partir de situaciones problémicas.

1.3.2 Procedimiento docente - educativo

El correcto procedimiento del proceso ABP, deberá sin duda combinar los siguientes componentes:

1.3.2.1 Problemas Académicos

Los problemas académicos vienen directamente relacionados con lo que el estudiante debe conocer para hacer frente las interrogaciones hechas por el

maestro, es decir, es la materia básica que el alumno debe conocer para estar preparado correctamente en el tema. Desde el punto de vista del ABP, es la herramienta elemental a partir de la cual el alumno partirá para hacer suyo conocimientos más profundos.

En efecto, el objetivo final del enfoque ABP es el de capacitar al alumno para afrontar problemas profesionales en su ámbito de especialización.

Esto presupone por una lado una tarea de exploración de aquellos problemas profesionales (perfil profesional). Entonces, a partir de los problemas profesionales detectados debe replantearse el sistema académico (pénsum de estudios) para determinar los problemas académicos estratégicos que den al alumno el soporte suficiente para afrontar los problemas profesionales.

Estos problemas académicos tienen que ser adecuadamente articulados o coordinados y tienen que basarse sin duda en el entrenamiento fundamentado en situaciones problémicas.

Se deberá manejar aspectos y problemas repetitivos de temáticas fundamentales para a partir de conflictos y actividades cognoscitivas abordar aspectos y problemas no estereotipados a través de situaciones problémicas; ayudas y recursos didácticos, talleres y laboratorios manejando la zona de desarrollo efectivo.

Luego deberá reforzarse el aprendizaje y la apertura cognoscitiva para profundizar los problemas académicos buscando la parte aplicativa mediante la realización de proyectos manejando la zona de desarrollo próximo orientado a enfocar problemas profesionales.

1.3.2.2 Problemas Profesionales

Los problemas profesionales son la aplicación de los conocimientos que el estudiante ha adquirido en temas más específicos y prácticos relacionados

directamente con la especialidad, es el ámbito en el que el alumno va a ejercer su actividad pues es lo que exige el medio profesional.

En este caso está orientado hacia el Control Inteligente basado en redes neuronales y lógica difusa. Dentro de este ámbito, se ha detectado como problemas profesionales, a través de conversaciones con expertos, de la realización de proyectos de titulación y del estudio de información existente a los sistemas de lógica difusa y simulación de sistemas mediante redes neuronales.

Los sistemas de lógica difusa comprometen los siguientes temas: la fusificación y defusificación que involucran el manejo de conjuntos difusos; mecanismos de inferencia que implica el manejo de relaciones difusas y lógica difusa; y, la utilización de las bases de conocimiento que manejan expresiones y lógica difusas.

En cuanto a las redes neuronales se tiene: la configuración, interconexión y el entrenamiento de neuronas artificiales y la simulación de sistemas que implican la utilización de las redes así formadas.

1.4 ESTRUCTURA MODULAR

Partiendo de la hipótesis que da origen a este trabajo: "La utilización de un módulo didáctico ABP (Aprendizaje Basado en Problemas) mejora el proceso enseñanza - aprendizaje del control inteligente", se ha orientado cada tema hacia los principios fundamentales del ABP, estos son los problemas profesionales y los académicos, además hacia la consecución de los objetivos propuestos inicialmente; y se ha dividido al módulo de la forma que se aprecia en el esquema de la figura 1.1

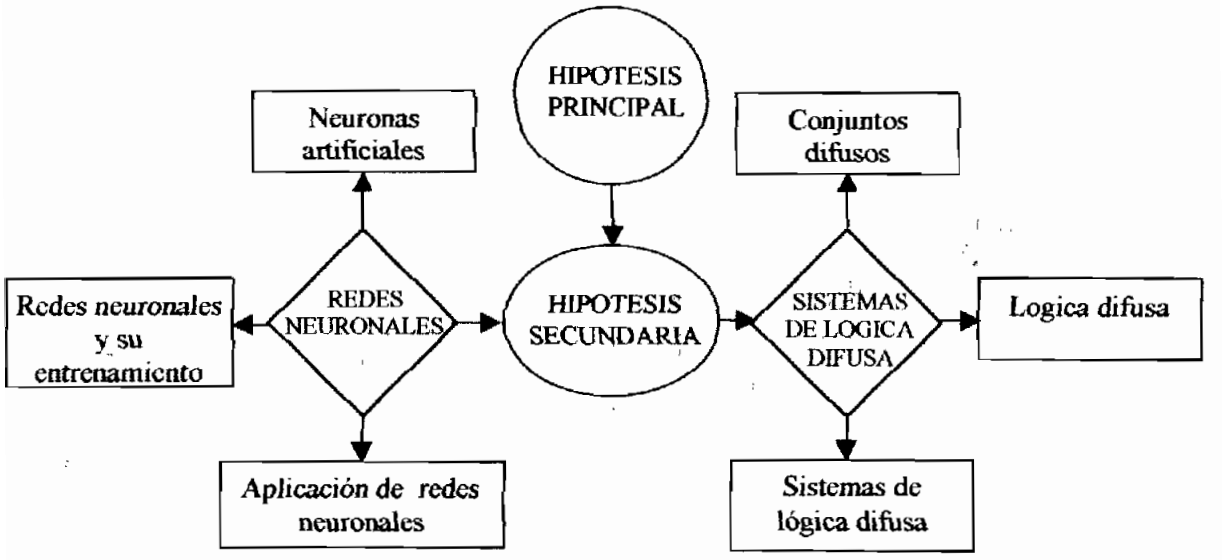


Fig. 1.1 Estructura modular para Control Inteligente

Como hipótesis de segundo grado se maneja, "La utilización del control inteligente puede ser adecuadamente visualizada mediante el paquete computacional MatLab".para poder apreciar las relaciones causa y efecto de los fundamentos teóricos y aplicaciones, cuya práctica se realiza a través de la implementación de una unidad de software.

Como categorías se tienen las redes neuronales y los sistemas de lógica difusa, éstas se subdividen en las unidades correspondientes para abordar los problemas académicos y profesionales en las cuales se manejarán las respectivas variables e indicadores.

El módulo entonces, comprende 7 unidades básicas, cada una contiene temas concernientes tanto a problemas académicos como a profesionales.

Cada unidad está estructurada de la siguiente forma:

- Objetivo

  Señala el propósito específico, la meta a la cual el alumno debe llegar. Puesto que el sistema es de auto aprendizaje, se deberá poner especial atención en verificar que ha alcanzado dicho objetivo.

- Marco teórico

  Contiene la información científica adecuada para estimular y facilitar el auto aprendizaje en el estudiante. Contará con conceptos, ejemplos y preguntas adecuados a cada tema, promoviendo siempre el proceso enseñanza - aprendizaje activo.

- Talleres

  Encaminados al ejercicio en los temas relacionados con los problemas académicos, es el elemento instruccional que propone el trabajo grupal para dar solución a cierto planteamiento.

- Laboratorio

  Comprende la utilización programada del software de soporte didáctico.

- Proyectos

  Es la teoría aplicada a problemas propios de control con el enfoque profesional, comprende la realización de tareas más elaboradas para que el alumno adapte los conocimientos teóricos con los prácticos.

Referencias Bibliográficas

[1] **'Impacto'**, Revista de la Unidad de Educación a Distancia, UNED, del Centro de Estudios de Postgrado, CEPOSTG, de la Universidad de Loja.

[2] Ruiz Ma. Luisa, Eldredge Graciela, Cárdenas Susana, **'Módulo autoinstruccional del proceso de enseñanza - aprendizaje'**, DINACAPED

[3] Brito Delfín M., **'Cómo desarrollar las asignaturas técnicas con un enfoque problémico'**, Editorial Pueblo y Educación

[4] Herrara Hermógenes, **'Diccionario de Filosofía'**, Colección Tesi

CAPÍTULO II

DESARROLLO DEL MÓDULO PARA CONTROL INTELIGENTE

2.1. ESTRUCTURA DEL MÓDULO DIDÁCTICO

Con base en lo expuesto en el primer capítulo, el enfoque problémico se deriva en dos principales puntos de partida para el correcto proceso enseñanza - aprendizaje, el problema académico y el problema profesional.

Los módulos son preparados entonces, tomando en cuenta esta estructura diferenciada como método de enseñanza; se tiene pues correctamente enmarcados los temas que atañen a cada problema de forma que, para el estudiante el proceso se desarrolle de manera natural y consecutiva.

Tomando en cuenta lo expuesto, se han dividido el módulo en seis unidades:

REDES NEURONALES

1. Neuronas artificiales
2. Redes Neuronales y entrenamiento de redes
3. Aplicaciones

SISTEMAS DIFUSOS

4. Conjuntos difusos
5. Lógica difusa
6. Sistemas de lógica difusa
7. Manual de usuario del Toolbox de Lógica Difusa de Matlab

En donde se reconocen como problemas académicos y profesionales lo esquematizado en la figura 2.1:

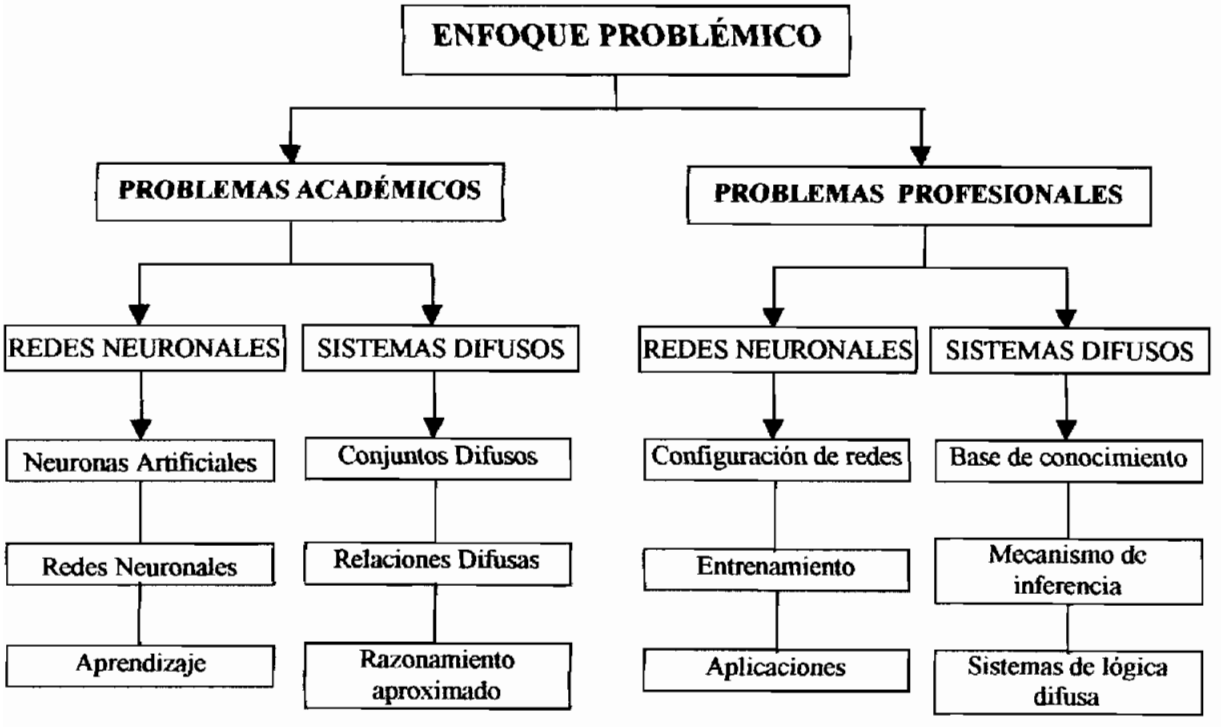


Fig. 2.1 Problemas profesionales y académicos

El formato de texto del resto del capítulo se lo hace siguiendo la estructura modular para posteriormente extraerlo del proyecto de titulación como un texto de ayuda didáctica.

2.2. MÓDULO

NEURONAS ARTIFICIALES

1. OBJETIVOS

- Conocer las características de una neurona artificial.
- Reconocer los tipos de neuronas artificiales, sus ventajas e inconvenientes.

2. MARCO TEÓRICO

2.1. INTRODUCCIÓN

El ser humano siempre se ha maravillado respecto a las grandes potencialidades con que cuenta. La capacidad de discernir es propia de las personas, es lo que nos distingue de todo lo demás. A lo largo de los años, los científicos han intentado descifrar el funcionamiento enigmático de nuestro propio cerebro para obtener soluciones tanto en el área médica como en psicología.

¿Por qué somos capaces de, por ejemplo, percibir imágenes confusas y las computadoras no?. Esto se debe principalmente a que la arquitectura del cerebro humano es distinta a la de una computador convencional, el paralelismo masivo y la interconectividad que se observa en los sistemas biológicos son los responsables de nuestra alta capacidad de procesamiento.

Una mejor comprensión de la neurona biológica y de sus interconexiones, permitió a los investigadores plantear modelos matemáticos para realizar las primeras pruebas y probar sus teorías.

Partiendo del sueño de conseguir que las máquinas sean capaces de simular el comportamiento humano, los investigadores logran los primeros trabajos de acercamiento en las décadas de los 50's y 60's con circuitos electrónicos

emuladores de redes neuronales. Más tarde sería implementado en computadores como un medio más flexible de simulación. Los resultados iniciales fueron impresionantes, se logró su aplicación en análisis de electrocardiograma, predicción del clima y visión artificial, entre otros; todo esto con redes de una sola capa de neuronas que sería conocido como *Perceptrón*. El porvenir tecnológico parecía promisorio. Pero, a pesar de que la red de capa simple había podido resolver ciertos problemas, había fallado con muchos otros, lo que llevó al matemático Marvin Minsky a realizar profundos estudios y determinar las limitaciones del Perceptrón; todo esto fue publicado en su libro "Perceptrons" (Minsky y Papert, 1969); esto logró detener por lo menos durante dos décadas el avance masivo de las investigaciones de las redes neuronales, sin embargo, no todos pararon, por lo que resurgió nuevamente con fuerza con innovadores resultados.

2.1.1 Características

Muchas de las características con las que cuentan las redes neuronales han sido propias de los seres humanos, esto hace precisamente de las redes unos de los elementos importantes en la computación en nuestros días.

- ***Aprendizaje:*** Son capaces de aprender sobre la base de experiencias, ésta es una de las principales características, pues es lo que se había buscado al tratar de crear computadores inteligentes. Se tiene el concepto aprendizaje mediante ejemplos, cuya ventaja al momento de resolver problemas radica en que no es necesario un mecanismo estricto para transformar algorítmicamente las entradas en salidas deseadas, mas bien, lo que se necesita para la mayoría de las redes es una colección de ejemplos representativos de la aplicación deseada. Esto se logra con un entrenamiento basado en algoritmos computacionales.

- ***Generalización:*** Luego del entrenamiento aplicado a la red, ésta será capaz de resolver el problema con diferentes entradas y salidas sin necesidad de

repetir el entrenamiento cada vez que se realice un cambio al problema inicial. La red es capaz de hallar sus propias soluciones para problemas concretos, dándosele únicamente ejemplos del comportamiento deseado.

- ***Abstracción:*** Existen redes neuronales que adecuadamente entrenadas pueden, crear un símbolo ideal a partir de entradas incompletas, depurar una imagen y completar la idea. La red responderá con alguna salida incluso en el caso en que se le presenten entradas que nunca haya visto, tales como tramas que contengan ruido. Tal es así que, a partir de una imagen distorsionada, por ejemplo una letra, puede dar una respuesta real correcta (una letra perfecta) de algo que nunca conoció.

- ***Aplicabilidad:*** Si bien la red neuronal es un instrumento de procesamiento inteligente, hay sin duda aplicaciones en los que su uso resultaría improcedente. Sin embargo, cada vez su aplicabilidad es más amplia, el rango de aplicaciones que tienen estas redes sin duda superarán las expectativas, pues es la técnica preferida en la actualidad.

2.1.2 Aplicaciones

Cuando se demostraron con éxito las primeras aplicaciones de las redes neuronales, se pensaba que éstas eran una panacea, pues era la consecución del proyecto más ambicioso dentro de la comunidad científica, hacer de las máquinas entes inteligentes.

Ahora se conoce que, si bien no son el remedio para todos los males, el campo de aplicación es bastante extenso y con seguridad se incrementará puesto que las investigaciones continúan exitosamente.

A continuación, se lista algunas de las aplicaciones importantes:

Manufacturación.- Control de procesos de fabricación, diseño y análisis de productos, proceso y diagnóstico de la maquinaria, identificación de partículas en tiempo real, sistemas de inspección de calidad visual, comprobación de cerveza, análisis de calidad de sueldas, predicción de calidad del papel, análisis de calidad de partes de computadoras, análisis de productos químicos, análisis de mantenimiento de maquinaria, modelación dinámica de procesos químicos.

Electrónica.- Predicción de códigos secuenciales, distribución de chips en circuitos integrados, control de procesos, análisis de falla de chips, máquina visual, sintetizador de voz, modelación no lineal.

Medicina.- Diseño de prótesis, analizador de células cancerosa en mamas, optimización del tiempo en transplantes, reducción de extensión hospitalaria, mejora la calidad de los hospitales.

Aviación.- Simuladores de vuelo, sistemas de control de aviación, perfeccionamiento del piloto automático, simulación de componentes de aviones y detección de fallas en los mismos.

Financiero.- Estimación del estado real, asesoramiento de préstamos, valuación de bonos corporativos, análisis del uso de líneas de crédito, análisis financiero corporativo.

Lenguaje.- Reconocimiento y compresión de palabras, clasificación de vocales, sintetiza texto a palabras.

Robótica.- Control de trayectorias, controles de manipulación, sistemas de visión.

Telecomunicaciones.- Compresión de datos e imágenes, traducción del lenguaje hablado en tiempo real, tratamiento de sistemas de pago al cliente.

Se aplica en muchas otras áreas tales como: seguridad, transporte, entretenimiento, petróleos, seguros, automovilismo, banca, etc.

2.2 DE LAS NEURONAS BIOLÓGICAS A LAS ARTIFICIALES

Las neuronas son células básicas del sistema nervioso. Poseen un núcleo y varias prolongaciones citoplasmáticas y funcionan de acuerdo a procesos electroquímicos [1].

En una neurona biológica se distinguen tres partes principales (figura 2.2): el cuerpo de la célula, las dendritas y el axón, cada una de las cuales cumple funciones específicas.

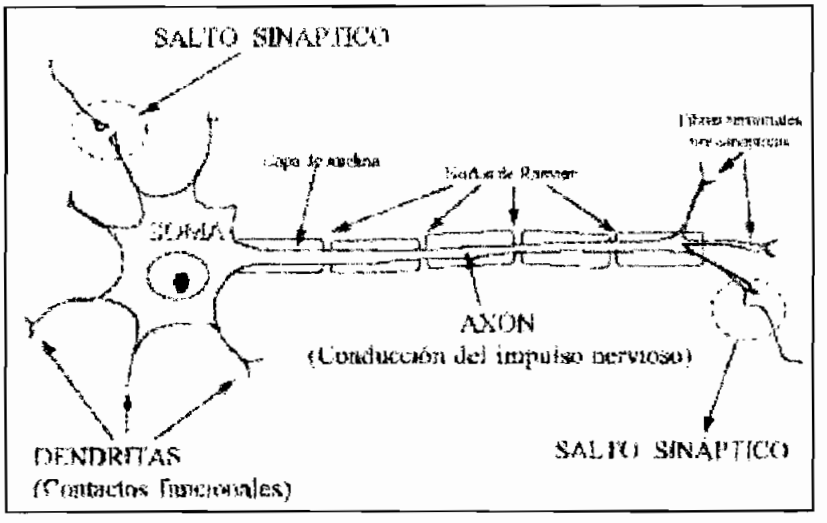
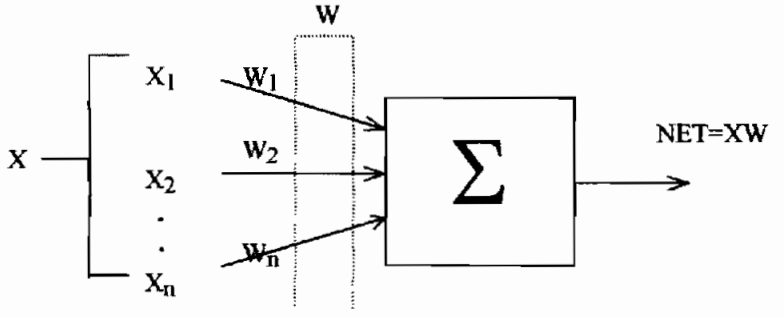


Fig. 2.2 Neurona Biológica

Simplificando el complejo funcionamiento de las neuronas se resume lo siguiente: Desde muchas otras neuronas llegan señales que son captadas por las *dendritas* a través de la unión sináptica, de allí pasan al cuerpo de la célula en donde son promediadas mediante complicados procesos químicos, si el promedio resultante logra sobrepasar dentro de cierto intervalo de tiempo un valor conocido como *umbral*, la neurona se dispara emitiendo una señal que viaja a través del *axón* hacia otras neuronas; caso contrario, no se producirá ningún tipo de estímulo.

Analógicamente, la neurona artificial es básicamente un elemento de todo o nada, es decir, se puede o no tener una respuesta o estímulo. Funciona de la siguiente forma: llega a la neurona artificial un conjunto de entradas (salidas de otras

neuronas) que se multiplican con pesos (análogo a las conexiones sinápticas), el resultado de sumar estos productos causará que la neurona se dispara o se inhiba. La configuración básica se aprecia en la figura 2.3:

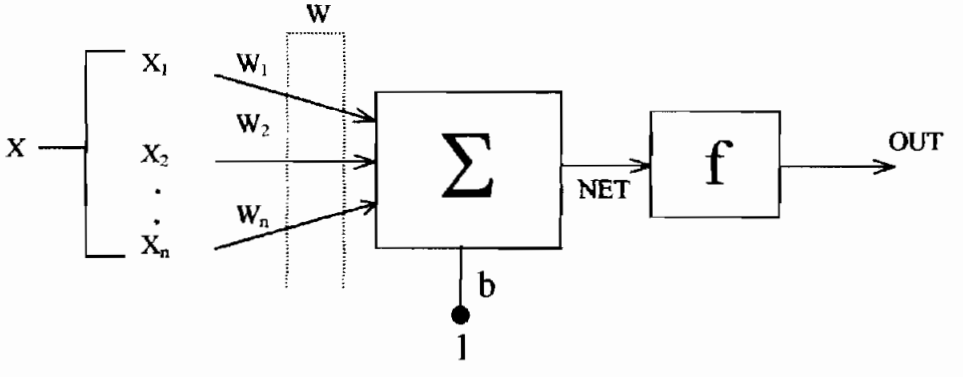


$$NET = x_1.w_1 + x_2.w_2 + ... + x_n.w_n = \sum_{i=1}^{n} x_i.w_i$$

Fig. 2.3 Neurona Artificial

Donde X y W son vectores.

Pero en general una neurona puede presentar tres partes básicas:



$$OUT = f(X.W + b)$$

Fig. 2.4 Modelo de neurona con función de activación y tendencia

1. Las conexiones o pesos en donde son ponderadas las entradas. El peso es parte básica de una neurona artificial pues, sirve para determinar si esa entrada influirá de manera significativa en el resultado final.

2. El sumador, que como su nombre lo indica, suma las entradas ponderadas y cuya respuesta es un valor de comparación.
3. Una función de activación que limita la salida de la neurona a un valor normalizado en un rango de [0,1] o de [-1,1].
4. Usualmente se añade valores escalares a las neuronas llamadas *bias (b)* o *tendencia*, y funcionan como un offset , es fácilmente anexionado pues es un peso entrenable que siempre tiene una valor de entrada igual a la unidad (1).

Las neuronas son elementos capaces de aprender de experiencias o ejemplos. Esto se hace mediante un entrenamiento que consiste básicamente en hacer variar los pesos de tal forma que la respuesta final se acerque a la real o deseada (obtener la relación entrada - salida deseada).

2.3 PERCEPTRÓN

Los primeros trabajos importantes en la modelación de redes neuronales artificiales fueron alcanzados en los años 40, cuando se concibió un primer modelo que causaría revuelo y mucho optimismo en las décadas subsiguientes. Consiste en un conjunto de entradas que son multiplicadas con sus respectivos pesos, y cuyos resultados son sumados para obtener un valor, el mismo que es comparado con otro llamado umbral que no es más que una función paso o limitador duro; si el resultado es mayor que el umbral, la salida es uno (1), caso contrario es cero (0).

Este modelo de neurona simple sería conocido como *PERCEPTRÓN* y su estructura se observa en la gráfica 2.7

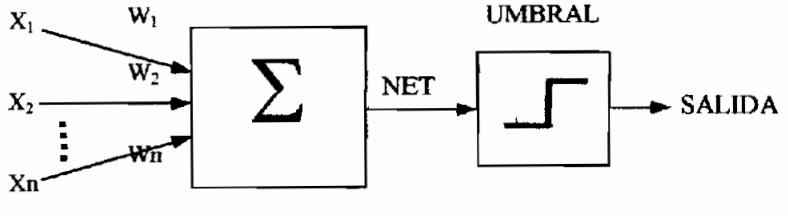


Fig. 2.5 Neurona Perceptrón

El perceptrón era un dispositivo de aprendizaje, es decir, que en su configuración inicial no era capaz de, por ejemplo distinguir tramas interesantes, sin embargo era capaz de adquirir esta capacidad a través de un proceso de aprendizaje. Éste consistía en incrementar o decrementar los pesos de las entradas dependiendo si éstos contribuían o no a alcanzar la respuesta correcta.

Posteriormente, se formularon estudios importantes entre ellos el de Rosenblatt (1962) quien propuso un importante teorema relacionado con el aprendizaje del perceptrón.

Sin embargo del trabajo positivo que pudo realizar una neurona perceptrón, tal como el reconocimiento de funciones, su mejor trabajo se logra cuando trabaja como red perceptrón.

Para muchos el perceptrón se había convertido en una panacea, pero pronto esta idealización sería contrarrestada debido a un estudio serio realizado y publicado por Minsky y Papert en 1969, donde se demostraban las grandes limitaciones del perceptrón; este hecho tendió un velo sobre las investigaciones y a no ser por unos pocos que continuaron con su labor, habría sido el fin de las redes neuronales.

El perceptrón fue concebido principalmente como un ***separador lineal***, el éxito inicial lo obtuvo en el reconocimiento de tramas linealmente separables.

Se formulará un ejemplo simple como un primer acercamiento al tratamiento de esta teoría:

Considere el comportamiento de la compuerta lógica AND (tabla. 2.1). Para cada par de entradas se tiene una única salida. Una separación lineal propone dividir el plano representado en la figura 2.6 por una línea recta de tal forma que, queden agrupados a cada lado de los semiplanos así formados, los pares de entradas cuyas respuestas o salidas sean las mismas.

La línea que separa el plano viene dada por la ecuación que la representa una recta:

$$NET = x_1.w_1 + x_2.w_2$$

en donde:

NET valor a comparar con el umbral

w_1 y w_2 pesos

x_1 y x_2 variables de entrada

x_1	x_2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.1 Compuerta lógica AND

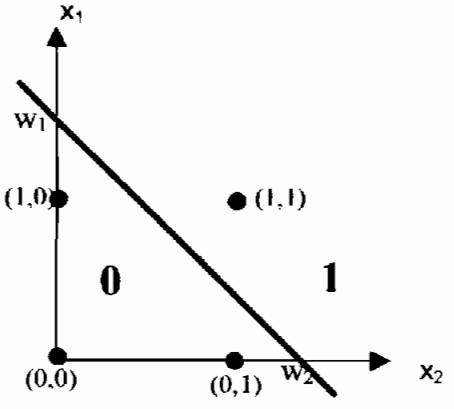


Fig. 2.6 Separabilidad lineal

Este tipo de tareas, y por supuesto aún más complejas, fueron realizadas con éxito por el perceptrón, sin embargo falló en otras, una de ellas tan sencilla, la función lógica XOR.

2.4 SEPARABILIDAD LINEAL

La separabilidad lineal, propia de la neurona perceptrón, está limitado a los problemas en los cuales se tiene un conjunto de puntos (valores de entradas) que son geométricamente separables [2]. En otras palabras, si una línea recta o un plano pueden ser dibujados para separar a los vectores de entrada en sus categorías correctas, los vectores son linealmente separables.

El espacio dimensional en donde se realiza la separación depende del número de entradas, así, para dos entradas como en el ejemplo anterior se tiene un plano bidimensional, para tres entradas, tridimensional y, en general para *n*

entradas se tendrá un espacio *n* - dimensional. La separación la hará un *"hiperplano"* de dimensiones n-1, un objeto geométrico que divide el espacio de dimensión *n*. Sin embargo, es muy difícil que en la práctica un problema con un gran número de entradas sea linealmente separable.

Es claro que mientras más complejo es el problema planteado, más entradas serán necesarias para representarlo adecuadamente, por esto, las habilidades de representación del perceptrón están seriamente restringidas para la solución de problemas sencillos. Aún más, para iniciar el planteamiento del problema será necesario saber si las variables de entrada pueden ser geométricamente separables, caso contrario, no es aplicable esta neurona artificial.

¿Cómo vencer este gran inconveniente que restringió el uso de las neuronas perceptrón?. En los años 60s, los investigadores estaban conscientes de las limitaciones de los perceptrones. La solución, neuronas artificiales que no utilizan una función lineal como función de transferencia. Estas pueden realizar una clasificación más general de puntos que son contenidos en regiones convexas.

2.5 NEURONA ADALINE

La neurona adaline (ADAptive Linear Element) es en sí muy paraecida a la neurona perceptrón. La diferencia más importante radica en la función de activación, la neurona adaline utiliza la función lineal (purelin), la misma que le permite obtener cualquier valor de salida. La arquitectura se muestra en la figura 2.7.

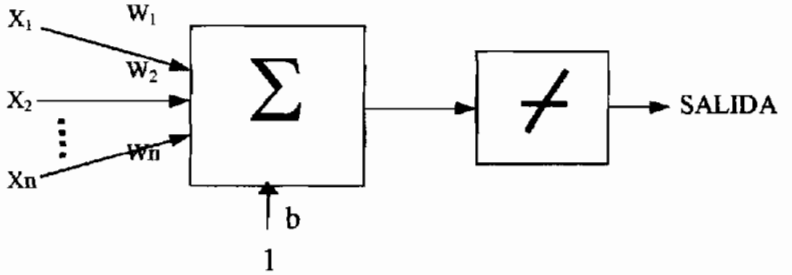


Fig. 2.7 Neurona Adaline

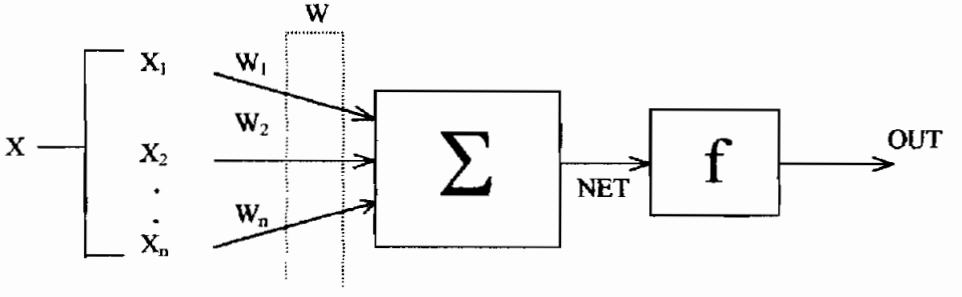
Las neuronas adaline tienen la misma limitación del perceptrón, puesto que pueden resolver únicamente problemas que sean linealmente separables.

Sin embargo, cuando se configura como red puede tener aplicaciones muy interesantes tales como cancelación de ruido y predicción.

2.6 NEURONA BACKPROPAGATION (RETROPROPAGACIÓN)

La separabilidad lineal es un serio limitante en la capacidad del perceptrón para representar funciones pues, es imprescindible conocer si la función dada cumple con este requisito. Las neuronas de retropropagación no tienen esta restricción ya que son capaces de representar cualquier función con un número finito de discontinuidades.

La retropropagación es en realidad un algoritmo matemático poderoso utilizado para entrenar redes que toman su nombre, estas redes están formadas por neuronas artificiales que tienen la configuración mostrada en la figura 2.7:



$$NET = x_1.w_1 + x_2.w_2 + \ldots + x_n.w_n = \sum_{i=1}^{n} x_i.w_i$$

$$OUT = f(NET)$$

Fig. 2.7 Neurona con función de activación no lineal

donde *f* es la función de activación y usualmente es la sigmoide, cuyas características SE mencionará más adelante; esta función introduce no linealidades en la neurona de retropropagación, lo que la hace poderosa frente a

las neuronas perceptrón. Sin embargo, se puede usar otras funciones, el único requisito es que sean derivables en todo su rango.

Como parte de una red estas neuronas cuentan con características y potencialidades muy importantes tales como:

- Son capaces de procesar en paralelo.
- No es necesario programarlas explícitamente, son capaces de "aprender" la relación entre un conjunto de entradas dadas como ejemplo y de aplicar la misma relación para nuevas entradas.
- Tienen la habilidad de concentrarse en las características de una entrada arbitraria que se asemejase a otras vistas previamente y es capaz de ignorar el ruido.

2.7 FUNCIONES DE TRANSFERENCIA DE LAS NEURONALES ARTIFICIALES

La señal NET = X.W, donde X y W son el vector de entrada y peso respectivamente, generalmente es nuevamente procesada por una función de transferencia conocida también como función de activación que producirá la señal de salida definitiva de la neurona.

Esta podría ser cualquiera, incluso una función lineal, no obstante, es mejor elegir la que más aproxime su comportamiento a la característica no lineal de una neurona biológica.

Una neurona perceptrón trabaja con la función de transferencia llamado "limitador duro".

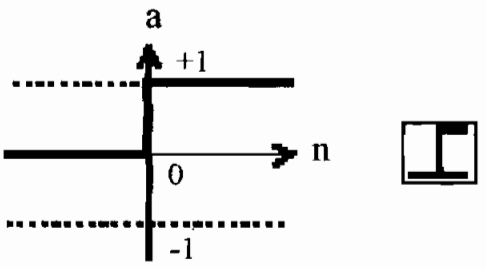


Fig. 2.8 Función limitador duro

Matemáticamente,

$$a = \begin{cases} 1 \text{ si } n \geq 0 \\ 0 \text{ si } n < 0 \end{cases}$$

Esta función es la que da al perceptrón la habilidad de clasificar vectores de entrada.

Las neuronas de retropropagación trabajan con funciones no lineales que tengan la característica de ser derivable en todo su rango. La más utilizada es la función log sigmoide.

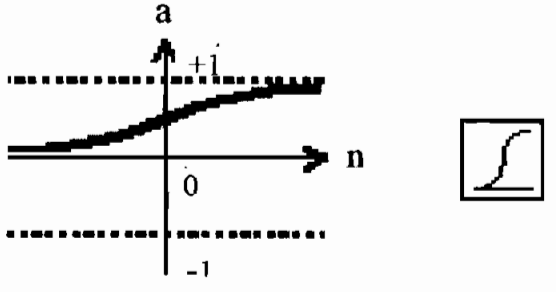
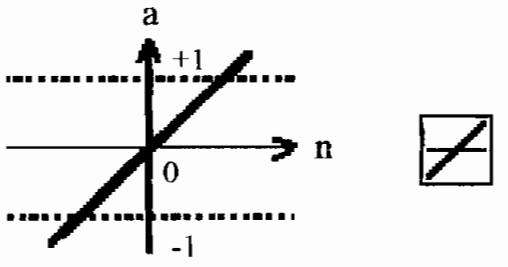


Fig. 2.9 Función log-sigmoide

La función Logsig genera valores de salida entre 0 y 1 a medida que los valores de entrada cambian de negativo a positivo infinito.

Ocasionalmente, la función de transferencia lineal Purelin es usada en las redes Backpropagation.

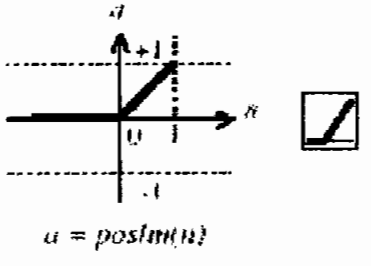


a = purelin(n)

Fig. 2.12 Función lineal

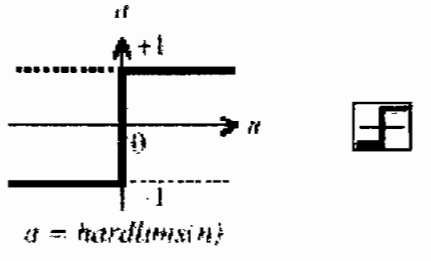
Cuando se usa con una neurona artificial la función sigmoide, entonces la salida se limita a un rango pequeño. Si se usan las neuronas de características lineales la salida puede asumir cualquier valor.

Aunque las tres funciones anteriores son las más utilizadas, alternativamente, pueden usarse varias funciones más tales como:



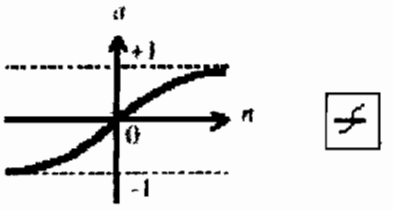
a = poslin(n)

Positive Linear Transfer Funct.



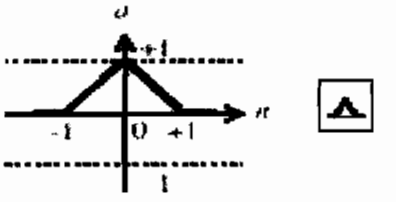
a = hardlims(n)

Symmetric Hard Limit Trans. Funct.



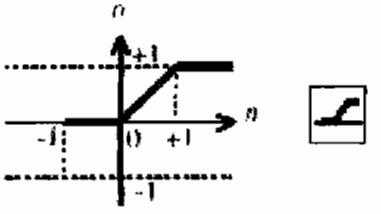
$a = tansig(n)$

Tan-Sigmoid Transfer Function



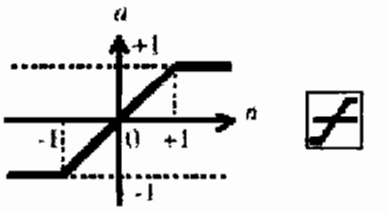
$a = tribas(n)$

Triangular Basis Function



$a = satlin(n)$

Satlin Transfer Function



$a = satlins(n)$

Satlins Transfer Function

3. TALLER

3.1 Conceptos

1. ¿Qué es una neurona perceptrón?
2. Demuestre que con el perceptrón no se puede representar la función XOR
3. ¿Qué significa que una función es linealmente separable?
4. ¿Qué es la retropropagación?
5. ¿Cuál es la importancia de contar con funciones de transferencia no lineales?

6. ¿Qué función cumple la bia en la separabilidad lineal?. Explique gráficamente.

7. ¿Cuáles son las diferencias entre las neuronas perceptrón y las adaline?. ¿Cuáles son los beneficios?

8. ¿Por qué es necesario en las neuronas de retropropagación que la función de transferencia o activación sea derivable en todo su rango?

REFERENCIAS BILIOGRÁFICAS

1. Diccionario de Medicina Océano Mosby

2. Freeman James A., Skapura David M., **"Redes Neuronales "**, Addison-Wesley/Diaz de Santos.

3. Wasserman Philip D., **"Neural Computing"**, Van Nostrand Reinhold.

4. Demuth Howard, Beale Mark, **" Neural Network Toolbox"**, The MathWorks, Inc.

REDES NEURONALES ARTIFICIALES Y SU ENTRENAMIENTO

1. OBJETIVOS

- Reconocer las configuraciones más importantes de redes neuronales artificiales.
- Iniciarse en el conocimiento de los algoritmos de entrenamiento de las redes neuronales.

2. MARCO TEÓRICO

2.1 CONFIGURACIÓN DE REDES NEURONALES ARTIFICIALES

Una red se define mediante el número de capas, el número de neuronas por capa y la función de activación que caracteriza a la red, el vector de entradas, una matriz de pesos, una o varias capas de neuronas denominadas capas ocultas y un vector de salida. Una capa es la agrupación de varias neuronas en un mismo nivel.

2.2 CLASIFICACIÓN DE LAS REDES ARTIFICIALES

Las redes neuronales artificiales se clasifican en:

- Por el número de capas.
  - Capa simple o monocapa
  - Multicapa.

- Por la forma de alimentación
  - Redes de alimentación directa
  - Redes recurrentes

2.2.1 POR EL NÚMERO DE CAPAS

2.2.1.1 Monocapa

Se llaman redes monocapa o de capa simple porque cuentan únicamente con una capa de cómputo o procesamiento que corresponde a la salida. Las entradas son conectadas a los nodos de entrada cuyo papel únicamente es distribuirlas y por tanto no son consideradas como una capa.

Ésta es la configuración de redes más básica y su representación se puede observar en la figura 2.1

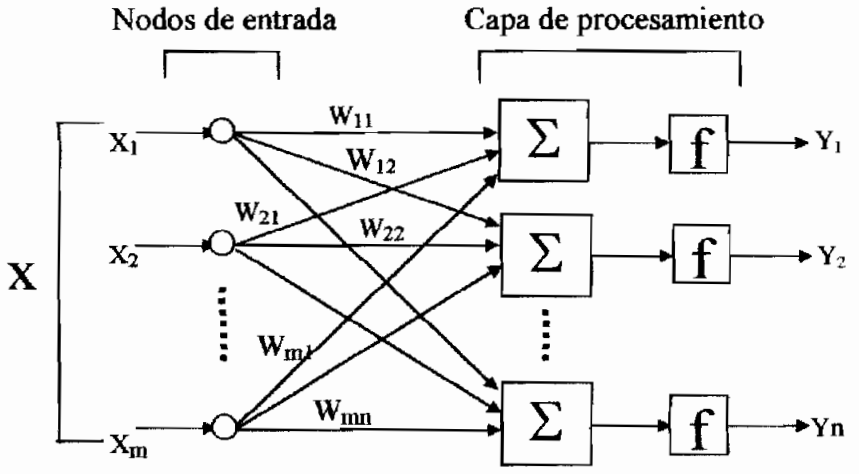


Fig. 2.1 Red de capa simple

La notación utilizada para los pesos es W, estos son elementos de una matriz cuya dimensión es mxn, en donde m está dado por el número de entradas y n por el número de neuronas de la capa.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & w_{m3} & \cdots & w_{mn} \end{bmatrix}$$

De esta forma se reconoce el origen o entrada y el destino o salida. Así, para w_{32} se entiende que proviene de la entrada 3 y va hacia la segunda neurona.

Cada entrada es ponderada con un peso perteneciente a la matriz W, distribuyéndose por cada una de las neuronas de la capa de procesamiento, pasando por su respectiva función de transferencia, el resultado es un vector fila de dimensión nx1.

La figura 2.2 muestra una forma abreviada de una red monocapa que además está conformada por *tendencias* (*bias*).

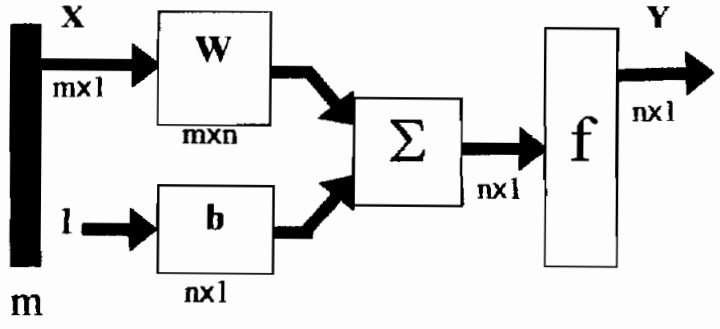


Fig. 2.2 Red monocapa con tendencia

2.2.1.2 MULTICAPA

Corresponde a una red con varias capas de procesamiento en cascada en donde han sido conectadas dos o más redes de capa simple; las salidas de la primera constituyen las entradas de la segunda y así sucesivamente.

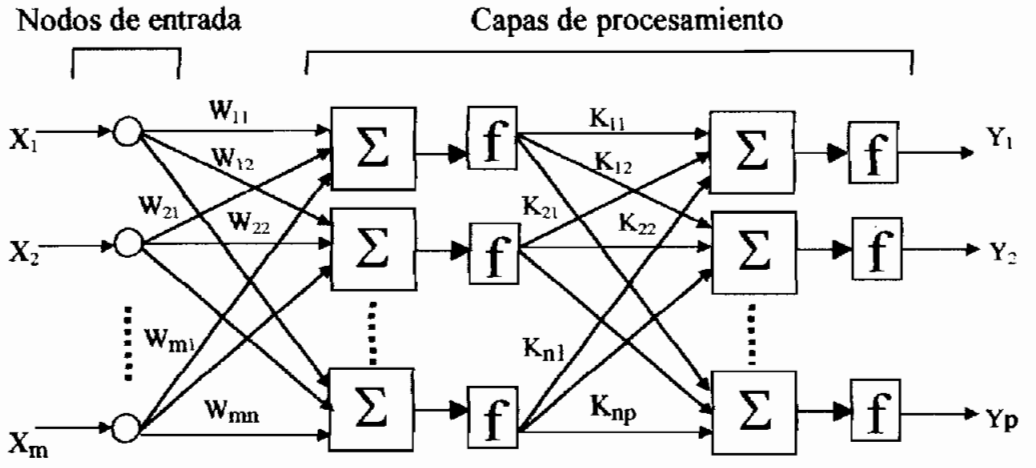


Fig 2.3 Red multicapa

En una red multicapa, la última capa es llamada *capa de salida* y las anteriores a ésta se denominan *capas ocultas*. Para el ejemplo, se tiene una capa de salida y una capa oculta, sin embargo, en la configuración de redes muchas veces se desconoce el número de capas ocultas que posee la red.

La ventaja de las redes multicapa radica en la función de transferencia que se usa en cada capa, si éstas son lineales, nuestra red multicapa lineal equivaldrá a una red de capa simple. Por tanto, el poderío de estas redes consiste en la no linealidad de las funciones de transferencia.

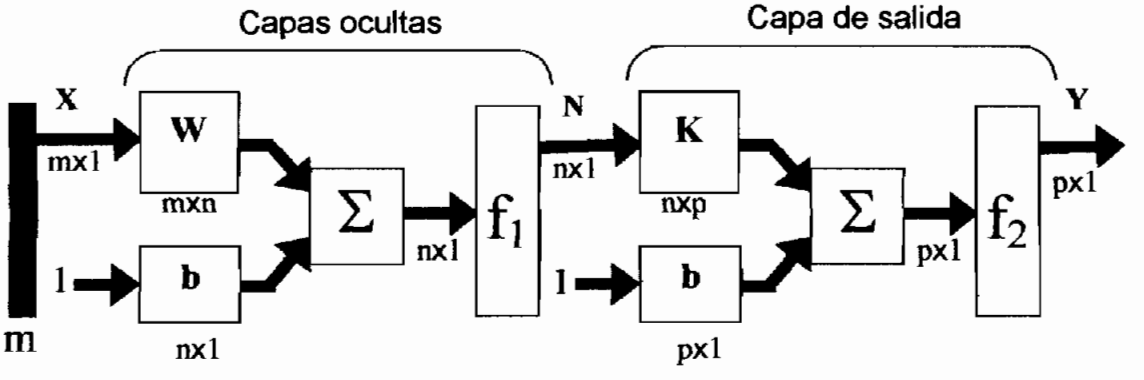


Fig. 2.4 Red multicapa con tendencia

2.2.2 POR LA FORMA DE ALIMENTACIÓN

2.2.2.1 Redes de alimentación directa

Las redes de alimentación directa son todas aquellas redes cuyas conexiones van siempre hacia adelante. La salida de este tipo de redes es determinada solamente por la entrada actual y los valores de los pesos. Dentro de esta clasificación incurren las redes perceptrón, adaline y de retropropagación.

2.2.2.2 Redes recurrentes

Las redes recurrentes son aquellas que incluyen en su arquitectura un lazo de realimentación, es decir, existe una conexión de pesos que va desde la salida de la capa hacia la entrada de la misma o de capas previas.

Cuando la salida de una red recircula hacia atrás a la entrada, la salida dependerá de la entrada actual y de su salida previa; por esta razón, las redes recurrentes pueden exhibir propiedades muy similares a una memoria a corto plazo en los humanos, pues su estado depende en parte de sus entradas previas.

Hay dos configuraciones importantes de este tipo de redes: Elman y Hopfield.

La red Elman presenta comúnmente dos capas como se observa en la figura 2.5.

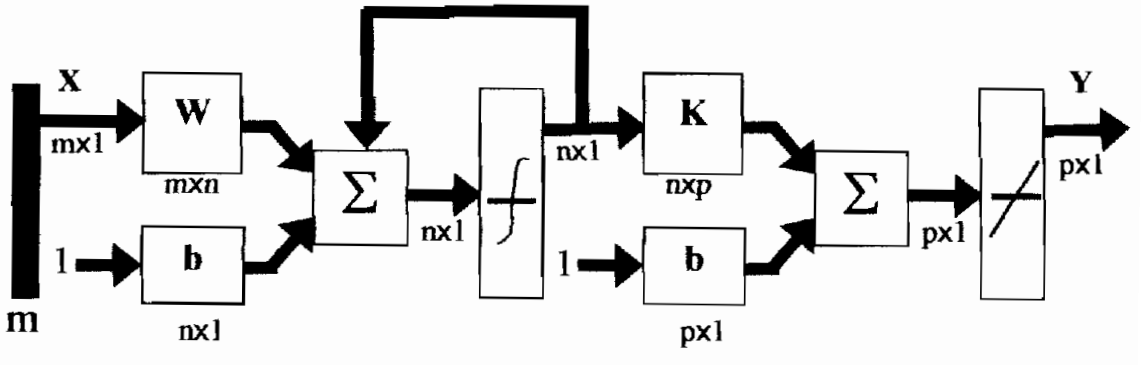


Fig. 2.5 Red recurrente Elman

Mientras que la red Hopfield presenta la siguiente arquitectura:

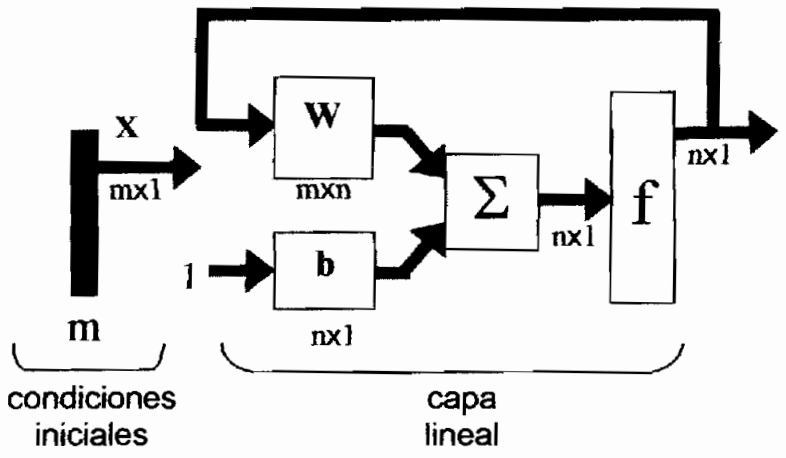


Fig. 2.6 Red recurrente Hopfield

Note que el vector X es la entrada inicial, pero en cuanto la red calcule la salida, ésta llegará a ser la nueva entrada.

Para revisar más sobre las redes recurrentes, se sugiere revisar [2] y [3].

2.3 ENTRENAMIENTO

Para que la red pueda realizar alguna tarea en particular, es necesario entrenarla, esto es, aplicar un conjunto de entradas y mediante algún procedimiento obtener un conjunto de salidas deseadas. Tal procedimiento determinará básicamente una variación en los pesos de conexión mientras convergen a ciertos valores, de forma que los elementos del conjunto de entradas aplicados secuencialmente a la red, produzcan la salida esperada. Los conjuntos de entrada y salida son manejados como vectores.

El entrenamiento puede ser de dos tipos: supervisado y sin supervisión.

El ***entrenamiento supervisado*** implica aplicar a la red *pares de entrenamiento*, formados por n pares de vectores de entradas y salidas deseadas u objetivos.

$$[(x_1,o_1)\ (x_2,\ o_2)\ ...(x_n\ ,o_n)]$$

Donde

x_n es el n-ésimo vector de entrada a la red

o_n es el n-ésimo vector de salida deseada (vector objetivo)

Se ingresa un vector de entrada y se calcula la salida, ésta se compara con el objetivo y se evalúa el error, si existe, se modifican los pesos de tal forma que, la siguiente vez la salida calculada se aproxime lo mejor posible a la salida objetivo.

Dentro de esta categoría están los algoritmos que se verán a continuación.

En el ***entrenamiento sin supervisión***, no hay vectores objetivos disponibles ni comparaciones para determinar la respuesta ideal, la red determina su propio comportamiento solamente basándose en las entradas.

2.4 REDES PERCEPTRÓN

Una red perceptrón es una red cuyas capas de neuronas de procesamiento están conformadas por perceptrones.

Las redes perceptrón se caracterizan por que la función de activación para cada neurona es un limitador duro, esta característica limita las utilidades de estas redes.

Como se indicó en la primera unidad, cuando se usa neuronas perceptrón, la aplicación debe ser linealmente separable.

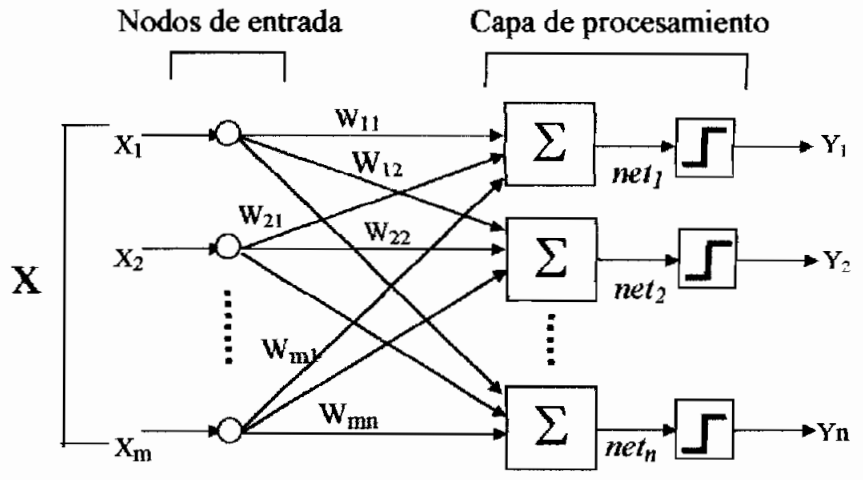


Fig. 2.6 Red perceptrón

2.4.1 ENTRENAMIENTO DE UNA RED PERCEPTRÓN

Rossemblatt en 1962 proporcionó al mundo un algoritmo que hacía al perceptrón capaz de representar funciones.

El entrenamiento, como se mencionó anteriormente, consiste en hacer variar el valor de los pesos de tal forma que se consiga una correcta relación entrada - salida de datos.

Se introduce los pares de entrenamiento formados por los vectores de entradas **X** y el de salidas deseadas **O**. El objetivo es, disminuir el error dado por δ y definido como la diferencia entre el valor objetivo (o) y la salida real (y), $\delta = o - y$.

Note que el vector de salidas deseadas **O** deberá contener valores binarios únicamente, debido a que la red perceptrón trabaja con la función lineal *hardlim*.

El algoritmo de entrenamiento del perceptrón se resume en los siguientes pasos [2]:

1. Asignar valores iniciales randómicos pequeños a los pesos

2. Aplicar una entrada modelo y calcular la salida,

$$net_j = \sum_i x_i . w_{ij}$$

3. Aplicar la función de activación lineal a la señal *net* para cada neurona, como sigue:

$y_j = 1$ si net_j es mayor al valor umbral θ_j

caso contrario $y_j = 0$

donde θ es el valor asociado a la función de activación hardlim

4. Calcular el error para cada neurona

$$\delta_j = o_j - y_j$$

5. Modificar el valor actual de cada peso

$$w_{ij}(t+1) = w_{ij}(t) + \eta . \delta_j . x_i$$

donde

η es un parámetro de velocidad de aprendizaje

$w_i(t)$ es el peso antes de ajustar

$w_i(t+1)$ es el peso después de ajustar

6. Repetir a partir del paso 2 hasta que el error sea mínimo

2.5 REDES ADALINE

Las redes adaline, o madaline por muchas adalines, no difieren mucho de las redes perceptrón, pues como se había mencionado, las neuronas que la conforman son muy similares a las perceptrón, excepto en la función de activación lo que da una ventaja sobre las anteriores.

En la red adaline se hace uso la regla de aprendizaje de mínimos cuadrados LMS que es mucho más poderosa que la regla de aprendizaje del perceptrón.

La figura 2.7 muestra la arquitectura básica de una red adaline.

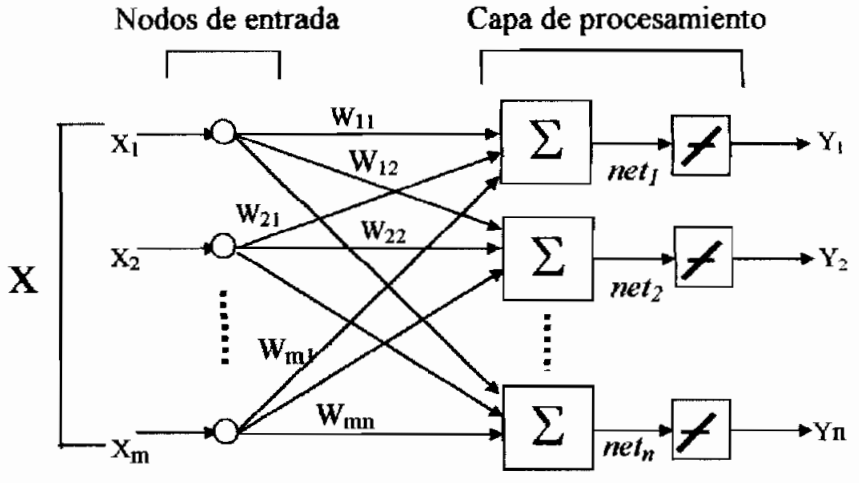


Fig. 2.7 Red Adaline

2.5.1 ENTRENAMIENTO DE UNA RED ADALINE

Como se mencionó, las redes adaline son entrenadas mediante la regla de aprendizaje LMS o algoritmo de Widrow-Hoff. Widrow y su alumno Hoff (1969) extendieron el algoritmo de aprendizaje del perceptrón para salidas continuas. Además desarrollaron la prueba matemática que la red podía converger eventualmente para cualquier función que es capaz de representar.

El algoritmo LMS está basado en el método del descenso más pronunciado (steepest descent procedure). Aquí, nuevamente la red es entrenada sobre ejemplos de comportamiento.

Sean $(x_1, o_1), (x_2, o_2), \ldots (x_L, o_L)$ los pares de entrenamiento, lo que se desea es minimizar el error cuadrático medio entre la salida real y la salida deseada. El error cuadrático medio (MSE) o error esperado se define en la forma:

$$\langle \varepsilon^2 \rangle = \frac{1}{L}\sum_{k=1}^{L}\varepsilon_k^2$$

donde: ε_K es el error de la k-ésima salida y_K y el valor deseado, $\varepsilon_K = d_K - y_K$

L es el número de vectores de entrada

Las ecuaciones de entrenamiento de las Adaline son muy similares a las del perceptrón, la diferencia crítica está en el cálculo del error, cuando los valores ya no son binarios. Entonces, en cada iteración se hará lo siguiente [1]:

1. Se aplica un vector de entrada, $\mathbf{x}_K$ en las entradas del Adaline.

2. Se determina el valor del error cuadrático, $\varepsilon_K^2(t)$, empleando el valor actual del vector de pesos **w**

$$\varepsilon_K^2(t) = (o_K - w^t(t)x_K)^2$$

3. Se calcula una aproximación de $\nabla\xi(t)$ (gradiente de error) empleando $\varepsilon_K^2(t)$, como aproximación de $\langle \varepsilon_K^2 \rangle$:

$$\nabla\varepsilon_K^2(t) \approx \nabla\langle \varepsilon_K^2 \rangle$$

$$\nabla\varepsilon_K^2(t) = -2\varepsilon_K(t)x_K$$

que es el gradiente de la ecuación del paso 2

4. Se actualiza el vector de pesos

$$w(t+1) = w(t) + 2\mu\varepsilon_K x_K$$

donde μ determina la estabilidad y velocidad de convergencia del vector de pesos hacia el valor de error mínimo.

5. Se repiten los pasos 1 a 4 hasta que el error se haya reducido a un valor aceptable.

La definición el parámetro μ es muy importante, si es muy grande, la convergencia no se producirá nunca.

2.6 REDES DE RETROPROPAGACIÓN (BACKPROPAGATION)

Debido a que las redes de capa simple no proporcionaron una solución satisfactoria para problemas que no eran linealmente separables, se buscó otras alternativas para solucionarlos.

La solución la constituían las redes multicapa, la ventaja, poseen más de una capa de procesamiento, pero su fuerza radica principalmente en la función de activación asociada a cada neurona, pues esta no es lineal. A pesar de contar con una solución teórica, estas no fueron usadas debido a que no existía un adecuado algoritmo de entrenamiento que permitiera aleccionar las capas ocultas. No fue hasta el año de 1986, cuando investigadores presentaron al mundo un algoritmo claro y conciso llamado de retropropagación.

Las redes de retroalimentación frecuentemente están conformadas por varias capas de neuronas llamadas ocultas con función de transferencia sigmoide (generalmente, pero puede usarse otras) y una capa de salida. Las capas con neuronas no lineales permiten a la red entrenarse para conseguir una correcta relación de vectores entrada - salida tanto lineal como no lineal. La función de activación que se utilice en la capa de salida dependerá del tipo de respuesta deseada, la función sigmoide (log-sigmoide) proporciona valores de salida entre 0 y 1 mientras que con una función de activación lineal (purelin) la salida puede tomar cualquier valor. No debe usarse a la función umbral (harlim) como función de transferencia en ninguna de las capas, pues el algoritmo de entrenamiento requiere que dicha función sea derivable en todo el intervalo, y la hardlim no lo es justo en el umbral.

Este tipo de redes pueden ser utilizadas como un aproximador general de funciones, pues tienen una alta capacidad de representación. Una red multicapa

apropiada para usarse con el algoritmo de retropropagación es representada en la figura 2.7:

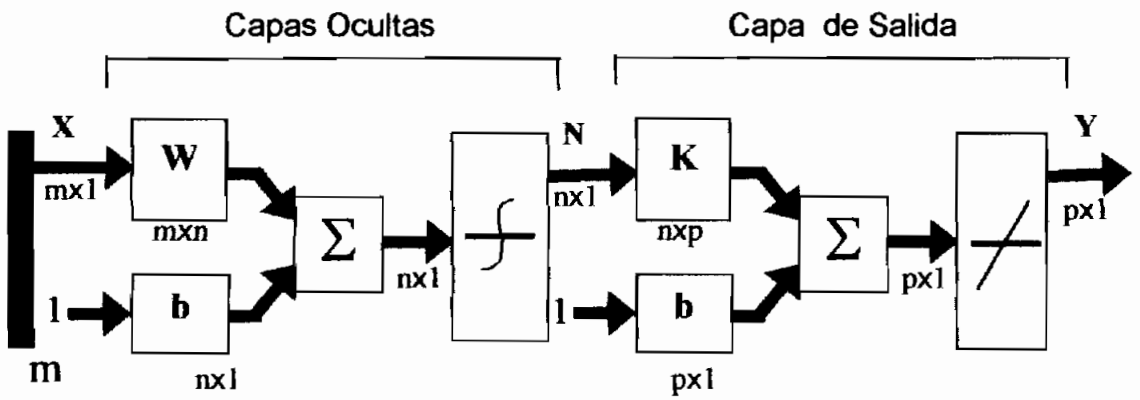


Fig. 2.7 Red de retropropagación

Se explicará ahora el funcionamiento de la red de retropropagación.

Se empieza por componer *pares de entrenamiento* formados por el vector de entrada y el de salidas deseadas u objetivo. Antes de empezar con el entrenamiento, todos los pesos de conexión deben ser inicializados en números randómicos pequeños, por ejemplo ± 0.5, asegurando así que la red no se sature con pesos más grandes. Una trama de los pares de entrenamiento es introducida a manera de ejemplo en la primera capa. Esta excitación inicial produce que la información se vaya propagando hacia el resto de capas (capas superiores), proporcionando una salida, entonces se calcula el error, comparando la salida real con la deseada.

Este error es transmito hacia atrás, es decir, hacia las capas intermedias, pero cada unidad recibirá una fracción de este error en proporción a lo que aportó para obtener la salida. Este proceso se repite capa por capa hasta que cada unidad en la red haya recibido una señal de error que describa su contribución relativa al error total. Se procede entonces a actualizar los pesos de conexión de cada unidad dependiendo de la señal de error que hayan percibido. Esto se repite hasta que la red converja a una solución para todas las tramas de entrenamiento.

Un esquema se muestra en la siguiente figura:

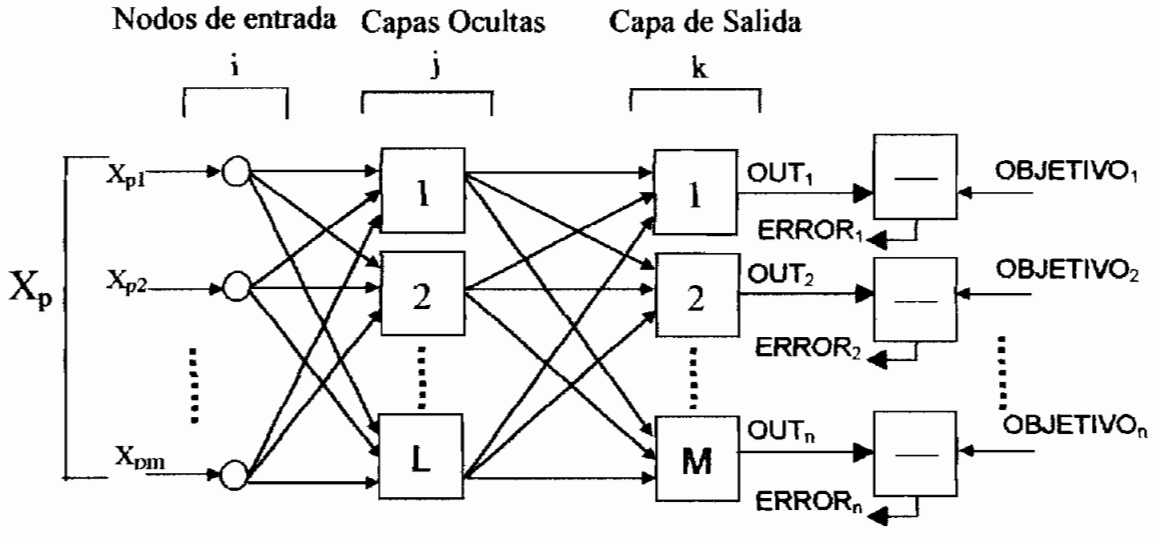


Fig. 2.8 Red de retropropagación

2.6.1 ENTRENAMIENTO DE LA RED

A continuación, se presenta paso a paso el algoritmo con las correspondientes ecuaciones, para un par de vectores de entrenamiento, de acuerdo a la regla *delta descendente* [1]:

1. Se aplica el vector de entrada $\mathbf{x_p} = (x_{p1}, x_{p2}, x_{p3}, .., x_{pm})$, en la capa de entrada de la red.

2. Se calculan los valores netos procedentes de las entradas para las unidades de la capa oculta.

$$neta_{pj}^{h} = \sum_{i=1}^{m} w_{ji}^{h} x_{pi} + b_{j}^{h}$$

donde:

j es el número de la capa oculta

p es el número de la entrada

w^h_{ji} es el peso de la conexión procedente de la i-ésima unidad de entrada

b^h_j es el término de tendencia (bias), su uso es opcional

h se refiere a magnitudes de la capa oculta

3. Se calculan las salidas de la capa oculta

$$i_{pj} = f^h_j\left(neta^h_{pj}\right)$$

que indica que la salida depende de la función de activación de la j-ésima capa oculta

4. Se pasa a la capa de salida. Se calculan los valores netos de las entradas para la k-ésima unidad de la capa de salida

$$neta^o_{pk} = \sum_{j=1}^{L} w^o_{kj} . i_{pj} + b^o_k$$

el índice o indica valores de la capa de salida

w^o_{kj} es el peso de conexión entre la j-ésima unidad de la capa oculta y la k-ésima unidad de la capa de salida

5. Se calculan las salidas

$$y_{pk} = f^k_o\left(neta^o_{pk}\right)$$

6. Se calculan los términos de error para las unidades de salida

$$\delta^o_{pk} = \left(o_{pk} - y_{pk}\right) f^{o'}_k\left(neta^o_{pk}\right)$$

donde

δ^o_{pk} indica el error de la k-ésima unidad de salida para el p-ésimo par de entrenamiento

o_{pk} es el valor de salida deseado (objetivo)

$f^{o'}_k$ es la derivada de la función de activación de la k-ésima unidad de salida

7. Se calculan los términos de error para las unidades ocultas

$$\delta^{h}_{pj} = f^{h'}_{j}\left(neta^{h}_{pj}\right)\sum_{k}\delta^{o}_{pj}w^{o}_{kj}$$

$f^{h'}_{j}$ es la derivada de la función de activación de la j-ésima unidad de la capa oculta

8. Se actualizan los pesos de la capa de salida

$$w^{o}_{kj}(t+1) = w^{o}_{kj}(t) + \eta.\delta^{o}_{pk}.i_{pj}$$

el factor η se denomina *parámetro de velocidad de aprendizaje*, que es positivo y de su selección depende el rendimiento de la red. Por lo general, este factor deberá ser pequeño (típicamente 0,05 a 0,25) para asegurarse que la red llegue a una solución. Este término indica el número de iteraciones en que incurrirá la red, mientras más pequeño es el valor, mayor será número de iteraciones.

9. Se actualizan los pesos de la capa oculta

$$w^{h}_{ji}(t+1) = w^{h}_{ji}(t) + \eta.\delta^{h}_{pj}.x_{i}$$

La forma de saber si la red está aprendiendo o no, es calculando el término de error:

$$E_{p} = \frac{1}{2}\sum_{k=1}^{M}\delta^{2}_{pk}$$

Cuando a este término se lo puede considerar pequeño, entonces el proceso se da por terminado. Sin embargo, muchas veces el proceso termina antes de llegar a un error mínimo.

2.5.1.1. Desventajas del algoritmo de entrenamiento de gradiente descendente

El algoritmo de gradiente descendente sigue la cuesta de la superficie del error que se extiende hacia abajo, mientras ajusta constantemente los pesos, es decir, busca minimizar el error.

En una red multicapa, la superficie de error puede ser muy compleja, llena de valles, colinas y barrancas, como se observa en la figura 2.9:

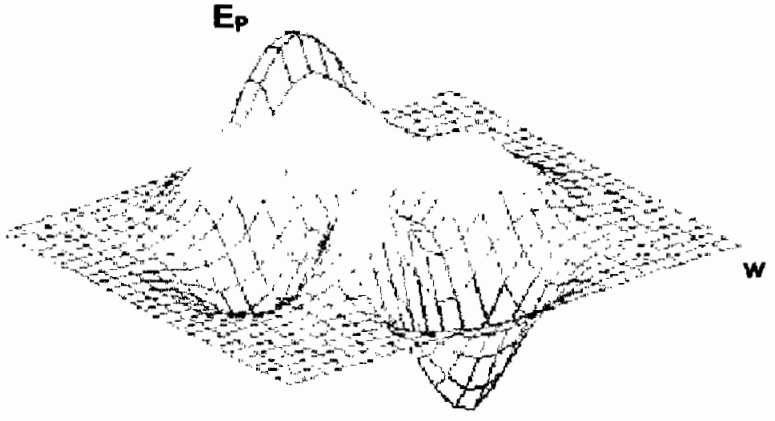


Fig. 2.9 Superficie de error

Se dice que la red converge a un *mínimo local* si al ajustar los pesos, la red tiende a un punto A (fig. 2.10), y al intentar reajustarlos moviéndolos a ambos lados de esta posición, el error calculado es mayor al determinado en A. Cuando esto sucede, la red parará el aprendizaje asumiendo que ha llegado a una solución, pero el error aún puede ser muy grande.

Con este algoritmo, la razón de entrenamiento debe ser pequeña para que haya estabilidad debido a los mínimos locales.

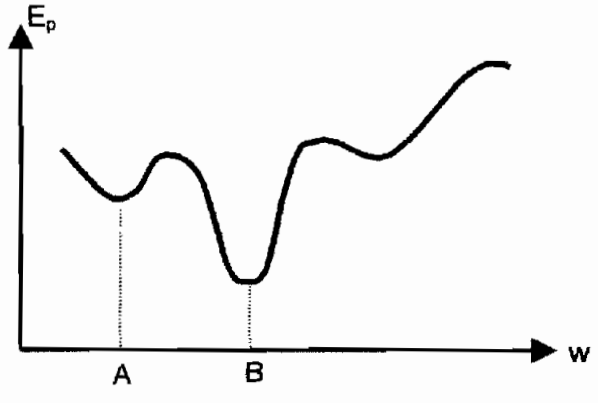


Fig. 2.10 Corte transversal de una superficie de error

Este algoritmo depende fuertemente de las derivadas parciales de la función de activación, en lo que se refiere a calcular el ancho del paso para el ajuste de pesos. Este hecho puede provocar la llamada "parálisis" en el entrenamiento de la red, si la derivada de, por ejemplo la función sigmoidal, se aproxima a cero, los cambios en los pesos serán infinitesimales produciendo un estancamiento virtual.

2.5.1.2 Otros métodos de entrenamiento

Se han hecho varias modificaciones al algoritmo de entrenamiento basado en el gradiente descendente para lograr mayor rapidez y estabilidad en la red, uno de los resultados es el método llamado "*momento*".

Este método implica añadir un término al ajuste del peso que es proporcional al valor de cambio previo del peso. Una vez un ajuste se ha hecho, es "recordado" y sirve modificar todo ajuste de peso subsecuente. La ecuación de ajuste se modifica como sigue:

$$\Delta w_{kj}^{o}(t+1) = \eta.\delta_{pk}^{o}.i_{pj} + \alpha\left[\Delta w_{kj}^{o}(t)\right]$$

$$w_{kj}^{o}(t+1) = w_{kj}^{o} + \Delta w_{hj}^{o}(t+1)$$

Donde α es el coeficiente "momento", y su valor está comúnmente alrededor de 0.9.

Con la variación "momento" se logra usualmente mayor rapidez que en el método de gradiente descendente simple, ya que permite altas velocidades de aprendizaje mientras mantiene la estabilidad, sin embargo continua siendo lento para algunas aplicaciones.

Otro método importante es el algoritmo combinado Cauchy - retropropagación. Con éste se evita totalmente el problema de los mínimos locales, la parálisis de la red y se aumenta eficazmente la velocidad de entrenamiento. El estudiante puede consultar sobre los algoritmos en [2] y [3].

2.6 OBSERVACIONES GENERALES

Hay varias consideraciones que el alumno deberá aplicar ya en la práctica.

2.6.1 DIMENSIONAMIENTO DE LA RED

No hay una regla que indique cuantos nodos o elementos conformarán una capa o incluso cuantas capas deben integrar la red.

Muchas veces es suficiente una red de dos capas (una capa oculta y una de salida), sin embargo, para problemas más complejos una red con más de una capa oculta puede ser más eficiente (aprender más rápido).

En cuanto a las neuronas de cada capa, una pauta para escoger el número de nodos de entrada y los de la capa de salida puede ser la aplicación que se dará a la red. El número de nodos de la capa de salida puede depender del resultado que se desea, es decir, una salida analógica o digital.

Sin embargo, para definir la capa oculta no hay ningún patrón. La idea es usar el menor número de elementos, de modo que cada unidad de procesamiento no se convierta en una carga al momento de la simulación. En [1] sugiere que, aún cuando las entradas a una red son cientos o miles, los nodos de la capa oculta serán apenas una fracción de las entradas, no obstante, si la red no convergiera, se aumentará su número y si por el contrario converge, podría ser posible eliminar algunos nodos, aquellos que menos aportan a la consecución de resultados.

2.6.2 VECTORES DE ENTRENAMIENTO

Formar los pares de entrenamiento adecuados es fundamental para que la red converja a una solución.

¿Cuántos pares deben ser introducidos a la red para un apropiado entrenamiento?. En realidad, se puede aplicar a la red todos los pares que se tienen disponibles pero no es necesario, bastará con un pequeño subconjunto del total.

Si se quiere que la red sea capaz de reconocer tramas, aún si éstas vienen acompañadas por ruido, será conveniente incluir en el conjunto de entrenamiento pares que contengan ruido, de tal forma que, después de un entrenamiento adecuado, la red sea capaz de converger.

Si bien una red es capaz de generalizar, es decir, reconocer las similitudes entre los datos de entrenamiento y los que nunca ha visto, no es capaz de extrapolar. Esto significa que, si durante el entrenamiento se aplicó vectores muy definidos, la respuesta podrá ser imprecisa. Para evitar esto, cerciórese de que los datos de entrenamiento cubran el espacio de entradas esperado.

2.6.3 PARÁMETROS DE ENTRENAMIENTO

El *parámetro de velocidad de aprendizaje* η como se dijo anteriormente, indica el número de iteraciones que realizará la red, y puede tomar valores del orden de 0,01 a 1. No obstante, no se deberá abusar de este valor, es decir, escogerlo muy pequeño, pues puede provocar que el proceso sea innecesariamente muy largo y sobrecargado, y si por el contrario es muy grande, el proceso podría volverse inestable, es decir, no convergería nunca. No hay una regla que oriente tal decisión, la única guía la constituirá la experiencia.

Si la red no ha alcanzado un resultado aceptable, es posible cambiar el número de nodos en la capa oculta o algún parámetro de entrenamiento, inclusive se puede volver a entrenar a la red solamente cambiando los valores con que inician los pesos.

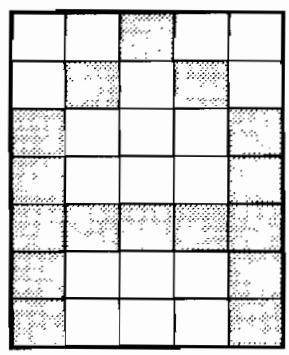
3. TALLER

1. Si un perceptrón simple dividía el plano en dos regiones separadas por una frontera lineal, una red perceptrón de 2 capas, ¿qué región de decisión es capaz de formar?. Justifique su respuesta.

2. ¿Cuántas capas serán necesarias para que una red perceptrón sea capaz de entrenarse para resolver el ejercicio de la compuerta XOR?. Grafique la región de decisión.

3. Considere una red de retropropagación de tres capas, tal que todos sus pesos han recibido el mismo valor inicial en todas las unidades. Demostrar que esta red nunca podrá aprender nada. Interpretar este resultado en términos de la superficie de error en el espacio de pesos.

4. Defina el número de elementos de entrada y el número de neuronas de salida que requerirán las siguientes aplicaciones. Explique su respuesta.
   a. Se desea entrenar una red para el reconocimiento de las letras del alfabeto. Suponga que las letras son representadas en un sistema de imágenes que digitaliza cada letra formando una matriz de 7x5 elementos como indica la figura.



   b. Se entrena a una red para realizar la compresión de datos de 4 a 1 de un sistema de video de 64 pixels. En este caso, se deberá indicar cuantas

neuronas debe contener la capa oculta que permitirá la transmisión del dato comprimido antes de recuperarse.

5. Realice un análisis entre las configuraciones estudiadas y en forma resumida indique las ventajas y deficiencias y cada una de ellas.

REFERENCIAS BIBLIOGRAFICAS

1. Freeman James A., Skapura David M., **"Redes Neuronales "**, Addison-Wesley/Diaz de Santos.

2. Wasserman Philip D., **"Neural Computing"**, Van Nostrand Reinhold.

3. Demuth Howard, Beale Mark, **" Neural Network Toolbox, Matlab"**, The MathWorks, Inc.

4. Winston Patrick Henry, **"Inteligencia Artificial"**, Adisson-Wesley Iberoamericana.

5. Sánchez José, "**Identificación, control y simulación de un sistema usando Redes Neurales"**, Tesis, Escuela Politécnica Nacional.

APLICACIONES DE LAS REDES NEURONALES ARTIFICIALES

OBJETIVOS

- Utilizar los comandos e instrucciones del Maltab que sirven para implemetar una red neuronal artificial.

- Implementar en la práctica una red neuronal usando el paquete computacional Matlab.

MARCO TEÓRICO

Para el entrenamiento y simulación de las redes neuronales artificiales se dispone de una muy buena herramienta, el paquete computacional MATLAB.

En general una red tiene la siguiente estructura y parámetros:
<<net

architecture:

numInputs: 1	número de vectores de entrada
numLayers: 1	número de capas de la red
biasConnect: [1]	bias conectadas a las neuronas de la red
inputConnect: [1]	conexión entre el vector de entrada y la primera capa
layerConnect: [0]	conexión entre capas intermedias
outputConnect: [1]	conexión entre la última capa y la salida
targetConnect: [1]	conexión de un vector de salidas objetivo

numOutputs: 1 (read-only)	número de vectores de salida
numTargets: 1 (read-only)	número de vectores de salidas objetivo
numInputDelays: 0 (read-only)	número de retardos a la entrada
numLayerDelays: 0 (read-only)	número de retardos de capa

subobject structures:

inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1 output
targets: {1x1 cell} containing 1 target
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights

functions:

adaptFcn: 'adaptwb'	Define la función a ser usada cuando la red adapta pesos y bias (no siempre se usa).
initFcn: 'initlay'	Define la función a usar para inicializar la matriz de pesos y el vector de bias de la red.
performFcn: 'mae'	Define la función que se usará para medir el rendimiento de la red.
trainFcn: 'trainwb'	Define la función a usar para entrenar la red.

parameters:

adaptParam: .passes	Define los parámetros y valores de la actual función adapt.
initParam: (none)	Define los parámetros y valores de la actual función de inicialización.
performParam: (none)	Define los parámetros y valores de la actual función de rendimiento.
trainParam: .epochs, .goal, .max_fail, .show, .time	Define los parámetros y valores de la actual función de entrenamiento (iteraciones, meta, max. error, mostrar , resultados después de # iteraciones, etc.)

weight and bias values:

IW: {1x1 cell} containing 1 input weight matrix
LW: {1x1 cell} containing no layer weight matrices
b: {1x1 cell} containing 1 bias vector

Para ampliar los conceptos anteriores, el estudiante puede referirse a [1].

2.1 PERCEPTRÓN

2.1.1 CREACIÓN DE UNA RED PERCEPTRÓN

Para crear una red perceptrón se usa la siguiente instrucción:

net = newp(PR,S)

Donde:

PR es un vector de R x 2, (R es el número de entradas a la red), en donde se asigna los valores máximo y mínimo de cada entrada.
S es número de neuronas

La función de transferencia es definida por defecto, pues las neuronas perceptrón usan la función limitador duro (hardlim).

Esta instrucción crea una red de una sola capa con un número específico de neuronas (S) y un solo vector de entrada con R elementos.

Por ejemplo:

net= newp([-1 1; 0 2],1);

Define una red de una neurona y un vector de entrada con 2 elementos, cuyos rangos son -1 a 1 y 0 a 2 respectivamente.

Examinar la estructura de los pesos de la red es posible escribiendo la instrucción:

inputweights = net.inputweights{1,1} (vector 1 a capa 1)

con esto,

```
inputweights =
        delays: 0
        initFcn: 'initzero'
        learn: 1
        learnFcn: 'learnp'
        learnParam: []
        size: [1 1]
        userdata: [1x1 struct]
        weightFcn: 'dotprod'
```

La función de aprendizaje por defecto es '*learnp*', la cual calcula, de acuerdo al error, los cambios necesarios en los pesos y bias para alcanzar la salida deseada. La función '*dotprod*' genera el producto de los elementos del vector de entrada con los pesos y añade las bias para generar la entrada a la función de transferencia.

Se visualizará de igual manera la configuración de las bias si se escribe la siguiente instrucción:

biases = net.biases{1}

Los valores de pesos y bias pueden ser inicializados con la función '*initzero*' con lo cual, éstos inician en cero; o con '*rands*', en cuyo caso los valores se inician en un valor randómico diferente de cero. La forma de hacerlo es con la instrucción:

```
net.inputweights{1,1}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
```

Sin embargo, se puede definir también los pesos y bias "manualmente", así, para una red de una neurona y dos elementos en el vector de entrada:

```
w = [1 -1];
net.IW{1,1}.value = w;
net.biases{1}.value = [1];
```

Luego de definir dicha función, se inicializa la red (los valores de pesos y bias) con la instrucción:

```
net = init(net);
```

2.1.2 SIMULACIÓN Y ENTRENAMIENTO

Para simular el comportamiento de la red se dispone del comando de Matlab 'sim', con el formato:

```
a = sim(net,P);
```

donde: net es el nombre de la red
P es el vector de entrada

Por ejemplo, sea el vector de entradas P = [1; 1], simular la red *net* da como resultado:

```
P = [1;1];
a = sim(net,P)
a = 1
```

Para otro vector P1 = [1; -1] se tiene

P1 = [1;-1]
a1 = sim(net,P1)
a1 = 0

Pero es posible ingresa las dos entradas en secuencia y se obtiene la salida en secuencia también:

P = {[1;1] [1;-1]};
a = sim(net,P)
a = [1] [0]

Ahora, la regla de aprendizaje learnp, cuyo algoritmo es el mismo que se había explicado en la unidad anterior, sigue una serie de pasos que se indican a continuación a modo de ejemplo:

Ejemplo 1:

1. Se crea la red perceptrón.
   net = newp([-1 1; -1 1],1)
   En este caso se configura una sola neurona y dos elementos para el vector de entrada.
   Note que al no variar la función de inicialización, los pesos iniciarán en 0. Obviaremos las bias.
   net =init(net);

2. Se define el vector de entrada y la salida deseada.
   P = [2; 1] vector de entrada
   T = [0] salida objetivo
3. Se simula la red.
   a = sim(net, P)
   a = 1

4. Se calcula el error.

   e = T - a

   e = -1

5. Se calcula el cambio que se hará a los pesos.

   dw = learnp(w,p, [],[],[],[],e,[],[],[])

   dw = learnp([0 0],P,[],[],[],[],1,[],[],[])

   dw = 1 0

   en el caso de las bias:

   db = learnp(b,1, [],[],[],[],e,[],[],[])

6. Se establece el nuevo valor de los pesos.

   $w = w_{viejo} + dw$

   $b = b_{viejo} + db$

   w = [0 0]+dw

   w = 1 0

7. Se vuelve a simular la red.

   a = sim(net,P)

   a = 1

8. Si no se obtiene la respuesta adecuada se vuelve al paso 4, caso contrario finaliza el entrenamiento.

Este algoritmo de aprendizaje asegura la convergencia de la red en un número finito de iteraciones, siempre y cuando el problema pueda ser resuelto con la separabilidad lineal.

2.1.2.1 Entrenamiento Adaptivo

El entrenamiento Adaptivo implica crear un lazo de cálculo con cada pasada de los elementos de entrenamiento. Una pasada (pass) es aplicar todas las entradas con sus respectivas salidas objetivo. El número de pasadas que

realizará el entrenamiento para intentar alcanzar una solución se define con la instrucción:

net.adaptParam.passes = #;

Ejemplo 2:

Se crea una red de dos neuronas y un vector de entrada con dos elementos, de rango -2 a 2.

```
net = newp([-2 2;-2 2],2);
P={[2;2][1;-2][-2;2][-1;1]};
T={[0;1][1;1][0;1][1;1]};
net.adaptParam.passes = 4;
[net,a,e]=adapt(net,P,T);
```

Luego de que se ha reajustado los pesos y bias por cuatro ocasiones, una a la vez, se obtiene el resultado deseado:

$$a\{1\} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad a\{2\} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad a\{1\} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad a\{2\} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Como se observa corresponde exactamente T.

El entrenamiento adaptivo usa la regla de aprendizaje del perceptrón como algoritmo y converge a una solución en un número finito de pasos, a menos que el problema no pueda ser resuelto mediante separación lineal.

2.2 RED ADALINE

2.2.1 CREACIÓN DE UNA RED ADALINE

2.2.1.1 Newlin

Una red ADALINE de capa simple puede definirse con:

net = newlin(PR,S,ID,LR);

donde: PR es una matriz Rx2 de los valores mínimos y máximos de los R elementos de entrada
S es el número de neuronas en la única capa
ID es el vector de retardo de entrada, por defecto es [0]
LR es la razón o velocidad de aprendizaje, por defecto 0.01

Es importante definir correctamente la razón de cambio antes del entrenamiento, pues si se la escoge LR muy grande la red puede volverse inestable, por el contrario, si se la escoge muy pequeña, la red puede tardar demasiado en converger. Una forma de verificar cual es el máximo valor que se puede usar es con la instrucción '*maxlinlr*', entonces,

LR = maxlinlr(P,'bias');

Y así aseguramos la estabilidad de la red.

Las funciones definidas por defecto cuando se crea una red ADALINE usando newlin son:

'dotprod' como la función de pesos, 'netsum' es la función de entradas de la red y 'purelin' es la función de transferencia. Los pesos y bias se inicializan con 'initzero' y son actualizados mediante 'learwh' (algoritmo de aprendizaje de

Widrow-Hoff). La adaptación (lazo de cálculo) y el entrenamiento se realizan con 'adaptwb' y 'trainwb'. Por último, el rendimiento es medido con 'mse' (error mínimo cuadrado).

La red así definida puede ser simulada usando la misma instrucción que usáramos para el perceptrón.

Ejemplo 3:

net = newlin([-2 2; -2 2],1);

crea una red lineal con una neurona y un vector de entrada con dos elementos de rangos [-2 2].

Recordemos que por defecto la red inicializa los valores de pesos y bias con cero. Para cambiar estos valores iniciales se escribe:

net.IW{1,1} = [1 5];
W = net.IW{1,1}
net.b{1} =[-2];
b = net.b{1}

Antes de simular la red definimos el vector de entrada P:
P = [1; 2];

Luego,
a = sim(net, P);

entonces, a = 9

2.2.1.2 Newlind

Para diseñar una red lineal ADALINE se utilza la función 'newlind'. A diferencia de otras, esta instrucción sólo necesita que el vector de entradas y de salidas objetivo sean conocidas.

net = newlind(P,T)

donde: P es una matriz RxQ de Q vectores de entrada.
T es una matriz SxQ de Q vectores de salida objetivo.

Ejemplo 4:

Diseñar una red lineal dadas las entradas P y salidas T.

P = [1 2 3];
T = [2.0 4.1 5.9];

Usando newlind se crea la red así:
net = newlind(P,T);

La arquitectura queda definida de la siguiente forma:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [1]

numOutputs: 1 (read-only)

numTargets: 1 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

Por último se puede realizar la simulación.
a = sim(net,P)

Esta función permite encontrar los pesos y bias adecuados para minimizar el error (mse).

2.2.1 ENTRENAMIENTO

Cuando se utiliza 'newlind' a la vez que se configura la red se la entrena para alcanzar la salida deseada. Esta es una función útil y descomplicada para conseguir buenos resultados. Más adelante se desarrollará un ejemplo.

Otra forma de entrenar una red ADALINE es usando la función 'train'. Esta función aplica cada vector de un conjunto de vectores de entrada, calcula la salida, compara con el objetivo, computa el error medio cuadrático. Si la meta es alcanzada o si el número de iteraciones (epoch), que es igual a una pasada, se alcanza, el entrenamiento termina.

Ejemplo 5:

Ingresando cuatro vectores de entrenamiento:
P = [2 1 -2 -1;2 -2 2 1];
P =

2	1	-2	-1
2	-2	2	1
P_1	P_2	P_3	P_4
↓	↓	↓	↓
T_1	T_2	T_3	T_4

se desea que la salida para cada vector sea dada por
T = [0 1 0 1];

El procedimiento es el siguiente.

Creamos la red de una neurona:
net = newlin([-2 2; -2 2],1);

Definimos el error meta al que hay que llegar:
net.trainParam.goal= 1e-5;

Empieza el entrenamiento:
[net, tr] = train(net,P,T);

Se produce la siguiente información:
TRAINWB, Epoch 0/100, MSE 0.5/0.1.
TRAINWB, Epoch 25/100, MSE 0.181122/0.1.
TRAINWB, Epoch 50/100, MSE 0.111233/0.1.
TRAINWB, Epoch 64/100, MSE 0.0999066/0.1.
TRAINWB, Performance goal met.

Simulando la red:
a = sim(net,P)
a = 0.0282 0.9672 0.2741 0.4320

Como se observa la respuesta no es la mejor, si se habría escogido una meta más pequeña es posible que se consiguiera un mejor resultado, sin embargo en algunos casos este error puede ser aceptable.

2.3 REDES DE RETROPROPAGACIÓN (BACKPROPAGATION)

2.3.1 CREACIÓN DE UNA RED DE RETROPROPAGACIÓN

Una red de retropropagación puede ser creada usando la instrucción:

net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

Donde:

PR matriz Rx2 de valores mínimos y máximos de R elementos de entrada
Si tamaño de la i-ésima capa, para Nl capas
TFi función de transferencia de la i-ésima capa, por defecto = 'tansig'
BTF función de entrenamiento de la red, por defecto = 'trainlm'
BLF función de aprendizaje de los pesos y las bias, por defecto = 'learngdm'
PF función de rendimiento , por defecto = 'mse'

Las funciones de transferencia pueden ser cualquier función, tales como 'tansig', 'logsig', o 'purelin'.

La función de entrenamiento puede ser cualquier función tales como 'trainlm', 'trainbfg', 'trainrp', 'traingd', etc.

El algoritmo de aprendizaje puede ser 'learngd', o 'learngdm'.

La función de rendimiento puede ser cualquiera de las diferentes funciones diferenciables de rendimiento tales como 'mse' o 'msereg'.

Se inicializan los pesos y bias de cada capa con 'initnw'

Ejemplo 6:

Para crear una red retropropagación de 3 capas se escribe lo siguiente:

```
net=newff([-1 2; 0 5],[3,2,1],{'tansig','logsig','purelin'},'traingd');
```

Esta instrucción crea una red con un vector de entrada de dos elementos y 3 capas. La primera capa con 3 neuronas y función de transferencia tansig, la segunda con 2 neuronas y función logsig, por último, la capa de salida con una sola neurona y función lineal purelin. La función de entrenamiento es traingd.

Además de crear la red, esta instrucción inicializa los pesos y umbrales dejando lista a la red para el entrenamiento.

Antes de pasar al entrenamiento, es recomendable inicializar pesos y umbrales.

La función de inicialización predefinida de la red es 'initlay' (net.initFcn = 'initlay'), ésta permite a cada capa usar su propia función de inicialización. Ahora para definir las funciones de inicialización para la primera capa se usa las siguientes líneas:

net.layers{1}.initFcn = 'initwb';	función de inicialización de la primera capa
net.inputWeights{1, 1}.initFcn = 'rands';	pesos que unen la entrada 1 con la capa 1
net.biases{1, 1}.initFcn = 'rands';	bia de la capa 1 neurona 1
net.biases{2, 1}.initFcn = 'rands';	bia de la capa 1 neurona 2
net = init(net);	inicializar pesos y bias

net.layers{i}.initFcn determina la función de inicialización de cada capa, las funciones comúnmente usadas son: 'initwb' o 'initnw'.

net.inputWeights{i,j}.initFcn puede usar la función randómica que asigna valores iniciales a los pesos entre un rango de -1 a 1.
net.inputbiases{i,j}.initFcn también suele usarse con la función randómica.

2.3.2 ENTRENAMIENTO

Luego de inicializados los pesos y bias la red finalmente está lista para ser entrenada.

El algoritmo de aprendizaje que utilizan estas redes es el de retropropagación. Hay muchas variantes de este algoritmo, sin embargo, el más simple de implementar es el gradiente descendente.

Así mismo, el gradiente descendente puede ser implementado de dos modos: modo incremental y modo por lotes. No obstante, los algoritmos de entrenamiento de gradiente descendente y su variación gradiente descendente con momento tanto en modo incremental como por lotes son muy lentos en la práctica. El alumno puede dirigirse a [1] para revisar la implementación de estos algoritmos.

Existen algoritmos de entrenamiento que convergen mucho más rápido que cualquiera de los anteriores.

En general, los algoritmos de entrenamiento requieren que se defina los siguientes parámetros:

- *epochs*, número de épocas, el entrenamiento parará si el número de iteraciones es mayor al número aquí definido.
- *show*, indica cada cuantas épocas se desplegará información sobre el entrenamiento.
- *goal*, error meta, el entrenamiento se detendrá una vez que se ha alcanzado este error.
- *time*, tiempo, otra razón por la cual el entrenamiento puede detenerse es si el tiempo que ha durado el proceso ha excedido este valor [segundos].

- *lr*, razón de aprendizaje.
- *Min_grad*, mínimo gradiente, si la magnitud del gradiente es menor a este parámetro, el entrenamiento cesa.
- *Max_fail*, máximo valor de fracaso, es usado para mejorar la generalización de la red.

Los parámetros más importantes que deben ser ingresados son: epoch, goal show y lr.

Las funciones ***'traingda'*** (sin momento) y ***'traingdx'*** (con momento) son algoritmos que adaptan la razón de aprendizaje durante el entrenamiento según el algoritmo se mueva por la superficie de rendimiento. Estas funciones cuentan con parámetros adicionales que permites el ajuste de lr, estos son 'max_perf_inc', 'lr_dec' y 'lr_inc'. Si durante el entrenamiento el nuevo error excede al anterior por más de 'max_perf_inc' (típicamente 1.04) se desechan los nuevos pesos y bias, además la proporción de aprendizaje se disminuye multiplicándola por 'lr_dec' (típicamente 0.7). Si por el contrario, el nuevo error es menor que el anterior, la razón de aprendizaje se incrementa ('lr_inc' frecuentemente = 1.05)

El algoritmo BFGS **'trainbfg'** introduce el método de Newton que es una alternativa a los métodos de gradiente conjugado pues consigue una rápida optimización. Este algoritmo generalmente converge en pocas iteraciones pero requiere más cálculos en cada iteración y más almacenamiento que los métodos de gradiente conjugado.

El algoritmo Levenberg-Marquardt **'trainlm'** es uno de los mejores y más rápidos en cuanto alcanzar la convergencia. Los parámetros de entrenamiento de 'trainlm' son: epochs, show, goal, time, min_grad, max_fail, mu, mu_dec, mu_inc, mu_max, mem_reduc. Mu, mu_dec, mu_inc, mu_max, son parámetros que permiten que la red se adapte al método de Newton de tal forma que la función de rendimiento se reduzca siempre luego de cada iteración del algoritmo; mem_reduc es usado para controlar la cantidad de memoria utilizada por el algoritmo.

A continuación se presenta una tabla que resume los algoritmos de entrenamiento que presenta Matlab:

Función	Descripción
traingd	Gradiente descendente básico. Respuesta lenta, puede ser usado en el modo de entrenamiento incremental.
traingdm	Gradiente descendente con momento. Generalmente más rápido que traingd. Puede ser usado en modo de entrenamiento incremental.
traingdx	Velocidad de entrenamiento adaptivo. Más rápido que traingd, pero solo puede ser usado en modo de entrenamiento por lotes (bach).
trainrp	Retropropagación elástica. Algoritmo simple en modo de entrenamiento incremental con rápida convergencia y mínimo requerimiento de memoria.
traincgf	Algoritmo de gradiente conjugado Fletcher-Reeves. Tienen un requerimiento de memoria mucho menor que otros algoritmos de gradiente descendente.
traincgp	Algoritmo de gradiente conjugado Polak-Ribiére. Requerimiento de memoria ligeramente mayor que traingcf. Convergencia más rápida en algunos problemas.
traincgb	Algoritmo de gradiente conjugado de Powell-Beale. Requerimiento de memoria ligeramente mayor a traincgp. Generalmente de convergencia más rápida.
trainscg	Algoritmo de gradiente conjugado escalado. Es el único algoritmo de gradiente conjugado que no requiere encontrar linealidad.
trainbfg	Método quasi-Newton BFGS. Requiere de memoria adicional para mantener la matriz Hessiana y tiene más trabajo computacional en cada iteración, pero generalmente converge en pocas iteraciones.
trainoss	Método de la secante en un paso. Es un arreglo entre los métodos del gradiente conjugado y de quasi-Newton.
trainlm	Algorimo Lavenberg-Marquardt. Es una algoritmo más rápido para redes de tamaño moderado. Tiene una reducción de memoria apreciable cuando el entrenamiento es grande.
trainbr	Regularización Bayesiana. Es una modificación del algoritmo de entrenamiento Lavenberg-Maquardt. Reduce la dificultad para determinar la arquitectura óptima de la red.

Tabla 2.1 Algoritmos de entrenamiento para redes de retropropagación en Matlab

Para entrenar una red utilizando cualquiera de los algoritmos de entrenamiento, se invoca la función 'train', ésta realiza el entrenamiento de acuerdo a NET.trainFcn y NET.trainParam, se escribe la siguiente instrucción:

[net, tr] = train(NET, P, T)

Donde

P entradas a la red

T objetivo de la red, por defecto = ceros.

net nombre de la red

tr registro de entrenamiento (época y rendimiento)

Ejemplo 7:

Diseñar y entrenar una red de retropropagación con trainlm

Se crea la red:

```
net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'trainlm');
```

Definimos los parámetros de entrenamiento:

```
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

Determinamos los datos de entrada y salidas objetivo:

```
P = [-1 -1 2 2;0 5 0 5];
T = [-1 -1 1 1];
```

Se entrena la red:

```
[net, tr] = train(net, P, T);
TRAINLM, Epoch 0/300, MSE 2.7808/1e-05, Gradient 7.77931/1e-10
TRAINLM, Epoch 4/300, MSE 3.67935e-08/1e-05, Gradient
```

0.000808272/1e-10
TRAINLM, Performance goal met.

```
tr =    epoch: [0 1 2]
        perf: [0.6947 0.0027 1.6470e-006]
        vperf: [NaN NaN NaN]
        tperf: [NaN NaN NaN]
        mu: [0.0010 1.0000e-004 1.0000e-005]
```

Luego se simula la red para comprobar los resultados:

```
a = sim(net,p)
a = -1.0000 -1.0000 1.0000 0.9996
```

Este algoritmo es uno de los métodos de entrenamiento que llegan a converger más rápido con redes de tamaño moderado (varios cientos de pesos).

Para profundizar sobre estos y otros algoritmos consulte en [1].

2.4 IMPLEMENTACIÓN DE UN DETECTOR DE AMPLITUDES

Ahora se procederá a realizar una aplicación práctica de las redes neuronales artificiales.

El objetivo es lograr que una red neuronal pueda advertir cuál es la amplitud de una señal seno, es decir, que al ingresar formas de onda senoidales de diferentes amplitudes ésta entregue una respuesta en forma de escalón de amplitud igual a la de la forma senoidal correspondiente. Esta aplicación ha sido tomada del toolbox de Matlab, appelm1.

Puesto que se requiere un reconocimiento de un modelo temporal y espacial, se trabajará con redes recurrentes, que como se mencionó anteriormente, son redes que incluyen en su arquitectura un lazo de realimentación.

En Matlab, se define dos tipos de redes recurrentes, la de Elman y la Hopfield. Para esta aplicación es adecuado trabajar con la red de Elman.

2.4.1 GENERACION DE ENTRADAS Y SALIDAS

Se puede definir para la entrada dos señales senoidales de diferentes amplitudes, sin embargo, con el fin de que la red sea capaz de generalizar los resultados, se ampliará el número de ejemplos a cinco señales senoidales dentro de un período de tiempo de 1 a 20 segundos.

P1 = sin(1:20); señal senoidal de amplitud 1
P2 = sin(1:20)*4; señal senoidal de amplitud 4
P3 = sin(1:20)*2; señal senoidal de amplitud 2
P4 = sin(1:20)*0.2; señal senoidal de amplitud 0.2

La figura 2.1 muestra la señal completa que será introducida a la red, ésta ha sido combinada en secuencias de manera que cada señal ocurra dos veces. La instrucción para combinar las formas de onda es:

P = [P1 P2 P3 P4 P1 P2 P3 P4];

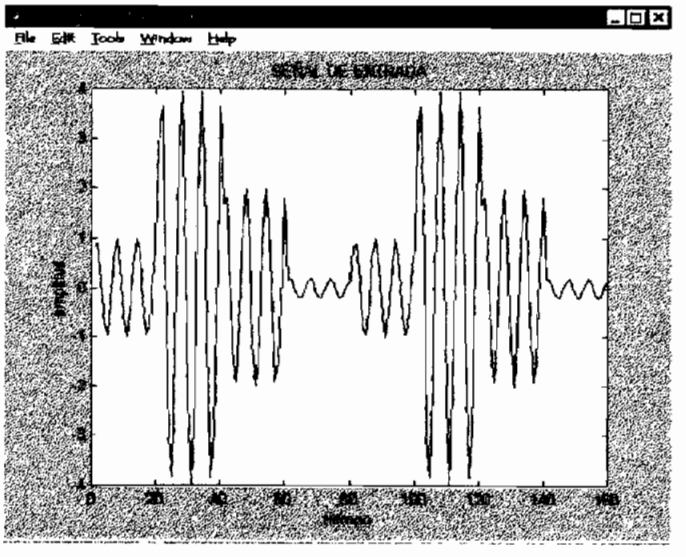


Fig. 2.1 Generación de la señal de entrada a la red

De igual manera, se genera una salida objetivo para cada señal senoidal dependiente de la amplitud:

```
T1 = ones(1,20);
T2 = ones(1,20)*4;
T3 = ones(1,20)*2;
T4 = ones(1,20)*0.2;
```

Asimismo, se combinarán las cinco salidas para obtener el resultado graficado en la figura 2.2:

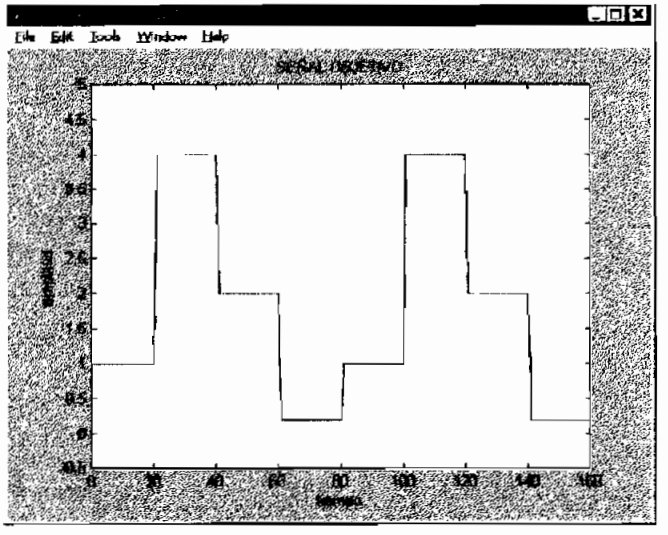


Fig. 2.2 Generación de la salida objetivo

La salida identificará la amplitud de cada señal como se observa en la figura 2.3.

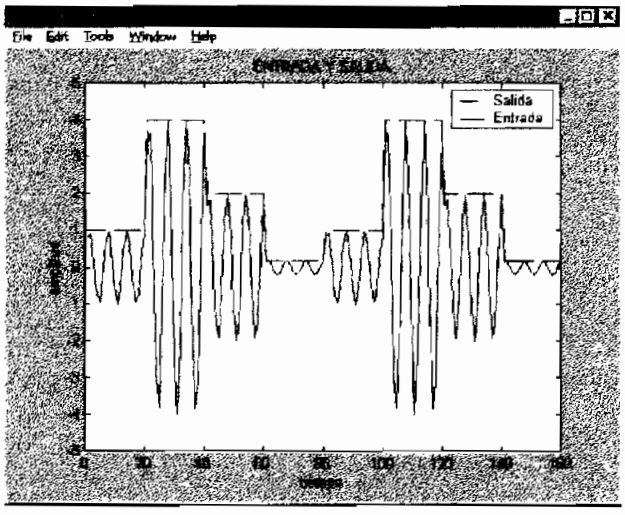


Fig. 2.3 Detección de la amplitud

Para entrenar esta red, la entrada y salida deben ser ingresadas como datos secuenciales, esto se logra con las siguientes instrucciones:

```
Pseq = con2seq(P);
Tseq = con2seq(T);
```

2.4.2 CREACIÓN DE LA RED

La red estará formada por dos capas: la capa recurrente y la salida. La capa queda definida con la siguiente instrucción:

```
dam = newelm([-4,4],[20,1],{'tansig','purelin'},'traingdx');
```

Así, la red de nombre dam (**d**etección de **am**plitud) posee en la primera capa veinte neuronas y en la de salida una sola. Las funciones de activación son tansig y purelin respectivamente, y la función de entrenamiento en traingdx. El rango para los elementos de la entrada va de -4 a 4.

2.4.3 ENTRENAMIENTO

Antes de entrenar la red se debe establecer los parámetros más importantes, estos son:

dam.trainparam.epochs=1000;	número de iteraciones
dam.trainparam.goal=1e-5;	error meta
dam.trainparam.show=50;	número de iteraciones luego de las cuales se mostrarán los resultados

Se inicia el entrenamiento:

```
[dam,tr]=train(dam,Pseq,Tseq);

TRAINGDX, Epoch 0/1000, MSE 0.0118639/1e-006, Gradient 0.0128786/1e-006
TRAINGDX, Epoch 50/1000, MSE 0.0116513/1e-006, Gradient 0.0117428/1e-006
TRAINGDX, Epoch 100/1000, MSE 0.0100131/1e-006, Gradient 0.1523/1e-006
TRAINGDX, Epoch 150/1000, MSE 0.00955025/1e-006, Gradient 0.0174674/1e-006
TRAINGDX, Epoch 200/1000, MSE 0.00887748/1e-006, Gradient 0.0102632/1e-006
TRAINGDX, Epoch 250/1000, MSE 0.00858379/1e-006, Gradient 0.0575988/1e-006
TRAINGDX, Epoch 300/1000, MSE 0.00838669/1e-006, Gradient 0.0104617/1e-006
TRAINGDX, Epoch 350/1000, MSE 0.00809214/1e-006, Gradient 0.03719/1e-006
TRAINGDX, Epoch 400/1000, MSE 0.00798235/1e-006, Gradient 0.0109827/1e-006
TRAINGDX, Epoch 450/1000, MSE 0.0076795/1e-006, Gradient 0.0801661/1e-006
TRAINGDX, Epoch 500/1000, MSE 0.00752887/1e-006, Gradient 0.0186203/1e-006
TRAINGDX, Epoch 550/1000, MSE 0.00721417/1e-006, Gradient 0.0517116/1e-006
TRAINGDX, Epoch 600/1000, MSE 0.00715524/1e-006, Gradient 0.0205324/1e-006
TRAINGDX, Epoch 650/1000, MSE 0.00704423/1e-006, Gradient 0.00988316/1e-006
TRAINGDX, Epoch 700/1000, MSE 0.0070069/1e-006, Gradient 0.0535488/1e-006
TRAINGDX, Epoch 750/1000, MSE 0.0069256/1e-006, Gradient 0.00955892/1e-006
TRAINGDX, Epoch 800/1000, MSE 0.00695366/1e-006, Gradient 0.104693/1e-006
TRAINGDX, Epoch 850/1000, MSE 0.00681698/1e-006, Gradient 0.0110775/1e-006
TRAINGDX, Epoch 900/1000, MSE 0.00680545/1e-006, Gradient 0.0704938/1e-006
TRAINGDX, Epoch 950/1000, MSE 0.00674987/1e-006, Gradient 0.0177513/1e-006
```

TRAINGDX, Epoch 1000/1000, MSE 0.00674595/1e-006, Gradient 0.00899046/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.

La red termina el entrenamiento antes de alcanzar el rendimiento deseado porque llegó al límite del número de iteraciones.

En realidad, se realizó tres entrenamientos, es decir, 3000 iteraciones luego de lo cual se obtuvo un error de 0.0005. Gráficamente, los cambios del error se observan en la figura 2.4.

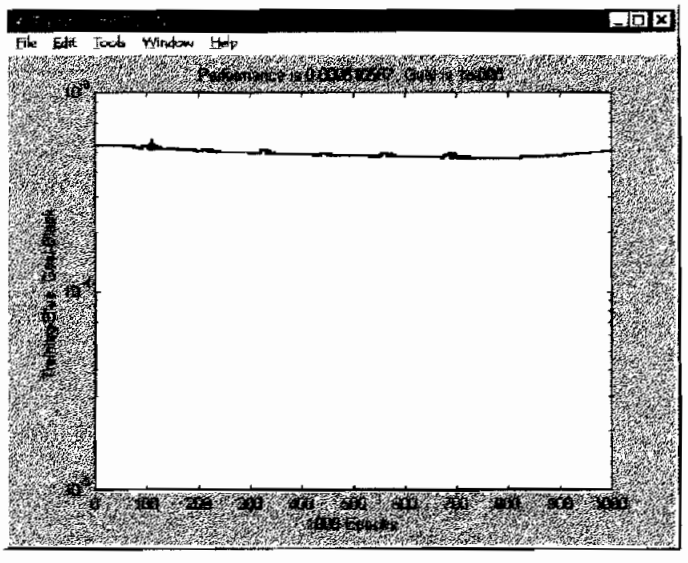


Fig. 2.4 Rendimiento vs iteraciones

A continuación se comprobará si con el error obtenido se alcanza un comportamiento satisfactorio.

4.2.4 SIMULACIÓN

La simulación se la realiza utilizando la instrucción sim:

```
Y = sim(dam,Pseq);
```

Puesto que los vectores de entrada y salida son secuenciales, no es posible graficarlos directamente con la instrucción *plot*, entonces, la instrucción es plot(time,T,'--',time,cat(2,Y{:})), con ella se grafica la salida simulada y la salida deseada a la vez de manera que se pueda comparar los resultados. La figura 2.4 expone los resultados:

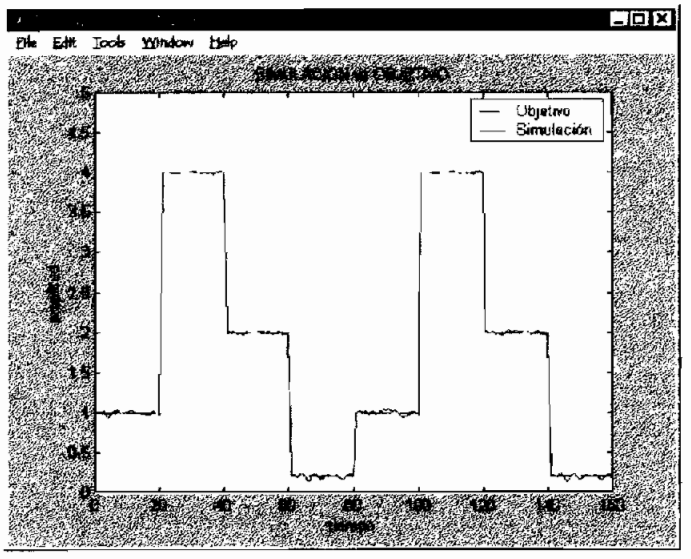


Fig. 2.4 Resultado de la simulación

Se puede apreciar que el resultado es bueno sin embargo no es el mejor. El siguiente paso será comprobar si la red es capaz de generalizar.

4.2.5 COMPROBANDO EL COMPORTAMIENTO DE LA RED

Para verificar si en efecto la red puede generalizar, se comprobará con dos nuevas señales senoidales.

La señal de entrada está compuesta por dos señales senoidales de magnitudes 3.5 y 0.5.

```
P5 = sin(1:20)*3.5;
P6 = sin(1:20)*0.5;
```

```
P = [P5 P6 P5 P6];
Ps = con2seq(P);
```

Mientras que la señal de salida èsta deberá ser:

```
T5 = ones(1,20)*3.5;
T6 = ones(1,20)*0.5;
T = [T5 T6 T5 T6];
```

Esta vez, se generará la salida únicamente para comparar con la simulación.

Entonces, simulando Y=sim(dam,Ps) se tiene:

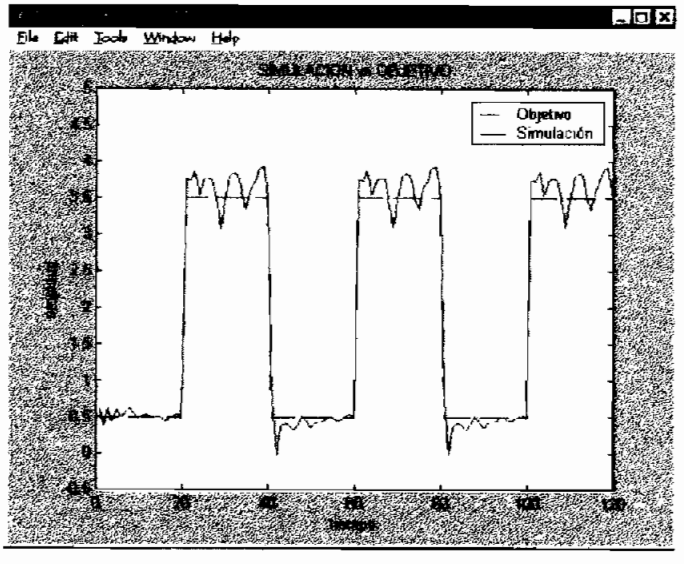


Fig. 2.5 Detección de la amplitud de una nueva señal

El resultado no es perfecto, sin embargo se tiene una muy buena aproximación.

Entrenar a la red con un mayor número de neuronas pudiera permitir un mejor resultado.

3. TALLER

Para los siguientes ejercicios utilice el paquete computacional MATLAB.

1. Compruebe que el perceptrón no es capaz de simular el comportamiento de la compuerta XOR.

2. Diseñe y entrene una red que se capaz de resolver el literal anterior.

3. Diseñe y entrene una red con un número mayor de neuronas. Varíe además el algoritmo de entrenamiento, de tal forma que cumpla eficazmente el objetivo de detectar amplitudes.

4. Diseñe una red de retropropagación para que detecte la amplitud de señales. ¿Logra la red cumplir el objetivo?. Compare los resultados con los de la red de Elman.

Utilizando el programa tutorial SINTEL resuelva el siguiente numeral.

5. Cree y entrene una red que le permita detectar la amplitud de las señales. Diséñela probando con varias conexiones, números de capas y de neuronas por capa. Compare resultados con los ejercicios de los numerales 3 y 4.

4. PROYECTO

Diseñe y entrene una red que sea capaz de actuar como un integrador.

REFERENCIAS BIBLIOGRÁFICAS

1. Demuth Howard, Beale Mark, **" Neural Network Toolbox, Matlab"**, The MathWorks, Inc.

2. Sánchez José, **"Identificación, control y simulación de un sistema usando redes neurales"**, Tesis, Escuela Politécnica Nacional

3. Becerra Isaac, **"Simulación de la máquina de inducción utilizando redes neuronales"**, Proyecto de titulación, Escuela Politécnica Nacional

CONJUNTOS DIFUSOS

1. OBJETIVOS

- Familiarizar al estudiante con los conceptos básicos de conjuntos difusos.
- Manejar las operaciones y relaciones complejas entre conjuntos difusos.

2. MARCO TEÓRICO

2.1 INTRODUCCIÓN

Incertidumbre es la condición en la cual la posibilidad de error existe, pues no hay seguridad.

Para el planteamiento de problemas de ingeniería, usualmente se evitaba la aparición de incertidumbre, los planteamientos caían dentro de un marco estrictamente matemático, la exactitud era importante y no se concibe la modelación de un sistema fuera de este planteamiento.

Lofti Zadeh publica en 1965 una investigación, donde introduce el concepto de lógica difusa, pues su interés era representar problemas de sistemas complejos utilizando un modelo simple. Hasta al momento, la herramienta matemática había resultado pobre y no proveía una solución simple.

Mientras en la ciencia se evitaba trabajar bajo incertidumbre, en la cotidianeidad es ineludible. Los seres humanos expresamos muchas situaciones diarias por medio de expresiones vagas pero con un alto contenido significativo, tal como aquellas que expresan temperatura, por ejemplo, "caliente, frío" o "casi lleno, casi vacío" para indicar la cantidad de líquido que contiene un tanque. Con este tipo de expresiones, podría ser más sencillo describir las cualidades de un sistema sin necesidad de recurrir a un modelo matemático preciso. Aquí radica precisamente la importancia de la lógica difusa, pues se construye sobre la

estructura de descripción cualitativa utilizadas en el idioma cotidiano, por esto es fácil usar.

Se puede apreciar la necesidad de introducir la lógica difusa aún con un ejemplo simple. Suponga que se desea definir un conjunto que indique estaturas de personas adultas altas. Alguien podría especificar un rango de 1.80 m hacia arriba, pero, ¿puede una persona de 1.79 m de altura ser considerada de baja estatura?. Esta disyuntiva lleva a la conclusión de que estos elementos no pueden ser representados por conjuntos de límites precisos como los conjuntos clásicos, es entonces imperiosa una nueva teoría que facilite la representación del nuevo conocimiento impreciso, difuso.

2.1.1 Ventajas de usar sistemas difusos

- **Es conceptualmente fácil de entender.**

Los conceptos matemáticos detrás del razonamiento difuso son muy simples.

- **Es tolerante ante datos imprecisos.**

Todo a nuestro alrededor es impreciso, y la mayoría las cosas incluso son imprecisas en la inspección cuidadosa. El razonamiento difuso se construye cualitativamente entendiendo el proceso en lugar de cambiar el enfoque hacia el extremo de la precisión.

- **Permite modelar funciones no lineales de complejidad arbitraria.**

Se puede crear un sistema difuso para emparejar cualquier juego de datos de entrada-salida.

- **Puede construirse sobre la experiencia de expertos.**

En el contraste directo a redes neuronales que toman los datos de entrenamiento y generan los modelos opacos, impenetrables, la lógica difusa permite contar con la experiencia de las personas que entienden su sistema.

• **Puede usarse con técnicas de mando convencionales.**

Los sistemas difusos no reemplazan necesariamente los métodos del mando convencionales. En muchos casos los sistemas difusos simplifican su aplicación.

2.2 CONJUNTOS DIFUSOS

El universo de discurso U contiene todos los elementos que pueden entrar en consideración para formar parte de un conjunto. El universo depende del contexto, por ejemplo, se tendrá un universo de números positivos entre 0 y 10 [m] si lo que se representa es el nivel de un tanque. Cuando un conjunto no puede ser medido con escala numérica se dice entonces que los elementos son tomados de un continuo psicológico, por ejemplo escala de sabores {amargo, dulce, salado, agrio, picante}.

La función que une cada elemento del universo de discurso con un número o grado de pertenencia se denominan *función de membresía* $\mu_F(x)$. Cada elemento del universo de discurso está asociado a un grado de membresía en mayor o menor grado, incluso el cero. El rango de asignación es [0,1]. Se designará con un grado de membresía de 1 a los elementos que mejor se adapten al concepto del conjunto, 0 a los que no pertenezcan en absoluto y los valores intermedios dependientes del criterio del experto.

Un *conjunto difuso* es un conjunto sin límites claramente definidos, y puede tener elementos que le pertenecen solo parcialmente según la asignación de la función de membresía.

Los conjuntos difusos son representados por su función de membresía en forma gráfica, tabular o analítica.

La representación gráfica es adecuada cuando el conjunto está definido en el espacio Euclidiano de una o dos dimensiones. Las figuras más empleadas son: triangular, trapezoidal, gaussiana y sigmoide, las mismas que se indican en la figura 2.1:

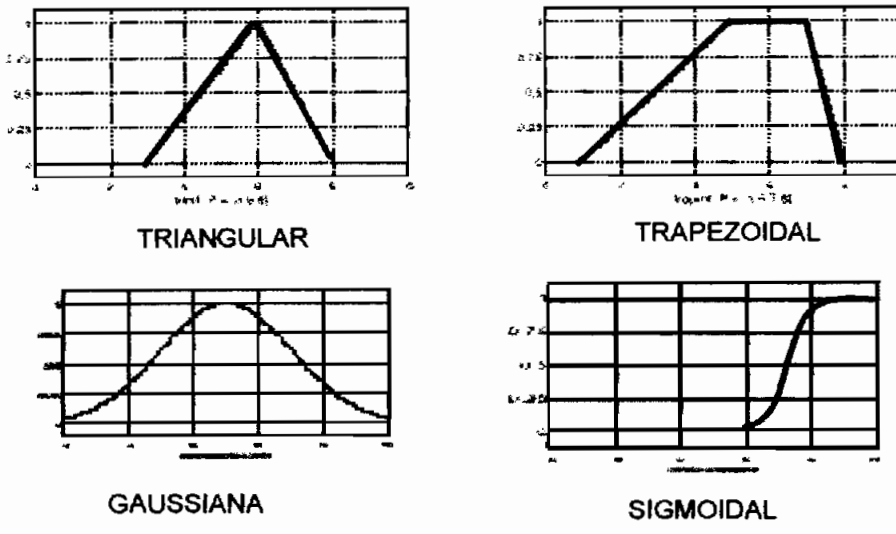


Fig. 2.1 Representación gráfica de las funciones de membresía

Una de las gráficas muy útiles es la singleton (figura 2.2), que representa al conjunto difuso con un único punto de pertenencia al universo U con función de membresía $\mu_F(x) = 1$.

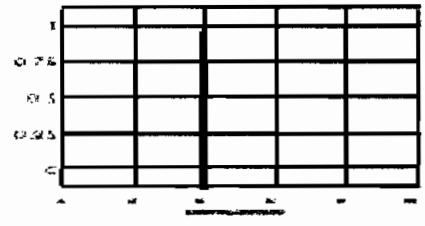


Fig. 2.2 Función de membresía singleton

El conjunto difuso F puede ser denotado como un conjunto de pares ordenados de elementos *x* y la función de membresía $\mu_F(x)$, esta forma de representación es conocida como representación tabular o de lista:

$$F = \{x, \mu_F(x)/x \in U\}$$

Los elementos x pueden ser representados, siempre y cuando U sea un universo continuo, de la siguiente forma:

$$F = \int_U \mu_F(x)/x$$

Donde el signo de integración representa el conjunto de todos los puntos que pertenecen al conjunto universo asociados a la función $\mu_F(x)$

Por el contrario, si el universo U es discreto, se lo simboliza:

$$F = \sum_U \mu_F(x)/x$$

Nuevamente, el signo de suma no implica a dicha operación aritmética, sino el conjunto de puntos que pertenecen al universo de discurso U con su función de membresía asociada $\mu_F(x)$

Ejemplo 1:

Sea el conjunto F, cuyos elementos son los números enteros cercanos a 10.

$$F = 0.1/7 + 0.5/8 + 0.8/9 + 1/10 + 0.8/11 + 0.5/12 + 0.1/13$$

Se puede notar tres puntos importantes:

1. Los números enteros que no han sido escritos tienen función de membresía 0 (cero).
2. Todos los valores de la función de membresía excepto el que pertenece a 10, son escogidos de acuerdo al criterio del experto basado en lo que se entiende por "cercano".
3. La función de membresía es simétrica alrededor del 10, pues existen número enteros cercanos tanto por la derecha como por la izquierda, sin embargo, esto no es norma como se verá más adelante.

Ejemplo 2:

En la tabla 2.1, mediante función de membresía se representa tabularmente el término lingüístico "joven".

Etiqueta	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	X_{11}
Edades (años)	0	5	10	15	20	25	30	35	40	45	50
Grado de membresía	1	1	1	1	1	1	0,8	0,6	0,4	0,2	0,1

Tabla 2.1 Representación tabular de una función de membresía

O también:

J(x) = 1/0+ 1/5+1/10+ 1/15+ 1/20+ 1/25+ 0.8/30+ 0.6/35+ 0.4/40+ 0.2/45+ 0.1/50

La representación gráfica mediante la función singleton se indica en la figura 2.3:

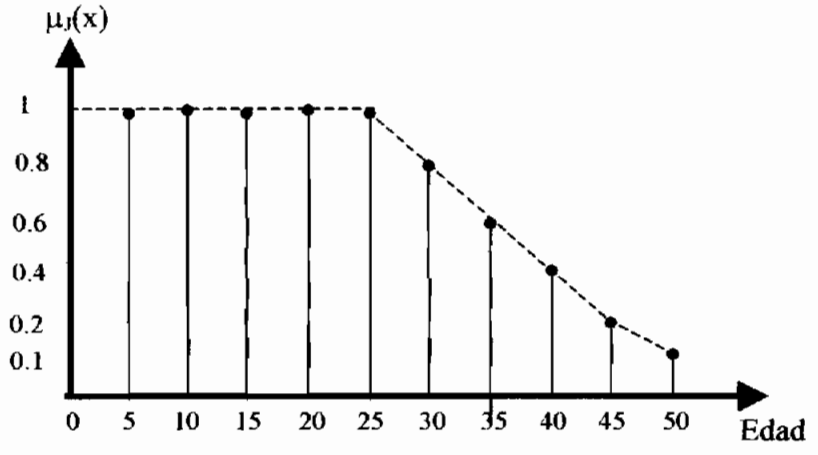


Fig. 2.3 Representación gráfica del término lingüístico "joven"

Cuando el universo en el que está definido el conjunto difuso es muy extenso o se trata de conjuntos continuos y no hay forma de describirlo tabularmente, es mejor definirlo en forma analítica, esto es, representar mediante ecuaciones la membresía del conjunto.

Así, la representación triangular de una función de membresía puede ser definida en general:

$$\mu_F(x) = \begin{cases} b\left(1 - \dfrac{|x-a|}{s}\right) & \text{cuando } a\text{-}s \le x \le a+s \\ 0 & \text{en otro caso} \end{cases}$$

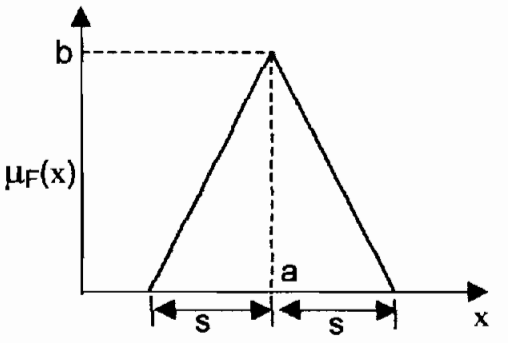


Fig. 2.4 Función de membresía triangular

Otra clase de función de membresía es la trapezoidal, cuya representación analítica es la siguiente:

$$\mu_F(x) = \begin{cases} \dfrac{(a-x)\cdot e}{a-b} & \text{cuando } a \le x \le b \\ e & \text{cuando } b \le x \le c \\ \dfrac{(d-x)\cdot e}{d-c} & \text{cuando } c \le x \le d \\ 0 & \text{en otro caso} \end{cases}$$

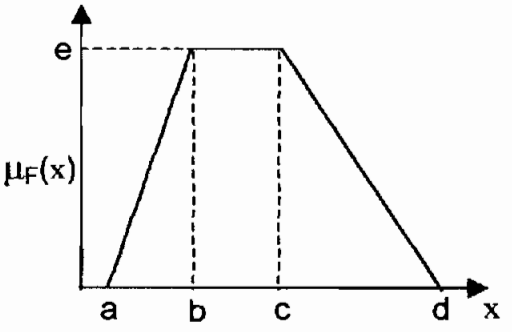


Fig. 2.5 Función de membresía trapezoidal

La campana gaussiana es también muy utilizada, y se representa por la fórmula:

$$\mu_F(x) = c.e.^{\frac{(x-a)^2}{b}}$$

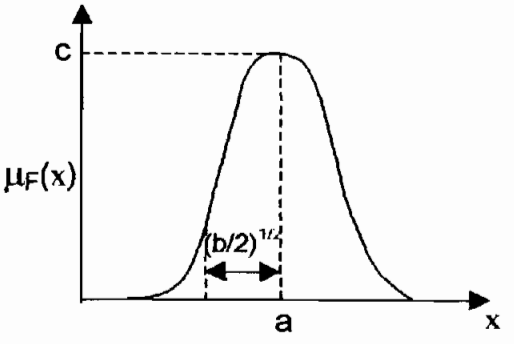


Fig. 2.6 Función de membresía campana

Ejemplo 3:

Usando funciones de membresía triangulares y trapezoidales, se representará los conjuntos difusos pertenecientes a la variable lingüística "temperatura" en la figura 2.7:

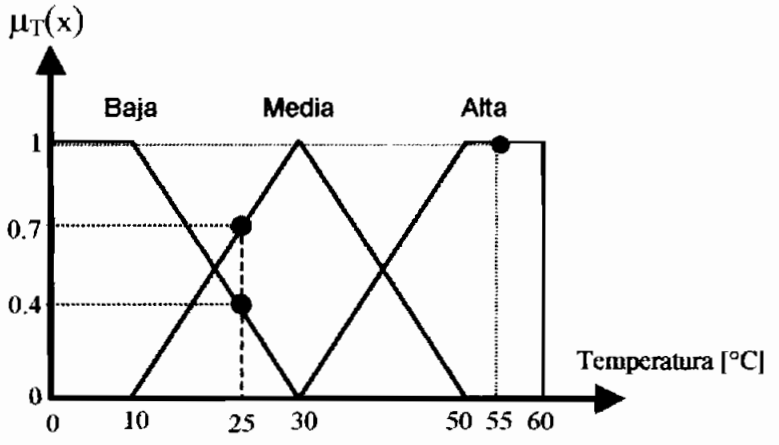


Fig. 2.7 Conjuntos difusos representados por funciones triangulares

Se puede observar que, x = 55 tiene un grado de membresía 1 en el conjunto difuso *Alta*. Las funciones de membresía pueden ser superpuestas. Esto se debe que en la teoría de conjuntos difusos, un elemento puede pertenecer a dos conjuntos a la vez, algo que es imposible en los conjuntos clásicos. Así, si trazamos una línea vertical para un valor x = 25 (una temperatura de 25°C), se puede observar que corta dos funciones de membresía (baja y media) asignándosele dos grados de membresía diferentes: $\mu_{Media}(x) = 0.7$ y $\mu_{Baja}(x) = 0.4$

Utilizando la figura 2.7 se realizará la representación analítica mediante ecuaciones de las funciones de membresía del término lingüístico "temperatura".

Para la función de membresía media:

$$\mu_{Media}(x) = \begin{cases} 1 \cdot \left(1 - \dfrac{|x-30|}{20}\right) & \text{cuando } 10 \le x \le 50 \\ 0 & \text{en otro caso} \end{cases}$$

Para la función de membresía baja:

$$\mu_{Baja}(x) = \begin{cases} 1 & \text{cuando } 0 \le x \le 10 \\ \dfrac{(30-x)\cdot 1}{20} & \text{cuando } 10 \le x \le 30 \\ 0 & \text{en otro caso} \end{cases}$$

Para la función de membresía alta:

$$\mu_{Alta}(x) = \begin{cases} \dfrac{(30-x)\cdot 1}{-20} & \text{cuando } 10 \le x \le 30 \\ 1 & \text{cuando } 50 \le x \le 60 \\ 0 & \text{en otro caso} \end{cases}$$

Una *variable lingüística* usa palabras u oraciones como valores en lugar de números como lo hacen las variables algebraicas. La razón por la cual se hace necesario el uso de expresiones ya sea en un lenguaje natural o artificial, es porque son menos específicas que un número (Zadeh).

Cada variable lingüística *u* puede ser usualmente descompuesta en varios términos que vienen a ser conjuntos difusos *T(u)* los cuales cubren el universo de discurso *U*. Los valores numéricos de la variable lingüística *u* son denotados x tal que $x \in U$.

Ejemplo 4:

Sea la variable lingüística temperatura (*u*), cuyos términos pueden ser los siguientes:

T(u) = {muy frío, frío, tibio, caliente, muy caliente}

Siempre dentro del universo de discurso U =[-10°C, 90°C], se puede considerar como *muy frío* los valores cercanos a 0°C, para *frío* aquellos que están alrededor de 20°C, como *tibio* los cercanos a 40°C, para *caliente* los valores cercanos a 60°C y como muy caliente aquellos que están alrededor de 80°C.

2.3 OPERACIONES CON CONJUNTOS DIFUSOS

2.3.1 Operaciones básicas con conjuntos difusos

Las operaciones entre conjuntos difusos crea un nuevo conjunto con una función de membresía propia.

2.3.1.1 Unión de conjuntos

Sean dos conjuntos difusos A y B definidos en el universo U. La unión se define como:

$$A \cup B = \max[\mu_A(x), \mu_B(x)]$$

La operación **max** es una comparación uno a uno entre los correspondientes ítems, de los cuales se escoge el mayor. En el caso de los conjuntos difusos, los ítems corresponden a las funciones de membresía.

2.3.1.2 Intersección de conjuntos

Sean dos conjuntos difusos A y B definidos en el universo U. La intersección se define como:

$$A \cap B = \min[\mu_A(x), \mu_B(x)]$$

La operación **min** es una comparación uno a uno entre los correspondientes ítems de los cuales se escoge el menor. En el caso de los conjuntos difusos, los ítems corresponden a las funciones de membresía.

Otra forma de realizar la operación de intersección es mediante una operación producto, que se definiría:

$$A \cap B = [\mu_A(x).\mu_B(x)]$$

cuyo resultado se obtiene de multiplicar los elementos de cada conjunto correspondientemente.

Ejemplo 5:

(Zimmermann, 1993): Una familia de cuatro personas quiere comprar una casa. Una indicación de cuan confortable la desean es el número de habitaciones en la casa, pero también quieren una casa grande.

Sea U= [1,2,3,4,5,6,7,8,9,10] el conjunto de casas disponibles descritas por su número de habitaciones.

El conjunto difuso confortable C puede ser descrito mediante funciones de membresía:

$\mu(C)$ = [0.2 0.5 0.8 1 0.7 0.3 0 0 0 0]

Sea $\mu(G)$ que identifica al conjunto difuso Grande G:

$\mu(G)$ = [0 0 0.2 0.4 0.6 0.8 1 1 1 1]

La unión de confortable y grande $\mu_{C \cup G}$ es:

$\mu_{C \cup G}$ = [0.2 0.5 0.8 1 0.7 0.8 1 1 1 1]

Una casa de cuatro habitaciones es completamente satisfactoria, porque es confortable, y de siete a diez lo es también porque significa que es grande.

La intersección de los conjuntos C y G es:

$\mu_{C \cap G}$ = [0 0 0.2 0.4 0.6 0.3 0 0 0 0]

o por la intersección usando el producto:

$\mu_{C \cap G}$ = [0 0 0.16 0.4 0.42 0.24 0 0 0 0]

Para interpretar esto, cinco dormitorios son óptimos, pero sólo es satisfactorio para el grado 0.6. La segunda mejor solución es la casa de cuatro dormitorios.

2.3.1.3 Complemento

El complemento de A es

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

Así, cada valor miembro de A es substraído de 1.

Ejemplo 6:

Del conjunto difuso C (Confortable), el complemento es:

$\mu_{\overline{C}}$ = [0.8 0.5 0.2 0 0.3 0.7 1 1 1 1]

Es importante notar que en la teoría de conjuntos difusos no se cumplen algunas propiedades de los conjuntos claros tales como:

$A \cup A \neq U$ ley de contradicción

$A \cap A \neq \varnothing$ ley del medio excluido

Pero hay otras propiedades que si son aplicables (lo que se cumple para la unión se cumple también para la intersección):

$A \cup B = B \cup A$	Conmutativa
$(A \cup B) \cup C = A \cup (B \cup C)$	Asociativa
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributiva
$\overline{A \cup B} = \overline{A} \cap \overline{B}$	DeMorgan
$(A \cup B) \cap A = A$	Absorción
$A \cup A = A$	Idempotencia

2.3.1.4 Norma t y norma s

La co-norma t conocida también como *norma s* y la *norma t* son los operadores que se utilizan para realizar las operaciones de unión e intersección.

La **norma s** es un operador utilizado para la unión de conjuntos difusos y se simboliza con $\oplus$.

La unión de dos conjuntos difusos es frecuentemente formada usando uno de las siguientes operaciones de la norma s:

$x \oplus y = x + y - xy$ suma algebraica

$x \oplus y = \max(x, y)$ máximo

$x \oplus y = \min(1, x+y)$ suma límite

$x \oplus y = x$ si $y = 0$

$x \oplus y = y$ si $x = 0$ } suma drástica

$x \oplus y = 1$ si $x, y > 0$

La **norma t**, representado por el símbolo * es un operador utilizado en la operación de intersección.

La intersección de dos conjuntos es frecuentemente formado usando una de las siguientes operaciones dadas por la norma t

$x * y = xy$ producto algebraico

$x * y = \min(x, y)$ mínimo

$x * y = \max(0, x+y-1)$ producto límite

$x * y = x$ si $y = 1$

$x * y = y$ si $x = 1$ } producto drástico

$x * y = 0$ si $x, y < 1$

Además, pares de normas s y t satisfacen las siguientes generalizaciones de la ley de DeMorgan (Zimmerman):

$s[\mu_A(x), \mu_B(x)] = c\{t[c(\mu_A(x)),c(\mu_B(x))]\}$, $x \in U$

$t[\mu_A(x), \mu_B(x)] = c\{s[c(\mu_A(x)),c(\mu_B(x))]\}$, $x \in U$

ya que c indica complemento, lo anterior es igual a:

$\max[\mu_A(x), \mu_B(x)] = 1-\max[1-\mu_A(x),1-\mu_B(x)]$

$\min[\mu_A(x), \mu_B(x)] = 1-\min[1-\mu_A(x),1-\mu_B(x)]$

2.3.2 Operaciones Complejas con Conjuntos Difusos

2.3.2.1 Cortes α de conjuntos difusos

A pesar de las importantes diferencias entre los conjuntos difusos y los clásicos (crisp), existe una forma de representar un conjunto difuso en base conjuntos clásicos.

"Un conjunto difuso dado en U, está siempre asociado a una familia de subconjuntos crisp en U. Cada uno de estos subconjuntos están formados por todos los elementos de U cuyos grados de membresía en el conjunto difuso están restringidos para algunos subconjuntos crisp dados entre [0,1]. Esta es una restricción de los grados de membresía que son *mayores que* o *iguales a* algún valor escogido α entre [0,1]. Cuando esta restricción es aplicada a un conjunto difuso A se obtiene un subconjunto crisp ${}^{\alpha}A$ del conjunto universal U, el cual es llamado un corte α de A". [2]

$$ {}^{\alpha}A = \{x \in U \mid \mu_A(x) \geq \alpha\} \text{ para } \alpha \text{ de } [0,1] $$

El corte α de un conjunto difuso A es el conjunto crisp ${}^{\alpha}A$ que contiene todos los elementos del conjunto universo U cuyos grados de membresía en A son *mayores que* o *iguales a* el valor especificado por α.

Ejemplo 7:

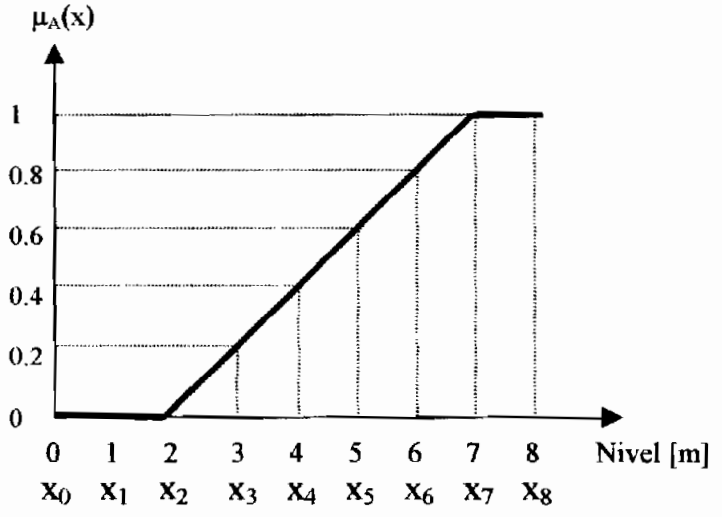


Fig. 2.8 Función de membresía del conjunto difuso nivel

En la figura 2.8 se presenta el conjunto difuso Nivel A y su respectiva función de membresía $\mu_A(x)$ que especificaremos como discreta definida en el conjunto universo U.

$$A = 0/0 + 0/1 + 0/2 + 0.2/3 + 0.4/4 + 0.6/5 + 0.8/6 + 1/7 + 1/8$$

$$A = 0/x_0 + 0/x_1 + 0/x_2 + 0.2/x_3 + 0.4/x_4 + 0.6/x_5 + 0.8/x_6 + 1/x_7 + 1/x_8$$

Escogeremos restricciones para diferentes valores de α y formaremos así los respectivos cortes α.

$\alpha = 0$:

$${}^{0}A = \{0,\ldots, 8\} = 1/x_0 + 1/x_1 + 1/x_2 + 1/x_3 + 1/x_4 + 1/x_5 + 1/x_6 + 1/x_7 + 1/x_8$$

$\alpha = 0.2$:

$${}^{0.2}A = \{3,\ldots, 8\} = 0/x_0 + 0/x_1 + 0/x_2 + 1/x_3 + 1/x_4 + 1/x_5 + 1/x_6 + 1/x_7 + 1/x_8$$

$\alpha = 0.4$:

$${}^{0.4}A = \{4,\ldots, 8\} = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 1/x_4 + 1/x_5 + 1/x_6 + 1/x_7 + 1/x_8$$

$\alpha = 0.6$:

$${}^{0.6}A = \{5,\ldots, 8\} = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 1/x_5 + 1/x_6 + 1/x_7 + 1/x_8$$

$\alpha = 0.8$:

$^{0.8}A = \{6,..., 8\} = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 0/x_5 + 1/x_6 + 1/x_7 + 1/x_8$

$\alpha = 1$:

$^{1}A = \{7,..., 8\} = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 0/x_5 + 0/x_6 + 1/x_7 + 1/x_8$

Los conjuntos así formados son conjuntos crisp o clásicos pues los grados de pertenencia se limitan estrictamente a 0 o 1.

Ahora, convertiremos los cortes α a conjuntos especiales difusos $_{\alpha}A$ aplicando la siguiente fórmula a cada uno de ellos:

$$_{\alpha}A = \alpha.{}^{\alpha}A$$

$_{0.2}A = 0.2.\ ^{0.2}A = 0/x_0 + 0/x_1 + 0/x_2 + 0.2/x_3 + 0.2/x_4 + 0.2/x_5 + 0.2/x_6 + 0.2/x_7 + 0.2/x_8$

$_{0.4}A = 0.4.\ ^{0.4}A = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0.4/x_4 + 0.4/x_5 + 0.4/x_6 + 0.4/x_7 + 0.4/x_8$

$_{0.6}A = 0.6.\ ^{0.6}A = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 0.6/x_5 + 0.6/x_6 + 0.6/x_7 + 0.6/x_8$

$_{0.8}A = 0.8.\ ^{0.8}A = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 0/x_5 + 0.8/x_6 + 0.8/x_7 + 0.8/x_8$

$_{1}A = 1.\ ^{1}A = 0/x_0 + 0/x_1 + 0/x_2 + 0/x_3 + 0/x_4 + 0/x_5 + 0/x_6 + 1/x_7 + 1/x_8$

Cualquier conjunto difuso puede ser completamente caracterizado por los cortes α al realizar la unión de los conjuntos especiales difusos tenemos el conjunto difuso original A:

$$A = {}_{0.2}A \cup {}_{0.4}A \cup {}_{0.6}A \cup {}_{0.8}A \cup {}_{1}A$$

$$A = 0/x_0 + 0/x_1 + 0/x_2 + 0.2/x_3 + 0.4/x_4 + 0.6/x_5 + 0.8/x_6 + 1/x_7 + 1/x_8$$

Dicho de otra forma, el conjunto difuso original es una combinación lineal de los cortes α

2.3.2.2 Principio de extensión

Es una forma de fusificar una función crisp.

Ejemplo 8:

Está establecido que el ritmo cardiaco máximo es función de la edad, la cual está dada en forma discreta en la tabla 2.2

Edad	20	22	24	26	28	30	32	34	36	38	40	45	50	55	60	65	x
Máx. Ritmo Cardiaco	200	198	196	194	192	190	189	187	187	184	182	179	175	171	160	150	y

Tabla 2.2 Ritmo cardiaco en función de la edad

Defina, dentro del rango de edades dado, un conjunto difuso razonable para capturar el concepto de *joven*. Determine entonces, usando el principio de extensión el correspondiente conjunto difuso que caracterice el máximo ritmo del corazón en una persona joven.

Tanto x como y son conjuntos crisp que se relacionan por la función crisp $y = f(x)$. El primer paso será definir el conjunto difuso *joven* adecuado, $J = \mu_J / x$ cuya función de membresía se observa en la figura 2.8:

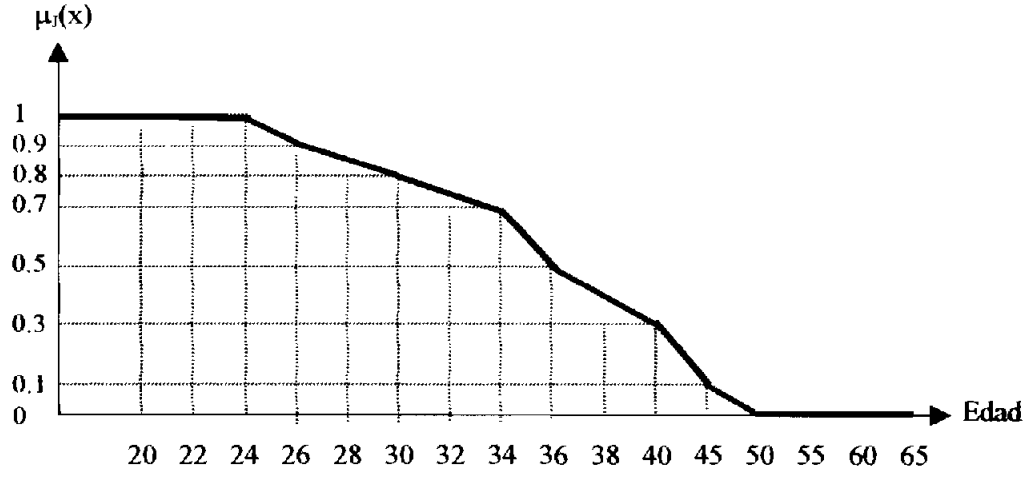


Fig. 2.8 Función de membresía del conjunto joven

J= 1/20 + 1/22 + 1/24 + 0.9/26 + 0.8/28 + 0.8/30 + 0.7/32 + 0.7/34 + 0.5/36 + 0.3/38 + 0.3/40 + 0.1/45 + 0/50 + 0/55 + 0/60 + 0/65

Ahora, el principio de extensión consiste en crear un junto B, dependiente de A por una función f la misma que, para cada valor de x en U asigna un valor particular $y = f(x)$ de acuerdo a la tabla 2.2. Podemos expresarlo así:

$$B = \mu_A(x)/f(x)$$

Entonces, aplicándolo tenemos:

B = 1/f(20) + 1/f(22) + 1/f(24) + 0.9/f(26) + 0.8/f(28) + 0.8/f(30) + 0.7/f(32) + 0.7/f(34) + 0.5/f(36) + 0.3/f(38) + 0.3/f(40) + 0.1/f(45) + 0/f(50) + 0/f(55) + 0/f(60) + 0/f(65)

B= 1/200 + 1/198 + 1/196 + 0.9/194 + 0.8/192 + 0.8/190 + 0.7/189 + 0.7/187 + 0.5/187 + 0.3/184 + 0.3/182 + 0.1/179 + 0/175 + 0/171 + 0/160 + 0/150

Observe que el ritmo cardiaco de 187 está asociado con más de una edad (34 y 36), y se ha asignado dos grados de membresía diferentes (0.7 y 0.5). Debido a esta disyunción, se ha tomado el elemento que tiene el mayor grado de membresía, para nuestro ejemplo 0.7.

Finalmente tenemos el conjunto difuso que captura el concepto de máximo ritmo cardiaco en una persona joven, dado por:

B= 1/200 + 1/198 + 1/196 + 0.9/194 + 0.8/192 + 0.8/190 + 0.7/189 + 0.7/187 + 0.3/184 + 0.3/182 + 0.1/179 + 0/175 + 0/171 + 0/160 + 0/150

Este principio está limitado a conjuntos universales finitos, y dice:

Principio de extensión: Sea f: U→ V, donde U y V denotan conjuntos crisp finitos, una función dada. Dos funciones pueden ser inducidas a partir de f. Uno, denotado por $\bar{f}$, es una función desde $\mathcal{F}$(U) a $\mathcal{F}$(V). La otra, denotada por $\bar{f}^{-1}$, es una función desde $\mathcal{F}$(V) a $\mathcal{F}$(U). Estas funciones están definidas por:

$$[\bar{f}(A)](y) = \max_{x|y=f(x)} A(x)$$

Para cualquier A ∈ $\mathcal{F}$(U), y ∈ V, y

$$[\bar{f}^{-1}(B)](x) = B(f(x))$$

Para cualquier B ∈ $\mathcal{F}$(V), x ∈ U.

2.4 RELACIONES DIFUSAS

La relación de conjuntos difusos describe el grado de presencia o ausencia de asociación, interacción o conexión entre elementos de dos o más conjuntos.

Dicha relación está definida sobre los universos de discursos que forman un producto cartesiano.

Sean U y V dos universos, la relación difusa R(U, V) genera un conjunto difuso definido en el espacio del producto cartesiano U x V, es decir, es un subconjunto de dicho espacio caracterizado por la función de membresía μ_R(x, y) donde x ∈ U y y ∈V. Matemáticamente:

$$R(U, V) = [((x, y), \mu_R(x, y)) \mid (x, y) \in U \times V)]$$

La relación puede ser una herramienta útil pues eleva el grado de expresividad de variables lingüísticas, tales como: x es más grande que y, x está muy cerca de y, la planta y es mucho más rápida que la z, el documento d es más importante que el c, etc.

2.4.1 Representación

- Matrices

La relación puede ser representada mediante arreglos n - dimensionales.

Sean U y V parte de la relación binaria R(U, V) donde:

$U = \{ x_1, x_2, x_3, ..., x_n \}$ y $V = \{ y_1, y_2, y_3, ..., y_n \}$

Manteniendo el orden de los elementos en los conjuntos se define la matriz:

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{bmatrix}$$

donde $r_{ij} = R(x_i, y_j)$ para cada $i = 1, 2, 3, ..., n$ y $j = 1, 2, 3, ..., m$

- Proyección (mapping)

Es una representación visual de una relación binaria difusa. Consiste en enlazar mediante conexiones los pares ordenados del producto cartesiano indicando el grado de membresía.

Sean

$U = \{ x_1, x_2, x_3, x_4 \}$ y $V = \{ y_1, y_2, y_3, y_4 \}$

la relación R puede ser visualizada en la figura 2.9:

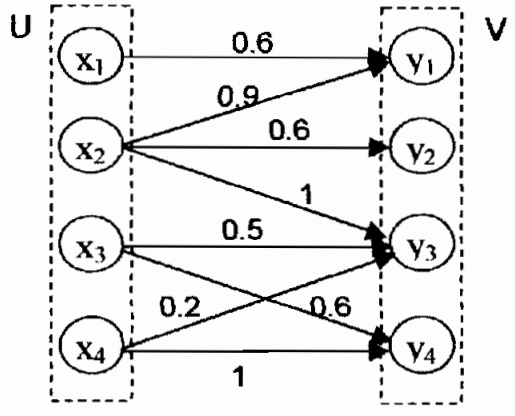


Fig. 2.9 Representación gráfica de una relación

Se evidencia la fuerza de la asociación entre los conjuntos dados por los grados de membresía.

Ejemplo 9:

Establecer una función de membresía que represente la oración "x es mucho más grande que y " si, U = { x_1, x_2, x_3} y V = { y_1, y_2, y_3, y_4 }

La representación de $\mu_{MG}(x, y)$ matricial es:

$$\mu_{MG}(x, y) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0.8 & 1 & 0 & 0.7 \\ 0 & 0.8 & 0.1 & 0 \\ 1 & 0 & 0.7 & 0.8 \end{pmatrix} \end{array}$$

Gráficamente:

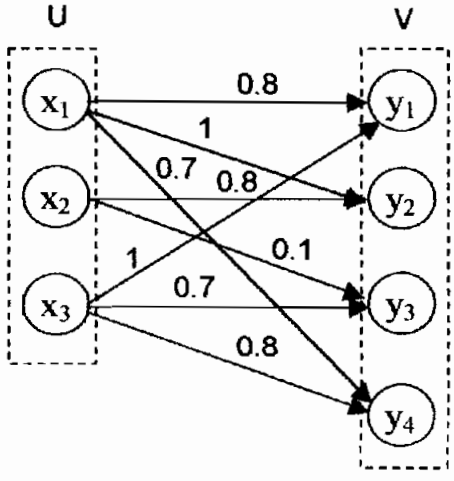


Fig. 2.10 Representación gráfica de la composición R(U,V)

2.4.2 Operaciones en las relaciones binarias difusas

Para las relaciones binarias difusas se puede hacer extensivas las operaciones básicas de unión, intersección y complemento, utilzando los operadores norma s y norma t.

Así, la unión e intersección se definen:

$\mu_{P \cup Q}(x,y) = \mu_P(x,y) \oplus \mu_Q(x,y)$

$\mu_{P \cap Q}(x,y) = \mu_P(x,y) * \mu_Q(x,y)$

donde P y Q son dos relaciones difusas en el mismo espacio del producto U x V

De igual manera, son aplicables los cortes alfa (α) y otras operaciones básicas.

Pero hay dos operaciones importantes que son aplicables a todos los conjuntos difusos, éstas son las operaciones inversa y de composición.

2.4.2.1 Operación inversa

El inverso de una relación binaria difusa R definida sobre el universo cartesiano U x V usualmente denotado R^{-1}, es una relación definida sobre el producto V x U:

$$R^{-1}(y,x) = R(x,y)$$

es fácilmente deducible que:

$$(R^{-1})^{-1} = R$$

Cuando R es una matriz, la inversa se obtiene intercambiando las filas por las columnas, esta matriz es llamada transpuesta.

Ejemplo 10:

$$R = \begin{bmatrix} 0.5 & 1 & 0.9 & 0 \\ 0.8 & 0.3 & 0.6 & 1 \\ 0 & 0.4 & 0.8 & 0 \end{bmatrix} \qquad R^{-1} = \begin{bmatrix} 0.5 & 0.8 & 0 \\ 1 & 0.3 & 0.4 \\ 0.9 & 0.6 & 0.8 \\ 0 & 1 & 0 \end{bmatrix}$$

2.4.2.2 Composición

Para llevar a cabo la composición de relaciones difusas P y Q, es necesario que las mismas sean compatibles, es decir, los espacios cartesianos U x V y V x W

deben tener en común al conjunto V, tal que (x, y) ∈ P(U, V) y (y, z) ∈ Q(V, W), por ejemplo, "x es más grande que y y y está más cerca de z".

Así, la composición max-min de las relaciones difusas P(U,V) y Q(V,W), expresada por sus funciones de membresía es:

$$\mu_{PoQ}(x,z) = \left\{(x,z), \max_{y \in Y} \min\left[\mu_P(x,y), \mu_Q(y,z)\right]\right\}$$

La composición max-producto de las relaciones difusas P(U,V) y Q(V,W), expresada por sus funciones de membresía es:

$$\mu_{PxQ}(x,z) = \left\{(x,z), \max_{y \in Y}\left[\mu_P(x,y).\mu_Q(y,z)\right]\right\}$$

Si se expresa las relaciones como matrices, para la composición max min se tiene:

$\mu_P = [p_{ij}]$, $\mu_Q = [q_{jk}]$, $\mu_R = [r_{ik}]$

entonces la composición $\mu_R = \mu_P$ o μ_Q , se define

$[r_{ik}] = [p_{ij}]$ o $[q_{jk}]$

donde $r_{ik} = \max_j \min (p_{ij}, q_{jk})$

De igual manera, la composición max-producto expresado como matrices se define:

$\mu_P = [p_{ij}]$, $\mu_Q = [q_{jk}]$, and $= [r_{ik}]$

entonces la composición $\mu_R = \mu_P \times \mu_Q$, se define

$[r_{ik}] = [p_{ij}] \times [q_{jk}]$

donde $r_{ik} = \max_j \text{prod} (p_{ij}, q_{jk})$

Ejemplo 11: (Klir)

Sean

U = {p1, p2, p3, p4}

V = {s1, s2, s3 }

W = {e1 ,e2, e3, e4, e5}

Un conjunto de pacientes, un conjunto de síntomas y un conjunto de enfermedades, respectivamente. Asuma que una relación difusa P sobre U x V es definida por la función de membresía así:

$$\mu_P(p,s) = \begin{bmatrix} 0 & 0.3 & 0.4 \\ 0.2 & 0.5 & 0.3 \\ 0.8 & 0 & 0 \\ 0.7 & 0.7 & 0.9 \end{bmatrix}$$

Esta relación, la cual es obtenida examinando a los pacientes, describe cuán fuertemente se han manifestado los síntomas en cada uno de los pacientes. Asociada a la relación difusa Q definida sobre V x W se tiene a la matriz:

$$\mu_Q(s,e) = \begin{bmatrix} 0.7 & 0 & 0 & 0.3 & 0.6 \\ 0.5 & 0.5 & 0.8 & 0.4 & 0 \\ 0 & 0.7 & 0.2 & 0.9 & 0 \end{bmatrix}$$

Esta relación describe un segmento del conocimiento médico expresando cómo cada síntoma es asociado a una enfermedad.

Realizando la composición max min $\mu_{P \circ Q}(p, e) = \mu_R(p, e)$ se tiene

$$\begin{bmatrix} 0 & 0.3 & 0.4 \\ 0.2 & 0.5 & 0.3 \\ 0.8 & 0 & 0 \\ 0.7 & 0.7 & 0.9 \end{bmatrix} \circ \begin{bmatrix} 0.7 & 0 & 0 & 0.3 & 0.6 \\ 0.5 & 0.5 & 0.8 & 0.4 & 0 \\ 0 & 0.7 & 0.2 & 0.9 & 0 \end{bmatrix} = \begin{bmatrix} 0.3 & 0.4 & 0.3 & 0.4 & 0 \\ 0.5 & 0.5 & 0.5 & 0.4 & 0.2 \\ 0.7 & 0 & 0 & 0.3 & 0.6 \\ 0.7 & 0.7 & 0.7 & 0.9 & 0.6 \end{bmatrix}$$

La relación así formada $\mu_R(p, e)$ expresa la asociación entre pacientes y enfermedades.

Esto se ha obtenido luego de aplicar la fórmula antes indicada, así:

$0.3 = r_{11} = \max[\min(p_{11},q_{11}), \min(p_{12},q_{21}), \min(p_{13},q_{31})]$

$= \max[\min(0, 0.7), \min(0.3, 0.5), \min(0.4, 0)]$

$= \max[0, 0.3, 0]$

$0.7 = r_{43} = \max[\min(p_{41},q_{13}), \min(p_{42},q_{23}), \min(p_{43},q_{33})]$

$= \max[\min(0.7, 0), \min(0.7, 0.8), \min(0.9, 0.2)]$

$= \max[0, 0.7, 0.2]$

Una forma más sencilla de realizar una composición, teniendo un universo de discurso discreto, es seguir los siguientes pasos.

- Para la composición *max min*:

1. Escriba cada elemento de la matriz producto P(x,y).Q(y,z), pero;
2. Trate cada multiplicación como una operación *min*, y entonces
3. Trate cada suma como una operación *max*.

- Para la composición *max producto*:

1. Escriba cada elemento de la matriz producto P(x,y).Q(y,z), pero;
2. Trate cada multiplicación como una multiplicación algebraica, y entonces
3. Trate cada suma como una operación *max*.

En las composiciones difusas, no se obtiene el mismo resultado para la *max min* y en la *max producto*.

Para resumir matemáticamente las fórmulas de composición max-min y max-producto, se tiene la composición supremo-asterisco:

$$\mu_{PoQ}(x,z) = \sup_{y \in V} \left| \mu_P(x,y) * \mu_Q(y,z) \right|$$

en la cual, supremo indica la operación máximo y el asterisco (norma t) puede ser la operación mínimo o producto

En la operación de composición se tiene dos importantes propiedades:

$$(P \circ Q) \circ R = P \circ (Q \circ R)$$

y

$$(P \circ Q)^{-1} = Q^{-1} \circ P^{-1}$$

Cabe indicar, que para las relaciones difusas no se cumple la propiedad conmutativa:

$$P \circ Q \neq Q \circ P$$

2.5 ACTIVACIÓN DE UNA RELACIÓN DIFUSA

La composición supremo-asterisco, fórmula matemática introducida para definir ambos tipos de composición (max-min y max-producto), puede ser interpretada mediante el diagrama de bloques de la figura 2.11

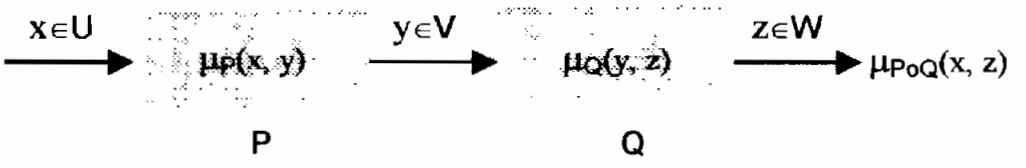


Fig. 2.11 Interpretación de la composición supremo-asterisco

Ahora, si P no es una relación sino solamente un conjunto difuso, en lugar de $\mu_P(x, y)$ se tendría $\mu_P(x)$, por ejemplo, "y es mediano y z es más pequeña que y", entonces U = V, por tanto:

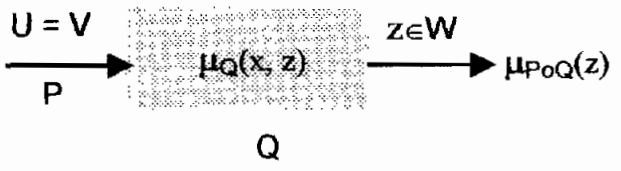


Fig. 2.12 Interpretación de la composición si P es un conjunto difuso

Luego, la composición supremo-asterisco es:

$$\sup_{y \in V} \left[\mu_P(x,y) * \mu_Q(y,z) \right] = \sup_{x \in U} \left[\mu_P(x) * \mu_Q(x,z) \right]$$

cuya única variable de salida es z. Entonces,

$$\mu_{PoQ}(z) = \sup_{x \in U} \left[\mu_P(x) * \mu_Q(x,z) \right]$$

Esto nos lleva a un importante concepto que será tratado más adelante, la implicación difusa, con la cual una regla dispara una relación.

Esto es especialmente importante para las aplicaciones en que utilizan sistemas de lógica difusa.

3. TALLER

1. Indique 3 diferencias entre conjuntos clásicos y conjuntos difusos

2. Escoja un universo de discurso adecuado para el conjunto "*temperaturas en Quito*"

3. Partiendo de la variable lingüística *edades*, enumere 5 términos que pertenezcan esta variable.

4. Dado el conjunto

S={50,100,150,200,250, 300, 350, 400, 450,500, 550, 600,700,800,900,100}
Donde S representa los sueldos de varios empleados de una compañía, asigne grados de membresía de tal forma que, el conjunto capture el concepto de *sueldos razonables*.

5. Dentro del universo de discurso [0, 100] defina la función de membresía de los conjuntos difusos *joven, edad madura, viejo* y
   a. Dibuje sus funciones de membresía
   b. Determine el complemento de estos tres conjuntos
   c. Determine la unión y la intersección para cada par de conjuntos

6. Usando la tabla 2.2, y usando el principio de extensión determine el correspondiente conjunto difuso que recoja el concepto máximo ritmo cardiaco en una persona de edad media.

7. Enumere algunos ejemplos de relaciones difusas que se encuentra a diario.

8. Para el ejemplo dado de relaciones, realice la composición max product y verifique que no se obtiene la misma respuesta

9. Dadas las matrices

$$R = \begin{bmatrix} 0.8 & 0.9 & 0.2 \\ 1 & 0.3 & 0.5 \end{bmatrix} \quad \text{y} \quad S = \begin{bmatrix} 0.4 & 0.7 \\ 0.2 & 0.9 \\ 0.5 & 0.7 \end{bmatrix}$$

que representan relaciones binarias difusas. Calcule las siguientes composiciones:
   a. R o S y R x S
   b. $\overline{R}$ o $\overline{S}$
   c. S^{-1} x R^{-1}
   d. R^{-1} o (R o S)

REFERENCIAS BIBLIOGRÁFICAS

1. Klir George, St. Clair Ute,Yuan Bo, **"Fuzzy Set Theory, Foundations and Applications"**, Prentice Hall

2. Mendel Jerry, **"Fuzzy Logic Systems for Ingineering: A Tutorial"**, IEEE

3. Jantzen Jan, **"Tutorial of Fuzzy Logic"**, Universidad Técnica Denmark

4. **"Fuzzy Logic Toolbox User's Guide"** , The MathWorks, Inc.

LÓGICA DIFUSA

OBJETIVOS

- Extender los conceptos desde la lógica clásica a la lógica difusa.
- Introducir al alumno en las reglas de conocimiento de implicación difusa necesarias para la aplicaciones de ingeniería.

MARCO TEÓRICO

2.1 LÓGICA CLÁSICA

La lógica matemática es una forma de lenguaje simbólico, en el que se emplea símbolos apropiados que siguen reglas bien determinadas.

El lenguaje simbólico utilizado en las matemáticas clásicas es preciso y no da lugar a falsas interpretaciones.

Sean las siguientes afirmaciones:
El proceso paró
La caldera está encendida
La Tierra gira alrededor de la Luna

Este tipo de oraciones son llamadas enunciados simples y son frases declarativas que pueden ser verdaderas (V) o falsas (F), pero no ambas a la vez. La verdad o falsedad de un enunciado simple es su valor de verdad. Tales enunciados son representados por símbolos *p*, *q*, *r*, etc., y son llamados proposiciones.

Varias proposiciones pueden ser combinadas para formar proposiciones compuestas por medio de conectivos.

El proceso paró **y** la caldera está encendida
p q

Los conectores más comunes son:

- La negación, que se denota con el símbolo ~ (no) y se antepone al enunciado.

$$\sim p \qquad \text{se lee} \qquad \text{'no p'}$$

- La conjunción $\wedge$ (y) en donde, el valor de verdad afirmativo de la proposición así compuesta está dado por la afirmación simultánea de las proposiciones simples.

$$p \wedge q \quad \text{se lee} \qquad \text{'p y q'}$$

- La disyunción inclusiva $\vee$ (o) donde la afirmación es dada independientemente por cada una de las proposiciones simples, basta con que una de ellas sea verdadera.

$$p \vee q \qquad \text{se lee} \qquad \text{'p o q'}$$

En la tabla 2.1 se indican los respectivos valores de verdad.

p	q	p∧q	p∨q	~ p
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Tabla 2.1 Valor de verdad de los conectores lógicos y, o, no

Otro tipo de conector es el de implicación ($\rightarrow$) que expresa una condición y se lee: *p* IMPLICA *q* , o también SI *p* ENTONCES *q* ; donde *p* es el antecedente y *q* es el consecuente. Para la implicación, una proposición es verdadera si se cumple una de las tres siguientes afirmaciones:

1. El antecedente es verdadero, el consecuente es verdadero
2. El antecedente es falso, el consecuente es falso
3. El antecedente es falso, el consecuente es verdadero

Adicionalmente, se usa con frecuencia el bicondicional ($\leftrightarrow$) que indica una relación de equivalencia, *p* SI Y SOLAMENTE SI *q*.

La posibilidad de verdad o falsedad de una o varias proposiciones puede ser expresada en una tabla lógica, y permite a su vez demostrar relaciones.

Ejemplo 1:

p	q	r	$p \wedge q$	$q \vee r$	$(p \wedge q) \rightarrow (q \vee r)$
V	V	V	V	V	V
V	V	F	V	V	V
V	F	V	F	V	V
V	F	F	F	F	V
F	V	V	F	V	V
F	V	F	F	V	V
F	F	V	F	V	V
F	F	F	F	F	V

Tabla 2.2 Tabla lógica

El resultado obtenido es una **tautología**. La tautología es una proposición formada por otras proposiciones (*p, q, r*, etc.) cuyo valor de verdad es siempre verdadero, independiente de los valores de verdad de cada variable que la conforma (*p, q, r*, etc.)

2.2 LÓGICA DIFUSA

Extender la teoría de la lógica binaria a la difusa provee de métodos que posibilita representar e interpretar adecuadamente el significado de términos ambiguos, es decir, la lógica difusa permite describir y razonar un problema a partir de información imprecisa.

Mientras que en la lógica clásica, una preposición se restringe a dos valores lógicos, verdadero o falso y no ambos, en la lógica difusa a una proposición puede asignársele valores de verdad intermedios, es decir en el rango de [0, 1].

La primera extensión a realizarse es reemplazar la proposición binaria (p, q, r, etc.) por funciones de membresía difusas.

Los conectores utilizados son los mismos que se conocen en la lógica clásica, una proposición que es modificada negándola es llamada negación y su conector representativo es $\sim$, así mismo, el conector conjuntivo *y* ($\wedge$) y el disyuntivo *o* ($\vee$) que enlazan dos frases o proposiciones simples.

Dos proposiciones pueden unirse también mediante un condicional o implicación, "SI antecedente ENTONCES consecuente", representado por el signo $\rightarrow$ pero la definición de la implicación binaria es modificada para hacerla útil a los requerimientos de los sistemas difusos, esto se discutirá más adelante.

La conjunción, disyunción y negación en la lógica difusa pueden ser resueltas mediante las operaciones intersección, unión y complemento respectivamente, es decir, se evidencia una equivalencia entre las operaciones básicas de conjuntos difusos y los conectores lógicos; esto se observa en la tabla 2.3:

μ_A	μ_B	$\min(\mu_A,\mu_B)$
0	0	0
0	1	0
1	0	0
1	1	1

y

μ_A	μ_B	$\max(\mu_A,\mu_B)$
0	0	0
0	1	1
1	0	1
1	1	1

o

μ_A	$1-\mu_A$
0	1
1	0

no

Tabla. 2.3 Equivalencia entre operaciones de conjuntos y conectores lógicos

Pero en la lógica difusa, como ya se dijo, los valores de verdad no se restringen a ser verdadero o falso (1 o 0), sino que se puede tener valores intermedios, como se observa en el ejemplo de proposición difusa de la figura 2.1 y en las funciones continuas de la figura 2.2:

Sea la proposición: La temperatura de 50° es alta.

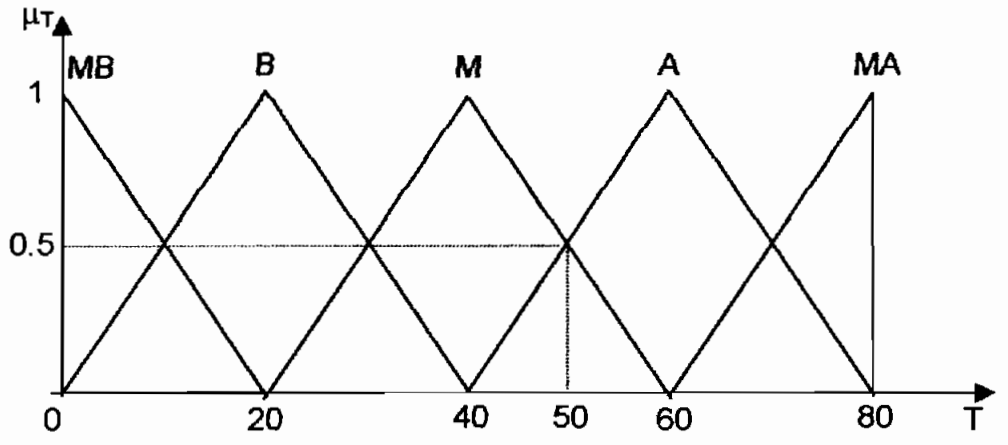


Fig. 2.1 Variable lingüística

En el contexto de la variable lingüística temperatura T de la figura 2.1 se han definido los conjuntos difusos MB, M, M, A y MA para las temperaturas muy baja, baja, media, alta y muy alta respectivamente. Entonces, de acuerdo a la proposición inicial, 50° está asociada a la función de membresía de temperatura alta μ_A con 0.5 y de temperatura muy alta μ_{MA} por el valor de 0.5. Estos grados de membresía vienen a ser los valores de verdad de dicha proposición.

En la figura 2.2 se tiene las funciones de membresía μ_A y μ_B asociadas a las proposiciones difusas sobre las cuales se ejemplifica el uso de los operadores lógicos difusos.

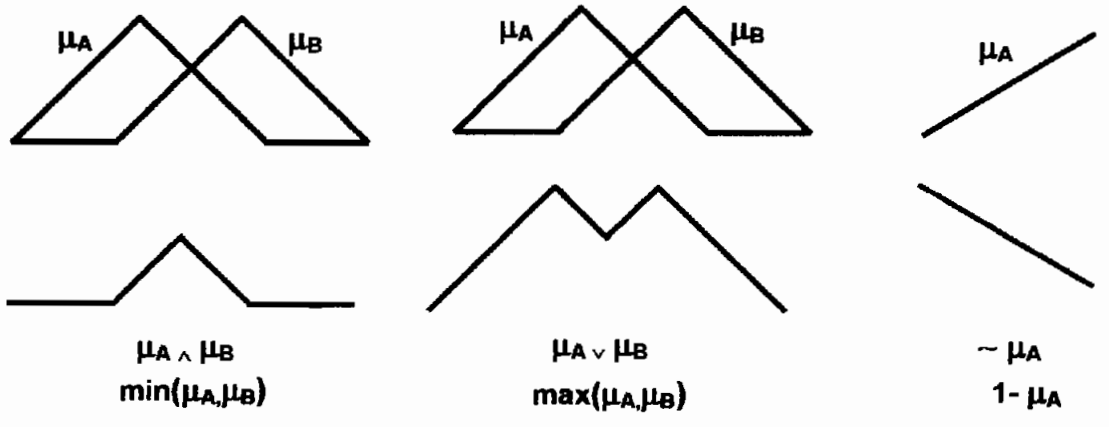


Fig. 2.2 Aplicación de los operadores lógicos en funciones continuas

Lo que se puede evidenciar con estas gráficas, es que, si por ejemplo en el punto en que se intersecan las funciones de membresía de los conjuntos A y B ambas valen 0.5 (distinto de 1 o 0), la operación lógica no dará un limitado valor de verdad falso o verdadero sino un *quizá*.

A las funciones de membresía de los conjuntos difusos se les asocia grados de verdad a través de la relación: $t(x) = \mu(x)$, gráficamente:

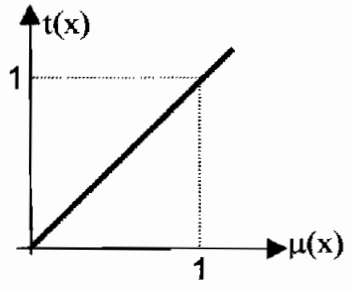


Fig. 2.3 Relación grado de verdad - función de membresía

2.2.1 Modificadores lingüísticos

Un modificador lingüístico es una operación que modifica el significado de un término, convirtiéndose así en una operación que actúa sobre las funciones de membresía de los conjuntos difusos.

Los modificadores permiten cuantificar de cierta manera muchas expresiones cotidianas, tales como: muy, más o menos, un poco, exactamente

A continuación, se indican algunos modificadores:

muy A $\equiv$ A^2

más o menos A $\equiv$ $A^{1/2}$

extremadamente A $\equiv$ A^3

ligeramente A $\equiv$ $A^{1/3}$

Como se observa, un modificador se define en general A^p donde el rango de p es [0 - ∞]. Cuando p es ∞ el modificador es llamado *exactamente*

Los modificadores lógicos cambian la relación t(x) =μ(x), así:

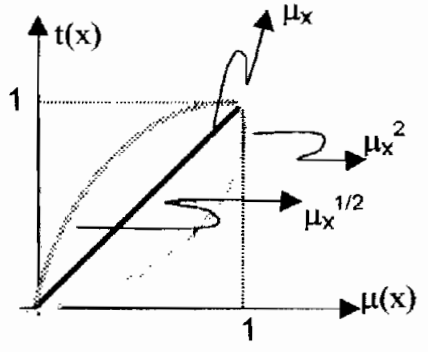


Fig. 2.3 Modificadores lógicos

Ejemplo 2:

alto = {0.2 0.6 0.8 1}

muy alto = $alto^2$ = {0.04 0.36 0.64 1}

muy muy alto = $alto^4$ = {0.0016 0.13 0.41 1}

2.3 REGLAS DE CONOCIMIENTO

Una **regla de conocimiento** es una forma de codificar la experiencia del experto de tal forma que, una acción produzca la reacción correspondiente, luego de haberse sometido a un juicio lógico dado por el **proceso de inferencia**.

Una regla de conocimiento puede tomar la forma de una implicación difusa, así:

SI {variable lingüística} es {etiqueta} **ENTONCES** {variable lingüística} es {etiqueta}

La colección de reglas de conocimiento forman la base de conocimiento.

En general, la base de conocimiento toma la forma:

$$\mathcal{R}^{(\ell)}: \text{SI } x_1 \text{ es } A_1^{(\ell)} \text{ Y } \ldots \text{ Y } x_n \text{ es } A_n^{(\ell)} \text{ ENTONCES } y \text{ es } B^{(\ell)}$$

Donde:

ℓ = 1, 2, 3, 4, ... M.

$A_i^{(\ell)}$ y $B^{(\ell)}$ son etiquetas de los conjuntos difusos caracterizados por las funciones de membresía $\mu_{A_i^{(\ell)}}(x_i)$ en U_i y $\mu_{B^{(\ell)}}(y)$ en V respectivamente.

$x = col(x_1, x_2, ..., x_p) \in U_1 \times \ldots \times U_P$ y $y \in V$ son variables lingüísticas.

En el siguiente ejemplo se expondrá el formato en que puede presentarse al usuario final una base de conocimiento:

Se tiene un sistema en que se desea controlar posición lineal y angular de un objeto. Las variables de entrada son: posición (r) y ángulo (θ); y como salida la variable fuerza (f) (r, θ y f son variables lingüísticas). Considere ahora las siguientes reglas de conocimiento:

$\mathcal{R}^{(1)}$: SI la posición (r) es negativa grande Y posición angular (θ) es negativo grande ENTONCES la fuerza (f) es negativo grande

$\mathcal{R}^{(2)}$: SI r es CE Y θ es NG ENTONCES f es NP

$\mathcal{R}^{(3)}$: SI r es PG Y θ es NG ENTONCES f es CE

$\mathcal{R}^{(4)}$: SI r es NG Y θ es CE ENTONCES f es NP

$\mathcal{R}^{(5)}$: SI r es CE Y θ es CE ENTONCES f es CE

$\mathcal{R}^{(6)}$: SI r es PG Y θ es CE ENTONCES f es PP

$\mathcal{R}^{(7)}$: SI r es NG Y θ es PG ENTONCES f es CE

$\mathcal{R}^{(8)}$: SI r es CE Y θ es PG ENTONCES f es PP

$\mathcal{R}^{(9)}$: SI r es PG Y θ es PG ENTONCES f es PG

Los nombres de las etiquetas han sido abreviados por practicidad y facilidad, así:

Bajo (BA), medio (ME), alto (AL), negativo grande (NG), positivo grande (PG), cero (CE), positivo mediano (PM), positivo pequeño (PP),

Esta información, suele ser sumarizada en una matriz (tabla 2.1), conocida como **memoria asociativa difusa (FAM)** donde, las etiquetas correspondientes a las variables de entrada están en color azul y las correspondientes a la de salida en color verde.

		r		
		NG	**CE**	**PG**
θ	**PG**	CE	PP	PG
	CE	NP	CE	PP
	NG	NG	NP	CE

Tabla 2.1 Memoria asociativa difusa

Ya para aplicaciones prácticas, puede ser útil reemplazar las palabras por números, como en el ejemplo de la tabla 2.2:

		r		
		-2	**0**	**2**
θ	**2**	0	1	2
	0	-1	0	1
	-2	-2	-1	0

Tabla 2.2 Memoria asociativa difusa reemplazando palabras por números

La FAM puede ser útil hasta para 3 variables de entrada, si se sobrepasa de este número, el arreglo n-dimensional se hace menos amigable para el usuario, en este caso, suele utilizarse Matrices Programables de Lógica Difusa, que son mapeos multidimensionales que permiten visualizar simultáneamente varias entradas y salidas [2].

Debe acentuarse, que el formato FAM asume implícitamente que el conectivo lógico entre las entradas es siempre **Y** u **O** con tal de que sea el mismo funcionamiento para todas las reglas y no una mezcla de conectivos.

No obstante, las reglas con conectivo lógico **Y** no son las únicas, a continuación se mencionará algunas que pueden resultar importantes en la práctica.

- *Reglas mixtas.* Son reglas donde se ha combinado los conectivos lógicos, así:

  $\mathcal{R}^{(1)}$: SI x_1 es $A_1^{(l)}$ Y. . . Y x_n es $A_n^{(l)}$ O x_{n+1} es $A_{n+1}^{(l))}$ Y. . . Y x_p es $A_p^{(l)}$ ENTONCES y es $B^{(l)}$

  En este caso, la regla puede ser dividida en dos, de este modo:
  $\mathcal{R}^{(1)}$: SI x_1 es $A_1^{(l)}$ Y. . . Y x_n es $A_n^{(l)}$ ENTONCES y es $B^{(l)}$
  $\mathcal{R}^{(2)}$: SI x_{n+1} es $A_{n+1}^{(l))}$ Y. . . Y x_p es $A_p^{(l)}$ ENTONCES y es $B^{(l)}$

- *Reglas comparativas.* Cuando se tiene reglas con formato comparativo, por ejemplo, "u el más pequeño, v el más grande", pueden ser reformuladas con el formato:

  SI x es P ENTONCES y es G

  Donde P es el conjunto difuso representativo "el más pequeño" y G "el más grande".

- *Reglas a menos que.* A veces las reglas se declaran usando el conectivo "a menos que", estas pueden reescribirse usando operaciones lógicas, inclusive las leyes de DeMorgan, por ejemplo:

  y es $B^{(l)}$ a menos que x_1 es $A_1^{(l)}$ Y. . . Y x_n es $A_n^{(l)}$

  usando la ley DeMorgan $\overline{A \cap B} = \bar{A} \cup \bar{B}$ esto puede ser expresado de la

siguiente forma:

SI no(x_1 es $A_1^{(i)}$ Y. . . Y x_n es $A_n^{(i)}$) ENTONCES y es $B^{(i)}$

SI x_1 no es $A_1^{(i)}$ O. . . O x_n no es $A_n^{(i)}$ ENTONCES y es $B^{(i)}$

- *Reglas cuantificadoras.* Son aquellas que incluyen los cuantificadores "algunos" o "todos". Debido a la dualidad entre la lógica proposicional y la teoría de conjuntos, el cuantificador "algunos" indica que se debe aplicar la operación de unión a los antecedentes o consecuentes en donde dicho cuantificador se ha empleado; de igual manera, se aplicará el operador de intersección a los antecedentes o consecuentes en los cuales se use el cuantificador "todos".

Pueden también presentarse reglas MIMO (entradas múltiples - salidas múltiples), estas pueden modificarse en varias reglas de entradas múltiples - salida simple para facilitar el análisis.

Puesto que las reglas de conocimiento tienen la forma de una implicación, es necesario definirla dentro del contexto de la lógica difusa.

2.4 PROCESO DE INFERENCIA

La *implicación* aplicada a una función de membresía mide el grado de verdad de la relación de implicación entre x y y.

Cuando se define la operación de implicación entre dos conjuntos difusos, existen algunas soluciones diferentes sugeridas en la literatura, así:

$$\mu_{A\to B}(x,y) = 1 - \min[\mu_A(x), 1-\mu_B(y)] \quad \text{porque} \quad p \to q \leftrightarrow \sim(p \wedge \sim q)$$

$$\mu_{A\to B}(x,y) = \max[1-\mu_A(x), \mu_B(y)] \quad \text{porque} \quad p \to q \leftrightarrow (\sim p \vee q)$$

Pero la teoría de la lógica difusa se enfrenta con un gran inconveniente al momento de trasladar desde la lógica tradicional la operación lógica de implicación, a continuación se explicar por qué.

Si solo se traslada las nociones desde la lógica tradicional hacia la difusa, como se lo ha venido haciendo, esta teoría ya no tendría una utilidad válida en la ingeniería, esto significa que no hay una equivalencia para la implicación difusa como para el resto de operadores. Para efectos de aplicaciones de ingeniería tal como el control, el principio de *causa y efecto* es fundamental, por obvias razones, si hay una causa debe llevar a un efecto. Las definiciones de implicación difusa anteriormente señaladas no cumplen con este principio, pues presentan un *efecto* aún sin una causa válida. El lector puede encontrar las correspondientes demostraciones en la referencia [1].

Esto se evidencia en la tabla 2.4:

$\mu_A(x)$	$\mu_B(y)$	$1-\mu_A(x)$	$1-\mu_B(y)$	$\max[1-\mu_A(x), \mu_B(y)]$	$1-\min[\mu_A(x), 1-\mu_B(y)]$
1	1	0	0	1	1
1	0	0	1	0	0
0	1	1	0	1	1
0	0	1	1	1	1

Tabla 2.4 Tabla de verdad de la implicación lógica difusa incorrecta

En la tabla se observa que la regla se dispara para casi cualquier caso, evidentemente esto no es conveniente para los fines que se persigue.

Mamdani formuló adecuadamente la implicación difusa de manera que cumpla, con el principio básico de ingeniería, causa y efecto, se la conoce como *implicación mínimo* y se define:

$$\mu_{A\to B}(x,y) = \min[\mu_A(x), \mu_B(y)]$$

Otra definición frecuentemente utilizada que también observa el principio es la que propuso más tarde Larsen conocida como *implicación producto*:

$$\mu_{A\to B}(x,y) = \mu_A(x).\mu_B(y)$$

Para ambas definiciones, conocidas como *implicaciones de Mamdani*, A y B son dos conjuntos difusos definidos no necesariamente en el mismo universo.

La tabla 2.5 confirma lo dicho:

$\mu_A(x)$	$\mu_B(y)$	$\min[\mu_A(x), \mu_B(y)]$	$\mu_A(x).\ \mu_B(y)$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0

Tabla 2.5 Implicación mínimo y producto

Éstas reglas de inferencia se utilizan también porque requieren menor esfuerzo computacional.

Es indudable que las *implicaciones de Mamdani* no tienen relación alguna con la implicación binaria, pues preservan la causa y efecto al disparar la regla $\mu_{A\to B}(x,y)$ solo cuando el antecedente y el consecuente son verdaderos.

Ahora, para razonar sobre el conocimiento manifiesto como reglas SI ... ENTONCES, se deberá formular primero otras reglas llamadas de inferencia.

Inferir significa concluir, deducir o llegar a una consecuencia lógica a partir de cierta evidencia; entonces, una **regla de inferencia** es un mecanismo que puede producir una salida, a partir de una colección de reglas de conocimiento.

La más famosa regla de inferencia es el *Modus Ponens*, cuyo método de deducción en la lógica tradicional es:

Hecho u	Observación	:	x es A
Regla o	Conocimiento	:	*SI* x es A *ENTONCES* y es B
	Conclusión	:	y es B

Sin embargo, en la lógica difusa para aplicaciones de ingeniería, la extensión del Modus Ponens es el *Modus Ponens Generalizado*, que establece:

Observación	:	x es A^*
Conocimiento	:	*SI* x es A *ENTONCES* y es B
Conclusión	:	y es B^*

Donde A, B, A^* y B^* son conjuntos difusos que cumplen las siguientes condiciones:

$A \neq A^*$, pero A es similar a A^*; y,

$B \neq B^*$, pero B es similar a B^*

Es importante que cumplan con esto, pues en el Modus Ponens Generalizado una regla se activa si, el grado de similitud entre la observación y el antecedente de la regla de conocimiento es diferente de cero, así, el resultado de su activación, la conclusión, tendrá un grado de similitud diferente de cero con el consecuente de la regla de conocimiento.

Esta es la diferencia radical entre el Modus Ponens y el Modus Ponens Generalizado, ya que en el primero, la observación debe ser igual al antecedente de la regla de conocimiento, con lo cual ésta se "dispara", luego, la conclusión es exactamente el mismo consecuente de la regla activada.

Ejemplo 2:

Considere la regla de conocimiento "SI la temperatura en la habitación está elevada ENTONCES encienda el ventilador a velocidad moderada". El conjunto A

es: la temperatura en la habitación está elevada y el conjunto B es: encienda el ventilador a velocidad moderada.

Ahora, dada la siguiente observación: la temperatura en la habitación es de 40°C, ésta se constituye en A^*. Es visible que $A \neq A^*$, pero como se dijo, A es similar a A^*.

Este grado de similitud entre A y A^* hace propicia la activación de la regla inicial cuya conclusión será: encienda el ventilador a velocidad alta.

La conclusión pasa a ser B^* que por supuesto no es igual a B, pero tienen su grado de similitud.

A partir de su comportamiento deductivo, se puede interpretar al Modus Ponens Generalizado con la gráfica de la figura 2.4:

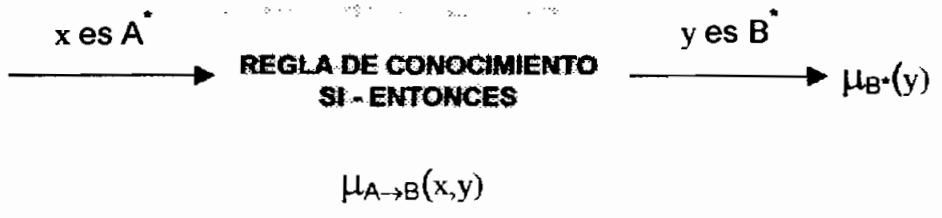


Fig. 2.4 Interpretación del Modus Ponens Generalizado

Esta interpretación gráfica permite identificar a la regla de inferencia Modus Ponens Generalizado, como una composición difusa, cuya primera relación la constituye A^*, y por tanto, $\mu_{B^*}(y)$ viene a ser el resultado de dicha composición.

Matemáticamente, y usando para la composición al operador supremo - asterisco, se define al Modus Ponens Generalizado,

$$\mu_{B^*}(y) = \sup_{x \in A^*} [\mu_{A^*}(x) * \mu_{A \to B}(x, y)]$$

Frecuentemente el conjunto A^* es singleton difuso, entonces se simplifica el Modus Ponens Generalizado, si $\mu_{A^*}(x) = 0$ si $x \neq x'$

$$\mu_{B^*}(y) = [1 * \mu_{A \to B}(x', y)]$$

Sea cual sea la norma t que se use como operador, el resultado de dicha operación será:

$$\mu_{B^*}(y) = \mu_{A \to B}(x', y)$$

De acuerdo a la explicación que se hiciera sobre la implicación difusa, se determina que la implicación que se utiliza con la regla de inferencia Modus Ponens Generalizado es la implicación de Mamdani , con lo cual se cumplen las propiedades deseables para aplicaciones de lógica difusa en ingeniería.

$$\mu_{B^*}(y) = \min[\mu_A(x'), \mu_B(y)]$$

$$\mu_{B^*}(y) = \mu_A(x') \cdot \mu_B(y)$$

TALLER

1. Explique por qué no es posible la extensión de la implicación desde la teoría de clásica a la difusa.

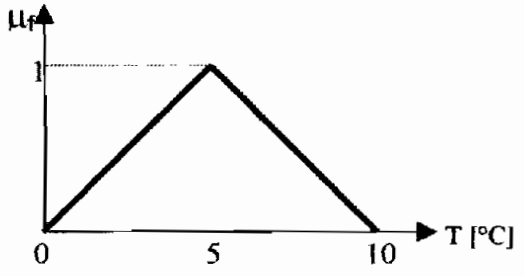
2. Según su criterio, asigne valores de verdad a las siguiente proposiciones:

   a. Júpiter está cerca de la Tierra
   b. Marte está cerca de la Tierra
   c. Plutón está cerca de la Tierra

   d. 22 es mayor que 20
   e. 20.5 es mayor que 20
   f. 30 es mayor que 20

3. De 2 ejemplos para cada tipo de reglas que se mencionaron anteriormente.

4. Dada la función de membresía μ_f que caracteriza al conjunto difuso *frío*.



Grafique nuevamente dicha función cuando ha sido afectada por los siguientes modificadores lingüísticos:

- muy
- más o menos
- extremadamente
- ligeramente

5. Repita el ejercicio anterior si la función de membresía es gaussiana.

6. Demuestre que la expresión para la implicación $\mu_{A\to B}(x,y)=1-\min[\mu_A(x),1-\mu_B(y)]$ aplicada al a la fórmula del Modus Ponens Generalizado $\mu_{B^*}(y)=\sup_{x\in A^*}[\mu_{A^*}(x)*\mu_{A\to B}(x,y)]$, no cumple con el principio de causa y efecto . Asuma que $\mu_{A^*}(x) = 1$ para $x = x'$ y $\mu_{A^*}(x) = 0$ para todas las $x \in U$ con $x \neq x'$

REFERENCIAS BIBLIOGRÁFICAS

1. Mendel Jerry, **"Fuzzy Logic Systems for Ingineering: A Tutorial"**, IEE.

2. Sánchez Alberto, **"Simulación Gráfica en Windows de un Elevador Empleando Técnicas de Control Difuso"**, Tesis, Escuela Politécnica Nacional

3. Jantzen Jan, **"Tutorial on Fuzzy Logic"**, Universidad Técnica de Denmark.

4. **"Fuzzy Logic Toolbox User's Guide"** , The MathWorks, Inc.

5. Londoño Nelson, Bedoya Hernando, **"Serie Matemática Progresiva"**, Editorial Norma.

SISTEMAS DE LOGICA DIFUSA

1. OBJETIVOS

- Conocer la estructura de un sistema de lógica difusa.
- Introducir al alumno en el diseño de un controlador usando como herramienta básica la lógica difusa

2. MARCO TEÓRICO

Se ha comprendido que la diferencia básica entre la lógica difusa y la clásica es que el valor de verdad de una proposición difusa puede tomar valores en un rango de [0, 1]. Sin embargo, cuando se quiere implementar un controlador difuso, lo que se tiene tanto en la entrada como en la salida son valores numéricos, por tanto, es necesario adicionar a nuestro sistema interfaces que nos permitan transformar los datos de entrada, de conjuntos clásicos a difusos, y viceversa para la salida de nuestro sistema.

Un sistema difuso tiene la estructura esquematizada en la figura 2.1

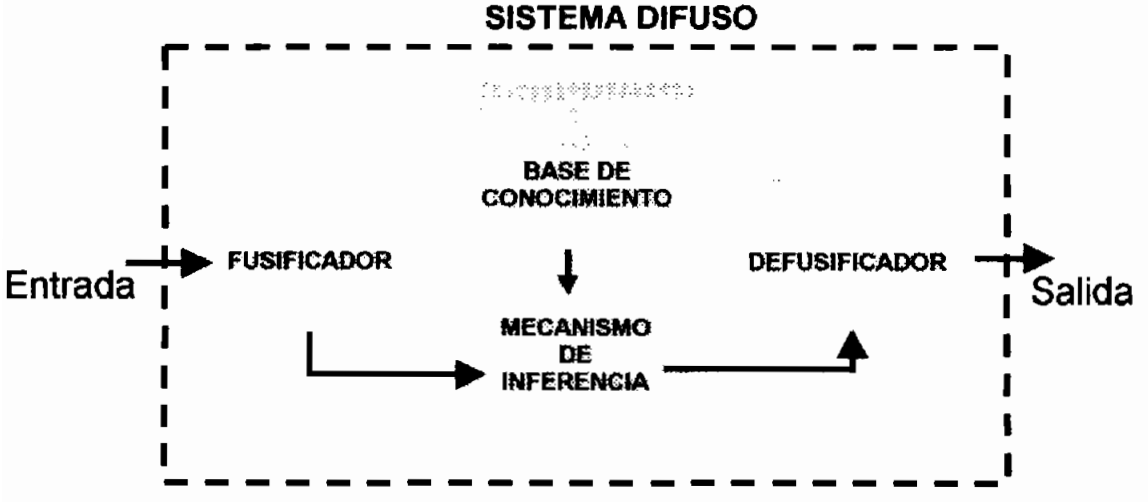


Fig. 2.1 Diagrama de bloques de un sistema difuso

Este tipo de sistema es conocido como sistema de Mamdani, y sus elementos básicos son: fusificador, reglas base, máquina de inferencia y defusificador.

2.1 FUSIFICADOR

Un fusificador $\mathcal{F}$ es una interface entre el medio ambiente y el mecanismo de inferencia, pues convierte un valor numérico x' a un conjunto difuso $\mu_A(x')$.

El conjunto difuso debe cumplir los siguientes axiomas:

$\mu_A(x') = 1$

$\mu_A(x')$ es una función decreciente de $\| x - x' \|$

Los fusificadores recomendados en la literatura son: triangular, gaussiano y singleton, este último es predominante porque simplifica enormemente los cálculos. Esto se debe a que la operación supremo asterisco de composición que utiliza la regla de inferencia desaparece cuando se usa este fusificador.

El fusificador singleton genera un conjunto difuso A^* con un solo elemento, cuya función de membresía está dada por:

$\mu_{A^*}(x) = 1$ para $x = x'$; y,

$\mu_{A^*}(x) = 0$ para todos los $x \in U$ con $x \neq x'$

No obstante, no siempre es adecuado utilizar la función singleton, especialmente cuando la medida a fusificar contiene ruido, se recurre entonces a otros fusificadores, tales como los mencionados anteriormente, triangular o gaussiano; estos introducen una medida de incertidumbre que al final, es el propósito de trabajar con lógica difusa.

Un fusificador no singleton es aquel que cumple con:

$\mu_{A^*}(x') = 1$; y

$\mu_{A^*}(x)$ decrece desde la unidad en la medida en que x se mueve lejos de x'

Mientras más anchas son estas funciones, mayor es la incertidumbre sobre x', sin embargo, la complejidad matemática puede limitar la implementación de estos fusificadores.

El proceso de fusificación en sí, se refiere a evaluar cada función de membresía de la variable lingüística en el valor numérico de la variable de entrada.

Ahora bien, hay dos cuestiones fundamentales a las que el diseñador se enfrentará:
¿Cómo determinar la forma de los conjuntos? y,
¿Cuántos conjuntos son necesarios y suficientes?

De acuerdo a la teoría de conjuntos difusos, la forma de la función de membresía es subjetivo, sin embargo, existen unas cuantas sugerencias que podrían servir para realizar un diseño óptimo. El manual para los productos TILShell (Hill, Horstkotte & Teichrow, 1990) recomienda lo siguiente [1]

- Empiece con conjuntos triangulares. Todas las funciones de membresía para una entrada o salida particular deberían ser triángulos simétricos del mismo ancho. Las que están más hacia la izquierda y la derecha deberían ser trapezoidales.

- El sobrelapamiento debería ser de al menos el 50%. Los anchos inicialmente deberían ser tales que cada valor del universo sea miembro de al menos dos conjuntos, excepto posiblemente para los elementos que están en los extremos finales. Si por otra parte, hay un vacío (espacio) entre dos conjuntos las reglas no dispararán para los valores de aquel intervalo. Consecuentemente el funcionamiento del controlador no está definido.

Ejemplo 1:

En el siguiente ejemplo se precederá a fusificará gráficamente, utilizando funciones triangulares:

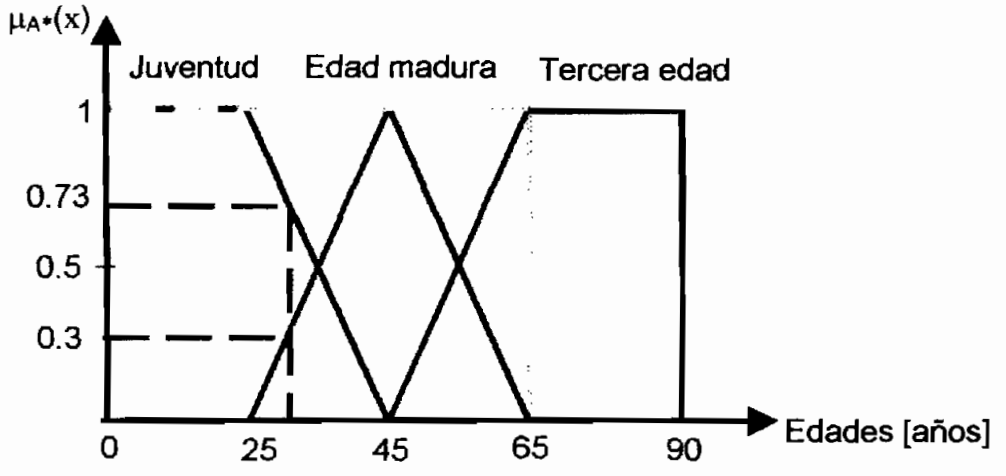


Fig. 2.2 Fusificación gráfica de edades

Fusificaremos dos valores numéricos de la variable Edades, 30 y 65 años.

Al fusificar el valor 30 [años], se observa que se le asigna dos diferentes grados de membresía: 0.73 en el conjunto difuso etiquetado "juventud" y 0.3 en el conjunto difuso etiquetado "edad madura"; contrariamente, al fusificar el valor 65 (años), sólo se asigna un grado de membresía de valor 1, correspondiente al conjunto difuso "tercera edad".

2.2 BASE DE CONOCIMIENTO

Las reglas base o base de conocimiento es una colección de reglas con la forma **SI** {variable lingüística} es {etiqueta} **ENTONCES** {variable lingüística} es {etiqueta} y las funciones de membresía que caracterizan a los conjuntos difusos.

El número posible de reglas que compondrá la base de conocimiento es:

$$\#\text{ de posibles combinaciones} = \prod_{i=1}^{n} N_i$$

donde: i es el número de variables de entrada

N es el número de etiquetas para cada una de las variables

Puesto que en la unidad "Lógica Difusa" se explicó ampliamente este tema, aquí se propondrá únicamente un ejemplo

Ejemplo 2:

Suponga un sistema en el que se tiene como entradas las variables: error (e) y cambio de error (ė), ambas muy comunes en las aplicaciones de ingeniería; y como salida la variable salida (u) (e, ė y u son variables lingüísticas) considere entonces las siguientes reglas que describen un controlador difuso PD:

Con dos variables de entrada de tres etiquetas cada una, el número de combinaciones posibles es: 3 x 3 = 9 reglas

$\mathcal{R}^{(1)}$: SI el error (e) es negativo grande Y el cambio de error (ė) es negativo grande ENTONCES la salida (u) es negativo grande

$\mathcal{R}^{(2)}$: SI e es CE Y ė es NG ENTONCES u es NP

$\mathcal{R}^{(3)}$: SI e es PG Y ė es NG ENTONCES u es CE

$\mathcal{R}^{(4)}$: SI e es NG Y ė es CE ENTONCES u es NP

$\mathcal{R}^{(5)}$: SI e es CE Y ė es CE ENTONCES u es CE

$\mathcal{R}^{(6)}$: SI e es PG Y ė es CE ENTONCES u es PP

$\mathcal{R}^{(7)}$: SI e es NG Y ė es PG ENTONCES u es CE

$\mathcal{R}^{(8)}$: SI e es CE Y ė es PG ENTONCES u es PP

$\mathcal{R}^{(9)}$: SI e es PG Y ė es PG ENTONCES u es PG

La memoria asociativa difusa FAM de esta base de conocimiento se presenta en la tabla 2.1:

		e		
		NG	**CE**	**PG**
ė	**PG**	CE	PP	PG
	CE	NP	CE	PP
	NG	NG	NP	CE

Tabla 2.1 FAM de variables error, cambio de error y salida

Al iniciar el análisis, normalmente se deben seleccionar las entradas que sean determinantes en el comportamiento dinámico del sistema de lógica difusa y determinar, en base a la experiencia de un experto o basados en el conocimiento de la dinámica del sistema, las reglas de conocimiento y la matriz de memoria asociativa difusa.

Sin embargo, en otras aplicaciones el conocimiento que describe el comportamiento deseado está contenido en datos, en este caso, el diseñador debe obtener las reglas IF ... Then a partir de dichos datos. En este caso se puede combinar con otras técnicas como la utilización de redes neuronales. Se tiene entonces un sistema neuro -difuso. Las redes neuronales pueden aprender de los datos mientras que la inferencia se realiza mediante lógica difusa.

2.3 MECANISMO DE INFERENCIA

El mecanismo de inferencia es el proceso mediante el cual se combinan las reglas SI - ENTONCES en el universo $U = U_1 \times U_2 \times ... \times U_P$ para deducir una salida en el universo V. En el sistema difuso, es el mecanismo de inferencia en donde se realiza todas las manipulaciones lógicas, es el corazón mismo del sistema.

Puesto que se inferirá a partir de la base de conocimiento, cada regla tiene varios antecedentes, en forma general se puede expresarlo así:

$$A_1^{(l)} \times A_2^{(l)} \times \ldots \times A_n^{(l)} = A$$

$$B^{(l)} = B$$

$$A \rightarrow B$$

Se considera cada regla como una inferencia, y a la inferencia como un sistema que mapea conjuntos difusos en conjuntos difusos por medio de:

$$\mu_{A \rightarrow B}(x, y)$$

Donde $x = col(x_1, x_2, \ldots, x_P)$

Consecuentemente:

$$\mu_{R^{(l)}}(x, y) = \mu_{A \rightarrow B}(x, y)$$

$$\mu_{R^{(l)}}(x, y) = \mu_{A_1'}(x_1) * \ldots * \mu_{A_p'}(x_p) * \mu_{B'}(y)$$

Donde A ahora es un vector porque las reglas tienen múltiples premisas, entonces, de la ℓ-ésima regla, la entrada está dada por el conjunto difuso A_x cuya función de membresía en $x = x'$ es:

$$\mu_{A_x}(x') = \mu_{x_1}(x'_1) * \ldots * \mu_{x_P}(x'_p)$$

La operación asterisco es una operación de la norma t, mínimo o producto.

Cada $\mathcal{R}^{(l)}$ genera la salida difusa $B^{(l)} = A_x * \mathcal{R}^{(l)}$ tal que:

$$\mu_{B^l}(y) = \mu_{A_x * R^{(l)}}(y)$$

$$\mu_{B^l}(y) = \sup_{x \in A_x} \left[\mu_{A_x}(x) * \mu_{A \rightarrow B}(x, y) \right]$$

Donde, como se vió en la unidad anterior, la implicación a utilizarse es la de Mamdani (min o producto).

Es justo indicar, que en una memoria asociativa no existirán más de cuatro reglas activas al mismo tiempo.

El conjunto difuso final, $B = A_x * [\mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(M)}]$ el cual está determinado por todas las reglas de la base de conocimiento, se obtiene combinando B' y las funciones de membresía asociadas. Una forma de combinar las reglas para formar el conjunto de salida, es usar la conorma-t (operación max) con lo cual se ha conseguido buenos resultados en cuanto a aplicaciones de ingeniería se refiere.

Ejemplo 3:

Suponga que se ha fusificado cuatro entradas, dos de error y dos de cambio de error, cuyos grados de membresía son:

$$\mu_{PG}(e') = 0.4 \qquad \mu_{NG}(e') = 0.7$$

$$\mu_{NG}(\dot{e}') = 0.8 \qquad \mu_{CE}(\dot{e}') = 0.6$$

Las combinaciones que buscaremos en la FAM son:

$$\mu_{PG}(e') - \mu_{NG}(\dot{e}')$$

$$\mu_{PG}(e') - \mu_{CE}(\dot{e}')$$

$$\mu_{NG}(e') - \mu_{NG}(\dot{e}')$$

$$\mu_{NG}(e') - \mu_{CE}(\dot{e}')$$

Así, al dirigirse a la FAM (tabla 2.1) se verificará las regla que fueron activadas:

$\mathcal{R}^{(1)}$: SI e es PG y ė es NG ENTONCES u es CE

$\mathcal{R}^{(2)}$: SI e es PG y ė es CE ENTONCES u es PP

$\mathcal{R}^{(3)}$: SI e es NG y ė es NG ENTONCES u es NG

$\mathcal{R}^{(4)}$: SI e es NG y ė es CE ENTONCES u es NP

El siguiente paso es calcular los valores inferidos, para esto, se operará con el producto tanto para la conjunción de las premisas como para la implicación:

$\mathcal{R}^{(1)}$: $\mu_{CE*}(u) = \mu_{PG}(e') \cdot \mu_{NG}(\dot{e}') \cdot \mu_{CE}(u)$

$= 0.4 \cdot 0.8 \cdot \mu_{CE}(u)$

$= 0.32 \cdot \mu_{CE}(u)$

$\mathcal{R}^{(2)}$: $\mu_{PP*}(u) = \mu_{PG}(e') \cdot \mu_{CE}(\dot{e}') \cdot \mu_{PP}(u)$

$= 0.4 \cdot 0.6 \cdot \mu_{PP}(u)$

$= 0.24 \cdot \mu_{PP}(u)$

$\mathcal{R}^{(3)}$: $\mu_{NG*}(u) = \mu_{NG}(e') \cdot \mu_{NG}(\dot{e}') \cdot \mu_{NG}(u)$

$= 0.7 \cdot 0.8 \cdot \mu_{NG}(u)$

$= 0.56 \cdot \mu_{NG}(u)$

$\mathcal{R}^{(4)}$: $\mu_{NP*}(u) = \mu_{NG}(e') \cdot \mu_{CE}(\dot{e}') \cdot \mu_{NP}(u)$

$= 0.7 \cdot 0.6 \cdot \mu_{NP}(u)$

$= 0.42 \cdot \mu_{NP}(u)$

El resultado en forma gráfica observa en la figura 2.3:

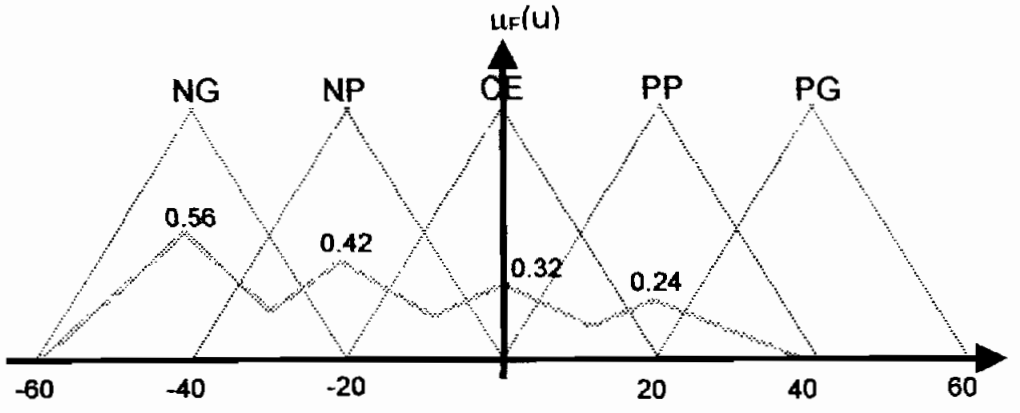


Fig. 2.3 Resultado del mecanismo de inferencia usando el producto

Si se utiliza para la implicación el mínimo en lugar del producto, lo que se obtiene son triángulos truncados

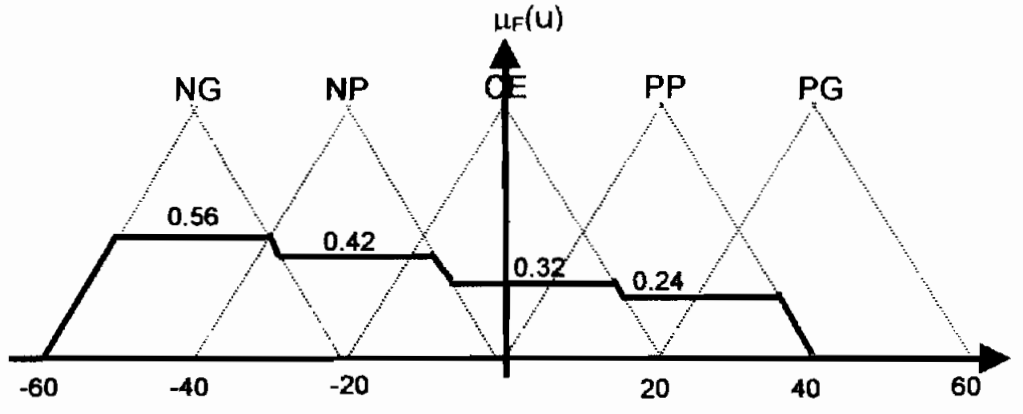


Fig. 2.4 Resultado del mecanismo de inferencia usando el mínimo

El valor numérico que se utilizará para ejercer cierto control, resulta luego de aplicar a la interface de defusificación los conjuntos difusos inferidos.

2.4 DEFUSIFICACIÓN

Luego del proceso de inferencia, lo que se obtiene es el conjunto difuso B que debe ser convertido a valores numéricos para que pueda ser aplicado como mecanismo de control. Este proceso de transformación de difuso a numérico es

llamado defusificación. En la literatura se menciona varios métodos de defusificación, revisaremos dos de los más importantes:

2.4.1 CENTRO DE GRAVEDAD (COG)

Este defusificador determina la salida $\bar{y}$ del FLS (siglas en inglés de Sistema de Lógica Difusa), mediante el cálculo del centro de gravedad del conjunto difuso B, mediante la fórmula:

$$\bar{y} = \frac{\sum_{i=1}^{M} c_i \int \mu_B^*(y)dy}{\sum_{i=1}^{M} \int \mu_B^*(y)dy}$$

Donde: M el número total de reglas inferidas

c_i valor en el cual $\mu^*_B(u)$ es máximo

$\int \mu_B^*(y)dy$ área bajo la función $\mu^*_B(u)$

Una condición importante que se debe cumplir es que la función a defusificar debe estar sobre un universo finito, es decir, no se debe usar este método con funciones de membresía saturadas.

Calcular el centro de gravedad se reduce a calcular las áreas de las funciones de membresía triangulares o triangulares truncadas. Si la función de membresía es gaussiana, hay que integrar el área bajo dicha curva.

Ejemplo 4:

Defusificaremos mediante el centro de gravedad el conjunto de salida dado en la figura 2.4. Para iniciar, en la figura 2.5 se observa las cuatro superficies a calcular:

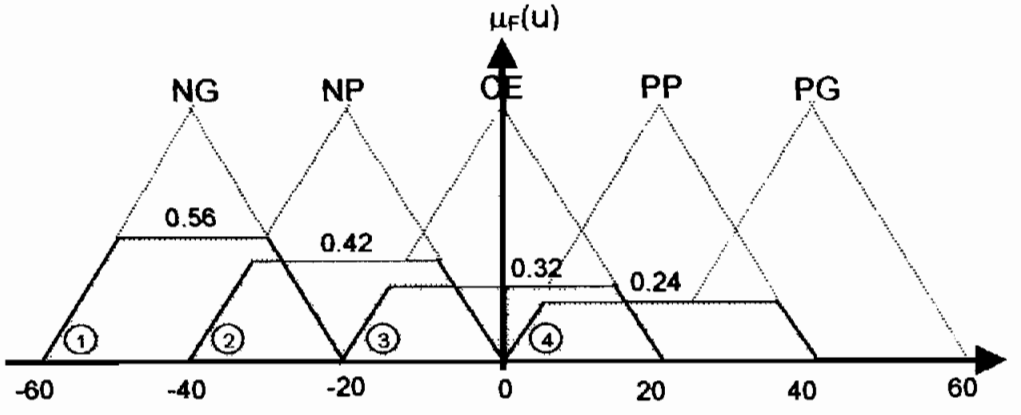


Fig. 2.5 Defusificación por el método centro de gravedad

Para calcular el área del triángulo truncado 1:

$$A1 = b\left(h - \frac{h^2}{2}\right) = 40\left(0.56 - \frac{0.56^2}{2}\right) = 16.13$$

Donde: b es la base del triángulo truncado
h es la altura

El procedimiento se repite para las demás figuras; luego:

$$\bar{y} = \frac{(-40)(16.13) + (-20)(13.27) + (0)(10.75) + (20)(8.45)}{16.13 + 13.27 + 10.75 + 8.45} = -15.25$$

2.4.2 CENTRO PROMEDIO (COA)

Sea $\bar{y}^l$ el centro de gravedad del conjunto difuso $B^{(l)}$ (el cual está asociado a la regla de activación $\mathcal{R}^{(l)}$) este defusificador primero evalúa $\mu_B(y)$ en y^l y entonces calcula la salida del FLS mediante:

$$y_h = \frac{\sum_{l=1}^{M} \bar{y}^l \cdot \mu_{B^{(l)}}(\bar{y}^l)}{\sum_{l=1}^{M} \mu_{B^{(l)}}(\bar{y}^l)}$$

Este defusificador es sencillo de usar pues los puntos medios de las funciones de membresía son fácilmente conocidos:

- En la función triangular, el vértice.
- En la función gaussiana, el valor centro de la función.
- En la función de membresía simétrica trapezoidal es el punto medio de su soporte.

Ejemplo 5:

Defusificar nuevamente el conjunto difuso dado por la figura 2.4, usando el defusificador centro promedio.

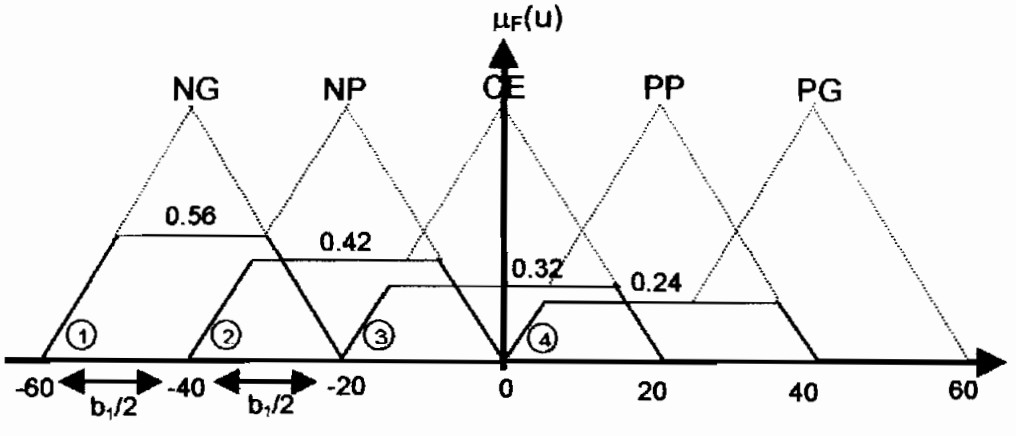


Fig. 2.6 Defusificación por el método centro promedio

Reemplazando valores en la fórmula tenemos:

$$y_h = \frac{(-40)(0.56) + (-20)(0.42) + (0)(0.32) + (20)(0.24)}{0.56 + 0.42 + 0.32 + 0.24} = -16.88$$

2.4.3 DEFUSIFICADOR CENTRO DE GRAVEDAD PARA FUNCIONES SINGLETON

Si las funciones de membresía del conjunto final (de salida) son singleton, la salida viene dada por:

$$y_S = \frac{\sum_{l=1}^{M} \mu_{B^{(l)}}(y^l) y^l}{\sum_{l=1}^{M} \mu_{B^{(l)}}(y^l)}$$

donde: y^l es la posición de la singleton ℓ en el universo

$\mu_{B(\ell)}(y^l)$ es el valor de membresía de ℓ-ésima regla

Ejemplo 6:

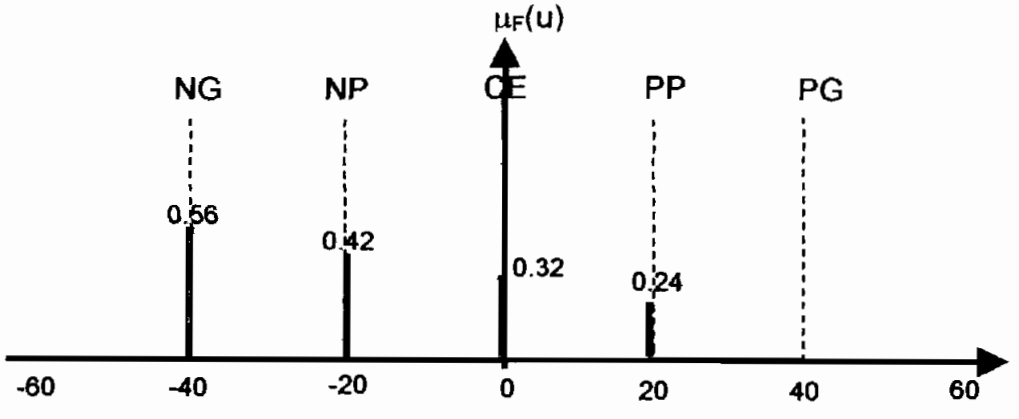


Fig. 2.7 Funciones de membresía Singleton

Defusificando las funciones implicadas tenemos:

$$y_S = \frac{(-40)(0.56) + (-20)(0.42) + (0)(0.32) + (20)(0.24)}{0.56 + 0.42 + 0.32 + 0.24} = -16.88$$

Otro método muy conocido es el máximo.

2.4.4 Máximo

Este defusificador entrega como salida del FLS el valor de y para el cual $\mu_B(y)$ es máximo.

En nuestro caso, u = -40, no refleja un resultado coherente y dependiendo de la aplicación esto podría estropear el control.

2.5 SUPERFICIE DE CONTROL

La relación entre entradas y salidas puede ser visualmente inspeccionadas con la ayuda de un gráfico. Con dos entradas y una salida éste gráfico resulta una superficie donde se aprecia la linealidad o no de la relación final del sistema (figura 2.8).

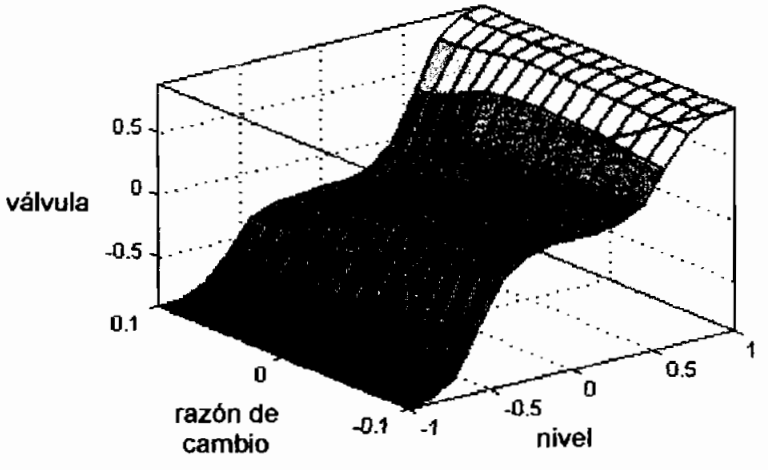


Fig. 2.8 Ejemplo de una superficie de control

Dicha linealidad depende de varios parámetros del diseño, tales como:

- La base de conocimiento. La posición forma y número de conjuntos difusos así como la escala no lineal de las entradas causa transformaciones no lineales. Las reglas generalmente expresan una estrategia de control no lineal.

- El mecanismo de inferencia. Si los conectivos y y o son implementados como por ejemplo *min* y *max* respectivamente introducen no linealidades.

- El defusificador. Muchos métodos de defusificación son no lineales.

En este caso, la superficie no lineal resulta de cinco conjuntos difusos y las variables de entrada son *nivel* y *razón de cambio* y la de salida es *válvula*.

2.6 SISTEMAS DIFUSOS DE TAKAGI - SUGENO

El sistema difuso propuesto por Takagi y Sugeno usa la fusificación singleton y sugiere una manera diferente de plantear las reglas de conocimiento, estas toman la siguiente forma:

$$\mathcal{R}^{(l)}: \text{SI } x_1 \text{ es } A_1^{(l)} \text{ Y. . . Y } x_n \text{ es } A_n^{(l)} \text{ ENTONCES } y = h^{(l)}(x)$$

La función de salida es una función arbitraria de la entrada x. En muchos casos, la salida es una constante o una combinación lineal de las variables de la entrada, por ejemplo:

SI error (e) es cero y cambio de error (ė) es cero ENTONCES $h = c$

Donde: c es una constante numérica (crisp)

SI e es cero y ė es cero ENTONCES $h = a + b \times e + c \times \dot{e}$

Donde: a, b y c son constantes crisp

En general,

$$h^{(l)}(x) = c_0 + c^{(l)}_1.x_1 + \ldots + c^{(l)}_n.x_n$$

Virtualmente, cualquier función puede ser usada como consecuente.

En la inferencia se procede de la forma usual, con el disparo asociado a cada regla.

La salida defusificada es un promedio de pesos de las salidas individuales de cada regla, viene dado por:

$$y = \sum_{l=1}^{M} \frac{\mu_{A^{(l)}}(x)}{\sum_{k=1}^{M} \mu_{A^{(k)}}(x)} h^{(l)}(x)$$

donde los pesos $\mu_{A^{(l)}}(x)$ son calculados de acuerdo a:

$$\mu_{A^{(l)}}(x) = \prod_{i=1}^{n} \mu_{A_i^{(l)}}(x_i)$$

2.7 IMPLEMENTACION DE UN SISTEMA LÓGICO DIFUSO

El control difuso provee una metodología formal para representar, manipular e implementar el conocimiento heurístico humano sobre como controlar un sistema. Si explotamos el uso de esta información heurística podemos conseguir un mejor rendimiento de los sistemas de control, mediante el diseño de controladores difusos.

El diagrama de bloques de la figura 2.8 muestra el diagrama de bloques de un típico controlador difuso en un sistema de control de lazo cerrado.

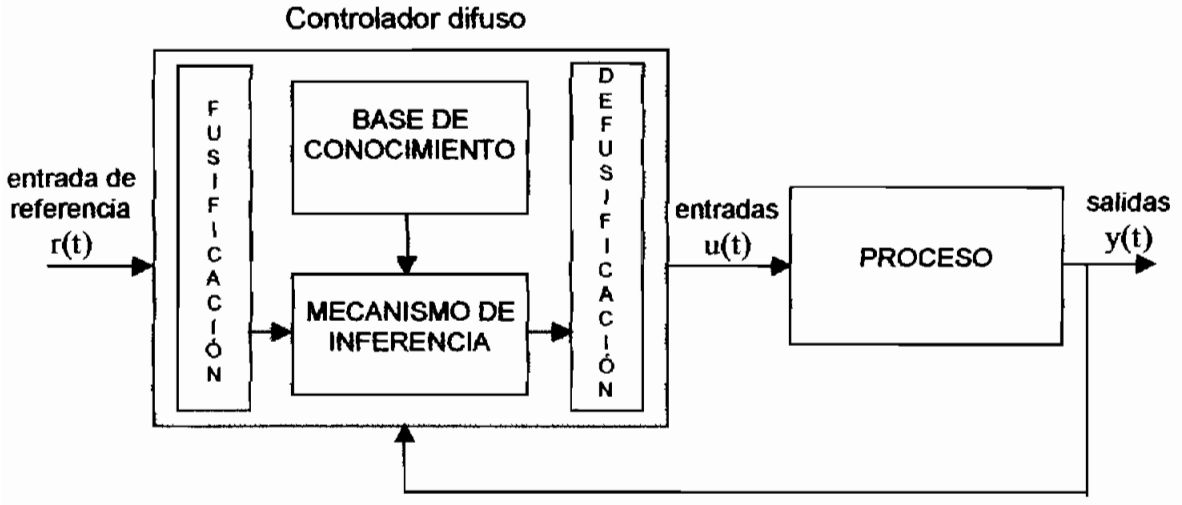


Fig. 2.8 Arquitectura de un controlador difuso

El controlador difuso tiene cuatro componentes básicos:

1. La base de conocimiento, que contiene la experiencia del experto en forma de reglas (conjunto de reglas) que indican cómo controlar mejor el sistema.
2. El mecanismo inferencia, en donde se evalúa que reglas son relevantes en cierto caso y entonces decide cual debe ser la entrada a la planta.
3. La interface de fusificación, modifica las entradas de tal forma que puedan ser interpretadas y comparadas en las reglas de la base de conocimiento.
4. La interface de defusificación, convierte las conclusiones alcanzadas por el mecanismo de inferencia en entradas para la planta.

Ahora se procederá a implementar un FLS sobre la base de los conocimientos adquiridos previamente con el estudio de las primeras unidades utilizando el péndulo invertido como un problema conveniente para ilustrar el diseño y el mecanismo de operación básica de un sistema de control difuso.

El siguiente ejemplo fue tomado de [3]. En la figura 2.9. se ha representado el problema del péndulo invertido sobre un carro.

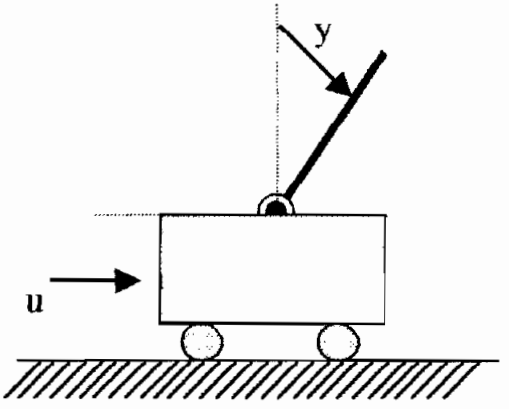


Fig. 2.9 Péndulo invertido sobre un coche

El péndulo puede girar sobre el plano vertical. Se desea mantener la velocidad del coche constante de manera que ésta mantenga el péndulo en posición vertical.

En el gráfico se denota como "y" a la posición angular del péndulo respecto a la posición deseada y "u" a la fuerza que mueve al coche para tratar de equilibrar el péndulo.

Escoger las variables de entrada al controlador es muy importante y depende exclusivamente del experto (diseñador). Se podría elegir como variables de entrada:

$e(t) = r(t) - y(t)$ error que posee el péndulo respecto a la posición deseada r

$\frac{d}{dt}e(t)$ cambio en el error

Así que las variables sobre las cuales se tomará decisiones (variables lingüísticas) son "error" (e) y "cambio en el error" (ė).

La variable de salida del controlador en este caso es la fuerza que mueve el coche, "fuerza" (u).

Puesto que se desea balancear el péndulo verticalmente, la posición deseada será r = 0, por tanto,

$$e = -y$$

y

$$\frac{d}{dt}e = -\frac{d}{dt}y$$

Asignaremos ahora a las variables linguísticas: "error", "cambio en el error" y "fuerza" los siguientes valores lingüísticos o etiquetas:

negrande	(NG)	ó	-2
negpequeño	(NP)	ó	-1
cero	(CE)	ó	0
pospequeño	(PP)	ó	1
posgrande	(PG)	ó	2

Cada etiqueta representa obviamente a un conjunto difuso.

A continuación, cuantificaremos el comportamiento dinámico del péndulo invertido, para esto, es importante comprender muy bien la dinámica del proceso. Cada una de las siguientes afirmaciones cuantifica una situación diferente en la que se encuentra el péndulo:

- La afirmación "error es posgrande" puede representar la situación en la que el péndulo forma un ángulo significativo con la vertical hacia la izquierda.

- La expresión "error es negpequeño" puede representar la situación donde el péndulo está ligeramente separado hacia la derecha de la vertical, pero no tan cerca de la vertical como para justificar cuantificarla como "cero" y no tan lejos que justifique cuantificarla como "negrande".

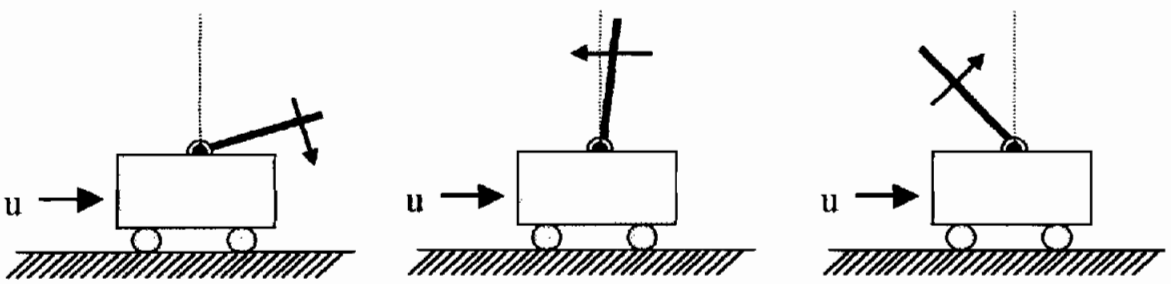
- La afirmación "error es cero" puede representar la situación donde el péndulo está muy cerca de la posición vertical (un cuantificador lingüístico no es

preciso, así que se cuantificará como "cero" a cualquier valor de error alrededor de e(t) = 0).

- La afirmación "error es posgrande y cambio en el error es pospequeño" puede representar la situación donde el péndulo está a la izquierda de la vertical y, ya que dy/dt < 0, el péndulo está alejándose de la posición vertical (note que este caso el péndulo se está moviendo en sentido contrario a las manecillas del reloj).

- La afirmación "error es negpequeño y cambio de error es pospequeño" puede representar la situación donde el péndulo está ligeramente a la derecha de la vertical y, ya que dy/dt < 0 el péndulo está moviéndose hacia la posición vertical (en este caso, el péndulo también está moviéndose en sentido contrario a las manecillas del reloj).

Haciendo uso del conocimiento sobre como controlar la posición del péndulo, se procederá a capturar el conocimiento del experto en las reglas de conocimiento. Iniciaremos con las reglas que representan las tres situaciones de la figura 2.10:



(a) (b) (c)

Fig. 2.9 Péndulo invertido en varias posiciones

$\mathcal{R}^{(1)}$: SI e es NG y é es NG ENTONCES F es PG

La regla cuantifica la situación de la figura 2.10(a) donde el péndulo tiene un ángulo positivo grande y se está moviendo en el sentido de las manecillas del

reloj, así que es claro que se debería aplicar una fuerza positiva grande (hacia la derecha).

$\mathcal{R}^{(2)}$: SI e es CE y ė es PP ENTONCES F es NP

Esta regla cuantifica la situación de la figura 2.10(b) donde el péndulo forma un ángulo muy cercano a cero con la vertical y se está moviendo en contra de las manecillas del reloj, así que deberemos aplicar una fuerza negativa pequeña (hacia la izquierda) para contrarrestar el movimiento de tal forma que se mueva hacia cero.

$\mathcal{R}^{(3)}$: SI e es PG y ė es NP ENTONCES F es NP

La regla cuantifica la situación de la figura 2.10(c) en la cual el péndulo lejos hacia la izquierda de la vertical y se está moviendo en el sentido de las agujas del reloj; así que se debería aplicar una fuerza negativa pequeña (hacia la izquierda), de manera que asista el movimiento.

El número de posibles combinaciones para este ejemplo son 25 reglas (5 variables de entrada y 5 etiquetas por cada una). Presentaremos las reglas de la base de conocimiento en la FAM y se usará números para representar las etiquetas:

		ė				
		-2	**-1**	**0**	**1**	**2**
e	**-2**	2	2	2	1	0
	-1	2	2	1	0	-1
	0	2	1	0	-1	-2
	1	1	0	-1	-2	-2
	2	0	-1	-2	-2	-2

Tabla 2.2 FAM para el péndulo invertido sobre un coche

Note que hay una especie de simetría en el cuerpo de la tabla respecto a los diagonales que la forman. Esta simetría no es accidental pues surge debido a la simetría de la dinámica del sistema.

Escoger las funciones de membresía asociadas a cada conjunto difuso depende del criterio del diseñador, pero como se indicó en las secciones anteriores, la función triangular puede resultar la más adecuada, entonces utilizaremos triángulos simétricos. Ahora especificaremos las funciones de membresía para cada uno de los 15 conjuntos difusos pertenecientes a las variables de entrada "error", "cambio en el error" y a la de salida "fuerza", como se muestran en las figuras 2.11, 2.12 y 2.13 respectivamente:

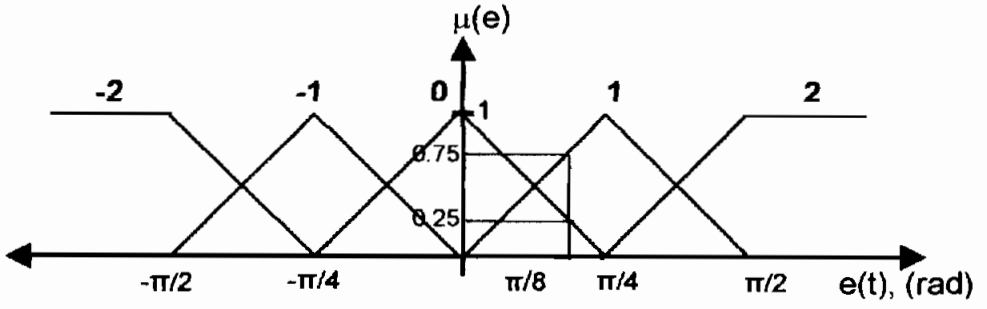


Fig. 2.11 Funciones de membresía de los conjuntos difusos de la entrada e

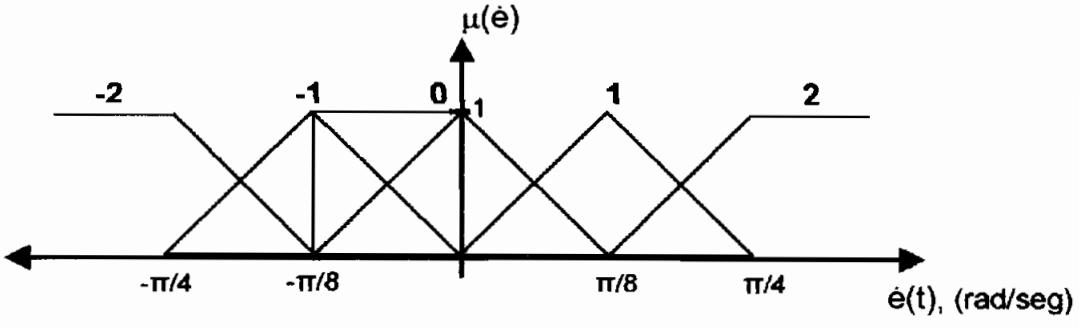


Fig. 2.12 Funciones de membresía de los conjuntos difusos de la entrada ė

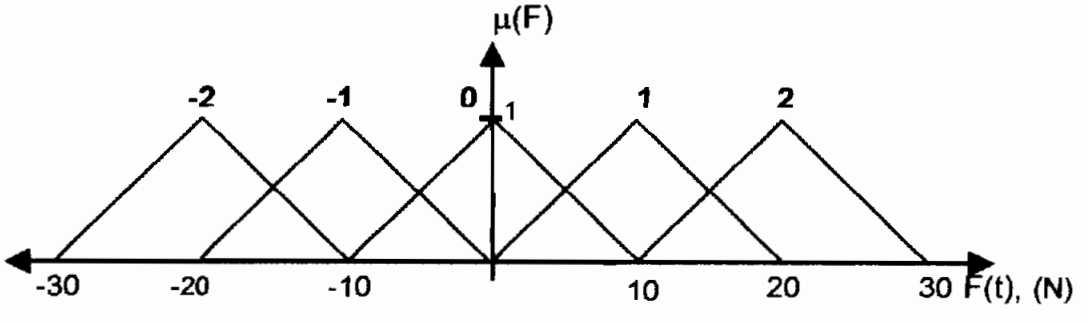


Fig. 2.13 Funciones de membresía de los conjuntos difusos de la salida F

Observe en las figuras 2.11 y 2.12 que las funciones de membresía que se encuentran en los extremos son saturadas, es decir que, a partir de cierto valor del universo de discurso la membresía es siempre 1; esto da la idea de que el experto ha agrupado todos los valores más grandes (en otro caso los más pequeños) en un solo conjunto, este uso de las funciones de los extremos caracteriza la expresión "mayor que", para los del extremo derecho y "menor que", para los extremos izquierdos.

Las reglas de conocimiento del controlador difuso recoge las variables lingüísticas, valores lingüísticos, las funciones de membresía asociadas y el conjunto de todas las reglas lingüísticas (tabla 2.2), así que se ha completado la descripción del péndulo invertido.

Ahora fusificaremos dos datos de entrada.

Si $e = (\pi/4 - \pi/16) = e'$, luego de fusificar se tendrá:

$\mu_{PP}(e') = 0.75$ o lo que es lo mismo $\mu_1(e') = 0.75$

$\mu_{CE}(e') = 0.25$ o $\mu_0(e') = 0.25$

Y, si $\dot{e} = -\pi/8 = \dot{e}'$, el proceso de fusificación da como resultado:

$\mu_{NP}(\dot{e}') = 1$ o $\mu_{-1}(\dot{e}') = 1$

El siguiente paso es buscar en la FAM que reglas se activan para:

$$\mu_0(e') \text{ y } \mu_{-1}(\dot{e}')$$

$$\mu_1(e') \text{ y } \mu_{-1}(\dot{e}')$$

		ė				
		-2	-1	0	1	2
e	-2	2	2	2	1	0
	-1	2	2	1	0	-1
	0	2	1	0	-1	-2
	1	1	0	-1	-2	-2
	2	0	-1	-2	-2	-2

Tabla 2.3 Activación de reglas

Usando el min tanto para las premisas como para el consecuente se tiene:

$\mathcal{R}^{(1)}$: $\mu_{PP^*}(u) = \min\{\min[(\mu_{CE}(e'), \mu_{NP}(\dot{e}')], \mu_{PP}(u)\}$

$= \min\{\min[0.25, 1], \mu_{PP}(u)\}$

$= \min\{0.25 . \mu_{PP}(u)\}$

$\mathcal{R}^{(2)}$: $\mu_{CE^*}(u) = \min\{\min[(\mu_{PP}(e'), \mu_{NP}(\dot{e}')], \mu_{CE}(u)\}$

$= \min\{\min[0.75, 1], \mu_{CE}(u)\}$

$= \min\{0.75 . \mu_{CE}(u)\}$

Gráficamente:

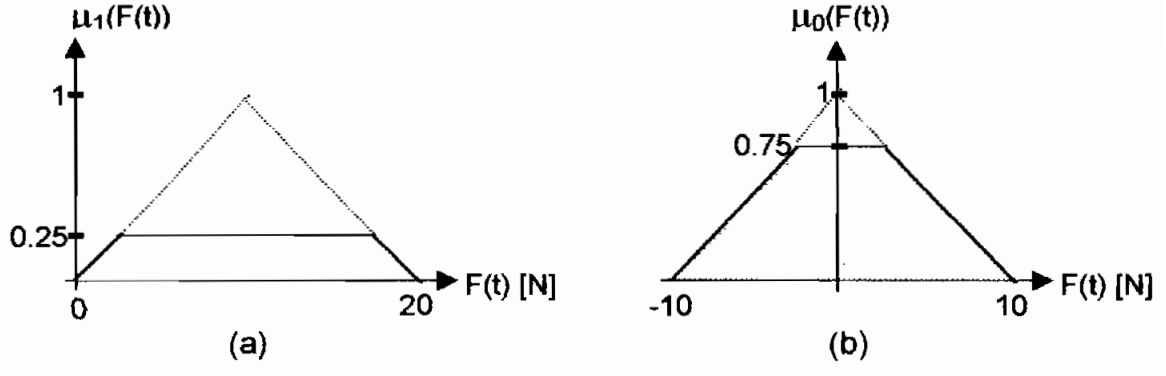


Fig. 2.13 Funciones de membresía de los conjuntos implicados (a) "pospequeño", (b) "cero"

El conjunto difuso implicado final se muestra en la figura 2.14.

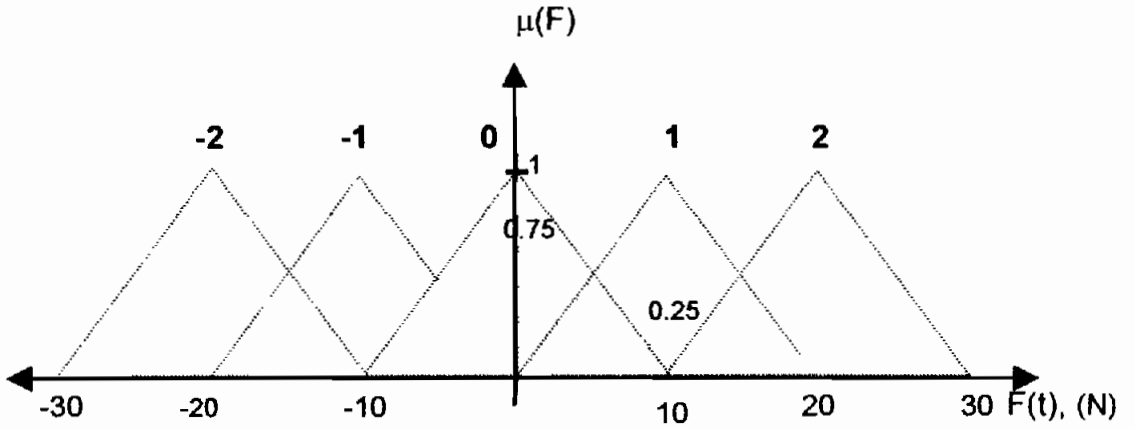


Fig. 2.14 Conjunto difuso implicado

Para defusificar el conjunto difuso final se utilizará en este caso el método del centro de gravedad.

$$\text{Area1} = b\left(h - \frac{h^2}{2}\right) = 20\left(0.75 - \frac{0.75^2}{2}\right) = 9.375$$

$$\text{Area2} = 20\left(0.25 - \frac{0.25^2}{2}\right) = 4.375$$

$$y = \frac{(0)(9.375) + (10)(4.375)}{9.375 + 4.375} = 3.18$$

Entonces, para la posición en la cual: e(t) = (π/4 - π/16) = (33.75°) y ė(t) = -π/8 = (-22.5), el péndulo está ligeramente hacia la izquierda de la vertical y se está moviendo en el sentido de las manecillas del reloj, la fuerza que se le aplica al coche es de 3.18 **[N]**.

3. TALLER

1. Diseñe el sistema lógico difuso para el péndulo invertido usando funciones de membresía gaussianas; y, encuentre la fuerza que se debe aplicar:

a. para $e(t) = \pi/2$ y $\dot{e}(t) = \pi/4$, usando el método de defusificación del centro promedio, del centro de gravedad y máximo.
b. para $e(t) = -\pi/2$ y $\dot{e}(t) = \pi/4$, usando el método de defusificación del centro promedio, centro de gravedad y máximo.

2. Diseñe el sistema lógico difuso para el péndulo invertido usando funciones de membresía singleton; y, encuentre la fuerza que se debe aplicar:
   a. para $e(t) = \pi/2$ y $\dot{e}(t) = \pi/4$, usando el método de defusificación máximo.
   b. para $e(t) = -\pi/2$ y $\dot{e}(t) = \pi/4$, usando el método de defusificación para las funciones singleton.

3. Diseñe el sistema lógico difuso para el péndulo invertido usando siete funciones de membresía para cada universo de discurso distribuyéndolas uniformemente. Use funciones triangulares y defusifique siguiendo las condiciones (a) y (b) del ejercicio 1.

4. Estudie los efectos de añadir más conjuntos difusos y por tanto más reglas al sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. Jantzen Jan, **"Tutorial of Fuzzy Logic"**, Universidad Técnica Denmark.

2. Mendel Jerry, **"Fuzzy Logic Systems for Ingineering: A Tutorial"**, IEEE.

3. Passino Kevin, **"Fuzzy Control"**, Adisson - Wesley.

4. Sánchez Alberto, **"Simulación gráfica en Windows de un elevador empleando técnicas de control difuso"**, Tesis, Escuela Politécnica Nacional.

5. **"Fuzzy Logic Toolbox User's Guide"** , The MathWorks, Inc.

MANUAL DE USUARIO DEL TOOLBOX DE SISTEMAS LÓGICOS DIFUSOS DE MATLAB

1. OBJETIVOS

- Presentar al alumno en forma sencilla los utilitarios del toolbox de sistemas difusos que presenta MATLAB.

- Familiarizar al alumno con la herramienta computacional que le permitirá realizar diseños de sistemas lógicos difusos.

2. MARCO TEÓRICO

El toolbox de Matlab FUZZY permite trabajar con dos tipos de inferencias: de Mamdani y de Sugeno.

Para crear un sistema difuso, Matlab presenta tres pantallas diferentes:
- Editor del sistema de inferencia difuso
- Editor de las funciones de membresía
- Editor de las reglas de conocimiento

Cuando se ha escogido el sistema de inferencia Sugeno, se habilita otra pantalla de edición:
Editor de anfis (adaptive neuro-fuzzy inference system)

Además incluye pantallas de visualización:
- Ver reglas
- Ver superficie de control

2.1 EDITOR DEL SISTEMA DE INFERENCIA DIFUSO

La primera pantalla en presentarse es la del editor del sistema de inferencia (figura 2.1).

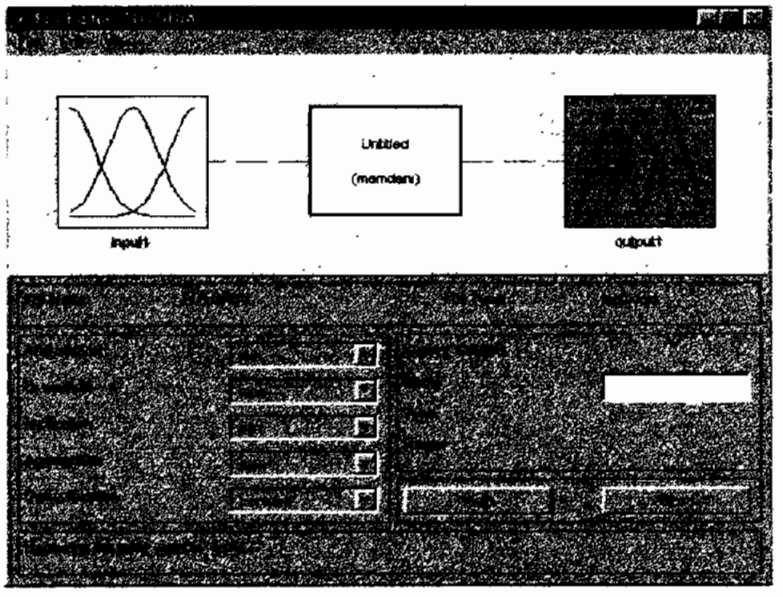


Fig. 2.1 Pantalla de edición del sistema de inferencia FIS

Lo primero que se debe escoger es, con que sistema de inferencia se trabajará, para esto, en el menú **File** se encuentran ambas opciones: *new mamdani FIS*, *new sugeno FIS* (fuzzy inference system).

Utilizaremos el diseño del controlador del nivel de un tanque, realizado en Matlab.

Se escogerá la inferencia de mamdani, ahora se puede iniciar el diseño del sistema.

En el menú **Edit** se encuentra las opciones añadir entradas, añadir salidas o eliminarlas. Para este ejemplo se tiene dos variables de entrada: level (nivel) y rate (cambio), por tanto será necesario añadir una. Se cuenta con solo una variable de salida llamada valve (válvula).

Se ofrece una serie de opciones definidas a continuación:

✓ AND METHOD: conector conjunción. Tipo de operación min o prod.

- ✓ OR METHOD: conector disyunción. Tipo de operación max o probor (que se calcula prob(a,b) = a+b -ab).
- ✓ IMPLICATION: define el tipo de implicación, presenta dos opciones min o prod.
- ✓ AGREGGATION: es el proceso por el cual la salida de cada regla es combinada para formar un único conjunto difuso de salida, pueden ser: max, prob o sum.
- ✓ DEFUZZIFICATION: ofrece varios métodos de defusificación: centroide (centro de gravedad), bisector, mom, lom y som, los tres últimos son variaciones del defusificador máximo.

Para el ejemplo se adoptará:

AND METHOD: prod

OR METHOD: probor

IMPLICATION: prod

AGREGGATION: max

DEFUZZIFICATION: centroide (centro de gravedad)

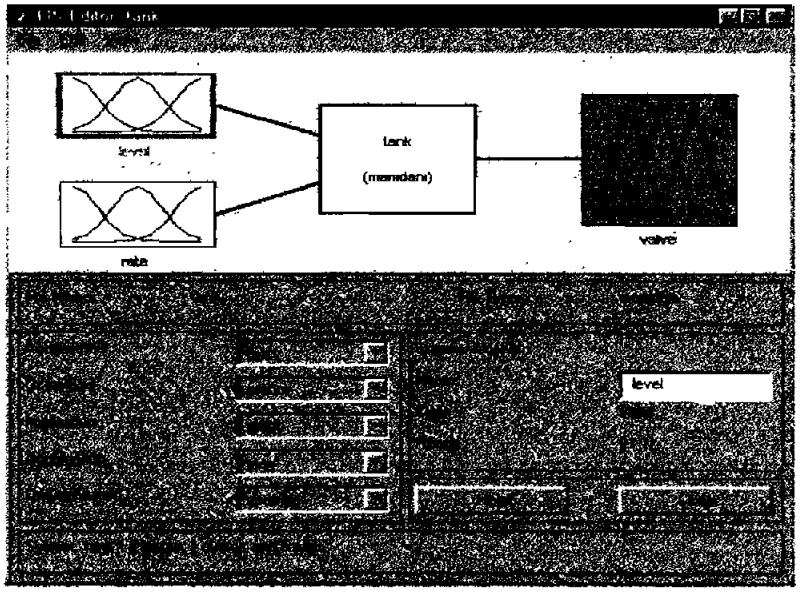


Fig. 2.2 Selección y edición en la pantalla del FIS

Para definir los nombres de las variables seleccionamos con el cursor una entrada a la vez hasta que se enmarque en rojo y en NAME escribimos el nombre como se indica en la figura 2.2.

Escogiendo desde el menú **View:** *Edit memberships functions* o dando doble click con el cursor sobre el gráfico de una de las variables, se abre la siguiente pantalla (figura 2.3).

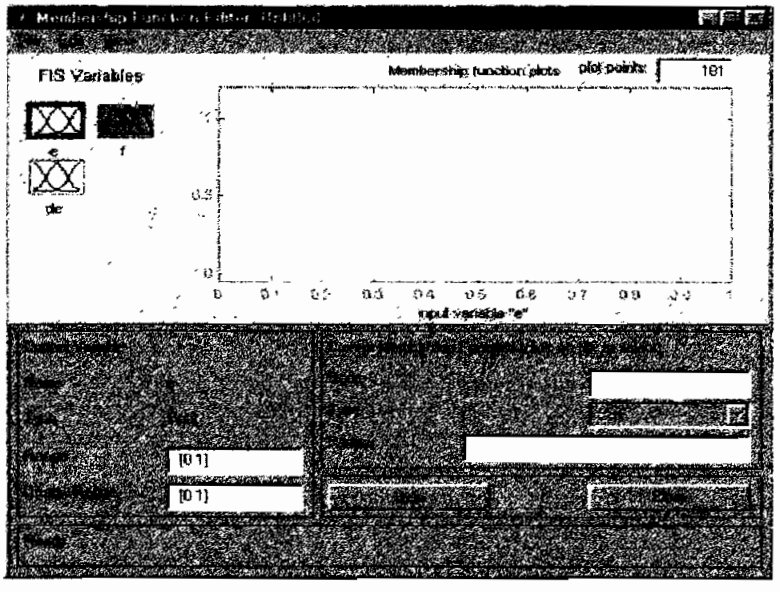


Fig. 2.3 Pantalla de edición de las funciones de membresía

Para añadir funciones de membresía, en el menú **Edit** la opción add MFs despliega la pantalla de la figura 2.4, en donde se indicará la forma y el número de la función.

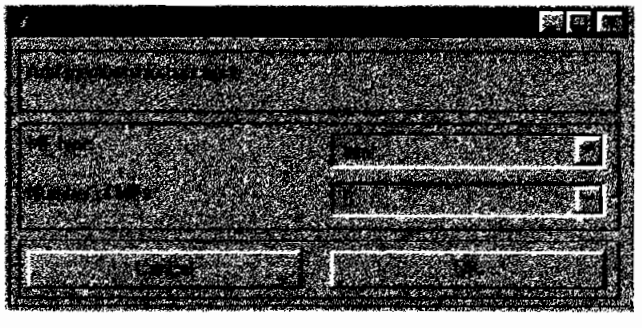


Fig. 2.4 Añadir funciones de membresía

Luego de definido el número y forma de las funciones, se indica el rango o universo de discurso para la variable de entrada, y escogiendo con el mouse cada uno de los conjuntos difusos se le da un nombre, forma individual y los límites de acuerdo a la figura que se ha escogido.

Se observa en la figura 2.5, que para la variable error se crearon 3 MFs gaussianas.

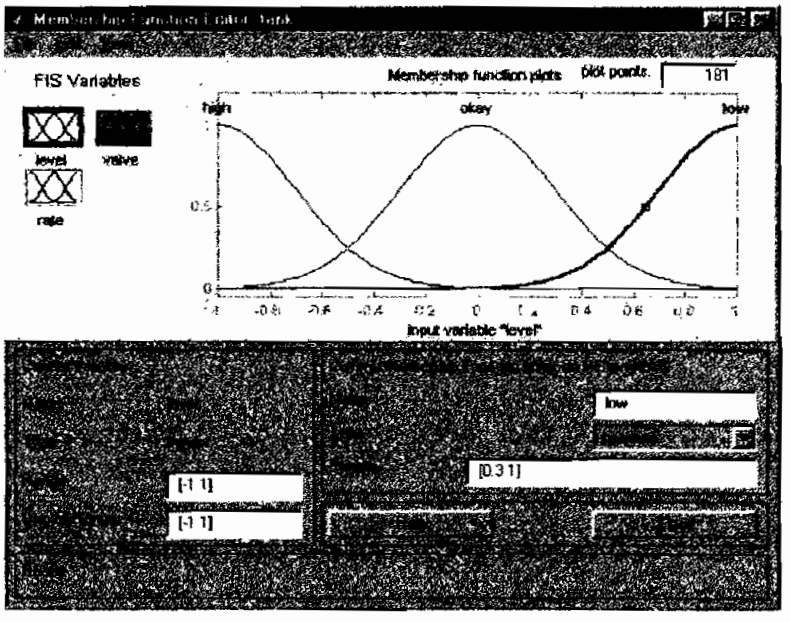


Fig. 2.5 Funciones de membresía de la variable de entrada e

Se sigue el mismo procedimiento para todas las variables.

Ahora, se procede a editar las reglas de conocimiento. En **View** se escoge *Edit rules*, enseguida se despliega la pantalla mostrada en la figura 2.6.

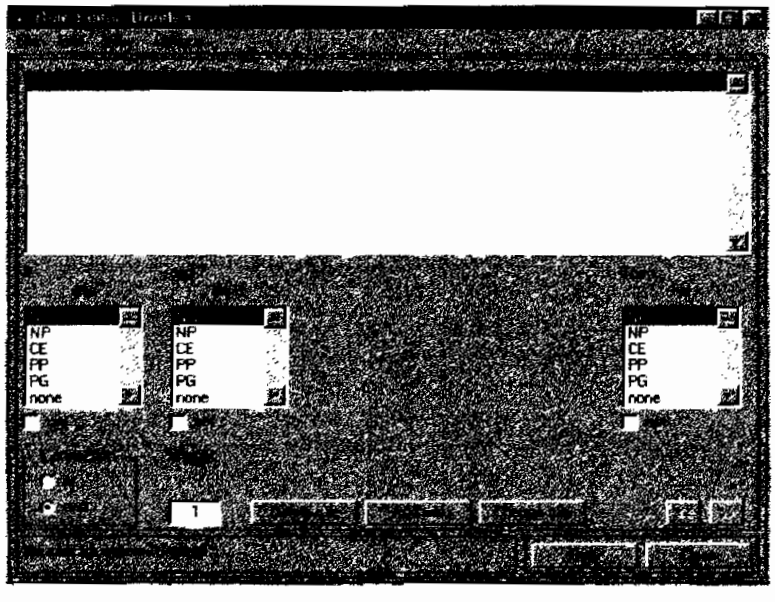


Fig. 2.6 Edición de las reglas de la base de conocimiento

Para formar cada una de las reglas, se procede a señalar una de cada grupo según sea la regla, se escoge la forma de conexión (y, o) e incluso se puede negarlas. Enseguida se presiona *Add rule* e inmediatamente aparecerá en la pantalla la regla con el formato;

If (level is okay) and (rate is negative) then (valve is open_slow) (1)

Aunque este formato podría variar dependiendo de la presentación que se desee al escoger en **Options** la opción *Formats*.

El número entre el paréntesis al final de la regla, indica el peso (weight) asociado a ésta, el cual es aplicado al número dado por el antecedente, puede ser cualquier número entre 0 y 1, pero por defecto será 1, lo que indica que no afecta en nada a las reglas.

A continuación, en la figura 2.7 se muestran las 5 reglas que forman la base de conocimiento del sistema que controla el nivel del tanque.

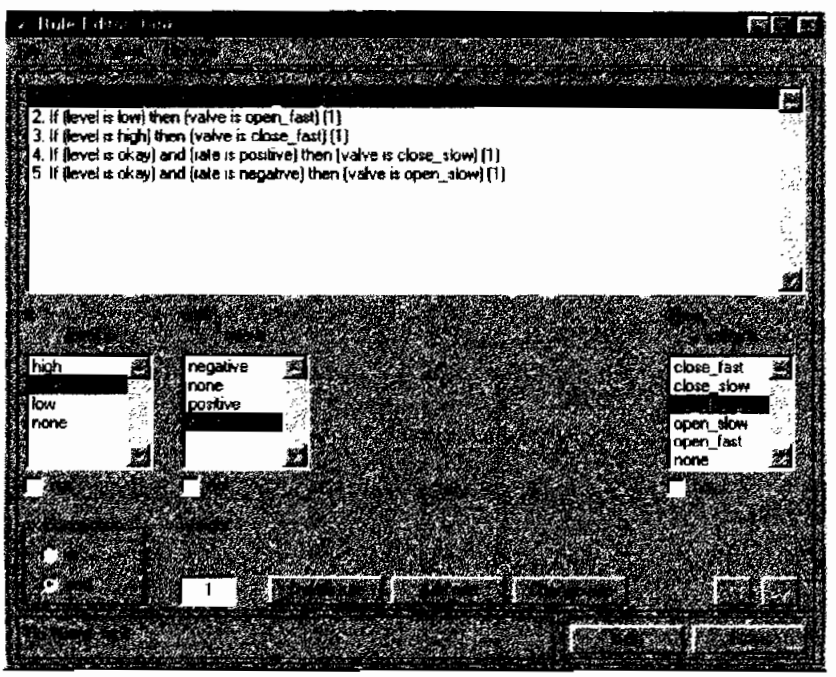


Fig. 2.7 Base de conocimiento del péndulo invertido con 25 reglas

Con este sencillo procedimiento se ha terminado el diseño del sistema lógico difuso.

Para observar como trabaja el sistema, nos dirigimos nuevamente al menú **View** y tomamos la opción *View rules*, pantalla que se indica en la figura 2.8.

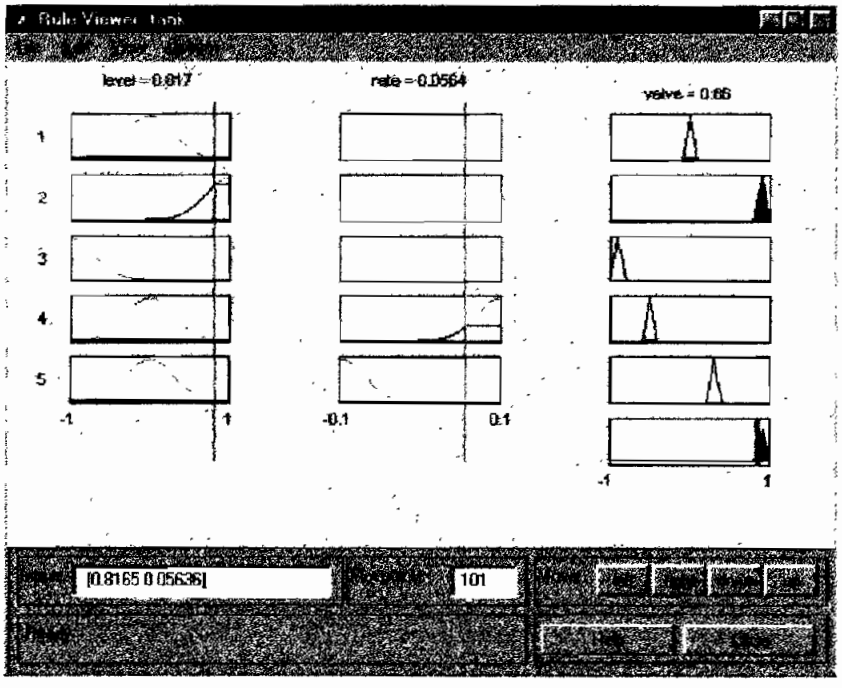


Fig. 2.8 Revisión de reglas y resultados

Las dos columnas de la izquierda, muestran los conjuntos difusos de las variables de entrada; y la columna de la derecha, los de la salida.

Las funciones de membresía coloreadas con amarillo, son aquellas que intervendrán en el proceso de inferencia, y la medida de coloración indica el grado de membresía; mientras que en la columna de la derecha, las funciones coloreadas con azul indican que reglas y en que medida se han activado, es decir el resultado del proceso de inferencia.

En *Input* se puede escribir el valor de entrada de cada variable que será fusificado, en este caso, level = 0.817 y rate = 0.0563.

El gráfico final de la columna derecha indica el conjunto inferido final, de donde se obtiene el valor crisp de salida.

En la parte superior de las gráficas de entrada se observa los valores a fusificar y en la de salida está el resultado de aplicar la defusificación, en nuestro caso centroide, que da como resultado una señal de control valve = 0.86.

El resto de opciones desplazan los gráficos hacia la derecha, izquierda, etc.

Por último, se puede examinar la superficie de control eligiendo en **View** la opción *View surface*. La superficie resultante para este ejercicio se muestra en la figura 2.9.

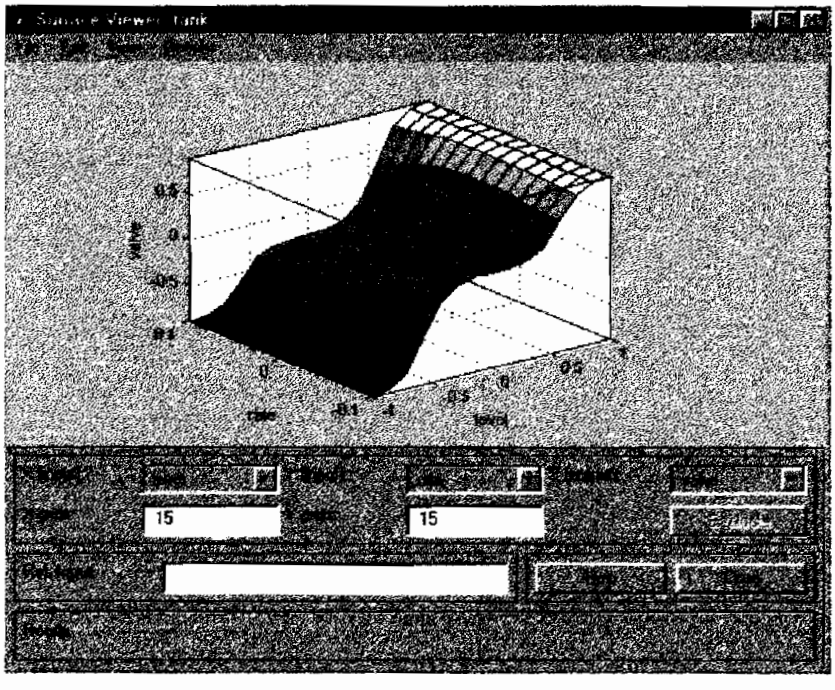


Fig. 2.9 Superficie de control del sistema de nivel del tanque

Para trabajar con un sistema tipo Takagi-Sugeno se trabaja de la misma manera que para el sistema tipo Mamdani. Las únicas distinciones son el hecho que todas las funciones de membresía de salida son singleton, y la implicación y métodos de agregación son fijos y no pueden revisarse, así se observa en la

figura 2.10. El método de implicación es la multiplicación (prod), y el operador de agregación incluye todas las funciones singleton.

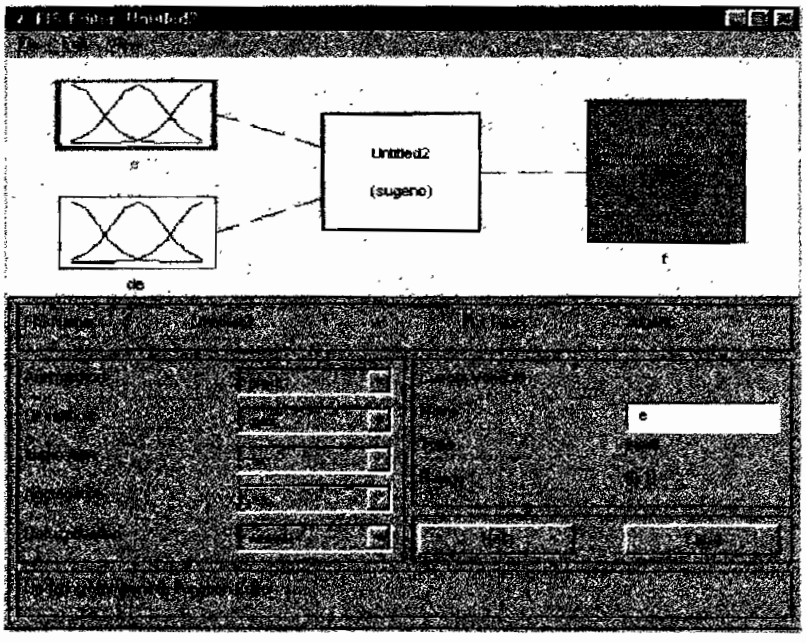


Fig. 2.10 Editor FIS de un sistema sugeno

El toolbox de Matlab permite modelar sistemas tipo sugeno solamente de orden cero (contantes) y uno (lineal), entonces, al dirigirse al editor de MFs se presenta la siguiente pantalla (figura 2.11).

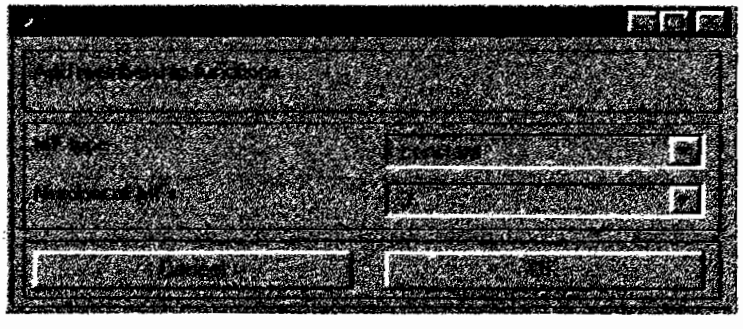


Fig. 2.11 Añadir funciones de membresía de salida en un sistema sugeno

Escogiendo la salida constante, la presentación de las funciones es la siguiente:

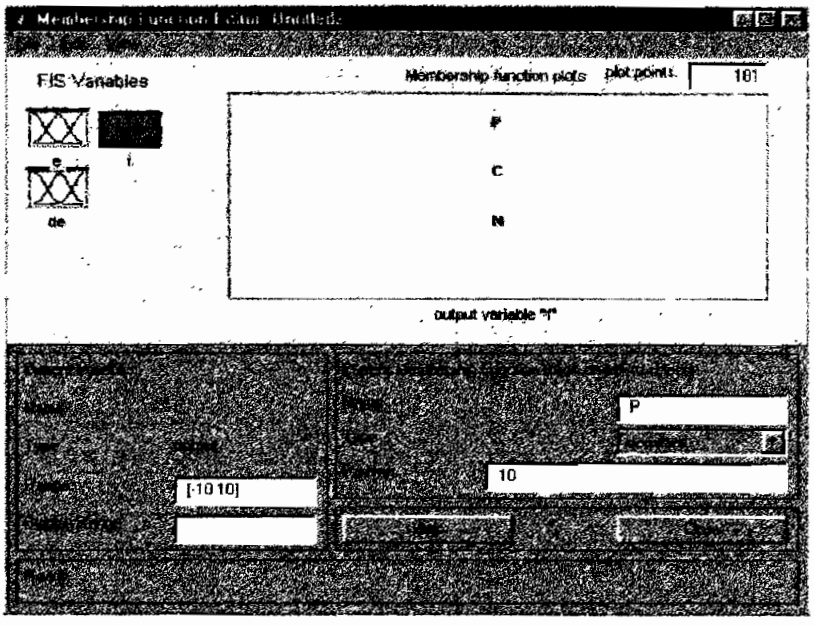


Fig. 2.12 Funciones de membresía de salida constantes del sistema tipo sugeno

Para dos variables de entrada, e y de, con tres funciones de membresía triangulares cada una, y una variable de salida f, se generan nueve reglas de conocimiento. Al revisar el resultado de la inferencia en el visualizador de reglas se tiene (figura 2.13):

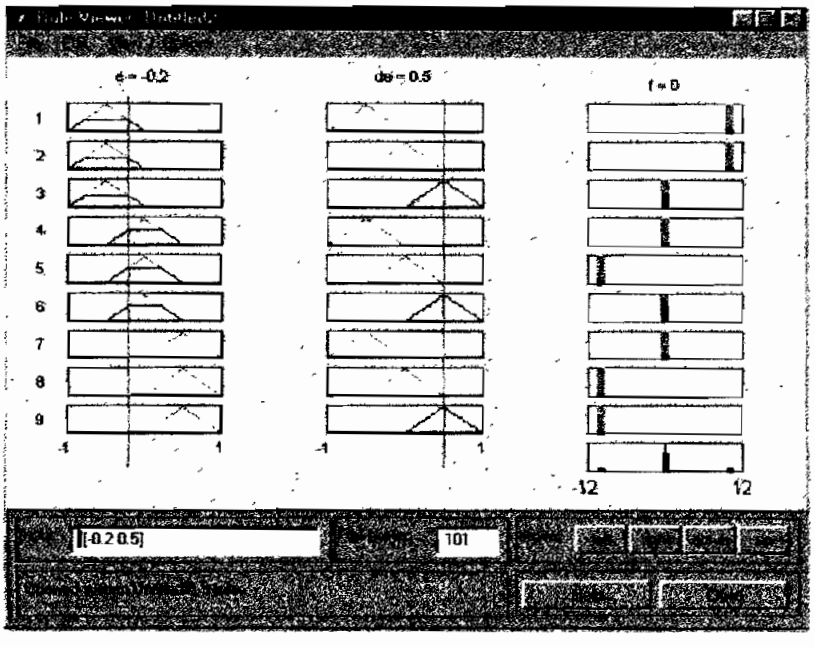


Fig. 2.13 Inferencia del sistema tipo sugeno

Como se observa, las salidas son singleton y la coloración en azul indica el grado de membresía resultado de la implicación.

Cuando no es posible modelar una sistema difuso de acuerdo a las variables que describen la planta, pero contamos con una colección de datos de entrada/salida, la función ANFIS del toolbox construye un sistema de inferencia difuso (FIS) cuyos parámetros de la función de membresía son afinados (ajustados) usando un algoritmo de retropropagación exclusivamente, o en combinación con un método del tipo mínimos cuadrados. Esto permite a los sistemas difusos aprender de los datos que están modelando.

La sigla ANFIS deriva su nombre del sistema de inferencia neuro-difuso adaptivo.

El toolbox de Matlab presenta la pantalla que se muestra en la figura 2.14, especial para realizar este entrenamiento, pero sólo para sistemas del tipo sugeno, si se ingresa en el menú **View** a *Edit Anfis*.

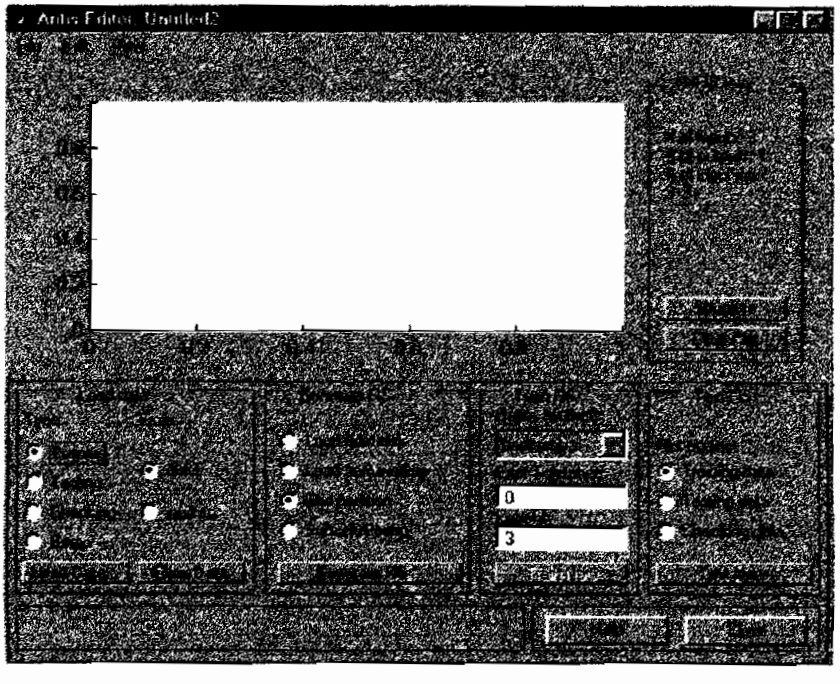


Fig. 2.14 Editor Anfis

La pantalla consta de cuatro menús claramente definidos, y el área para gráficos.

- LOAD DATA: Carga los datos, entrenando, probando, y verificando según se vaya seleccionando los botones apropiados pertenecientes a esta columna. Los datos cargados se trazan en la región destinada a gráficos.

- GENERATE FIS: Genera un nuevo modelo FIS o carga un modelo FIS existente.

- TRAIN FIS: Aquí se escogerá los parámetros del método de optimización para modelar el FIS: retropropagación o una mezcla de retropropagación y mínimos cuadrados (método híbrido), además del número de épocas y la tolerancia de error.

  Presionando el botón *Train Now* empezará el entrenamiento, ajustando los parámetros de la función de membresía a medida que grafica los resultados y el error en el área correspondiente.

- TEST FIS: Dibuja la salida del FIS modelado versus los datos de salida de entrenamiento, de prueba, o de comprobación, seleccionando el botón *Test Now*.

Para observar gráficamente la estructura FIS modelada, una vez que un FIS ha sido generado o se ha cargado, se selecciona el botón *Structure*. Aparecerá algo similar a lo presentado en la figura 2.15.

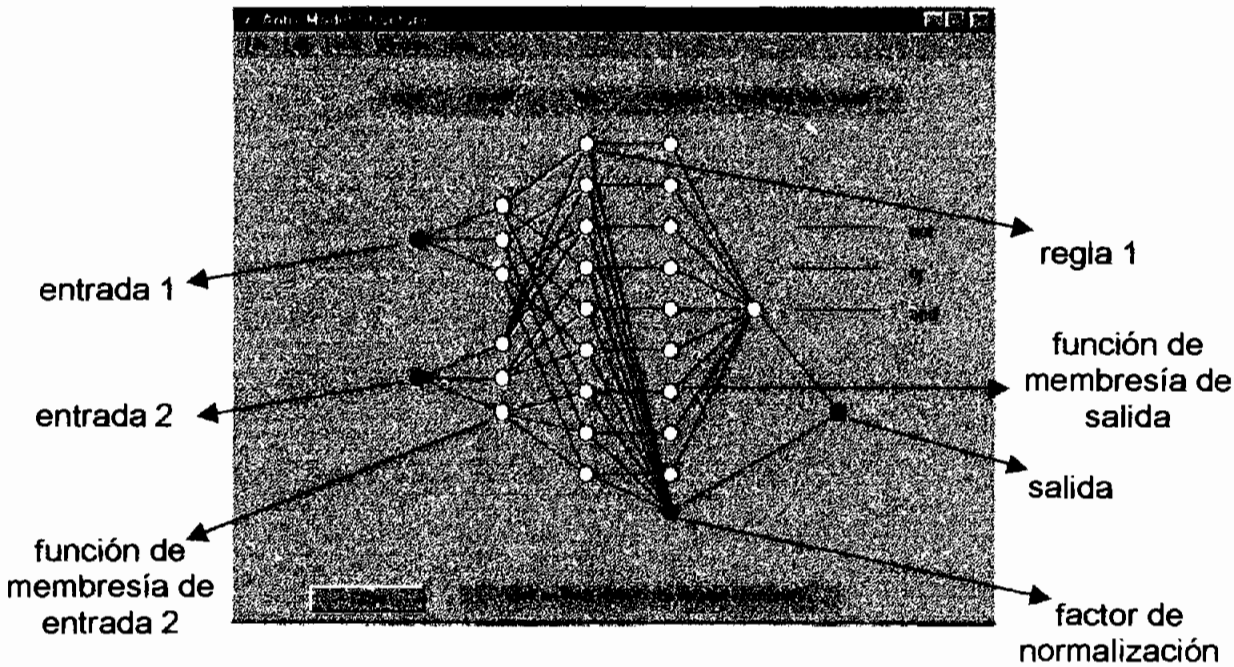


Fig. 2.15 Estructura de una FIS

Es una estructura tipo red, similar a una red neuronal, la cual proyecta las entradas a través de las funciones de membresía de las entradas y los parámetros asociados, pasa por las reglas y, llega a las salidas a través de las funciones de membresía y parámetros asociados a las salidas. Puede usarse para interpretar el mapa de entrada/salida.

3. TALLER

Usando el toolbox de Matlab, implemente sistemas diseñados en el taller de la unidad anterior y compare los resultados. Analice las diferencias si existen.

4. PROYECTOS

En base a los conocimientos adquiridos, diseñe los controladores difusos en Matlab que permitan:

- Realizar el control de velocidad del Motomatic
- Controlar del nivel de un tanque

CAPITULO III

DESARROLLO DEL SOFTWARE DE SOPORTE PARA EL MÓDULO

El software que se desarrollará en este capítulo pretende constituirse en una herramienta capaz de introducir al alumno en el diseño de los sistemas inteligentes.

Es importante aclarar, que el software implementado a través de las herramientas que brinda Matlab corresponde a las Redes Neuronales Artificiales, pues se pensó que era posible ofrecer al estudiante una interface que le permitiera desarrollar sus aplicaciones en una forma amigable, ya que Matlab provee los comandos necesarios en su toolbox, pero realizar un diseño puede convertirse en una tarea larga y tediosa. No se hizo lo mismo respecto a los Sistemas Lógicos Difusos, puesto que Matlab posee una poderosa interface gráfica para el usuario GUI, que maneja el toolbox de los Sistemas de Lógica Difusa, convirtiendo el diseño de un sistema en una tarea sencilla y además didáctica.

Sin embargo, como el objetivo de este proyecto es tratar el Control Inteligente (Redes Neuronales Artificiales y Sistemas de Lógica Difusa) en forma unificada, este programa une ambas interfaces, la perteneciente a Matlab y la implementada especialmente para este trabajo.

En el capítulo segundo, se expuso a manera de manual de usuario la forma en que el alumno podría manejar la GUI de Matlab, ahora, se procederá a describir que contiene y como fue realizado este software muy sencillo de utilizar.

3.1 REQUERIMIENTOS PARA EL SOFTWARE

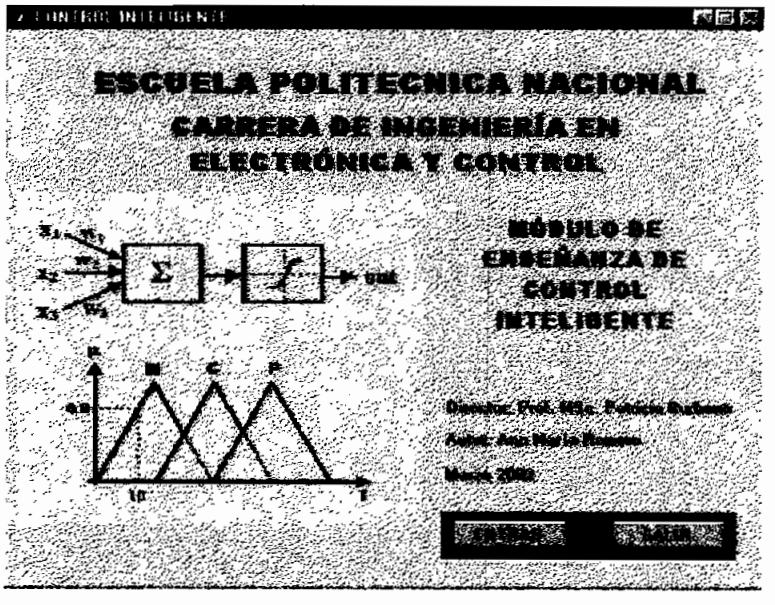
Se empezará por especificar las características que deberá poseer el equipo en donde se instalará el programa.

Los requerimientos del hardware son los siguientes:

- Procesador 586, PENTIUM (I, II, III), CELERON
- Windows 95, 98, NT o Milenium
- Capacidad de memoria RAM de 28 Mb o mayor
- Espacio libre en el disco duro de 500 Mb
- Monitor VGA, SVGA o superior
- Floppy 31/2
- Unidad CD - ROM

3.2 ESTRUCTURA DEL SOFTWARE

La pantalla de presentación del programa SINTEL es la siguiente:



Pantalla de presentación del programa SINTEL

Luego de la presentación se despliega la pantalla del primer menú.

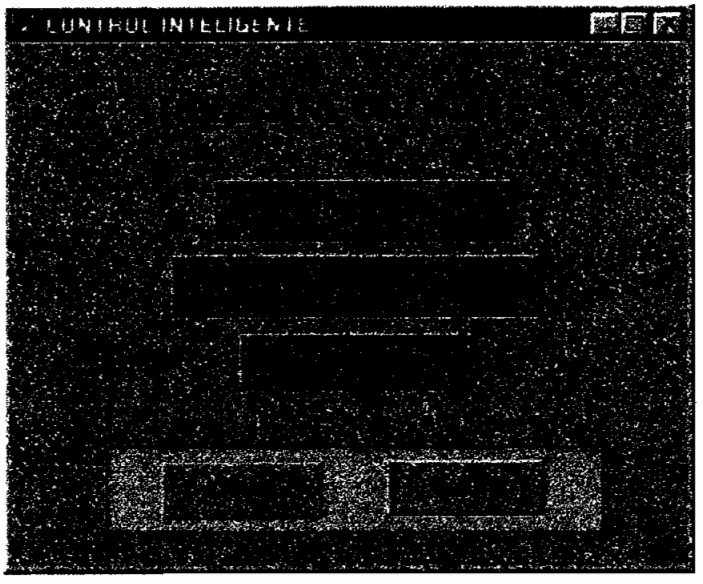


Fig. 3.1 Menú principal

El menú de la figura 3.1 presenta dos opciones. Presionando el botón de Sistemas de Lógica Difusa inmediatamente se conecta con la interface GUI perteneciente al Matlab. Si por el contrario, se escoge la opción Redes Neuronales, se accesa al menú presentado en la figura 3.2.

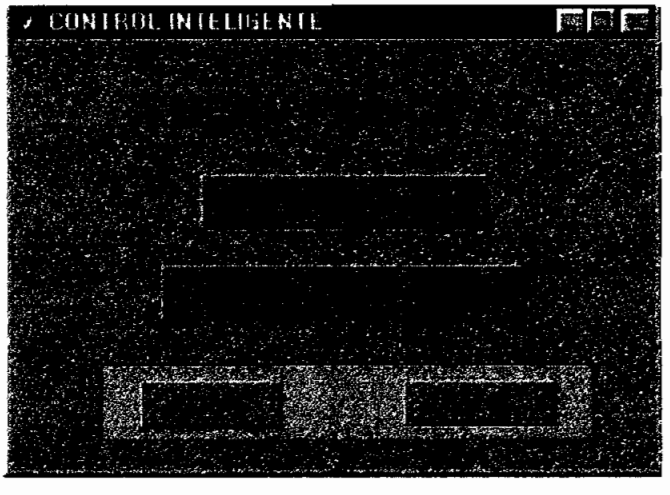


Fig. 3.2 Menú Redes Neuronales Artificiales

El menú redes neuronales presenta tres alternativas, detalladas a continuación.

3.2.1 NEURONAS ARTIFICIALES

En Neuronas Artificiales se diseñó una pantalla que esquematiza de manera didáctica una red neuronal artificial con dos entradas. La pantalla se observa en la figura 3.3.

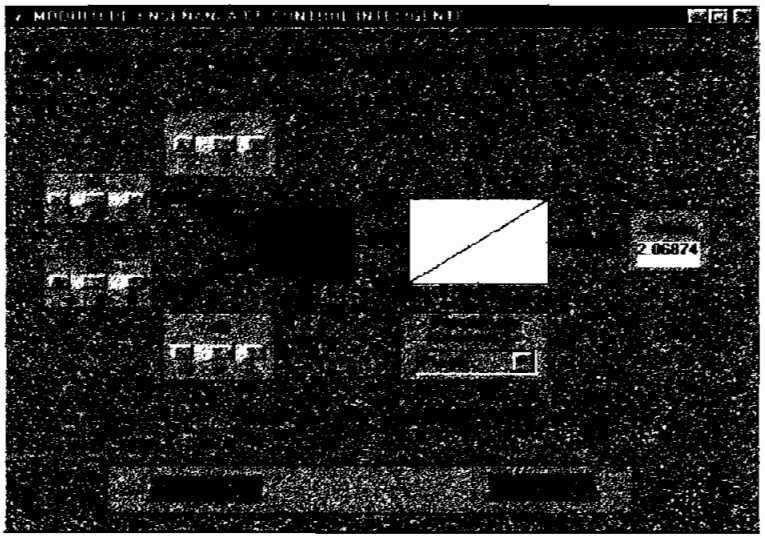


Fig. 3.3 Modelo de una neurona artificial

En esta presentación didáctica, el alumno puede observar el cambio en la salida de la neurona a medida que perturba las entradas, los pesos y la función de transferencia. Se ha dispuesto dos entradas de rango -5 a 5 y -3 a 3. Así mismo, los pesos pueden ser variados en un rango de -2 a 2. La bia o tendencia b no es modificable y asume un valor randómico entre cero y uno. Variando la función de activación o transferencia, el alumno puede verificar lo dicho en la teoría, esto es, los límites dentro de los cuales se mueven los valores de salida. Se ha tomado en cuenta las funciones que se usan comúnmente: hardlim, hardlims, satlin, satlins, purelin, logsig y tansig.

3.2.2 REDES NEURONALES Y SU ENTRENAMIENTO

Al hacer la elección Redes Neuronales y su Entrenamiento, se presenta un nuevo menú.

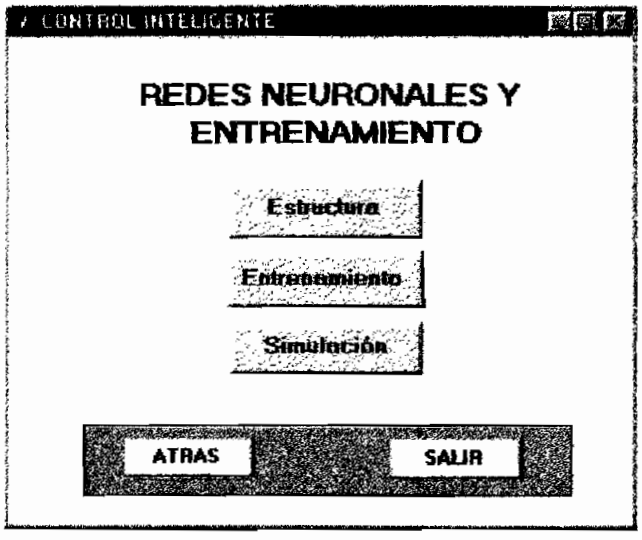


Fig. 3.4 Menú Redes Neuronales y su Entrenamiento

La figura 3.4 exhibe ahora tres alternativas.

3.2.2.1 ESTRUCTURA

Esta opción permite al alumno diseñar una red, configurando desde las entradas hasta las salidas. La figura 3.5 muestra el menú de configuración.

Fig. 3.5 Menú para configurar una red

- **Entradas**

Antes de proceder a configurar las entradas, se despliega una pantalla informativa, en donde se explica en forma corta que significa cada parámetro.

Los parámetros son los siguientes:

- Número de entradas. Permite especificar el número de vectores de entrada que poseerá la red. Para esta aplicación se ha limitado el número de vectores a tres.
- Número de elementos. Indica cuantos elementos poseerá cada vector de entrada. Cada vector podrá tener un máximo de cinco elementos. Defina con cero (0) las entradas de los vectores que no existen o no ha definido.
- Rango de valores. Define los valores máximo y mínimo de los elementos del vector. Deje en blanco los rangos de los vectores no definidos.

En la figura 3.6 se presenta la pantalla informativa.

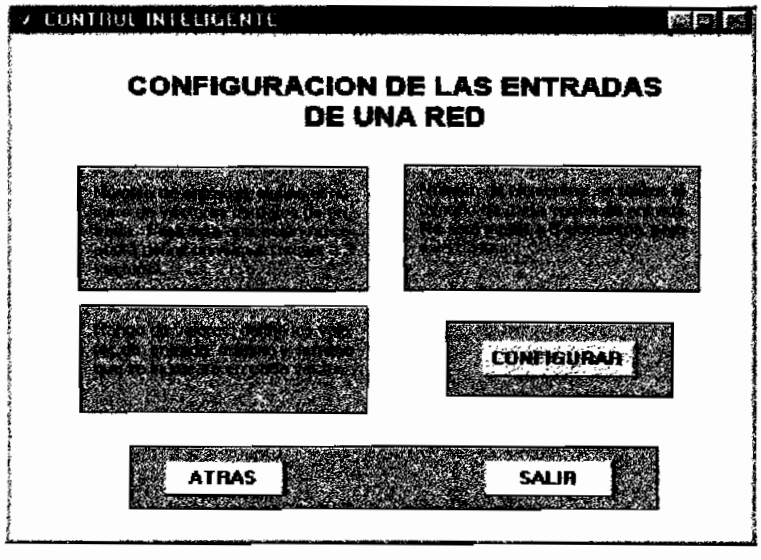


Fig. 3.6 Configuración de las entradas. Pantalla informativa

Al presionar el botón CONFIGURAR, se pasa a la definir los parámetros descritos. La pantalla (figura 3.7) es la siguiente:

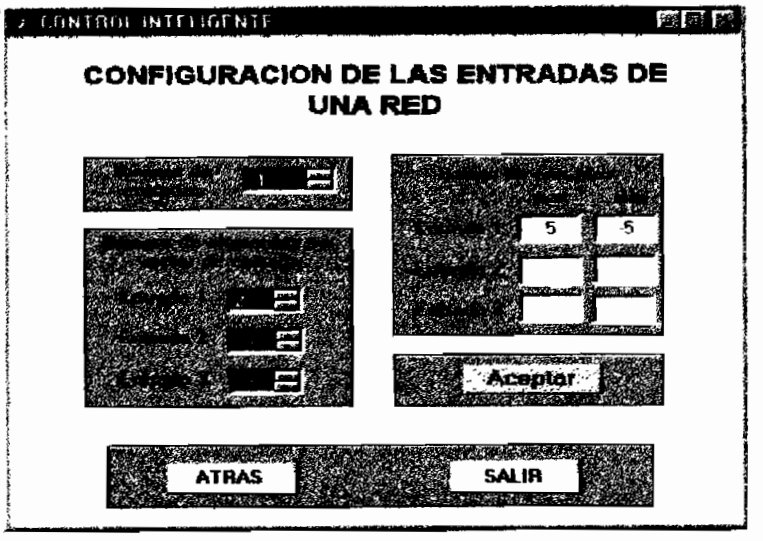


Fig. 3.7 Configuración de las entradas de la red

Escoja el número correspondiente y de un "click" sobre él con el mouse para aceptar. Cuando ha terminado de configurar todos los parámetros, presione ACEPTAR para volver al menú de configuración de la figura 3.5.

- **Capas**

Antes de configurar las capas, se presenta nuevamente una pantalla informativa, figura 3.8, que explica cada parámetro.

Fig. 3.8 Pantalla informativa sobre los parámetros para configurar las capas

- Número de capas. Define el número de capas que compondrán la red. Para esta aplicación, el número máximo de capas es tres, e incluye la capa de salida. Este número suele ser suficiente para alcanzar un objetivo.
- Número de neuronas por capa. Aquí se precisará el número de neuronas que poseerá cada capa. No hay un límite para el número máximo de neuronas, sin embargo, debe recordar que un número muy grande no siempre redunda en eficiencia y vuelve los cálculos muy largos . Escriba cero (0) si esa capa no ha sido definida.
- Función de transferencia. El usuario puede definir capa por capa la función de activación que necesite o desee. Se ofrece las siete funciones más comunes. Si la capa no se ha definido, escoja <<ninguna>>.
- Bias o tendencias. Especifica si se ubicará una bia y en qué capa. Esto es opcional y se constituye en una especie de umbral. Las bias al igual que los pesos son entrenables.

Presionando el botón CONFIGURAR, se tiene la pantalla de configuración, como se expone en la figura 3.9.

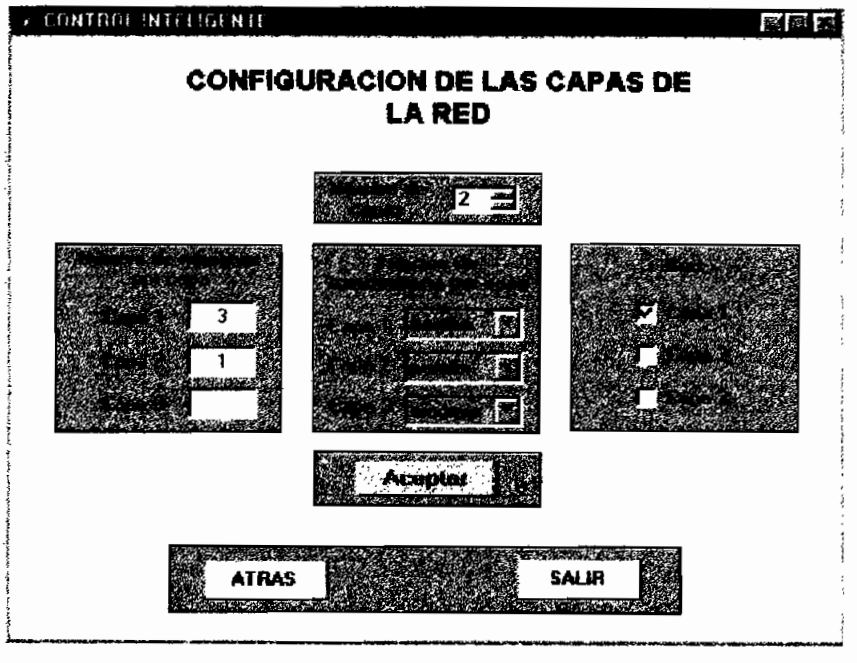


Fig. 3.9 Configuración de las capas de la red

El número de salidas viene dado automáticamente al escoger el número de neuronas de la capa de salida. Su determinación dependerá de la aplicación que se le de a la red. No será necesario escribir nada en caso de no existiera determinada capa.

Al ACEPTAR, se retornará al menú de la figura 3.5.

- **Conexiones**

Especificar las conexiones de la red es muy importante, porque define la organización que observará cada elemento que conforma la red. El usuario deberá tener clara dicha disposición, entendiendo cada paso a medida que avanza la configuración.
Nuevamente, se presenta la pantalla informativa que se visualiza en la figura 3.10.

Fig. 3.10 Pantalla informativa sobre la configuración de las conexiones de la red

- Conexión desde las entradas hacia las capas. El usuario precisará si el i-ésimo vector de entrada es conectado a la j - ésima capa. Un vector puede

conectarse a más de una capa a la vez; así mismo, una capa puede tener más de un vector de entrada.

- Conexión entre capas intermedias. Determina las conexiones entre capas. Una capa puede conectarse a sí misma, realizando de esta forma una realimentación.
- Conexión de salidas y objetivos. Conectar una salida significa que esa capa va a entregar un resultado, que puede tomarse para algún fin. Un objetivo, determina una comparación entre la salida de la capa y la salida deseada u objetivo con el objeto de calcular un error.

La pantalla de configuración es la siguiente (figura 3.11):

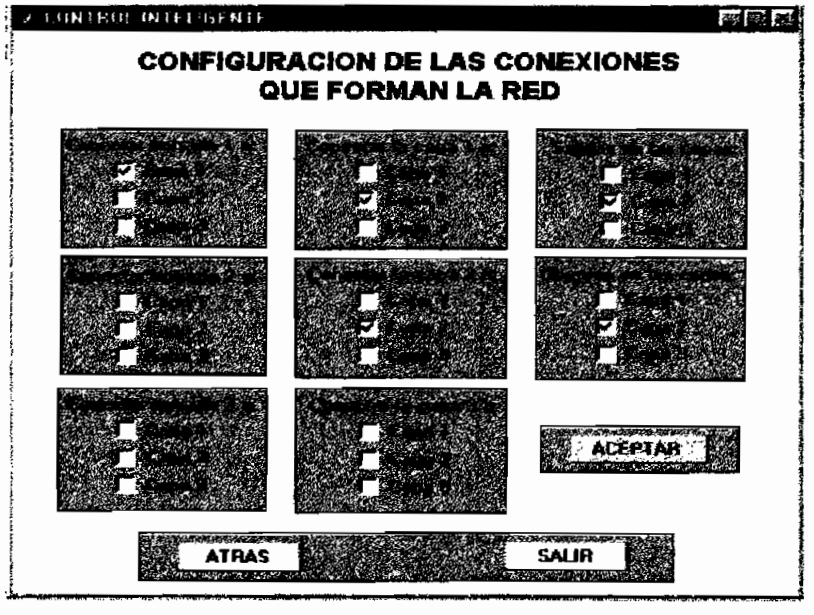


Fig. 3.11 Configuración de las conexiones de la red

Luego de definidas las conexiones, presione ACEPTAR, entonces regresará automáticamente el menú principal para configuración (pantalla 3.5).

Si se ha seguido los pasos correctamente configurando toda la red, se habilita en el menú principal de configuración el botón CONFIGURAR, al presionarlo, el programa crea la red <<net>> de acuerdo a las elecciones del usuario.

Los pasos que se realiza para la configurar la red son los siguientes:

net = network;	crea la red de nombre net
net.numInputs = 1;	número de vectores de entrada
net.numLayers = 2;	número de capas
net.inputConnect = [0 0; 1 0];	conexión de las entradas, es una matriz mxn, donde m es el número de capas y n es el número de entradas, en este caso 2x2. Así, la conexión (2,1), representará la presencia de un peso de entrada que va hacia la segunda capa desde la primera entrada, los ceros (0) representan la ausencia de peso de entrada, es decir, no hay conexión.
net.biasConnect = [1; 0];	conexión de bias, es un vector rx1, donde r es el número de capas. En este ejemplo hay una conexión de bias en la capa uno.
net.layerConnect = [0 0; 1 1];	conexión entre capas, representa la presencia o ausencia de un peso de conexión de capas. Es una matriz mxm donde m es el número de capas, en este caso, se tiene las capas uno y dos conectadas a la capa dos.
net.outputConnect = [0 1];	conexión de salida, es un vector 1xr, donde r es el número de capas. Para esta red, la salida está en la capa 2.

net.targetConnect = [0 1];	conexión de objetivo, al igual que en la conexión de salidas, es un vector 1xr.
net.layers{n}.size = 3;	número de neuronas en la capa n.
net.layers{n}.transferFcn = 'logsig';	función de transferencia de las neuronas de la capa n.
net.layers{n}.initFcn = 'initnw';	función de inicialización de la capa n.
net.inputs{n}.range = [min max; min max];	rangos máximo y mínimo del vector de entrada n. Es una matriz sx2, donde s es el número de elementos del vector, en este caso 2 elementos.

net.layerWeights{1,1}.delays=1;

net.initFcn = 'initlay';	Función de inicialización de la red.
net.performFcn = 'mse';	Función de rendimiento de la red.

Con todos estos parámetros la red queda totalmente definida, terminando con el proceso de estructuración. Aparecerá entonces la pantalla de la figura 3.12, en donde se presenta dos opciones: *Reconfigurar* y *Continuar.*

La primera alternativa regresa al usuario al menú de configuración (figura 3.5), mientras que la segunda retorna al menú principal de redes neuronales (figura 3.4).

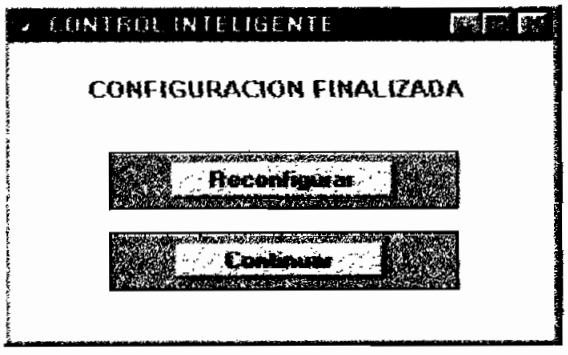


Fig. 3.12 Fin de la configuración

3.2.2.2 ENTRENAMIENTO

La etapa de entrenamiento está deshabilitada mientras no se haya ingresado a la simulación. La razón es que para realizar el entrenamiento se debe contar con los vectores de entrada X y de salidas objetivo O, y estas se ingresan antes de la simulación.

Una vez que se ha ingresado los vectores X y O, el botón Entrenamiento se habilita y al presionarlo se exhibe la pantalla de la figura 3.13:

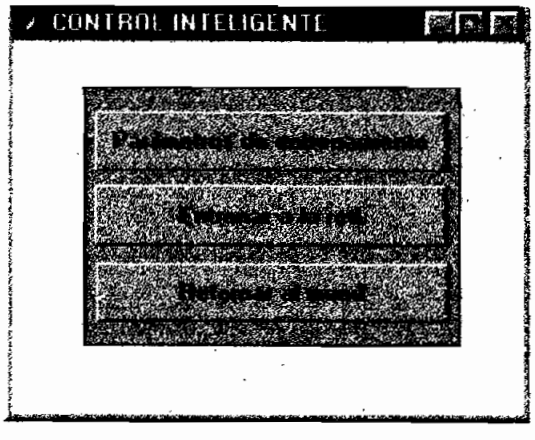


Fig. 3.13 Entrenamiento de la red

Este submenú presenta dos nuevas opciones al usuario. En "Parámetros de entrenamiento", se configura tres de los parámetro más importantes que definen el entrenamiento de la red. La pantalla correspondiente se indica en la figura 3.15.

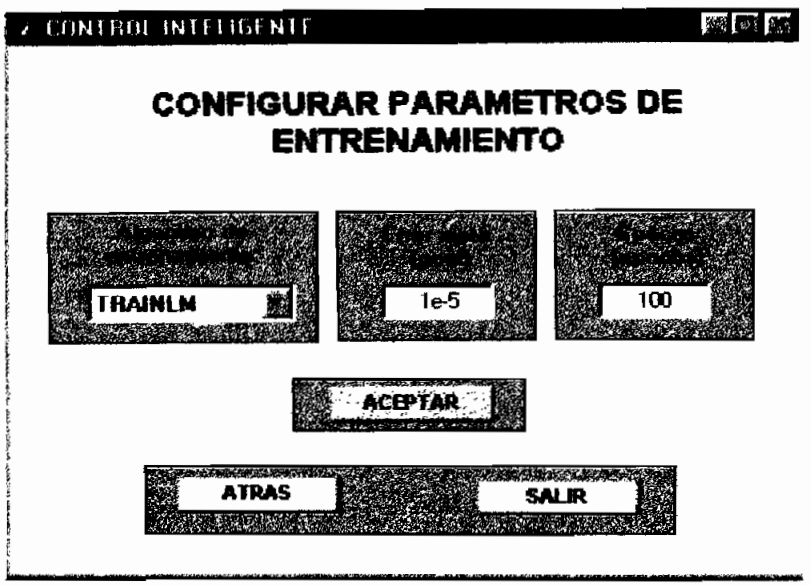


Fig. 3.14 Parámetros de entrenamiento

Aquí el usuario podrá elegir entre varios algoritmos de entrenamiento, además deberá definir el error deseado y el número de iteraciones de entrenamiento. Los demás parámetros quedan configurados de acuerdo a los valores por defecto de cada algoritmo.

Al ACEPTAR quedarán configurados los parámetros de entrenamiento e inicia el entrenamiento con las siguientes instrucciones (los valores que aquí se muestran son un ejemplo):

```
net.trainFcn = 'trainlm';
net.trainParam.goal = 1e-5;
net.trainParam.epochs = 100;
```

Hay ocasiones en las que será necesario volver a entrenar a la red, entonces el alumno escogerá del submenú anterior la opción "Entrenar a la red", con esto nuevamente inicia el entrenamiento. Ya sea el entrenamiento por primera vez o las veces que sean necesarias, se despliega el gráfico de entrenamiento error vs epochs como se indica en la figura 3.15. En la pantalla de comandos de Matlab aparecen los datos de entrenamiento que indica el error y el gradiente que va calculando la red. Al final se indica si se ha alcanzado o no la meta, como se observa a continuación.

TRAINLM, Epoch 0/100, MSE 1.2817/1e-010, Gradient 151.696/1e-010
TRAINLM, Epoch 25/100, MSE 0.00018993/1e-010, Gradient 0.0253095/1e-010
TRAINLM, Epoch 44/100, MSE 0.000179411/1e-010, Gradient 0.00749196/1e-010
TRAINLM, Maximum MU reached, performance goal was not met.

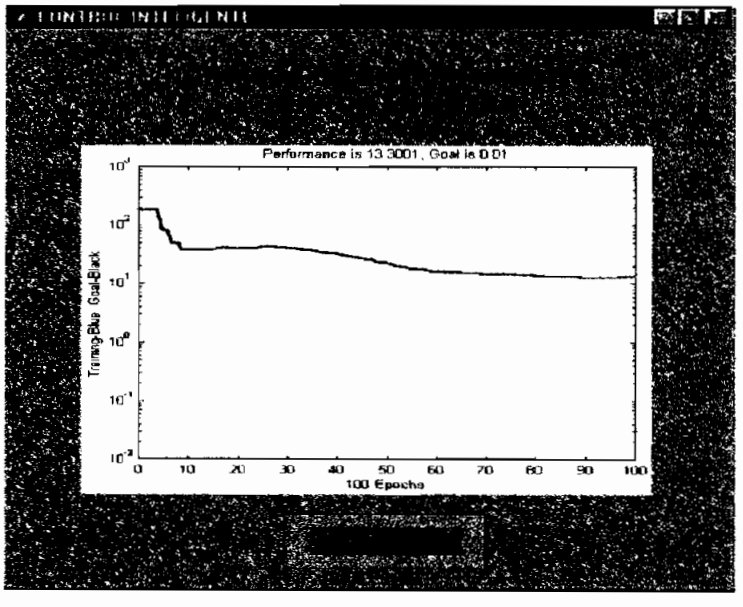


Fig. 3.15 Gráfico de resultados del entrenamiento

3.2.2.3 SIMULACIÓN

Al ingresar a esta opción, el primer paso es definir los vectores P (de entrada) y T (de salidas objetivo). Se desplegará el submenú de la figura 3.16.

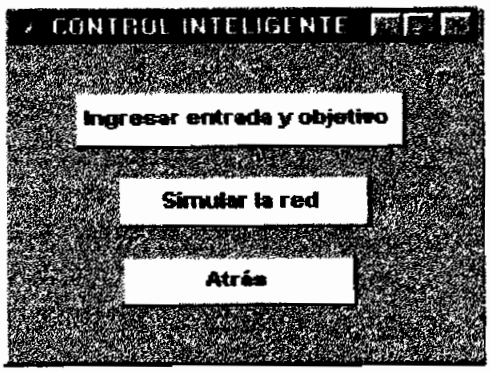


Fig. 3.16 Submenú de ingreso de datos y simulación

Al escoger la alternativa **Ingresar entradas y salidas**, se presenta la pantalla que se muestra en la figura 3.17.

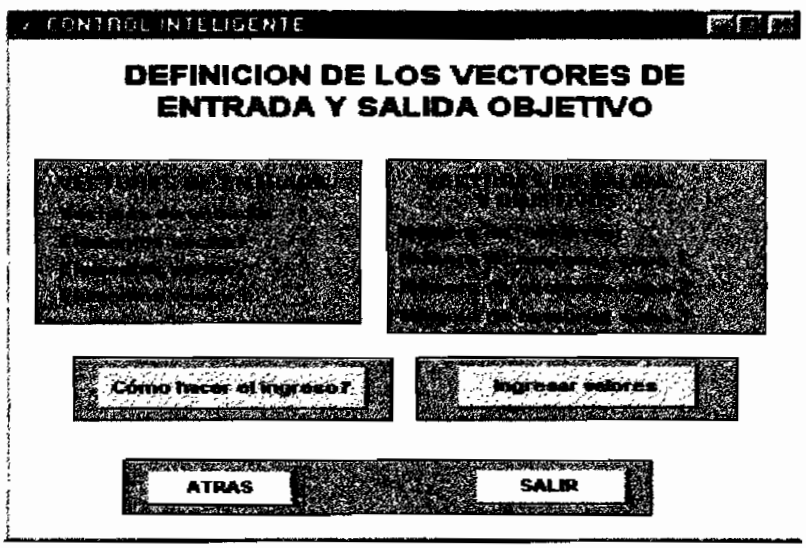


Fig. 3.17 Ingresar vectores para simulación

Como se aprecia, lo que aparece es información de la red sobre las entradas y salidas, esto se hace para que el usuario tenga claro cuantos datos debe ingresar para cada vector.

Hay dos nuevas alternativas:

- *¿Cómo hacer el ingreso?.* Se despliega una pantalla que indica el formato de ingreso de estos vectores en Matlab. La figura 3.18 expone dicha ayuda por medio de un ejemplo.

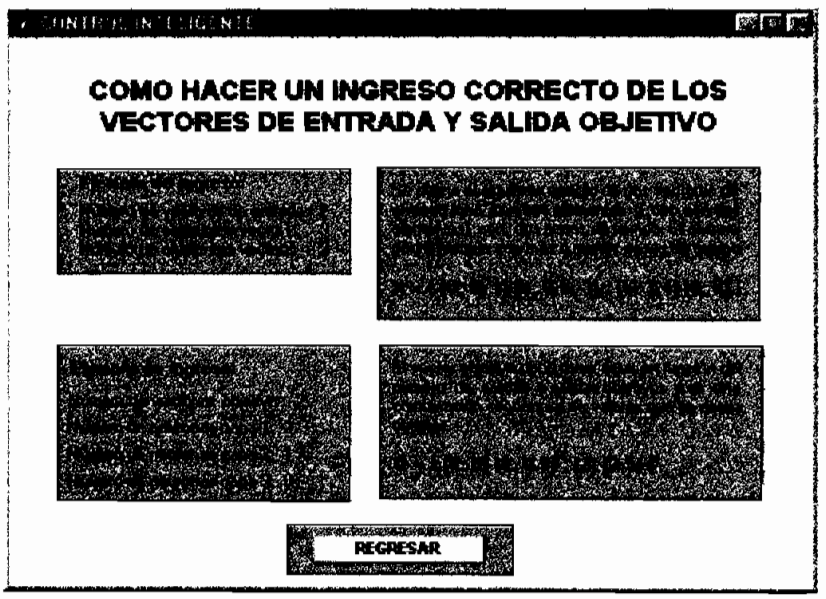
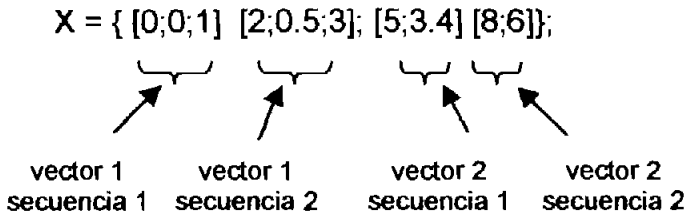


Fig. 3.18 Ejemplo de ingreso de vectores

En el primer recuadro se indican los siguientes datos:
Número de vectores de entrada: 2
Número de elementos del vector1: 3
Número de elementos del vector2: 2

Estos datos son necesarios para armar el vector de entrada cuyos datos ingresarán secuencialmente, el formato es el siguiente:



Para la salida, el recuadro indica lo siguiente:

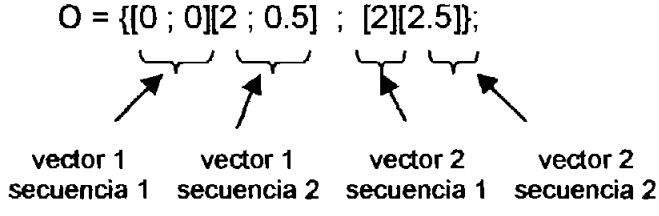
Número de vectores objetivo: 2

Número de neuronas capa 1: 2

Número de neuronas capa 2: 1

Número de neuronas capa 3: 0

Los datos sugieren que las salida objetivo se definieron para las capas uno y dos.



- *Ingresar valores.* Despliega la pantalla de edición de vectores que muestra la figura 3.19.

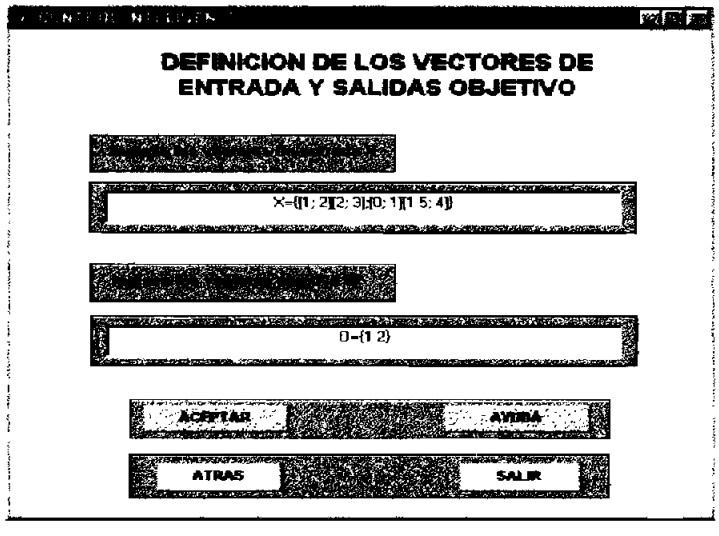


Fig. 3.19 Edición de los vectores de entrada y salida objetivo

Hay dos nuevas posibilidades, AYUDA permite retornar a la pantalla de la figura 3.18 y ACEPTAR conduce al submenú de la figura 3.16.

En el submenú, el siguiente paso es **Simular**, la pantalla es la siguiente:

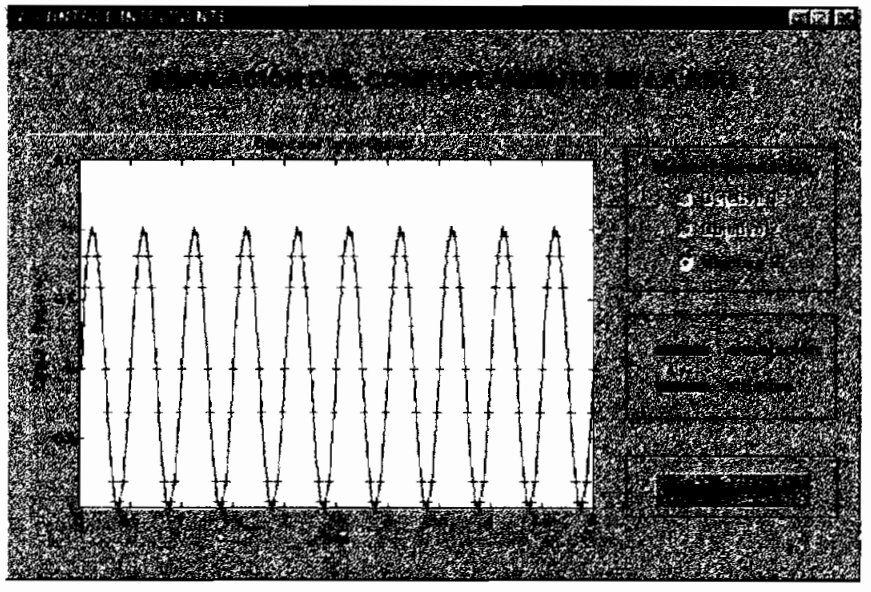


Fig. 3.20 Gráfica datos simulación y datos objetivo

De acuerdo al número de objetivos que se tenga, se podrá activar el gráfico correspondiente en el recuadro Mostrar Simulación

3.2.3 APLICACIÓN

Se ha escogido una aplicación que resulta muy didáctica para el uso estudiantil es la 'Detección de amplitudes'.

El objetivo es entrenar a la red de tal manera que ésta sea capaz de determinar la amplitud de la señal que ha ingresado a ella. Esto se hará utilizando el software de soporte, con instrucciones diferentes a las que se había utilizado en el capítulo anterior. La idea se muestra en la gráfica de la figura 3.21, que es la pantalla de ingreso a la aplicación. Aquí, la señal de entrada está en color azul y

es una señal senoidal cuya amplitud varía en el tiempo. La salida está en verde y son escalones de amplitud igual a la de la entrada a medida que varía.

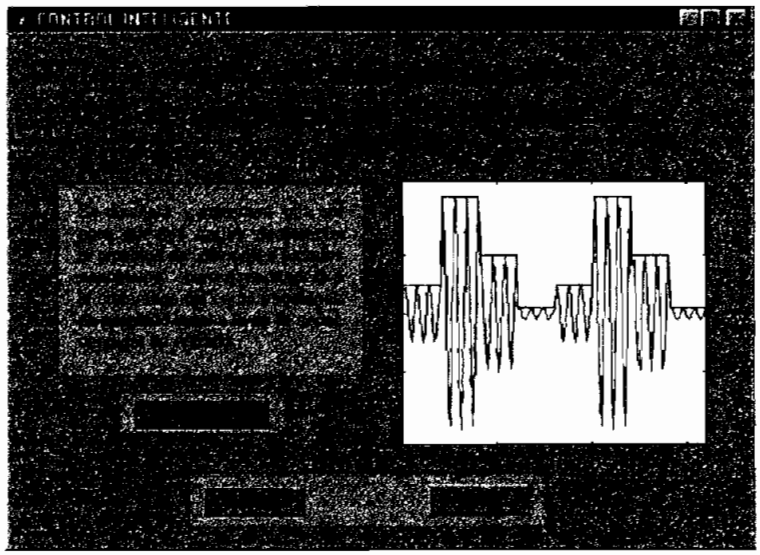


Fig. 3.21 Pantalla de inicio de la aplicación

En esta parte se explica brevemente el objetivo de esta aplicación. Cuando el usuario presiona el botón ENTRAR, se presenta la siguiente pantalla.

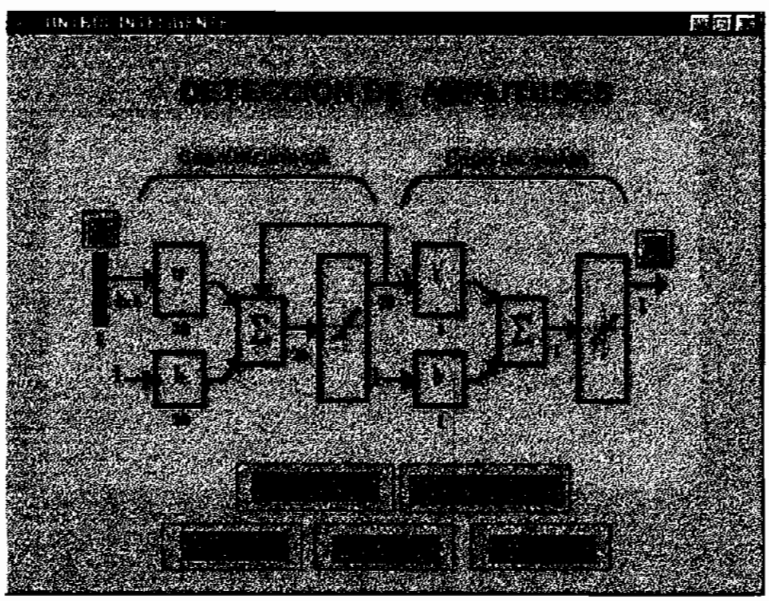


Fig. 3.22 Menú de aplicación

En esta gráfica se aprecia la red que será configurada para realizar la acción deseada. Se observa que es una red recurrente, pues ésta son las que mejor responden a este tipo de requerimiento.

Desde aquí, el usuario podrá acceder a varias pantallas que indican como está configurada exactamente la red.

Al presionar INFORMACION se desplegará la pantalla de la figura 3.23. Aquí se explica abreviadamente porqué usar la red recurrente.

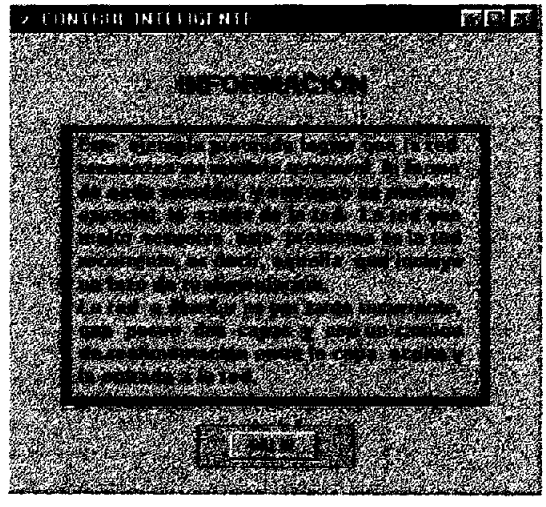


Fig. 3.23 Información

Al presionar 'salir', regresará a la pantalla de la figura 3.22.

A continuación se puede ingresar, presionando el botón X, a la pantalla que muestra la configuración de la entrada.

La acción de cada ingreso ha sido cancelada de manera que la configuración sea fija. Para la aplicación es necesaria una sola entrada, de un elemento de rango -5 a 5.

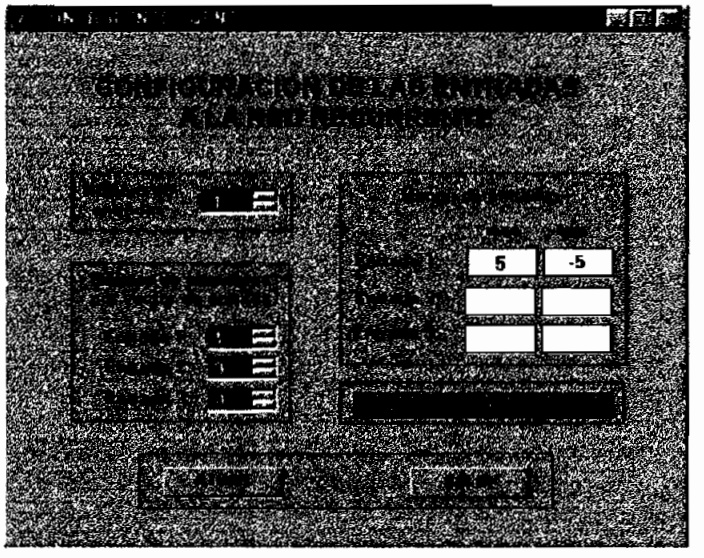


Fig. 3.24 Configuración de la entrada

Es importante que el alumno conozca exactamente como fue generada la entrada. Al presionar 'Ver generación de la entrada' se presenta la pantalla informativa de la figura 3.25.

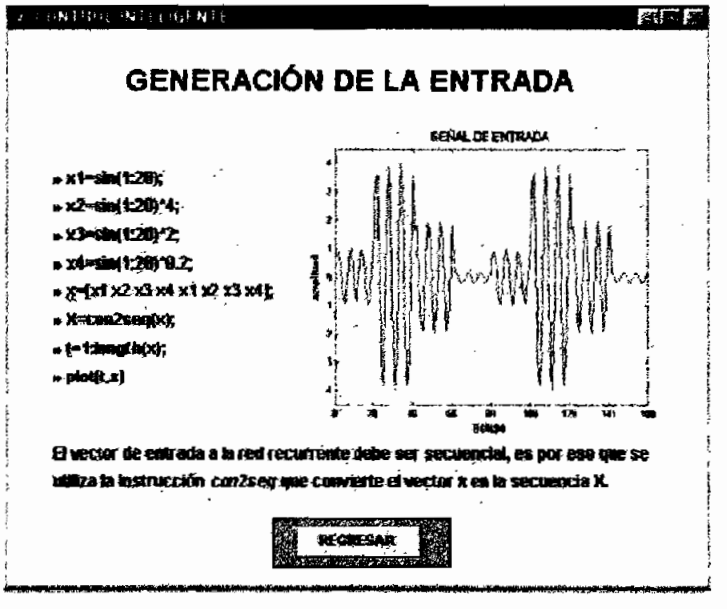


Fig. 3.25 Generación de la entrada la entrada

Se presenta cada una de las instrucciones que fueron utilizadas inclusive para observar en forma gráfica la señal de entrada que está formada por cuatro

señales individuales de diferentes amplitudes que fueron combinadas para obtener una sola señal que varía en el tiempo.

Para conocer de la configuración de la red, en la pantalla de la figura 3.22 se escogerá la opción **CONFIGURACION**, se pasa entonces a un submenú en donde se puede escoger entre, ver la configuración de las capas o ver las conexiones. A continuación se muestran las pantallas respectivas:

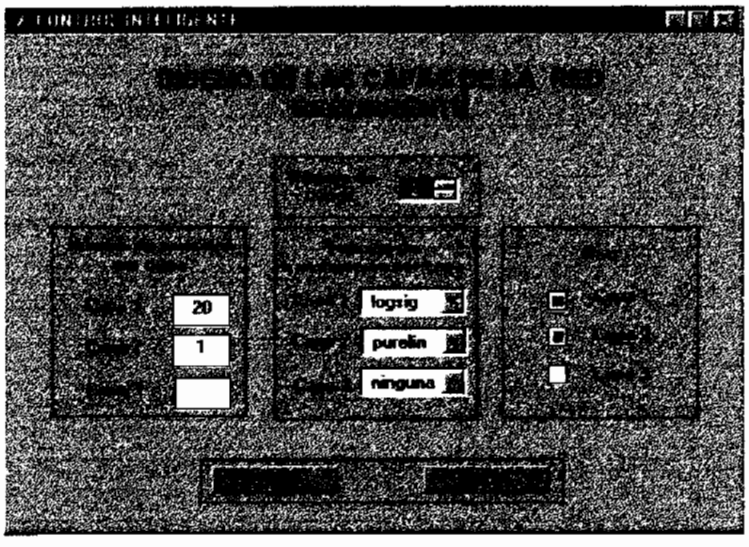


Fig. 3.26 Configuración de las capas de la red

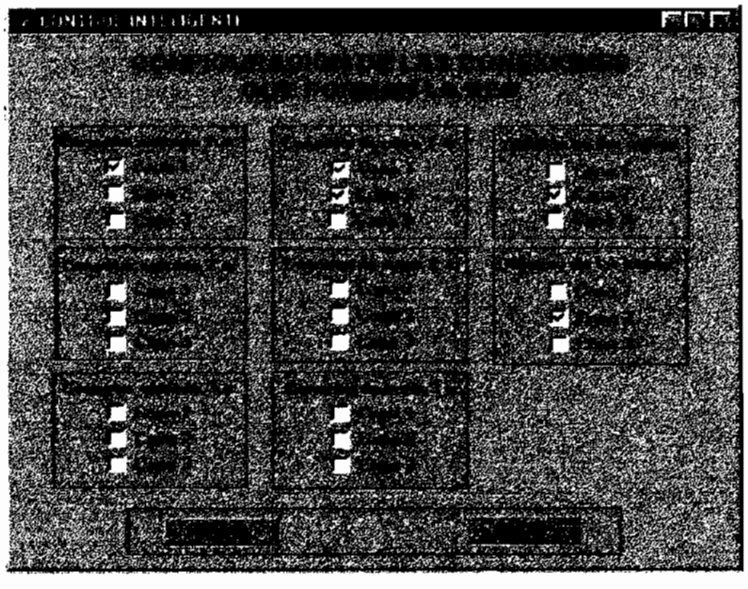


Fig. 3.27 Configuración de las conexiones de la red

Cuando desde la pantalla de la figura 3.22 se escoge la opción O, se despliega información sobre cómo se generó la señal de salida objetivo de la red. La figura 3.28 presenta la pantalla.

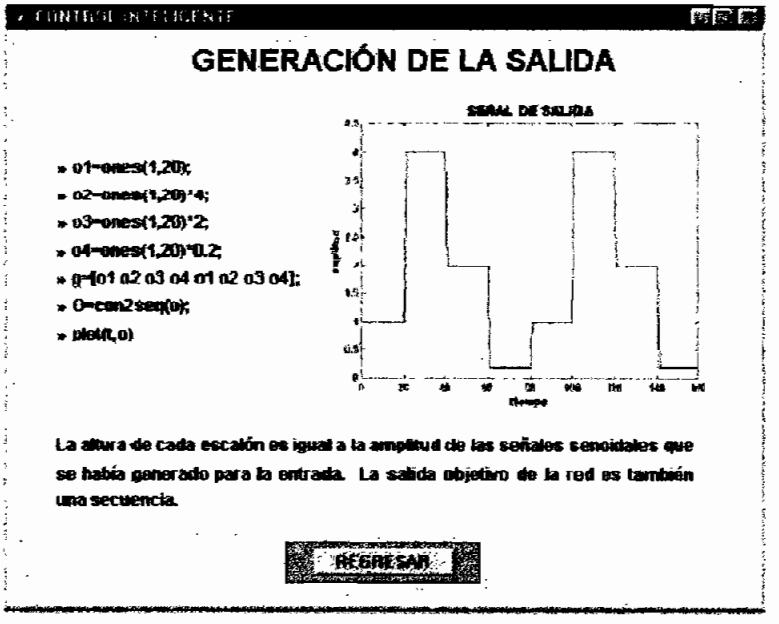


Fig. 3.28 Generación de la salida

La siguiente opción en la figura 3.22 es CONTINUAR. Al presionarlo se pasa a observar los parámetros de entrenamiento de la red.

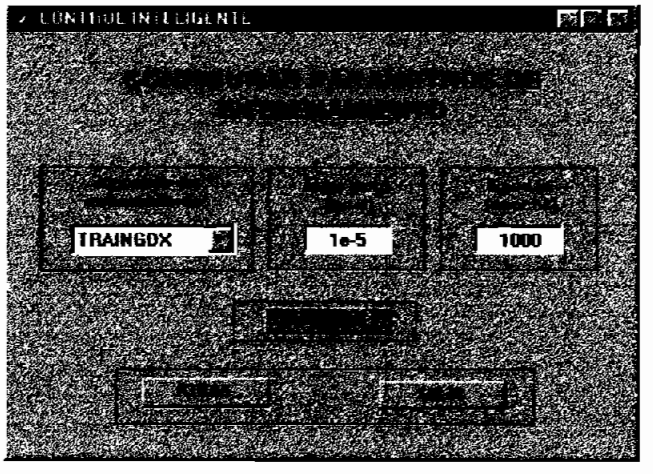


Fig. 3.29 Parámetros de entrenamiento de la red

La red ha quedado configurada usando las siguientes instrucciones:

```
dam=network;
dam.numInputs = 1;
dam.numLayers = 2;

dam.inputConnect = [1; 0];
dam.biasConnect = [1; 1];
dam.layerConnect = [1 0 ; 1 0];
dam.outputConnect = [0 1];
dam.targetConnect = [0 1];

dam.layers{1}.size = 20;
dam.layers{1}.transferFcn = 'tansig';
dam.layers{1}.initFcn = 'initnw';
dam.layers{2}.size = 1;
dam.layers{2}.transferFcn = 'purelin';
dam.layers{2}.initFcn = 'initnw';

dam.inputs{1}.size=1;
dam.layerWeights{1,1}.delays=1;

dam.initFcn = 'initlay';
dam.performFcn = 'mse';
dam=init(dam);

dam.trainFcn='traingdx';
dam.trainparam.epochs=1000;
dam.trainparam.show=50;
dam.trainparam.goal=1e-5;
```

Al presionar el botón ENTRENAR, empieza el entrenamiento de la red y se despliega la pantalla que muestra el gráfico propio del entrenamiento; este proceso dura aproximadamente 15 minutos y depende del número de neuronas

que se utiliza, además de la rapidez del computador. La instrucción para entrenar es [dam,dtr]=train(dam,Pseq,Tseq);

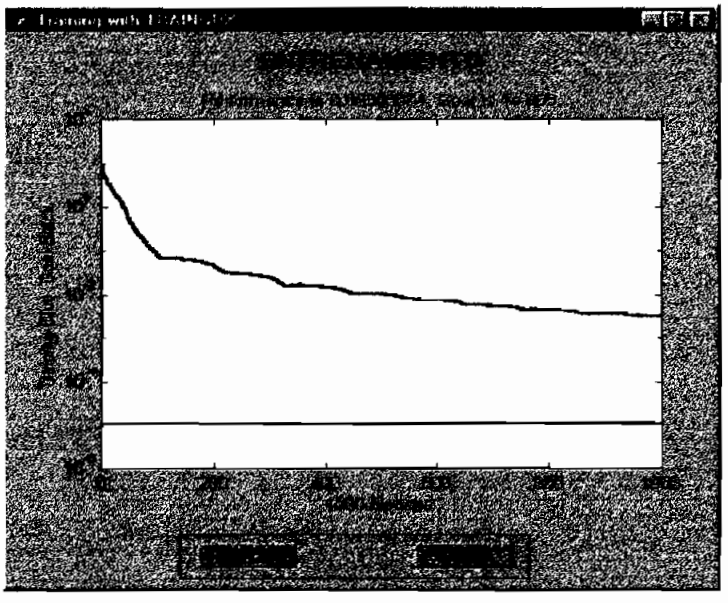


Fig. 3.30 Gráfica de entrenamiento de la red

Cuando el entrenamiento ha terminado, se puede simular el comportamiento de la red.

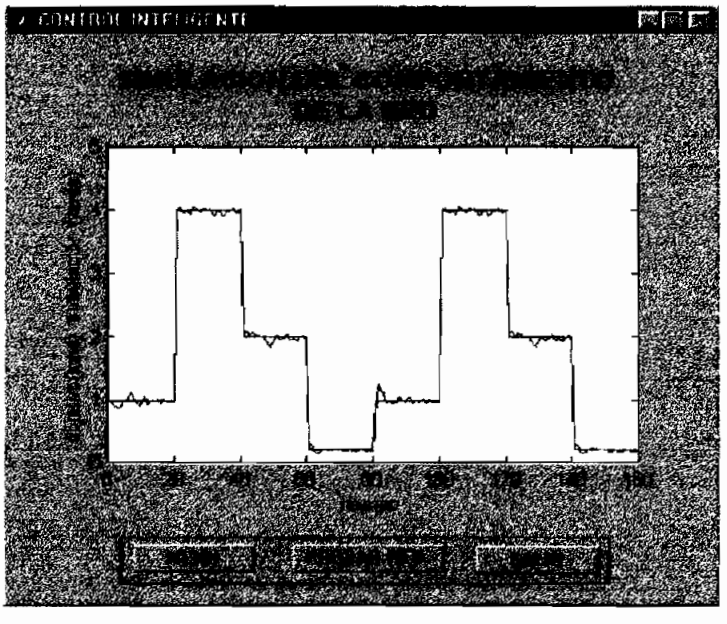


Fig. 3.31 Simulación de la red

Desde aquí, se podrá probar como funciona la red luego del entrenamiento presionando el botón PROBAR RED.

La figura 3.32 muestra la pantalla en donde el usuario puede variar una señal senoidal de dos amplitud diferentes y de esta manera observar como responde la red.

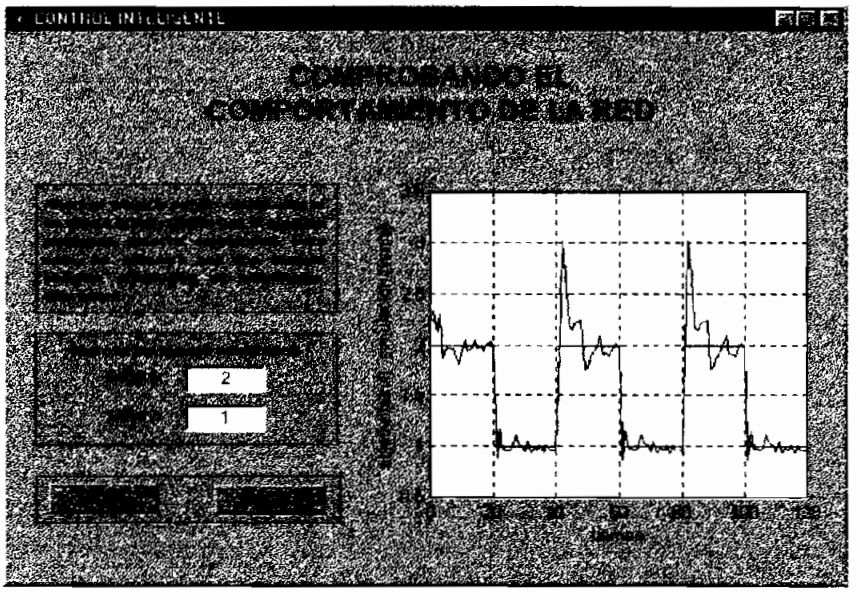


Fig. 3.32 Funcionamiento de la red

Desde cada una de las pantallas anteriores el alumno puede salir del programa presionando la tecla 'SALIR' o podrá regresar a la pantalla anterior presionando la tecla 'ATRAS'. En otras la opción MENU permitirá regresar al menú del cual se partió.

CAPÍTULO IV

RESULTADOS

4.1 LA ENSEÑANZA PROBLÉMICA Y EL CONTROL INTELIGENTE

La enseñanza problémica abre un nuevo sendero pedagógico para manejar materias técnicas, puesto que eleva la calidad cognoscitiva en el alumno, sembrando iniciativa y creatividad que son los fundamentos para desarrollar exitosamente las carreras técnicas en especial las ingenierías como es nuestro caso.

El módulo didáctico ha demostrado ser una herramienta de enseñanza eficaz, puesto que resume en cada una de las unidades los conceptos e ideas básicas que permiten al alumno iniciarse en el campo del Control Inteligente. Ahora bien, si por sí solo el módulo es efectivo, la potencialidad se eleva aún más, si éste es diseñado de acuerdo al enfoque problémico o ABP.

Cuando el alumno se auto-educa, puede suceder que inicialmente no detecte la falta de cierto conocimiento, a pesar de que ha comprendido la teoría a él presentada. Esto es normal, sin embargo, si el módulo carece de un adecuado planteamiento, el proceso puede quedar inconcluso, negándole la oportunidad de introducirse en nuevos temas y profundizar en otros.

Proponer ejercicios y problemas que activen la cognoscitividad del estudiante es la medida que indica la utilidad o inutilidad del producto educativo. En el presente trabajo, se ha procurado sugerir ejercicios que ayuden al estudiante a reconocer una necesidad de conocimiento, favoreciendo de esta manera el crecimiento intelectual.

Al aplicar la teoría pedagógica del ABP a la enseñanza de Control Inteligente se tiene como resultado un módulo didáctico provechoso, que permite al alumno

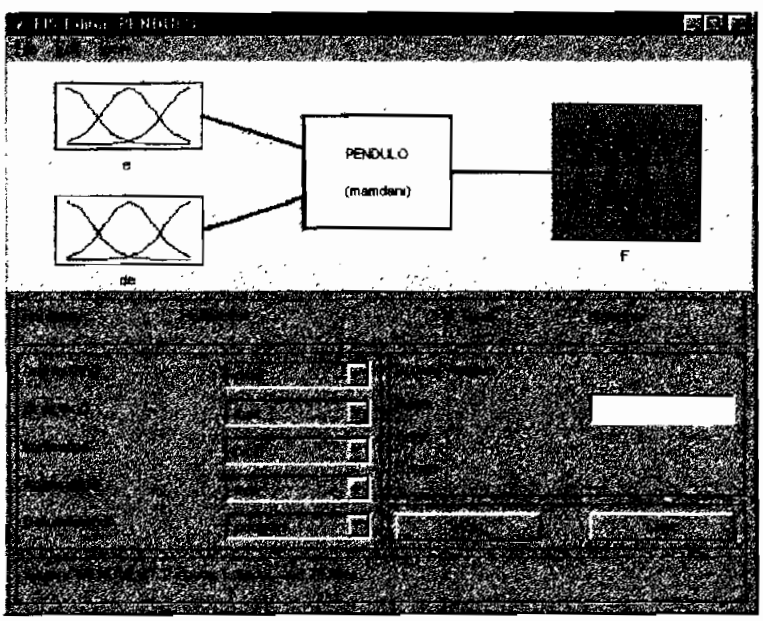
relacionarse con las nuevas técnicas de control y hacer suyo el conocimiento en forma efectiva.

4.2 RESULTADOS DEL DISEÑO DE UN SISTEMA LÓGICO DIFUSO

El sistema de control difuso se ha implementado para controlar la posición del péndulo invertido y se ha utilizado la interfaz gráfica de Matlab que permitirá observar el proceso de fusificación, inferencia y defusificación.

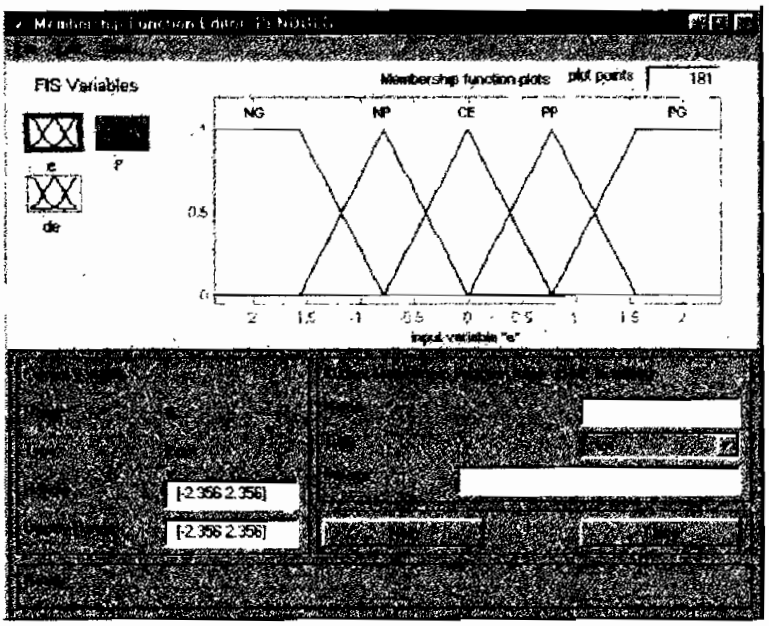
El sistema consta de dos variables de entrada, error (e) y cambio del error (de), y una variable de salida, fuerza (F), y se han especificado las operaciones lógicas de la siguiente forma:

AND	producto
Implicación	producto
Agregación	máximo
Defusificación	centroide

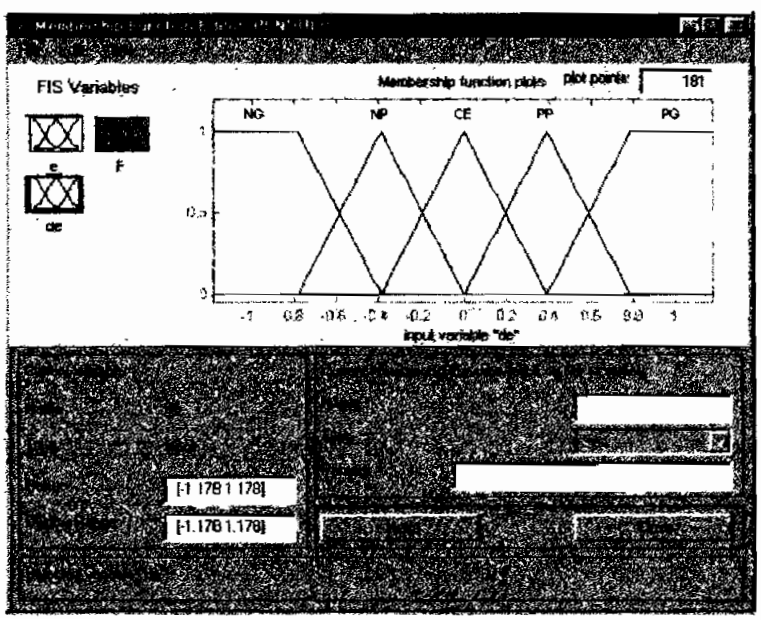


Las funciones de membresía asociadas a los conjuntos difusos de la variable de entrada **e**, son triángulos simétricos y trapecios en el caso en que estén más hacia los extremos. Las etiquetas de cada conjunto son: Negativo Grande (NG),

Negativo Pequeño (NP), Cero (CE), Positivo Grande (PG) y Positivo Pequeño (PP). El universo de discurso se ha definido en el rango de [-3π/4 , 3π/4].

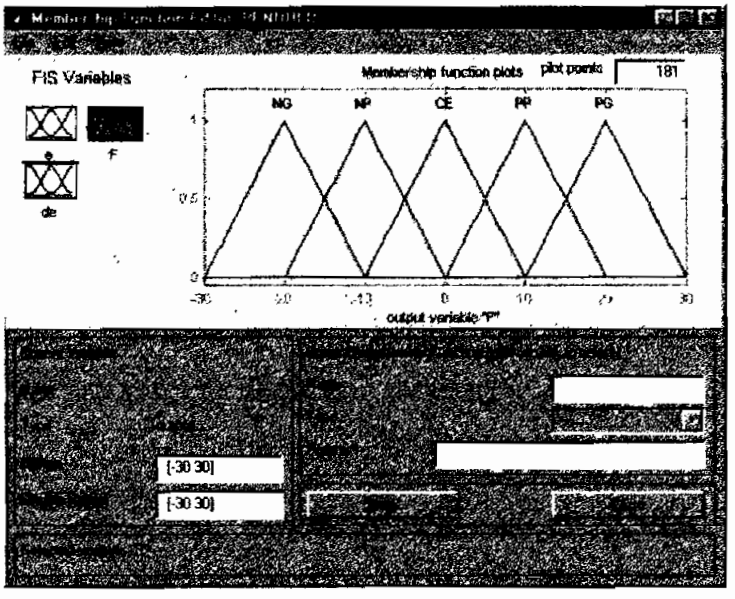


Para la variable de entrada ***de***, se utiliza las mismas formas de funciones de membresía e incluso las mismas etiquetas. Se tienen sin embargo, un universo de discurso diferente, en este caso el rango es [- 3π/8 3π/8].

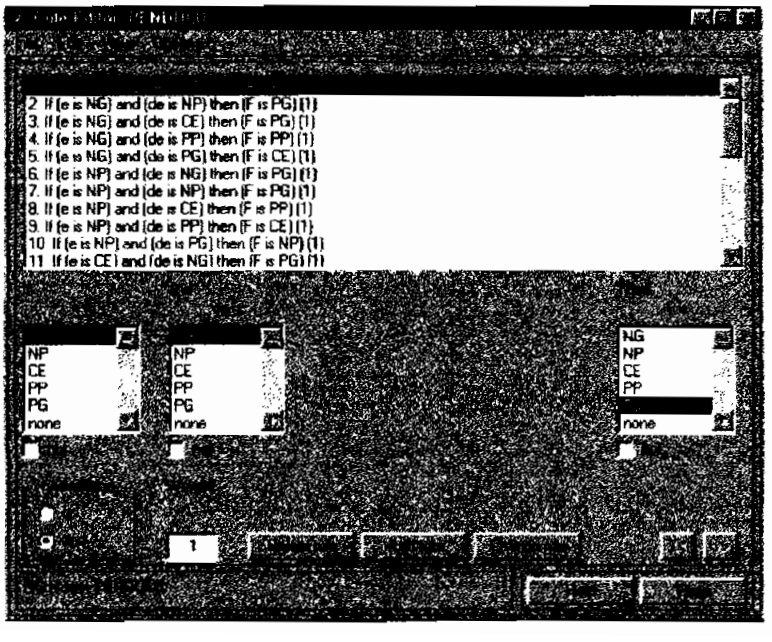


Para la variable de salida, ***F***, se ha definido asimismo cinco conjuntos difusos pero todas las funciones de membresía son triangulares.

Las etiquetas son también en este caso NG, NP, CE, PP y PG. El universo de discurso para esta variable es [-30 30].

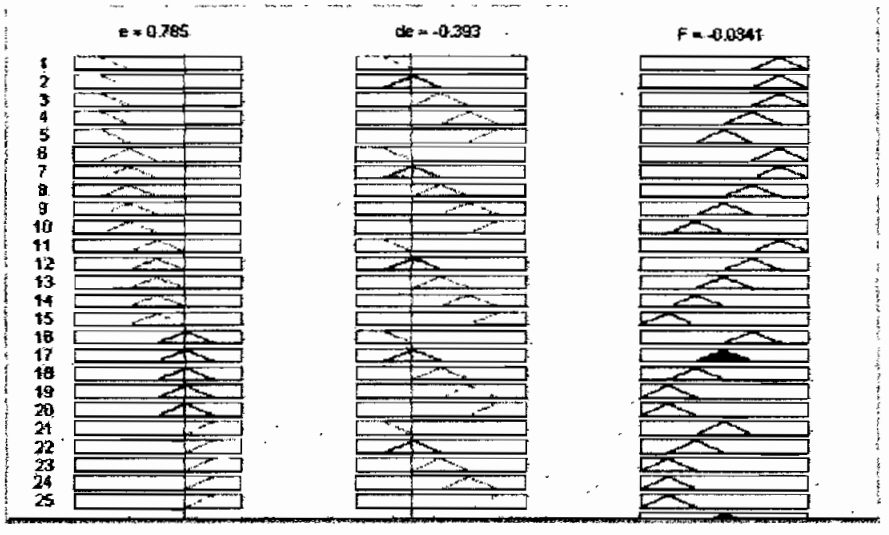


Puesto que se cuenta con dos variables de entrada, cada una con cinco conjuntos difusos, el número de reglas que resultan de todas posibles combinaciones es 25.



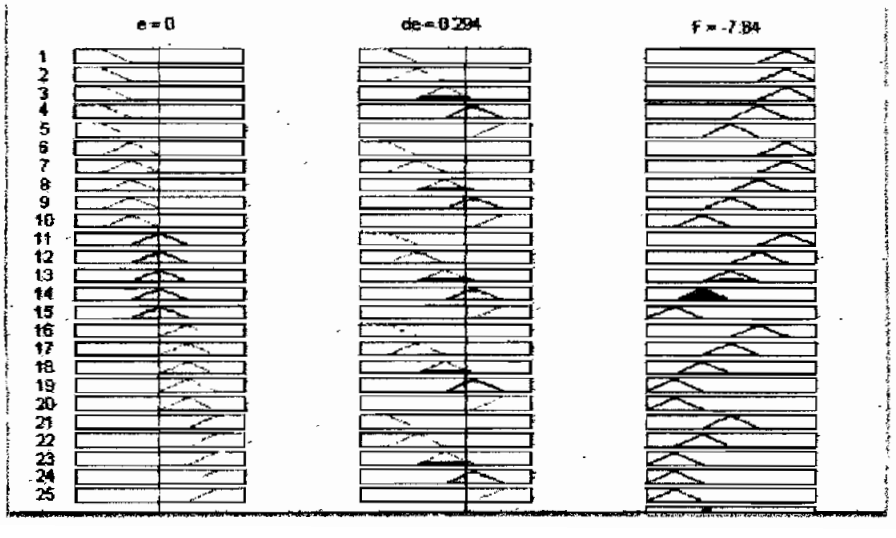
Terminado el diseño del sistema lógico difuso, se presenta el resultado para varias condiciones:

Para $e = \pi/4$ y $de = -\pi/8$ se debe aplicar una fuerza $F = -0.0341$



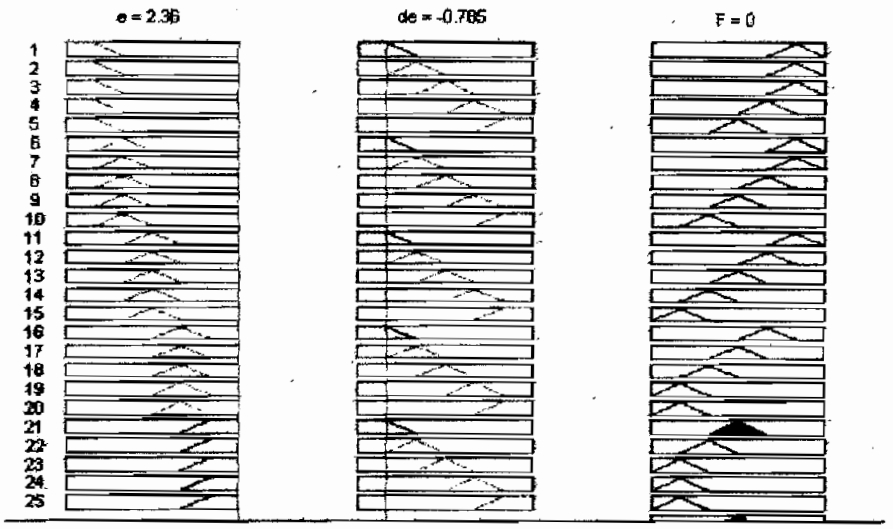
Cuando el péndulo está ligeramente hacia la izquierda de la vertical, y se está moviendo con una pequeña velocidad en el sentido de las manecillas del reloj, la fuerza que se debe aplicar una fuerza negativa (hacia la izquierda) muy pequeña (-0.031 **N**).

- Para $e = 0$ y $de = \pi/8$-$\pi/32$, la fuerza F a aplicarse es $F = -7.84$



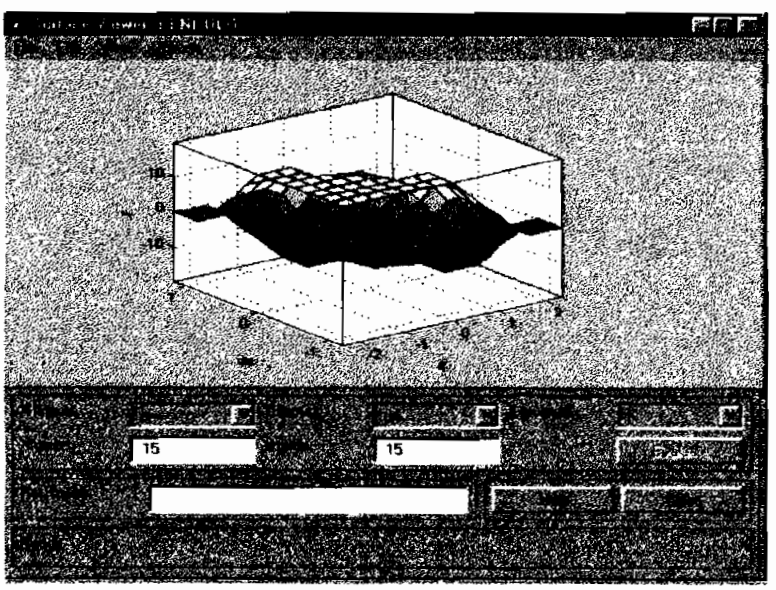
Si el péndulo está sobre la vertical (o muy cerca de ella), y se está moviendo en sentido contrario a las manecillas del reloj, la fuerza que se debe aplicar al coche es negativa (-7.54 N).

- Para e = $3\pi/4$ y de = $-\pi/4$, la fuerza es F = 0

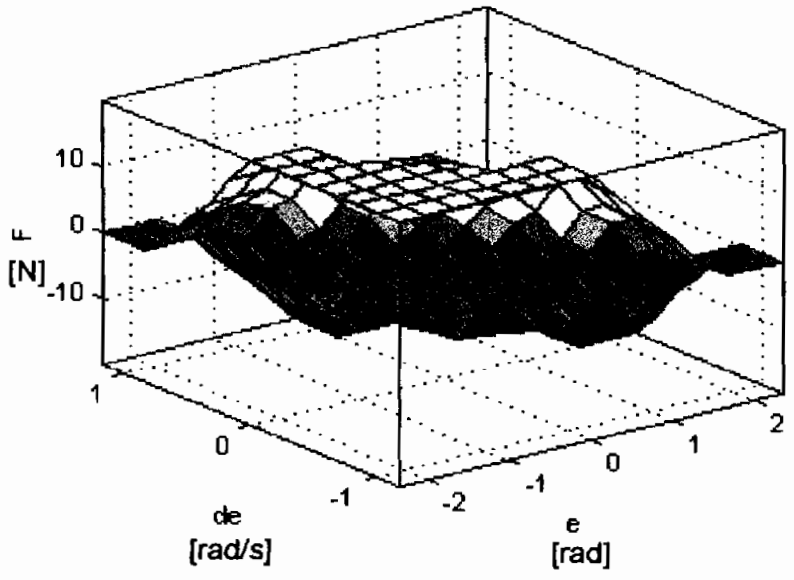


Cuando el péndulo está lejos hacia la izquierda de la vertical y se está moviendo en el sentido de las manecillas del reloj, no se aplica fuerza alguna.

La superficie de control para este sistema se muestra a continuación.



La superficie de control representa en forma compacta toda la información del controlador difuso. Para probar si cumple con lo dicho, se tomará dos valores: e = $3\pi/4$ y de = $-\pi/4$, examine en la superficie de control para verificar que en efecto, la fuerza F es cero.



4.3 RESULTADO DEL ENTRENAMIENTO DE LA RED RECURRENTE

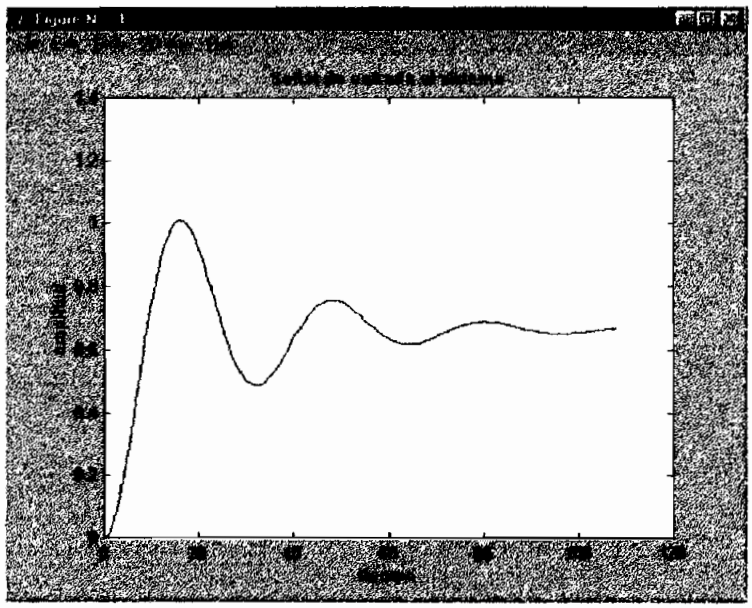
Con la ayuda del software de soporte realizado en Matlab se procederá a diseñar, entrenar y simular una red neuronal sencilla de manera que se aprecie los resultados más claramente.

El objetivo es recurrir a una red neural que permita modelar una función lineal, pues se había dicho que las redes no lineales son capaces de representar cualquier función. La red que mejor se desempeña con este tipo de planteamientos es la red recurrente, entonces, se trabajará con ella.

Vamos a suponer que se tiene un sistema al cual ingresa la siguiente señal:

```
» a=tf(2,[2, 1, 3]);
» b=step(a);
```

```
» t=1:length(b);
» plot(t,b)
```



Para entrenar la red se necesita que tanto la entrada como la salida sean secuenciales, entonces se transforma el vector fila *b* a vector columna *x*, y luego se lo transforma a una secuencia:

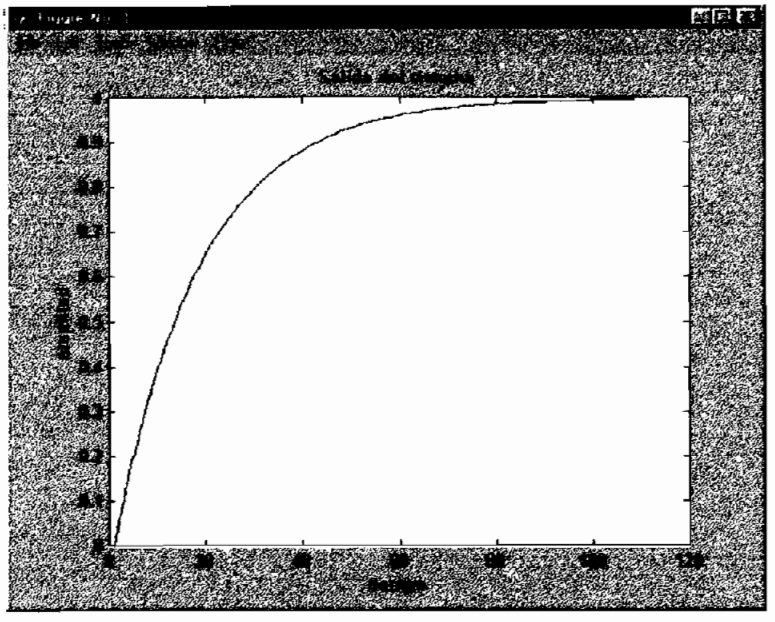
```
» x=b';
» X=con2seq(x);
```

La salida será la respuesta de cierto sistema, la cual se originó con las siguientes líneas:

```
» c=tf(1,[1, 1]);
» d=step(c);
» plot(t,d)
```

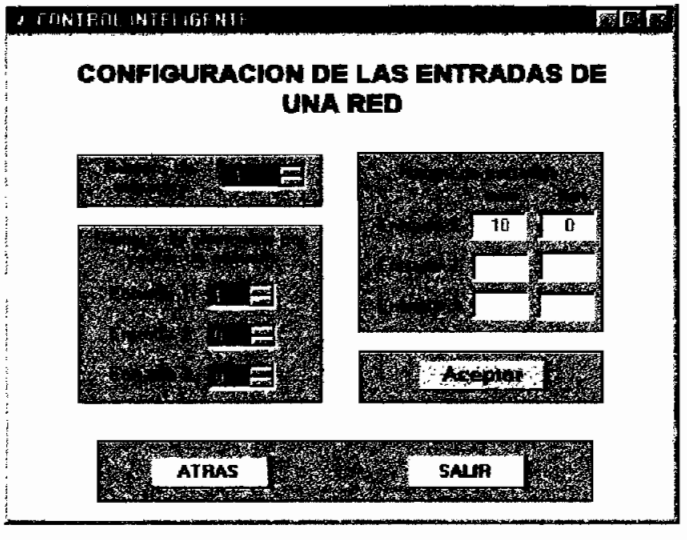
Y de igual manera se la transformará a una secuencia:

```
» o=d';
» O=con2seq(o);
```



Se inicia el programa **Sintel** para crear la red. A continuación se presenta las pantallas, en el orden en el cual transcurre el proceso de diseño y entrenamiento.

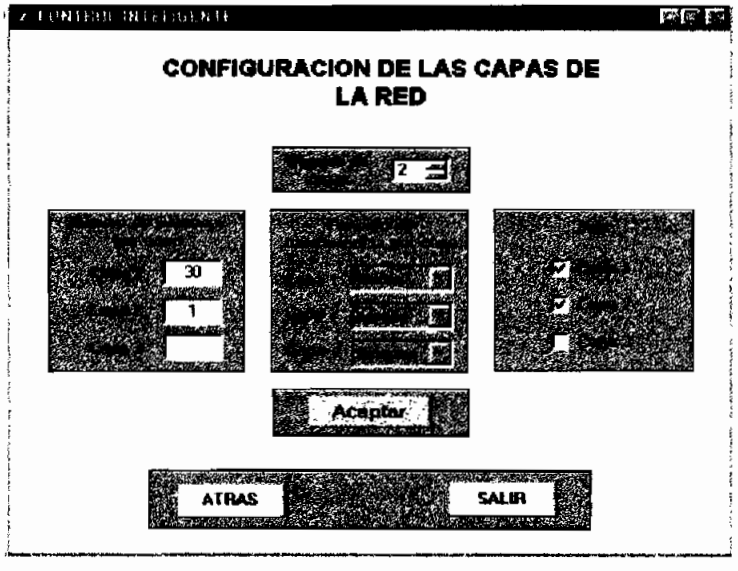
Se necesita una sola entrada, con un solo elemento que varía en el tiempo, de rango 0 a 10.



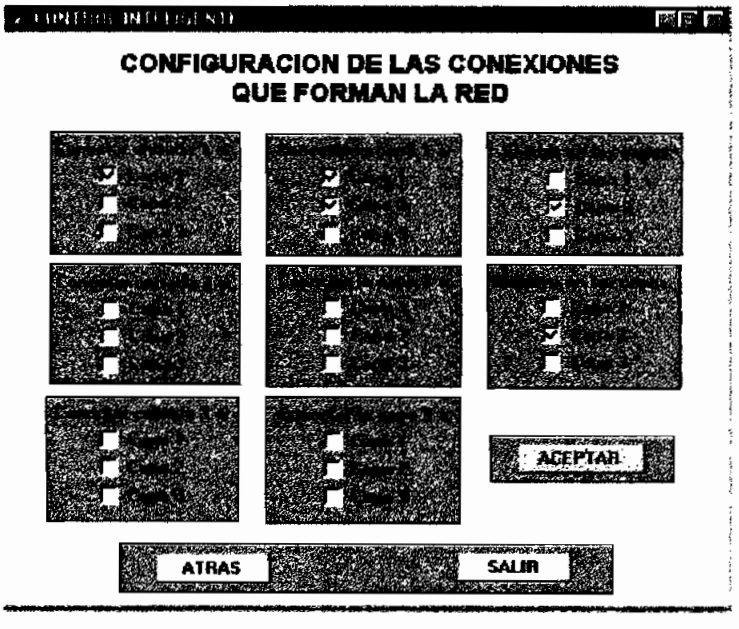
La red está compuesta de dos capas con un lazo de realimentación que va de la salida de la primera capa hacia la entrada. La primera capa tiene 30 neuronas

con función de activación 'tansig' y la de salida tiene una sola neurona con función de transferencia 'purelin'. Cada capa se configura además con bias.

Se podría decir que el número de neuronas es un tanto alto pero, a pesar de que el tiempo de entrenamiento aumenta, el resultado alcanzado es muy bueno.

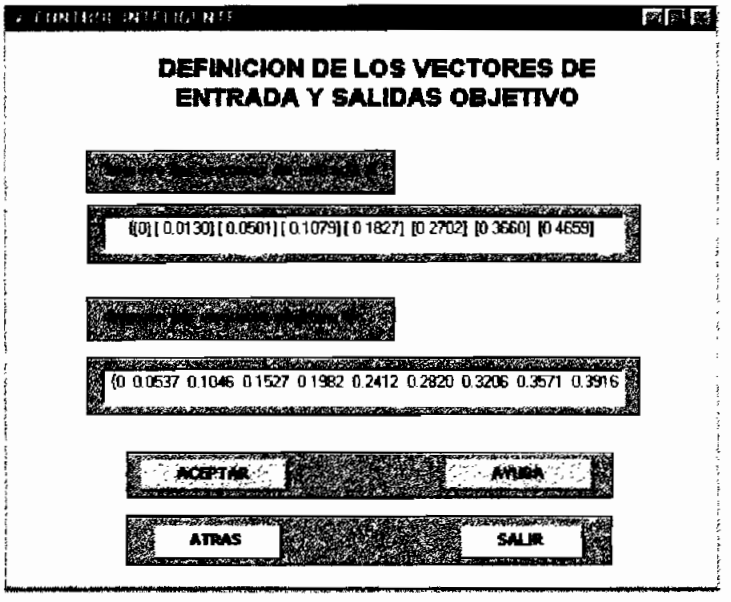


Como se mencionó anteriormente, la red será recurrente, entonces las conexiones serán las indicadas en la siguiente pantalla:



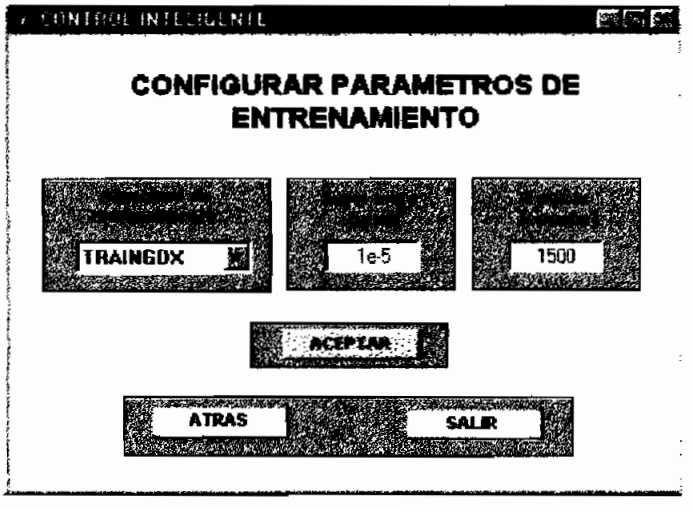
La única salida es conectada a la segunda capa y de igual forma el objetivo.

Luego de configurada la estructura de la red, se procederá a hacer el ingreso de los vectores de entrada y salida objetivo.

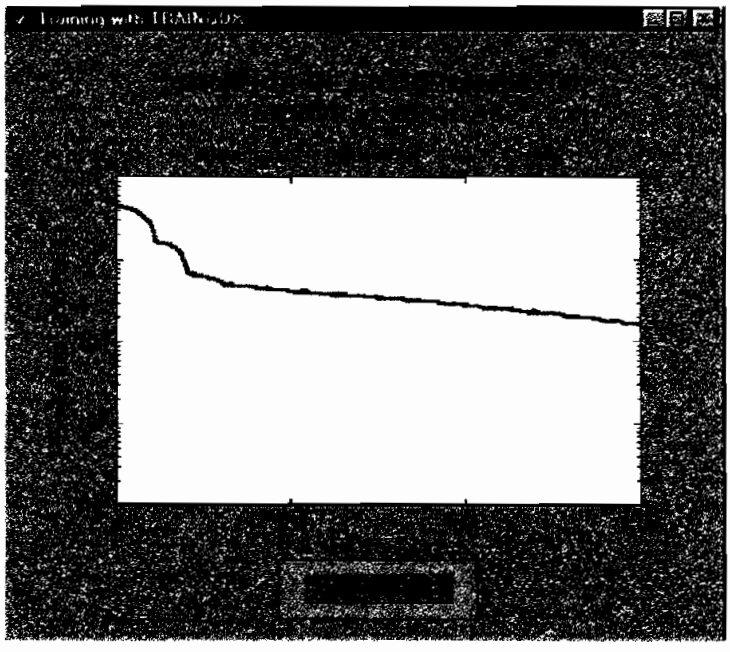


Una vez ingresado los vectores, se configuran los parámetros de entrenamiento.

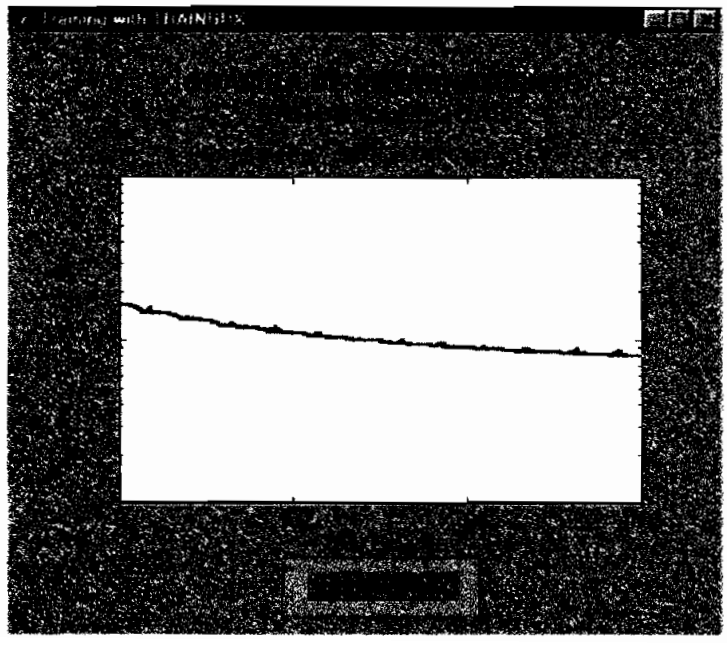
El algoritmo de entrenamiento escogido es 'traingdx', el rendimiento o error meta es 1e-5 y las iteraciones son 1500; y esto se observa en la siguiente pantalla:



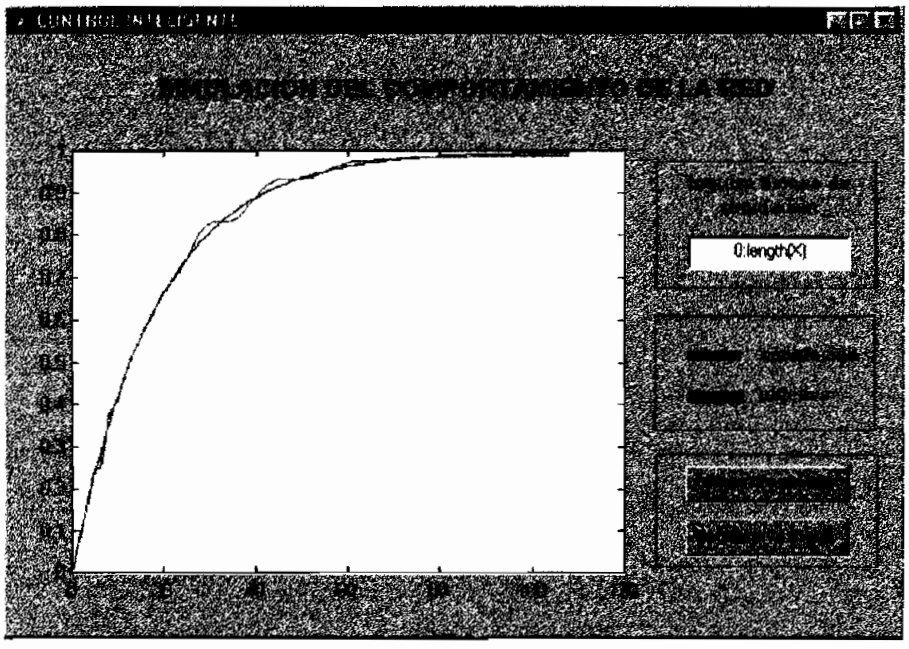
Se inicia el entrenamiento y el resultado, luego de 1500 iteraciones se muestra a continuación:



El rendimiento alcanzado es de 0.00164 y la meta era 1e-5. Este error no permite conseguir un comportamiento satisfactorio, por tanto se vuelve a entrenar a la red. Luego de otras 3000 iteraciones al fin el rendimiento es bueno. La gráfica final de entrenamiento es la siguiente:



El error alcanzado es 7.980e-5, y con este resultado se procede a simular el comportamiento de la red.



El tiempo ingresado para realizar la simulación es:

l = length(x);	longitud del vector de entrada 109 elementos
t = 0:108;	tiempo de simulación ingresado

Se observa que el resultado es muy bueno. La red simula el comportamiento de la planta entregando una señal muy similar al modelo deseado.

Para un mejor desempeño, se podría continuar entrenando a la red de manera que el error disminuya hasta alcanzar la meta o incrementar el número de neuronas en la capa oculta. Esto sin embargo incide en el tiempo de entrenamiento, pues los cálculos se vuelven más laboriosos. El tiempo total que duro el entrenamiento de la red del ejemplo fue 40 minutos, por supuesto, luego de realizar 4500 iteraciones.