ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA MECÁNICA

DESARROLLO E IMPLEMENTACIÓN DEL ALGORITMO DE UN CONTROLADOR NEURO-DIFUSO TIPO ANFIS APLICADO A UN SISTEMA AERODINÁMICO TRMS (TWIN ROTOR MIMO SYSTEM)

TESIS PREVIA A LA OBTENCIÓN DEL GRADO DE MAGISTER EN DISEÑO PRODUCCIÓN Y AUTOMATIZACIÓN

JUNIOR RAFAEL FIGUEROA OLMEDO

jrfo_roddy@hotmail.com

DIRECTOR: ING. WILLAN LEOPOLDO MONAR MONAR MSc.

William.monar@epn.edu.ec

Quito, Septiembre 2016

DECLARACIÓN

Yo, JUNIOR RAFAEL FIGUEROA OLMEDO, declaro que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional, puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Ing. Junior Rafael Figueroa Olmedo

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por el Ing. Junior Rafael Figueroa Olmedo bajo mi supervisión.

ING. WILLAN MONAR MSC.
DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Agradezco a Dios y a la vida por permitirme culminar un nuevo ciclo de estudios siempre en la compañía y con el respaldo de personas que realmente desean mi bienestar.

Agradezco al Ingeniero Willan Monar por su aporte académico durante el desarrollo de esta tesis, puesto que gracias a su asesoramiento y disponibilidad de tiempo, fue posible cumplir con éxito los objetivos planteados.

Agradezco a mis amigos y compañeros de estudio Johana Celi, Alejandra Fernández, Luisa Sotomayor y Hamilton Nuñez porque junto a ellos alcanzamos y superamos todos los retos académicos que se presentaron en el transcurso de esta maestría.

Agradezco a los docentes de la Maestría de Diseño Producción y Automatización, los cuales supieron compartir sus conocimientos, experiencias laborales y trabajos de investigación, en todo el transcurso de esta mención.

Y agradezco a los buenos amigos, amigas, compañeros y personas cercanas que de una u otra manera formaron parte de todo este proceso de estudio. Un agradecimiento especial a mi hermano Roddy Figueroa por su compañía incondicional y puesto que estuvo presente en todo momento dándome ánimos.

Junior.

DEDICATORIA

Dedico este trabajo a mis padres (Soris y Oriol) y hermanos (Roddy y Cindy), ya que gracias a su amor, comprensión y apoyo incondicional pude culminar esta nueva etapa de estudios.

Junior.

RESUMEN

El objetivo principal de este trabajo de investigación fue desarrollar el algoritmo de programación de un sistema neuro-difuso conocido como ANFIS (*Adaptive Network based Fuzzy Inference System*) y aplicarlo en procesos de modelamiento y control sobre un sistema aerodinámico TRMS (*Twin Rotor MIMO System*).

Actualmente, dentro del campo de la Ingeniería de Control se ha percibido un crecimiento en la investigación y desarrollo práctico de sistemas de inteligencia artificial como son los sistemas neuro-difusos. Muchas aplicaciones enmarcadas en este tipo de controles han sido publicadas como artículos científicos en diferentes revistas, en actas de congresos, en reuniones académicas, en informes técnicos que publican instituciones académicas, en libros, entre otros. Como resultado de este gran número de investigaciones, desde hace varios años se han desarrollado e implementado diferentes tipos de sistemas neuro-difusos que tienen como característica común el uso y combinación de las Redes Neuronales Artificiales y los Sistemas de Inferencia Difusos.

Debido a que se ha comprobado a nivel internacional que el modelo neuro-difuso tipo ANFIS es uno de los esquemas más exitosos que combinan los beneficios de las redes neuronales y la lógica difusa en una sola cápsula, es el sistema que se ha elegido en este trabajo de investigación con la finalidad de realizar el análisis de su funcionamiento y arquitectura, teniendo como meta final el poder desarrollar e implementar su algoritmo de programación utilizando el software Matlab y su entorno Simulink.

Lo que se pretende con la implementación de este sistema neuro-difuso tipo ANFIS es resolver problemas de control, no sólo de una manera novedosa, sino sobre todo tener mejores soluciones y más eficientes. La aplicación de los sistemas ANFIS como controladores puede ser muy extenso en la industria, debido a la necesidad cada vez mayor de automatizar y optimizar procesos complejos.

El desempeño del controlador ANFIS se evaluó implementándolo sobre el sistema de control TRMS (*Twin Rotor MIMO System*) junto con un controlador PID convencional, para posteriormente realizar un análisis estadístico comparativo utilizando como marco de referencia la integral del error absoluto (IAE), obtenida desde las salidas de ambos sistemas de control. Previamente a la implementación física del controlador ANFIS se realizaron pruebas simuladas de funcionamiento utilizando como planta el modelo matemático del sistema TRMS.

PRESENTACIÓN

El presente trabajo de investigación está estructurado en cuatro capítulos que se sintetizan a continuación.

En el Capítulo 1, se presentan los fundamentos teóricos necesarios para comprender y poder implementar el sistema neuro-difuso tipo ANFIS, por dicha razón sólo se explican y se abarcan ciertos temas relevantes y fundamentales de los sistemas difusos, un método de mínimos cuadrados para la identificación de los parámetros de un sistema y el algoritmo de aprendizaje por retropropagación para redes neuronales adaptativas, enfocando el estudio en las ideas que son empleadas por la red ANFIS. Se realiza la descripción de una regla de aprendizaje híbrida que nace de la combinación del algoritmo de retropropagación y el estimador de mínimos cuadrados recursivos.

En el Capítulo 2, se describe el funcionamiento y el modelado matemático de la planta TRMS sobre la cual se va a aplicar el controlador ANFIS, con la finalidad de realizar sobre este modelo las pruebas de simulación. Además se explica de forma detallada la arquitectura de la red ANFIS y la incorporación de la regla de aprendizaje híbrida dentro de este tipo de sistemas. Finalmente se desarrolla el algoritmo de programación utilizando el software Matlab junto con la herramienta gráfica de Simulink y se realizan las respectivas pruebas simuladas.

En el Capítulo 3, se implementa la red ANFIS sobre el sistema físico TRMS, con la finalidad de que actúe como un controlador. Luego se verifica la validez del controlador ANFIS y también se implementa un controlador PID clásico que ya viene incorporado y predefinido en los ejemplos proporcionados por el fabricante de la planta TRMS. Al finalizar este capítulo se realiza un análisis comparativo de desempeño de ambos controladores empleando un indicador de rendimiento conocido como integral del error absoluto (IAE) el cual será utilizado como dato de entrada a un método estadístico conocido como el Test de Wilcoxon.

En el Capítulo 4, se exponen las conclusiones y recomendaciones que se establecieron mediante el análisis de los resultados obtenidos tras el desarrollo de este trabajo.

Al final del documento se incluyen como anexos la explicación de cómo funcionan y se crean las funciones S-Function de Matlab, cómo utilizar la función reshape y como último anexo se presenta todo el algoritmo de programación de la red ANFIS desarrollado en Matlab.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

La modelación y control de sistemas basados en herramientas de matemática convencional (por ejemplo ecuaciones diferenciales) no es muy adecuada para tratar con sistemas inciertos y mal definidos, más aún cuando se desea plasmar la experiencia humana de dicho proceso.

El incremento de las demandas tecnológicas en nuestros tiempos, ha generado sistemas muy complejos que requieren controladores altamente sofisticados para asegurar un alto desempeño dentro de condiciones adversas. Estas y otras condiciones de control no se pueden cumplir con controladores convencionales, debido principalmente a la falta de conocimiento preciso acerca del proceso que se desea controlar. La adquisición de conocimiento adecuado del sistema (modelo matemático) en ocasiones es problemática o impráctica debido a la complejidad del sistema y al hecho de que la estructura y los parámetros en muchos sistemas cambian de manera significativa e impredecible con el tiempo. Es bajo estas condiciones en donde se utilizan las técnicas de la Inteligencia Artificial, que traducidas a la Ingeniería de Control se enmarcan dentro del conjunto de Controladores Inteligentes.

El Control Inteligente es la disciplina donde los métodos de control se desarrollan para emular ciertas características importantes del ser humano. Estas características incluyen adaptación y aprendizaje, planeación bajo gran incertidumbre y el trabajo con gran cantidad de datos. Las áreas donde se realizan trabajos alrededor del control inteligente son: redes neuronales, control difuso, algoritmos genéticos, sistemas expertos, sistemas híbridos (combinación de más de una de las técnicas anteriores), entre otros. Lo que se pretende con estos métodos en ingeniería es resolver los problemas de modelamiento y control, no sólo de una manera novedosa, sino sobre todo tener mejores soluciones, más eficientes y mejor planeadas.

La fusión de las Redes Neuronales Artificiales (RNA) y los Sistemas de Inferencia Difusos (FIS) han atraído el interés creciente de investigadores en diversas áreas científicas y de la ingeniería, debido a las facilidades de adaptación de estos sistemas inteligentes para resolver los problemas del mundo real. Una Red Neuronal Artificial aprende desde cero mediante el ajuste de las interconexiones entre las capas. Un Sistema de Inferencia Difuso es un marco popular de la informática basado en el concepto de la teoría de conjuntos difusos, reglas difusas *si-entonces* y el razonamiento difuso. Las ventajas de una combinación de las RNA y los FIS son obvias.

En esta perspectiva, el objetivo de este trabajo de tesis es desarrollar e implementar una arquitectura híbrida conocida como Redes Adaptativas basadas en Sistemas de Inferencia Difuso, o simplemente ANFIS (Adaptive Network based Fuzzy Inference System) desarrollada por Jang (1993) [1], la cual sirve de base para la construcción de un conjunto difuso de reglas *if-then* con funciones de pertenencia apropiadas para generar los pares de entrenamiento de entrada-salida estipuladas, dentro de una red neuronal adaptativa.

El objetivo de este capítulo es presentar los fundamentos teóricos necesarios para comprender y poder implementar un controlador neuro-difuso tipo ANFIS presentado en el Capítulo 2, por dicha razón sólo se explican y se abarcan ciertos temas relevantes y fundamentales de los sistemas difusos, un método de mínimos cuadrados para la identificación de los parámetros de un sistema y el algoritmo de aprendizaje por retropropagación para redes neuronales adaptativas, enfocando el estudio en las ideas que son empleadas por el controlador ANFIS. Actualmente existe mucha literatura donde se puede consultar y profundizar los conceptos en los cuales el lector tenga dudas y desconocimientos. Además en este capítulo se realiza una breve explicación de los tipos de sistemas híbridos que surgen al unir los sistemas de inferencia difusos y las redes neuronales artificiales, que comúnmente son conocidos como sistemas o controladores neuro-difusos.

1.2 LÓGICA Y CONTROLADORES DIFUSOS [1]

En este apartado se realiza la explicación de los conceptos fundamentales de lógica y controladores difusos utilizados en la arquitectura ANFIS. Se inicia con ciertas definiciones utilizadas en la formulación de las reglas difusas *if-then*; luego se detalla el funcionamiento de los elementos que conforman un Sistema de Inferencia Difuso (FIS) centrando el estudio en un tipo particular de estos sistemas conocido como Modelo Difuso Sugeno; posteriormente se describen los principales métodos de partición del espacio de entrada de los modelos difusos y al final se realiza una breve explicación del empleo de un FIS como un controlador difuso dentro de un sistema de control.

1.2.1 REGLAS DIFUSAS IF-THEN

Las reglas difusas *if-then* (*si-entonces*) o sentencias condicionales difusas son expresiones de la forma *if A then B* (*si A entonces B*), donde *A* y *B* son las etiquetas de los conjuntos difusos caracterizados por funciones de pertenencias apropiadas. Las reglas difusas combinan uno o más conjuntos difusos de entrada, llamados *antecedentes o premisas (A)*, y les asocian un conjunto difuso de salida, llamado *consecuente o consecuencia (B)*.

Debido a su forma concisa, las reglas difusas *if-then* a menudo se emplean para capturar los modos imprecisos de razonamiento que juegan un papel esencial en la capacidad humana de tomar decisiones en un entorno de incertidumbre e imprecisión. Un ejemplo que describe un hecho simple es

***if** presión es alta **then** volumen es pequeño*

donde, la *presión* y el *volumen* son conocidas como las *variables lingüísticas*, *alta* y *pequeño* son los *valores lingüísticos* o *etiquetas* que son caracterizados por funciones de pertenencia o membresía (MF).

---

[1] La mayoría de los temas tratados en este apartado se basan en la publicaciones de Jang (1993), Jang & Sun (1995), Jang, Sun & Mizutani (1997).

Otra forma de la regla difusa *if-then*, propuesta por Takagi y Sugeno (1985), tiene conjuntos difusos implicados sólo en la parte premisa. Mediante este tipo de regla, se puede describir la fuerza de resistencia de un objeto en movimiento de la siguiente manera:

$$\textbf{\textit{if}}\ \textit{velocidad es alta,}\ \textbf{\textit{then}}\ \textit{fuerza} = k.(velocidad)^2$$

donde, *alta* en la parte premisa es una *etiqueta lingüística* caracterizada por una función de pertenencia apropiada. Sin embargo, la parte consecuente es descrita por una ecuación no-difusa en función de la variable de entrada *velocidad*.

Como se ha visto, las reglas difusas permiten expresar el conocimiento que se dispone sobre la relación entre antecedentes y consecuentes. Para expresar este conocimiento de forma completa normalmente se precisa de varias reglas *if-then*, que se agrupan formando lo que se conoce una *base de reglas*, es decir, el conjunto de reglas que expresan las relaciones conocidas entre antecedentes y consecuentes.

Ambos tipos de reglas difusas *if-then* se han utilizado ampliamente tanto en el modelado y control de sistemas. A través del uso de etiquetas lingüísticas y funciones de pertenencia, una regla difusa puede capturar fácilmente el espíritu de una *"regla de oro"* utilizada por los seres humanos. Desde otro ángulo, debido a la fase de clasificación en la parte de las premisas, cada regla difusa *if-then* puede ser vista como una descripción local del sistema bajo consideración. Las reglas difusas forman una parte fundamental del sistema de inferencia difuso que será descrito a continuación.

1.2.2 SISTEMA DE INFERENCIA DIFUSO (FIS)

Se conoce como Sistema de Inferencia Difuso o FIS *(Fuzzy Inference Systems)* a los sistemas que interpretan las reglas del tipo *if-then* de una base de reglas, con el fin de obtener los valores de salida a partir de los actuales valores de las

variables lingüísticas de entrada al sistema. Estos sistemas se han aplicado con éxito en campos como el control automático, clasificación de datos, análisis de decisiones, sistemas expertos y visión por computador.

Debido a su naturaleza multidisciplinaria, los sistemas de inferencia difusos también son conocidos como sistemas basados en reglas difusas, sistemas expertos difusos, modelos difusos, memorias asociativas difusas (FAM), controladores difusos (cuando se utilizan como controladores) o simplemente sistemas difusos. Básicamente, un sistema de inferencia difusa se compone de cinco bloques funcionales, según se muestra en la Figura 1.1.

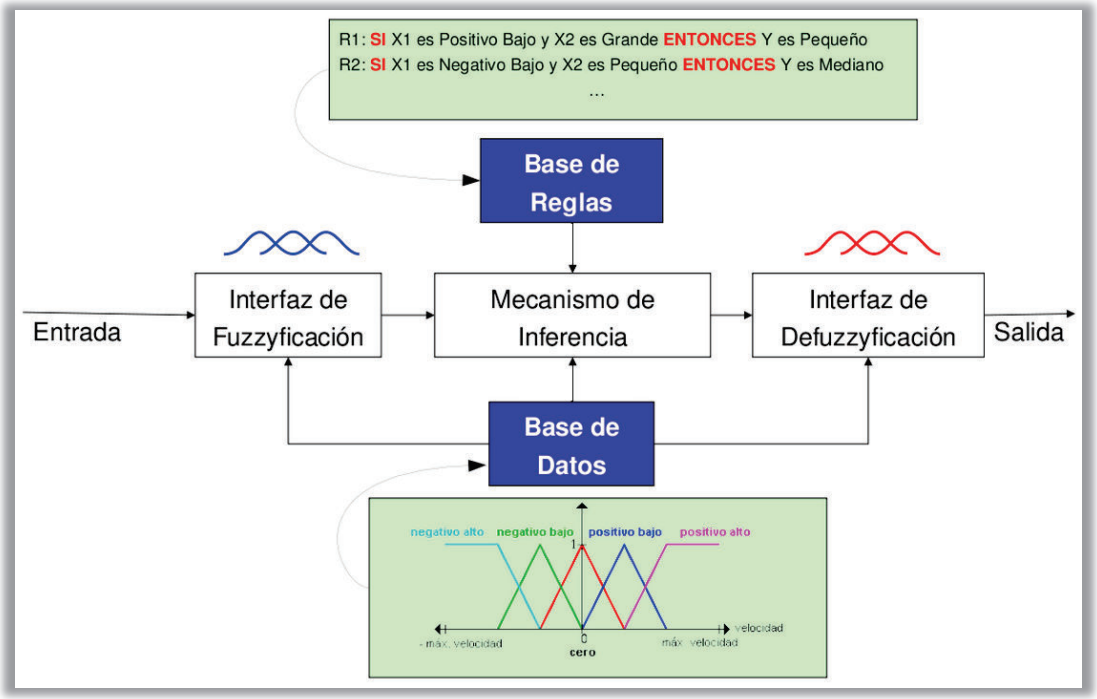


Figura 1.1. Diagrama de bloques de un Sistema de Inferencia Difuso

La Base de Reglas contiene un cierto número de reglas difusas *if-then*. La Base de Datos define las funciones de pertenencia (MF) de los conjuntos difusos utilizados en las reglas difusas. El Mecanismo de Inferencia o Unidad de Toma de Decisiones lleva a cabo las operaciones de inferencia sobre las reglas difusas y una condición dada para obtener una salida o conclusión razonable. La Interfaz de Fusificación transforma las entradas nítidas (entradas no difusas) en grados de equivalencia con los valores lingüísticos, es decir, establece una relación entre

puntos de entrada no difusas al sistema y sus correspondientes conjuntos difusos. La Interfaz de Defusificación transforma los resultados difusos de la inferencia en una salida nítida (salida no difusa).

Por lo general, la Base de Reglas y la Base de Datos se denominan conjuntamente como la Base de Conocimiento. En este sistema se emplea el concepto de *razonamiento difuso* (también conocido como razonamiento aproximado), el cual es un procedimiento de inferencia utilizado para deducir conclusiones a partir de un conjunto de reglas difusas *if-then* y una o más condiciones. Los pasos del razonamiento difuso (operaciones de inferencia sobre reglas difusa *if-then*) realizados por los sistemas de inferencia difusos son:

a. Compara las variables de entrada con las funciones de pertenencia (*membership function MF*) en la parte de la premisa para obtener los valores de pertenencia (o medidas de compatibilidad) de cada etiqueta lingüística. Este paso a menudo es llamado *fusificación*.

b. Combina (a través de un operador específico como la norma-T) los valores de pertenencia de la parte de la premisa para obtener la intensidad de disparo (peso) de cada regla.

c. Genera una salida del consecuente (ya sea difusa o nítida) de cada regla dependiendo la intensidad de disparo. Se dice que una variable es nítida cuando esta puede representarse con un valor numérico.

d. Agrega todas las salidas de los consecuentes para obtener una salida nítida (salida no difusa). Este paso es llamado *defusificación*.

Tenga en cuenta que el sistema básico de inferencia difusa puede receptar entradas difusas o entradas nítidas. A menudo es necesario tener una salida nítida, especialmente en una situación en la que se utiliza un sistema de inferencia difusa como un controlador. Por lo tanto se necesita de una estrategia de defusificación para extraer un valor nítido que mejor resuma un conjunto difuso. Un sistema de inferencia difuso con una salida nítida se muestra en la

Figura 1.2, donde la línea discontinua indica un sistema de inferencia difuso básico con salida difusa y el bloque de defusificación sirve para transformar una salida difusa en una salida nítida.

Con entradas y salidas nítidas, un sistema de inferencia difuso implementa un mapeo no lineal a partir de su espacio de entrada al espacio de salida. Este mapeo se logra mediante una serie de reglas difusas *if-then*, cada una de las cuales describe un comportamiento local. En particular, el antecedente o premisa de cada regla representa una región difusa del espacio de entrada y el consecuente proporciona las salidas correspondientes.

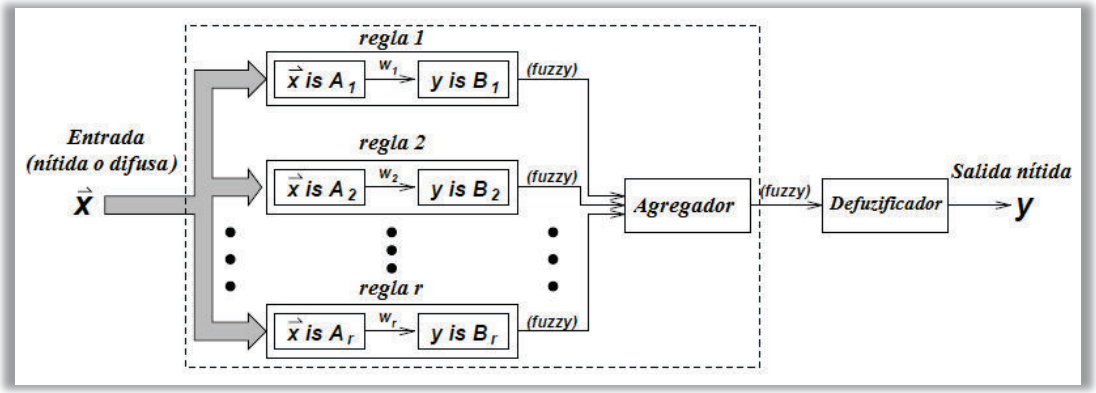


Figura 1.2. Diagrama de bloques equivalente del Sistema de Inferencia Difuso [3]

1.2.3 TIPOS DE SISTEMAS DE INFERENCIA DIFUSOS

Existen diferentes tipos de Sistemas de Inferencia Difusos, los cuales se han utilizado en diversas aplicaciones. Las diferencias básicas pueden estar en los consecuentes de sus reglas difusas, así como en los métodos de agregación y defusificación que emplean. Los nombres que se les han dado normalmente se toman de las personas que primero los propusieron. Dependiendo de los tipos de razonamiento difuso y las reglas difusas empleadas, los sistemas de inferencia difusos se pueden clasificar en tres tipos (Figura 1.3).

Tipo 1 o Modelo Difuso Tsukamoto: La salida total *z* es el peso promedio de la salida nítida de cada regla inducida por la intensidad de disparo de la regla (el producto o mínimo de los grados de coincidencia con la parte premisa) y las funciones de pertenencia de salida. Las funciones de pertenencia de salida utilizadas en este esquema deben ser monótonamente no decrecientes.

Tipo 2 o Modelo Difuso Mamdani: La salida difusa total se obtiene mediante la aplicación de la operación *"max"* a las salidas difusas (cada una de los cuales es igual al mínimo de la intensidad de disparo y al mínimo de la función de pertenencia de salida de cada regla). Varios esquemas se han propuestos para elegir la salida nítida final *z* en base a la salida difusa total; algunos de ellos son el centro del área, la bisectriz del área, la media de la máxima, el criterio de máxima, etc.

Tipo 3 o Modelo Difuso Sugeno: En este tipo de razonamiento difuso se emplean reglas difusas *if-then* de Takagi y Sugeno (1985). La salida de cada regla es una combinación lineal de las variables de entrada más un término constante y la salida final es el peso promedio de la salida de cada regla.

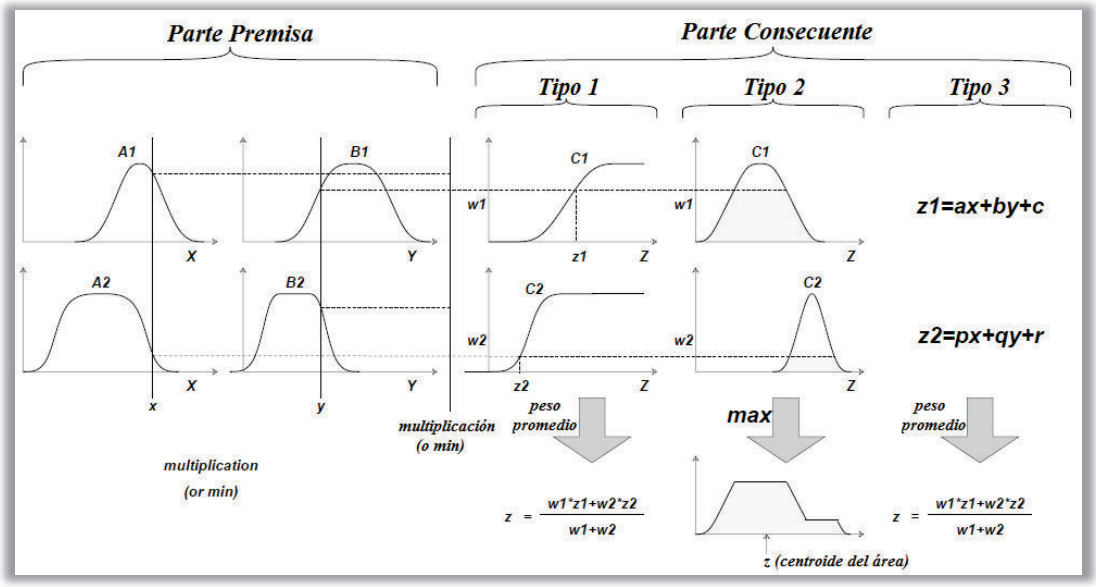


Figura 1.3. Reglas difusas if-then y mecanismos de razonamiento difusos comúnmente usados [1]

La Figura 1.3 utiliza un sistema de inferencia difuso con dos entradas (dos reglas) para mostrar los diferentes tipos de reglas difusas y el razonamiento difuso mencionados anteriormente. Tenga en cuenta que la mayoría de las diferencias se encuentran en la especificación de la parte consecuente (funciones de pertenencia monótonamente no decrecientes o en forma de campana, o funciones nítidas) y por lo tanto los esquemas de defusificación (peso promedio, centroide del área, etc.) también son diferentes.

El caso particular del que se ocupa el presente trabajo, se basa en un sistema de inferencias difuso Tipo 3 o Modelo Difuso Sugeno. Por ello, a continuación solo se describe en qué consiste este sistema difuso, que será retomado nuevamente en el Capítulo 2, al hacer la descripción de la arquitectura ANFIS.

1.2.3.1 Modelo Difuso Sugeno

El modelo difuso Sugeno (también conocido como *modelo difuso TSK)* fue propuesto por Takagi, Sugeno y Kang, en un esfuerzo por desarrollar un método sistemático para generar reglas difusas a partir de un conjunto de datos de entradas y salidas dadas (Bravo & García, 2002) [4]. Una regla difusa típica en un modelo difuso Sugeno tiene la forma:

$$\textit{if } x \textit{ is } A \textit{ and } y \textit{ is } B \textit{ then } z = f(x, y)$$

donde, *A* y *B* son conjuntos difusos en la parte de antecedentes y $z = f(x, y)$ es una función nítida en la parte de consecuente. Normalmente $f(x, y)$ es un polinomio dependiente de las variables de entrada x e y, pero puede ser cualquier función siempre que pueda describir apropiadamente la salida del modelo dentro de la región difusa especificada por el antecedente de la regla. Cuando $f(x, y)$ es un polinomio de primer orden, el sistema de inferencia difuso resultante es llamado un *modelo difuso Sugeno de primer orden*; si f es una constante, entonces se tiene un *modelo difuso Sugeno de orden cero* (Bravo & García, 2002) [4]. La Figura 1.4 muestra el procedimiento de razonamiento difuso para un modelo difuso Sugeno de primer orden.

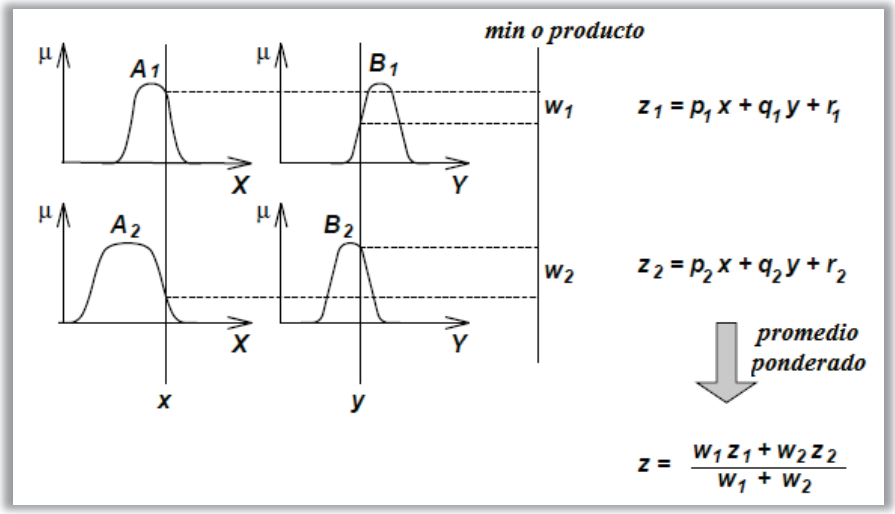


Figura 1.4. Modelo difuso Sugeno de primer orden [3]

Tenga en cuenta que los bloques del agregador y del defusificador representados en la Figura 1.2 son reemplazados por la operación *promedio ponderado (weighted average)* para obtener la salida total. En la práctica, la operación de promedio ponderado se sustituye con la operación *suma ponderada (weighted sum)* es decir, $z = \omega_1 z_1 + \omega_2 z_2$, con el fin de reducir la carga de cálculo, especialmente en el entrenamiento del sistema de inferencia difuso. Sin embargo, esta simplificación podría conducir a la pérdida de significados lingüísticos de las funciones de pertenencia MF, a menos que la suma de las intensidades de disparo (es decir, $\sum_i \omega_i$) sea cercana a la unidad (Bravo & García, 2002) [4].

Para aclarar estos conceptos, supóngase un ejemplo arbitrario de un modelo difuso Sugeno de primer orden con dos entradas, una salida y cuatro reglas, éste se expresa como:

a. Si *X* es pequeño e *Y* es pequeño, entonces *z = -x + y + 1*
b. Si *X* es pequeño e *Y* es grande, entonces *z = -y + 3*
c. Si *X* es grande e *Y* es pequeño, entonces *z = -x - 3*
d. Si *X* es grande e *Y* es grande, entonces *z - x + y + 2*

En la Figura 1.5 se indica el razonamiento difuso aplicado a este ejemplo.

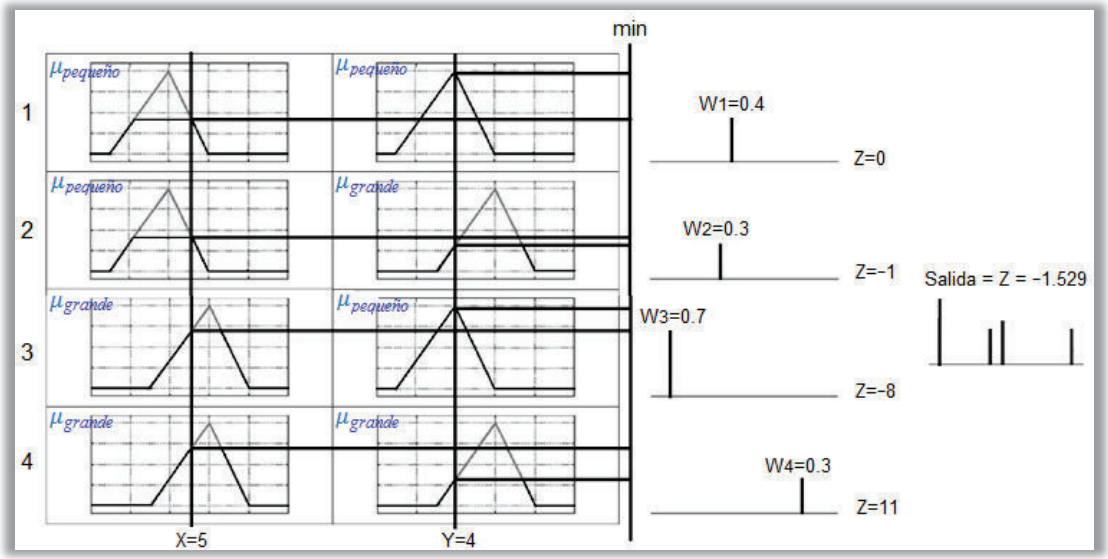


Figura 1.5. Ejemplo de un Modelo Difuso tipo Sugeno de primer orden

Donde z_1 y ω_1 son calculados de la siguiente manera:

$$z_1 = -5 + 4 + 1 = 0 \qquad \omega_1 = \min(0.7; 0.4) = 0.4$$

Se realiza el mismo procedimiento con el resto de variables y al final se obtiene la salida total como

$$z = \frac{\omega_1 z_1 + \omega_2 z_2 + \omega_3 z_3 + \omega_4 z_4}{\omega_1 + \omega_2 + \omega_3 + \omega_4} = \frac{0.4(0) + 0.3(-1) + 0.7(-8) + 0.3(11)}{0.4 + 0.3 + 0.7 + 0.3} = -1.529$$

1.2.4 MÉTODOS DE PARTICIÓN DEL ESPACIO DE ENTRADA DE LOS MODELOS DIFUSOS

En un Sistema de Inferencias Difuso, el antecedente de una regla define una región difusa local, mientras que el consecuente describe el comportamiento dentro de esa región mediante varios componentes. El espacio difuso de entrada se puede particionar con diferentes métodos para formar los antecedentes de las reglas difusas (Bravo & García, 2002) [4]. Los componentes consecuentes podrían ser una función de pertenencia MF (modelos difusos Tsukamoto y

Mamdani), una constante (modelo Sugeno de orden cero) o una ecuación lineal (modelo Sugeno de primer orden). Diferentes componentes consecuentes resultan en diferentes sistemas de inferencia difusos, pero sus antecedentes son siempre los mismos. Por lo tanto, los siguientes métodos de partición de los espacios de entrada utilizados para formar los antecedentes de las reglas difusas son aplicables a todos los tres tipos de sistemas de inferencia difusos antes mencionados. Los tres métodos básicos son: partición de rejilla, partición de árbol y partición dispersa. Un ejemplo de estos métodos de partición en el espacio de entrada de dos dimensiones se muestra en la Figura 1.6.

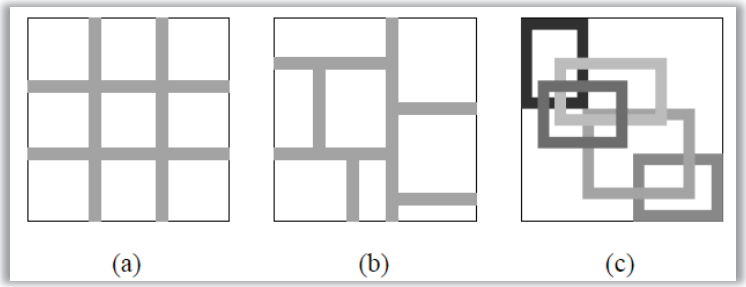


Figura 1.6. Varios métodos de particionar el espacio de entrada. (a) Partición de rejilla; (b) Partición de árbol; (c) Partición dispersa [3]

Partición de Rejilla (*Grid Partition*): La Figura 1.6(a) ilustra una partición de rejilla típica en un espacio de entrada de dos dimensiones. Este método tiene fácil interpretabilidad y es el más ampliamente utilizado para la generación de reglas difusas. Esta estrategia de partición sólo requiere un pequeño número de funciones de pertenencias MF para cada variable de entrada; sin embargo, tiene algunos problemas cuando se tiene un número moderadamente grande de entradas, puesto que el número de reglas difusas crece exponencialmente con la dimensión de entrada, según se indica en la siguiente relación:

$$N_{reglas} = {N_{MF}}^{N_{in}}$$

donde, N_{reglas} es el número de reglas, N_{MF} el número de funciones de pertenencia por entrada y N_{in} el número de entradas. Por ejemplo, un modelo difuso con 10 entradas y 2 MF por cada entrada, podría resultar en $2^{10} = 1024$ reglas difusas *if-then*, lo cual es prohibitivamente grande. Dado que cada regla

tiene pocos parámetros para ajustar, existen demasiados parámetros por adaptar durante el proceso de aprendizaje. Demasiadas reglas difusas también dañan la interpretabilidad del sistema difuso. Este problema, normalmente se conoce como *la maldición de la dimensionalidad* y puede ser corregido por otras estrategias de partición (Jang, Sun & Mizutani, 1997) [2].

La partición de rejilla es una estructura estática y es apropiada cuando se cuenta con un conjunto de datos de entrada dimensionalmente pequeño y es elegida a menudo en el diseño de un controlador difuso, que generalmente implica sólo algunas variables de estado como las entradas al controlador.

Partición de Árbol (*Tree Partition*): La Figura 1.5(b) muestra una partición de árbol típica, en la cual cada región se puede especificar en forma única a través de un árbol de decisiones. Por lo general se suelen emplear métodos heurísticos basados en la distribución de ejemplos de entrenamiento o métodos de identificación de parámetros para encontrar una estructura de árbol adecuada.
Esta partición resuelve el problema de un incremento exponencial del número de reglas, sin embargo se requieren más funciones de pertenencia por cada entrada para definir tales regiones difusas y esas MF generalmente no tienen un significado lingüístico claro tal como “pequeño”, “grande”, etc. (Bravo & García, 2002) [4].

Partición Dispersa (*Scatter Partition*): Como se muestra en la Figura 1.5(c), este tipo de partición cubre un subconjunto de todo el espacio de entrada que caracteriza a una región de posible ocurrencia de los vectores de entrada, así puede limitar el número de reglas a una cantidad razonable, donde el número de reglas difusas es igual al número de funciones de pertenencia de cada entrada:

$$N_{reglas} = N_{MF}$$

La partición dispersa suele generar menos regiones difusas en comparación a las técnicas de partición de rejilla y de árbol, debido a la característica de agrupamiento natural de los patrones de entrenamiento. Los algoritmos de

agrupamiento difusos (*fuzzy clustering algorithms*) forman una familia de técnicas de generación de reglas. Los datos de entrenamiento se reúnen en grupos homogéneos y una regla se asocia a cada grupo. Los conjuntos difusos no son compartidos por las reglas, pero cada uno de ellos se adapta a una regla en particular. Por lo tanto, los conjuntos difusos resultantes suelen ser difíciles de interpretar y esto hace difícil estimar la función de transformación global directamente desde el consecuente de salida de cada regla (Jang, Sun & Mizutani, 1997) [2].

1.2.5 CONTROLADORES DIFUSOS

Como ya se ha mencionado anteriormente cuando los Sistemas de Inferencia Difusos son utilizados como controladores estos reciben el nombre de Controladores Difusos o FCL (*Fuzzy Logic Controllers*) y son sin duda la aplicación más extendida de la lógica difusa en la Ingeniería de Control. Como se puede observar en la Figura 1.7, para controlar un proceso o sistema se hace uso de un módulo controlador, que recibe como entrada una o varias variables de control llamadas generalmente referencias, $\boldsymbol{r}(t)$, y una y varias variables de salida del propio proceso, $\boldsymbol{y}(t)$, produciendo como salida una o varias variables, que se conocen como señales de control o actuadores $\boldsymbol{u}(t)$. Normalmente el objetivo del sistema de control es mantener $\boldsymbol{r}(t) = \boldsymbol{y}(t)$ (Martin del Brio & Sanz, 2008) [5].

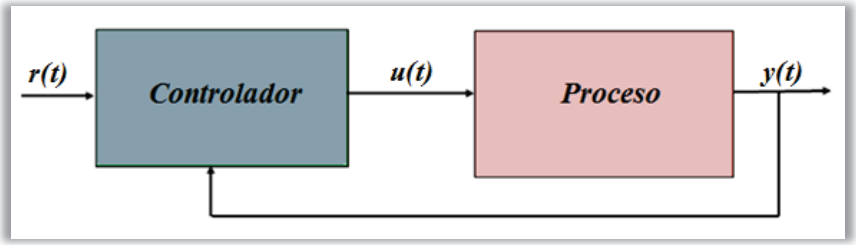


Figura 1.7. Control directo de un proceso o sistema

Por ejemplo, en el caso de una calefacción doméstica, el controlador recibe una consigna de temperatura que fija el usuario y mide la temperatura de la habitación por medio de un sensor. En función de los valores de estas entradas, el

controlador conecta o desconecta el sistema de calentamiento, si la calefacción es eléctrica actúa sobre los radiadores de cada habitación y si es de gas enciende o apaga el quemador. La estructura típica de un controlador basado en un sistema difuso puede verse en la Figura 1.8.

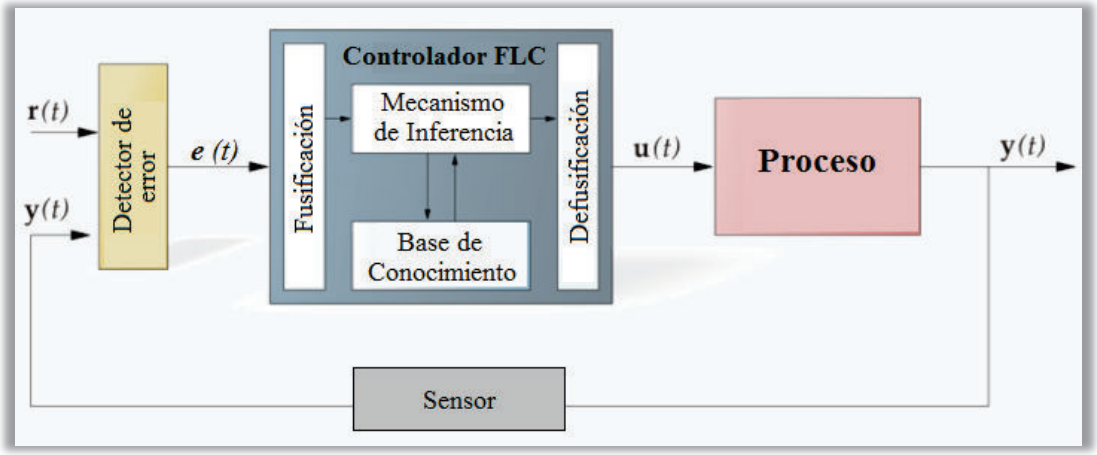


Figura 1.8. Estructura de un controlador difuso FLC

Aunque no se muestra en la Figura 1.8, se puede colocar un primer bloque antes del controlador FLC para realizar un preprocesado de las variables de entrada $e(t)$ y así proporcionar el vector de entradas al controlador difuso o FLC. El controlador difuso aplica la entrada que es recibida por la base de reglas, para obtener la salida. Finalmente, esta salida puede requerir un procesado final (potsprocesado), con el fin de adecuarla al proceso que se ha de controlar (Martin del Brio & Sanz, 2008) [5].

La estructura interna de un controlador difuso ya fue explicada en el apartado 1.2.2. Un primer elemento llamado fusificador realiza la conversión de valores discretos a términos difusos. Su salida es utilizada por el mecanismo de inferencia difuso para aplicarla a cada una de las reglas *if-then* de la base de reglas, de acuerdo al tipo de sistema de inferencia difuso empleado (modelo difuso Tsukamoto, Mandani o Sugeno). Finalmente, el defusificador transformará estos conjuntos difusos en un valor nítido (valor numérico).

1.3 MÉTODO DE MÍNIMOS CUADRADOS [2]

El método de mínimos cuadrados es una herramienta matemática poderosa y bien desarrollada que se ha propuesto y utilizado en una variedad de áreas por décadas, incluyendo el control adaptativo, procesamiento de señales y las estadísticas. Hoy en día todavía demuestra ser una herramienta esencial e indispensable para la construcción de modelos matemáticos lineales y los mismos conceptos fundamentales también pueden extenderse a modelos no lineales. Por lo tanto, basta con decir que el método de mínimos cuadrados lineal proporciona la base matemática más básica e importante para resolver muchos problemas de modelado y control de sistemas.

El método de mínimos cuadrados es utilizado por el algoritmo de aprendizaje de la red ANFIS para determinar ciertos parámetros lineales dentro de la estructura neuronal, por dicha razón en este apartado se realiza una descripción de los fundamentos matemáticos empleados en este tipo de arquitectura neuro-difusa.

1.3.1 INTRODUCCIÓN A LA IDENTIFICACIÓN DE SISTEMAS

El problema de determinar un modelo matemático para un sistema desconocido (también referido como el sistema de destino) mediante la observación de sus pares de datos de entrada y salida se denomina generalmente como identificación del sistema.

Los propósitos de identificación del sistema son múltiples, por ejemplo: predecir el comportamiento de un sistema, como en la predicción de series temporales y el pronóstico del tiempo; explicar las interacciones y relaciones entre entradas y salidas de un sistema; diseñar un controlador basado en el modelo de un sistema, como en el control de un intercambiador de calor o un manipulador robótico; entre otros. Generalmente la identificación del sistema implica dos pasos:

---

[2] Los temas y ecuaciones presentados en este apartado se basan en el Capítulo 5 del libro publicado por Jang, Sun & Mizutani (1997).

a. Identificación de la Estructura**:** en este paso, se necesita aplicar el conocimiento a priori sobre el sistema de destino para determinar una clase de modelos dentro de la cual se lleva a cabo la búsqueda del modelo más adecuado. Por lo general, esta clase de modelos se denota por una función $y = f(\boldsymbol{u}; \boldsymbol{\theta})$, donde y es la salida del modelo, $\boldsymbol{u}$ es el vector de entrada y $\boldsymbol{\theta}$ es el vector de parámetros. La determinación de la función f depende del problema y se basa en la experiencia e intuición del diseñador, así como de las leyes de la naturaleza que rigen el sistema de destino.

b. Identificación de Parámetros: en el segundo paso, la estructura del modelo ya es conocida y todo lo que se necesita hacer es aplicar técnicas de optimización para determinar el vector de parámetros $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$ tal que el modelo resultante $\hat{y} = f\left(\boldsymbol{u}; \widehat{\boldsymbol{\theta}}\right)$ puede describir el sistema apropiadamente, es decir, se aproxime lo mejor posible a la estructura del modelo real $y = f(\boldsymbol{u}; \boldsymbol{\theta})$ (Jang, Sun & Mizutani, 1997) [2].

Si no se tiene ningún conocimiento a priori sobre el sistema de destino, la identificación de la estructura se convierte en un problema difícil y se tiene que seleccionar la estructura por pruebas de ensayo y error. Afortunadamente, en la actualidad se cuenta con mucho conocimiento acerca de las estructuras de la mayoría de los sistemas de ingeniería y procesos industriales; por lo general es posible derivar una clase específica de modelos (función con parámetros), que pueden describir el funcionamiento del sistema de destino. En consecuencia, el problema de identificación del sistema se reduce por lo general a la de identificación de los parámetros. El problema de la identificación de los parámetros es por tanto de gran importancia y en consecuencia este apartado se dedica principalmente a esta clase de problema.

La Figura 1.9 ilustra un diagrama esquemático del proceso de identificación de parámetros, donde una entrada $\boldsymbol{u_i}$ se aplica tanto al sistema como al modelo y la diferencia entre la salida del sistema de destino $\boldsymbol{y_i}$ y la salida del modelo aproximado $\widehat{\boldsymbol{y}}_{\boldsymbol{i}}$ se utiliza de una manera apropiada para actualizar un vector de parámetros $\boldsymbol{\theta}$ y reducir esta diferencia. Tenga en cuenta que el conjunto de datos

compuesto por m pares deseados de entrada y salida $(\boldsymbol{u_i}; \boldsymbol{y_i})$, $i = 1, \ldots, m$, a menudo es conocido como el *conjunto de datos de entrenamiento* o *conjunto de datos de muestra*. En el caso más general, $\boldsymbol{u_i}$ y $\boldsymbol{y_i}$ representan los vectores de entrada y de salida deseadas, respectivamente.

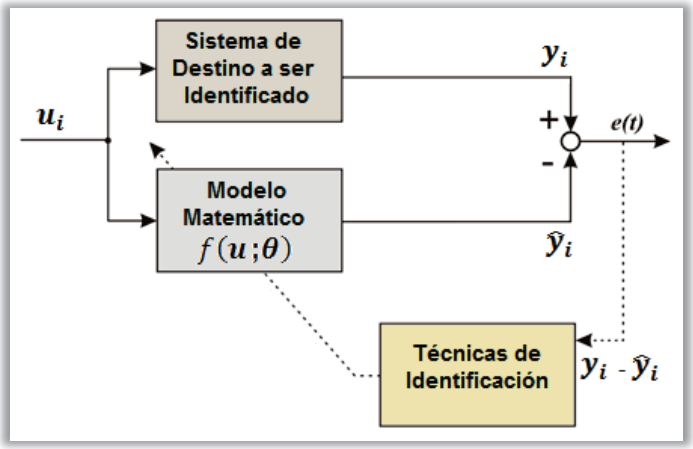


Figura 1.9. Diagrama de bloques para la identificación de parámetros

En general, la identificación del sistema no es un proceso de un solo paso; se tiene que hacer tanto la identificación de la estructura como la identificación de los parámetros varias veces, hasta que se encuentre un modelo satisfactorio, de la siguiente manera:

a. Especifique y parametrice una clase de modelos matemáticos representando el sistema a ser identificado.

b. Realice la identificación de los parámetros para elegir los parámetros que mejor se adapten al conjunto de datos de entrenamiento.

c. Realice las pruebas de validación necesarias para verificar si el modelo identificado responde correctamente a un conjunto de datos desconocidos. Este conjunto de datos no pertenece al conjunto de datos de entrenamiento y se lo conoce como el conjunto de datos de prueba, validación o control.

d. Finalice el procedimiento una vez que los resultados de la prueba de validación sean satisfactorios. De lo contrario, seleccione otra clase de modelos y repita los pasos b al d.

Antes de profundizar en las partes centrales del algoritmo de aprendizaje de la arquitectura ANFIS, se introducirá una clase de métodos estándares de mínimos cuadrados utilizados en la identificación de sistemas con modelos lineales y que en forma general es conocido como Estimador de Mínimos Cuadrados o simplemente LSE por sus siglas en inglés *Least Squares Estimator*.

1.3.2 ESTIMADOR DE MÍNIMOS CUADRADOS

El problema general del método de mínimos cuadrados toma como punto de partida que la salida de un modelo lineal y está definida por la siguiente expresión lineal parametrizada:

$$y = \theta_1 f_1(\boldsymbol{u}) + \theta_2 f_2(\boldsymbol{u}) + \cdots + \theta_n f_n(\boldsymbol{u}) \qquad (1.1)$$

Donde, $\boldsymbol{u} = [u_1, u_2, \dots, u_p]^T$ es el vector de entrada del modelo, $f_1, f_2, \dots, f_n$ son funciones conocidas de $\boldsymbol{u}$ y $\theta_1, \theta_2, \dots, \theta_n$ son los parámetros desconocidos a ser estimados. En estadística, la tarea de ajuste de datos utilizando un modelo lineal se conoce como *regresión lineal*. La ecuación 1.1 también se conoce como la *función de regresión* y los parámetros θ_i son llamados *coeficientes de regresión* (Jang, Sun & Mizutani, 1997) [2].

Para identificar los parámetros desconocidos θ_i, normalmente se deben realizar experimentos con la finalidad de obtener un conjunto de datos de entrenamiento compuesto de pares de datos $\{(\boldsymbol{u}_i; y_i), i = 1, 2, \dots m\}$; éstos representan los pares deseados de entrada y salida del sistema de destino a ser modelado. Al sustituir cada par de datos en la ecuación 1.1, se obtiene un conjunto de m ecuaciones lineales:

$$\begin{aligned} f_1(\boldsymbol{u}_1)\theta_1 + f_2(\boldsymbol{u}_1)\theta_2 + \cdots + f_n(\boldsymbol{u}_1)\theta_n &= y_1 \\ f_1(\boldsymbol{u}_2)\theta_1 + f_2(\boldsymbol{u}_2)\theta_2 + \cdots + f_n(\boldsymbol{u}_2)\theta_n &= y_2 \\ \vdots \qquad\qquad\qquad \vdots \qquad\quad \vdots & \\ f_1(\boldsymbol{u}_m)\theta_1 + f_2(\boldsymbol{u}_m)\theta_2 + \cdots + f_n(\boldsymbol{u}_m)\theta_n &= y_m \end{aligned} \tag{1.2}$$

Usando notación matricial, la ecuación anterior se puede escribir en una forma concisa como:

$$\boldsymbol{A\theta} = \boldsymbol{y} \tag{1.3}$$

donde, $\boldsymbol{A}$ es una matriz de $m\,x\,n$ (algunas veces llamada la *matriz de diseño*), definida como:

$$\boldsymbol{A} = \begin{bmatrix} f_1(\boldsymbol{u}_1) & \cdots & f_n(\boldsymbol{u}_1) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{u}_m) & \cdots & f_n(\boldsymbol{u}_m) \end{bmatrix}$$

$\boldsymbol{\theta}$ es un vector de parámetros desconocidos de $n\,x\,1$:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

$\boldsymbol{y}$ es un vector de salida de $m\,x\,1$:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

La *i-ésima* fila de la matriz aumentada $[\boldsymbol{A} \mid \boldsymbol{y}]$, denotada por $[\boldsymbol{a}_i^T \mid y_i]$, se relaciona con el *i-ésimo* par de datos de entrada y salida $(\boldsymbol{u_i}; y_i)$ mediante la siguiente relación:

$$\boldsymbol{a}_i^T = [f_1(\boldsymbol{u}_i), f_2(\boldsymbol{u}_i), \ldots, f_n(\boldsymbol{u}_i)]$$

Dado que la mayoría de los cálculos se basan en las matrices $\boldsymbol{A}$ y $\boldsymbol{y}$, a veces se

referirá a $(\boldsymbol{a}_i^T; y_i)$ como el par de datos *i-ésimo* del conjunto de datos de entrenamiento.

Para identificar una solución exacta del vector desconocido $\boldsymbol{\theta}$, es necesario que $m \geq n$. Si $\boldsymbol{A}$ es una matriz cuadrada $(m = n)$ e invertible, se puede resolver $\boldsymbol{\theta}$ de la ecuación 1.3 como sigue

$$\boldsymbol{\theta} = \boldsymbol{A^{-1}y} \tag{1.4}$$

Sin embargo, normalmente m es mayor que n, indicando que se tienen más pares de datos que parámetros ajustables. En este caso, una solución exacta que satisfaga las m ecuaciones no siempre es posible, debido a que los datos pueden tener ruido o el modelo puede no ser adecuado para describir al sistema de destino. Así, la ecuación 1.3 se debe modificar para incorporar un *vector de error* $\boldsymbol{e}$ a fin de considerar el ruido aleatorio o el error de modelado, de tal forma que

$$\boldsymbol{A\theta} + \boldsymbol{e} = \boldsymbol{y} \tag{1.5}$$

Ahora, en lugar de encontrar la solución exacta de la ecuación 1.3, se busca obtener un $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$ que minimice la suma del error cuadrado, definido por:

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y_i - \boldsymbol{a}_i^T \boldsymbol{\theta})^2 = \boldsymbol{e}^T \boldsymbol{e} = (\boldsymbol{y} - \boldsymbol{A\theta})^T (\boldsymbol{y} - \boldsymbol{A\theta}) \tag{1.6}$$

donde, $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{A\theta}$ es el vector de error producido para una elección en particular de $\boldsymbol{\theta}$. Debe notarse que $E(\boldsymbol{\theta})$ está en forma cuadrática y tiene un único mínimo en $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$. El siguiente teorema establece una condición necesaria que debe satisfacer el estimador de mínimos cuadrados $\widehat{\boldsymbol{\theta}}$.

Teorema: Estimador de mínimo cuadrados

El error cuadrático en la ecuación 1.6 se minimiza cuando $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$, y es conocido como el Estimador de Mínimos Cuadrados (LSE – *Least Squares Estimator*), el cual satisface la siguiente ecuación normal:

$$\boldsymbol{A^T A\,\widehat{\theta} = A^T y} \tag{1.7}$$

Si $\boldsymbol{A^T A}$ es invertible, $\widehat{\boldsymbol{\theta}}$ es único y está dado por:

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{A^T A})^{-1}\boldsymbol{A^T y} \tag{1.8}$$

Comprobación:

Existen varios métodos disponibles en la literatura para encontrar el estimador de mínimos cuadrados para la ecuación 1.3. Un enfoque sencillo es establecer la derivada de $E(\boldsymbol{\theta})$ con respecto a $\boldsymbol{\theta}$ igual a cero. Obsérvese que el término $\boldsymbol{\theta^T A^T y = y^T A\,\theta}$ es un escalar y $E(\boldsymbol{\theta})$ se puede expandir como:

$$E(\boldsymbol{\theta}) = (\boldsymbol{y^T - \theta^T A^T})(\boldsymbol{y - A\theta}) = \boldsymbol{\theta^T A^T A\theta - 2y^T A\theta + y^T y} \tag{1.9}$$

Luego la derivada de $E(\boldsymbol{\theta})$ es

$$\frac{dE(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = 2\boldsymbol{A^T A\theta - 2A^T y} \tag{1.10}$$

Al establecer $\frac{dE(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = 0$ en $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$, se obtiene la ecuación normal

$$\boldsymbol{A^T A\,\widehat{\theta} = A^T y} \tag{1.11}$$

Si $\boldsymbol{A^T A}$ es invertible, entonces $\widehat{\boldsymbol{\theta}}$ puede ser resuelto por:

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{A^T A})^{-1}\boldsymbol{A^T y} \tag{1.12}$$

A partir de la ecuación 1.9, el error mínimo cuadrado conseguido por $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$ se puede encontrar mediante:

$$E(\widehat{\boldsymbol{\theta}}) = \big(\boldsymbol{y - A\,\widehat{\theta}}\big)^T\big(\boldsymbol{y - A\,\widehat{\theta}}\big) = \boldsymbol{y^T y - y^T A(A^T A)^{-1}A^T y} \tag{1.13}$$

Sin embargo, si $\boldsymbol{A^T A}$ no es invertible, entonces el LSE no es único y se debe utilizar el concepto de *inverso generalizado* para encontrar $\widehat{\boldsymbol{\theta}}$. Sin pérdida de generalidad, aquí se asumirá que $\boldsymbol{A^T A}$ es invertible.

La derivación del teorema anterior se basa en la suposición de que cada elemento del vector de error $\boldsymbol{e}$ tiene el mismo peso al evaluar el error cuadrático total. Una generalización de dicha expresión, considera que cada término del error puede ponderarse de una manera diferente. Específicamente, si se toma a $\boldsymbol{W}$ como la matriz de pesos deseados, la cual es simétrica y definida positiva, entonces, el error cuadrático ponderado es:

$$E_{\boldsymbol{W}}(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{A\theta})^T \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{A\theta}) \tag{1.14}$$

Al minimizar $E_{\boldsymbol{W}}(\boldsymbol{\theta})$ con respecto a $\boldsymbol{\theta}$ se obtiene el *estimador de mínimos cuadrados ponderado* $\widehat{\boldsymbol{\theta}}_{\boldsymbol{W}}$.

$$\widehat{\boldsymbol{\theta}}_{\boldsymbol{W}} = (\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{y} \tag{1.15}$$

Obviamente, $\widehat{\boldsymbol{\theta}}_{\boldsymbol{W}}$ se reduce a $\widehat{\boldsymbol{\theta}}$ cuando $\boldsymbol{W}$ es elegida como una matriz de identidad.

1.3.3 ESTIMADOR DE MÍNIMOS CUADRADOS RECURSIVO

El estimador de mínimos cuadrados obtenido en la sección anterior se puede expresar como

$$\boldsymbol{\theta}_k = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{y} \tag{1.16}$$

donde, por sencillez, no se utiliza el símbolo ^ sobre la variable $\boldsymbol{\theta}_k$. Se asume que el número de filas de $\boldsymbol{A}$ y $\boldsymbol{y}$ es k, este subíndice se ha agregado a la ecuación anterior para denotar el número de pares de datos empleados para el estimador $\boldsymbol{\theta}$. También se puede interpretar al subíndice k como una medida del tiempo si los pares de datos están disponibles en orden secuencial.

Supóngase que un nuevo par de datos $(\boldsymbol{a}^T, y)$ está disponible como la $(k+1)$-*iésima* entrada en el conjunto de datos. Entonces en lugar de utilizar todos los $k+1$ pares de datos disponibles para volver a calcular el estimador de mínimos cuadrados $\boldsymbol{\theta}_{k+1}$, se desea encontrar una manera de emplear el $\boldsymbol{\theta}_k$ ya

disponible, para obtener $\boldsymbol{\theta}_{k+1}$ con un mínimo esfuerzo. En otras palabras, la tarea es determinar una forma de utilizar el nuevo par de datos $(\boldsymbol{a}^T, y)$ para actualizar $\boldsymbol{\theta}_k$ apropiadamente y encontrar $\boldsymbol{\theta}_{k+1}$. Este problema se conoce como *identificación por mínimos cuadrados recursivos* (Jang, Sun & Mizutani, 1997) [2]. Obviamente, $\boldsymbol{\theta}_{k+1}$ se puede expresar como

$$\boldsymbol{\theta}_{k+1} = \left(\begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}\right)^{-1} \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\boldsymbol{y}\\ y\end{bmatrix} \tag{1.17}$$

Para simplificar la notación, se introducen dos matrices de dimensiones $\boldsymbol{P_k}$ y $\boldsymbol{P}_{k+1}$, definidas por:

$$\boldsymbol{P}_k = (\boldsymbol{A}^T\boldsymbol{A})^{-1} \tag{1.18}$$

$$\boldsymbol{P}_{k+1} = \left(\begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}\right)^{-1} = \left(\begin{bmatrix}\boldsymbol{A}^T & \boldsymbol{a}\end{bmatrix} \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{a}^T\end{bmatrix}\right)^{-1} = (\boldsymbol{A}^T\boldsymbol{A} + \boldsymbol{a}\boldsymbol{a}^T)^{-1} \tag{1.19}$$

Estas dos matrices se relacionan entre sí como sigue:

$$\boldsymbol{P}_k^{-1} = \boldsymbol{P}_{k+1}^{-1} - \boldsymbol{a}\boldsymbol{a}^T \tag{1.20}$$

Empleando las ecuaciones 1.18 y 1.19 para definir $\boldsymbol{\theta}_k$ y $\boldsymbol{\theta}_{k+1}$, se obtiene lo siguiente relación:

$$\boldsymbol{\theta}_k = \boldsymbol{P}_k\boldsymbol{A}^T\boldsymbol{y} \tag{1.21}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{P}_{k+1}(\boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{a}y) \tag{1.22}$$

$\boldsymbol{\theta}_{k+1}$ se puede expresar en términos de $\boldsymbol{\theta}_k$ eliminando $\boldsymbol{A}^T\boldsymbol{y}$ en la ecuación 1.22. Así, se tiene la siguiente relación:

$$\boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{P}_k^{-1}\boldsymbol{\theta}_k \tag{1.23}$$

Sustituyendo este resultado en la ecuación 1.22 y aplicando la ecuación 1.20, se obtiene:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{P}_{k+1}(\boldsymbol{P}_k^{-1}\boldsymbol{\theta}_k + \boldsymbol{a}y)\\ \boldsymbol{\theta}_{k+1} &= \boldsymbol{P}_{k+1}[(\boldsymbol{P}_{k+1}^{-1} - \boldsymbol{a}\boldsymbol{a}^T)\boldsymbol{\theta}_k + \boldsymbol{a}y]\\ \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \boldsymbol{P}_{k+1}\boldsymbol{a}(y - \boldsymbol{a}^T\boldsymbol{\theta}_k)\end{aligned} \tag{1.24}$$

De tal forma que $\boldsymbol{\theta}_{k+1}$ queda expresada en función del valor anterior $\boldsymbol{\theta}_k$ y del nuevo par de datos $(\boldsymbol{a}^T, y)$. El nuevo estimador $\boldsymbol{\theta}_{k+1}$ es igual al estimador anterior $\boldsymbol{\theta}_k$ más un término de corrección basado en el nuevo par de datos $(\boldsymbol{a}^T, y)$; este término de corrección es igual a un *vector de ganancia de adaptación* $\boldsymbol{P}_{k+1}\boldsymbol{a}$ multiplicado por el error de predicción producido por el estimador previo, esto es, $y - \boldsymbol{a}^T\boldsymbol{\theta}_k$.

Sin embargo, en la ecuación 1.19, para el cálculo de $\boldsymbol{P}_{k+1}$ se debe obtener una matriz inversa de $n\,x\,n$, lo cual tiene un alto costo computacional. Así, de la ecuación 1.20, se tiene que:

$$\boldsymbol{P}_{k+1} = (\boldsymbol{P}_k^{-1} + \boldsymbol{a}\boldsymbol{a}^T)^{-1} \tag{1.25}$$

Utilizando la fórmula de inversión de una matriz, la cual establece que si $\boldsymbol{A}$ y $\boldsymbol{I} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}$ son matrices cuadradas invertibles, entonces la siguiente igualdad es válida: $(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{I} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1}$. Asumiendo que $\boldsymbol{A} = \boldsymbol{P}_k^{-1}$, $\boldsymbol{B} = \boldsymbol{a}$ y $\boldsymbol{C} = \boldsymbol{a}^T$, se obtiene la siguiente fórmula incremental para $\boldsymbol{P}_{k+1}$:

$$\begin{aligned}\boldsymbol{P}_{k+1} &= \boldsymbol{P}_k - \boldsymbol{P}_k\boldsymbol{a}(\boldsymbol{I} + \boldsymbol{a}^T\boldsymbol{P}_k\boldsymbol{a})^{-1}\boldsymbol{a}^T\boldsymbol{P}_k\\ \boldsymbol{P}_{k+1} &= \boldsymbol{P}_k - \frac{\boldsymbol{P}_k\boldsymbol{a}\boldsymbol{a}^T\boldsymbol{P}_k}{1 + \boldsymbol{a}^T\boldsymbol{P}_k\boldsymbol{a}}\end{aligned} \tag{1.26}$$

En resumen, el *Estimador de Mínimos Cuadrados Recursivo* (*RLSE – Recursive Least Squares Estimator*) para el problema planteado por la expresión $\boldsymbol{A}\boldsymbol{\theta} = \boldsymbol{y}$, donde la *k-ésima* fila $(1 \le k \le m)$ de $[\boldsymbol{A} \mid \boldsymbol{y}]$ denotada por $[\boldsymbol{a}_k^T \mid y_k]$, se obtiene en forma secuencial y puede ser calculado como sigue:

$$\begin{cases} \boldsymbol{P}_{k+1} = \boldsymbol{P}_k - \dfrac{\boldsymbol{P}_k\boldsymbol{a}_{k+1}\boldsymbol{a}_{k+1}^T\boldsymbol{P}_k}{1 + \boldsymbol{a}_{k+1}^T\boldsymbol{P}_k\boldsymbol{a}_{k+1}} \\ \boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{P}_{k+1}\boldsymbol{a}_{k+1}(y_{k+1} - \boldsymbol{a}_{k+1}^T\boldsymbol{\theta}_k) \end{cases} \tag{1.27}$$

donde, $1 \leq k \leq m-1$ y el estimador de mínimos cuadrados total $\widehat{\boldsymbol{\theta}}$ es igual a $\boldsymbol{\theta}_m$, que es el estimador que emplea todos los m pares de datos.

Para iniciar el algoritmo de la ecuación 1.27, se deben seleccionar los valores iniciales de $\boldsymbol{\theta}_o$ y $\boldsymbol{P}_o$. Una forma de evitar la determinación de estos valores iniciales es recolectar los primeros n puntos de datos y resolver $\boldsymbol{\theta}_n$ y $\boldsymbol{P}_n$ directamente a partir de las siguientes ecuaciones:

$$\begin{cases} \boldsymbol{P}_n = (\boldsymbol{A}_n^T \boldsymbol{A}_n)^{-1} \\ \boldsymbol{\theta}_n = \boldsymbol{P}_n \boldsymbol{A}_n^T \boldsymbol{y}_n \end{cases}$$

donde $[\boldsymbol{A}_n | \boldsymbol{y}_n]$ es la matriz de datos, compuesta de los primeros n pares de datos. Así, se puede iniciar las iteraciones del algoritmo desde el $(n+1)$-*ésimo* par de datos. Sin embargo, algunas veces es más conveniente usar las fórmulas recursivas de la ecuación 1.27 durante el proceso de identificación. Para ello, hay que notar que:

$$\boldsymbol{P}_k = (\boldsymbol{P}_o + \boldsymbol{A}_k^T \boldsymbol{A}_k)^{-1} \tag{1.28}$$

y el valor correspondiente de $\boldsymbol{\theta}_k$ es:

$$\boldsymbol{\theta}_k = \boldsymbol{P}_k + (\boldsymbol{A}_k \boldsymbol{y}_k + \boldsymbol{P}_o^{-1} \boldsymbol{\theta}_o) \tag{1.29}$$

donde, $[\boldsymbol{A}_k | \boldsymbol{y}_k]$ es la matriz de datos, compuesta de k pares de datos. Si se selecciona $\boldsymbol{P}_o = \alpha \boldsymbol{I}$ se tiene que:

$$\lim_{\alpha \to \infty} \boldsymbol{P}_o^{-1} = \lim_{\alpha \to \infty} \frac{1}{\alpha} \boldsymbol{I} = 0$$

Por consiguiente, eligiendo un valor grande para α, se pueden forzar las ecuaciones 1.28 y 1.29 a ser arbitrariamente cercanas a la ecuación 1.27, independientemente del valor de $\boldsymbol{\theta}_o$. En la práctica y por conveniencia, $\boldsymbol{\theta}_o$ es normalmente una matriz cero o nula.

1.3.4 LSE RECURSIVO PARA SISTEMAS VARIANTES EN EL TIEMPO

De la ecuación 1.18, se tiene que $\boldsymbol{P}_k = (\boldsymbol{A}^T\boldsymbol{A})^{-1}$, donde k es el número de pares de datos encontrado hasta ahora y también representa el número de filas de $\boldsymbol{A}$. Si k es mayor que el número de parámetros de ajuste contenidos en $\boldsymbol{\theta}$ y los pares de datos contienen suficiente información, entonces $\boldsymbol{A}^T\boldsymbol{A}$ es generalmente definida positiva y como k tiende al infinito, el término $\frac{1}{k}\boldsymbol{A}^T\boldsymbol{A}$ se aproxima a una matriz constante invertible. Por lo tanto, se tiene

$$\lim_{k\to\infty} \boldsymbol{P}_k = \lim_{k\to\infty} \frac{1}{k}\left(\frac{1}{k}\boldsymbol{A}^T\boldsymbol{A}\right)^{-\mathbf{1}} = 0$$

lo que indica que la ganancia de adaptación $\boldsymbol{P}_{k+1}\boldsymbol{a}_{k+1}$ en la ecuación 1.27, disminuye en cada iteración. Esta es una consecuencia directa del error cuadrado definido en la ecuación 1.6, que trata cada componente de error por igual. Para los sistemas invariantes en el tiempo, esto funciona bien, ya que el decremento de la ganancia de adaptación significa que se está acercando al punto óptimo en el espacio de parámetros. Sin embargo, para sistemas variantes en el tiempo, este no es el caso, ya que el decremento de la ganancia de adaptación no puede realizar el seguimiento de los parámetros óptimos cambiantes. Una forma sencilla de resolver este problema es restablecer la matriz $\boldsymbol{P}_k$ a $\boldsymbol{P}_o$ de vez en cuando, ya que el LSE converge rápidamente a los parámetros óptimos actuales. Por supuesto, el tiempo obvio para restablecer $\boldsymbol{P}_k$ es cuando se sospecha que se ha producido un cambio de parámetro significativo (Jang, Sun & Mizutani, 1997) [2].
Otra forma de hacer frente a los sistemas variantes en el tiempo es introducir un factor de olvido λ, el cual pone énfasis en los datos más recientes:

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{m} \lambda^{m-i}(y_i - \boldsymbol{a}_i^T\boldsymbol{\theta})^2 = (\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\theta})^T\boldsymbol{W}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\theta}) \qquad (1.30)$$

donde, $\boldsymbol{W}$ es una matriz diagonal dada por:

$$\boldsymbol{W} = \begin{bmatrix} \lambda^{m-1} & 0 & \dots & 0 \\ 0 & \lambda^{m-2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

y $0 < \lambda \leq 1$. De la ecuación 1.15, el LSE correspondiente que minimiza la medida del error ponderado anterior se define por

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{y} \tag{1.31}$$

Para obtener las fórmulas para un LSE recursivo con un factor de olvido, nuevamente se define

$$\boldsymbol{\theta}_k = (\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{y} \tag{1.32}$$

donde, $[\boldsymbol{A} \mid \boldsymbol{y}]$ contiene los k pares de datos y $\boldsymbol{\theta}_k$ es el LSE que usa estos k pares de datos. Entonces se tiene:

$$\boldsymbol{\theta}_{k+1} = \left(\begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\lambda\boldsymbol{W} & 0 \\ 0 & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}^T\end{bmatrix}\right)^{-1} \begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\lambda\boldsymbol{W} & 0 \\ 0 & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{y} \\ y\end{bmatrix}$$

$$\boldsymbol{\theta}_{k+1} = (\lambda\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A} + \boldsymbol{a}\boldsymbol{a}^T)^{-1}(\lambda\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{y} + \boldsymbol{a}y)$$

donde, $(\boldsymbol{a}^T; y)$ es el $(k+1)$-*ésimo* par de datos. Para simplificar la notación, se vuelve a introducir las variables $\boldsymbol{P}_k$ y $\boldsymbol{P}_{k+1}$, que se definen de forma ligeramente diferente en relación a lo anterior:

$$\boldsymbol{P}_k = (\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A})^{-1}$$

$$\boldsymbol{P}_{k+1} = \left(\begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}^T\end{bmatrix}^T \begin{bmatrix}\lambda\boldsymbol{W} & 0 \\ 0 & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{a}^T\end{bmatrix}\right)^{-1} = (\lambda\boldsymbol{A}^T\boldsymbol{W}\boldsymbol{A} + \boldsymbol{a}\boldsymbol{a}^T)^{-1}$$

donde, $\boldsymbol{P}_k$ y $\boldsymbol{P}_{k+1}$ se relacionan a través de

$$\lambda\boldsymbol{P}_k^{-1} = \boldsymbol{P}_{k+1}^{-1} - \boldsymbol{a}\boldsymbol{a}^T \tag{1.33}$$

Utilizando $\boldsymbol{P}_k$ y $\boldsymbol{P}_{k+1}$, se puede reescribir $\boldsymbol{\theta_k}$ y $\boldsymbol{\theta_{k+1}}$ de la siguiente manera:

$$\boldsymbol{\theta}_k = \boldsymbol{P}_k \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{y} \tag{1.34}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{P}_{k+1}(\lambda \boldsymbol{A}^T \boldsymbol{W} \boldsymbol{y} + \boldsymbol{a} y) \tag{1.35}$$

Si se elimina $\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{y}$ y $\boldsymbol{P}_k$ de las tres ecuaciones anteriores, se tiene la fórmula recursiva para $\boldsymbol{\theta}_{k+1}$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{P}_{k+1} \boldsymbol{a}(y - \boldsymbol{a}^T \boldsymbol{\theta}_k) \tag{1.36}$$

donde, $\boldsymbol{P}_{k+1}$ puede ser expandida a partir de la ecuación 1.33, utilizando la fórmula de inversión de matriz de Lemma se tiene:

$$\boldsymbol{P}_{k+1} = \frac{1}{\lambda}\left(\boldsymbol{P}_k - \frac{\boldsymbol{P}_k \boldsymbol{a} \boldsymbol{a}^T \boldsymbol{P}_k}{\lambda + \boldsymbol{a}^T \boldsymbol{P}_k \boldsymbol{a}}\right) \tag{1.37}$$

Es obvio que la ecuación 1.37 se reduce a la ecuación 1.26 si $\lambda = 1$. Si λ es pequeño, entonces los datos recientes se ponderan y el algoritmo es más capaz de seguir los parámetros variantes en el tiempo. Sin embargo, al mismo instante los estimadores también pueden fluctuar para reflejar ruido y perturbaciones. Por lo tanto, el valor de λ es a menudo dependiente de la tarea a realizar y tiene que ser determinado experimentalmente (Jang, Sun & Mizutani, 1997) [2].

Para iniciar el algoritmo de las ecuaciones 1.36 y 1.37, se deben seleccionar los valores iniciales de $\boldsymbol{\theta}_o$ y $\boldsymbol{P}_o$, empleando las mismas técnicas aplicadas en el LSE recursivo e incluyendo el factor de olvido λ.

1.4 REDES NEURONALES ADAPTATIVAS

El propósito general de este trabajo no es el de profundizar todas las temáticas relacionadas a las redes neuronales artificiales, sin embargo interesa estudiar el comportamiento y los algoritmos de aprendizaje empleados por una clase en particular de estas redes conocida como redes adaptivas, ya que una red ANFIS no es más que un tipo especial de red adaptativa.

En esta sección se describe la arquitectura y los algoritmos de aprendizaje de las redes adaptativas, que de hecho pertenecen a un superconjunto de redes neuronales con capacidad de aprendizaje supervisado. En primera instancia se va explicar el algoritmo de aprendizaje de retropropagación aplicado a una red Perceptron Multicapa MLP para posteriormente emplear un análisis similar en una red adaptativa puesto que esta es también una red multicapa. Dado que la regla básica de aprendizaje de una red adaptativa se basa en el método del gradiente, que es notoria por su lentitud y la tendencia a quedar atrapado en mínimos locales, en este apartado también se explica una regla de aprendizaje híbrida que puede acelerar el proceso de aprendizaje sustancialmente.

1.4.1 ARQUITECTURA

Como su nombre lo indica, una red adaptativa (*adaptive network*), es una estructura de red cuyo comportamiento global de entrada-salida queda determinado por una colección de parámetros modificables. Específicamente, la configuración de una red adaptiva (ver Figura 1.10) se compone de un conjunto de nodos conectados por enlaces direccionados, donde cada nodo es una unidad de proceso que realiza una función particular en las señales que entran a él, para generar una sola salida de nodo y cada enlace especifica la dirección del flujo de señales de un nodo a otro (Bravo & García, 2002) [4]. La Figura 1.10 ilustra una típica red adaptativa multicapa con dos entradas y dos salidas.

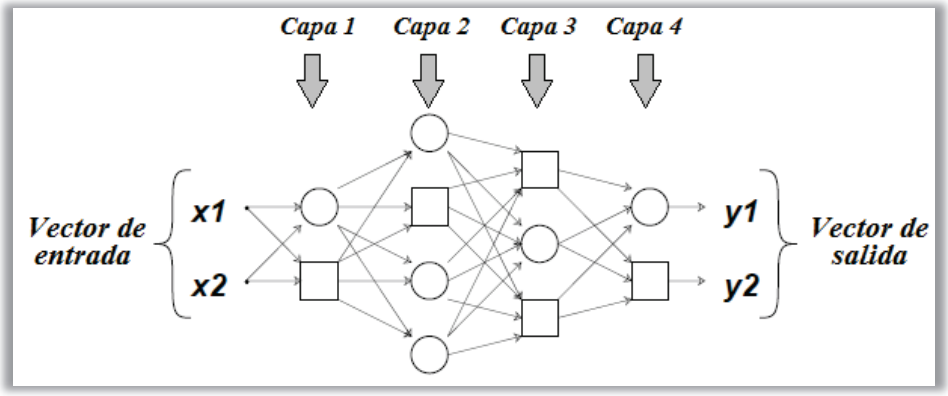


Figura 1.10. Un ejemplo de red adaptativa [1]

Generalmente, una función de nodo es una función parametrizada con parámetros modificables; cambiando dichos parámetros, se modifica la función de nodo y en consecuencia el comportamiento global de la red adaptiva. La regla de aprendizaje determina la forma en que estos parámetros deben ajustarse para minimizar una medida de error dada, la cual es una expresión matemática que mide la desviación entre la salida real de la red y una salida deseada (Bravo & García, 2002) [4].

Los parámetros de una red adaptativa están distribuidos en sus nodos, por lo que cada nodo tiene un conjunto de parámetros locales ($\alpha, \beta, \gamma, ...$). La unión de estos conjuntos de parámetros locales forman el conjunto de parámetros total de la red. Si el conjunto de parámetros de un nodo tiene elementos ajustables, entonces su función de nodo depende de los valores de estos parámetros; en este trabajo se usará un cuadrado para representar este tipo de *nodo adaptivo*. Por otra parte, si un nodo tiene un conjunto de parámetros sin elementos ajustables, entonces su función es fija y se representa con un círculo (*nodo fijo*). Los enlaces de una red adaptiva se utilizan para señalar la dirección de propagación de las salidas de los nodos; generalmente no hay pesos o parámetros asociados con ellos (Bravo & García, 2002) [4].

Las redes adaptivas generalmente se clasifican en dos categorías de acuerdo al tipo de enlaces que tienen: *redes adaptativas anticipativas* (*feedforward adaptive netwroks*) y *redes adaptativas recurrentes* (*recurrent adaptive netwroks*). En la primera de ellas, la información fluye o se propaga en un único sentido desde los nodos de entrada hasta llegar a la capa de salida de la red neuronal. La red adaptativa de la Figura 1.10 es un ejemplo de una red con propagación hacia adelante (red anticipativa). Si existe algún enlace de realimentación que forme una trayectoria circular en una red, entonces la red es recurrente (Jang, Sun & Mizutani, 1997) [2].

En la representación en capas de la red adaptativa anticipativa de la Figura 1.10, se puede observar que no existen enlaces (vínculos) entre los nodos de una misma capa y además las salidas de los nodos en una capa específica están

siempre conectadas a los nodos de las capas sucesivas. Esta representación es generalmente preferida debido a su modularidad, puesto que los nodos de una misma capa tienen la misma funcionalidad o generan el mismo nivel de abstracción sobre los vectores de entrada.

Conceptualmente, una red adaptativa con propagación hacia delante (red anticipativa) realiza una transformación estática entre sus espacios de entrada y de salida; esta transformación puede ser una relación lineal simple o una altamente no lineal, dependiendo de la estructura de la red y la función de cada nodo. El objetivo de este tipo de configuración es lograr construir una red neuronal que realice una transformación no lineal y que esté regulada por un conjunto de pares de datos deseados de entrada-salida de un sistema que se quiera modelar. A este conjunto de datos normalmente se les conoce como *conjunto de datos de entrenamiento* y a los procedimientos que se utilizan para ajustar los parámetros de optimización del desempeño de la red, normalmente se les llama *reglas de aprendizaje o algoritmos de adaptación* (Jang, Sun & Mizutani, 1997) [2].

El desempeño de una red se determina midiendo el error entre la salida deseada y la salida de la red bajo las mismas condiciones de entrada. A esta variable se le llama la *medida del error* y puede tomar diferentes formas para diferentes aplicaciones. En general, una regla de aprendizaje se deriva al aplicar una técnica de optimización específica a una medida de error dada (Bravo & García, 2002) [4].

1.4.2 REGLA DE APRENDIZAJE DE RETROPROPAGACIÓN PARA UNA RED MULTICAPA MLP [3]

En la Figura 1.11 se presenta la estructura del Perceptron Multicapa (MLP de sus siglas en inglés *Multi Layer Perceptron*), que a diferencia del Perceptron y del Adeline, posee al menos tres niveles de neuronas, el primero es el de entrada, luego viene un nivel o capa oculta y, finalmente, el nivel o capa de salida. Se puede proponer más de una capa oculta, pero en general no se recomienda pues

[3] Los temas y ecuaciones presentados en este apartado se basan en el Capítulo 3 del libro publicado por Caicedo & Lopez (2008).

se aumenta fuertemente la complejidad computacional del algoritmo de aprendizaje y para la gran mayoría de las aplicaciones prácticas, es suficiente un MLP con una única capa oculta.

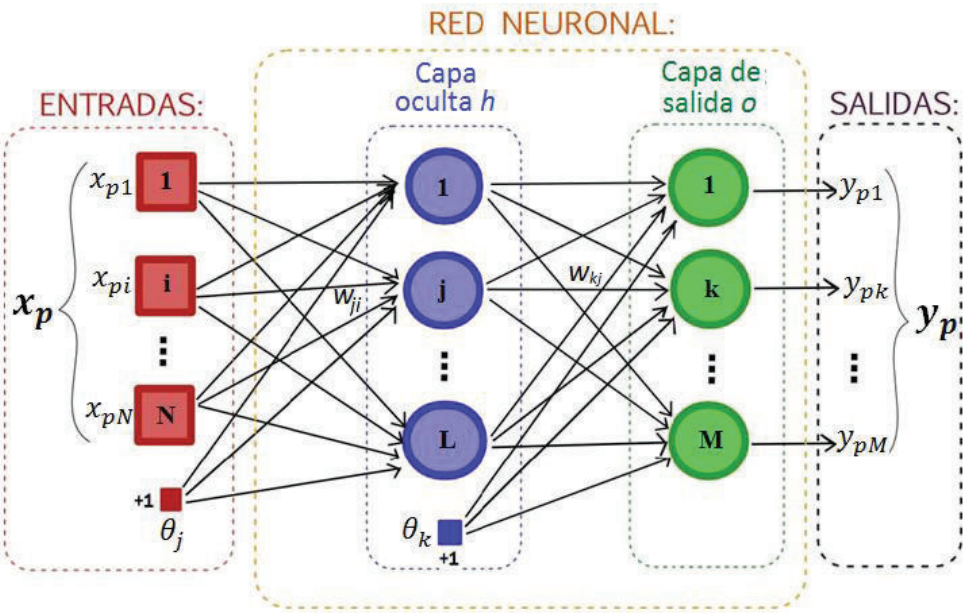


Figura 1.11. Arquitectura general de una red multicapa

En las redes neuronales artificiales, el término conectividad se refiere a la forma como una neurona de una capa cualquiera está interconectada con las neuronas de la capa previa y la siguiente. Para el MLP, la conectividad es total porque si se toma una neurona del nivel de entrada, ésta estará conectada con todas las neuronas de la capa oculta siguiente, una neurona de la capa oculta tendrá conexión con todas las neuronas de la capa anterior y de la capa siguiente. Para la capa de salida, sus neuronas estarán conectadas con todas las neuronas de la capa oculta previa, según se muestra en la Figura 1.11. Por lo general, se le implementan unidades de tendencia o umbral $(\theta_j \; y \; \theta_k)$ con el objetivo de hacer que la superficie de separación no se quede anclada en el origen del espacio $n\text{-}dimensional$ en donde se esté realizando la clasificación. La función de activación que utilizan las neuronas de una red MLP suele ser lineal o en la mayoría de los casos sigmoidal (Caicedo & López, 2009) [6].

Las interrogantes que surgieron para este tipo de arquitectura fueron: ¿Cómo entrenar un Perceptron Multicapa? ¿Cómo evaluar el error en las capas ocultas si no hay un valor deseado conocido para las salidas de estas capas? Una respuesta a estas interrogantes la planteó formalmente Werbos, cuando formuló el algoritmo de aprendizaje de *Retropropagación* (*Backpropagation*).

Como el error de la capa de salida es el único que puede calcularse de forma exacta, el algoritmo de aprendizaje de retropropagación propone propagar hacia atrás este error para estimar el error en las salidas de las neuronas de las capas ocultas, con el fin de modificar los pesos sinápticos (parámetros) de estas neuronas.
Antes de profundizar matemáticamente en el algoritmo de aprendizaje, se indicará su funcionamiento de manera conceptual:

a. Se selecciona un conjunto de patrones (vector de salidas deseadas y vector de entradas) claramente representativos del problema a solucionar, con los cuales se va a entrenar a la red. Esta fase es fundamental pues dependiendo de la calidad de los datos utilizados para el entrenamiento, será la calidad de aprendizaje de la red.

b. Se aplica el vector de entrada y se calculan las salidas de las neuronas ocultas, luego se propagan estos valores hasta calcular las salidas finales de la red.

c. Se calcula el error entre los valores deseados y las salidas de la red.

d. Se propaga el error hacia atrás, es decir, se estima el error en la capa oculta con base del error de la capa de salida.

e. Se modifican los pesos de la capa de salida y de las capas ocultas con base a una estimación de cambio de los pesos Δw en cada una de las capas que a su vez, depende del cálculo del error de la capa de salida y de la estimación del error en las capas ocultas realizado en los pasos anteriores.

f. Se verifica la condición de parada del algoritmo ya sea porque el error calculado en la salida es inferior al impuesto por el problema bajo análisis o por ejemplo, porque se ha superado un número determinado de iteraciones, en cuyo caso se considera que es necesario hacer ajustes al diseño de la red, pues la solución no converge o simplemente que se debe ampliar el número máximo de iteraciones a realizar.

g. En caso de no cumplir ninguna de las condiciones de parada, se vuelve a presentar a la red un patrón de entrenamiento.

1.4.2.1 Nomenclatura del Algoritmo de Retropropagación

Antes de formular matemáticamente el algoritmo, con la ayuda de la Figura 1.11, se definirá la notación que se seguirá a los largo de este apartado. Los subíndices i, j y k identifican todos los posibles nodos de las diferentes capas de la red.

Tabla 1.1. Nomenclatura empleada en el algoritmo de retropropagación

$\boldsymbol{x_p}$	Patrón o vector de entrada.
x_{pi}	Entrada i-ésima del vector de entrada $\boldsymbol{x_p}$.
N	Dimensión del vector de entrada.
P	Número de ejemplos, vectores de entrada y salidas diferente.
L	Número de neuronas de la capa oculta h.
M	Número de neuronas de la capa de salida o, dimensión del vector de salida.
w_{ji}^{h}	Peso de interconexión entre la i-ésima neurona de la entrada y la j-ésima de la capa oculta.
θ_j^{h}	Término de tendencia de la j-ésima neurona de la capa oculta.
$Neta_{pj}^{h}$	Entrada neta de la j-ésima neurona de la capa oculta.
y_{pj}^{h}	Salida de la j-ésima neurona de la capa oculta.
f_j^{h}	Función de activación de la j-ésima unidad oculta.
w_{kj}^{o}	Peso de interconexión entre la j-ésima neurona de la capa oculta y la k-ésima de la capa de salida o.

Tabla 1.1. Nomenclatura empleada en el algoritmo de retropropagación (continuación…)

θ_k^o	Término de tendencia de la k-ésima neurona de la capa de salida.
$Neta_{pk}^o$	Entrada neta de la k-ésima neurona de la capa de salida.
y_{pk}	Salida de la k-ésima unidad de salida.
f_k^o	Función de activación de la k-ésima unidad de salida.
d_{pk}	Valor de salida deseado para la k-ésima neurona de la capa de salida.
e_p	Valor del error para el p-ésimo patrón de aprendizaje.
η	Taza o velocidad de aprendizaje.
δ_{pk}^o	Término de error para la k-ésima neurona de la capa de salida o.
δ_{pj}^h	Término de error para la j-ésima neurona de la capa oculta h.
$f_j'^h$	Derivada de la función de activación de la j-ésima neurona de la capa oculta.
$f_k'^o$	Derivada de la función de activación de la k-ésima neurona de la capa de salida.

Modificado de Caicedo & López (2009)

1.4.2.2 Algoritmo de Retropropagación: Regla Delta Generalizada

El objetivo buscado con el aprendizaje en las redes neuronales multicapa, es el de poder establecer una transformación matemática que relacione adecuadamente los pares ordenados de ejemplos de entradas de excitación a la red y su correspondiente salida deseada. La calidad de la estimación va a depender, fundamentalmente, del número de ejemplos disponibles del problema bajo observación y que la muestra sea lo suficientemente representativa. A continuación se va a describir las etapas de este algoritmo.

Procesamiento de datos hacia adelante "feedforward"

Como primer paso se estimula la red neuronal con el vector de entrada:

$$\boldsymbol{x_p} = \begin{bmatrix} x_{p1} & x_{p2} & \dots & x_{pi} & \dots & x_{pN} \end{bmatrix}^T \tag{1.38}$$

Se calcula la entrada neta de la j-ésima neurona de la capa oculta:

$$Neta_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h}\, x_{pi} + \theta_{j}^{h} \tag{1.39}$$

Se calcula la salida de la j-ésima neurona usando la función de activación y la entrada neta:

$$y_{pj}^{h} = f_{j}^{h}\left(Neta_{pj}^{h}\right) \tag{1.40}$$

Una vez calculadas las salidas de las neuronas de la capa oculta, éstas se convierten en las señales de excitación de las neuronas de la capa de salida y así se puede calcular la entrada neta de la k-ésima neurona de la capa de salida:

$$Neta_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o}\, y_{pj}^{h} + \theta_{k}^{o} \tag{1.41}$$

Con base en la función de activación de la k-ésima neurona de la capa de salida se puede calcular la salida estimada por la red neuronal ante el estímulo de entrada:

$$y_{pk} = f_{k}^{o}\left(Neta_{pk}^{o}\right) \tag{1.42}$$

Error en las capas oculta y de salida

Antes de continuar con el cálculo del error en las capas oculta y de salida, se debe recordar como modifica la Regla Delta o LMS (*Least Mean Square*) los pesos sinápticos de una red neuronal con base a la ecuación 1.43:

$$\begin{gathered} w_i(t+1) = w_i(t) + 2\eta e_p x_i \\ e_p = d - y \end{gathered} \tag{1.43}$$

En la Regla Delta, e_p se define como el error generado por una única neurona o elemento de procesamiento. Con el Algoritmo de Retropropagación se calcula el

error global, considerando todas las unidades de procesamiento y sumando el aporte al error de cada una de las neuronas,

$$E_p = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{M} e_{pk}^2 \tag{1.44}$$

El aporte unitario al error global e_{pk} se define como el error de la $k\text{-}\acute{e}sima$ neurona y se calcula con la ecuación 1.45, donde d_{pk} representa la salida deseada.

$$e_{pk} = d_{pk} - y_{pk} \tag{1.45}$$

La búsqueda del mínimo en la superficie de error se fundamenta en el cálculo de su *gradiente descendente* ∇E_p. Para efectos de la deducción de este algoritmo se va a considerar que el análisis se lo realiza considerando el $p\text{-}\acute{e}simo$ patrón de aprendizaje $\{\boldsymbol{x}_p, \boldsymbol{d}_p\}$, por lo que en la ecuación 1.44, se eliminan los términos correspondientes a la sumatoria desde $p = 1$ hasta P.

En el primer paso se va a calcular la derivada del error global respecto del peso w_{kj}^o. Para efectos de este procedimiento, se sustituye en la ecuación 1.44 el valor de e_{pk} que se obtuvo en la ecuación 1.45 y se procede al cálculo de esta derivada.

$$E_p = \frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2$$

$$\frac{\partial E_p}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o}\left[\frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2\right]$$

$$\frac{\partial E_p}{\partial w_{kj}^o} = \frac{\partial}{\partial w_{kj}^o}\left[\frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - f_k{}^o\left(Neta_{pk}^o\right)\right)^2\right]$$

$$\frac{\partial E_p}{\partial w_{kj}^o} = -\left(d_{pk} - f_k{}^o\left(Neta_{pk}^o\right)\right).f_k{}^{o\prime}\left(Neta_{pk}^o\right).\frac{\partial Neta_{pk}^o}{\partial w_{kj}^o}$$

donde $f_k{}^{o\prime}\left(Neta_{pk}^o\right) = \frac{\partial f_k{}^o\left(Neta_{pk}^o\right)}{\partial w_{kj}^o}$

Ahora se calcula la derivada de la entrada neta, usando la ecuación 1.41.

$$\frac{\partial Neta_{pk}^{o}}{\partial w_{kj}^{o}} = \frac{\partial}{\partial w_{kj}^{o}}\left[\sum_{j=1}^{L} w_{kj}^{o}\, y_{pj}^{h} + \theta_{k}^{o}\right] = y_{pj}^{h}$$

Con base a este resultado, se puede calcular el gradiente que corresponde a la derivada del error global respecto al peso w_{kj}^{o}

$$\frac{\partial E_p}{\partial w_{kj}^{o}} = -\left(d_{pk} - y_{pk}^{o}\right).{f_k^{o}}'\left(Neta_{pk}^{o}\right).y_{pj}^{h}$$

La intención de aplicar el concepto del gradiente descendente es buscar paulatinamente un punto de error mínimo siempre guiados por el valor negativo de la derivada del error global, por lo que la expresión que se utilizará corresponde al negativo del gradiente del error $\left(-\nabla E_p\right)$.

$$-\nabla E_p = -\frac{\partial E_p}{\partial w_{kj}^{o}} = \left(d_{pk} - y_{pk}^{o}\right).{f_k^{o}}'\left(Neta_{pk}^{o}\right).y_{pj}^{h} \qquad (1.46)$$

El proceso de entrenamiento de la red busca como objetivo fundamental modificar el peso w_{kj}^{o}, esta modificación se realiza con base en la ecuación 1.47.

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \Delta w_{kj}^{o}(t) = w_{kj}^{o}(t) + \eta\left(-\nabla E_p\right) \qquad (1.47)$$

Se sustituye el valor del gradiente en esta ecuación y se obtiene la ecuación 1.48 para la modificación de los pesos,

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\left(d_{pk} - y_{pk}^{o}\right).{f_k^{o}}'\left(Neta_{pk}^{o}\right).y_{pj}^{h} \qquad (1.48)$$

De esta expresión se infiere que la función de activación f_k^{o} debe ser derivable, por consiguiente, tanto en las capas ocultas como de salida, generalmente se escoge una función de activación lineal o sigmoidal. En el primer caso, la derivada es uno y en el segundo caso, el resultado se muestra a continuación.

$$f_k^{\,o}\left(Neta_{pk}^{o}\right) = Neta_{pk}^{o} \quad \rightarrow \quad f_k^{\,o'}\left(Neta_{pk}^{o}\right) = 1$$

$$f_k^{\,o}\left(Neta_{pk}^{o}\right) = \frac{1}{1 + e^{-Neta_{pk}^{o}}} \rightarrow$$

$$f_k^{\,o'}\left(Neta_{pk}^{o}\right) = f_k^{\,o}(1 - f_k^{\,o}) = y_{kp}^{o}\left(1 - y_{pk}^{o}\right)$$

Luego de calcular las derivadas dependiendo del tipo de función de activación que se escoja, la ecuación 1.48 se cambia por la 1.49, si la función de activación es lineal,

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\left(d_{pk} - y_{pk}^{o}\right).y_{pj}^{h} \tag{1.49}$$

Para el caso de una función de activación sigmoidal, resulta la ecuación 1.50

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta\left(d_{pk} - y_{pk}^{o}\right).y_{kp}^{o}\left(1 - y_{pk}^{o}\right).y_{pj}^{h} \tag{1.50}$$

Con el fin de simplificar las expresiones 1.49 y 1.50 en una única ecuación para facilitar su representación en el algoritmo, se define el término del error en las neuronas de la capa de salida, con base en la ecuación 1.51.

$$\delta_{pk}^{o} = \left(d_{pk} - y_{pk}^{o}\right).f_k^{\,o'}\left(Neta_{pk}^{o}\right) = e_{pk}\,.f_k^{\,o'}\left(Neta_{pk}^{o}\right) \tag{1.51}$$

Por lo que la ecuación de modificación de pesos se la unifica en la expresión 1.52.

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta.\delta_{pk}^{o}\,.y_{pj}^{h} \tag{1.52}$$

1.4.2.3 Actualización de Pesos de las Capas Ocultas

Para la capa oculta no es posible calcular el error directamente, ya que no se conocen las salidas deseadas de esta capa. Sin embrago, si se usa el concepto de retropropagación, se observa que existe una manera de estimar este error de la capa oculta partiendo del error en la capa de salida (Caicedo & López, 2009) [6]. A continuación se detalla el proceso de actualización de los pesos de la capa oculta y, en particular, el peso w_{ji}^{h}. Se retoma en la ecuación 1.53, el error global

en la capa de salida para el patrón $p\text{- }ésimo$.

$$E_p = \frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2 \tag{1.53}$$

Ahora se puede calcular la gradiente descendente $-\nabla E_p$ con respecto a w_{ji}^{h}.

$$\frac{\partial E_p}{\partial w_{ji}^{h}} = \frac{\partial}{\partial w_{ji}^{h}}\left[\frac{1}{2}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2\right] \tag{1.54}$$

$$\frac{\partial E_p}{\partial w_{ji}^{h}} = -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right).\frac{\partial y_{pk}^{h}}{\partial w_{ji}^{h}} \tag{1.55}$$

Aplicando la regla de la cadena sucesivamente para el cálculo de la derivada interna se obtiene:

$$\frac{\partial E_p}{\partial w_{ji}^{h}} = -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right).\frac{\partial y_{pk}^{h}}{\partial Neta_{pk}^{o}}.\frac{\partial Neta_{pk}^{o}}{\partial y_{pj}^{h}}.\frac{\partial y_{pj}^{h}}{\partial Neta_{pj}^{h}}.\frac{\partial Neta_{pj}^{h}}{\partial w_{ji}^{h}} \tag{1.56}$$

$$\frac{\partial E_p}{\partial w_{ji}^{h}} = -\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right).f_k^{o}\left(Neta_{pk}^{o}\right).w_{kj}^{o}.f_j^{h'}.x_{pi} \tag{1.57}$$

En la ecuación 1.57 se calculó el gradiente del error global respecto de los pesos de la capa oculta y con base en este valor, se define la expresión para la modificación de los pesos de la capa oculta como:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \Delta w_{ji}^{h}(t) = w_{ji}^{h}(t) + \eta\left(-\frac{\partial E_p}{\partial w_{ji}^{h}}\right) \tag{1.58}$$

Al reemplazar en esta ecuación la expresión del gradiente, se obtiene la ecuación 1.59 que permite actualizar los pesos de la capa oculta.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\left(\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right).f_k^{o}\left(Neta_{pk}^{o}\right).w_{kj}^{o}.f_j^{h'}.x_{pi}\right) \tag{1.59}$$

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta.f_j^{h'}.x_{pi}\left(\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right).f_k^{o}\left(Neta_{pk}^{o}\right).w_{kj}^{o}\right) \tag{1.60}$$

De manera similar a lo planteado en la capa de salida, se define en la ecuación 1.60 el término del error en la capa oculta como:

$$\delta_{pj}^{h} = f_j^{h'}\left(Neta_{pj}^{h}\right)\sum_{k=1}^{M}\delta_{pk}^{o}\,.\,w_{kj}^{o} \qquad (1.61)$$

En la ecuación 1.61 se observa que el término de la sumatoria, representa matemáticamente el concepto de Retropropagación, ya que el error de la capa oculta está dado en función de los pesos sinápticos y de los términos de error de la capa de salida. Finalmente, la modificación de los pesos sinápticos de la capa oculta se la realiza con base en la ecuación 1.62.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta\,.\,\delta_{pj}^{h}.\,x_{pi} \qquad (1.62)$$

1.4.2.4 Pasos del algoritmo de retropropagación

A continuación se resumen todos los pasos que intervienen en el algoritmo de retropropagación para una red MLP.

a. Inicializar los pesos del MLP.

b. Mientras la condición de parada sea falsa se ejecutan los pasos 3 al 12.

c. Se aplica un vector de entrada $x_p = \left[x_{p1} \quad x_{p2} \quad \cdots \quad x_{pi} \quad \cdots \quad x_{pN}\right]^T$.

d. Se calculan los valores de las entradas netas para la capa oculta.

$$Neta_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h}\,x_{pi} + \theta_j^{\,h}$$

e. Se calcula la salida de la capa oculta $y_{pj}^{h} = f_j^{\,h}\left(Neta_{pj}^{h}\right)$

f. Se calculan los valores netos de entrada para la capa de salida.

$$Neta_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o}\,y_{pj}^{h} + \theta_k^{o}$$

g. Se calculan las salidas de la red neuronal $y_{pk} = f_k^{\,o}\left(Neta_{pk}^{o}\right)$

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

$$\delta_{pk}^{o} = \left(d_{pk} - y_{pk}^{o}\right).{f_k}^{o'}\left(Neta_{pk}^{o}\right)$$

i. Se estiman los términos de error para las unidades ocultas.

$$\delta_{pj}^{h} = f_j^{h'}\left(Neta_{pj}^{h}\right)\sum_{k=1}^{M}\delta_{pk}^{o}.w_{kj}^{o}$$

j. Se actualizan los pesos en la capa de salida.

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta.\delta_{pk}^{o}.y_{pj}^{h}$$

k. Se actualizan los pesos en la capa oculta.

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta.\delta_{pj}^{h}.x_{pi}$$

l. Se verifica si el error global cumple con la condición de finalizar.

$$E_p = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{M}\left(d_{pk} - y_{pk}\right)^2$$

1.4.2.5 Algoritmo Gradiente Descendente con Velocidad de Aprendizaje η Variable

Uno de los inconvenientes que presenta el algoritmo básico de gradiente descendente con Retropropagación es tener el parámetro de aprendizaje η fijo. Este parámetro cuyo valor depende de la aplicación que se esté solucionando, también puede variar en el proceso de aprendizaje con el fin de modificar el tamaño de la variación de los pesos Δw_i, para acelerar la convergencia del algoritmo de aprendizaje (Caicedo & López, 2009) [6]. Una estrategia efectiva para variar el parámetro de aprendizaje es el incremento o disminución en cada iteración, dependiendo de la manera cómo evolucione el error de entrenamiento, con base en la regla descrita en la ecuación 1.63.

$$\eta_{k+1} = \begin{cases} \rho\,\eta_k, & si\ E_p(w_{k+1}) < E(w_k) \\ \\ \sigma\,\eta_k, & si\ E_p(w_{k+1}) \geq E(w_k) \end{cases} \qquad (1.63)$$

donde, $\rho > 1$ y $0 < \sigma < 1$. Los parámetros ρ, σ y η_o son inicializados de manera

heurística. Generalmente ρ es definido con un valor cercano a uno (por ejemplo $\rho = 1.1$), para evitar incrementos exagerados en el error de entrenamiento y σ con un valor tal que reduzca η rápidamente para de esta manera abandonar valores elevados de la razón de aprendizaje (por ejemplo $\sigma = 0.5$).

Pasos del algoritmo gradiente descendente con η variable

a. Definir los pesos iniciales. *Condición_parar* = Falsa

b. Mientras *Condición_parar* = Falsa ejecutar los pasos c al l.

c. Se aplica un vector de entrada $x_p = \begin{bmatrix} x_{p1} & x_{p2} & \cdots & x_{pi} & \cdots & x_{pN} \end{bmatrix}^T$.

d. Se calculan los valores de las entradas netas para la capa oculta.

$$Neta_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h}\, x_{pi} + \theta_{j}^{h}$$

e. Se calcula la salida de la capa oculta $y_{pj}^{h} = f_{j}^{h}\left(Neta_{pj}^{h}\right)$

f. Se calculan los valores netos de entrada para la capa de salida.

$$Neta_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o}\, y_{pj}^{h} + \theta_{k}^{o}$$

g. Se calculan las salidas de la red neuronal $y_{pk} = f_{k}^{o}\left(Neta_{pk}^{o}\right)$

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

$$\delta_{pk}^{o} = \left(d_{pk} - y_{pk}^{o}\right).\, f_{k}^{o'}\left(Neta_{pk}^{o}\right)$$

i. Se estiman los términos de error para las unidades ocultas.

$$\delta_{pj}^{h} = f_{j}^{h'}\left(Neta_{pj}^{h}\right) \sum_{k=1}^{M} \delta_{pk}^{o}\,.\, w_{kj}^{o}$$

j. Se calcula el error global de salida.

$$E_p = \frac{1}{2P} \sum_{p=1}^{P} \sum_{k=1}^{M} \left(d_{pk} - y_{pk}\right)^2$$

k. Se hace que $E_{prev} = E_p$

l. Si $E_{prev} < E_{min}$ entonces

Condición_parar = Verdadera y salir

caso contrario

Condición_parar = Falsa

m. Se actualizan los pesos en la capa de salida. $w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta . \delta_{pk}^{o} . y_{pj}^{h}$

n. Se actualizan los pesos en la capa oculta. $w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta . \delta_{pj}^{h} . x_{pi}$

o. Se calcula el error E_p.

p. Si $E_p < E_{prev}$ entonces

$E_{prev} = E_p$

$\eta_{k+1} = \rho\, \eta_k$, donde $\rho > 1$

retornar al paso c

caso contrario

$\eta_{k+1} = \sigma\, \eta_k$, donde $0 < \sigma < 1$

retornar al paso m

1.4.2.6 Aprendizaje con Término de Momento

El algoritmo de Retropropagación encuentra un valor mínimo de error (local o global) navegando con el gradiente descendente por la superficie de error. La velocidad de convergencia se controla con el valor del parámetro de aprendizaje η, normalmente dicho parámetro debe ser una número pequeño, en el rango de 0.05 a 0.25. Si se selecciona un η pequeño, entonces el ajuste a los pesos sinápticos que se da de iteración a iteración será pequeño pero con aprendizaje seguro, además puede presentarse un alto número de iteraciones y tardar mucho el tiempo de ejecución de este algoritmo (aprendizaje lento). En contraposición, si η se selecciona grande, entonces el aprendizaje será rápido, pero existe la

posibilidad de que la red se vuelva inestable (se presentan oscilaciones).

Una forma sencilla de incrementar el parámetro de la velocidad de aprendizaje (acelerar la convergencia) de modo que se evite la posibilidad de inestabilidad consiste en modificar la regla delta mostrada en las ecuaciones 1.52 y 1.62, incluyendo un término de momento en la ecuación de la variación del peso (Sánchez & Alanis, 2006) [7]. Para los pesos de la capa de salida se tiene las siguientes expresiones:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \Delta w_{kj}^{o}(t) \tag{1.64}$$

$$\Delta w_{kj}^{o}(t) = \eta . \delta_{pk}^{o} . y_{pj}^{h} + \alpha \Delta w_{kj}^{o}(t-1) \tag{1.65}$$

donde, α es la *constante de momento*, usualmente un número positivo y menor que la unidad $(0 \leq \alpha < 1)$. Si se considera que $\Delta w_{kj}^{o}(t-1) = w_{kj}^{o}(t) - w_{kj}^{o}(t-1)$ y se reemplaza esta expresión en la ecuación 1.65, entonces la ecuación 1.64 queda modificada como:

$$w_{kj}^{o}(t+1) = w_{kj}^{o}(t) + \eta . \delta_{pk}^{o} . y_{pj}^{h} + \alpha\left[w_{kj}^{o}(t) - w_{kj}^{o}(t-1)\right] \tag{1.66}$$

Como se observa en la ecuación 1.66, además de utilizar el valor de los pesos en el instante t, se utiliza una fracción del valor de los pesos del instante anterior $t-1$, que se controlará con el valor de α. Con este procedimiento se desea emular al *momento físico* considerando la tendencia en el aprendizaje que traía la red en el instante anterior (Caicedo & López, 2009) [6].

Siguiendo el mismo procedimiento explicado anteriormente y tomando en consideración la ecuación 1.62, para la capa oculta se tiene la siguiente expresión incluyendo la constante de momento:

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \eta . \delta_{pj}^{h} . x_{pi} + \alpha\left[w_{ji}^{h}(t) - w_{ji}^{h}(t-1)\right] \tag{1.67}$$

1.4.3 REGLA DE APRENDIZAJE DE RETROPROPAGACIÓN PARA UNA RED ADAPTATIVA [4]

En esta sección se presenta una regla de aprendizaje básica para redes adaptativas anticipativas (*feedforward adaptive netwroks*), cuyo funcionamiento se fundamenta en el método de máxima pendiente (gradiente descendente) discutido en la sección 1.4.2. Casi todas las consideraciones de aprendizaje explicadas en la sección anterior para las redes neuronales MPL son válidas para las redes adaptativas, salvo el caso de ciertos aspectos que serán aclarados.

La parte central de una regla de aprendizaje para una red adaptativa consiste en obtener de forma recursiva un vector gradiente en el que cada elemento se define como la derivada de una medida del error con respecto a un parámetro. Esto se hace por medio de la regla de la cadena y el método se conoce generalmente como la *regla de aprendizaje de retropropagación* (*backpropagation learning rule*) porque el vector gradiente se calcula en la dirección opuesta al flujo de la salida de cada nodo. Una vez que se obtiene el vector gradiente, se utilizan técnicas de optimización y regresión basadas en derivadas para ajustar los parámetros (Jang, Sun & Mizutani, 1997) [2].

Si se considera una red adaptiva de avance que tiene L capas y la k-*ésima* capa tiene N nodos, entonces la salida y la función del i-*ésimo* nodo $(i = 1, 2, \dots, N)$ en la k-*ésima* capa se pueden representar como O_i^k y f_i^k respectivamente, como se observa en el ejemplo de la Figura 1.12. Puesto que la salida de un nodo depende de las señales que entran a él y de su conjunto de parámetros, se tiene la siguiente expresión general para la función de nodo:

$$O_i^k = f_i^k\left(O_1^{k-1}, O_2^{k-1}, \dots, O_N^{k-1}, \alpha, \beta, \gamma, \dots\right) \tag{1.68}$$

donde α, β, γ, etc son los parámetros pertenecientes a este nodo.

[4] La mayoria de temas y ecuaciones presentadas en este apratado se basan an las publicaciones realizadas por Jang (1993), Jang & Sun (1995) y Jang, Sun & Mizutani (1997).

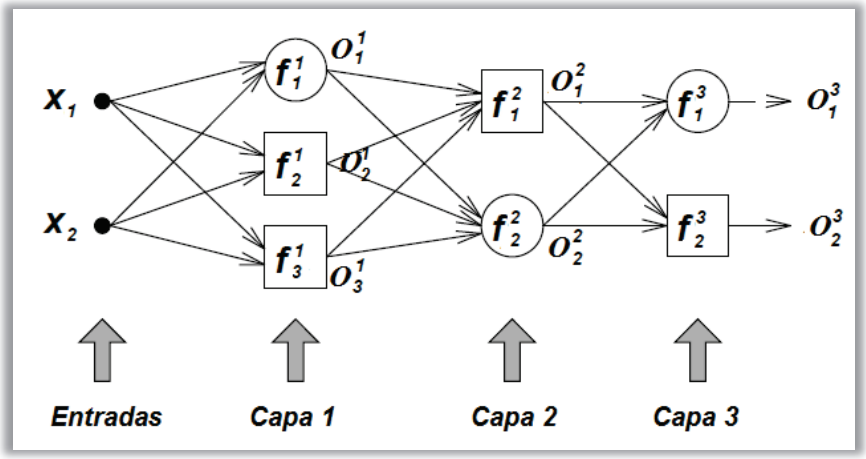


Figura 1.12. Representación de las capas de una red neuronal adaptativa [3]

Asumiendo que el conjunto de datos de entrenamiento dado tiene P entradas, se puede definir una *medida del error* para el p*-ésimo* $(1 \leq p \leq P)$ patrón de aprendizaje de la última capa de salida L (con $1 \leq m \leq M$ nodos), como la suma de los errores cuadráticos:

$$E_p = \frac{1}{2}\sum_{m=1}^{M}\left(d_{m,p} - O_{m,p}^{L}\right)^2 \tag{1.69}$$

donde, $d_{m,p}$ es el m*-ésimo* componente del p*-ésimo* vector de salida deseado y $O_{m,p}^{L}$ es el m*-ésimo* componente del vector de salida correspondiente a la última capa L, producido al presentar el p*-ésimo* vector de entrada a la red (Bravo & García, 2002) [4]. Así, la tarea es minimizar la medida de error global, la cual se define como

$$E = \sum_{p=1}^{P} E_p \tag{1.70}$$

Recordar que la definición de E_p en la ecuación 1.69 no es universal; existen otras definiciones de E_p que son posibles emplear para situaciones o aplicaciones específicas. Por lo tanto, se va evitar el uso de una expresión explícita para la medida de error E_p y así formular expresiones generales. Además, se asume que

E_p sólo depende de los nodos salida; aquellas situaciones más generales se discutirán más adelante.

Para emplear el método gradiente descendente en la minimización de la medida del error, primero debe obtenerse el vector gradiente. Antes de hallar dicho vector primero debería explicarse la siguiente relación causal.

Figura 1.13. Relación causal de las variables en una red adaptativa

En otras palabras, un cambio pequeño en un parámetro α afectará la salida del nodo que contiene al propio α; a su vez, éste afectará la salida de la capa final de la red y, por consiguiente, a la medida del error. Por lo tanto, el concepto básico al calcular el vector gradiente es propagar un tipo de información derivada, iniciando desde la capa de salida y retrocediendo capa por capa, hasta alcanzar la capa de entrada (Bravo & García, 2002) [4].

Con el fin de desarrollar un procedimiento de aprendizaje que implemente el gradiente descendente en E sobre el espacio de parámetros, primero se tiene que calcular la velocidad de error $\partial E_p/\partial O$ para el p-*ésimo* patrón de entrenamiento y para cada nodo de salida de O. La velocidad de error para el i-*ésimo* nodo de salida de la capa L, puede calcularse fácilmente a partir de la ecuación 1.69:

$$\frac{\partial E_p}{\partial O_{i,p}^L} = \frac{\partial}{\partial O_{i,p}^L}\left[\frac{1}{2}\sum_{i=1}^{M}\left(d_{i,p} - O_{i,p}^L\right)^2\right] = -\left(d_{i,p} - O_{i,p}^L\right) \qquad (1.71)$$

Esta expresión fue denomina la *derivada ordenada* por Werbos. Para facilitar la discusión, se define un término de error ϵ_i^L como $\epsilon_i^L = \partial E_p/\partial O_{i,p}^L$. La diferencia entre la derivada ordenada y la derivada parcial ordinaria radica en la manera de

diferenciar la función. Para un nodo interno de salida (es decir, la k-*ésima* capa tiene que ser diferente de L, $k \neq L$), la derivada parcial de $\partial E_p / \partial O_{i,p}^k$ es igual a cero, ya que E_p no depende de $O_{i,p}^k$ directamente. Sin embargo, es obvio que E_p depende de $O_{i,p}^k$ indirectamente, ya que un cambio en $O_{i,p}^k$ propagará información a la capa de salida a través de rutas indirectas y así producirá un cambio correspondiente en el valor de E_p. Por lo tanto, el término de error ϵ_i^L puede ser visto como la relación de estos dos cambios cuando se hacen infinitesimales.
Para el i-*ésimo* nodo interno de la k-*ésima* capa de la red, el término de error ϵ_i^L puede ser derivado mediante la regla de la cadena:

$$\epsilon_i^k = \frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \cdot \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k} \tag{1.72}$$

donde, $1 \leq k \leq (L-1)$ y $\#(k+1)$ representa el número de nodos de la capa $k+1$ de la red. Esto es, el término de error de un nodo interno en la capa k puede ser expresado como una combinación lineal del término de error presente en los nodos de la capa $k+1$. Por lo tanto, para todo $1 \leq k \leq L$ y $1 \leq i \leq N$, es posible encontrar $\epsilon_i^k = \partial E_p / \partial O_{i,p}^k$ aplicando inicialmente la ecuación 1.71 una vez para obtener los términos de error en la capa de salida L y luego aplicando la ecuación 1.72 iterativamente, hasta alcanzar la capa k deseada. Este proceso es conocido como el *aprendizaje de retropropagación*, puesto que los términos de error se obtienen secuencialmente partiendo desde de la capa de salida hacia la de entrada (Bravo & García, 2002) [4].

El vector gradiente se define como la derivada de la medida de error con respecto a cada parámetro, así se tiene que aplicar la regla de la cadena de nuevo para encontrar el vector gradiente. Ahora si α es un parámetro del i-*ésimo* nodo de la k-*ésima* capa de la red, se tiene:

$$\frac{\partial E_p}{\partial \alpha} = \frac{\partial E_p}{\partial O_{i,p}^k} \cdot \frac{\partial O_{i,p}^k}{\partial \alpha} = \epsilon_i^k \cdot \frac{\partial O_{i,p}^k}{\partial \alpha} \tag{1.73}$$

Tenga en cuenta que si se permite que el parámetro α sea compartido entre diferentes nodos, entonces la ecuación 1.73 se debe cambiar a una forma más general como:

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \cdot \frac{\partial O^*}{\partial \alpha} \tag{1.74}$$

donde, S es el conjunto de nodos que contienen a α como un parámetro. La derivada de la medida del error global E con respecto a α es:

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} = \sum_{p=1}^{P} \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \cdot \frac{\partial O^*}{\partial \alpha} \tag{1.75}$$

En consecuencia, la fórmula de actualización del parámetro genérico α para el método gradiente descendente simplemente es:

$$\Delta\alpha(t) = -\eta \frac{\partial E}{\partial \alpha}$$

$$\alpha(t+1) = \alpha(t) + \Delta\alpha(t) = \alpha(t) - \eta \frac{\partial E}{\partial \alpha} \tag{1.76}$$

donde, η es la *velocidad de aprendizaje*, la cual puede ser expresada como:

$$\eta = \frac{\rho}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha}\right)^2}} \tag{1.77}$$

en donde, ρ es el *tamaño del paso*, esto es, la dimensión de cada transición a lo largo de la dirección del gradiente en el espacio de parámetros. Generalmente se cambia el tamaño del paso para variar la velocidad de convergencia. Como se observó en las ecuaciones indicadas en esta sección, la matemática utilizada en el método de Aprendizaje de Retropropagación es muy fuerte, por lo que se procede a indicar los pasos a seguir para la implementación de esta regla:

Pasos del algoritmo de retropropagación para una red adaptativa

a. Inicializar los parámetros de los nodos $(\alpha, \beta, \gamma, \dots)$ con valores pequeños al azar.

b. Mientras la condición de parada sea falsa se ejecutan los pasos c al k.

c. Se aplica un vector de entrada $x_p = \begin{bmatrix} x_{p1} & x_{p2} & \cdots & x_{pi} & \cdots & x_{pN} \end{bmatrix}^T$.

d. Se calculan las salidas de la red neuronal.

$$O_i^L = f_i^L\left(O_1^{L-1}, O_2^{L-1}, \dots, O_{\#(L-1)}^{L-1}, \alpha, \beta, \gamma, \dots\right)$$

donde $\#(L-1)$ corresponde al número de nodos de la antepenúltima capa.

f. Se calcula los términos de error entre la salida de la red y la salida deseada

$$\epsilon_i^L = \frac{\partial E_p}{\partial O_{i,p}^L} = \frac{\partial}{\partial O_{i,p}^L}\left[\sum_{i=1}^{M}\left(d_{i,p} - O_{i,p}^L\right)^2\right] = -\left(d_{i,p} - O_{i,p}^L\right)$$

g. Se estiman los términos de error para las capas ocultas.

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

donde $1 \leq k \leq (L-1)$ y $\#(k+1)$ representa el número de nodos de la capa $k+1$ de la red.

h. Se calcula el vector gradiente de las capas ocultas.

$$\frac{\partial E_p}{\partial \alpha} = \frac{\partial E_p}{\partial O_{i,p}^k} \cdot \frac{\partial O_{i,p}^k}{\partial \alpha} = \epsilon_i^k \cdot \frac{\partial O_{i,p}^k}{\partial \alpha}$$

i. Se acumula el valor del vector gradiente en el error total.

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} = \sum_{p=1}^{P} \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \cdot \frac{\partial O^*}{\partial \alpha}$$

donde S es el conjunto de nodos que contienen a α como un parámetro

j. Se actualizan los valores del parámetro genérico α.

$$\alpha(t+1) = \alpha(t) - \eta \frac{\partial E}{\partial \alpha} \qquad con \quad \eta = \frac{\rho}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha}\right)^2}}$$

k. Se verifica si el error global cumple con la condición de finalizar.

$$E = \sum_{p=1}^{P} E_p = \frac{1}{2} \sum_{p=1}^{P} \sum_{m=1}^{M} \left(d_{m,p} - O_{m,p}^{L}\right)^2$$

Existen dos paradigmas de aprendizaje para satisfacer las necesidades ante diferentes aplicaciones. Uno es el *aprendizaje fuera de línea* (o *aprendizaje por lote*), en el cual la fórmula de actualización del parámetro α se basa en la ecuación 1.75 y la acción de actualización tiene lugar solamente hasta que todo el conjunto de datos de entrenamiento se ha presentado a la red, esto es, después de cada *época* o *barrido*. Por otro lado, en el *aprendizaje en línea* (*online* o *aprendizaje patrón por patrón* o *aprendizaje incremental*), los parámetros son actualizados inmediatamente después de que cada par de datos entrada/salida (patrón de entrenamiento) se presenta a la red y la fórmula de actualización se basa en la ecuación 1.73 (Jang, Sun & Mizutani, 1997) [2].

En la práctica, es posible combinar estos dos modos de aprendizaje y actualizar el parámetro después que los p datos de entrenamiento de entrada se han presentado, donde $1 \leq p \leq P$ y algunas veces es referido como el *tamaño de época*. Estos dos tipos de paradigmas de aprendizaje se describen con mayor detalle en la sección 1.4.4, donde se introduce una regla de aprendizaje híbrida.

1.4.4 REGLA DE APRENDIZAJE HÍBRIDA: COMBINANDO LSE Y RETROPROPAGACIÓN [5]

Aunque en una red adaptiva se puede aplicar la regla de retropropagación o el gradiente descendente para identificar sus parámetros, generalmente estos métodos de optimización son lentos para converger y tienden a quedar atrapados en mínimos locales.

El método que aquí se propone para determinar los parámetros de la red, es una regla de aprendizaje híbrida, la cual combina el método del gradiente descendente (GD) con el estimador de mínimos cuadrados (LSE), descritos en el apartado 1.4.3 y 1.3.3, respectivamente. Esto parte del hecho de que la salida de una red adaptable (suponiendo que sólo hay una) es lineal en algunos de sus parámetros y éstos se pueden identificar en forma rápida con el método LSE lineal (Bravo & García, 2002) [4].

Antes de empezar con la explicación del algoritmo de aprendizaje híbrido se realizará una definición referente a una función compuesta, puesto que este tipo de función será empleada en el desarrollo matemático de los siguientes paradigmas de aprendizaje. Para la descripción se tomará como ejemplo los conjuntos mostrados en la Figura 1.14.

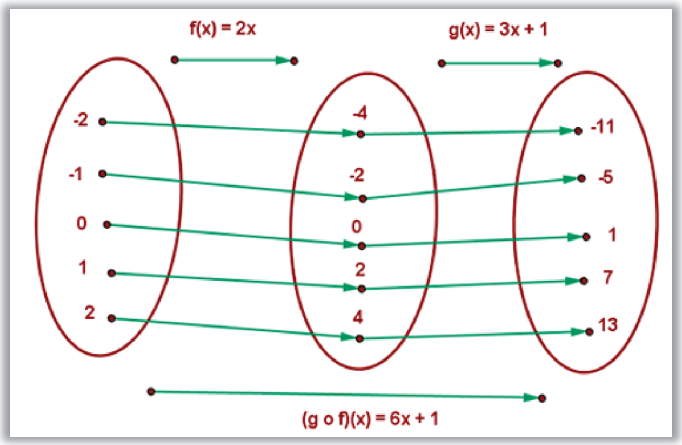


Figura 1.14. Ejemplo de una función compuesta

[5] Los temas y ecuaciones presentados en este apartado se basan en el Capítulo 8 del libro publicado por Jang, Sun & Mizutani (1997).

Función Compuesta: Si se tienen dos funciones, $f(x)$ y $g(x)$, de modo que el dominio de la segunda esté incluido en el recorrido la primera, se puede definir una nueva función que asocie a cada elemento del dominio de $f(x)$ el valor de $g[f(x)]$. Esta nueva función $g[f(x)]$ se la denomina *función compuesta* y se la representa como $(g \circ f)(x)$. Para aclarar esta explicación tómese como ejemplo las funciones $f(x) = 2x$ y $g(x) = 3x + 1$, entonces:

$$(g \circ f)(x) = g[f(x)] = g(2x) = 3(2x) + 1 = 6x + 1$$
$$Si\ x = 1\ \ entonces\ \ (g \circ f)(1) = 6(1) + 1 = 7$$

1.4.4.1 Aprendizaje Fuera de Línea (Off-Line o Aprendizaje por Lote)

Por sencillez en el análisis, se asume que la red adaptativa bajo consideración tiene sólo una salida representada por:

$$Out = F(\boldsymbol{i}, S) \qquad (1.78)$$

donde, $\boldsymbol{i}$ es el vector o conjunto de las variables de entrada, S es el conjunto de parámetros y F es la función global implementada para modelar la red adaptativa. Si existe una función H tal que la función compuesta $H \circ F$ sea lineal en algunos de los elementos de S, entonces esos elementos pueden ser identificados con el método de mínimos cuadrados. Más formalmente, si el conjunto de parámetros S se puede descomponer en dos conjuntos:

$$S = S_1 \oplus S_2 \qquad (1.79)$$

donde, $\oplus$ representa una suma directa, tal que $H \circ F$ es lineal en los elementos de S_2, entonces al aplicar H a la ecuación 1.78 se obtiene:

$$H(Out) = H \circ F(\boldsymbol{i}, S) \qquad (1.80)$$

que es lineal en los elementos de S_2. Si se tienen los valores de los elementos de S_1, se pueden insertar m datos de entrenamiento en la ecuación 1.80 y obtener una ecuación matricial de la forma:

$$\boldsymbol{A\theta} = \boldsymbol{y} \tag{1.81}$$

donde $\boldsymbol{\theta}$ es un vector desconocido cuyos elementos son los parámetros en S_2. Esta ecuación representa el problema estándar de mínimos cuadrados lineales y la mejor solución para $\boldsymbol{\theta}$ (la cual minimiza $\|\boldsymbol{A\theta} - \boldsymbol{y}\|^2$) es el estimador de mínimos cuadrados (LSE) $\widehat{\boldsymbol{\theta}}$, definido por:

$$\widehat{\boldsymbol{\theta}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{y} \tag{1.82}$$

donde, $\boldsymbol{A}^T$ es la transpuesta de $\boldsymbol{A}$ y $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ es la pseudo inversa de $\boldsymbol{A}$ siempre que $\boldsymbol{A}^T\boldsymbol{A}$ sea invertible. Por supuesto, también se puede emplear la fórmula del LSE recursivo introducido en la sección 1.3.3 para calcular $\widehat{\boldsymbol{\theta}}$. En concreto, si $\boldsymbol{a}_i^T$ es el *i-ésimo* vector fila de la matriz de $\boldsymbol{A}$ definida en la ecuación 1.81 y y_i el *i-ésimo* elemento de $\boldsymbol{y}$; entonces $\boldsymbol{\theta}$ puede ser calculado de manera iterativa de la siguiente manera:

$$\begin{aligned} \boldsymbol{P}_{i+1} &= \boldsymbol{P}_i - \frac{\boldsymbol{P}_i\boldsymbol{a}_{i+1}\boldsymbol{a}_{i+1}^T\boldsymbol{P}_i}{1 + \boldsymbol{a}_{i+1}^T\boldsymbol{P}_i\boldsymbol{a}_{i+1}} \\ \boldsymbol{\theta}_{i+1} &= \boldsymbol{\theta}_i + \boldsymbol{P}_{i+1}\boldsymbol{a}_{i+1}(y_{i+1} - \boldsymbol{a}_{i+1}^T\boldsymbol{\theta}_i) \end{aligned} \tag{1.83}$$

donde, $1 \leq i \leq m-1$ y el estimador de mínimos cuadrados global $\widehat{\boldsymbol{\theta}}$ es igual a $\boldsymbol{\theta}_m$, que es el estimador que emplea todos los m pares de datos.

Las condiciones iniciales necesarias para determinar la ecuación 1.83 son $\boldsymbol{\theta}_o = 0$ y $\boldsymbol{P}_o = \gamma\boldsymbol{I}$ donde γ es un número positivo grande e $\boldsymbol{I}$ es la matriz identidad de dimensión $m\,x\,m$. Los efectos de estas condiciones iniciales sobre la identificación de $\widehat{\boldsymbol{\theta}}$ fueron descritos en la sección 1.3.3. Cuando se trabaja con redes adaptativas de múltiples salidas (esto es, la salida Out en la ecuación 1.78 es un vector columna), la ecuación 1.83 se sigue aplicando con la excepción de que y_i representa ahora la *i-ésima* fila de la matriz $\boldsymbol{Y}$ y se denota como $\boldsymbol{y}_i^T$.

Ahora bien, se puede combinar el método del gradiente descendente (GD) con el estimador de mínimos cuadrados (LSE) para actualizar los parámetros en una red adaptativa. Para poder aplicar el aprendizaje híbrido en una forma iterativa, cada

época de este procedimiento de aprendizaje está compuesta de un *paso en avance (forward pass)* y un *paso en retroceso (backward pass)* (Bravo & García, 2002) [4].

En el paso en avance, después de que se proporciona el vector de entrada, las señales funcionales se propagan hacia adelante para calcular las salidas de cada nodo en la red, capa por capa, hasta obtener la fila correspondiente de las matrices $\boldsymbol{A}$ e $\boldsymbol{y}$ de la ecuación 1.81. Este proceso se repite para todos los pares de datos de entrenamiento para formar la matriz completa $\boldsymbol{A}$ e $\boldsymbol{y}$; a continuación, los parámetros en S_2 se identifican mediante la fórmula de la pseudo inversa en la ecuación 1.82 o las fórmulas de los mínimos cuadrados recursivos en la ecuación 1.83. Después de que se identifican los parámetros en S_2, se puede calcular la medida del error para cada par de datos de entrenamiento (Bravo & García, 2002) [4].

En el paso en retroceso, las términos de error (la derivada de la medida del error con respecto a cada salida de nodo**,** ver las ecuaciones 1.71 y 1.72) se propagan desde la última salida hacia la última entrada; en este proceso el vector gradiente del error se acumula para cada entrada de datos de entrenamiento. Después de realizar el mismo procedimiento para todos los datos de entrenamiento, se actualizan los parámetros en S_1 mediante el método gradiente descendente de la ecuación 1.76 (Jang, Sun & Mizutani, 1997) [2].

Para los valores fijos de los parámetros dados en S_1, se puede garantizar que los parámetros así encontrados en S_2 están en el punto óptimo global en el espacio de parámetros de S_2 debido a la elección de la medida del error cuadrático. Esta regla de aprendizaje híbrida no sólo puede disminuir la dimensión del espacio de búsqueda por el método del gradiente descendente, sino que, en general, reducirá substancialmente el tiempo de convergencia (Bravo & García, 2002) [4].

1.4.4.2 Aprendizaje en Línea (On-Line o Aprendizaje Patrón por Patrón)

Si los parámetros son actualizados después de cada presentación de los datos, se tiene un esquema de *aprendizaje en línea* o *patrón por patrón*. Esta estrategia de aprendizaje es vital en la identificación de parámetros en línea para sistemas con características cambiantes. Para modificar la regla de aprendizaje *off-line*, a fin de obtener una versión en línea, es obvio que el gradiente descendente debe basarse en E_p (ecuación 1.73) en lugar de E. Estrictamente hablando, esto no es un procedimiento de búsqueda real de gradiente para minimizar E, sin embargo, se aproximará a uno si la velocidad de aprendizaje es pequeña (Bravo & García, 2002) [4].

Para que la fórmula de mínimos cuadrados recursivo considere las características variantes en el tiempo de los datos de entrada, los efectos de los pares de datos pasados deben decaer conforme los nuevos pares de datos son presentados. Un método simple para ello es formular la medida del error cuadrático como una versión ponderada que asigne mayores factores de peso a los pares de datos más recientes. Esto equivale a agregar un *factor de olvido* λ a la fórmula recursiva original (Jang, Sun & Mizutani, 1997) [2], como se muestra a continuación:

$$\boldsymbol{P}_{i+1} = \frac{1}{\lambda}\left[\boldsymbol{P}_i - \frac{\boldsymbol{P}_i\, \boldsymbol{a}_{i+1}\, \boldsymbol{a}_{i+1}^T \boldsymbol{P}_i}{\lambda + \boldsymbol{a}_{i+1}^T \boldsymbol{P}_i\, \boldsymbol{a}_{i+1}}\right] \tag{1.84}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \boldsymbol{P}_{i+1}\boldsymbol{a}_{i+1}(y_{i+1} - \boldsymbol{a}_{i+1}^T \boldsymbol{\theta}_i)$$

donde, los valores típicos de λ están entre 0.9 y 1. Entre más pequeño sea el valor de λ, se tendrá un efecto más rápido sobre el decaimiento de los datos pasados. Sin embargo, en ocasiones un λ muy pequeño puede originar inestabilidad numérica, por lo que debe evitarse.

1.4.4.3 Diferentes Modos de Combinar el Gradiente Descendente y el LSE

En este punto, debe mencionarse que la complejidad computacional del estimador de mínimos cuadrados (LSE) es generalmente mayor que la del método de gradiente descendente (GD) en el proceso de adaptación. Sin embargo, para lograr un nivel de rendimiento prescrito, el LSE es generalmente mucho más rápido. En consecuencia, en función de la disposición de los recursos computacionales y el nivel requerido de rendimiento para actualizar los parámetros de red adaptativa, se puede elegir entre al menos cuatro tipos de reglas de aprendizaje híbridas que combinan GD y LSE en diferentes grados, de la siguiente manera (Jang, Sun & Mizutani, 1997) [2]:

a. Solamente un pase del LSE: los parámetros no lineales son fijos mientras que los parámetros lineales son identificados por la aplicación de una sola vez del LSE.

b. Solamente el Gradiente Descendente: todos los parámetros son tratados como no lineales y son actualizados al aplicar el GD de forma iterativa, tanto en el paso en avance y en el de retroceso.

c. Un pase del LSE seguido de un GD iterativo: el LSE se aplica sólo una vez al comienzo para obtener los valores iniciales de los parámetros lineales, y luego el GD se hace cargo para actualizar todos los parámetros de forma iterativa.

d. Gradiente Descendente y LSE: en primer lugar se identifican los parámetros lineales y no lineales. Cada iteración (época) del Gradiente Descendente es utilizada para actualizar los parámetros no lineales, seguido por el LSE para identificar los parámetros lineales. Esta es la regla de aprendizaje híbrida propuesta en este trabajo.

La elección de uno de los métodos antes mencionados se basa en un compromiso entre la complejidad y el rendimiento computacional.

1.4.5 REDES NEURONALES PARA CONTROL DE PROCESOS [6]

Las metodologías de control tradicionales se basan principalmente en la teoría de sistemas lineales, mientras que los sistemas reales presentan características no lineales y tienen dinámicas que son difíciles o imposibles de modelar, ya sea por la presencia de ruido, incertidumbre, múltiples lazos, etc., que crean problemas a los ingenieros para tratar de diseñar algoritmos de control. Es, por tanto, un gran desafío para los ingenieros diseñar un algoritmo de control eficiente. Desde el punto de vista del diseño, las especificaciones para los algoritmos de control deben ser lo suficientemente simples para ser implementadas y comprendidas con propiedades tales como la capacidad de aprendizaje, la robustez y la no linealidad. Una de las razones por las que las redes neuronales se han vuelto muy populares en aplicaciones de control es que cumplen algunos de estos criterios para el diseño e implementación. Desde un punto de vista práctico, el paralelismo inherente y la capacidad de adaptación rápida de las redes neuronales son ventajas adicionales.

La potencia de los algoritmos de aprendizaje, la variedad de arquitecturas y la capacidad de entrenamiento a partir de las funciones de entrada/salida y/o datos experimentales hacen de las redes neuronales artificiales la tecnología preferida para muchas aplicaciones. Las redes neuronales ofrecen soluciones simples a problemas complejos de control. El éxito del algoritmo de retropropagación para entrenar redes con múltiples capas ha contribuido en la aplicación de las redes neuronales para fines de control. El uso de las redes neuronales en aplicaciones de control, incluyendo el control de procesos, robótica, aplicaciones de fabricación, entre otros, ha experimentado recientemente un rápido crecimiento.

El objetivo básico del control neuronal es proporcionar la señal de entrada apropiada a un sistema físico dado (proceso o planta) para producir una respuesta deseada. Normalmente hay dos pasos a seguir cuando se utilizan redes neuronales para el control: *la identificación del sistema* (planta o proceso) y *el diseño de control*.

---

[6] Los temas presentados en este apartado se basan en el Capítulo 5 del libro publicado por Siddique & Adeli (2013).

1.4.5.1 Identificación del Sistema

El concepto clave de la identificación es el proceso de determinar un modelo dinámico para un sistema desconocido. El modelo identificado se puede utilizar posteriormente para fines de control. La identificación del sistema consta de dos pasos principales: el primer paso es elegir un modelo paramétrico apropiado y el segundo paso es ajustar los parámetros del modelo de acuerdo con algunas leyes de adaptación de modo que la respuesta del modelo ante una señal de entrada pueda aproximar la respuesta del sistema real.

El problema de la identificación de la estructura del modelo y la estimación de sus parámetros puede formularse como un problema de aprendizaje y un mapeo entre los espacios de entrada/salida conocidos. Las redes neuronales multicapas tienen buenas capacidades de aproximación, las cuales proporcionan una herramienta poderosa para la identificación de sistemas desconocidos con no linealidades. La red neuronal se compone de señales retardadas de las entradas y salidas, con un suficiente número de capas y neuronas, que son capaces de igualar el comportamiento de entrada/salida del correspondiente mapeo no lineal de la planta. Esto implica que la función no lineal de la planta se sustituye por una red neuronal con matrices de peso fijo, pero desconocidos, la cual aprende utilizando un algoritmo de aprendizaje adecuado y un conjunto de datos disponibles.

En la fase de identificación del sistema, se desarrolla un modelo de red neural de la planta a ser controlada. El identificador se compone de una red neural multicapa incorporada en paralelo con un sistema dinámico, donde la estructura proviene de la función de error estándar en la literatura de identificación y control del sistema. La información estructural está en realidad contenida en la conectividad y los pesos de la red neuronal. La identificación del sistema puede llevarse a cabo de dos maneras: *identificación del modelo hacia adelante* e *identificación del modelo directo inverso*.

La primera etapa de la identificación del modelo hacia delante (o simplemente identificación directa) es entrenar una red neuronal para representar la dinámica

hacia adelante de la planta. El error entre la salida de la planta y_p y la salida del modelo de red neuronal y_m se utiliza como la señal de entrenamiento de la red neural. La configuración básica para la identificación del modelo hacia delante se muestra en la Figura 1.15 (a). El modelo estimado mediante la red neuronal utiliza las entradas y salidas actuales y anteriores de la planta real (que se muestran como señales retardadas en la figura) para predecir los valores futuros de la salida de la planta. La red puede ser entrenada fuera de línea (*offline)* en modo por lotes, utilizando datos recogidos de la operación de la planta. Cualquiera de los algoritmos de entrenamiento de retropropagación discutidos en los apartados 1.4.2, 1.4.3 y 1.4.4 puede ser utilizado para el entrenamiento de la red.

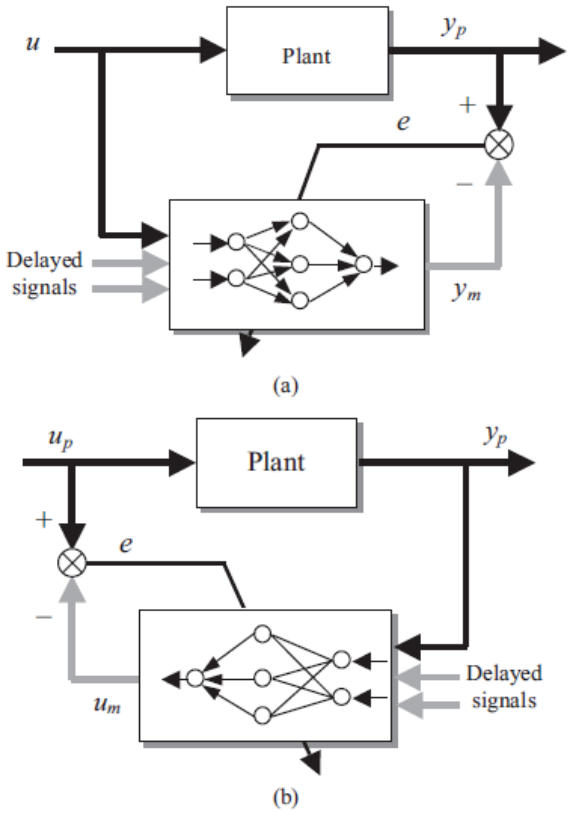


Figura 1.15. Identificación de la planta utilizando redes neuronales. (a) Identificación hacia adelante; (b) Identificación directa inversa [8]

Tal vez uno de los esquemas de modelos neuronales más ampliamente aplicados es el enfoque del modelo directo inverso, en donde, mediante la salida de la planta real y_p la red neuronal determina cuál debió ser la entrada u_p que la

produjo. Una vez que una red neuronal ha sido entrenada para aprender el comportamiento inverso de la planta, entonces esta puede ser configurada para controlar directamente la planta. La configuración básica del modelo directo inverso se muestra en la Figura 1.15 (b). En la arquitectura del modelo directo inverso, la red es entrenada fuera de línea (*offline)* utilizando patrones obtenidos a partir de las características de lazo abierto de la planta (o en lazo cerrado).

Existen otros modelos para la identificación de sistemas no lineales mediante redes neuronales, pero en este documento solo se van a tratar los dos antes mencionados.

1.4.5.2 Diseño de Control

La investigación sobre los sistemas de control basados en redes neuronales ha recibido considerable atención en los últimos años. Las estructuras de control neural más ampliamente utilizadas son similares a las empleadas en los sistemas de control adaptativos. Una red neuronal se utiliza para estimar el sistema no lineal desconocido y la formulación de control es entonces diseñada usando la red neuronal estimada. El proceso de estimación utiliza las entradas y salidas medidas del sistema real y se logra a través del uso de varios tipos de arquitecturas de redes neuronales. El objetivo de este apartado es presentar algunas de las estructuras típicas de control mediante redes neuronales. Los esquemas considerados son: Control Directo, Control Indirecto, Control por Retropropagación a Través del Tiempo y Control Directo Inverso.

Control Directo o Aprendizaje Especializado

En la arquitectura de control directo (Figura 1.16), el controlador neuronal es entrenado en línea (*online*) para minimizar alguna norma del error entre la señal de salida y_p y la de referencia r. El error se propaga hacia atrás a través de la planta en cada muestra, para ajustar los parámetros de la red. Esta arquitectura fue nombrada como control de aprendizaje especializado por Psaltis et al. (1999).

Una ventaja de este tipo de control es que la identificación de la planta no está implicada en este método.

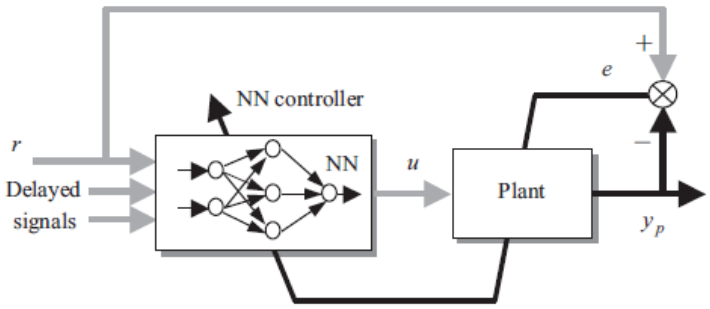


Figura 1.16. Arquitectura de control directo o aprendizaje especializado [8]

Sin embargo, debido a la ubicación de la planta, el Jacobiano de la planta (es decir,$\partial y/\partial u$) es requerido, el cual es difícil de obtener si la dinámica de la planta no se conoce a priori. Con el fin de evitar esto, los elementos del Jacobiano pueden ser aproximados por sus signos, que son las orientaciones de los parámetros de control que influyen en las salidas de la planta.

El entrenamiento involucra a la señal de referencia r como señal de entrada al controlador neuronal. El error $e = r - y_p$ es propagado hacia atrás a través de la planta y utilizado para ajustar los parámetros del controlador. El objetivo del entrenamiento del controlador neuronal es generar la señal de control requerida u para conducir la planta a la salida deseada, tal que la salida de la planta y_p coincida con la señal de referencia r sobre el entrenamiento por épocas k, es decir, $\lim_{k\to\infty} |r - y_p| \leq \varepsilon$ donde $\varepsilon \geq 0$.

La desventaja del control directo es que la estabilidad inicial de la planta no está garantizada para este método de control. A pesar de las desventajas, muchos investigadores han aplicado el esquema de control adaptativo directo a una variedad de sistemas no lineales desconocidos con gran éxito y un mejor rendimiento.

Control Indirecto o Aprendizaje Inverso Especializado

Las principales desventajas en el esquema de control directo eran la determinación del Jacobiano y la estabilidad inicial de la planta. Cuando el inverso de la planta (o el Jacobiano de la planta) no está bien definido o la estabilidad inicial de la planta es fundamental, un esquema de control indirecto es considerablemente más exitoso que el control directo.

En esta arquitectura, un emulador neuronal de la planta es entrenado para representar la respuesta de la planta. El esquema del entrenamiento del emulador neuronal se muestra en la Figura 1.17 (a), que es el mismo descrito en la identificación hacia delante de la planta. La diferencia entre la salida de la planta y_p y la respuesta del emulador $\hat{y}$ se utiliza para ajustar los parámetros del emulador neuronal. Una vez que el emulador neuronal está lo suficiente entrenado, este es utilizado para entrenar un controlador neuronal con la misma señal de referencia r y del error $e_c = \hat{y} - r$. El error e_c se propaga hacia atrás a través del emulador neuronal para ajustar los parámetros del controlador neuronal. El entrenamiento del controlador neuronal es mostrado en Figura 1.17 (b).

Este esquema de control puede ser encontrado con diferentes nombres en la literatura, como el control inverso anticipativo (Narendra, 1995) y aprendizaje inverso especializado (Hunt et al., 1992) [8]. La ventaja añadida del control indirecto es que los parámetros del emulador neuronal pueden ser reajustados en línea durante el funcionamiento del controlador si es que el emulador neuronal parece no ser exacto. Una variante cercana del control indirecto es el control por modelo interno, en el cual un modelo interno se coloca en paralelo con la planta y un controlador es utilizado en serie con la planta. El modelo interno es eventualmente un modelo de la planta hacia adelante como se muestra en la Figura 1.15 (a). Detalles de la arquitectura de control por modelo interno se pueden encontrar en Sen et al. (1998).

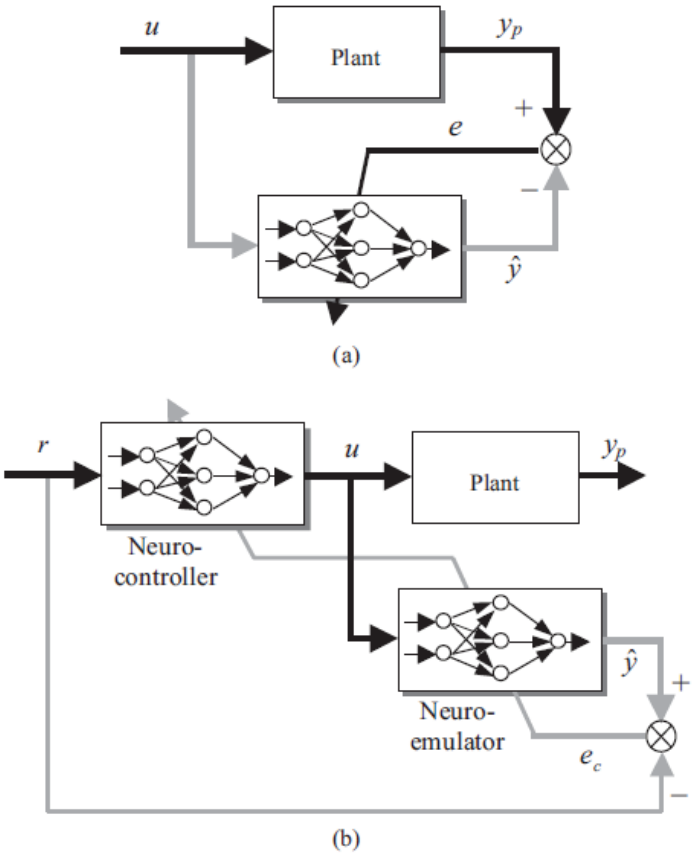


Figura 1.17. Arquitectura del control indirecto. (a) Entrenamiento del emulador neuronal; (b) Entrenamiento del controlador neuronal [8]

Una de las desventajas del esquema de control indirecto parece ser la robustez del controlador, puesto que no hay un lazo de retroalimentación utilizado en la estrategia de control. Mientras que el entrenamiento del controlador neuronal emplea la misma señal de referencia r y la señal de error $e_c = \hat{y} - r$ (que se propaga hacia atrás a través del controlador neuronal), una mala convergencia y un control inestable pueden ser un problema en la etapa inicial. A pesar de estas desventajas, el control indirecto tiene más éxito que el control directo y ha encontrado una amplia gama de aplicaciones.

Control por Retropropagación a Través del Tiempo

Hay otro modelo de arquitectura de control de red neural que utiliza un algoritmo de aprendizaje de retropropagación que fue propuesto por Jordan y Rumelhart (1990), Narendra & Parthasarathy (1990) y Nguyen & Widrow (1990) [8]. Werbos

(1990b) clasificó este método como la arquitectura de retropropagación a través del tiempo. La arquitectura se asemeja a muchas estructuras tradicionales de control adaptativo, conocidas como controles indirectos adaptativos.

En este esquema de control se utilizan dos redes neuronales para controlar la planta, como se muestra en la Figura 1.18. La primera red neuronal es un emulador neuronal, el cual puede ser entrenado fuera de línea (offline) usando una arquitectura de aprendizaje generalizada o incluso en línea (online) mediante la inyección de entradas aleatorias (randómicas) para aprender la dinámica hacia delante de la planta. La segunda red neuronal es un controlador. Esta arquitectura permite el entrenamiento en línea (online) del controlador neuronal así como el error $e = |r - y_p|$ puede ser propagado hacia atrás a través del emulador en cada muestra. Algunas aplicaciones de esta arquitectura se pueden encontrar en Omatu et al. (1995).

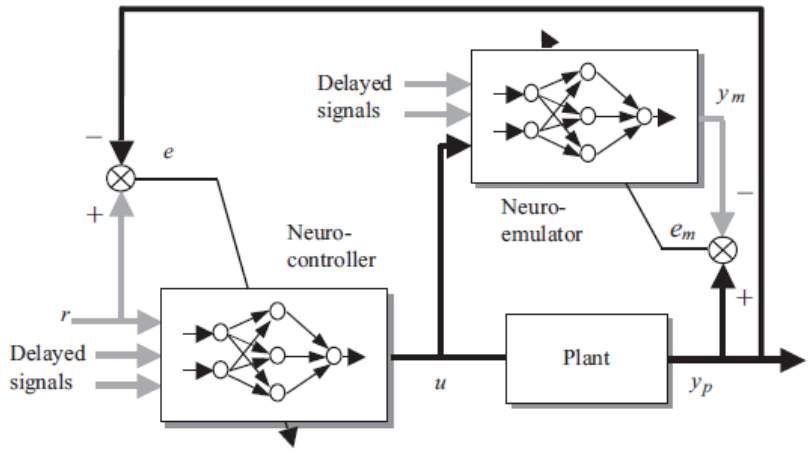


Figura 1.18. Arquitectura de retropropagación a través del tiempo [8]

Control Directo Inverso

Este es el esquema de control que más se ha utilizado en las redes neuronales. Como el término lo sugiere, un control directo inverso utiliza un modelo inverso de la planta como controlador del sistema. El término “control directo inverso” es adoptado desde Werbos (1990a).

El desarrollo del aprendizaje inverso en el diseño de controladores neuronales implica dos fases. En la fase de aprendizaje Figura 1.19 (a), una técnica en línea (*online*) o fuera de línea (*off-line*) se utiliza para modelar la dinámica inversa de la planta. La salida estimada $\hat{u}$ es comparada con la señal entrenada u y el error $e = u - \hat{u}$ se utiliza para entrenar el modelo inverso. El modelo inverso obtenido, se utiliza entonces para generar acciones de control en la fase de aplicación, en donde el modelo estimado se conecta en cascada con la planta para funcionar como si se tratase de un controlador de lazo abierto y los parámetros de este controlador neuronal se ajustan directamente.

Estas dos fases pueden proceder de forma simultánea, según se muestra en la Figura 1.19 (b), por lo tanto, este método de diseño se integra perfectamente en el esquema clásico de control adaptativo. La señal de referencia r debe cubrir un espacio de entrada/salida lo suficientemente grande, mientras se realiza la construcción del modelo inverso antes indicado.

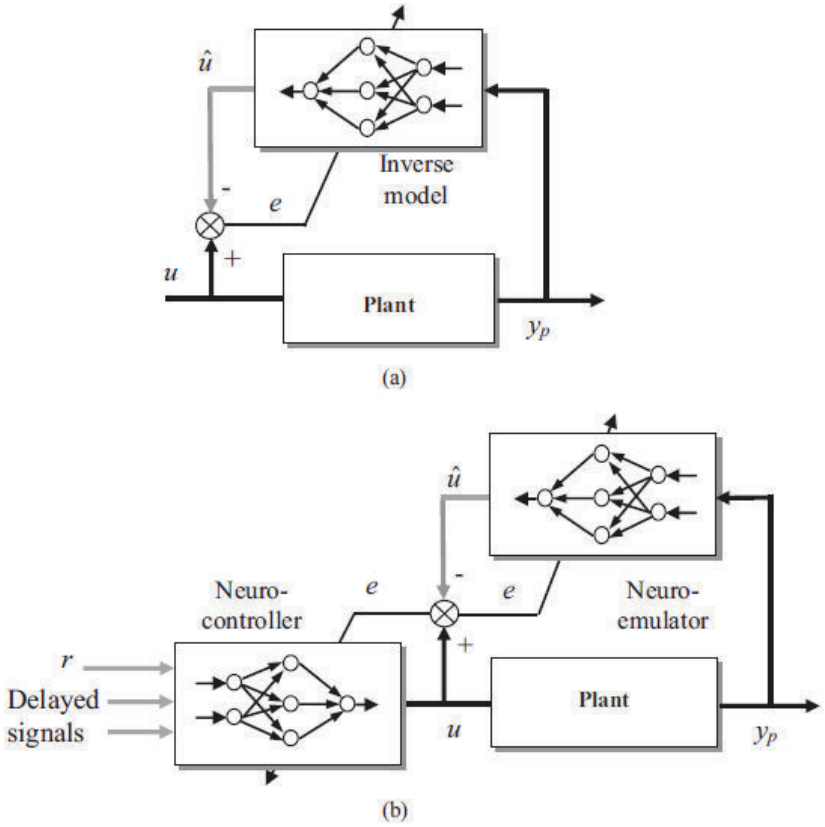


Figura 1.19. Arquitectura del control directo inverso. (a) Modelamiento inverso; (b) Control en lazo abierto [8]

El control directo inverso se basa en la suposición de que existe una correlación de uno a uno desde el estado de la entrada hacia el estado de salida y de que la planta debe permanecer estable en lazo abierto. También se asume que el orden de la planta (es decir, el número de variables de estado) es conocido y todas las variables de estado son medibles. Existen ciertos problemas que se pueden experimentar con este tipo de control debido a las siguientes razones: falta de robustez, lo que resulta del hecho de que no se utiliza ningún error de realimentación directo; aprendizaje ineficiente, causado por la selección de un rango operacional impropio de los datos; los datos operativos reales pueden ser difíciles de definir a priori.

Para superar el problema de la robustez, algunos investigadores usan un controlador PI convencional para incorporar el lazo de realimentación requerido, esto también ayuda a reducir la sensibilidad de todo el sistema de control. Algunos investigadores han comprobado que el controlador neuronal inverso tiene ventajas mucho más sobresalientes en comparación a una gama de controladores como el control de lógica difusa, el control predictivo generalizado y el control PI.

1.5 INTEGRACIÓN DE LÓGICA DIFUSA Y REDES NEURONALES

La lógica difusa y las redes neuronales artificiales tienen propiedades computacionales particulares que las hacen adecuadas para ciertos problemas específicos y para otros no. Por ejemplo, mientras las redes neuronales ofrecen ventajas como el aprendizaje, adaptación, tolerancia a fallas, paralelismo y generalización, no son buenas para explicar cómo han alcanzado sus decisiones y el proceso de aprendizaje es relativamente lento. En cambio los sistemas lógicos difusos, los cuales razonan con información imprecisa a través de un mecanismo de inferencia bajo incertidumbre lingüística, son buenos explicando sus decisiones pero no pueden adquirir automáticamente las reglas que usan para tomarlas (Fuller, 2000) [9]. La Tabla 1.2 muestra una comparación de las propiedades de estas dos tecnologías.

Tabla 1.2. Propiedades de las redes neuronales y los sistemas difusos

Habilidades	Tipo	Sistemas Difusos	Redes Neuronales
Adquisición del Conocimiento	Entradas	Humanos expertos	Conjuntos de muestras
	Herramientas	Iteraciones	Algoritmos
Incertidumbre	Información	Cuantitativa y Cualitativa	Cuantitativa
Razonamiento	Afinación	Enfoque heurístico	Percepción
	Mecanismo	Base de datos y reglas	Cálculo en paralelo
	Velocidad	Baja	Alta
Adaptación	Tolerancia a fallos	Baja	Muy alta
	Aprendizaje	Inducción	Ajuste de pesos
Lenguaje Natural	Implementación	Explícita	Implícita
	Flexibilidad	Alta	Baja

Los sistemas difusos son más favorables en que su comportamiento puede ser explicado basado en reglas difusas y por lo tanto su rendimiento se puede ajustar sintonizando las reglas. Pero ya que, en general, la adquisición del conocimiento es difícil y también el universo de discurso de cada variable de entrada tiene que ser dividido en varios intervalos, las aplicaciones de los sistemas difusos se limitan a los ámbitos donde el conocimiento del experto está disponible y el número de variables de entrada es pequeño.

El análisis anterior revela que los inconvenientes y ventajas relacionados con estos enfoques parecen complementarios y, por tanto, es natural considerar la construcción de un sistema integrado que combine ambos conceptos. Los sistemas neuro-difusos integrados combinan la capacidad de aprendizaje de las redes neuronales artificiales con el poder de interpretación lingüística de los sistemas de inferencia difusos, obteniéndose los siguientes resultados:

a. Aplicabilidad de los algoritmos de aprendizaje desarrollados para redes neuronales.

b. Posibilidad de promover la integración de conocimiento (implícito que puede ser adquirido a través del aprendizaje y explícito que puede ser explicado y entendido).

c. La posibilidad de extraer conocimiento para una base de reglas difusas a partir de un conjunto de datos.

De esta manera, es posible llevar el aprendizaje de bajo nivel y el poder computacional de redes neuronales a los sistemas difusos y llevar el pensamiento y razonamiento humano *if-then (si-entonces)* de alto nivel a las redes neuronales. En este contexto, las redes neuronales se utilizan para sintonizar las funciones de pertenencia de los sistemas difusos que se emplean como sistemas de toma de decisiones para el control de equipos. Aunque la lógica difusa puede codificar el conocimiento de un experto utilizando directamente reglas con etiquetas lingüísticas (*if-then*), por lo general le toma mucho tiempo diseñar y ajustar las funciones de pertenencia que definen cuantitativamente estas etiquetas lingüísticas. Las técnicas de aprendizaje de las redes neuronales pueden automatizar este proceso y reducir sustancialmente el tiempo y costo de desarrollo mientras se mejora la precisión y el rendimiento (Fuller, 2000) [9].

Para superar el problema de la adquisición de conocimiento, las redes neuronales se extienden para extraer automáticamente las reglas difusas a partir de datos numéricos. Los enfoques cooperativos utilizan las redes neuronales para optimizar ciertos parámetros de un sistema difuso ordinario o para preprocesar datos y extraer reglas difusas (control) a partir de datos.

1.5.1 TIPOS DE SISTEMAS NEURO-DIFUSOS

Existen varias maneras de combinar las redes neuronales y la lógica difusa. La clasificación de los sistemas neuro-difusos que se presenta en este trabajo, fue propuesta por (Nauck, 1995) en su artículo *"Beyond Neurofuzzy: Perspectives and Directions".* Las razones para usar tal clasificación es que muchos de los sistemas neuro-difusos implementados hasta hoy, pueden ser enmarcados dentro de estas

divisiones, que se basan en las diferentes combinaciones de las redes neuronales y los sistemas difusos. Los esfuerzos de la fusión de estas dos tecnologías se pueden caracterizar y agrupar en tres modelos principales: modelos neuro-difusos concurrentes, modelos neuro-difusos cooperativos y modelos neuro-difusos híbridos.

1.5.1.1 Modelos Neuro-Difusos Concurrentes

Una red neuronal y un sistema difuso trabajan juntos en la misma tarea, pero sin interactuar el uno en el otro, es decir, ninguno ellos se utilizan para determinar los parámetros del otro.

En un modelo concurrente, la red neuronal ayuda al sistema difuso continuamente (o viceversa) para determinar los parámetros necesarios especialmente si las variables de entrada del controlador no se pueden medir directamente. Tales combinaciones no optimizan el sistema difuso, pero ayudan a mejorar el rendimiento del sistema en general. El aprendizaje tiene lugar sólo en la red neural y el sistema difuso se mantiene sin cambios durante esta fase.
Por lo general, la red neuronal preprocesa las entradas o posprocesa las salidas del sistema difuso. En algunos casos, las salidas difusas pueden no ser directamente aplicables al proceso, en estos casos, la red neuronal puede actuar como un postprocesador de salidas difusas, como se muestra en la Figura 1.20.

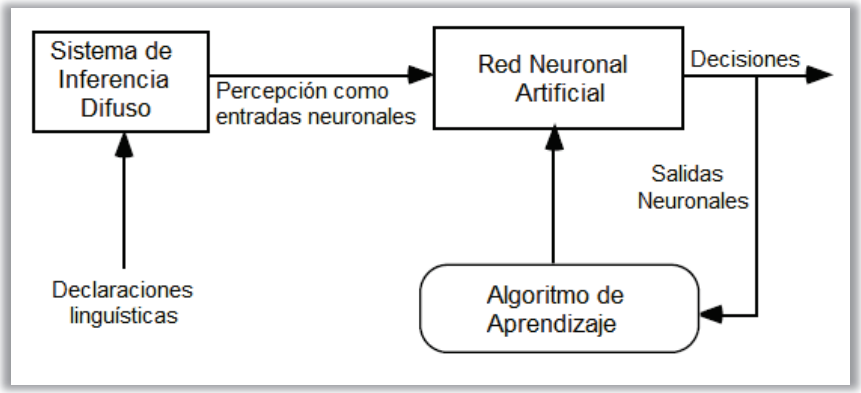


Figura 1.20. Ejemplo esquemático de un modelo concurrente

1.5.1.2 Modelos Neuro-Difusos Cooperativos

En la forma más simple, un modelo cooperativo puede ser considerado como un preprocesador donde el mecanismo de aprendizaje de las redes neuronales determina las funciones de pertenencia del Sistema de Inferencia Difuso o las reglas difusas a partir de los datos de entrenamiento. En la Figura 1.21 se muestra un ejemplo de este tipo de configuraciones.

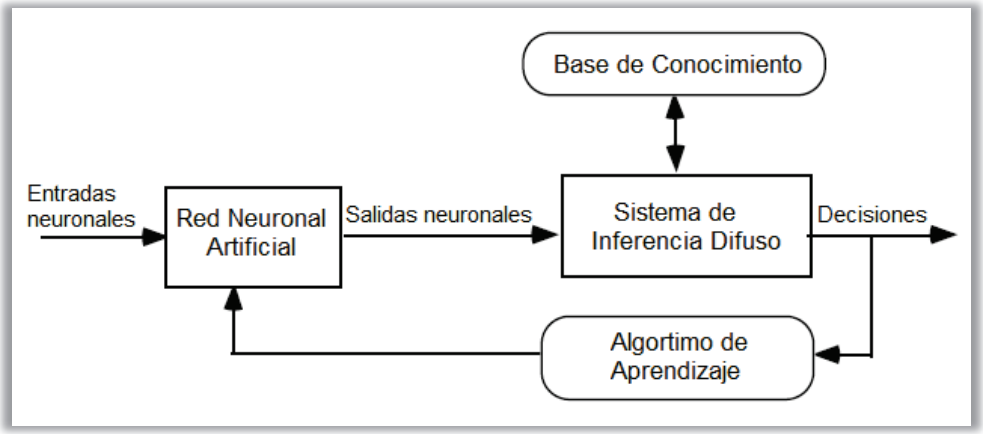


Figura 1.21. Ejemplo esquemático de un modelo cooperativo

En estos modelos, se distinguen dos fases: la de entrenamiento y la de funcionamiento. En la primera, la red neuronal interactúa con el sistema difuso determinando los parámetros del mismo, mientras en la segunda, la red neuronal desaparece dejando sólo el sistema difuso. La base de reglas es usualmente determinada por un enfoque de agrupación (mapas de auto-organización) o algoritmos de agrupamiento difuso. Las funciones de pertenencia se suelen aproximar mediante las redes neuronales a partir de los datos de entrenamiento.

1.5.1.3 Modelos Neuro-Difusos Híbridos

Un sistema neuro-difuso híbrido consiste de un sistema difuso tradicional (tipo Mamdani o Takagi Sugeno) excepto que cada etapa, puede ser representada por una capa de neuronas a las que se puede proveer capacidades de aprendizaje de redes neuronales para optimizar el conocimiento del sistema.

En un modelo híbrido, los algoritmos de aprendizaje de la red neuronal se utilizan para determinar los parámetros de los sistemas de inferencia difusos. Una forma común de aplicar un algoritmo de aprendizaje sobre un sistema difuso es representar este en una arquitectura especial de redes neuronales. Sin embargo, los algoritmos de aprendizaje convencionales de las redes neuronales (gradiente descendente) no se pueden aplicar directamente a un sistema como las funciones utilizadas en el proceso de inferencia que son por lo general no diferenciables. Este problema puede abordarse mediante el uso de funciones diferenciables en el sistema de inferencia difuso o combinando el algoritmo de aprendizaje neuronal estándar con ciertas técnicas de optimización, como por ejemplo el estimador de mínimos cuadrados LSE.

Existen diferentes variantes de los modelos neuro-difusos híbridos (Mamdani o Takagi Sugeno) que han ido apareciendo a medida que se ha ido profundizando en la investigación, diseño e implementación de estos nuevos sistemas de control inteligentes (A. Abraham, 2005) [10]. Entre los métodos híbridos más destacados se tienen los siguientes: Adaptive Network Based Fuzzy Inference System (ANFIS), Fuzzy Adaptive Learning Control Network (FALCON), Generalized Approximate Reasoning Based Intelligent Control (GARIC), Neuro-Fuzzy Controller (NEFCON), Neuro-Fuzzy Classification (NEFCLASS), Neuro-Fuzzy Function Approximation (NEFPROX), Fuzzy Inference Environment Software with Tuning (FINEST), Self Constructing Neural Fuzzy Inference Network (SONFIN), Fuzzy Net (FUN), Evolving Fuzzy Neural Networks (EFuNN), Dynamic Evolving Fuzzy Neural Networks (dmEFuNNs).

En el presente documento se va hacer referencia al modelo ANFIS, el cual va hacer profundizado en el Capítulo 2. ANFIS implementa un sistema de inferencia difuso Takagi-Sugeno y tiene una arquitectura formada por cinco capas, tal y como se muestra en el ejemplo de la Figura 1.22, la explicación de esta arquitectura será retomada en el Capítulo 2.

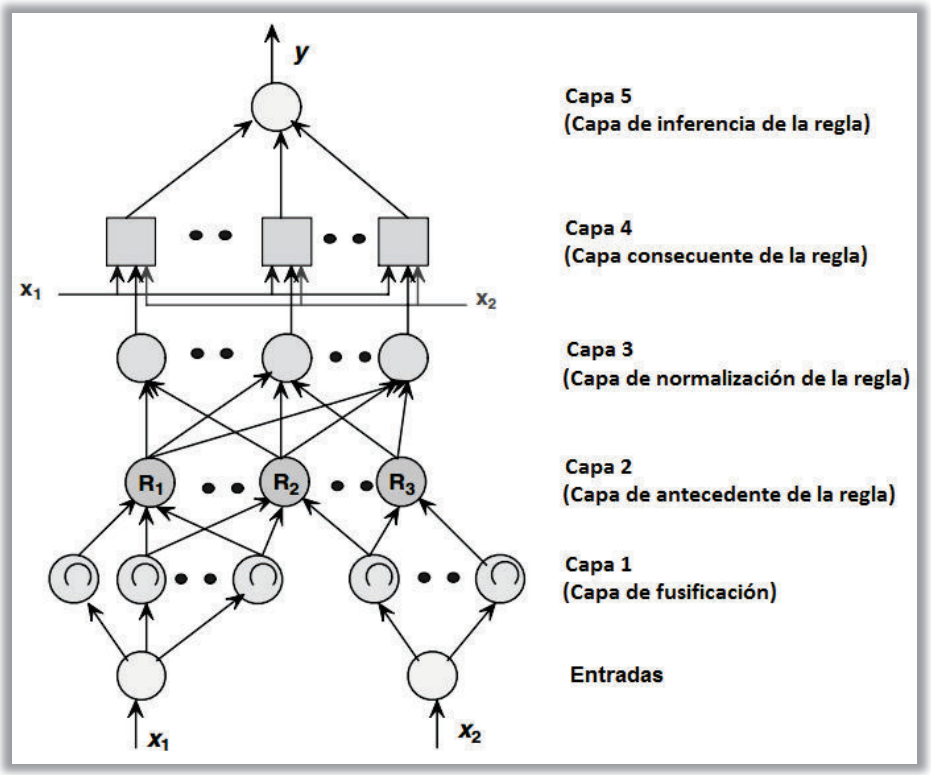


Figura 1.22. Arquitectura de un modelo neuro-difuso tipo ANFIS [10]

1.5.2 CONTROL NEURO-DIFUSO

Como ya se ha mencionado, si en un sistema de control convencional se sustituye el bloque del controlador con redes neuronales o sistemas de inferencia difusos, entonces se tendrá sistemas de control neuronales o difusos, respectivamente, en donde los controladores están destinados a cumplir los objetivos de control establecidos.

En la misma línea, el control neuro-difuso se refiere a los métodos de diseño para los controladores de lógica difusa que emplean técnicas de redes neuronales. En particular, este trabajo centrará su estudio en los métodos de diseño para el controlador ANFIS (Redes Adaptativas basadas en Sistemas de Inferencia Difuso, ver Capítulo 2).

Como se ha demostrado en los apartados anteriores, los Sistemas de Inferencia

Difusos (FIS) y los Perceptrones Multicapas (MLP) son casos especiales de una estructura computacional más general conocida como las redes adaptativas, por lo tanto, ambos de estos casos heredan la capacidad de aprendizaje de retropropagación de las redes adaptativas. Sin embargo, el Sistema de Inferencia Difuso es superior al Perceptrón Multicapa puesto que puede representar el conocimiento estructurado, mientras que el segundo es más o menos como una caja negra. Como resultado, se puede identificar algunas propiedades únicas de los controladores ANFIS: capacidad de aprendizaje, funcionamiento en paralelo, representación del conocimiento estructurado y mejor integración con otros métodos de diseño de control.

La mayoría de los métodos de diseño neuro-difusos para la construcción de un controlador ANFIS, se derivan directamente de los esquemas de control para redes neuronales explicados en el apartado 1.4.5 y estos métodos por lo general se aplican directamente a los casos más generales de diseño de controladores mediante redes adaptativas.

La mayoría de los controladores neuronales o difusos son no lineales; por lo tanto, el análisis riguroso para los sistemas de control neuro-difusos es difícil y sigue siendo un área desafiante para una mayor investigación. Por otra parte, un controlador neuro-difuso por lo general contiene un gran número de parámetros; por lo que es más versátil que un controlador lineal en el trato con las características no lineales de la planta. Por lo tanto, los controladores neuro-difusos casi siempre superan a los controladores lineales puros si se diseñan correctamente.

CAPÍTULO 2
DESARROLLO DEL CONTROLADOR ANFIS

2.1 INTRODUCCIÓN

Ciertas arquitecturas y reglas de aprendizaje de las redes adaptativas ya se han descrito en el Capítulo 1. Funcionalmente, casi no hay restricciones en las funciones de nodo de una red adaptativa a excepción del requisito de la diferenciabilidad parcial; estructuralmente, la única limitación de la configuración de la red es que debe ser de tipo *feedforward* (de avance). Debido a estas restricciones mínimas, las redes adaptativas pueden ser utilizadas directamente en una amplia variedad de aplicaciones de modelado, toma de decisiones, procesamiento de señales y sistemas de control.

En este capítulo, se estudia y se desarrolla el algoritmo computacional de una clase en particular de redes adaptativas que son funcionalmente equivalentes a los Sistemas de Inferencia Difusos. La arquitectura propuesta es conocida como ANFIS, por sus siglas en inglés *Adaptive Network based Fuzzy Inference System*, que traducido al español es interpretado como *Redes Adaptativas basadas en Sistemas de Inferencia Difuso*. El modelo neuro-difuso ANFIS es uno de los esquemas más exitosos que combinan los beneficios de las redes neuronales y la lógica difusa en una sola cápsula (Jang, 1993) [1].

En primer lugar se describe el funcionamiento y el modelado matemático de la planta TRMS sobre la cual se va a aplicar el controlador ANFIS, con la finalidad de realizar las pruebas de simulación que posteriormente serán implementadas sobre la planta física real. Luego se describe de forma detallada la arquitectura del sistema ANFIS con el fin de aplicar sobre esta la regla de aprendizaje híbrida e implementar el algoritmo de programación utilizando el software Matlab junto con la herramienta gráfica de Simulink.

2.2 SISTEMA AERODINÁMICO TRMS

El Sistema MIMO de Rotores Gemelos (Twin Rotor MIMO System - TRMS), ver Figura 2.1, es un laboratorio diseñado por la empresa Feedback Instruments para realizar expresamente experimentos de control. En ciertos aspectos su comportamiento se asemeja al de un helicóptero, pero con algunas simplificaciones significantes. Desde el punto de vista de control ejemplifica un sistema MIMO no lineal de orden superior con acoplamientos cruzados muy significativos.

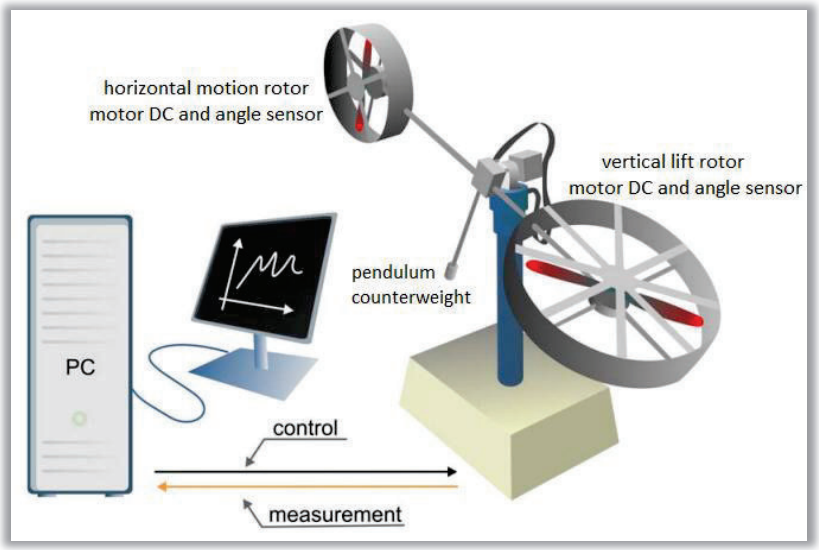


Figura 2.1. Esquema general de control del sistema TRMS [11]

El TRMS consta de una viga articulada en la base de tal manera que puede girar libremente al mismo tiempo en el plano vertical y horizontal (*pitch* y *yaw*). En ambos extremos de la viga hay dos rotores, el principal y el de la cola, estos son impulsados por motores de corriente continua. Si fuese necesario, uno o ambos ejes de rotación pueden ser bloqueados por medio de dos tornillos de bloqueo previstos para restringir físicamente la rotación tanto en el plano vertical u horizontal. Así, el sistema permite implementar sistemas de control con uno o dos grados de libertad (1 DOF o 2 DOF). El TMRS consta de un brazo de contrapeso que está fijado en el punto de pivote central de la viga, cuya finalidad es la de amortiguar los movimientos bruscos que puedan presentarse en el sistema.

El estado de la viga es descrito por 2 variables de proceso: el ángulo horizontal (φ) y ángulo vertical (ψ) medidos por los sensores de posición (encoders incrementales). El TRMS es controlado variando la tensión de los motores DC, por tanto, las entradas de control son las tensiones ($u_1(t)$ y $u_2(t)$) suministradas a los motores DC, cuya variación modifican la fuerza aerodinámica sobre los rotores. Un cambio en el valor de la tensión hace que cambie la velocidad de rotación de la hélice, lo que resulta en un cambio de la posición correspondiente de la viga.

2.2.1 MODELADO DINÁMICO DEL SISTEMA TRMS

Similar a la mayoría de los vehículos de vuelo, el TRMS se compone de varias partes elásticas tales como las superficies de los rotores y los motores. Las fuerzas aerodinámicas no lineales, la acción de la gravedad sobre el vehículo y las estructuras flexibles aumentan la complejidad del modelamiento del sistema y esto resulta en un análisis matemático difícil. Para fines de control, es necesario determinar un modelo representativo que emule las mismas características dinámicas del sistema TRMS real. El modelo matemático del TRMS se desarrolla considerando las siguientes suposiciones:

a. La dinámica del subsistema de la hélice puede ser descrita mediante ecuaciones diferenciales de primer orden y la fricción en el sistema es del tipo viscoso.

b. Basados en las ecuaciones de Lagrange, se puede clasificar y estudiar al sistema mecánico en dos partes: aquel donde se ejercen las fuerzas alrededor del eje horizontal y otro donde aparecen las fuerzas alrededor del eje vertical.

Los parámetros del sistema TRMS que serán empleados en este análisis son: el momento gravitacional M_{FG}, los momentos de fuerzas de fricción $M_{B\psi}$ y $M_{B\varphi}$, el momento giroscópico M_G, el momento de reacción cruzada M_R, el momento de inercia del rotor vertical I_1 y el momento de inercia del rotor horizontal I_2.

Considere la rotación de la viga en el plano vertical (alrededor del eje horizontal). Los torques de conducción son producidos por las hélices y la rotación puede ser descrita en principio como el movimiento de un péndulo. De acuerdo al diagrama mecánico mostrado en la Figura 2.2, las ecuaciones no-lineales que describen la dinámica del sistema TRMS pueden ser escritas de la siguiente manera.

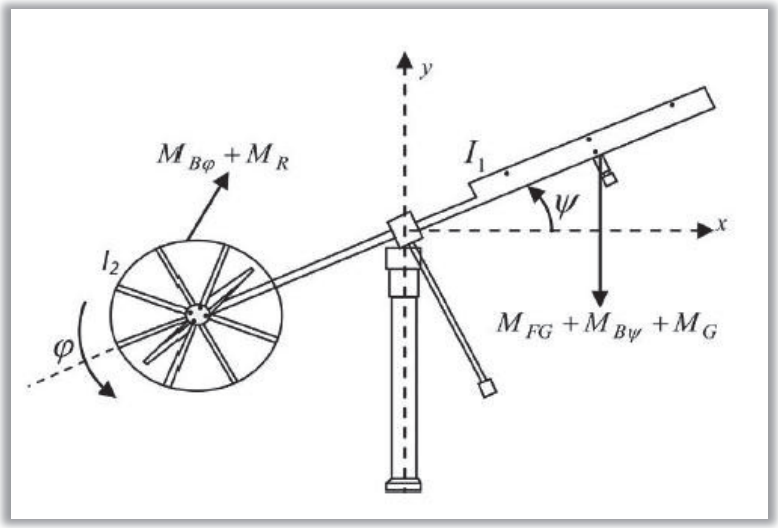


Figura 2.2. Diagrama de cuerpo libre del TRMS [12]

Modelo del Rotor Principal

La siguiente ecuación de momento se puede derivada para el movimiento vertical:

$$I_1 \ddot{\psi} = M_1 - M_{FG} - M_{B\psi} - M_G \qquad (2.1)$$

donde, el momento de la hélice principal M_1 es una función no lineal que representa las características estáticas del motor DC y está descrita por:

$$M_1 = a_1 \tau_1^2 + b_1 \tau_1 \qquad (2.2)$$

El resto de momentos que forman parten de la ecuación 2.1 quedan descritos por:

$$M_{FG} = M_g \, sin\psi$$

$$M_{B\psi} = B_{1\psi} \, \dot{\psi}$$

$$M_G = K_{gy} \, M_1 \, \dot{\varphi} \, cos\psi - K_{gx} \, \dot{\varphi}^2 \, sin2\psi$$

El motor y el circuito de control eléctrico son aproximados por una función de transferencia de primer orden, así en el dominio de Laplace el momento del motor τ_1 es descrito por:

$$\tau_1 = \frac{k_{11}}{T_{11}s + T_{10}} u_1 \tag{2.3}$$

donde, u_1 es el voltaje de entrada del motor DC, T_{11} es la constante de tiempo del rotor principal y k_{11} es la ganancia estática del motor DC.

Modelo del Rotor de Cola

Utilizando un análisis similar al anterior, se puede describir el movimiento de la viga en el plano horizontal como:

$$I_2\, \ddot{\varphi} = M_2 - M_{B\varphi} - M_R \tag{2.4}$$

donde, el momento de la hélice de cola M_2 es una función no lineal que representa las características estáticas del motor DC y está descrita por:

$$M_2 = a_2\tau_2^2 + b_2\tau_2 \tag{2.5}$$

El resto de momentos que forman parten de la ecuación 2.4 quedan descritos por:

$$M_{B\varphi} = B_{1\varphi}\, \dot{\varphi}$$

$$M_R = \frac{k_c(T_0 s + 1)}{T_p s + 1} M_1$$

El motor y el circuito de control eléctrico son aproximados por una función de transferencia de primer orden, así en el dominio de Laplace el momento del motor τ_2 es descrito por:

$$\tau_2 = \frac{k_{22}}{T_{21}s + T_{20}} u_2 \tag{2.6}$$

donde, u_2 es el voltaje de entrada del motor DC, T_{21} es la constante de tiempo del rotor de cola y k_{22} es la ganancia estática del motor DC.

Partiendo de las ecuaciones antes encontradas, se puede concluir que la dinámica total del sistema TRMS queda descrita por el siguiente par de ecuaciones:

$$\ddot{\psi} = \frac{1}{I_1}\left[-M_g\, sin\psi - B_{1\psi}\,\dot{\psi} + K_{gx}\,\dot{\varphi}^2\, sin2\psi + (a_1\tau_1^2 + b_1\tau_1)\left(1 - K_{gy}\,\dot{\varphi}\, cos\psi\right)\right]$$

$$\ddot{\varphi} = \frac{1}{I_2}\left[-B_{1\varphi}\,\dot{\varphi} - \frac{k_c(T_0 s + 1)}{T_p s + 1}(a_1\tau_1^2 + b_1\tau_1) + (a_2\tau_2^2 + b_2\tau_2)\right] \qquad (2.7)$$

Los diferentes parámetros del TRMS utilizados en su modelado dinámico se resumen en la Tabla 2.1.

Tabla 2.1. Parámetros físicos del sistema TRMS

Símbolo	Definición	Valor
I_1	Momento de inercia del rotor vertical	$6.8x10^{-2}kg.m^2$
I_2	Momento de inercia del rotor horizontal	$2x10^{-2}kg.m^2$
a_1	Parámetro de característica estática	0.0135
b_1	Parámetro de característica estática	0.0924
a_2	Parámetro de característica estática	0.02
b_2	Parámetro de característica estática	0.09
M_g	Momento gravitacional	$0.32\ N.m$
$B_{1\psi}$	Parámetro del momento de fricción	$6x10^{-3}N.m.s/rad$
$B_{1\varphi}$	Parámetro del momento de fricción	$1x10^{-1}N.m.s/rad$
K_{gy}	Parámetro del momento giroscópico	$0.05\ s/rad$
K_{gx}	Parámetro del momento giroscópico	$0.0163\ s/rad$
k_{11}	Ganancia del Motor 1	1.1
k_{22}	Ganancia del Motor 2	0.8
T_{11}	Parámetro denominador del Motor 1	1.2
T_{10}	Parámetro denominador del Motor 1	1
T_{21}	Parámetro denominador del Motor 2	1
T_{20}	Parámetro denominador del Motor 2	1
T_p	Parámetro de momento de reacción cruzada	2
T_0	Parámetro de momento de reacción cruzada	3.5
k_c	Parámetro de momento de reacción cruzada	-0.2

Modificado de (Zeghlache, Kara & Saigaa, 2014) [12]

El diagrama de bloques del modelo matemático determinado para el sistema TRMS, que incluye al rotor principal y al rotor de cola, se presenta en la Figura 2.3.

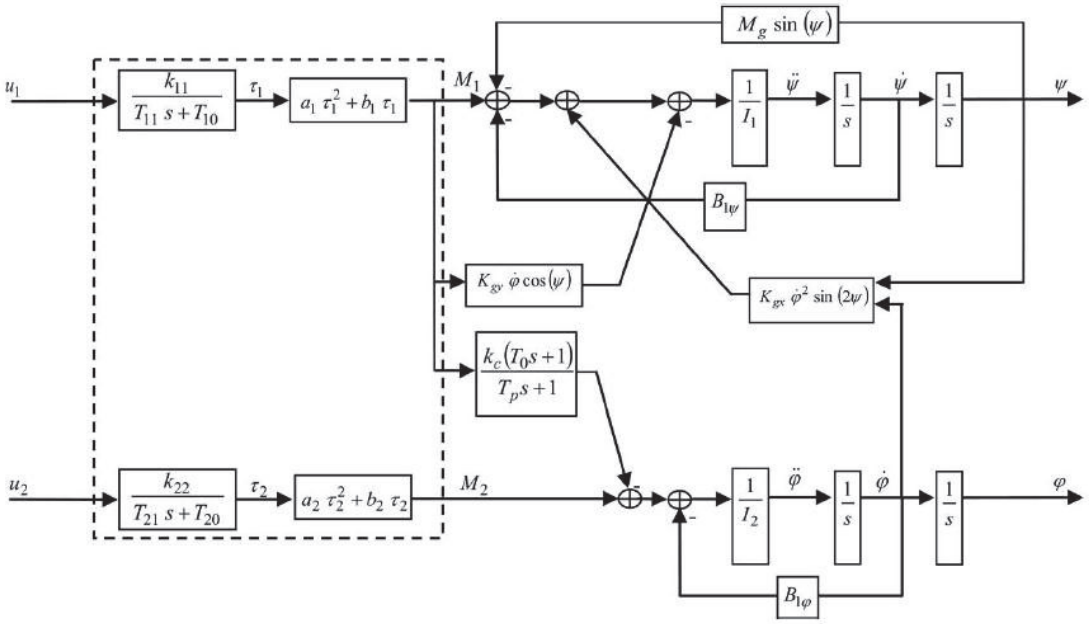


Figura 2.3. Diagrama de bloques del sistema TRMS [12]

El rango de las señales de control ($u_1(t)$ y $u_2(t)$) se establece en [-2.5V... + 2.5V]. Junto con el sistema TRMS, la empresa Feedback Instruments suministra un grupo de ejemplos de control desarrollados en el software Matlab y Simulink, cuyo propósito es el de familiarizar al usuario con el manejo del sistema; la explicación de estos ejemplos puede ser encontrada en el documento titulado como “Control Experiments” (Feedback, 2006) [11]. Para conocer cómo se lleva a cabo la puesta en marcha del sistema TRMS consulte el documento “Installation & Commissioning” (Feedback, 2006) [13].

2.3 REDES ADAPTATIVAS BASADAS EN SISTEMAS DE INFERENCIA DIFUSO: ANFIS

Los sistemas ANFIS por sus siglas en inglés *Adaptive Network based Fuzzy Inference System* o sistemáticamente equivalente *Adaptive Neuro-Fuzzy Inference System*, son sistemas basados en redes neurales adaptativas con inferencia difusa. En dichos sistemas se tiene como principio el uso de diferentes métodos para el ajuste de los parámetros, entre ellos se encuentra el método de retropropagación del error mediante el gradiente descendente y los mínimos cuadrados recursivos. El principio de funcionamiento de este tipo de topologías fue propuesto por (Jang,1993) [1] en su trabajo de tesis "*ANFIS: Adaptive Network based Fuzzy Inference System*".

El ANFIS modela un Sistema de Inferencia Difuso (FIS) en el cual sus parámetros se ajustan mediante un algoritmo de aprendizaje por retropropagación (propio de las redes neuronales) basándose en un conjunto de datos de entrada/salida (datos de entrenamiento), lo cual le permite al sistema aprender, es decir, le permite al sistema identificar y ajustar los parámetros de la estructura del FIS (reglas y funciones de pertenencia).

Entre las características y ventajas más relevantes de un sistema ANFIS se tienen: fácil de implementar, aprendizaje rápido y preciso, sólidas capacidades de generalización, facilidades de explicación a través de reglas difusas y fácil incorporación tanto de conocimientos lingüísticos y numéricos para la resolución de problemas (Jang & Sun, 1995; Jang et al ., 1997) [2] [3].

2.3.1 ARQUITECTURA DE LA RED ANFIS

La arquitectura que se presenta en el desarrollo del presente trabajo, es un tipo de red adaptativa, la cual funcionalmente es equivalente a un Sistema de Inferencias Difuso. Esta arquitectura puede representar tanto modelos difusos tipo Sugeno de primer orden y Sugeno de orden cero (ver apartado 1.2.3).

Debido a la mayor rapidez en el entrenamiento y a las mejores características que presentan los sistemas de primer orden sobre los de orden cero, son éstos con los que se desarrollan en el presente trabajo. La siguiente descripción se centrará en ello, teniendo en cuenta que fácilmente se puede hacer una generalización hacia los otros tipos de sistemas, como el de orden cero.

El modelo nuero-difuso ANFIS consta de cinco capas según se muestra en la Figura 2.4, la cual es una representación gráfica del modelo TSK (o modelo Sugeno) y en donde cada capa tiene un número de elementos distribuidos en $(nK, M, M, M, 1)$ respectivamente.

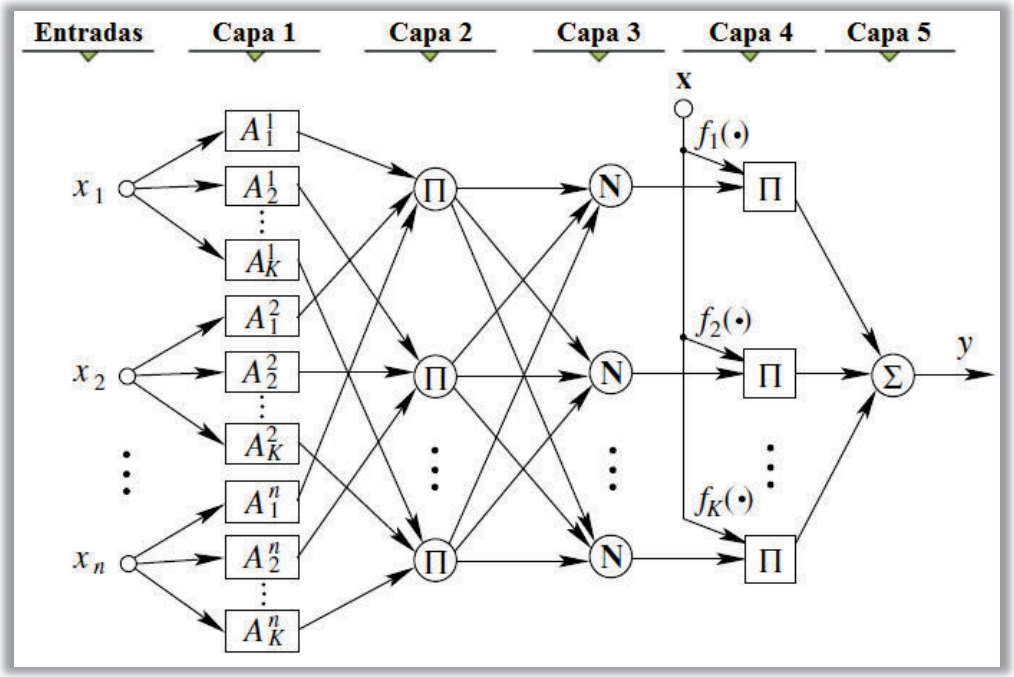


Figura 2.4. Arquitectura del modelo ANFIS tipo Sugeno de Primer Orden [14]

Para explicar el funcionamiento de la arquitectura mostrada en la Figura 2.4, se considera que el sistema de inferencia difuso consta de n entradas $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$ y una salida y. Para el modelo difuso tipo Takagi y Sugeno se tiene un conjunto de reglas difusas si- entonces *(if-then),* en donde la *j-ésima* regla para el modelo ANFIS se puede definir como:

$$\boldsymbol{if}\ x_1\ es\ A_j^1\ \text{y}\ x_2\ es\ A_j^2\ \text{y}\ \dots\ x_n\ es\ A_j^n\ \boldsymbol{then}\ y = f_j(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n} a_{ji}x_i + a_{j0}$$

o expresado en forma matricial:

$$Regla\ j\text{-}ésima\colon\ \boldsymbol{if}\ \ \boldsymbol{x}\ es\ \boldsymbol{A}_j\ \ \boldsymbol{then}\ \ y = f_j(\boldsymbol{x}) = \sum_{i=1}^{n} a_{ji}x_i + a_{j0}$$

para $j = 1, 2, \dots, K$ y en donde $\boldsymbol{A}_j = \left\{A_j^1,\ A_j^2, \dots, A_j^n\right\}$ es el conjunto difuso o conjunto de etiquetas lingüísticas (largo, corto, etc) asociado a la función de nodo y a_{ji} con $i = 1, 2, \dots, n$ son los parámetros del consecuente. En las siguientes líneas se describe el funcionamiento y las fórmulas asociadas a cada capa.

Capa 1 (Capa de Fusificación): En esta capa se lleva a cabo la *fusificación*. Esto significa que a cada entrada no difusa se le asigna un valor de pertenencia para cada subconjunto difuso que comprende el universo de discurso de esta entrada (ver sección 1.2.2). Esta capa consta de nK nodos y cada nodo en esta capa es un nodo adaptable cuya salida está definida por la siguiente función:

$$O_{ij}^{(1)} = \mu_{A_j^i}(x_i) \tag{2.8}$$

que corresponde al j*-ésimo* término lingüístico de la i*-ésima* variable de entrada x_i, para $i = 1, 2, \dots, n$ y $j = 1, 2, \dots, K$. En este contexto, A_j^i define una partición de un espacio de entrada x_i mediante una etiqueta lingüística (tal como "pequeño" o "alto") asociada con el nodo i, j. En otras palabras, $O_{ij}^{(1)}$ es el grado de pertenencia de un conjunto difuso A_j^i y especifica el grado en que la entrada x_i satisface al cuantificador A_j^i. Generalmente, la función de pertenencia MF utilizada para A_j^i, expresada como $\mu_{A_j^i}(x_i)$, es la función campana generalizada cuya expresión matemática está dada por:

$$\mu_{A_j^i}(x_i) = \mu\left(x_i;\ a_j^i,\ b_j^i, c_j^i\right) = \frac{1}{1 + \left[\left(\dfrac{x_i - c_j^i}{a_j^i}\right)^2\right]^{b_j^i}} \tag{2.9}$$

donde $\left\{a_j^i,\ b_j^i, c_j^i\right\}$ es el conjunto de parámetros que definen la posición y forma de la campana: a_j^i especifica la mitad del ancho de la campana, b_j^i (junto con a_j^i)

controlan las pendientes en los puntos de cruce (donde el valor MF es de 0,5) y c_j^i representa el centro de la MF; esto se ilustra en la Figura 2.5.

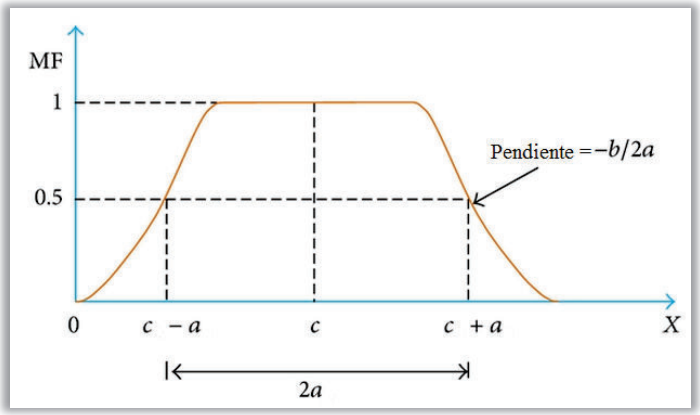


Figura 2.5. Función de pertenencia tipo campana

Los parámetros $\{a_j^i, b_j^i, c_j^i\}$ en esta capa son referidos como *parámetros de premisa* o *parámetros no lineales* y son ajustados durante el proceso de entrenamiento por el *algoritmo de retropropagación de error*. Cuando los valores de estos parámetros cambian, la forma de la campana variará, por consiguiente, así como la diversidad de formas que las funciones de pertenencia puedan presentar en las etiquetas lingüísticas de A_j^i. Alternativamente, aquí se puede utilizar la función gaussiana como función de pertenencia:

$$\mu_{A_j^i}(x_i) = e^{-\frac{1}{2}\left(\frac{x_i - c_j^i}{\sigma_j^i}\right)^2} \tag{2.10}$$

De hecho, cualquier función continua y derivable (en cualquier segmento), así como las funciones de pertenencia más usadas, la trapezoidal y la triangular, son también fuertes candidatas para ser funciones de nodo en esta capa (Jang, 1993). Si se toma como función de pertenencia a la función campana, entonces el número total de parámetros no lineales en esta capa está dado por la siguiente expresión

$$\#total\ de\ parámetros\ no\ lineales = 3 * \#variables\ de\ entrada * \#subconjuntos\ \text{difusos}$$

$$\#total\ de\ parámetros\ no\ lineales = 3 * n * K \tag{2.11}$$

Capa 2 (Capa del Antecedente de la Regla): Esta capa tiene M nodos difusos, en donde el valor de M depende del tipo de partición del espacio de entrada (ver apartado 1.2.4), así por ejemplo para una partición tipo rejilla se tendría que

$$M = \#reglas = \#subconjuntos\ difusos^{\#variables\ de\ entrada} = K^n \qquad (2.12)$$

y cada nodo realiza una operación AND difusa, así un nodo en esta capa representa la parte antecedente de una regla (ver apartado 1.2.1). Los nodos de esta capa son nodos fijos y se etiquetan con el símbolo Π; la salida de cada nodo es la operación norma-T de todas las señales que entran a él, esto da lugar a que la salida del nodo sea el producto de todas sus entradas:

$$O_m^{(2)} = w_m = \prod_{i=1}^{n} O_{ij}^{(1)} = \prod_{i=1}^{n} \mu_{A_j^i}(x_i) \qquad (2.13)$$

para $m = 1, 2, \dots, M$ y $j = 1, 2, \dots, K$. La salida de cada nodo en esta capa representa la intensidad de disparo (o el valor de activación) de la regla difusa correspondiente. En general, se puede emplear cualquier operador norma-T (que realice la operación AND difusa) como función de nodo en esta capa.

Capa 3 (Capa de Normalización de la Regla): Esta capa se compone sólo de nodos fijos etiquetados con el símbolo N. El m-*ésimo* nodo calcula la relación de la intensidad de disparo de la m-*ésima* regla con respecto a la suma de las intensidades de disparo de todas las reglas difusas. Esto da como resultado la normalización del valor de activación para cada regla difusa. Esta operación puede ser escrita como:

$$O_m^{(3)} = \overline{w_m} = \frac{O_m^{(2)}}{\sum_{k=1}^{M} O_k^{(2)}} \qquad (2.14)$$

para $m = 1, 2, \dots, M$. A cada salida de esta capa se le llama *intensidad de disparo normalizada*.

Capa 4 (Capa del Consecuente de la Regla): Cada nodo en esta capa es un nodo adaptable con una función definida por:

$$O_m^{(4)} = O_m^{(3)}.\, f_m(\boldsymbol{x}) = \overline{w_m}.\, f_m(\boldsymbol{x}) = \overline{w_m}.\, f_m(x_1, x_2, \dots, x_n) = \overline{w_m}.\left[\sum_{i=1}^{n}(a_{mi}x_i) + a_{m0}\right]$$

para $m = 1, 2, \dots, M$ y $i = 1, 2, \dots, n$; donde $f_m(\cdot)$ está dada para el m-*ésimo* nodo en la capa 4. Expandiendo el término de suma, la salida del nodo m también puede ser expresada como la siguiente función lineal:

$$O_m^{(4)} = \overline{w_m}.\,(a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + a_{m0}) \qquad (2.15)$$

donde $\overline{w_m}$ es el valor de activación normalizado de la m-*ésima* regla, calculada con la ayuda de la ecuación 2.14 y $\{a_{m1},\ a_{m2}, \dots, a_{mn}, a_{m0}\}$ es el conjunto de parámetros ajustables en este nodo. Los parámetros en esta capa son referidos como *parámetros del consecuente* o *parámetros lineales* del sistema ANFIS y son ajustados por el algoritmo de *mínimos cuadrados recursivos* (*RLS - Recursive Least Squares*, ver apartado 1.3).

El número total de parámetros lineales en esta capa está dado por la siguiente expresión

$$\#total\ de\ parámetros\ lineales = (\#variables\ de\ entrada + 1) * \#reglas$$
$$\#total\ de\ parámetros\ lineales = (n + 1) * \#reglas$$

Si se toma en consideración una partición del espacio de entrada tipo rejilla (ver apartado 1.2.4) entonces $\#reglas = \#subconjuntos\ difusos^{\#variables\ de\ entrada} = K^n$ y la ecuación anterior quedaría como:

$$\#total\ de\ parámetros\ lineales = (n + 1) * K^n \qquad (2.16)$$

Capa 5 (Capa de Inferencia de la Regla): El único nodo presente en esta capa, es un nodo fijo denotado por símbolo Σ, el cual calcula la salida global como la suma de todas las señales que entran a él:

$$O^{(5)} = salida\ global = y = \sum_{m=1}^{M} O_m^{(4)} = \sum_{m=1}^{M} \overline{w_m}.f_m(\boldsymbol{x}) = \frac{\sum_{m=1}^{M} w_m.f_m(\boldsymbol{x})}{\sum_{m=1}^{M} w_m}$$

$$y = \frac{\sum_{m=1}^{M}\left[\left(\prod_{i=1}^{n}\mu_{A_j^i}(x_i)\right)\left(\sum_{i=1}^{n}(a_{mi}x_i) + a_{m0}\right)\right]}{\sum_{m=1}^{M}\left(\prod_{i=1}^{n}\mu_{A_j^i}(x_i)\right)} \qquad (2.17)$$

De esta forma, lo que se obtiene es una red adaptiva que es funcionalmente equivalente a un Sistema de Inferencia Difuso tipo Sugeno.

Para aclarar los conocimientos antes explicados, a continuación se presenta un ejemplo de un modelo ANFIS en el cual se considera un Sistema de Inferencia Difuso con dos entradas $\boldsymbol{x} = [x_1,\ x_2]^T$ y una salida y, tal y como se observa en la Figura 2.6.

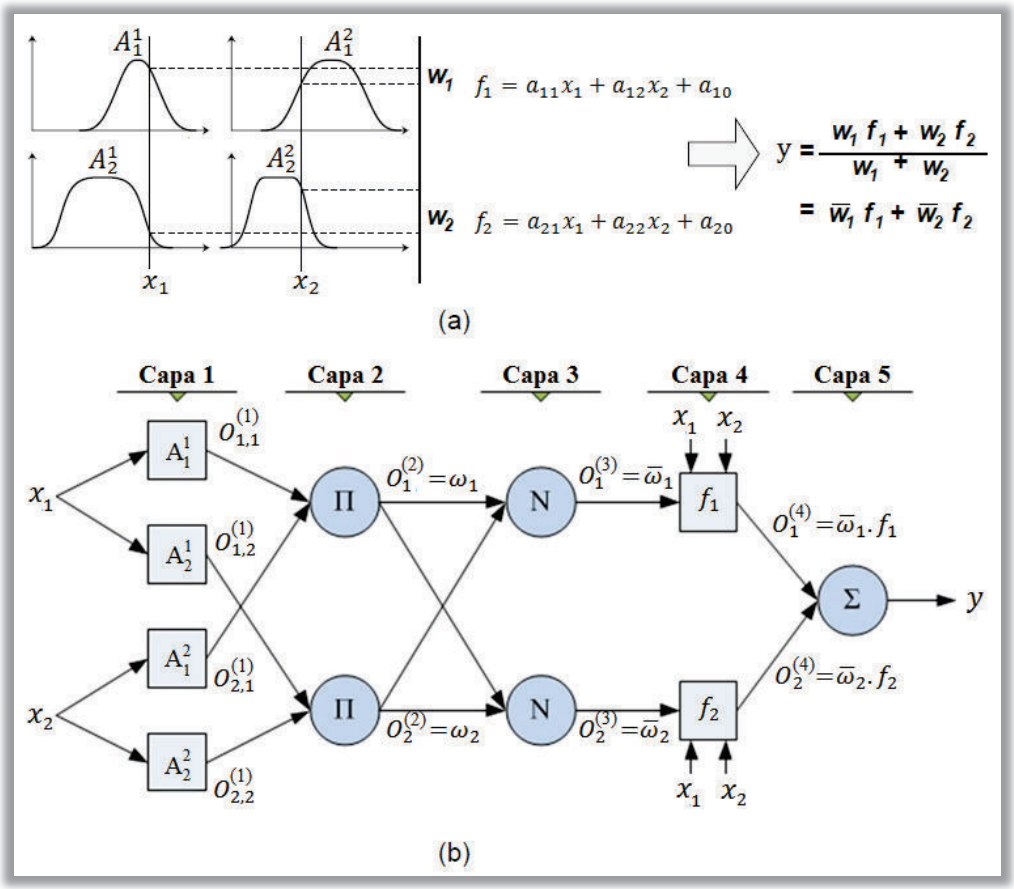


Figura 2.6. (a) Sistema de Inferencia Difuso tipo Sugeno de primer orden con dos entradas y dos reglas difusas; (b) Arquitectura ANFIS equivalente

La Figura 2.6.a ilustra el mecanismo de razonamiento para un Sistema de Inferencia Difuso tipo Sugeno; la arquitectura ANFIS correspondiente se muestra en la Figura 2.6.b, en la cual, los nodos en una misma capa realizan funciones similares; además se considera que la partición del espacio de entrada es del tipo dispersa (ver apartado 1.2.4 del Capítulo 1), es decir

$$\#reglas = \#subconjuntos\ difusos\ de\ cada\ entrada = 2$$

Para el modelo difuso tipo Sugeno de primer orden, el conjunto de las dos reglas difusas *si-entonces*, se define como:

Regla 1: $\boldsymbol{if}\ \ x_1\ es\ A_1^1\ \ \text{y}\ \ x_2\ es\ A_1^2\ \ \ \boldsymbol{then}\ \ f_1(x_1, x_2) = a_{11}x_1 + a_{12}x_2 + a_{10}$

Regla 2: $\boldsymbol{if}\ \ x_1\ es\ A_2^1\ \ \text{y}\ \ x_2\ es\ A_2^2\ \ \ \boldsymbol{then}\ \ f_2(x_1, x_2) = a_{21}x_1 + a_{22}x_2 + a_{20}$

Las salidas de los nodos de la capa 1 están dadas por:

$$\left[O_{1,1}^{(1)},\ O_{1,2}^{(1)},\ O_{2,1}^{(1)},\ O_{2,2}^{(1)}\right] = \left[\mu_{A_1^1}(x_1),\ \mu_{A_2^1}(x_1),\ \mu_{A_1^2}(x_2),\ \mu_{A_2^2}(x_2)\right]$$

La capa 2 se compone de neuronas difusas con un operador de agregación basado en el producto de la norma-T, por lo tanto las salidas de los nodos de esta capa están dadas por:

$$\left[O_1^{(2)},\ O_2^{(2)}\right] = \left[\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2)\ ,\ \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)\right]$$

La capa 3 es un normalizador, por tanto, las salidas de esta capa son:

$$\left[O_1^{(3)},\ O_2^{(3)}\right] = \left[\frac{O_1^{(2)}}{O_1^{(2)} + O_2^{(2)}}\ ,\frac{O_2^{(2)}}{O_1^{(2)} + O_2^{(2)}}\right]$$

$$\left[O_1^{(3)},\ O_2^{(3)}\right] = \left[\frac{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2)}{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2) + \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}\ ,\frac{\mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2) + \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}\right]$$

Los valores de las salidas de las neuronas difusas en la capa 4 son:

$$\left[O_1^{(4)},\ O_2^{(4)}\right] = \left[O_1^{(3)}.f_1\ ,\ O_2^{(3)}.f_2\right]$$

$$[O_1^{(4)},\ O_2^{(4)}] = \left[\frac{\left(\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2)\right).(a_{11}x_1 + a_{12}x_2 + a_{10})}{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2) + \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}, \frac{\left(\mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)\right).(a_{21}x_1 + a_{22}x_2 + a_{20})}{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2) + \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}\right]$$

Finalmente, la capa de salida calcula la acción de control mediante la siguiente suma:

$$y = O_1^{(4)} + O_2^{(4)}$$

$$y = \frac{\left(\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2)\right).(a_{11}x_1 + a_{12}x_2 + a_{10}) + \left(\mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)\right).(a_{21}x_1 + a_{22}x_2 + a_{20})}{\mu_{A_1^1}(x_1).\ \mu_{A_1^2}(x_2) + \mu_{A_2^1}(x_1).\mu_{A_2^2}(x_2)}$$

2.3.2 VARIANTES DE LA ARQUITECTURA ANFIS

La estructura de la red adaptativa ANFIS mostrada y estudiada hasta el momento no es única, ya que se pueden hacer otras construcciones equivalentes, así por ejemplo se pueden combinar las capas 3 y 4 para obtener una red equivalente con solo cuatro capas. De la misma forma, se puede realizar la normalización de los pesos en la última capa; la Figura 2.7 ilustra un modelo ANFIS de este tipo.

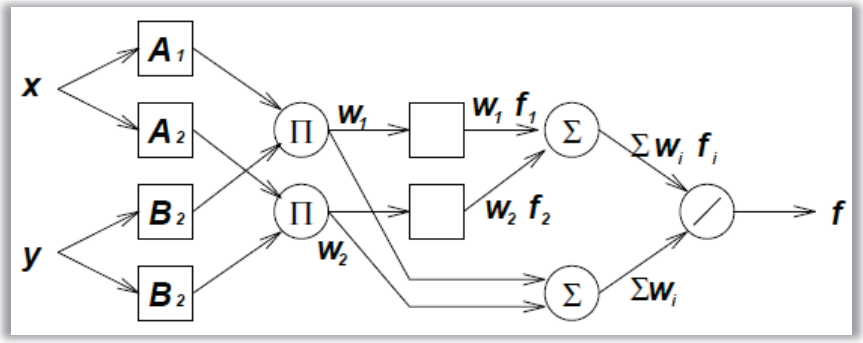


Figura 2.7. Arquitectura ANFIS para el modelo difuso Sugeno, donde la normalización de peso se realiza en la última capa [3]

En el caso extremo, incluso se puede reducir toda la red a un solo nodo de adaptación con el mismo conjunto de parámetros. Obviamente, la asignación de las funciones de nodo y la configuración de la red son arbitrarias, siempre y cuando cada nodo y cada capa realicen funcionalidades significativas y modulares (Jang et al., 1997) [2].

La extensión del modelo ANFIS tipo Sugeno al modelo ANFIS tipo Tsukamoto es sencilla, tal y como se muestra en la Figura 2.8, donde la salida de cada regla $(f_i,\ i = 1, 2)$ es inducida conjuntamente por una función de pertenencia consecuente y una intensidad de disparo (o valor de activación).

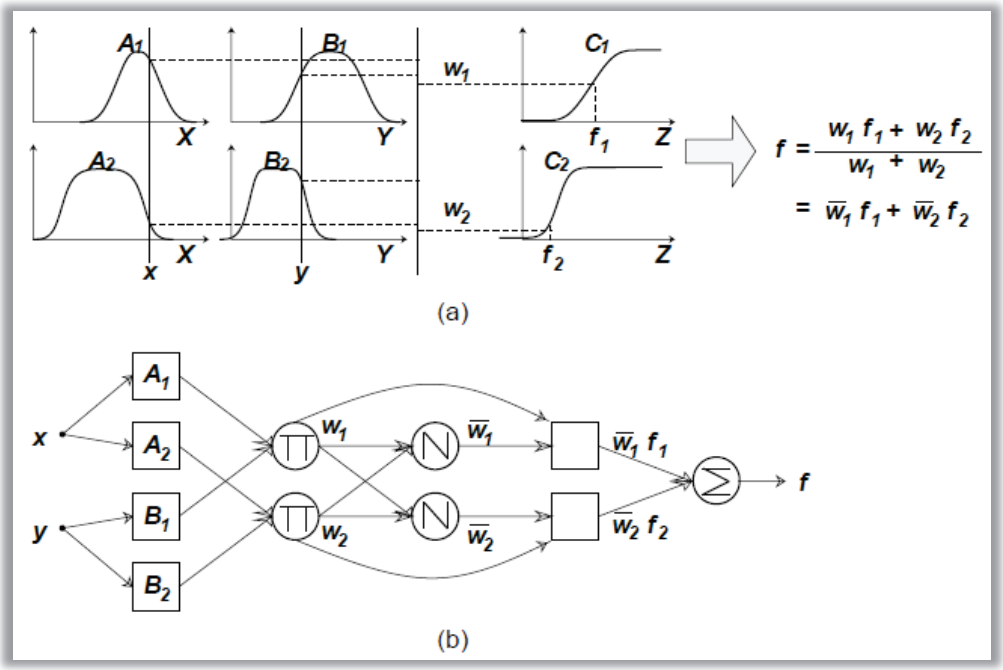


Figura 2.8. (a) Sistema de Inferencia Difuso tipo Tsukamoto con dos entradas y dos reglas difusas; (b) Arquitectura ANFIS equivalente [1]

Para el Sistema de Inferencia Difuso Mamdani con composición *max-min*, un modelo ANFIS correspondiente puede ser construido, siempre y cuando se utilicen aproximaciones discretas para reemplazar las integrales en el esquema defusificación centroide. Sin embargo, el modelo ANFIS resultante es mucho más complicado que el modelo ANFIS Sugeno o el modelo ANFIS Tsukamoto. La complejidad adicional en la estructura y el cálculo con la composición *max-min* no implica necesariamente una mejor capacidad de aprendizaje o potencia de aproximación (Jang et al., 1997) [2].

Para el desarrollo del algoritmo de programación que se implementará en este trabajo, sólo se empleará la arquitectura del modelo ANFIS tipo Sugeno de Primer Orden debido a su transparencia y eficiencia.

La Figura 2.9.a muestra una arquitectura ANFIS que es equivalente a un modelo difuso Sugeno de Primer Orden con dos entradas y nueve reglas, en donde cada entrada tiene asociada tres funciones de pertenencia MF. La Figura 2.9.b ilustra cómo está particionado el espacio de entrada en nueve regiones difusas superpuestas, en donde cada una de ellas se rige por una regla difusa *if-then (si-entonces).* En otras palabras, la parte premisa de una regla define una región difusa, mientras que la parte consecuente especifica la salida dentro de la región.

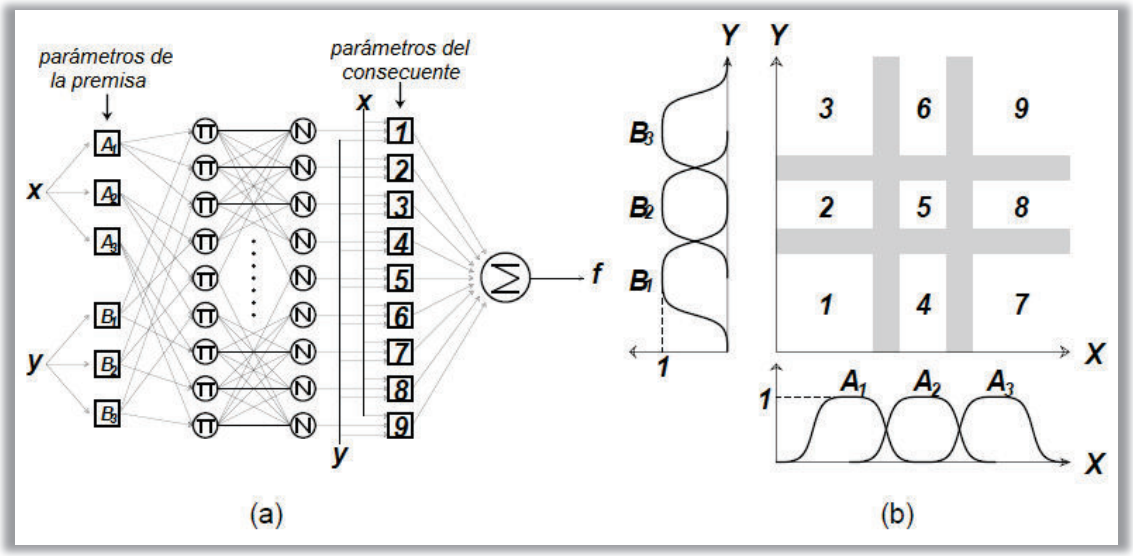


Figura 2.9. (a) Arquitectura ANFIS para un modelo difuso Sugeno con dos entradas y nueve reglas; (b) Espacio de entrada que es particionado en nueve regiones difusas [1]

A continuación se va a explicar cómo aplicar el algoritmo híbrido de aprendizaje desarrollado en la sección 1.4.4 del Capítulo 1 para identificar los parámetros del modelo neuro-difuso ANFIS.

2.3.3 ALGORITMO DE APRENDIZAJE HÍBRIDO

La representación en el apartado 2.3.1 del modelo neuro-difuso ANFIS es simplemente una representación gráfica de los pasos de cálculo en el procedimiento del Sistema de Inferencia Difuso Sugeno-Takagi. Para que esta representación sea más útil en la implementación de la ley de control, hay que dotarla de un algoritmo de aprendizaje eficiente. En las redes neuronales

convencionales, el algoritmo retropropagación se utiliza para aprender o ajustar los pesos sobre las conexiones entre las neuronas a partir de muestras de entrenamiento de entrada-salida.

En la estructura ANFIS, los parámetros de las premisas y los parámetros de los consecuentes juegan el papel de los pesos. Concretamente, cuando las funciones de pertenencia de A_j^i utilizadas en la parte *if* de las reglas son especificadas paramétricamente, es decir, se especifica la forma pero la función es determinada por un número finito de parámetros $\{a_j^i,\ b_j^i, c_j^i\}$, estos parámetros se denominan ***parámetros de premisa,*** mientras que los parámetros $\{a_{j1},\ a_{j2}, \dots, a_{jn}, a_{j0}\}$, en la parte *then* de las reglas son referidos como *parámetros del consecuente*. El algoritmo de aprendizaje híbrido para el modelo ANFIS consiste en ajustar el conjunto anterior de los parámetros a partir de una muestra de datos $[(x_1^p,\ x_2^p, \dots,\ x_n^p),\ y_d^p\ ]$ con $p = 1, 2, \dots, P$.

De la arquitectura ANFIS mostrada en la Figura 2.6 se observa que, si los valores de los parámetros de premisa son fijos, la salida global puede expresarse como una combinación lineal de los parámetros del consecuente. A partir de la ecuación 2.17, la salida y se puede rescribir como:

$$y = \overline{w_1}.(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + a_{10}) + \cdots + \overline{w_M}.(a_{M1}x_1 + a_{M2}x_2 + \cdots + a_{Mn}x_n + a_{M0})$$

$$y = (\overline{w_1}x_1)a_{11} + (\overline{w_1}x_2)a_{12} + \cdots + (\overline{w_1}x_n)a_{1n} + (\overline{w_1})a_{10} + \cdots + (\overline{w_M}x_1)a_{M1} + (\overline{w_M}x_2)a_{M2} + \cdots + (\overline{w_M}x_n)a_{Mn} + (\overline{w_M})a_{M0}$$

que es lineal en los parámetros del consecuente $a_{m1},\ a_{m2}, \dots, a_{mn}, a_{m0}$. De esta observación y a partir de lo explicado en el apartado 1.4.4, se tiene que:

$S = S_1 \oplus S_2$ → Conjunto total de parámetros

$S_1 = \{a_j^i,\ b_j^i, c_j^i\}$ → Conjunto de parámetros de premisa (no lineales)

$S_2 = \{a_{m1}, a_{m2}, \dots, a_{mn}, a_{m0}\}$ →Conjunto de parámetros del consecuente (lineales)

$H(\cdot)$ y $F(\cdot,\cdot)$ son la función identidad y la función del Sistema de Inferencia Difuso, respectivamente. Por consiguiente, el algoritmo de aprendizaje híbrido descrito en el apartado 1.4.4 se puede aplicar directamente en el ajuste de los parámetros del modelo ANFIS. Más específicamente, en el *paso en avance* del algoritmo de aprendizaje híbrido, las salidas de los nodos van hacia adelante hasta la capa 4 y los parámetros del consecuente se identifican por el *método de mínimos cuadrados recursivos* (*RLS - Recursive Least Squares*). En el *paso en retroceso*, las señales de error se propagan hacia atrás y los parámetros de la premisa se actualizan mediante el *método gradiente descendente del algoritmo de retropropagación* (Jang et al., 1997) [2]. En la Tabla 2.2 se resume las actividades en cada paso.

Tabla 2.2. Pasos en el procedimiento de aprendizaje híbrido para ANFIS

	Paso de Avance	**Paso en Retroceso**
Parámetros de premisa	Fijo	Gradiente descendente
Parámetros del consecuente	Estimador de mínimos cuadrados RLS	Fijo
Señales	Salidas de los nodos	Señales de error

Como se mencionó en la apartado 1.4.4, los parámetros del consecuente así identificados son óptimos bajo la condición de que los parámetros de premisa sean fijos. En consecuencia, el enfoque híbrido converge mucho más rápido, ya que reduce las dimensiones del espacio de búsqueda del método de retropropagación puro original. Por lo tanto, siempre se debe ver la posibilidad de descomponer el conjunto de parámetros en el primer lugar.

Como se discutió en el apartado 1.4.4.2, existen varias maneras de combinar el método del gradiente descendente con el método de mínimos cuadrados recursivos. Se puede elegir uno de estos métodos de acuerdo con los recursos computacionales disponibles y el nivel de rendimiento requerido.

2.3.3.1 Aplicación del Algoritmo de Retropropagación del Error

El algoritmo de retropropagación del error se emplea con el fin de ajustar los parámetros de la Capa 1 de la arquitectura ANFIS a través del método del gradiente descendente (según como se explicó en el apartado 1.4.3 para una red adaptativa general). Esos parámetros de premisa (también llamados parámetros no lineales) se actualizan en cada iteración (es decir, después de que se reciba cada par de entrada-salida durante el entrenamiento) con el fin de minimizar la siguiente función instantánea de error:

$$E_p(n) = \frac{1}{2}\sum_{m=1}^{M}[y_d(n) - Out^5(n)]^2 \tag{2.18}$$

donde, $y_d(n)$ es la salida deseada, $Out^5(n)$ es la salida real de la red ANFIS en el instante n y $m = 1, 2, \dots, M$ representa el número de salidas de la red, que en este en caso particular sería de $m = 1$. Por tanto la expresión a evaluar quedaría como:

$$E_p(n) = \frac{1}{2}[y_d(n) - Out^5(n)]^2 \tag{2.19}$$

Para cada par de datos de entrenamiento (entradas y salidas), la red ANFIS opera en modo de avance a fin de calcular la salida actual $Out^5(n)$. Después, a partir de la capa de salida (Capa 5) y propagándose hacia atrás, el error de retropropagación se ejecuta para calcular los derivadas $\partial E/\partial\alpha$ para cada nodo en cada Capa de la red, como se indica a continuación. Suponiendo que α es un parámetro ajustable de la red, (por ejemplo, a_j^i, b_j^i o , c_j^i que aparecen en la ecuación 2.9), entonces, este parámetro se actualiza en cada paso de tiempo por el método del gradiente descendente:

$$\Delta\alpha(n) = -\eta\frac{\partial E_p}{\partial\alpha}$$

$$\alpha(n+1) = \alpha(n) + \Delta\alpha(t) = \alpha(n) - \eta\frac{\partial E_p}{\partial\alpha} \tag{2.20}$$

donde η $(0 < \eta \leq 1)$ es la velocidad de aprendizaje de los parámetros de la red. A

continuación se explicará cómo determinar la señal de error $\delta_i^{(j)}$ correspondiente al i-*ésimo* nodo interno de la j-*ésima* capa de la red ANFIS.

Capa 5: En esta capa no existe ningún parámetro para ajustar. Sólo se tiene que calcular el error y propagarlo hacia atrás a la Capa 4:

$$\delta^{(5)} = -\frac{\partial E_p(n)}{\partial Out^5(n)} = y_d(n) - Out^5(n) \tag{2.21}$$

Capa 4: Los parámetros del consecuente en esta capa se ajustan a través del algoritmo RLS, por lo tanto, la retropropagación no juega ningún papel de ajuste aquí. En esta capa sólo se calcula una cantidad que va a ser utilizada en la siguiente capa:

$$\frac{\partial Out^5}{\partial Out_m^{(4)}} = 1, \qquad \forall\, m = 1, 2, \dots, M \tag{2.22}$$

donde, M representa el número total de variables de salida de la Capa 4.

Capa 3: De manera similar a la Capa 5, aquí sólo se tiene que calcular el error en cada nodo de esta capa:

$$\delta_k^{(3)} = -\frac{\partial E}{\partial Out_k^{(3)}} = -\sum_{m=1}^{M} \frac{\partial E_p}{\partial Out_k^{(3)}} = -\sum_{m=1}^{M} \frac{\partial E_p}{\partial Out^5} \frac{\partial Out^5}{\partial Out_m^{(4)}} \frac{\partial Out_m^{(4)}}{\partial Out_k^{(3)}}$$

Teniendo en cuenta las ecuaciones 2.21 y 2.22, la ecuación anterior puede ser simplificada como:

$$\delta_k^{(3)} = \sum_{m=1}^{M} \delta^{(5)} \frac{\partial Out_m^{(4)}}{\partial Out_k^{(3)}}, \quad k = 1, 2, \dots, \#reglas \tag{2.23}$$

A partir de la ecuación 2.15 y aplicando diferenciación se obtiene que:

$$\frac{\partial Out_m^{(4)}}{\partial Out_k^{(3)}} = a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n + a_{k0} \tag{2.24}$$

$$\frac{\partial Out_m^{(4)}}{\partial Out_k^{(3)}} = \sum_{i=1}^{n}(a_{ki}x_i) + a_{k0} = f_k \tag{2.25}$$

Capa 2:

$$\delta_{k_1}^{(2)} = -\frac{\partial E}{\partial Out_{k_1}^{(2)}} = -\sum_{k_2=1}^{\#reglas} \frac{\partial E}{\partial Out_{k_2}^{(3)}} \frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} = \sum_{k_2=1}^{\#reglas} \delta_{k_2}^{(3)} \frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} \tag{2.26}$$

A partir de la ecuación 2.14 y aplicando diferenciación se obtiene que:

$$\frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} = \begin{cases} \dfrac{\sum_{i=1}^{\#reglas}\left[Out_i^{(2)} - Out_{k_2}^{(2)}\right]}{\left[\sum_{i=1}^{\#reglas} Out_i^{(2)}\right]^2}, & cuando\ k_1 = k_2 \\ -\dfrac{Out_{k_2}^{(2)}}{\left[\sum_{i=1}^{\#reglas} Out_i^{(2)}\right]^2}, & cuando\ k_1 \neq k_2 \end{cases} \tag{2.27}$$

con $k_1, k_2 = 1, 2, \ldots, \#reglas$

Capa 1: Para esta capa se va a cambiar la nomenclatura de los parámetros a_j^i, b_j^i y c_j^i utilizadas en el apartado 2.3.1 por $a_{ij}^{(1)}$, $b_{ij}^{(1)}$ y $c_{ij}^{(1)}$, simplemente por cuestiones de interpretabilidad, donde el número 1 indica que estos parámetros pertenecen a los nodos de la Capa 1.

$$\begin{aligned} -\frac{\partial E}{\partial a_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}} \frac{\partial Out_{ij}^{(1)}}{\partial a_{ij}^{(1)}} \\ -\frac{\partial E}{\partial b_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}} \frac{\partial Out_{ij}^{(1)}}{\partial b_{ij}^{(1)}} \\ -\frac{\partial E}{\partial c_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}} \frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}} \end{aligned} \tag{2.28}$$

$$\delta_{ij}^{(1)} = -\frac{\partial E}{\partial Out_{ij}^{(1)}} = -\sum_{k=1}^{\#reglas} \frac{\partial E}{\partial Out_k^{(2)}} \frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} = \sum_{k=1}^{\#reglas} \delta_k^{(2)} \frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} \quad (2.29)$$

A partir de la ecuación 2.13 y aplicando diferenciación se obtiene que:

$$\frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} = \begin{cases} \frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}}, si\ y\ solo\ si\ Out_{ij}^{(1)}\ está\ conectada\ al\ k-ésimo\ nodo \\ 0 \qquad\qquad de\ otra\ manera \end{cases} \quad (2.30)$$

Para los parámetros de premisa de las reglas difusas y a partir de la ecuación 2.9 se obtienen las siguientes expresiones:

$$\frac{\partial Out_{ij}^{(1)}}{\partial a_{ij}^{(1)}} = \frac{2b_{ij}^{(1)}}{a_{ij}^{(1)}} Out_{ij}^{(1)}\left(1 - Out_{ij}^{(1)}\right)$$

$$\frac{\partial Out_{ij}^{(1)}}{\partial b_{ij}^{(1)}} = \begin{cases} -2\ ln\left|\frac{x_i - c_{ij}^{(1)}}{a_{ij}^{(1)}}\right| Out_{ij}^{(1)}\left(1 - Out_{ij}^{(1)}\right), & si\ y\ solo\ si\ x_i \neq c_{ij}^{(1)} \\ 0 & si\ y\ solo\ si\ x_i = c_{ij}^{(1)} \end{cases} \quad (2.31)$$

$$\frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}} = \begin{cases} \frac{2b_{ij}^{(1)}}{x_i - c_{ij}^{(1)}}\ Out_{ij}^{(1)}\left(1 - Out_{ij}^{(1)}\right), & si\ y\ solo\ si\ x_i \neq c_{ij}^{(1)} \\ 0 & si\ y\ solo\ si\ x_i = c_{ij}^{(1)} \end{cases}$$

Con $i = 1, 2, \ldots, \#Variables\ de\ entradas$ y $j = 1, 2, \ldots, \#Terminos\ linguísticos$

En caso de que se emplee la función de pertenencia en forma de campana (ecuación 2.10) las relaciones correspondientes son:

$$\frac{\partial Out_{ij}^{(1)}}{\partial \sigma_{ij}^{(1)}} = \frac{\left(x_i - c_{ij}^{(1)}\right)^2}{\left(\sigma_{ij}^{(1)}\right)^3} Out_{ij}^{(1)}$$

$$\frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}} = \frac{x_i - c_{ij}^{(1)}}{\left(\sigma_{ij}^{(1)}\right)^2} Out_{ij}^{(1)} \quad (2.32)$$

Y finalmente a partir de la ecuación 2.20 se obtiene:

$$a_{ij}^{(1)}(n+1) = a_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial a_{ij}^{(1)}}\right)$$
$$b_{ij}^{(1)}(n+1) = b_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial b_{ij}^{(1)}}\right) \quad (2.32)$$
$$c_{ij}^{(1)}(n+1) = c_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial c_{ij}^{(1)}}\right)$$

Con $i = 1, 2, \dots, \#Variables\ de\ entradas$ y $j = 1, 2, \dots, \#Terminos\ linguísticos$

2.3.4 ALGORTIMO DE PROGRAMACIÓN DE LA RED ANFIS

Como ya se ha explicado en los apartados anteriores, la red híbrida ANFIS utiliza cinco capas consecutivas y se aprovecha de 2 algoritmos de entrenamiento: la retropropagación de error y el RLS. Toda la experiencia adquirida durante el entrenamiento se almacena en 2 familias de parámetros, estos son los parámetros de la Capa 1 (parámetros no lineales o de premisa) y los parámetros de la Capa 4 (parámetros lineales o del consecuente). Los parámetros de premisa se sintonizan a través del algoritmo estándar de retropropagación de error, mientras que los parámetros del consecuente a través del algoritmo RLS clásico.

El algoritmo ANFIS tipo rejilla (Grid) se ha programado e implementado utilizando las herramientas de Matlab y Simulink. La lógica de programación del algoritmo ha sido desarrollada completamente mediante sintaxis de código Matlab mientras que la fase de verificación del funcionamiento del mismo ha sido implementada en el entorno gráfico de Simulink. Ambos entornos se comunican entre sí mediante un bloque especial conocido como S-Function; la explicación del funcionamiento, creación y configuración de este bloque esta detallada en el Anexo 1.

Existen seis tipos de tareas que realiza un S-Function y cada una de las cuales se designa por un número entero asignado a una variable conocida como $flag$; depende del algoritmo a implementar sobre este bloque para determinar cuáles

tareas serán utilizadas. El programa desarrollado en Matlab emplea esta variable $flag$ para determinar cómo distribuir el código de programación.

En las Figuras 2.10 – 2.15 se presentan los diagramas de flujos generales que determinan el funcionamiento del programa desarrollado en Matlab y en donde cada sub-algoritmo está en base a los posibles valores que puede tomar la variable $flag$ dentro del algoritmo ANFIS.

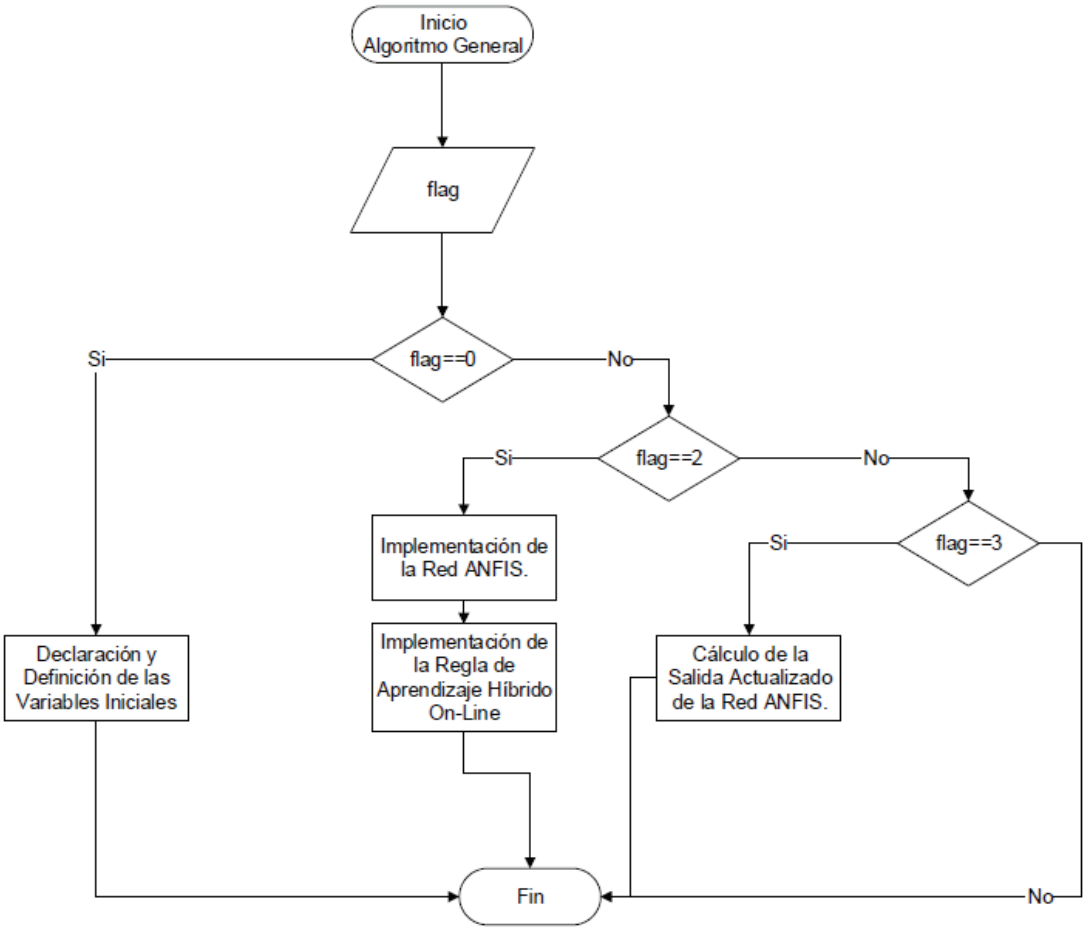


Figura 2.10. Diagrama de flujo general del algoritmo ANFIS

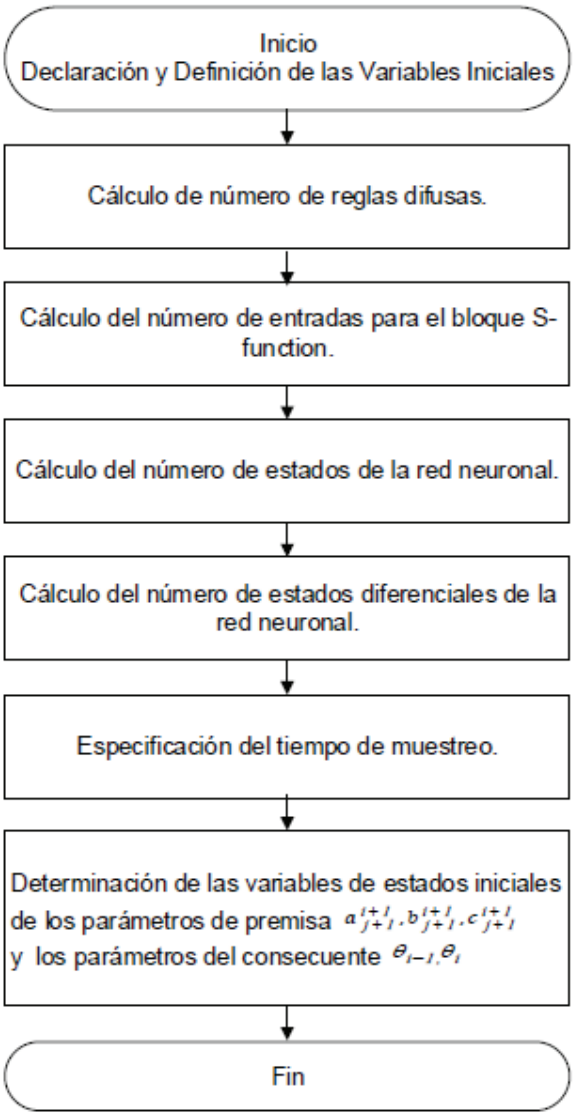


Figura 2.11. Diagrama de flujo del sub-algoritmo Declaración y Definición de las Variables Iniciales

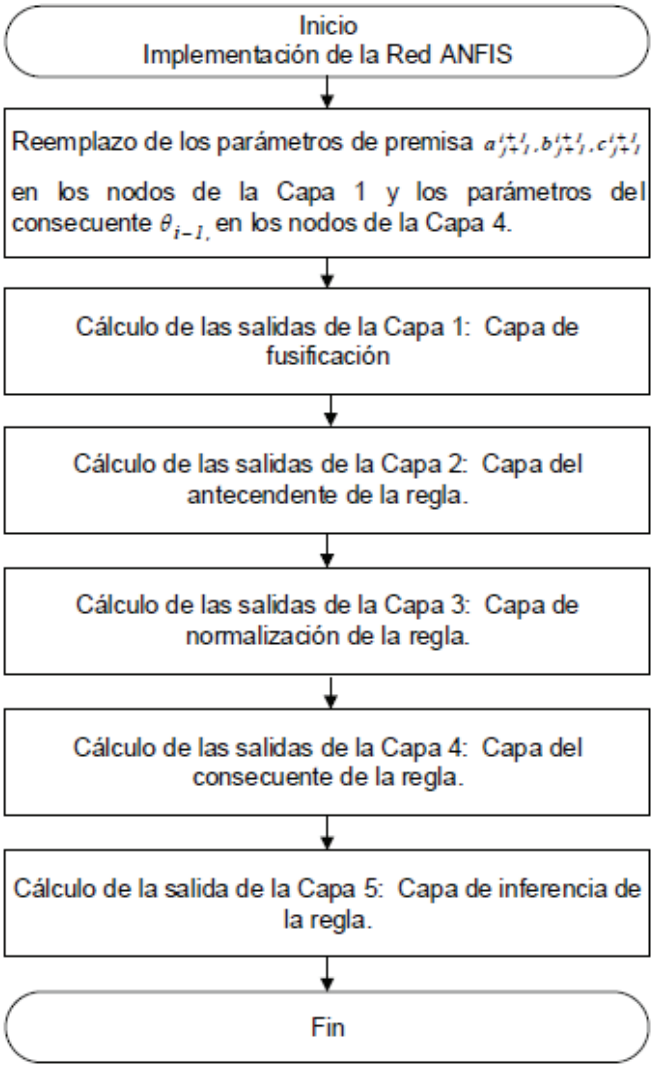


Figura 2.12. Diagrama de flujo del sub-algoritmo Implementación de la Red ANFIS

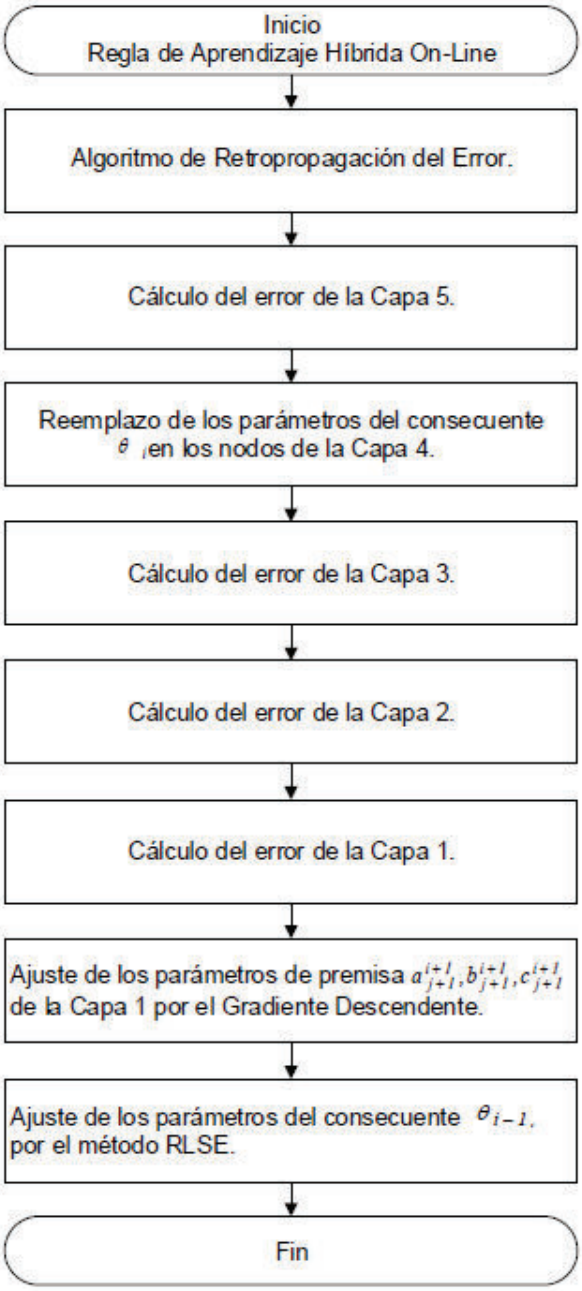


Figura 2.13. Diagrama de flujo del sub-algoritmo Implementación del Algoritmo de Aprendizaje Híbrido On-Line

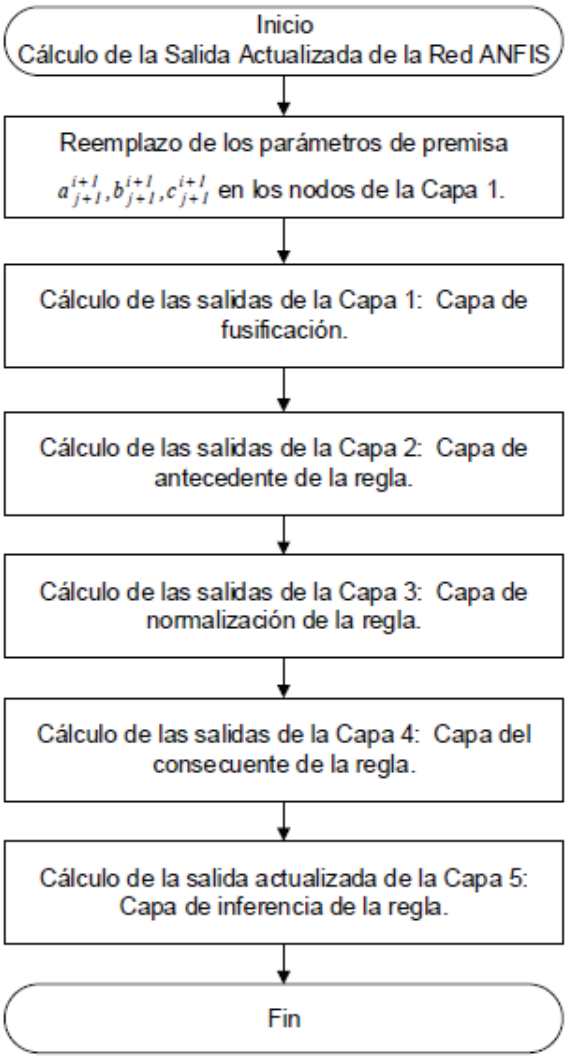


Figura 2.14. Diagrama de flujo del sub-algoritmo Cálculo de la Salida Actualizada de la Red ANFIS

El programa elaborado en Matlab (*anfisim_grid.m*) y que representa el funcionamiento del algoritmo ANFIS está diseñado en base a los diagramas de flujos antes indicados. El código completo puede ser revisado en el Anexo 3, en donde también se explica la lógica de programación mediante comentarios.

Una vez desarrollado el código de programación en Matlab se procedió a enlazarlo con el bloque S-Function de Simulink; el detalle de los pasos para realizar este proceso puede ser consultado en el Anexo 1. El bloque final que se

obtuvo y que representa a la red ANFIS es el mostrado en la Figura 2.15.

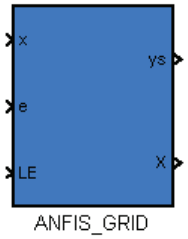


Figura 2.15. Bloque S-Function del algoritmo ANFIS tipo rejilla

La entrada x corresponde a un vector que recibe y contiene las entradas presentes en el sistema ANFIS. Si se necesita suministrar al bloque más de una entrada se debe utilizar un multiplexor.

La entrada e es donde se debe ingresar la señal de error de entrenamiento y que en una red ANFIS es un valor escalar que resulta de la diferencia de la señal de salida deseada y la señal de salida real ys.

La entrada LE es una variable de dos estados discretos y permite habilitar (LE=1) o desactivar (LE=0) el aprendizaje de la red ANFIS. Cuando LE=1 el entrenamiento de la red se lleva a cabo y después de varias iteraciones se puede cambiar el estado a LE=0 para comprobar la operación de la aproximación basada en la experiencia recogida por la formación previa.

El bloque ANFIS_GRID de la Figura 2.15 ha sido diseñado para que pueda desplegar una ventana de configuración como la mostrada en la Figura 2.16, en donde se pueden modificar varios de los parámetros que se utilizan para especificar la configuración física de la red ANFIS y variar ciertas constantes de entrenamiento empleadas en el algoritmo de aprendizaje. El significado de cada recuadro es fácil de identificar gracias a la descripción provista en la parte superior del mismo.

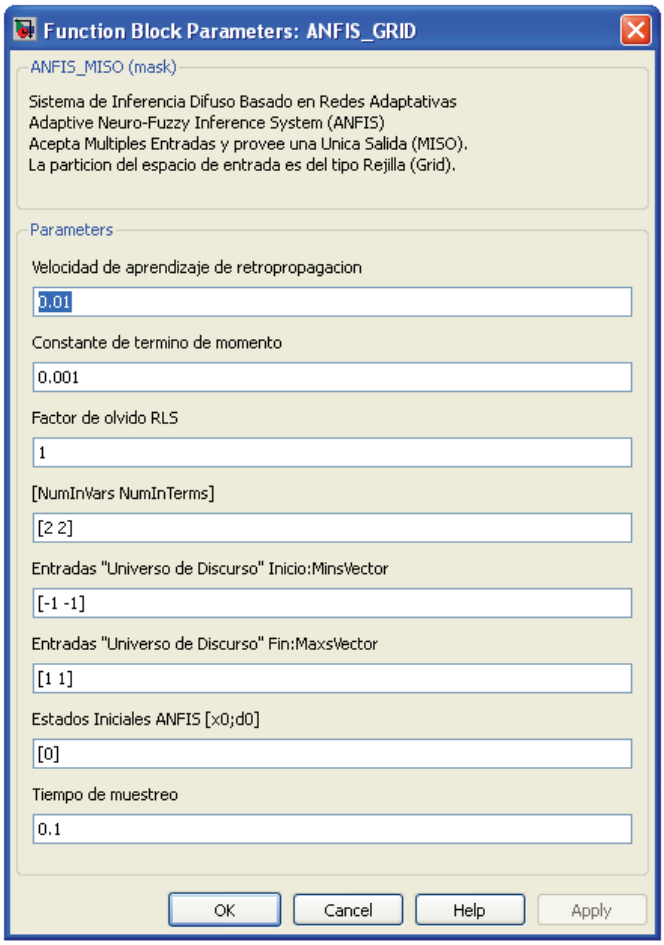


Figura 2.16. Ventana de configuración del bloque ANFIS_GRID

En la casilla etiquetada como “Estados Iniciales ANFIS [x0;d0]” se puede colocar el vector nulo indicando que todos los parámetros de la Capa 1 y la Capa 4 se inicializan en cero o se puede colocar el nombre de algún archivo que contiene un vector que almacena los parámetros de estas capas y cuyos valores la red ANFIS aprendió en algún entrenamiento previo. Todos estos valores serán posteriormente transferidos a la función *anfisim_grid.m* implementada en el programa desarrollado en Matlab y que tiene la siguiente forma:

```
function [out,Xt,str,ts] =
anfisim_grid(Ti,Xt,u,flag,Ita,alpha,lamda,NumInVars,NumInTerms,x0,T
)
```

El significado de cada una de los argumentos de entrada de la función *anfisim_grid.m* es detallado en la Tabla 2.3.

Tabla 2.3. Argumentos de Entrada de la Función anfisim_grid.m

Argumentos de Entrada de la Función anfisim_grid.m	
Nombre	**Descripción**
Ti	Tiempo de simulación actual
Xt	Vector columna correspondiente a las variables de estados de la red ANFIS.
u	Vector columna correspondiente a las variables de entrada de la red ANFIS.
Flag	Valor entero que indica la tarea a ser realizada por la S-function
Ita	Corresponde a la constante que representa la tasa o velocidad de aprendizaje (η) usada en el algoritmo de retropropagación para ajustar los parámetros no lineales de Capa 1.
alpha	Corresponde a la constante que representa el término de momento (α) usada en el algoritmo de retropropagación para ajustar los parámetros no lineales de Capa 1.
lamda	Este es el factor de olvido (λ) asociado con el algoritmo del estimador de mínimos cuadrados recursivos (RLSE) que se utiliza para ajustar los parámetros lineales de la Capa 4.
NumInVars	Número de variables de entrada de la red ANFIS.
NumInTerms	Número de subconjuntos difusos que dividen el dominio de cada variable de entrada, es decir, el número de variables lingüísticas ubicadas en la Capa 1. Este número es común para cada variable de entrada.
x0	Vector que almacena los estados iniciales de los parámetros no lineales de la Capa 1 y los parámetros lineales de la Capa 4.
T	Tiempo de muestreo

2.4 SIMULACIÓN DE LA RED ANFIS

Una vez determinado el modelo matemático de la planta TRMS e implementado el sistema ANFIS en el entorno Simulink, lo que resta es verificar la funcionalidad del programa desarrollado. Lo primero que se realizó es la traducción del diagrama de bloques de la Figura 2.3 a programación gráfica del entorno Simulink, utilizando como guía el diseño que viene en los ejemplos instalados del fabricante

de la planta TRMS (Feedback Instruments Limited), tal y como se muestra en la Figura 2.17.

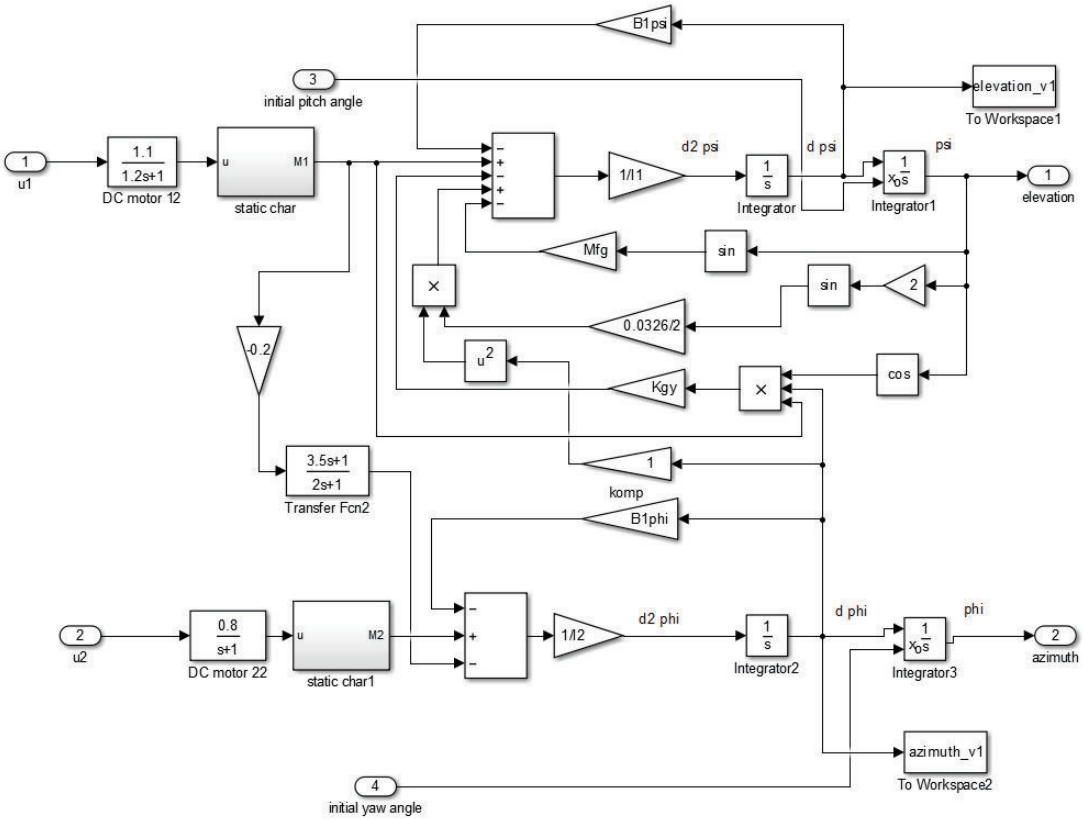


Figura 2.17. Modelo matemático de la planta TRMS utilizando Simulink

Todo el diagrama esquemático mostrado en la Figura 2.17 fue posteriormente encapsulado en único bloque como se muestra en la Figura 2.18, con la finalidad de reducir el espacio de simulación del entorno y poder ubicar otros elementos que interactúen con la planta.

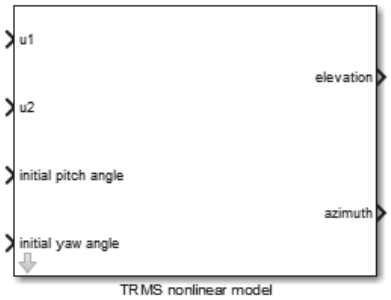


Figura 2.18. Modelo matemático encapsulado de la planta

Considerando que la red ANFIS es un tipo de red neuronal adaptativa, esta puede ser utilizada para la identificación y control de sistemas basados en los esquemas discutidos en el apartado 1.4.5 del Capítulo 1. Entonces como ya se cuenta con el modelo matemático de la planta, se procedió a utilizar la red ANFIS para que aprenda y replique el comportamiento del sistema TRMS basados en los esquemas de la Figura 1.15, en la cual se pueden observar las configuraciones gráficas para la identificación hacia adelante y la identificación directa inversa de la planta.

Se debe considerar que el TRMS es un sistema MIMO con dos entradas (u_1 y u_2) y dos salidas (φ y ψ) de tal manera que puede girar libremente al mismo tiempo en el plano vertical y horizontal (pitch y yaw). La red ANFIS es un sistema MISO que acepta varias entradas para aprender el comportamiento de una única salida, por dicha razón si se requiere aprender el comportamiento completo del TRMS se debería utilizar dos redes ANFIS para cada salida cuando se tratase de la identificación hacia adelante y la misma cantidad de redes ANFIS para el caso de la identificación directa inversa. Tomando en consideración lo antes manifestado, se va realizar a modo de ejemplo la identificación de la salida φ (levation-pitch) utilizando el método hacia adelante y la identificación de la entrada u_1 empleando el método directo inverso. En la Figura 2.19 se muestran las configuraciones para la identificación de la planta TRMS utilizando redes ANFIS según las consideraciones previas.

Como se observa en la Figura 2.19 se ha utilizado bloques generadores de señales como elementos de entrada a la planta y señales tipo escalón, que pueden ser alternadas o combinadas según los requerimientos. Cuando la identificación se realiza hacia adelante la red ANFIS utiliza como entrada las variables u_1, u_2 y φ, y es entrenada para que aprenda el comportamiento del movimiento en el plano vertical φ del TRMS.

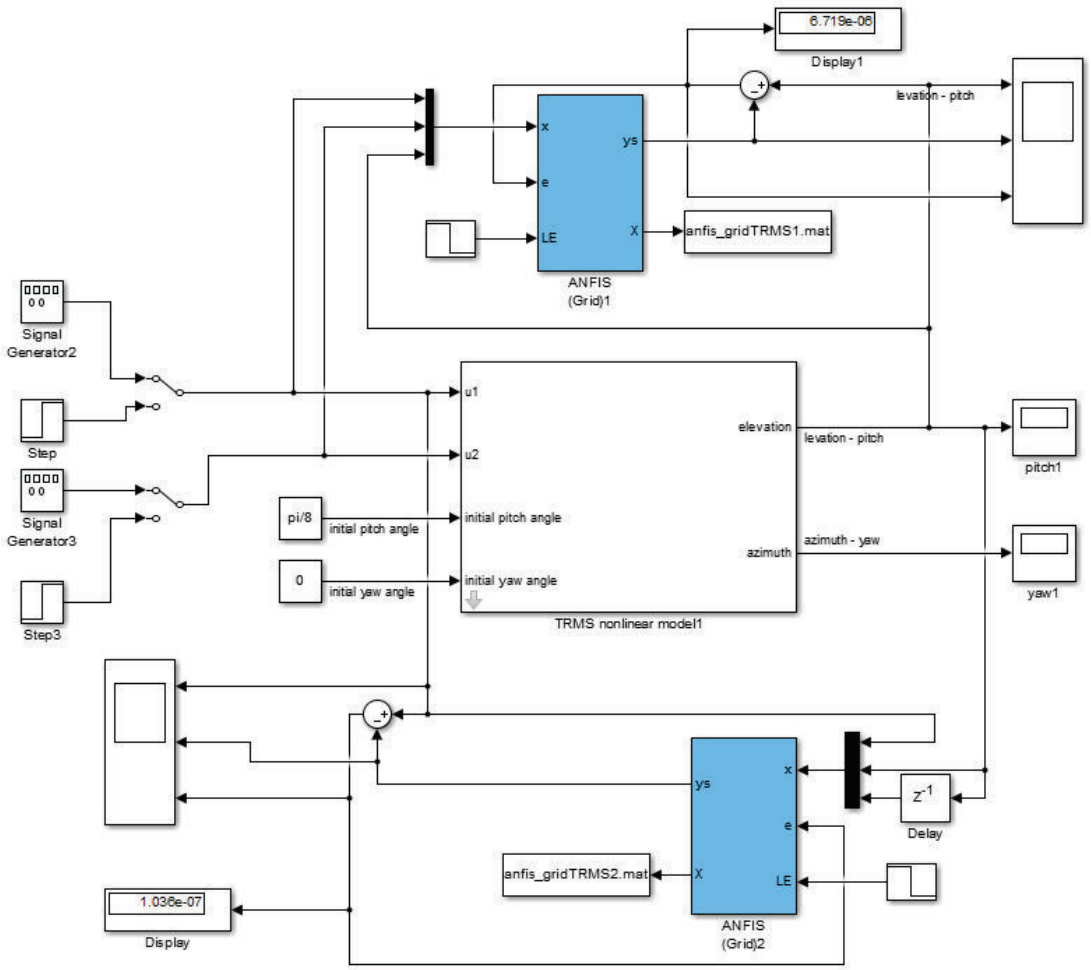


Figura 2.19. Identificación de la planta TRMS mediante redes ANFIS en modo hacia adelante y en modo directo inverso

En la Figura 2.20 se muestra la configuración de todos los parámetros de la primera red ANFIS, los cuales pueden ser modificados hasta obtener un error aceptable en el proceso de aprendizaje. Los valores de lta (η), alpha (α) y lamda (λ) han sido elegidos según las consideraciones prácticas explicadas en el apartado 1.4 del Capítulo 1. En el cuarto recuadro aparece el número 3 que indica el número de entradas a la red ANFIS y el número 2 que indica el número de nodos de la Capa 1 o subconjuntos difusos que dividen el dominio de cada variable de entrada, este valor es factible de cambiar, pero tenga en cuenta que a la red le llevaría más tiempo realizar el proceso de aprendizaje.

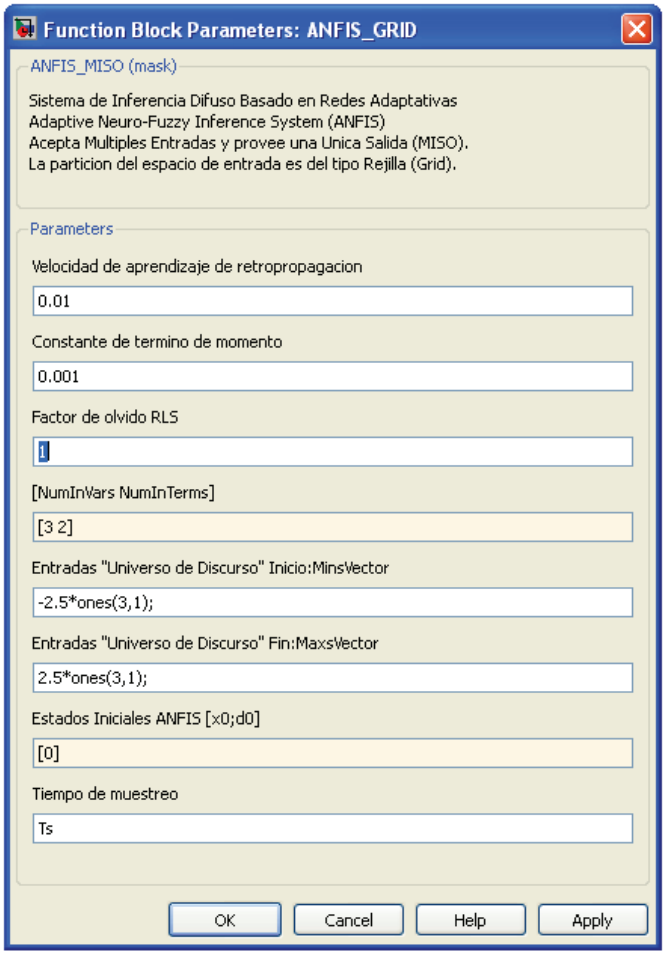


Figura 2.20. Ventana de configuración de la red ANFIS (Grid).

En el quinto y sexto recuadro se colocan dos vectores de tres elementos cada uno con valores de -2.5 y 2.5 respectivamente, los cuales indican los rangos mínimos y máximos (universo de discurso) en los que pueden variar las tres señales de entrada; estos valores han sido elegidos tomando consideración que este es el rango de voltajes de entrada de los motores que controlan los dos movimientos del sistema TRMS. En el antepenúltimo recuadro se coloca un vector nulo que indica que la red va empezar a entrenarse sin tener un conocimiento previo de los parámetros de premisa (Capa 1) ni los parámetros del consecuente (Capa 4). En el último recuadro se coloca la variable Ts que corresponde al tiempo de muestro y puede ser definida en la ventana de comandos de Matlab.

Cuando la identificación es directa inversa la red ANFIS utiliza como entrada las

variables u_1, φ y φ retrasada, y es entrenada para que aprenda el comportamiento de la entrada u_1 presentada al sistema TRMS. Los parámetros de la segunda red ANFIS han sido configurados con los mismos valores que la primera red (Figura 2.20) y como se mencionó anteriormente son factibles de modificar hasta obtener un error aceptable en el proceso de aprendizaje.

En las entradas LE (habilitación de aprendizaje) de ambas redes ANFIS se han colocado dos bloques de señales tipo escalón las cuales pueden tomar únicamente dos valores discretos (1 o 0) según se muestra en la ventana de configuración de la Figura 2.21.

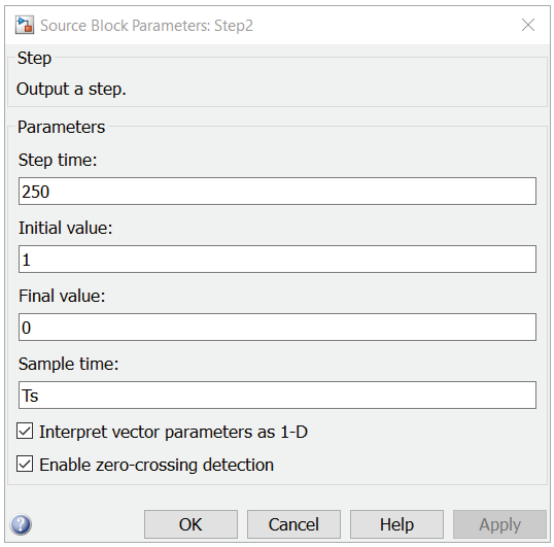


Figura 2.21. Ventana de configuración del bloque de la señal escalón

En las primeras 250 iteraciones LE=1 lo que pone en manifiesto que la red se está entrenando, es decir, que se están actualizando los parámetros de la Capa 1 y la Capa 4. En las siguientes 250 iteraciones LE=0 lo que indica que la red ANFIS ya no está en modo de aprendizaje sino en modo de prueba, puesto que los parámetros de premisa y del consecuente se mantienen fijos.

A la salida X de cada red ANFIS se han colocado dos bloques *To File* etiquetados como *anfis_gridTRMS1.mat* y *anfis_gridTRMS2.mat*, los cuales permiten almacenar en un arreglo los parámetros de premisa (Capa 1) y los parámetros del

consecuente (Capa 4) que la red ANFIS ha obtenido después de haber aprendido el comportamiento de las señales de salida y entrada de la planta TRMS respectivamente. Estos arreglos pueden ser posteriormente utilizados en un sistema de control junto con el TRMS, con la finalidad de obviar el paso de aprendizaje y directamente establecer los parámetros de control.

Una vez explicado los elementos que conforman el sistema de identificación de la Figura 2.19 se va a proceder a ejecutar la simulación y visualizar las señales de salida de la red ANFIS junto a las señales reales del sistema TRMS. Primero se van a emplear señales tipo seno como señales de entrada (u_1 y u_2), para lo cual se deben realizar las configuraciones indicadas en la Figura 2.22.

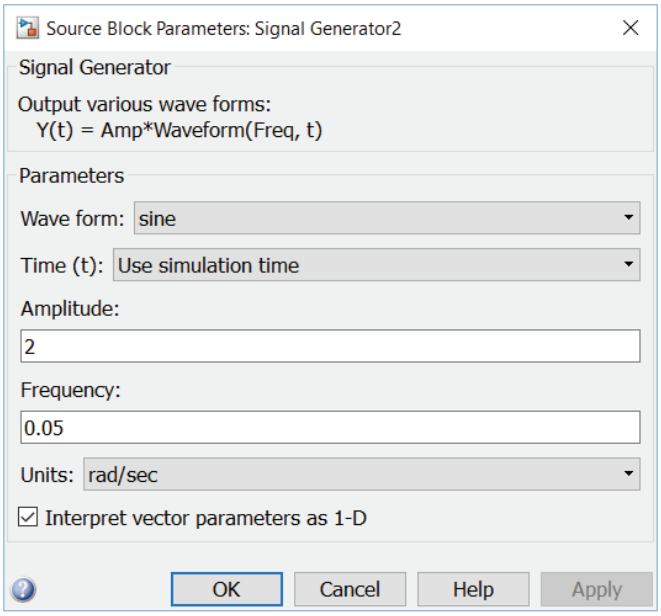


Figura 2.22. Ventana de configuración del bloque generador de señales

Tras ejecutar y finalizar la simulación se obtuvieron las señales mostradas en la Figura 2.23 y en la Figura 2.24.

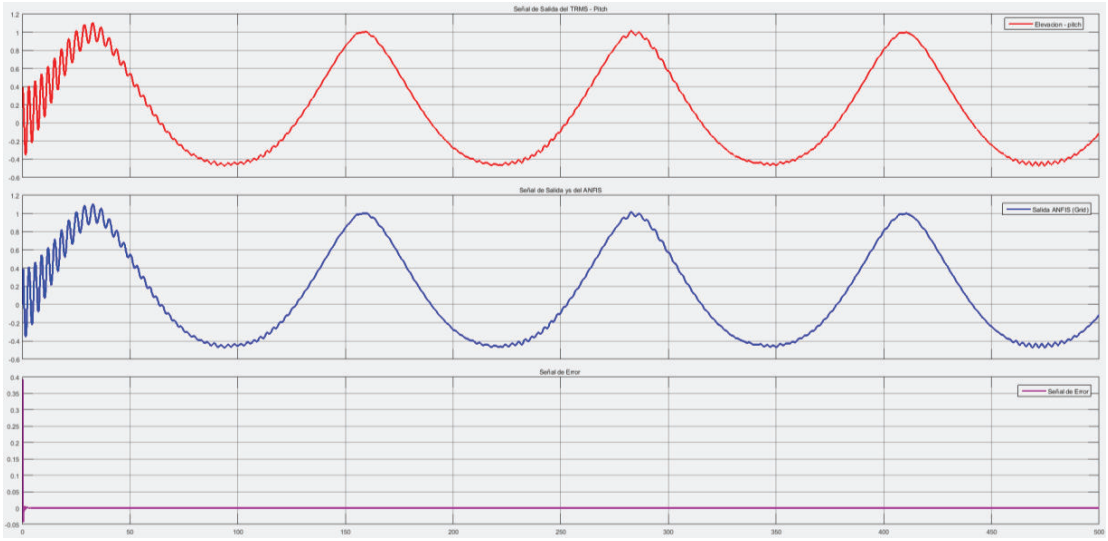


Figura 2.23. Señales resultantes del proceso de identificación hacia adelante

En la Figura 2.23 se pueden observar la señal de salida φ (en rojo) correspondiente movimiento en el plano vertical del TRMS, la señal de salida ys (en azul) de la red ANFIS y la señal de error (en violeta). Las 250 primeras iteraciones corresponden al proceso de aprendizaje de red ANFIS mientras los 250 restantes se utilizan en el proceso de comprobación para verificar que la red ANFIS aprendió el comportamiento de la señal de salida φ de la planta TRMS. Es evidente que la velocidad de aprendizaje implementada sobre el algoritmo híbrido está acorde a los resultados obtenidos y que se podría disminuir el número de iteraciones en el proceso de aprendizaje. El mayor error que se obtuvo fue en la primera iteración de aprendizaje que fue de 0.4 y que en las sucesivas iteraciones fue decayendo a valores mucho más pequeños. En el proceso de comprobación se obtuvo como error final un valor de $6.72x10^{-8}$ (ver Figura 2.19), que en términos de identificación de sistemas se puede considerar como insignificante y por tanto se puede asumir que el sistema aproximado es una fiel copia del sistema real.

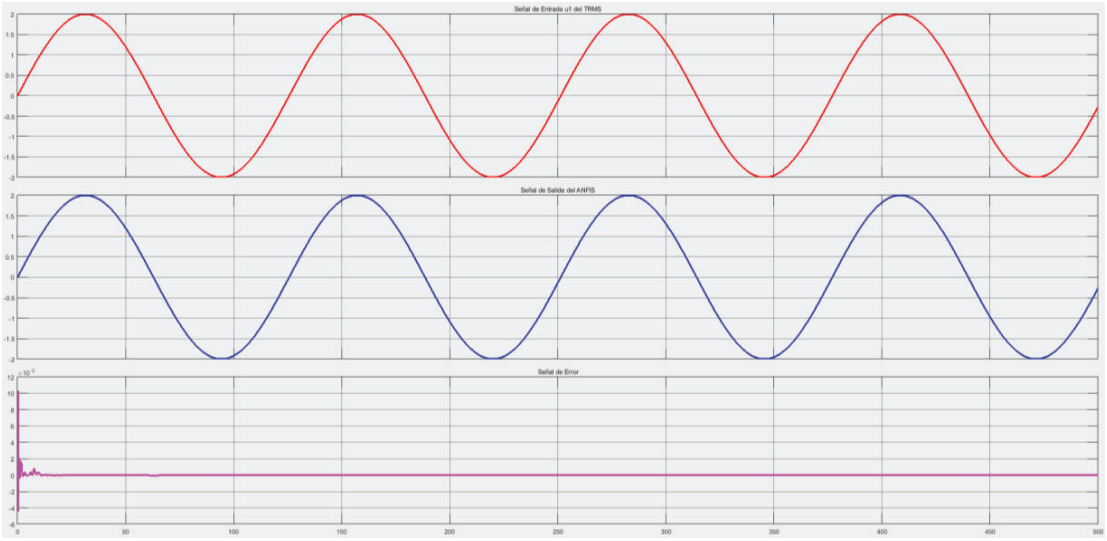


Figura 2.24. Señales resultantes del proceso de identificación directo inverso

En la Figura 2.24 se pueden observar la señal de entrada u1 (en rojo) que ingresa a la planta TRMS, la señal de salida ys (en azul) de la red ANFIS y la señal de error (en violeta). Las 250 primeras iteraciones corresponden al proceso de aprendizaje de red ANFIS mientras los 250 restantes se utilizan en el proceso de comprobación para verificar que la red ANFIS aprendió el comportamiento de la señal de entrada u1 de la planta TRMS. Al igual que en el proceso de identificación hacia adelante es notorio que la velocidad de aprendizaje implementada sobre el algoritmo híbrido está acorde a los resultados obtenidos y que se podría disminuir el número de iteraciones en el proceso de aprendizaje. El mayor error que se obtuvo fue en la primera iteración de aprendizaje que fue de $1x10^{-2}$ y que en las sucesivas iteraciones fue decayendo a valores mucho más pequeños. En el proceso de comprobación se obtuvo como error final un valor de $1.03x10^{-7}$ (ver Figura 2.19), que en términos de identificación de sistemas se puede considerar como insignificante y por tanto se puede asumir que el sistema aproximado es una fiel copia del sistema real.

Como se ha podido comprobar mediante la simulación en el entorno Simulink de Matlab, en ambos procesos de identificación el empleo de la red ANFIS ha resultado satisfactorio. En los dos ejemplos mostrados en la Figura 2.23 y la Figura 2.24 se ha partido de una señal de entrada tipo senoidal, pero es

aconsejable utilizar señales de entrada tipo aleatorias para tener un rango de señales más variado. El motivo de que no se haya utilizado en este apartado este tipo de señales es netamente por cuestiones de visualización, puesto que no es tan fácil de divisar y comparar la señal real de la planta TRMS con la señal aproximada por la red ANFIS. En la fase de implementación del controlador ANFIS sobre la planta física TRMS que será detallado en el Capítulo 3 se tomará en cuenta esta consideración.

CAPÍTULO 3
EVALUACIÓN DE DESEMPEÑO DEL CONTROLADOR ANFIS

3.1 INTRODUCCIÓN

En este capítulo, se detallará el procedimiento de implementación de la red ANFIS sobre el sistema físico TRMS, para que actúe como un controlador partiendo de señales de referencia o consigna. Luego se verificará la validez del controlador ANFIS que posteriormente será comparado con un controlador PID clásico que ya viene incorporado y predefinido en los ejemplos proporcionados por el fabricante de la planta TRMS.

Finalmente se realizará un análisis comparativo de desempeño de ambos controladores empleando un indicador de rendimiento conocido como integral del error absoluto (IAE) el cual será utilizado como dato de entrada a un método estadístico conocido como el Test de Wilcoxon que permitirá determinar si el controlador ANFIS supera los niveles de aceptación dentro de este proceso de control.

3.2 IMPLEMENTACIÓN DEL CONTROLADOR ANFIS SOBRE EL SISTEMA TRMS

Como se demostró en el apartado 2.4 del Capítulo 2 la red ANFIS puede ser empleada en el proceso de identificación de un sistema ya sea empleando el modo hacia adelante o el modo directo inverso. Otro de los usos que se les suele dar a las redes neuronales es en el diseño de controladores, puesto que la red ANFIS es un tipo en particular de red adaptativa y se enmarca muy bien dentro de este contexto. En esta sección se va implementar sobre la planta física TRMS (ver Figura 3.1) un Controlador Directo Inverso utilizando una red ANFIS según se explicó en el apartado 1.4.5.2 del Capítulo 1. La planta TRMS está conectada al

ordenador mediante una tarjeta PCI-1711 que permite establecer la comunicación y los procesos de control sobre el sistema.

Figura 3.1. Sistema físico Twin Rotor MIMO System – TRMS

Al finalizar el Capítulo 2 se mencionó que es aconsejable utilizar sobre la planta TRMS una señal de entrada u1 tipo aleatoria debido a que se contaría con un rango de operación más variado y para cualquier otro tipo de señal de entrada que se ingresara posteriormente a la red ANFIS esta respondería adecuadamente. El controlador directo inverso vasa su funcionamiento asumiendo que se ha determinado con anterioridad el modelo directo inverso de la planta TRMS; por dicha razón antes de implementar el controlador se requiere realizar el proceso de identificación tal y como se explicó en el apartado 2.4 del Capítulo 2, pero con una señal de entrada u1 aleatoria que varíe entre 2 y -2 (ver Figura 3.2). En este caso se va a disminuir a 50 el número de iteraciones (ver Figura 3.3) en el proceso de aprendizaje para comprobar y corroborar las afirmaciones emitidas en el Capítulo 2, en cuanto que la red ANFIS no necesita realizar muchas iteraciones sobre la planta TRMS para conseguir modelar el sistema inverso planteado.

Los parámetros de configuración de la red ANFIS fueron establecidos con los valores mostrados en la Figura 3.4. Nótese también que en esta ventana se está

especificando que la red ANFIS iniciará su proceso de aprendizaje asumiendo que no se tiene un conocimiento previo de los parámetros de premisa y los parámetros del consecuente.

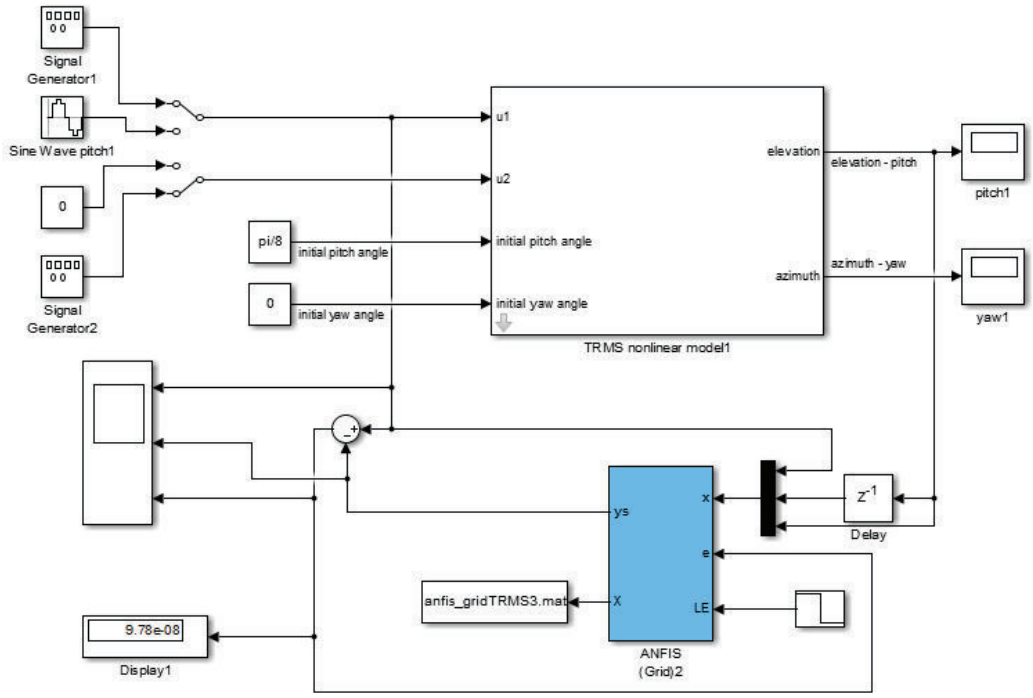


Figura 3.2. Identificación de la planta TRMS mediante redes ANFIS en modo directo inverso

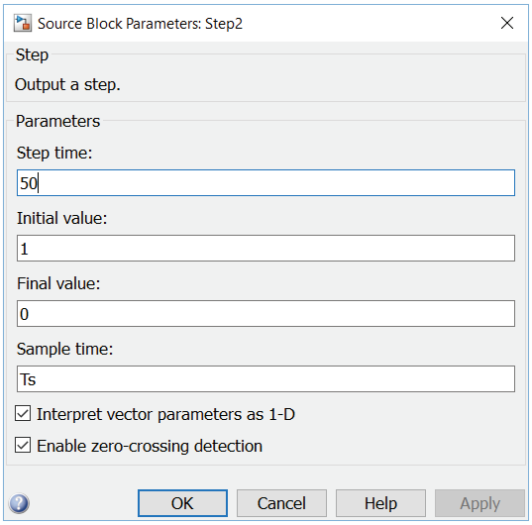


Figura 3.3. Ventana de configuración de la señal escalón que ingresa a la entrada LE de la red ANFIS

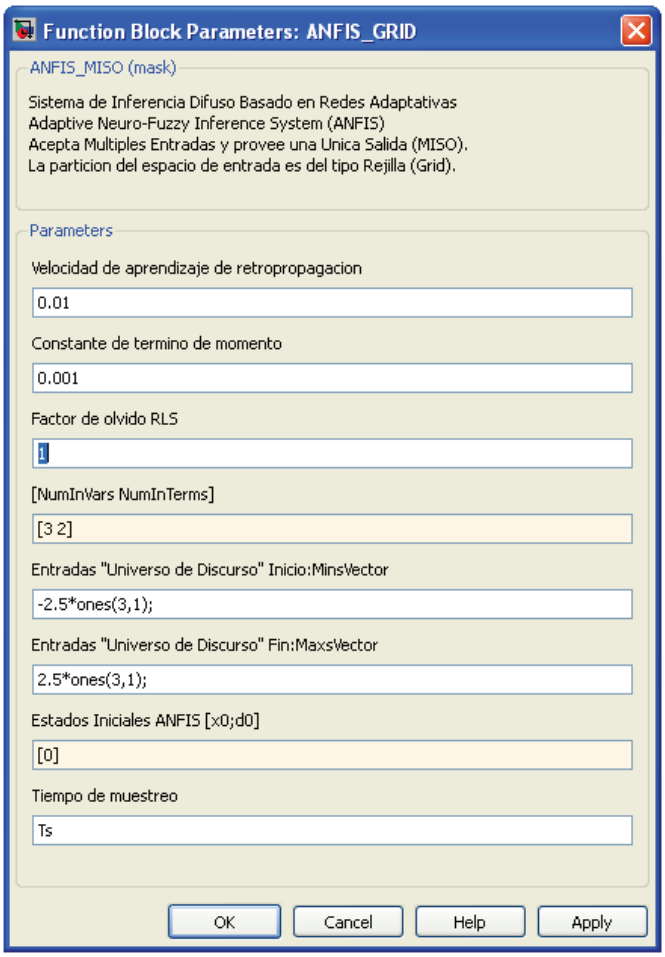


Figura 3.4. Ventana de configuración de la red ANFIS para el proceso de identificación

Una vez que ha finalizado la simulación del proceso de identificación en modo directo inverso se obtuvieron las señales mostradas en la Figura 3.5. La primera de ellas (color rojo) corresponde a la señal aleatoria de entrada u1, la segunda (color azul) es la señal de salida ys de la red ANFIS y la última (color violeta) corresponde a la señal de error obtenida de la diferencia de las dos señales anteriores. Como se puede observar tanto gráficamente en la Figura 3.5 como cuantitativamente en la Figura 3.2 el error obtenido es bien pequeño y su valor final es de $9.78x10^{-8}$, por lo cual, se puede afirmar que la red ANFIS ha aprendido lo más fielmente el comportamiento de la señal de entrada u1.

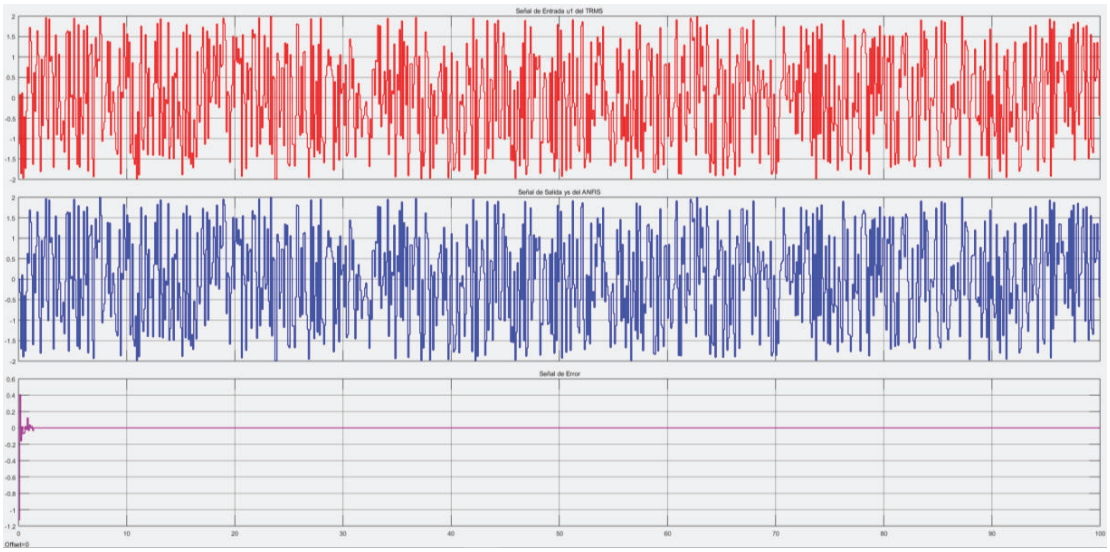


Figura 3.5. Señales resultantes del proceso de identificación directo inverso

La red ANFIS entrenada en modo inverso directo según se observa en la Figura 3.2 ahora será empleada como controlador directo inverso, es decir, pasa de ser un identificador ANFIS para convertirse en un controlador ANFIS según los criterios ya explicados en el apartado 1.4.5.2 del Capítulo 1. En una primera instancia se va a simular el comportamiento del controlador ANFIS junto con el modelo de la planta TRMS en el entorno Simulink, para posteriormente implementar dicho controlador en el mismo entorno de simulación pero conectado a la planta física mediante el ordenador y una tarjeta controladora. En la Figura 3.6 se muestra la configuración y los elementos necesarios para que la red ANFIS previamente entrenada forme parte del sistema de control de la planta TRMS.

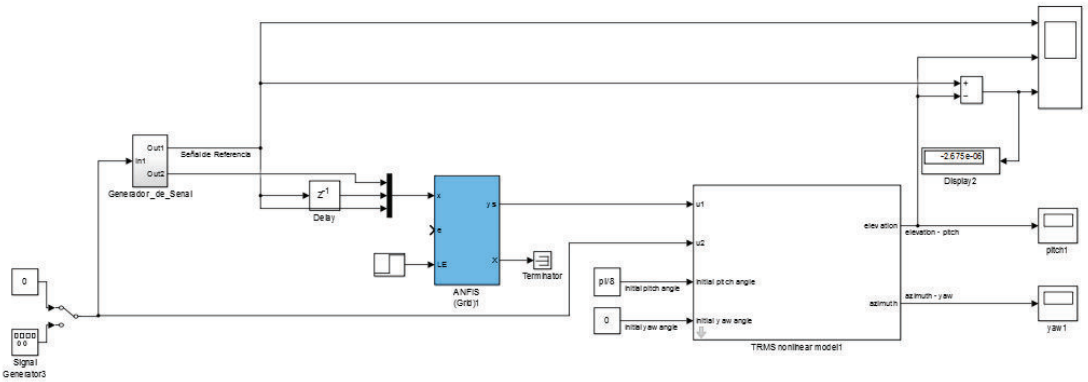


Figura 3.6. Sistema de Control Directo Inverso de la planta TRMS mediante una red ANFIS

Tener presente que ahora la señal escalón que ingresa a la entrada LE del controlador ANFIS siempre debe valer 0 puesto que ya no está entrenado a la red ANFIS, sino que se están empleando los parámetros lineales (Capa 1) y los parámetros no lineales (Capa 4) obtenidos en el proceso de identificación directo inverso. Para poder utilizar los parámetros adquiridos en el proceso de identificación se debe colocar en la ventana de comandos de MATLAB la instrucción *load('anfis_gridTRMS3.mat')* con lo cual se carga la matriz que contiene los parámetros antes mencionados y el entorno Simulink ahora puede hacer uso de estos valores siempre y cuando se coloque en la casilla etiquetada como *"Estados Iniciales ANFIS [x0;d0]"* el nombre del archivo *anfis_gridTRMS3.mat;* esta casilla se encuentra localizada en la ventana de configuración de la red ANFIS, tal y como se muestra en la Figura 3.7.

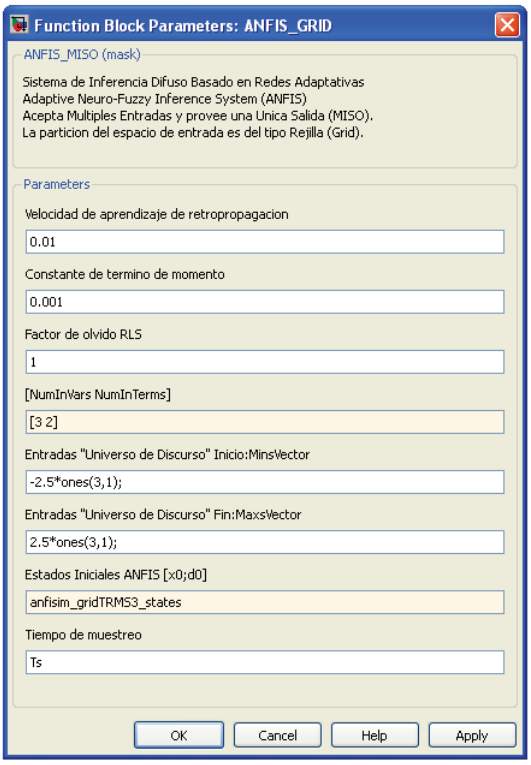


Figura 3.7. Ventana de configuración de la red ANFIS para el proceso de control

Una vez realizadas todas las configuraciones indicadas y culminado el proceso de simulación se obtuvieron las señales mostradas en la Figura 3.8. El error que se obtuvo al final fue de $-2.674x10^{-6}$ (ver Figura 3.6).

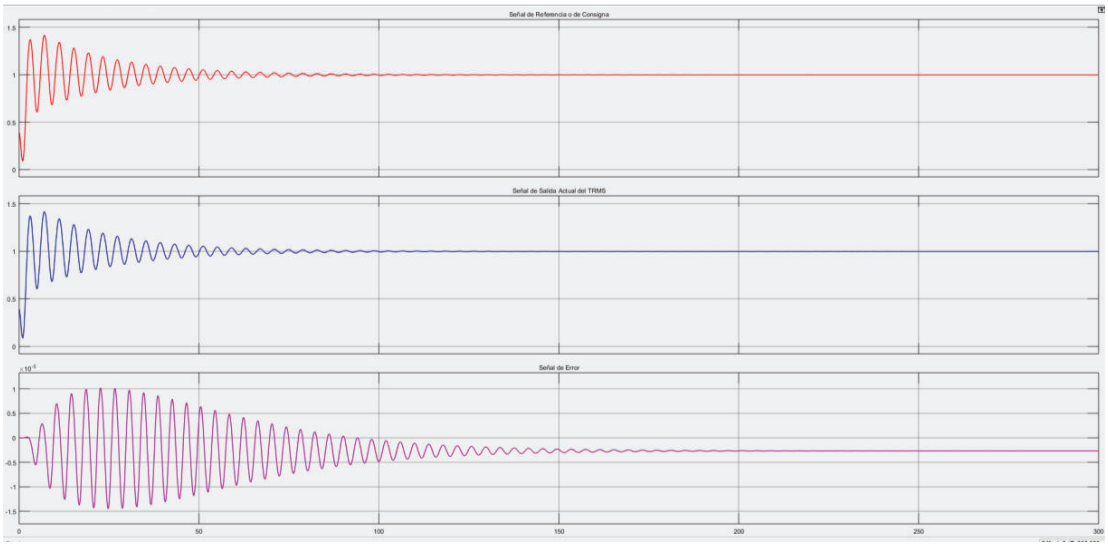


Figura 3.8. Señales obtenidas en el proceso de control directo inverso de la planta TRMS con una red ANFIS como controlador

Con la finalidad de verificar que el proceso de entrenamiento realizado con una señal de entrada tipo aleatorio es válido para cualquier tipo de entrada de referencia, se ingresó otra señal formada por diferentes frecuencias y se obtuvieron los resultados mostrados en la Figura 3.9. El error que se obtuvo al final fue de $7.33x10^{-4}$.

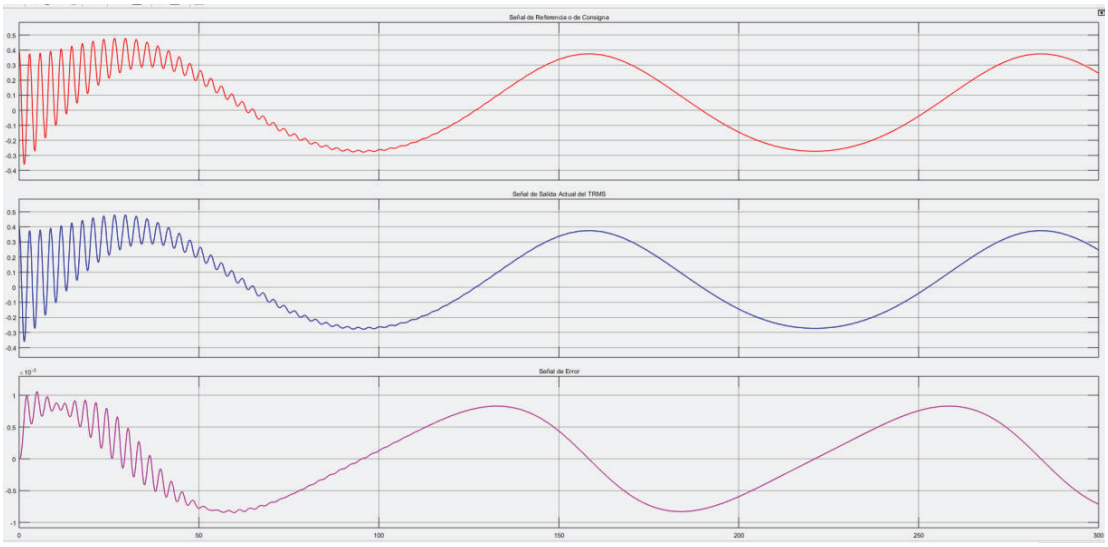


Figura 3.9. Señales obtenidas en el proceso de control directo con un señal de entrada diferente

Culminado el proceso de simulación del controlador ANFIS se procedió a realizar las pruebas del mismo sobre la planta física TRMS. Se partió de los ejemplos de tiempo real proporcionados por la empresa Feedback Instruments localizados en la ventana *TRMSModels* y agrupados en la carpeta *TRMS Real-Time Models*, según se muestra en la Figura 3.10. La explicación, uso y ejecución de estos ejemplos puede ser encontrada en el manual *"Twin Rotor MIMO System Control Experiments"* suministrado en la carpeta *Documentation Control Experiments*.

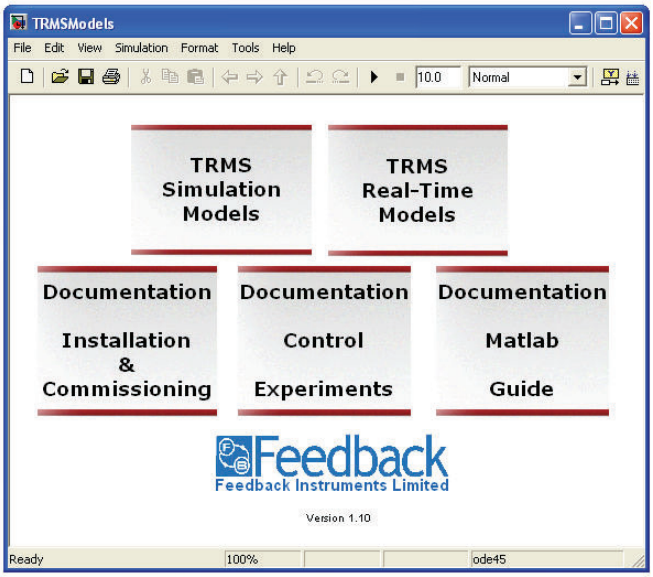


Figura 3.10. Ventana principal del sistema TRMSModels

Dentro de la carpeta *TRMS Real-Time Models* se localizan varios ejemplos tanto de identificación como de control del sistema TRMS basados en la Ingeniería de Control Clásica, como se puede observar en la Figura 3.11. De este conjunto se va utilizar como punto de partida el etiquetado como *TRMS_PID_Pitch* el cual corresponde a un sistema de control PID que maneja la salida del movimiento rotacional en el plano vertical φ (*pitch*). Los elementos de este sistema de control son mostrados en la Figura 3.12. El bloque etiquetado como *TRMS pitch and yaw* representa los sensores del sistema (encoders) y proporciona los ángulos reales (en radianes) del movimiento rotacional vertical y horizontal de las hélices de la planta TRMS. Los bloques etiquetados como *Rotor* y *Tail* corresponden a las entradas de las señales de control del PID hacia los motores de las dos hélices, respectivamente.

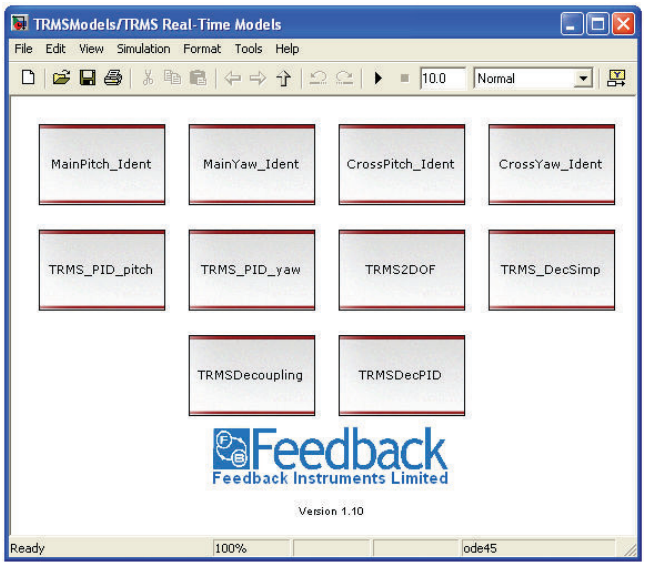


Figura 3.11. Ejemplos de identificación y control en tiempo real

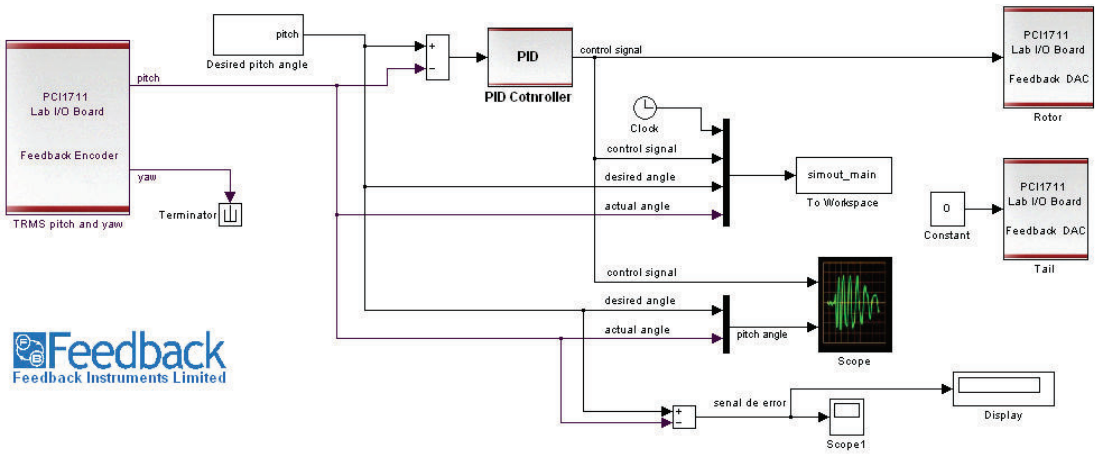


Figura 3.12. Sistema de control PID aplicado a la planta TRMS física

El resto de elementos localizados en la Figura 3.12 ya han sido utilizados en los procesos de simulación anteriores y no es necesaria su explicación. Como el objetivo de este trabajo es verificar el funcionamiento del controlador ANFIS aplicado sobre la planta TRMS se procede a eliminar el bloque PID y reemplazarlo por una red ANFIS cuyo entrenamiento esté basado en los esquemas presentados en las Figuras 3.2 y 3.6. La configuración final del sistema de control ANFIS es la mostrada en la Figura 3.13, a la cual se le ha agregado un bloque y un display de visualización para monitorear el error entre la señal del ángulo deseado y la señal del ángulo real.

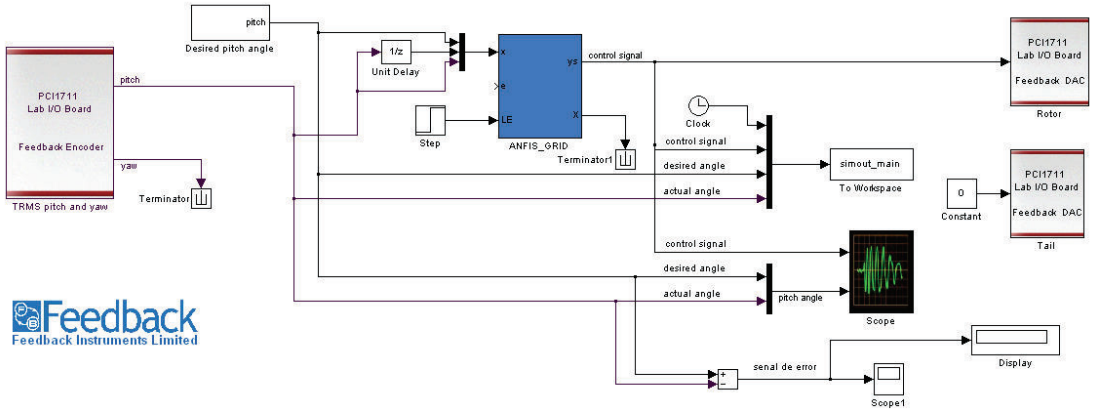


Figura 3.13. Sistema de control ANFIS aplicado a la planta TRMS física

Los resultados tras el proceso de verificación en tiempo real del controlador ANFIS son los mostrados en la Figura 3.14. La primera señal corresponde a la señal de control de la red ANFIS y en el segundo sub-panel se contraponen la señal de salida real con la señal deseada y se obtuvo un error final con un valor de $3.57x10^{-4}$, que está muy cercano al error obtenido en la fase de simulación.

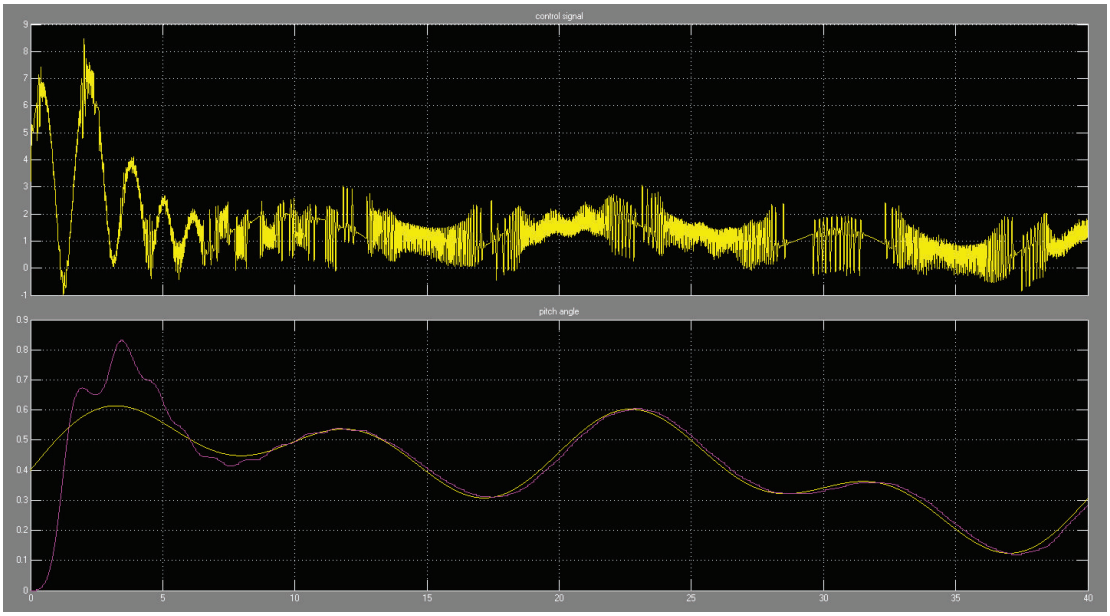


Figura 3.14. Señales obtenidas en el proceso de control ANFIS sobre la planta TRMS

3.3 IMPLEMENTACIÓN DE UN CONTROLADOR PID SOBRE EL SISTEMA TRMS

El control PID genera tres tipos de acciones (proporcional P, integral I y derivativa D) que se combinan entre sí para generar una única señal de control. Este control combinado reúne las mejores ventajas de cada una de las tres acciones de control individuales. El control PID genera una salida que es proporcional a la magnitud, a la duración y a la variación de la señal de error. La relación entre la salida y la entrada del controlador y considerando que $T_i = K_p/K_i$ y $T_d = K_d/K_p$ es:

$$m(t) = K_p \,.e(t) + K_i \int_0^t e(t)dt + K_d \,.\frac{de(t)}{dt} = K_p \,.\left[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d.\frac{de(t)}{dt}\right]$$

siendo la función de transferencia del controlador:

$$G_c(s) = \frac{M(s)}{E(s)} = K_p \,.\left[1 + \frac{1}{T_i \,.\, s} + T_d s\right]$$

La estructura en diagrama de bloques se muestra en la Figura 3.15, donde K_p es una constante conocida como ganancia proporcional, K_i es la ganancia integral y K_d es la ganancia derivativa. Variando el valor de estas constantes bajo ciertos criterios matemáticos o mediante criterios de sintonización (por ejemplo las reglas de sintonización de Ziegler y Nichols) se establece el funcionamiento del controlador PID.

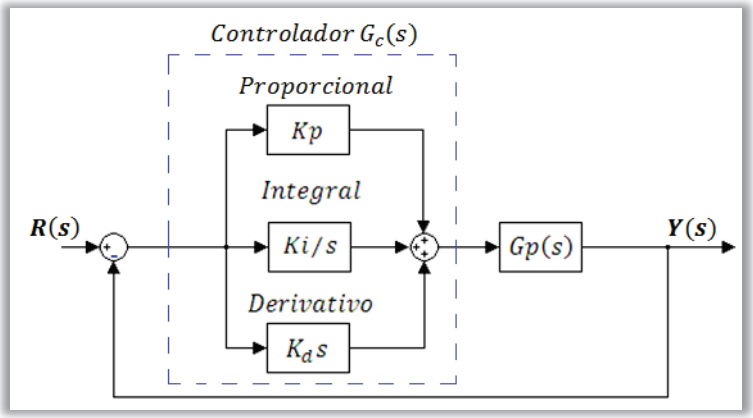


Figura 3.15. Diagrama de bloques de un controlador PID

En un control PID, en el inicio, toda la acción es D y en el infinito toda la acción es I. Si la señal de error tiene forma de rampa, la salida del controlador PID es la indicada en la Figura 3.16.

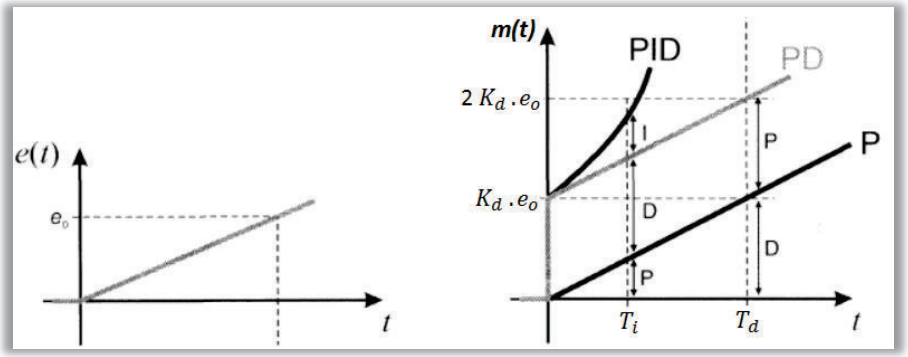


Figura 3.16. Respuesta del controlador PID

El controlador PID introduce al sistema dos ceros (uno por la acción integral y otro por la acción derivativa) que pueden estar situados en cualquier parte del lado izquierdo del plano s y un polo en el origen:

$$\frac{M(s)}{E(s)} = K_p\,.\left[1 + \frac{1}{T_i\,.\,s} + T_d s\right] = K_p\,.\left[\frac{(T_i\,.s) + 1 + (T_i.T_d.s^2)}{T_i.\,s}\right]$$

$$= K_p\,.\left[\frac{s + {}^{1}/_{T_i} + T_d s^2}{s}\right]$$

$$= K_p\,.T_d\left[\frac{s^2 + (1/T_d)s + 1/T_d T_i}{s}\right] = K_d\,.\left[\frac{(s + c_d)(s + c_i)}{s}\right]$$

donde $K_d = K_p.T_d$, $c_d = 1/T_d$ y $c_i = 1/T_i$. Este control es el más general, y con toda probabilidad es el tipo de controlador más utilizado a nivel industrial, ya que permite una óptima explotación de las características de las tres acciones básicas de control. Se puede considerar como un controlador proporcional, que dispone de un control integral para eliminar el error en estado estacionario y un control derivativo para mejorar la estabilidad relativa y aumentar la rapidez de la respuesta (responde rápidamente a cambios de error). La mejora de estabilidad relativa implica una respuesta transitoria con tiempos de adquisición y un valor de máximo sobreimpulso pequeños.

En esta combinación el control proporcional da forma a la curva de respuesta de la variable controlada. Produce más salida cuanto mayor sea el error y estabiliza la oscilación natural de la salida del sistema. El control integral disminuye el error y produce más salida cuanto más tiempo perdure el error. El control derivativo disminuye el tiempo durante el que cambia el error, prediciendo, con antelación el valor del cambio del error, y, frecuentemente, reduce el error ejecutando por adelantado las correcciones oportunas, con el fin de estabilizar más rápidamente la salida del sistema. Proporciona más salida cuanto más rápidamente se produce el error (adelanto de la respuesta).

Un controlador PID puede ser implementado mediante circuitos electrónicos o mediante software. En el caso particular de este trabajo se va emplear la segunda opción valiéndonos del hecho de que se cuenta de antemano con un sistema de control PID que viene incorporado en los ejemplos de tiempo real del sistema TRMS. El esquema de control PID es el mostrado en la Figura 3.17.

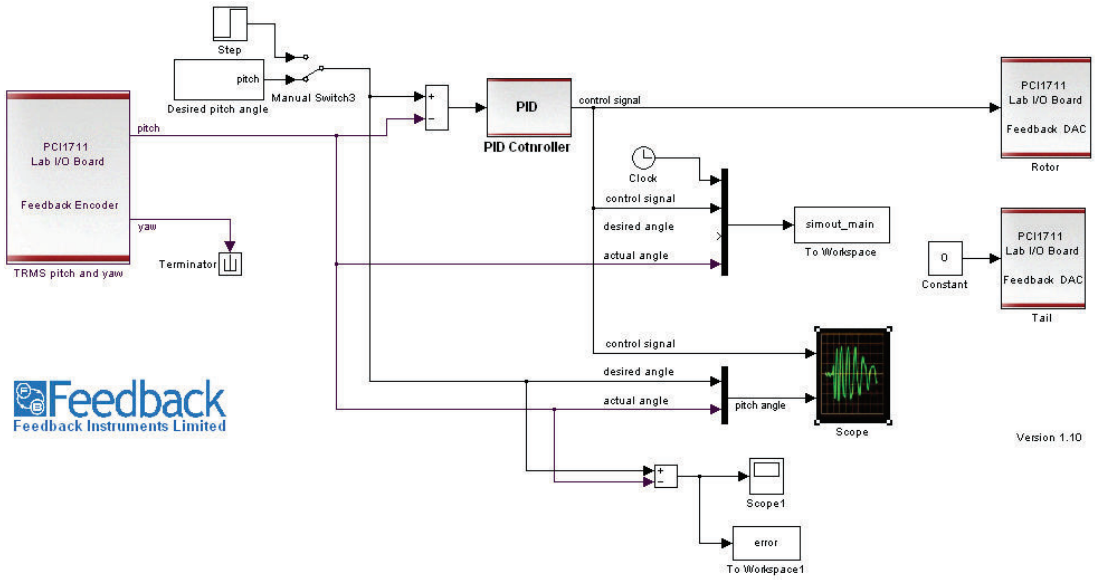


Figura 3.17. Sistema de control PID aplicado a la planta TRMS

Los valores de las constantes K_p , K_i y K_d han sido establecidos con los valores que vienen incluidos por defecto en el ejemplo del sistema de control PID (ver Figura 3.18). Estos valores pueden ser modificados según los criterios de control

de estado transitorio y estado estable requeridos por el sistema TRMS. En (K. Ogata, 2010) [15] y (R. Hernández, 2010) [16] se puede encontrar la explicación de un método de sintonización PID basado en las reglas de Ziegler y Nichols.

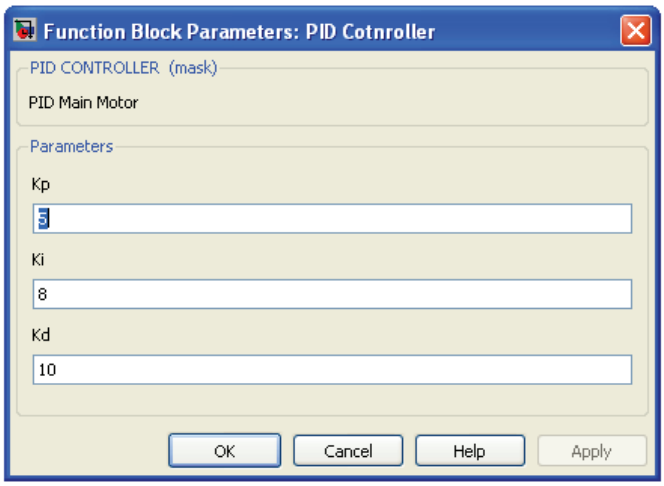


Figura 3.18. Valores asignados a las constantes K_p, K_i y K_d del controlador PID

Tras haber ejecutado el sistema de control PID de la Figura 3.17 se obtuvieron los resultados mostrados en la Figura 3.19. La primera señal corresponde a la señal de control del PID y en el segundo sub-panel se contraponen la señal de salida real con la señal deseada y se obtuvo un error final con un valor de 0.04

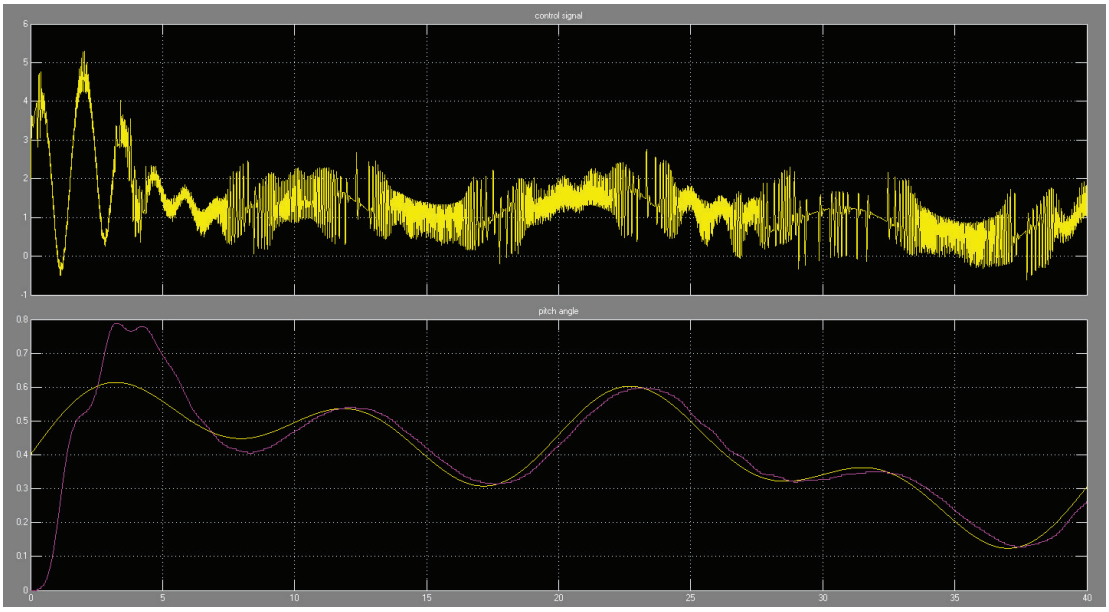


Figura 3.19. Señales obtenidas en el proceso de control PID de la planta TRMS

3.4 ANÁLISIS COMPARATIVO DE DESEMPEÑO DEL CONTROLADOR PID VS. EL CONTROLADOR ANFIS

Para hacer el análisis comparativo de ambos controlador se va hacer uso de un método estadístico conocido como Test de Wilcoxon, el cual es apropiado para el análisis de datos de un diseño de medidas repetidas con dos condiciones. Este método es utilizado cuando los datos no cumplen con los requisitos para una prueba paramétrica (es decir, si los datos no se distribuyen normalmente, si las varianzas de las dos condiciones son muy diferentes o si los datos son medidos en una escala ordinal). De lo contrario, si los datos cumplen con los requisitos para una prueba paramétrica, es mejor usar una prueba T de medidas repetidas.
La lógica detrás de la prueba de Wilcoxon es bastante simple. Los datos se clasifican para producir dos rangos totales, uno para cada condición. Si hay una diferencia sistemática entre las dos condiciones, entonces la mayoría de los rangos altos pertenecerán a una condición y la mayor parte de los rangos bajos pertenecerán a la otra. Como resultado, los rangos totales serán muy diferentes y uno de los rangos totales será bastante pequeño. Por otra parte, si las dos condiciones son similares, luego los rangos altos y bajos serán distribuidos de manera bastante uniforme entre las dos condiciones y los rangos totales serán bastante similares y bastante grandes. La prueba estadística de Wilcoxon "W" es simplemente el más pequeño de los rangos totales, para una mayor comprensión de este método puede consultar Levine et al. (2008) [21].

La función esencial de un sistema de control realimentado es el de reducir el error, $e(t)$, entre cualquier variable y su valor exigido a cero tan rápidamente como sea posible. Por lo tanto, cualquier criterio utilizado para medir la calidad de la respuesta del sistema debe tener en cuenta la variación de la señal $e(t)$, en todo el intervalo de tiempo. Cuatro criterios o indicadores de rendimiento básicos son de uso común: integral del error absoluto (IAE), integral del error cuadrado (ISE), integral del tiempo multiplicado por el error absoluto (ITAE) y la integral del tiempo multiplicado por el cuadrado del error (ITSE). Para cualquiera de los posibles criterios, la mejor respuesta corresponde al valor mínimo del criterio elegido.

Tenga en cuenta que en todos los casos o bien es el error absoluto o el error cuadrado el que está implicado: la integración directa del error produciría resultado cero, incluso si la respuesta del sistema fuese una oscilación de amplitud constante. Las integrales se evalúan en un período de tiempo fijo (en teoría hasta el infinito, pero por lo general hasta un tiempo suficientemente largo para que las respuestas se estabilicen).

Para realizar el Test de Wilcoxon se utilizará como datos de entrada los IAE de ambos controladores (PID y ANFIS), puesto que este indicador de rendimiento es uno de los más utilizados a nivel industrial en el campo de los sistemas de control. El Test de Wilcoxon es recomendado para muestras (n) pequeñas mayores a 25, por dicha razón se han calculado 35 valores de IAE partir de las señales de error $e(t)$ de ambos sistemas de control. En la Tabla 3.1 se demuestra mediante el Test de Wilcoxon, si el IAE del Controlador PID es igual o diferente al IAE del Controlador ANFIS. Hay que tomar en consideración las siguientes condiciones:

a. Si el IAE_{PID} es igual al IAE_{ANFIS}, no se lograría ningún tipo de optimización.

b. Si el IAE_{PID} es mayor al IAE_{ANFIS}, se estaría demostrando que se ha optimizado el sistema de control mediante una red ANFIS, pues se ha disminuido la pérdida de energía en el sistema.

c. Si el IAE_{PID} es menor al IAE_{ANFIS}, se estaría empeorando el sistema de control, ya que no habría ninguna disminución de energía, sino más bien un incremento de la misma.

Tabla 3.1. Muestras obtenidas para el Análisis del Test de Wilcoxon PID Vs. ANFIS, en el Proceso de Control del Sistema TRMS

TEST DE WILCOXON PARA DATOS PAREADOS, EN EL PROCESO DE CONTROL DEL SISTEMA TRMS							
HIPÓTESIS		**Principal (Ho):** IAE PID = IAE ANFIS			Media(IAE PID) = Media(IAE ANFIS)		
		Alternativa (Ha): IAE PID > IAE ANFIS			Media(IAE PID) > Media(IAE ANFIS)		
n	**IAE PID**	**IAE ANFIS**	**DIFERENCIAS**	**RANGOS ASIGNADOS**	**LIGADURA**	**DIFERENCIAS ORDENADAS**	**RANGOS DE ORDEN**
1	2.173	3.37E-04	2.172	21	1.5	2.113	1
2	2.133	3.43E-04	2.132	9	1.5	2.113	2
3	2.143	3.43E-04	2.142	15	4.5	2.122	3
4	2.192	3.50E-04	2.192	26	4.5	2.122	4
5	2.123	3.50E-04	2.122	4.5	4.5	2.122	5
6	2.133	3.43E-04	2.132	9	4.5	2.122	6
7	2.173	3.32E-04	2.172	21	9	2.132	7
8	2.192	3.27E-04	2.192	26	9	2.132	8
9	2.202	3.32E-04	2.202	29	9	2.132	9
10	2.212	3.32E-04	2.212	32.5	9	2.132	10
11	2.212	3.27E-04	2.212	32.5	9	2.132	11
12	2.123	3.50E-04	2.122	4.5	15	2.142	12
13	2.123	3.57E-04	2.122	4.5	15	2.142	13
14	2.143	3.65E-04	2.142	15	15	2.142	14
15	2.143	3.57E-04	2.142	15	15	2.142	15
16	2.123	3.65E-04	2.122	4.5	15	2.142	16
17	2.113	3.57E-04	2.113	1.5	15	2.142	17
18	2.113	3.22E-04	2.113	1.5	15	2.142	18
19	2.212	3.22E-04	2.212	32.5	21	2.172	19
20	2.192	3.22E-04	2.192	26	21	2.172	29
21	2.173	3.18E-04	2.172	21	21	2.172	21
22	2.133	3.18E-04	2.132	9	21	2.172	22
23	2.133	3.22E-04	2.132	9	21	2.172	23
24	2.133	3.27E-04	2.132	9	26	2.192	24
25	2.143	3.27E-04	2.142	15	26	2.192	25
26	2.143	3.50E-04	2.142	15	26	2.192	26
27	2.143	3.43E-04	2.142	15	26	2.192	27
28	2.143	3.43E-04	2.142	15	26	2.192	28
29	2.192	3.43E-04	2.192	26		2.202	29
30	2.173	3.57E-04	2.172	21	32.5	2.212	30
31	2.173	3.57E-04	2.172	21	32.5	2.212	31
32	2.212	3.43E-04	2.212	32.5	32.5	2.212	32
33	2.212	3.32E-04	2.212	32.5	32.5	2.212	33
34	2.212	3.43E-04	2.212	32.5	32.5	2.212	34
35	2.192	3.32E-04	2.192	26	32.5	2.212	35

Para determinar los valores de IAE de ambos sistemas de control, se utilizaron 100 muestras de error almacenadas en un arreglo y que fueron obtenidas desde el entorno Simulink mientras se realizaba el proceso de ejecución de los controladores ANFIS y PID. El código de programación que se implementó en Matlab para el cálculo del IAE es el que se muestra a continuación:

```
f0=abs(error(1));
IAE=0;
for k=2:1:length(error)
IAE=IAE+(f0+abs(error(k)))/2;
f0=abs(error(k));
end
disp('El error Absoluto es:')
IAE
```

Análisis estadístico: Siguiendo el procedimiento de Wilcoxon, las muestras son analizadas, arrojando los valores mostrados en la Tabla 3.2.

Tabla 3.2. Análisis del Test de Wilcoxon PID Vs. ANFIS, en el Proceso de Control del Sistema TRMS

T+			630
T-			0
T=min(T+,T-)			630
$Z = \frac{T - n\frac{(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}$			5.1594
Nivel de Confianza del 95% (Za)			1.96
Decisión	**Z ≤ Za**	**Se acepta la Ho**	
	Z > Za	**Se acepta la Ha**	OK
Interpretación			Como el IAE del sistema de control con PID, es mayor al IAE del sistema de control con ANFIS, se concluye que el controlador ANFIS es mejor en el proceso de control del TRMS, pues logra disminuir perdida de energía en el sistema

CAPÍTULO 4
CONCLUSIONES Y RECOMENDACIONES

Las siguientes conclusiones y recomendaciones son producto de la simulación, experimentación física y de la experiencia adquirida en el desarrollo del presente trabajo de investigación y por ende están enfocadas a los resultados obtenidos en el Capítulo 2 y Capítulo 3 de este documento.

4.1 CONCLUSIONES

- Se cumplió con gran éxito el objetivo general de este trabajo de investigación el cual era desarrollar el algoritmo computacional de un controlador neuro-difuso tipo ANFIS, como una buena alternativa para implementar sistemas de control sobre plantas muy complejas y con características no lineales, como es el caso del sistema TRMS (Twin Rotor MIMO System) en donde se realizaron las pruebas que validaron el correcto funcionamiento de la red ANFIS tanto como un elemento de modelamiento, así como un componente de control.

- Se desarrolló el algoritmo de programación de la arquitectura de la red ANFIS empleando conjuntamente el software Matlab y su entorno gráfico Simulink, para lo cual se utilizó un bloque S-function que permitió enlazar el código desarrollado en Matlab con el entorno Simulink en donde se realizaron las respectivas pruebas de funcionamiento sobre el modelo matemático de la planta TRMS previamente obtenido.

- El proceso de actualización y ajuste de los parámetros de la red adaptativa ANFIS está basado en una regla de aprendizaje híbrida implementada en software, la cual combina el método del gradiente descendente del algoritmo de retropropagación del error (que es propio de las redes

neuronales) con el estimador de mínimos cuadrados recursivo (RLSE). Más específicamente, en el paso en avance del algoritmo de aprendizaje híbrido, las salidas de los nodos van hacia adelante hasta la Capa 4 y los parámetros del consecuente (parámetros no lineales) se identifican por el método de mínimos cuadrados recursivos RLSE. En el paso en retroceso, las señales de error se propagan hacia atrás y los parámetros de la premisa (parámetros lineales) de la Capa 1 se actualizan mediante el método gradiente descendente del algoritmo de retropropagación. En consecuencia, este enfoque híbrido converge mucho más rápido, ya que reduce las dimensiones del espacio de búsqueda del método de retropropagación puro original.

- Se evaluó el desempeño del controlador neuro-difuso ANFIS aplicándolo sobre la planta física TRMS, bajo diferentes condiciones de operación correspondientes a la variación de las señales de referencia del sistema de control. Se comprobó que el error existente entre la señal de entrada deseada y la señal de salida real de la planta TRMS oscilaba con pequeñas variaciones dentro del rango de 3.18E-04 a 3.65E-04, que en términos de Ingeniería de Control puede ser considerado como un error insignificante y por tanto se puede asumir que la señal de salida es una fiel copia de la señal de entrada una vez que el sistema ha logrado estabilizarse.

- Se realizó un análisis comparativo de desempeño entre el controlador ANFIS y un controlador PID convencional que ya viene incorporado como ejemplo en el sistema TRMS. Luego se aplicó un método estadístico para dicho análisis conocido como Test de Wilcoxon y se empleó como datos de entrada para este método los IAE (integral del error absoluto) de ambos controladores (PID y ANFIS), puesto que este indicador de rendimiento es uno de los más utilizados a nivel industrial en el campo de los sistemas de control. Para evaluar el desempeño de estos controladores se usaron 35 valores de IAE que se obtuvieron a partir de las señales de error $e(t)$ de ambos sistemas de control. Al final se comprobó que el IAE del sistema de

control con PID, es mayor al IAE del sistema de control con ANFIS y se concluye que el controlador ANFIS es mejor en el proceso de control del TRMS, pues logra disminuir la pérdida de energía en el sistema.

4.2 RECOMENDACIONES

- Como ya se ha mencionado, la red ANFIS utiliza una partición tipo rejilla (grid) la cual presenta algunos problemas cuando se tiene un número moderadamente grande de entradas, debido a que el número de reglas difusas crece exponencialmente con la dimensión de la entrada $\left(N_{reglas} = {N_{MF}}^{N_{in}}\right)$ y dado que cada regla tiene pocos parámetros por ajustar existen demasiados parámetros por adaptar durante el proceso de aprendizaje, lo que provoca que la red ANFIS tenga una respuesta lenta. En un sistema de inferencia difuso convencional el número de reglas es determinado por un experto el cual está familiarizado con la planta a ser modelada o controlada; en este caso, para el proceso de control del sistema TRMS ningún experto está disponible y el número de funciones de pertenencia (MFs) asignado a cada variable de entrada ha sido elegido empíricamente, es decir, ingresando los datos de entrada a la red ANFIS, luego graficando la señal de salida y de error, para posteriormente examinarlas visualmente y emitir los respectivos juicios de validez. Para un conjunto de datos con más de tres entradas, las técnicas de visualización no son muy efectivas y la mayoría de las veces se debe depender de las pruebas de ensayo y error. Esta situación es similar a la de las redes neurales; simplemente no hay forma sencilla de determinar de antemano el número mínimo de unidades ocultas necesarias para alcanzar un nivel de rendimiento deseado. Existen otras técnicas para determinar los números de MFs y reglas, tales como los métodos de agrupamiento y CART, pero aquí no se han utilizado, dado que el propósito de este trabajo es demostrar la capacidad de aprendizaje de la red ANFIS. Por dicha razón se recomienda asignar a cada variable de entrada un número pequeño de funciones de pertenencia (por ejemplo empezar con 2 MFs), luego realizar las

respectivas pruebas de funcionamiento y verificar si el proceso de aprendizaje de la red ANFIS está dentro de los niveles de rendimiento requeridos, caso contrario, ir aumentando el número de MFs (de uno en uno) hasta conseguir que el nivel de aprendizaje de la red ANFIS esté dentro de los márgenes requeridos por el sistema de modelamiento y control.

- Se recomienda definir correctamente el universo de discurso de las funciones de pertenencia MFs presentes en los nodos de la Capa 1 y que corresponden al rango de variación de las señales de entrada a la red ANFIS. En el caso particular del sistema TRMS estos valores están dados por las señales de control enviadas a los actuadores (motores DC) de las hélices y que varían entre -2.5V a 2.5V. Si no se configura de forma adecuada estos valores no se conseguirá un buen proceso de aprendizaje de la red ANFIS.

- En el proceso de control del sistema TRMS mediante una red ANFIS se utilizó el esquema de Control Directo Inverso el cual es propio de los sistemas de control basados en redes neuronales. Existen ciertos problemas que se pueden experimentar con este tipo de control debido a las siguientes razones: falta de robustez, lo que resulta del hecho de que no se utiliza ningún error de realimentación directo; aprendizaje ineficiente, causado por la selección de un rango operacional impropio de los datos; los datos operativos reales pueden ser difíciles de definir a priori. Por estas razones y dependiendo del sistema a controlar, se recomienda experimentar con otros esquemas de control como los señalados en el apartado 1.4.5.2 o emplear alguna de las configuraciones de los sistemas de control adaptativos, que pueden ser consultadas en la literatura presentada dentro las referencias bibliográficas de este documento.

REFERENCIAS BIBLIOGRÁFICAS

[1] J. Jang (1993), “ANFIS: Adaptive-Network-Based Fuzzy Inference System”, University of California, Berkeley.

[2] J. Jang, C. Sun & E.Mizutani (1997), “Neuro-Fuzzy and Soft Computing”, Editorial Prentice-Hall, Estados Unidos de América, pp. 73-89, 95-117, 199-223, 335-363.

[3] J. Jang & C. Sun (1995), “Neuro-Fuzzy Modeling and Control”.

[4] Y. Bravo & D.García (2002), “ESTUDIO Y APLICACIÓN DE LOS MODELOS ANFIS DE MATLAB”, tesis EPN, Quito-Ecuador.

[5] B.M. del Brio & A. Sanz (2008), “Redes Neuronales y Sistemas Borrosos”, 3ra Edición, Editorial Alfaomega - Ra-ma, Madrid-España, pp. 271-283.

[6] E. Caicedo & L. Lopez (2008), “Una aproximación práctica a las Redes Neuronales Artificiales”, Primera Edición, Universidad del Valle, Colombia, pp. 75-107.

[7] A. Alanís & E. Sánchez (2006), “REDES NEURONALES: Conceptos fundamentales y aplicaciones a control automático”, Primera Edición, Editorial PRENTICE-HALL, España, pp. 63-84.

[8] N. Siddique & H. Adeli (2013), “Computational intelligence: synergies of fuzzy logic, neural networks, and evolutionary computing”, Editorial John Wiley & Sons, India, pp. 159-180.

[9] R. Fullér (2000), “Introduction to Neuro-Fuzzy Systems”, Advances in Soft Computing Series, Springer-Verlag, Berlin/Heildelberg.

[10] A. Abraham (2005), “Adaptation of Fuzzy Inference System Using Neural Learning”, Computer Science Department, Oklahoma State University, USA.

[11] Feedback Instrument (2006), “Twin Rotor MIMO System Control Experiments”, Park Road, Crowborough, East Sussex-UK.

[12] S. Zeghlache, K. Kara & D. Saigaa (2014), “Type-2 Fuzzy Logic Control of a 2-DOF Helicopter (TRMS system)”, Central European Journal of Engineering.

[13] Feedback Instrument (2006), “Twin Rotor MIMO System Installation & Commissioning ”, Park Road, Crowborough, East Sussex-UK.

[14] K.-L. Du and M.N.S. Swamy (2006), “Neural networks in a softcomputing framework”, Editorial Springer, Canada, pp. 389-392.

[15] K. Ogata (2010), “Ingeniería de control moderna”, Quinta Edición, Editorial Pearson Educación, Madrid, pp. 567-577.

[16] R. Hernández (2010), “Introducción a los sistemas de control: Conceptos, aplicaciones y simulación con MATLAB”, Primera Edición, Editorial Pearson Educación, México, pp. 371-381.

[17] H.T. Nguyen, N.R. Prasad, C.L. Walker & E.A. Walker (2003), “A First Course in FUZZY and NEURAL CONTROL”, Editorial Chapman & Hall/CRC, Estados Unidos de América, pp. 229-246.

[18] P.Ponce & F.Ramírez (2010), “Intelligent Control Systems with LabVIEW”, Editorial Springer, Londres, pp. 106-121.

[19] P.Ponce (2010), “INTELIGENCIA ARTIFICIAL CON APLICACIONES A LA INGENIERÍA”, Editorial Alfaomega Grupo Editor, México, pp. 244-284.

[20] I. Konsoulas (2015), “Adaptive Neuro-Fuzzy Inference Systems (ANFIS) Library for Simulink”, MathWorks.

[21] D. Levine, D. Stephan, T. Krehbiel & M. Berenson (2008), “STATISTICS FOR MANAGERS USING Microsoft Excel”, Quinta Edición, Editorial Prentice Hall, Estados Unidos de América.

ANEXOS

ANEXO I
FUNCIONES S-FUNCTION CON MATLAB Y SIMULINK

Una S-function (System-function) es un lenguaje de descripción de un bloque de Simulink escrito en código de MATLAB, C, C ++ o Fortran. Las S-functions en C, C ++ y Fortran son compiladas como archivos MEX mediante la utilidad *mex* antes de ser utilizadas en el entorno de Simulink.

Las S-functions siguen una forma general y pueden alojar sistemas continuos, discretos e híbridos. Al seguir una serie de reglas simples, se puede implementar un algoritmo de programación y utilizar el bloque S-Function de Simulink para añadir este algoritmo a un modelo Simulink. Dicho en otras palabras, una S-function es una función compuesta de dos elementos, un bloque de Simulink definible por el usuario y un programa escrito en cualquiera de los lenguajes de programación antes mencionados, el cual es llamado o cargado en el bloque.

Las S-functions son utilizadas para una variedad de aplicaciones, incluyendo: creación de nuevos bloques de propósito general; adición de bloques que representan controladores de dispositivos de hardware; incorporación de código C existente en una simulación; descripción de un sistema como un conjunto de ecuaciones matemáticas; descripción de modelos dinámicos complicados (con parámetros variantes con el tiempo, sistemas no lineales, con diferentes periodos de muestreo, etc.) y para animaciones gráficas.

El uso más común de las S-functions es para crear bloques personalizados en Simulink. Cuando se utiliza una S-function para crear un bloque de propósito general, se puede utilizar muchas veces este bloque en un modelo, variando los parámetros con cada instancia del bloque.

Las S-functions utilizan una sintaxis de llamado especial conocida como API S-function, la cual permite a este tipo de funciones interactuar con el motor de Simulink. Esta interacción es muy similar a la interacción que tiene lugar entre el motor y los bloques incorporados en Simulink.

CONCEPTOS PRELIMINARES

Para crear las S-functions, es necesario entender su funcionamiento. Tal conocimiento requiere una comprensión de cómo el motor Simulink simula un modelo, incluyendo las matemáticas de bloques. Esta sección comienza explicando las relaciones matemáticas entre las entradas, los estados y salidas de un bloque.

Matemáticas de bloques de Simulink

Un bloque de Simulink se compone de un conjunto de entradas, un conjunto de estados y un conjunto de salidas (ver Figura AI.1), donde las salidas están en función del tiempo de simulación, las entradas y los estados.

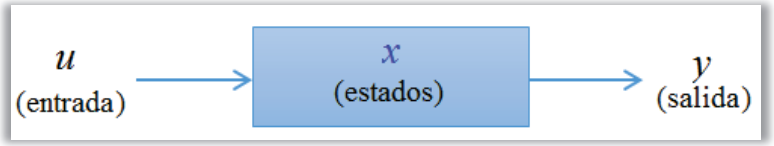


Figura AI.1. Representación general de un bloque en Simulink.

Las siguientes ecuaciones expresan las relaciones matemáticas entre las entradas, las salidas, los estados y el tiempo de simulación.

$$y = f_0(t, x.u) \qquad \text{(Salidas)}$$

$$\dot{x} = f_d(t, x.u) \qquad \text{(Derivadas)}$$

$$x_{d_{k+1}} = f_u\left(t, x_c, x_{d_k}, u\right) \qquad \text{(Actualización)}$$

donde $x = [x_c; x_d]$

Etapas de Simulación

La ejecución de un modelo Simulink se desarrolla en etapas. Primero viene la fase de inicialización, en donde el motor de Simulink incorpora bloques de librería en el modelo, propaga anchos de señales, tipos de datos y tiempos de muestreo, evalúa los parámetros del bloque, determina el orden de ejecución del bloque y asigna memoria. El motor entonces entra en un bucle de simulación, en donde cada pasada a través del bucle se conoce como un paso de simulación. Durante cada paso de simulación, el motor ejecuta cada bloque en el modelo en el orden

determinado durante la inicialización. Para cada bloque, el motor invoca funciones que calculan los estados del bloque, las derivadas y las salidas para el tiempo de muestreo actual.

La Figura AI.2 ilustra las etapas de una simulación. El bucle de integración se lleva a cabo sólo si el modelo contiene estados continuos. El motor ejecuta este bucle hasta que el solucionador alcanza la precisión deseada para los cálculos de estado. El bucle de simulación entonces continúa hasta que la simulación se ha completado.

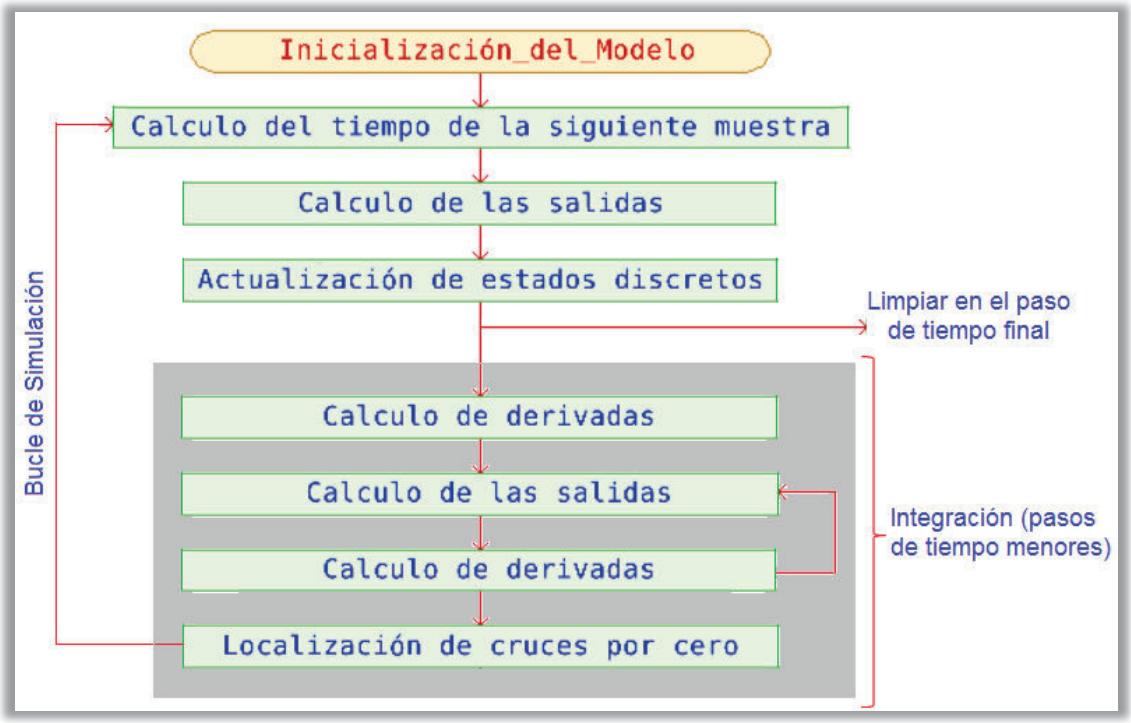


Figura AI.2. Fases de simulación de un modelo Simulink.

IMPLEMENTACIONES DISPONIBLES PARA S-FUNCTIONS

Se pueden implementar las funciones S-functions utilizando alguno de los siguientes caminos:

a. **Level-1 MATLAB S-function:** proporciona una sencilla interfaz de MATLAB para interactuar con una pequeña porción del API S-function. Contiene el protocolo con el cual Simulink puede acceder a información escrita en código de Matlab.

b. **Level-2 MATLAB S-function:** proporciona acceso a un conjunto más amplio del API S-function y soporta generación de código desarrollado por el usuario. En la mayoría de los casos, se utiliza este tipo de funciones cuando se quiere implementar la S-function en el entorno de MATLAB.

c. **C MEX S-function**: proporciona la mayor flexibilidad de programación en este tipo de funciones. Permite implementar el algoritmo como una *C MEX S-function* o escribir un contenedor *S-function* para llamar a un código existente en C, C++ o Fortran. Para escribir este tipo de funciones se requiere un amplio conocimiento del API S-function.

d. **S-Function Builder:** es una interfaz gráfica de usuario para la programación de un subconjunto de funcionalidades de la S-function. Si el usuario es nuevo en la escritura de funciones C MEX S-functions, puede utilizar el S-Function Builder para generar nuevas S-functions o incorporar código existente en C o C++ sin interactuar con el API S-function.

FUNCIONAMIENTO Y CONFIGURACIÓN DE UNA S-FUNCTION

En este trabajo se va a implementar un bloque de función mediante el método Level-1 MATLAB S-function, por dicha razón solo se explicará el funcionamiento y creación para este tipo de S-function. Para este método de implementación la función de MATLAB debe tener la siguiente forma

$$[sys, x0, str, ts] = f(t, x, u, flag, p_1, p_2, \dots)$$

donde f es el nombre de la S-function. Durante la simulación de un modelo, el motor de Simulink invoca repetidamente a f, usando el argumento $flag$ para indicar la tarea (o tareas) a realizar para una invocación en particular. La S-function realiza la tarea y retorna los resultados en un vector de salida denotado por $[sys, x0, str, ts]$.

Una plantilla para un Level-1 MATLAB S-function etiquetada como *sfuntmpl.m*, se encuentra localizada en *matlabroot/toolbox/simulink/blocks*. La plantilla consiste de una función de nivel superior y un conjunto de funciones locales, conocidas como métodos de devolución de llamada (*S-function callback methods*), donde

cada una es identificada por un valor en particular del $flag$. La función de nivel superior invoca a la función local indicada por el $flag$. Las funciones locales realizan las tareas reales requeridas por la S-function durante el proceso de simulación. El motor de Simulink pasa los siguientes argumentos a un Level-1 MATLAB S-function:

Tabla AI.1. Argumentos de entrada para una S-function.

t	Tiempo de simulación actual
x	Vector columna correspondiente a las variables de estados
u	Vector columna correspondiente a las variables de entradas
$flag$	Valor entero que indica la tarea a ser realizada por la S-function
$p_1, p_2, \ldots$	Parámetros adicionales

Existen seis tipos de tareas que realiza la S-function y cada una de las cuales se designa por un número entero asignado a la variable $flag$. En la Tabla AI.2 se describen los valores que puede asumir la variable $flag$.

Tabla AI.2. Valores y funciones de la variable $flag$

Valor del $flag$	Descripción
0	**Inicialización y Configuración** Define las características básicas del bloque S-Function, incluyendo los tiempos de muestreo, especificación y/o cálculo de las condiciones iniciales de las variables de estados continuos y discretos, configuración de los tamaños de los vectores de entrada y salida.
1	**Cálculo de Derivadas** Calcula las derivadas de las variables de estados continuos. También realiza cómputos que involucran a los vectores de

	entrada.
2	**Actualización de Estados Discretos** Actualiza los estados discretos, tiempos de muestreo y los principales requisitos del paso de tiempo (*time step*).
3	**Cálculo de las Salidas** Evalúa y calcula las variables de salida en función de los elementos del vector de estado (y en algunos casos, también los elementos de vector de entrada).
4	**Cálculo del Tiempo de la siguiente Petición** Calcula el tiempo de la siguiente petición (*hit*) en tiempo absoluto. Esta rutina se usa sólo cuando se especifica un tiempo de muestreo para una variable discreta en el método de configuración.
9	**Finalización** Rutinas y cálculos adicionales al final de la ejecución de la simulación.

Como ya se ha mencionado un tipo de implementación Level-1 MATLAB S-function retorna un vector de salida que contiene los siguientes elementos:

sys: vector principal de los resultados solicitados por Simulink. Dependiendo del valor de $flag$ enviado por Simulink, este vector contendrá diferente información, según se muestra en la Tabla A1.3.

El siguiente conjunto de tres argumentos de salida son utilizados por Simulink sólo cuando $flag = 0$, de lo contrario, se ignoran:

$x0$: vector columna con los valores de los estados iniciales (se coloca un vector vacío si no hay estados en el sistema).

str: originalmente destinado para uso futuro. Debe establecerse como una matriz vacía, [ ].

ts: matriz de dos columnas que contiene los tiempos de muestreo y los

tiempos *offsets* del bloque.

Por ejemplo, si desea que la S-function se ejecute en cada paso de tiempo (tiempo de muestreo continuo), se debe establecer ts a [0 0]. Si se requiere que la S-function se ejecute a la misma velocidad como la del bloque al que está conectado (tiempo de muestreo heredado), se establece ts a [-1 0]. Si se desea que esta se ejecute cada 0.25 segundos (tiempo de muestreo discreto) empezando en 0.1 segundos después del tiempo de inicio de la simulación, se ajusta ts a [0.25 0.1].

Tabla AI.3. Componentes del vector sys dependiendo del valor del $\boldsymbol{flag}$.

Si $\boldsymbol{flag = 0}$	$\boldsymbol{sys} = [\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{g}]$ Donde $\boldsymbol{a}$ = número de estados de tiempo continuo. $\boldsymbol{b}$ = número de estados de tiempo discreto. $\boldsymbol{c}$ = número de salidas (Nota: esto no es necesariamente el número de estados). $\boldsymbol{d}$ = número de entradas. $\boldsymbol{e} = \boldsymbol{0}$ (es requerido que sea 0, no se utiliza actualmente) . $\boldsymbol{f}$ = 0 (no) o 1 (sí) para alimentación algebraica directa a través de la entrada a la salida. Esto sólo es relevante si durante $\boldsymbol{flag = 3}$, las variables de salida dependen algebraicamente de las variables de entrada). $\boldsymbol{g}$ = número de tiempos de muestreo. Para procesos continuos, colocar un valor de 1.
Si $\boldsymbol{flag = 1}$	sys = vector columna de las derivadas de las variables de estado.
Si $\boldsymbol{flag = 3}$	sys = vector columna de las variables de salida.
Si $\boldsymbol{flag = 2, 4, 9}$	ya que estos indicadores no se utilizan en este trabajo, sólo pueden enviar un vector nulo: sys = [ ].

CREACIÓN DE UNA S-FUNCTION EN SIMULINK

Para incorporar una S-function del tipo Level-1 MATLAB S-function en un modelo Simulink, arrastre un bloque *S-Function* desde la biblioteca de bloques etiquetada como *User-Defined Functions* en el modelo. A continuación dar doble clic con el botón izquierdo del mouse y en la ventana que se despliega coloque el nombre de la función (que debe tener el formato solicitado para una S-function) en el campo *S-function name* del cuadro de diálogo del bloque, tal y como se ilustra en la Figura A1.3. En este ejemplo en particular la función desarrollada en MATLAB fue guardada con el nombre de *anfisim_grid.m* y corresponde al algoritmo de programación del sistema ANFIS utilizado en los procesos de control de este trabajo (para más detalle de la función ver el Anexo III).

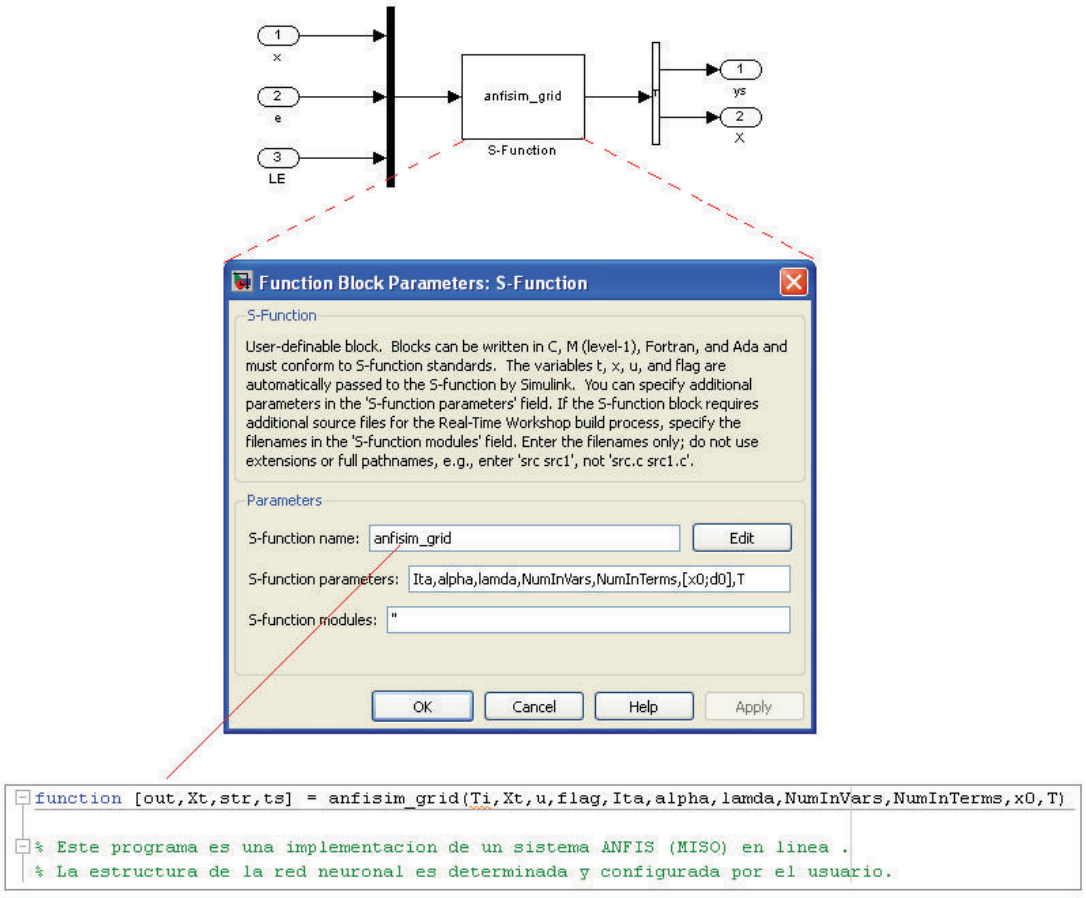


Figura AI.3. Creación de una S-function en Simulink.

Cuando se invoca al bloque S-Function, el motor de Simulink siempre pasa los parámetros del bloque t, x, u y $flag$, como argumentos de la función

anfisim_grid.m. El motor también puede pasar a la función *anfisim_grid.m* parámetros adicionales del bloque especificados por el usuario. El usuario puede especificar estos parámetros en el campo *S-function parameters* del cuadro de diálogo *S-Function Block Parameters*. Para utilizar este campo, se debe conocer de antemano los parámetros que la S-function requiere. Los parámetros se introducen uno tras otro separados por una coma y en el orden requerido por la función desarrollada en MATLAB. Los valores de los parámetros pueden ser constantes, nombres de variables definidas en MATLAB o el área de trabajo (*workspace*), o expresiones de MATLAB.

En el ejemplo mostrado en la Figura AI.3 los parámetros fueron definidos mediante los nombres de las siguientes variables: lta, alpha, lamda, NumInVars, NumInTerms, [x0,d0], T. Cada una de estas variables tiene un significado y propósito dentro del funcionamiento del algoritmo ANFIS. Una vez que se definen los parámetros adicionales, el motor de Simulink pasa estos parámetros a la función *anfisim_grid.m* y los coloca en la lista de argumentos de esta función en el orden en el cual estos fueron definidos en el cuadro de diálogo de bloque S-Function. Se puede utilizar esta capacidad para permitir que la misma S-function implemente varias opciones de procesamiento.

En este punto cabe mencionar que también es factible utilizar la opción de enmascaramiento para crear cuadros de diálogo e iconos personalizados para el bloque S-Function de Simulink. El cuadro de diálogo enmascarado permite que sea más fácil especificar los parámetros adicionales para la S-function. En primer lugar se va a crear un subsistema seleccionando todos los elementos relacionados con el bloque S-Function, luego se da clic derecho con el mouse y se elige la opción *Create Subsystem*, tal como se muestra en la Figura AI.4.

Tras realizar los pasos indicados, se genera un bloque como el mostrado en la Figura AI.5. Para asignarle el color al interior del bloque se debe dar clic derecho sobre el mismo y seguir la ruta *Background Color >Light Blue*.

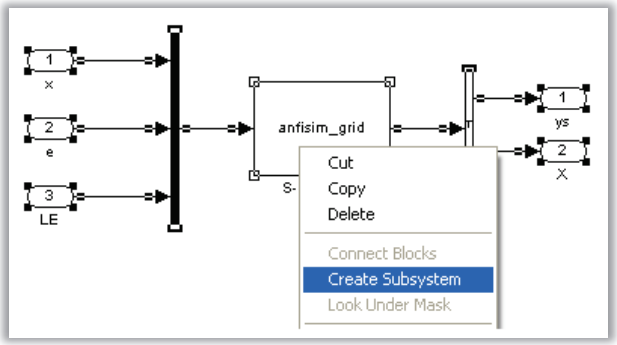


Figura AI.4. Creación de un Subsistema.

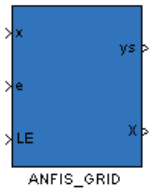


Figura AI.5. Presentación final de bloque del Subsistema.

Una vez creado el subsistema ANFIS_GRID es posible personalizarlo. El editor de máscaras se abre al hacer clic derecho en el menú contextual del bloque y seleccionando *Mask Subsystem*, tras realizar esta acción se despliega una ventana como la mostrada en la Figura AI.6. El Editor de Máscara (Mask Editor) contiene un conjunto de pestañas de paneles, cada una de las cuales permite definir una característica de la máscara: el panel *Icon & Ports* permite definir el icono del bloque; el panel *Parameters* permite definir e introducir los parámetros del subsistema, que en nuestro caso corresponden a los parámetros adicionales del bloque de la S-Function; el panel *Initialization* permite especificar los comandos de inicialización y el panel *Documentation* permite definir el tipo, la descripción y la ayuda de la máscara.

Los parámetros del bloque se definen en el panel mostrado en la Figura AI.7. En las celdas de la columna etiquetada como *Variable* se escriben los nombres de los parámetros que deben coincidir con los nombres de los parámetros adicionales definidos en el bloque de la S-Function. En las celdas de la columna etiquetada como *Prompt* se puede redactar una pequeña descripción del significado de las variables creadas. Para ir añadiendo variables se debe dar clic en el botón *Add*.

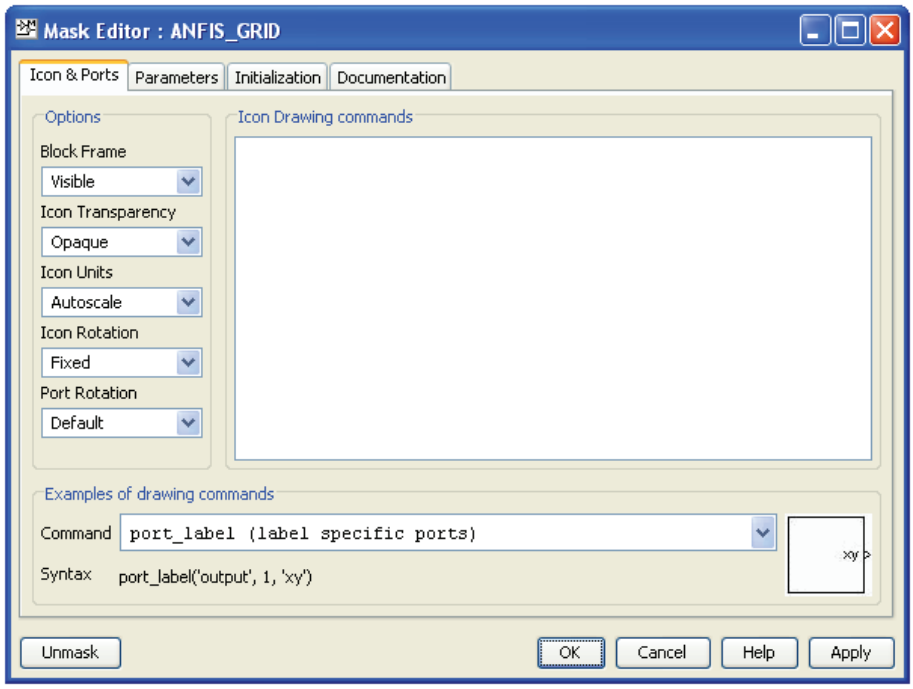


Figura AI.6. Ventana del Editor de Máscara del bloque ANFIS_GRID

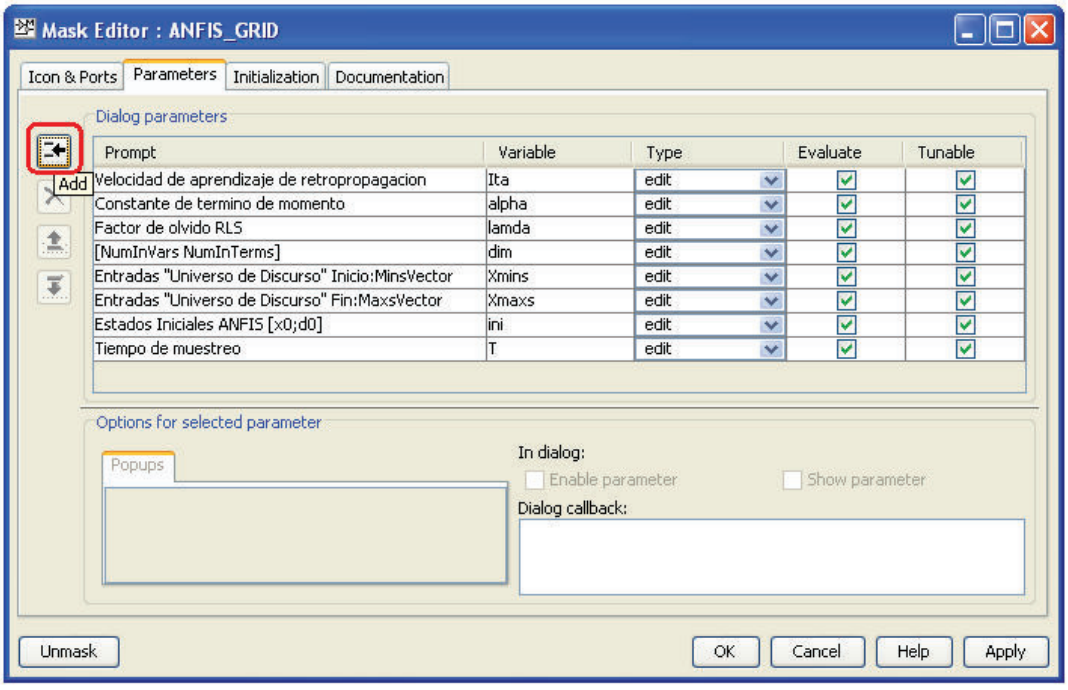


Figura AI.7. Ventana del Editor de Máscara – Creación de los parámetros.

En el panel *Initialization* se ha desarrollado un código de programación (ver Figura AI.8) que es capaz de determinar los parámetros iniciales del sistema ANFIS, tanto los parámetros no lineales de la Capa 1 como los parámetros lineales de la Capa 4. El objetivo de este código inicial es que la red neuronal empiece su proceso de aprendizaje partiendo de una información previa, con lo cual el tiempo de aprendizaje se acorta. Para ver el código completo de la inicialización revisar el Anexo III.

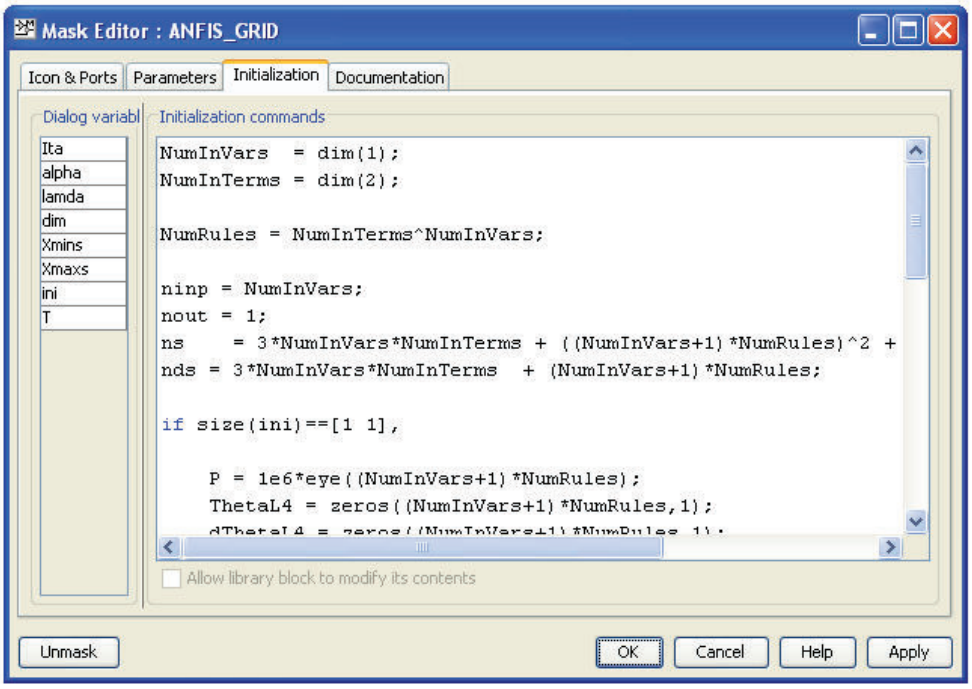


Figura AI.8. Ventana del Editor de Máscara – Definición del código de inicialización.

En el panel *Documentation* se ha realizado una pequeña descripción del subsistema, haciendo énfasis de que se trata de un bloque que representa el algoritmo de un sistema ANFIS (ver ventana de la Figura AI.9).

Una vez que se hayan realizado todas las configuraciones antes indicadas, se da clic en el botón OK. Para verificar que todos los cambios se han aplicado se da doble clic en el bloque ANFIS_GRID tras lo cual se despliega la ventana mostrada en la Figura AI.10. Según se observa se han creado satisfactoriamente todos los elementos y comentarios configurados en la ventana del editor de máscaras.

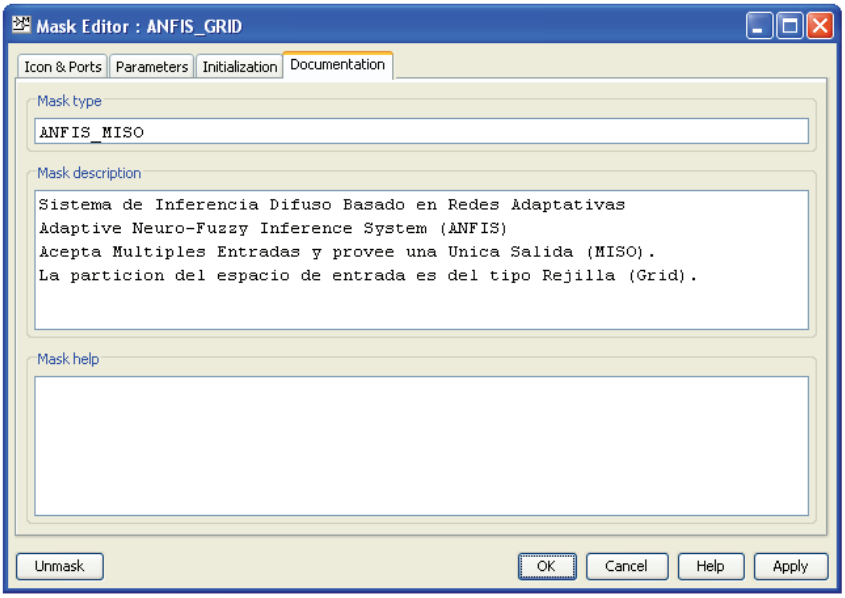


Figura AI.9. Ventana del Editor de Máscara – Descripción de la máscara.

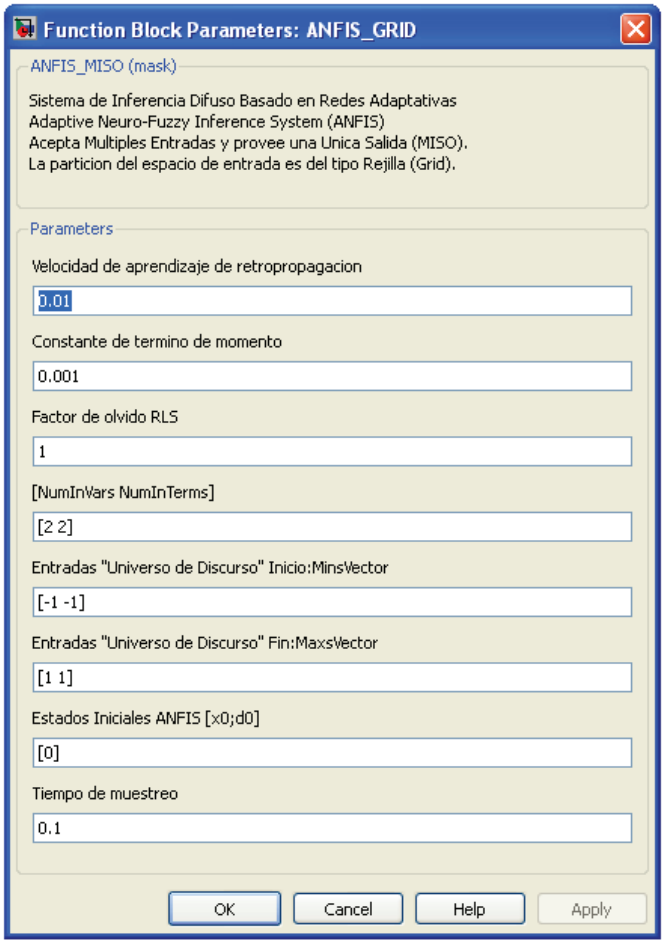


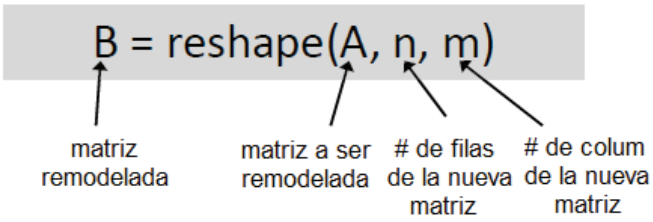
Figura AI.10. Ventana que se crea después de aplicar el enmascaramiento.

En los recuadros en blanco de la Figura AI.10 es donde se configuran los valores de los parámetros adicionales del bloque S-Function que posteriormente serán pasados a la función *anfisim_grid.m* y que son fáciles de identificar gracias a la descripción provista en la parte superior de cada recuadro.

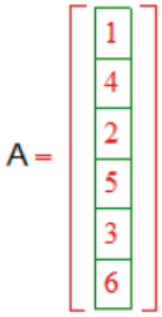
La máscara creada es factible de ser modificada o borrada, simplemente haciendo clic derecho en el menú contextual del bloque ANFIS_GRID y seleccionando *Edit Mask*.

ANEXO II
FUNCIÓN RESHAPE DE MATLAB

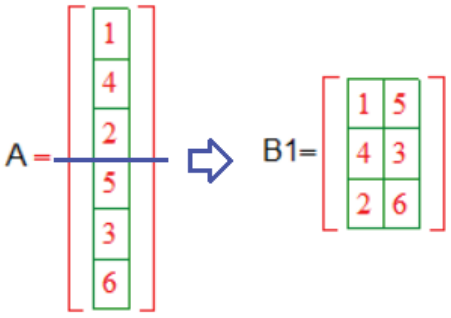
La función reshape (o remodelar en español) devuelve una nueva matriz B con n filas y m columnas, donde el producto n x m debe ser igual al número total de elementos de la matriz original A. La nueva matriz B cuenta con los mismos elementos que la original A. La estructura de esta función está dada por:



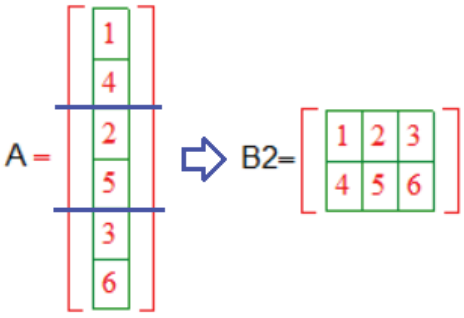
Si el producto n x m no es igual al número total de elementos de matriz original, entonces Matlab lanzará un aviso de error. A continuación se realizan varios ejemplos en donde se aplica la función reshape para remodelar una matriz original A = [1; 4; 2; 5; 3; 6]



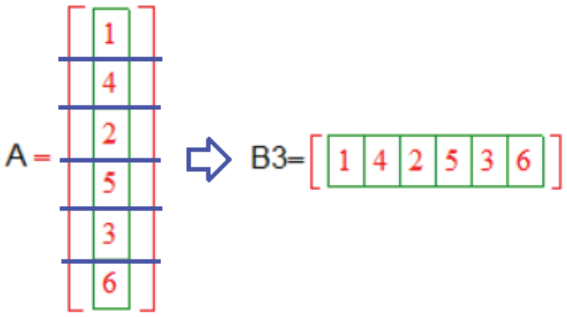
B1 = reshape (A, 3, 2)



B2 = reshape (A, 2, 3)

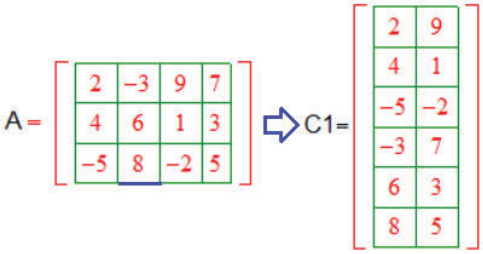


B3 = reshape (A, 1, 6)

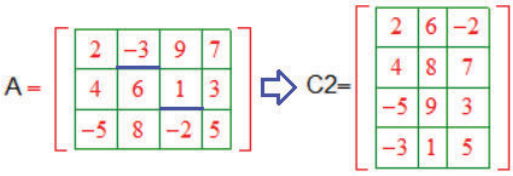


Como se observa en los ejemplos anteriores, la reordenación de los elementos se la realiza a partir de recortes basados en el número de filas requeridas en la nueva matriz y siguiendo el orden secuencial de los elementos de las columnas de la matriz original. Los siguientes ejemplos refuerzan lo mencionado.

C1 = reshape (A, 6, 2)



C2 = reshape (A, 4, 3)



ANEXO III

ALGORITMO DE PROGRAMACIÓN DE LA RED ANFIS

```
function [out,Xt,str,ts] =
anfisim_grid(Ti,Xt,u,flag,Ita,alpha,lamda,NumInVars,NumInTerms,x0,T)
```

```
% Este programa es una implementación de un sistema ANFIS (MISO) en linea (on-line).
% La estructura de la red neuronal es determinada y configurada por el usuario.
% El espacio de entrada es particionado usando el metodo tipo rejilla (grid).
% Los parámetros de premisa (no-lineales) en la Capa 1 son estimados por el método del
% Gradiente Descendente a traves de la Retropropagación del Error.
% Los parámetros del consecuente (lineales)en la Capa 4 son estimados por
% el algoritmo del Estimador de Minimos Cuadrados Recursivo (RLSE)

NumRules = NumInTerms^NumInVars;  % Número de reglas difusas para una particion tipo
Rejilla (Grid).
```

---------- % Información Inicial ----------

```
if flag == 0
```

```
        ninps = NumInVars+2;  % Número de entradas para el bloque S-Function: [ x e LE ]
        ns = 3*NumInVars*NumInTerms + ((NumInVars+1)*NumRules)^2 + (NumInVars+1)*NumRules;
%Número de estados de la red neuronal
        nds = 3*NumInVars*NumInTerms + (NumInVars+1)*NumRules;  %Número de estados
diferenciales de la red neuronal

        out = [0,ns+nds,1+ns+nds,ninps,0,1,1];  % #estados continuos, #estados discretos,
#salidas, #entradas, sin uso, f, #ts
        str = [];                               % Cuando flag=0 esta variable siempre se
establece como una matriz vacia
        ts = T;                                 % Especificación del tiempo de muestreo
        Xt = x0;                                % Vector columna con los valores de los
estados iniciales (parámetros de premisa y consecuente)
```

----------Actualización de Estados Discretos----------

```
elseif flag == 2
```

```
    x = u(1:NumInVars);
    e = u(NumInVars+1);
    learning = u(NumInVars+ 2);

if learning == 1
   %DETERMINACIÓN DE LOS PARÁMETROS QUE DEFINEN LA POSICIÓN Y LA FORMA DE LA
   %CAMPANA (PARÁMETROS DE PREMISA O PARÁMETROS NO LINEALES)
   off=1;
   off_end=NumInVars*NumInTerms;
   mean1=reshape(Xt(off:off_end),NumInVars,NumInTerms);  %Matriz de parámetros c
```

```
   off=off_end+1;
   off_end=off + NumInVars*NumInTerms-1;
   sigma1=reshape(Xt(off:off_end),NumInVars,NumInTerms); %Matriz de parámetros a

   off=off_end+1;
   off_end=off+NumInVars*NumInTerms-1;
   b1=reshape(Xt(off:off_end),NumInVars,NumInTerms);     %Matriz de parámetros b

   %MATRICES NECESARIAS PARA APLICAR LA REGLA DE APRENDIZAJE HÍBRIDA
   off=off_end+1;
   off_end=off + ((NumInVars+1)*NumRules)^2-1;
   P=reshape(Xt(off:off_end),(NumInVars+1)*NumRules,(NumInVars+1)*NumRules); % Matriz P
   off=off_end+1;
   off_end=off + (NumInVars+1)*NumRules-1;
   ThetaL4 = Xt(off:off_end);             %Vector de los parámetros lineales de la Capa 4

   off=off_end+1;
   off_end=off + NumInVars*NumInTerms-1;
   dmean1=reshape(Xt(off:off_end),NumInVars,NumInTerms);    % Presente con fines de
crecimiento futuro.

   off=off_end+1;
   off_end=off + NumInVars*NumInTerms-1;
   dsigma1=reshape(Xt(off:off_end),NumInVars,NumInTerms);   % Presente con fines de
crecimiento futuro.

   off=off_end+1;
   off_end=off + NumInVars*NumInTerms-1;
   db1=reshape(Xt(off:off_end),NumInVars,NumInTerms);       % Presente con fines de
crecimiento futuro.

   off=off_end+1;
   off_end=off + (NumInVars+1)*NumRules-1;
   dThetaL4 = Xt(off:off_end);  % Presente con fines de crecimiento futuro. No juega ningún
papel en esta versión.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %               OPERACIÓN HACIA ADELANTE            %
  %                                                   %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % CAPA 1: Capa de Fusificación
    In1 = x*ones(1,NumInTerms);
    Out1 = 1./(1 + (abs((In1-mean1)./sigma1)).^(2*b1));

  % CAPA 2: Capa del Antecedente de la Regla
    precond = combinem(Out1);
    Out2 = prod(precond,2)';
    S_2 = sum(Out2);

  % CAPA 3: Capa de Normalización de la Regla
    if S_2~=0
        Out3 = Out2/S_2;
    else
        Out3 = zeros(1,NumRules);
    end
```

```
  % CAPA 4: Capa del Consecuente de la Regla
    Aux1 = [x; 1]*Out3;

  % Nuevos datos de entrenamiento en forma de un vector columna.
    a = reshape(Aux1,(NumInVars+1)*NumRules,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SECCIÓN DE APRENDIZAJE DE PARÁMETROS         %
%               RETROPROPAGACIÓN DEL ERROR               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % CAPA 4.
    ThetaL4_mat = reshape(ThetaL4,NumInVars+1,NumRules);

  % Algoritmo de Retropropagación del Error
  % CAPA 3
    e3 = [x' 1]*ThetaL4_mat*e;

  % CAPA 2
    denom = S_2*S_2;
    ThetaE32 = zeros(NumRules,NumRules);
    if denom~=0
       for k1=1:NumRules
            for k2=1:NumRules
                 if k1==k2
                    ThetaE32(k1,k2) = ((S_2-Out2(k2))/denom)*e3(k2);
                 else
                    ThetaE32(k1,k2) = -(Out2(k2)./denom)*e3(k2);
                 end
            end
       end
    end

  % Suma ThetaE32 a lo largo de las filas para encontrar la contribución de cada
  % nodo de la capa 3 (indexado por k2) a un solo nodo de la capa 2(indexado por k1).

    e2 = sum(ThetaE32,2);

  % CAPA 1
    Q = zeros(NumInVars,NumInTerms,NumRules);
    for i=1:NumInVars
         for j=1:NumInTerms
              for k=1:NumRules
                   if Out1(i,j)== precond(k,i) && Out1(i,j)~=0
                       Q(i,j,k) = (Out2(k)/Out1(i,j))*e2(k);
                   end
               end
          end
     end

    ThetaE21 = sum(Q,3);

   % AJUSTE DE LOS PARÁMETROS DE PREMISA DE LA CAPA1 POR EL GRADIENTE DESCENDENTE
    if isempty(find(In1==mean1, 1))

        deltamean1 = ThetaE21.*(2*b1./(In1-mean1)).*Out1.*(1-Out1);
        deltab1 = ThetaE21.*(-2).*log(abs((In1-mean1)./sigma1)).*Out1.*(1-Out1);
```

```
        deltasigma1 = ThetaE21.*(2*b1./sigma1).*Out1.*(1-Out1);

        dmean1 = Ita*deltamean1 + alpha*dmean1;
        mean1 = mean1 + dmean1;

        dsigma1 = Ita*deltasigma1 + alpha*dsigma1;
        sigma1 = sigma1 + dsigma1;

        db1 = Ita*deltab1 + alpha*db1;
        b1 = b1 + db1;

       % Ordena los términos en la Capa 1.
           for j=1:NumInTerms-1
                if any(mean1(:,j)>mean1(:,j+1))
                   for i=1:NumInVars
                        [mean1(i,:) index1] = sort(mean1(i,:));
                        sigma1(i,:) = sigma1(i,index1);
                        b1(i,:) = b1(i,index1);
                   end
                end
           end
    end

  % Fijación de los parámetros del consecuente por el RLSE
    P = (1./lamda).*(P - P*(a*a')*P./(lamda+a'*P*a));
    ThetaL4 = ThetaL4 + P*a.*e;

  %%%%%%%%% FINAL DEL PROCESO DE APRENDIZAJE DE LOS PARÁMETROS %%%%%%%%%
  % Almacenamiento del vector de estado
  % Xt = [mean1 sigma1 b1 P ThetaL4 dmean1 dsigma1 db1 dThetaL4];

    Xt = [ reshape(mean1,NumInVars*NumInTerms,1);
             reshape(sigma1,NumInVars*NumInTerms,1);
             reshape(b1,NumInVars*NumInTerms,1);
             reshape(P,((NumInVars+1)*NumRules)^2,1);
             ThetaL4;
             reshape(dmean1,NumInVars*NumInTerms,1);
             reshape(dsigma1,NumInVars*NumInTerms,1);
             reshape(db1,NumInVars*NumInTerms,1);
             dThetaL4;];

end   % fin del bucle "if learning==1"

out=Xt;
```

---------- Salidas ----------

```
elseif flag == 3

 % Descomprimir los primeros parámetros de la red...
   off=1;
   off_end=NumInVars*NumInTerms;
   mean1=reshape(Xt(off:off_end),NumInVars,NumInTerms);

   off=off_end+1;
```

```
    off_end=off + NumInVars*NumInTerms-1;
    sigma1=reshape(Xt(off:off_end),NumInVars,NumInTerms);

    off=off_end+1;
    off_end=off+NumInVars*NumInTerms-1;
    b1 =reshape(Xt(off:off_end),NumInVars,NumInTerms);

    off=off_end+1;
    off_end=off + ((NumInVars+1)*NumRules)^2 - 1;
    % P = reshape(Xt(off:off_end),(NumInVars+1)*NumRules,(NumInVars+1)*NumRules);

    off=off_end+1;
    off_end=off + (NumInVars+1)*NumRules - 1;
    ThetaL4 = Xt(off:off_end);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %            OPERACIÓN HACIA ADELANTE                 %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % CAPA 1: Capa de Fusificación
    x = u(1:NumInVars);
    In1 = x*ones(1,NumInTerms);
    Out1 = 1./(1 + (abs((In1-mean1)./sigma1)).^(2*b1));

  % CAPA 2: Capa del Antecedente de la Regla
    precond = combinem(Out1);
    Out2 = prod(precond,2)';
    S_2  = sum(Out2);

  % CAPA 3: Capa de Normalización de la Regla
    if S_2~=0
        Out3 = Out2/S_2;
    else
        Out3 = zeros(1,NumRules);
    end

  % CAPA 4: Capa del Consecuente de la Regla
    Aux1 = [x; 1]*Out3;
    a = reshape(Aux1,(NumInVars+1)*NumRules,1);  % Nuevos datos de entrenamiento en forma
de un vector columna.

  % Capa 5: Capa de Inferencia de la Regla (Suma de nodos)
    outact = a'*ThetaL4;

  % Formacion del vector de salidas del bloque S-Function
    out=[outact;Xt];

else
   out=[];
end
```