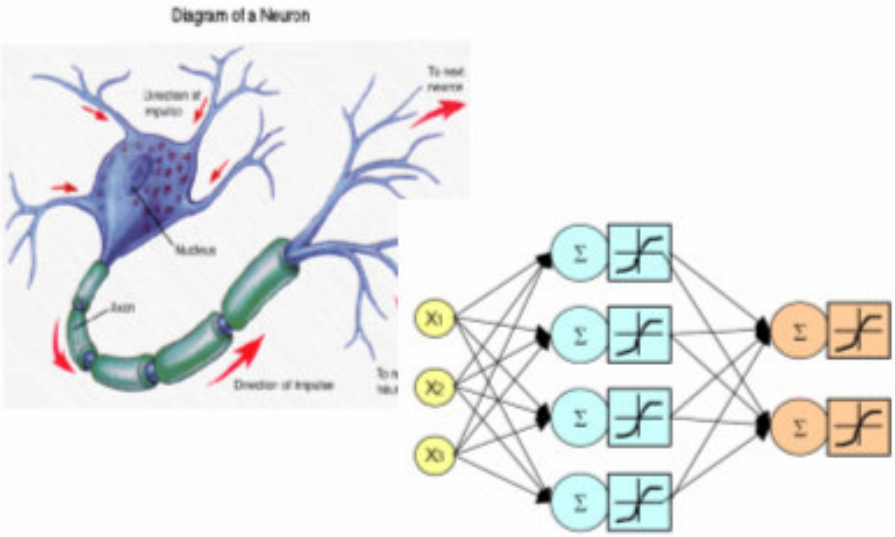
SQM

Departamento de Ingeniería Química
Universidad de Santiago de Chile

Curso de Capacitación
Fundamentos y Aplicaciones de Redes Neuronales en Ingenieria de Procesos



RELATOR DR. FRANCISCO CUBILLOS M.

CAPACITACION USACH **2007**

CAPITULO 1 INTRODUCCION

1.1 INTELIGENCIA ARTIFICIAL Y REDES NEURONALES

La inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión artificial, demostración de teoremas, procesamiento de información expresada mediante lenguajes humanos... etc. Algunos desarrollos de la inteligencia artificial se han transformado en poderosas herramientas de apoyo a las más diversas actividades humanas, entre las más conocidas podemos citar los Sistemas Expertos, Logica Difusa (Fuzzy Logic) y Redes Neuronales.

Las redes neuronales son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Si se examinan con atención aquellos problemas que no pueden expresarse a través de un algoritmo, se observará que todos ellos tienen una característica en común: la experiencia. El hombre es capaz de resolver estas situaciones acudiendo a la experiencia acumulada. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana. En definitiva, las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia.

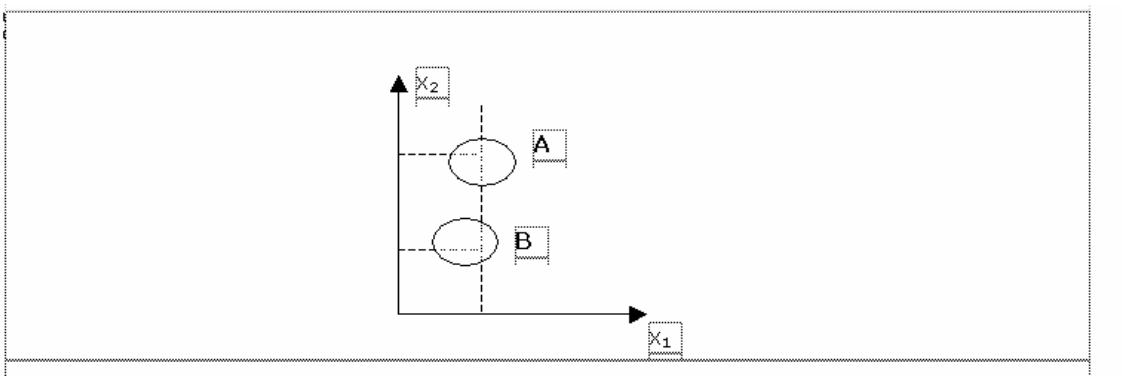
Una red neuronal es“un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona”.

Las Redes Neuronales constituyen una familia muy variada de arquitecturas. Están basadas en el modelo cerebral: las neuronas establecen conexiones entre ellas (sinapsis), de manera que cuando un animal recibe un estímulo, ciertas conexiones se refuerzan más que otras, provocando una cierta respuesta. Siempre que el animal reciba un estímulo (entrada) similar, generará la misma respuesta (aprendizaje): se puede decir que el cerebro reconoce diferentes patrones.

Este comportamiento es fácilmente caracterizable mediante un modelo matemático (simulación). El tratamiento de la información (computación) no va a ser el tradicional: se basa en la evolución temporal del sistema y en la interpretación de ciertos parámetros (información). El sistema se compone de un número elevado de unidades muy simples (neuronas) altamente interconectadas: el paralelismo es masivo. Se puede decir que una neurona es un tipo de autómata (sistema dinámico), de ahí el carácter temporal. Las Redes Neuronales artificiales pueden aprender modificando el "peso" de las conexiones entre las unidades; así es posible distinguir patrones.

La idea de las Redes Neuronales es definir una función a partir de la cual poder distinguir patrones con los datos de salida: dependiendo del valor obtenido catalogamos la entrada como perteneciente a un cierto grupo. El tipo de entradas y su número determinará la capacidad de discriminación de la Red.

En el ejemplo siguiente, una entrada (x_1) no es suficiente para determinar si estamos en el patrón A o el B. Con otra entrada (x_2) ya somos capaces de discriminar.



1.2 HISTORIA

McCullogh y Pitts realizaron en 1943 un estudio biológico del cerebro obteniendo un modelo formal de neurona, con lo que introdujeron así el concepto de umbral: una neurona responde a un cierto estímulo siempre que éste sobrepase un cierto umbral de activación. Posteriormente, en 1949, Hebb desarrolló el *Hebbian Learning:* aprendizaje mediante adaptación de sinapsis o reforzamiento de las conexiones.

En 1959, Rosenblat definió el perceptrón, uno de los conceptos más importantes dentro del desarrollo de las Redes Neuronales: el perceptrón consiste en una estructura más una regla de aprendizaje o regla del perceptrón. Esa estructura es la combinación de una neurona y una función de salida que es la que define el umbral de activación. La misión de la neurona es implementar una combinación lineal de las entradas. Cada entrada posee un peso, que se adapta temporalmente. Es esto lo que se conoce como aprendizaje.

Misky y Papert desarrollaron en 1969 un perceptrón unicapa que conseguía una clasificación de primer orden (XOR). Se planteó entonces el problema del entrenamiento de varias capas. Así, en 1974 Werbos definió el algoritmo de retropropagación y el uso de la función sigmoidal como función de salida de un perceptrón. El algoritmo de retropropagación permite modificar los pesos partiendo de la última capa hasta la inicial basándose en el error cometido en la iteración anterior. Ese error es la diferencia entre la salida de la Red Neuronal y la salida real que deberíamos haber obtenido. Como el algoritmo de retropropagación está basado en la derivada del error, se decidió utilizar la función sigmoidea en vez del escalón para representar el umbral de activación (la función escalón tiene derivada infinita en el origen).

Posteriormente se desarrollaron otros tipos de redes: Kohonen en los 70 creó los mapas topológicos y las memorias asociativas, y en 1982 Hopfield definió las redes de Hopfield.

Finalmente, en 1986 Rumelhart y McClelland desarrollaron el perceptrón multicapa, popularizándose así el algoritmo de retropropagación. En 1989, Cybenko, Hornik et al. y Funahashi definieron el perceptrón multicapa como el aproximador universal.

1.3 CLASIFICACION DE LAS REDES NEURONALES

Existen diferentes criterios de clasificación para las Redes Neuronales:

- **Caracterización temporal:** se refiere a la caracterización temporal de las entradas.
  - **Continua en el tiempo:** cuando la función de entrada es una función continua. Este tipo de redes se utiliza en sistemas donde es necesario una clasificación inmediata de las entradas, como por ejemplo un sistema de alarma que se active ante la presencia de ciertos parámetros peligrosos.
  - **Discreta:** cuando las entradas se toman en determinados instantes de tiempo. Se utilizan en sistemas donde no es necesario un control inmediato de las entradas y

sólo se busca ver su evolución sin saturar demasiado el sistema con una gran cantidad de datos.

- **Entrada:** se refiere a los valores que toman las entradas.
- **Binaria:** cero o uno. Se puede utilizar cuando las entradas son codificadas, por ejemplo, representando el intervalo sobre el que caen dentro de un rango. Si el rango se divide en *x* intervalos, cada entrada se representa con *x* valores, todos a cero menos uno, el que corresponde al intervalo de valores de la entrada.
- **Continua:** el valor de la entrada puede ser cualquiera dentro de un rango. Es posible definir diferentes rangos de variación.

- **Entrenamiento:** se refiere al aprendizaje seguido por la Red.
  - **Supervisado:** cuando se vigila la evolución de la Red. Por ejemplo, cuando se le ofrecen a la Red ejemplos señalando las salidas que se deberían obtener.
  - **Con recompensa/castigo:** cuando la Red acierta se le ofrece una recompensa. Cuando falla se le castiga. Así aprende cómo debe comportarse.
  - **No supervisado:** no se le dice a la Red lo que debe dar. Esto se suele utilizar en aquellos casos en que tenemos una serie de entradas y no sabemos cómo clasificarlas. Dependiendo de lo que se obtenga tras el entrenamiento tendremos un criterio de clasificación.
- **Realimentación:**
  - ***Feedforward*:** realimentación hacia delante. Es la estructura normal, donde las salidas de una capa se introducen en la siguiente.
  - ***Feedback:*** realimentación hacia atrás. Las salidas de ciertas capas se introducen en capas anteriores para que éstas sepan cuál ha sido el comportamiento posterior y se adapten en consecuencia.

Entre los modelos más representativos se encuentran los mapas asociativos, el perceptrón multicapa, las redes de Hopfield (memorias autoasociativas), los mapas topológicos autoorganizativos de Kohonen y las redes hebbianas

Caracterización de una Red Neuronal

Los tres puntos clave para desarrollar una Red Neuronal son:

- Caracterización de la neurona (unidad básica de computación).
- Definición de una topología de interconexión.
- Definición de unas reglas de aprendizaje.

1.4 USOS DE LAS REDES NEURONALES

Ventajas que ofrecen las redes neuronales: Debido a su constitución y a sus fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas. Entre las ventajas se incluyen:

- Aprendizaje Adaptativo. Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.

_ Auto-organización. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.

- Tolerancia a fallos. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.

_ Operación en tiempo real. Los cómputos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

_ Fácil inserción dentro de la tecnología existente. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

Usos: Las redes neuronales pueden utilizarse en un gran número y variedad de aplicaciones, tanto comerciales como militares. Se pueden desarrollar redes neuronales en

un periodo de tiempo razonable, con la capacidad de realizar tareas concretas mejor que otras tecnologías. Cuando se implementan mediante hardware (redes neuronales en chips VLSI), presentan una alta tolerancia a fallos del sistema y proporcionan un alto grado de paralelismo en el procesamiento de datos. Esto posibilita la inserción de redes neuronales de bajo coste en sistemas existentes y recientemente desarrollados. Hay muchos tipos diferentes de redes neuronales; cada uno de los cuales tiene una aplicación particular más apropiada. Algunas aplicaciones comerciales son:

_ Biología: Aprender más acerca del cerebro y otros sistemas.- Obtención de modelos de la retina.

_ Producción : Evaluación de probabilidad de formaciones geológicas y petrolíferas. - Identificación de candidatos para posiciones específicas.- Explotación de bases de datos.- Optimización de plazas y horarios en líneas de vuelo.- Optimización del flujo del tránsito y control de semáforos.- Reconocimiento de caracteres escritos. - Modelación de sistemas para automatización y control.

_ Medio ambiente: - Analizar tendencias y patrones de contaminantes.- Previsión del tiempo – trayectoria de huracanes

_ Finanzas: - Previsión de la evolución de los precios. - Valoración del riesgo de los créditos.- Identificación de falsificaciones. - Interpretación de firmas.

Manufactura: - Robots automatizados y sistemas de control (visión artificial y sensores

de presión, temperatura, gas, etc.). - Control de producción en líneas de procesos.- Inspección de la calidad.

_ Medicina:- Analizadores del habla para ayudar en la audición de sordos profundos.- Diagnóstico y tratamiento a partir de síntomas y/o de datos analíticos (electrocardiograma, encefalogramas, análisis sanguíneo, etc.). - Monitorización en cirugías.- Predicción de reacciones adversas en los medicamentos.- Entendimiento de la causa de los ataques cardíacos.

_ Militares: - Clasificación de las señales de radar.- Creación de armas inteligentes.- Optimización del uso de recursos escasos.- Reconocimiento y seguimiento de objetivos.

La mayoría de estas aplicaciones consisten en realizar un reconocimiento de patrones, como ser: buscar un patrón en una serie de ejemplos, clasificar patrones, completar una señal a partir de valores parciales o reconstruir el patrón correcto partiendo de uno distorsionado. Sin embargo, está creciendo el uso de redes neuronales en distintos tipos de sistemas de control. Desde el punto de vista de los casos de aplicación, la ventaja de las redes neuronales reside en el procesado paralelo, adaptativo y no lineal. El dominio de aplicación de las redes neuronales también se lo puede clasificar de la siguiente forma: asociación y clasificación, regeneración de patrones, regresión y generalización, y optimización.

CAPITULO 2 REDES NEURONALES

2.1 MODELO DE NEURONA

Una neurona artificial se implementa de la siguiente manera: las entradas (x) se introducen en las dendritas. Cada una posee un peso (w_{ij}, neurona j – entrada i). Dentro del cuerpo se aplican diferentes funciones de procesamiento (propagación, activación, salida) hasta llegar al axón (salida y). Todo esto queda reflejado en la siguiente figura:

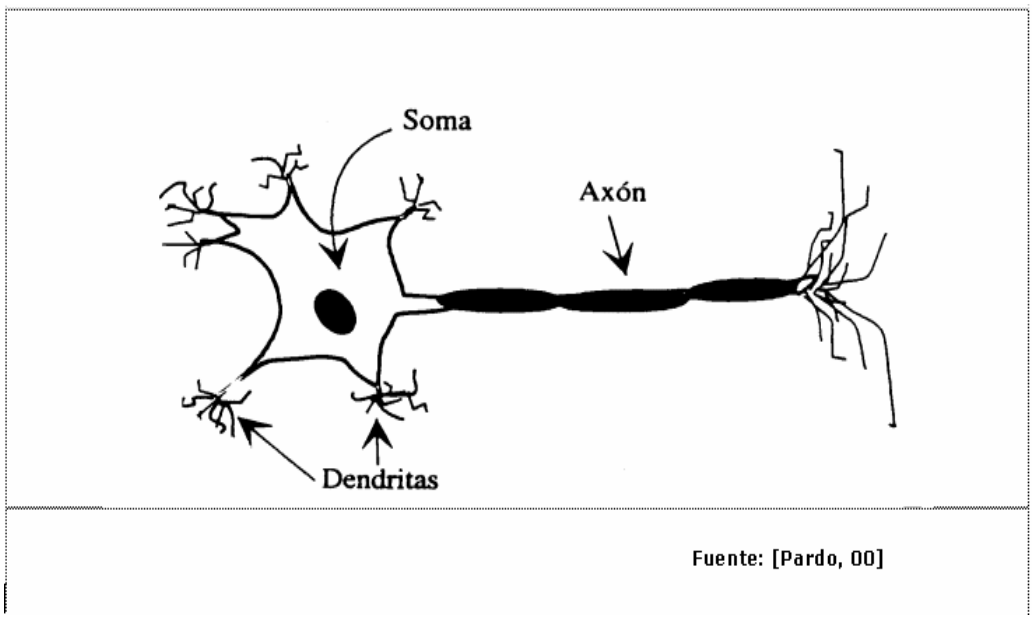


Figura Nº1(a) Neurona física.

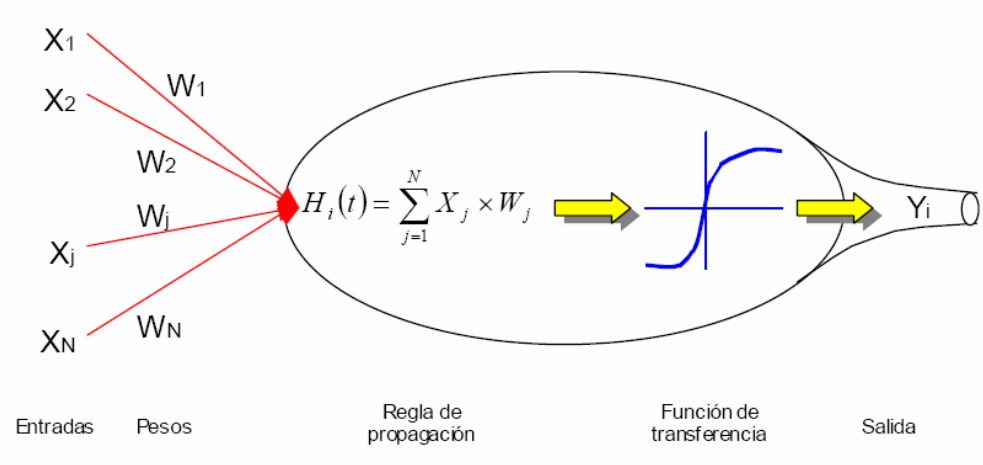


Figura Nº1(b) Representación matemática de una neurona.

En la práctica, las funciones de propagación y activación no suelen diferenciarse y constituyen una combinación lineal de las entradas o su distancia euclidiana lo que a veces se denomina "confluencia". La función de salida se establece fuera de la neurona.

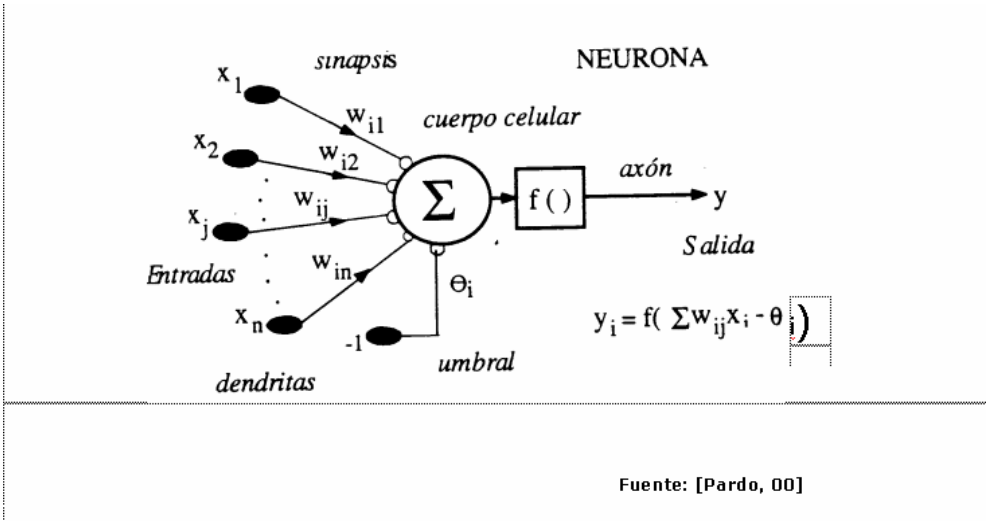


Figura Nº 2 Representación final de una neurona.

Aparte de las entradas propias de la neurona, se puede considerar un cierto umbral constante (q_i) o "bias" que ayuda a ajustar la salida deseada.

2.2 RED NEURONAL

Topología

La topología de refiere a la estructura de la red. Comprende varios aspectos:

- **Definición de conectividad:** se refiere a la forma de conexión entre las neuronas para formar una cierta estructura. De esta forma, podemos obtener redes monocapa o multicapa. Las redes monocapa consiguen una clasificación de primer orden como puede ser un XOR. Sin embargo, una red de varias capas puede diferenciar entre diferentes regiones, como se verá próximamente.
- **Temporización o sincronización del flujo de información:**
  - **Tiempo continuo-discreto:** se refiere a si la información se actualiza continuamente o sólo en ciertos instantes de tiempo.

- **Secuencia de cálculo:** flujo directo o lazos de realimentación. El flujo directo consiste en un mero paso de información de una capa a la siguiente (feeforward). Pero con la realimentación una capa obtiene información sobre lo que están haciendo las capas que la siguen, y así sucesivamente.

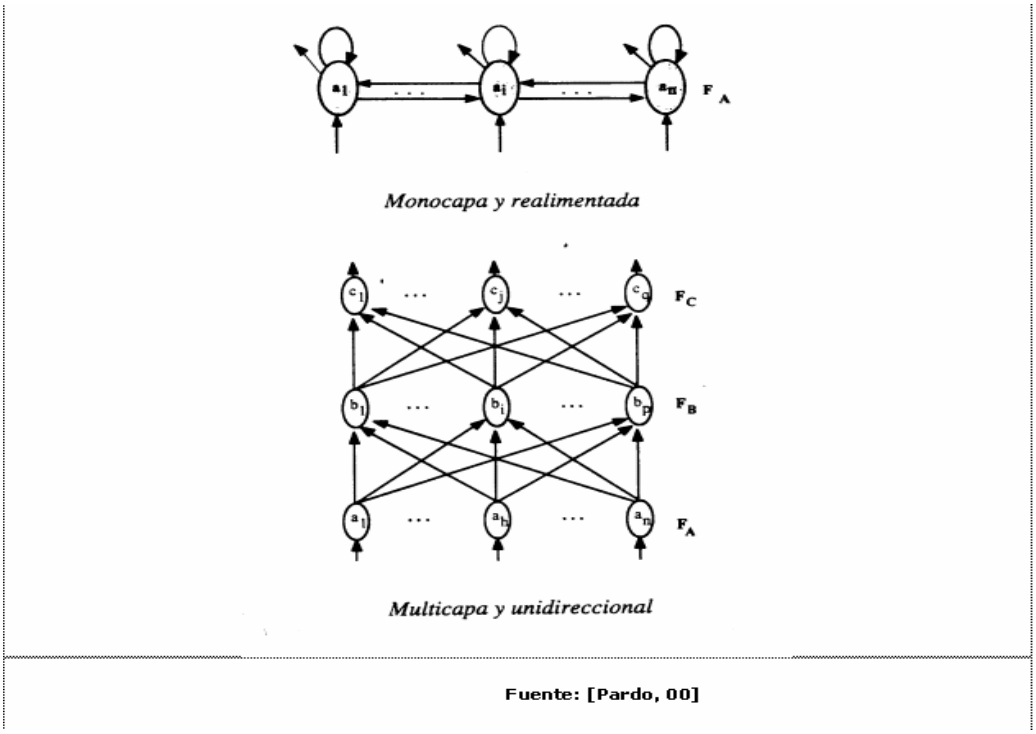


Figura 3 : Ejemplos de topologías.

En una red multicapa se definen tres tipos de capas: de entrada, oculta y de salida.

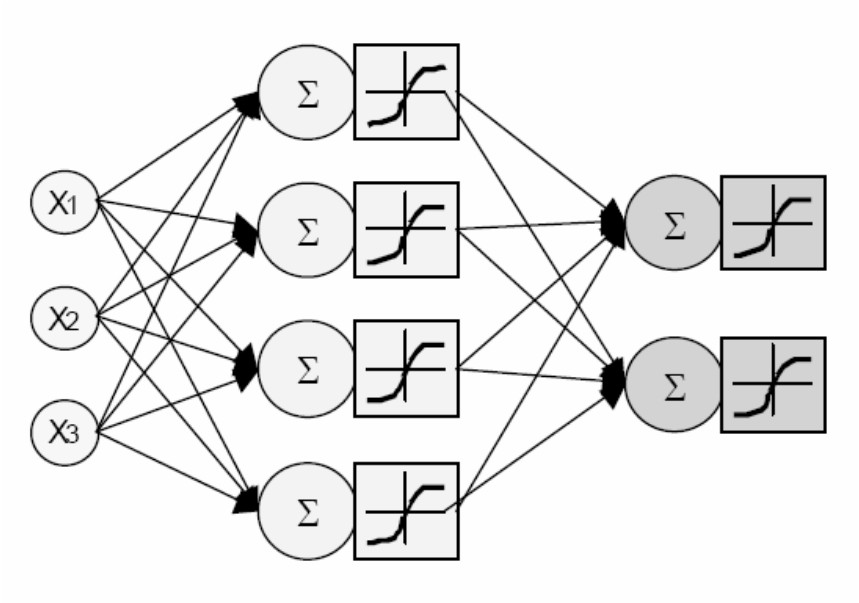


Figura Nº 4 : Red Neuronal de tres capas.

Dependiendo del número de capas ocultas que consideremos y moviéndonos dentro de un espacio bidimensional, se pueden definir regiones (conjunto de entradas que conforman un patrón) de diferentes formas, como se puede apreciar en la figura 5.

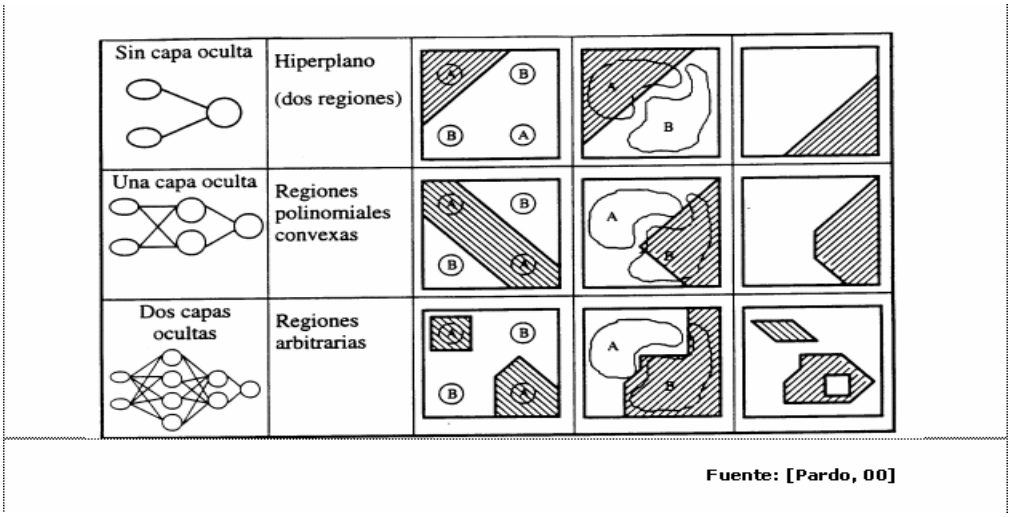


Figura Nº 5 : Regiones que se pueden delimitar según el número de Capas Ocultas que se consideren.

2.3 Entrenamiento o aprendizaje

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el mismo se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los sistemas biológicos existe una continua destrucción y creación de conexiones entre las neuronas. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma manera, una conexión se destruye cuando su peso pasa a ser cero.

Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por lo tanto, se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables ($dw_{ij}/dt = 0$). Un aspecto importante respecto al aprendizaje de las redes neuronales es el conocer cómo se modifican los valores de los pesos, es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información. Hay dos métodos de aprendizaje importantes que pueden distinguirse:

a- Aprendizaje supervisado.

b- Aprendizaje no supervisado.

Otro criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en considerar si la red puede aprender durante su funcionamiento habitual o si el aprendizaje supone la desconexión de la red, es decir, su inhabilitación hasta que el proceso termine. En el primer caso, se trataría de un aprendizaje *on line*, mientras que el segundo es lo que se conoce como *off line*. Cuando el aprendizaje es off line, se distingue entre una *fase de aprendizaje o entrenamiento* y una *fase de operación o funcionamiento*, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de test o prueba, que serán utilizados en la correspondiente fase. Además, los pesos de las conexiones permanecen fijos después que termina la etapa de entrenamiento de la red. Debido precisamente a su carácter estático, estos sistemas no presentan problemas de estabilidad en su funcionamiento. Una generalización de la fórmula o regla para decir los cambios en los pesos es la siguiente:

Peso Nuevo = Peso Viejo + Cambio de Peso

Aprendizaje no supervisado.

Las redes con aprendizaje no supervisado (también conocido como autosupervisado) no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta. Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada. Existen varias posibilidades en cuanto a la interpretación de la salida de estas redes, que dependen de su estructura y del algoritmo de aprendizaje empleado. En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada y las informaciones que se le han mostrado hasta entonces (en el pasado). En otro caso, podría realizar una clusterización (clustering) o establecimiento de categorías, indicando la red a la salida a qué categoría pertenece la información presentada a la entrada, siendo la propia red quien debe encontrar las categorías apropiadas a partir de las correlaciones entre las informaciones presentadas.

En cuanto a los algoritmos de aprendizaje no supervisado, en general se suelen considerar dos tipos, que dan lugar a los siguientes aprendizajes:

1) Aprendizaje hebbiano.

2) Aprendizaje competitivo y comparativo

Aprendizaje hebbiano.

Esta regla de aprendizaje es la base de muchas otras, la cual pretende medir la familiaridad o extraer características de los datos de entrada. El fundamento es una suposición bastante simple: si dos neuronas Ni y Nj toman el mismo estado simultáneamente (ambas activas o ambas inactivas), el peso de la conexión entre ambas se incrementa. Las entradas y salidas permitidas a la neurona son: {-1, 1} o {0, 1} (neuronas binarias). Esto puede explicarse porque la regla de aprendizaje de Hebb se originó a partir de la neurona biológica clásica, que solamente puede tener dos estados: activa o inactiva.

Aprendizaje competitivo y comparativo.

Se orienta a la clusterización o clasificación de los datos de entrada. Como característica principal del aprendizaje competitivo se puede decir que, si un patrón nuevo se determina que pertenece a una clase reconocida previamente, entonces la inclusión de este nuevo patrón a esta clase matizará la representación de la misma. Si el patrón de entrada se determinó que no pertenece a ninguna de las clases reconocidas anteriormente, entonces la estructura y los pesos de la red neuronal serán ajustados para reconocer la nueva clase.

Aprendizaje supervisado.

El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

En este tipo de aprendizaje se suelen considerar, a su vez, tres formas de llevarlo a cabo, que dan lugar a los siguientes aprendizajes supervisados:

1) Aprendizaje por corrección de error.

2) Aprendizaje por refuerzo.

3) Aprendizaje estocástico.

Aprendizaje por corrección de error.

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red, es decir, en función del error cometido en la salida.

Un ejemplo de este tipo de algoritmos lo constituye la *regla de aprendizaje del Perceptron*, utilizada en el entrenamiento de la red del mismo nombre que desarrolló Rosenblatt en 1958. Esta es una regla muy simple, para cada neurona en la *capa de salida* se le calcula la desviación a la salida objetivo como el error el cual luego se utiliza para cambiar los pesos sobre la conexión de la neurona precedente. El cambio de los pesos por medio de la regla de aprendizaje del Perceptron se realiza según la siguiente regla: $\Delta wij = ci*(aqi - outi)$;

donde: a*qi* es la salida deseada/objetivo de la neurona de salida *Ni*, (a*qi* – *outi*) la desviación objetivo de la neurona *Ni* y ci la tasa de aprendizaje.

Otro algoritmo muy conocido y que pertenece a esta clasificación es la *regla de aprendizaje Delta* o regla del mínimo error cuadrado (LMS Error: Least Mean Squared Error), que también utiliza la desviación a la salida objetivo, pero toma en consideración a todas las neuronas predecesoras que tiene la neurona de salida. Esto permite cuantificar el error global cometido en cualquier momento durante el proceso de entrenamiento de la red, lo cual es importante, ya que cuanta más información se tenga sobre el error cometido, más rápido se puede aprender. Luego el error calculado (δ) es igualmente repartido entre las conexiones de las neuronas predecesoras. Por último se debe mencionar la *regla de aprendizaje de propagación hacia atrás o de backpropagation*, también conocido como regla LMS multicapa, la cual es una generalización de la regla de aprendizaje Delta. Esta es la primera regla de aprendizaje que permitió realizar cambios sobre los pesos en las conexiones de la capa oculta.

Aprendizaje por refuerzo.

Se trata de un aprendizaje supervisado, más lento que el anterior, que se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado, es decir, de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada. En el aprendizaje por refuerzo la función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito = +1 o fracaso = -1), y en función de ello se ajustan los pesos.

Aprendizaje estocástico.

Consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad. En el aprendizaje estocástico se suele hacer una analogía heurística en términos de un comportamiento termodinámico, asociando a la red neuronal con un sólido físico que tiene cierto estado energético (simulating anealing), o al proceso evolutivo de los seres vivos (Algoritmos genéticos). Ambas técnicas son propias de los algoritmos de optimización meta-heuristicos.

CAPITULO 3 REDES MULTICAPAS

3.1 DEFINICION

Una red neuronal multicapa o (perceptron multicapa) es un conjunto de neuronas ordenadas en capas y conectadas convenientemente. Esta es la disposición más difundida dentro de las aplicaciones de redes neuronales. Sus caracteristicas son:

- Puede aproximar cualquier función continua Y = F (X) hasta una precisión dada.
- Generaliza la respuesta frente a nuevos patrones de entrada (basado en aproximación/interpolación).
- Idónea cuando sabemos qué queremos conseguir pero no cómo hacerlo y además disponemos de patrones de referencia: queremos que generalice a nuevos patrones.
- Aplicaciones:
  - Reconocimiento de patrones:
    - Reconocimiento de caracteres en imágenes.
    - Reconocimiento de fonemas en voz.
  - Predicción de series temporales (demandas, etc):
    - Identificación y control de sistemas.
    - Compresión de imágenes y reducción de dimensionalidad

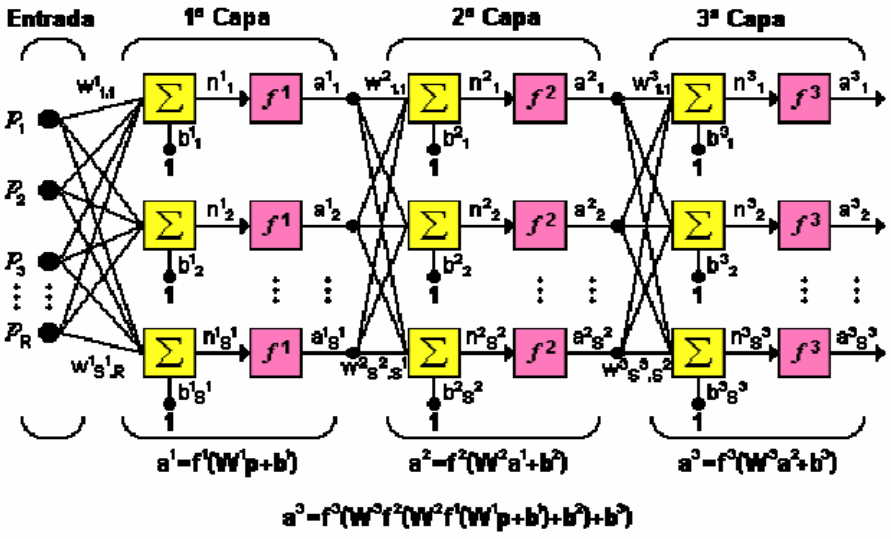


Figura Nº 6 : modelo de red multicapa

Este tipo de redes se caracterizan por su facilidad de implementación. Su aprendizaje se basa en la retropropagación: se parte de unos pesos iniciales en las conexiones interneuronales. Para un conjunto de entradas se obtiene una cierta salida. Basándose en que se conoce la salida que deberíamos haber obtenido (patrón catalogado – aprendizaje supervisado), calculamos el error. A partir de este error se modifican los pesos siguiendo el sentido inverso al de evolución de la Red (se parte de la salida hasta llegar a la entrada). De la misma manera se opera con el resto de entradas de entrenamiento. Se puede observar que el error irá disminuyendo a medida que se aplique el algoritmo.

Sin embargo un entrenamiento reiterado con las mismas entradas acaba provocando un sobre-entrenamiento a la Red Neuronal, memorizando características de un conjunto, impidiendo así que aprenda a generalizar. Por eso tras cada iteración hay que evaluar: introducir nuevos valores distintos a los de entrenamiento y calcular el error de salida. De esta manera se obtiene una función (error de evaluación) de la que nos interesa hallar su mínimo absoluto (puede haber mínimos locales). Determinando el número de iteraciones con que se alcanza dicho valor, nos aseguramos, en cierta forma, obtener un bajo error para cualquier conjunto de datos de entrada. Después, se puede aplicar un test con un conjunto nuevo de entradas que nos dará una medida de la capacidad de discriminación de la Red.

3.2 ESTRUCTURA DE UN PERCEPTRON

Un perceptrón es una estructura neuronal más una regla de aprendizaje. Como se explicaba anteriormente, una neurona se compone de un conjunto de entradas, cada una con un peso, que se introducen en el cuerpo de la misma para su procesamiento. Ese procesamiento puede consistir en:

- Combinación lineal:

$$y = \sum_i \omega_i x_i - \theta$$

- Distancia euclídea:

$$y = \sqrt{\sum_i (\omega_i x_i)^2} - \theta$$

Este resultado se introduce posteriormente en un bloque caracterizado por una de las siguientes funciones llamadas funciones de activación según se muestran en la siguiente tabla

Nombre	Relación Entrada /Salida	Icono	Función
Limitador Fuerte	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		*hardlim*
Limitador Fuerte Simétrico	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		*hardlims*
Lineal Positiva	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		*poslin*
Lineal	$a = n$		*purelin*
Lineal Saturado	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		*satlin*
Lineal Saturado Simétrico	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = +1 \quad n > 1$		*satlins*
Sigmoidal Logarítmico	$a = \frac{1}{1+e^{-n}}$		*logsig*
Tangente Sigmoidal Hiperbólica	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		*tansig*
Competitiva	$a = 1$ *Neurona con n max* $a = 0$ *El resto de neuronas*		*compet*

La función que representa el comportamiento ideal de una neurona es la función escalón: dependiendo de la entrada se activa la conexión entre neuronas (salida = 1) o no (salida = 0). Esta decisión puede depender de un cierto umbral (que la salida sea capaz de superarlo), representado en las expresiones anteriores por q: introducir este término negativo equivale a considerar que la función vale uno a partir de q y no a partir de cero como se representa en la tabla.

Aunque éste sea el comportamiento ideal, en la práctica se suele utilizar la función de tipo sigmoidal. Esto se debe a que es una función muy parecida al escalón y derivable en el entorno de cualquier punto del eje x. Como se verá, esto es necesario en el algoritmo de

retropropagación ya que está basado en la minimización del error con respecto al valor de los pesos, por lo que hay que derivar e igualar a cero.

El algoritmo de aprendizaje para un perceptrón aislado es el siguiente:

$$W_{k+1} = W_k - \eta \cdot X^t \left(y^d - y \right)$$

donde:

- W = vector de pesos.
- k = iteración actual.
- n = factor de aprendizaje.
- X = vector de entradas.
- y_d = salida deseada.
- y = salida obtenida en la iteración k.

El factor de aprendizaje n determina la rapidez del algoritmo pero también su exactitud. Cuanto mayor sea, necesitaremos menos iteraciones (antes alcanzaremos las cercanías del mínimo error de evaluación) pero el aprendizaje es muy grosero (es más probable que nos quedemos oscilando en las cercanías del mínimo error de evaluación que lo alcancemos realmente). Cuanto menor sea, más lento es pero más fino en el aprendizaje. Por lo tanto hay que llegar a un compromiso.

3.3 ALGORITMO DE RETROPROPAGACION

A continuación, se va a explicar el método utilizado para entrenar un perceptrón multicapa.

Algoritmo

El aprendizaje de un perceptrón multicapa es más complejo. Como se apuntó anteriormente, el entrenamiento de este tipo de Redes Neuronales es un entrenamiento supervisado. Se define un conjunto de pares de patrones (X_i, Y_i) de entrenamiento y se define una función de error (diferencia entre la salida deseada y la obtenida). Una vez obtenido dicho error se actualizan los pesos para minimizarlo. El procedimiento que se emplea es el descenso en la dirección del gradiente: una manera muy eficiente de implementarlo es a través de un

procedimiento equivalente a computar la Red hacia atrás. Esto da lugar al algoritmo de retropropagación:

$$E_T = \sum_{p=1}^{P} E_p = \frac{1}{2} \sum_{p=1}^{P} \left(d_p - O_p^S \right)^2$$

donde:

- E_T = error total de salida.
- E_p = error de la salida p.
- P = número de neuronas de la última capa.
- O_p^S = salida obtenida en la neurona p de la capa S (la de salida).
- d_p = salida esperada en la neurona p.

El algoritmo de aprendizaje es:

$$\omega_{ij}^L (k+1) = \omega_{ij}^L (k) - \mu \frac{\partial E_T}{\partial \omega_{ij}^L (k)}$$

donde:

- w_{ij}^L = peso de la entrada i de la neurona j en la capa L.
- k = iteración actual.
- m = factor de aprendizaje.

Pasos del algoritmo

Los pasos a seguir son:

1. Inicializar pesos y bias.
2. Presentar nuevo patrón y salida deseada.
3. Calcular salida de todos los nodos en red.
4. Calcular señales de error de todos los nodos: nodos de salida a nodos de primera capa.

5. Adaptar pesos en función de señales de error.
6. Ir al paso 2.

Mejoras y variantes

Se pueden introducir algunas mejoras a este algoritmo que pretenden asegurar la convergencia o hacer más rápido el proceso:

- **Inicialización de pesos:** enfocado a la convergencia.
- **Factor de aprendizaje:** enfocado a la convergencia.
- **Tamaño de la red:** enfocado a la generalización.
- **Término del momento:** enfocado a obtener mínimos locales en la función de error y a acelerar el proceso.

$$\omega_{ij}^{L}(k+1) = \omega_{ij}^{L}(k) - \mu \cdot \delta_{j}^{L} \cdot O_{j}^{L} + \alpha\left(\omega_{ij}^{L}(k) - \omega_{ij}^{L}(k-1)\right)$$

ALGORITMOS PARA ADAPTACION RECURSIVA DE REDES FFNN

Con la finalidad de adaptar recursivamente los pesos de una red neuronal *Feedforward*, debe ser convenientemente definida una función de costos que pondere la información pasada y la nueva información incorporada al sistema. Si suponemos que hemos muestreado el proceso por k instantes, podemos definir una función ponderada de costos entre los valores calculados por la red neuronal "**o**" y los deseados "**d**" según:

$$\mathbf{J_k} = \tfrac{1}{2} \sum_{i=1}^{k} \beta^{(k-i)} * \left((\underline{\mathbf{o}}-\underline{\mathbf{d}})^{t} * (\underline{\mathbf{o}}-\underline{\mathbf{d}}) \right)_i$$

donde $\beta \leq 1$, es una constante llamada factor de olvido.

Si incorporamos el próximo par de datos en el instante (k+1) tendrenos que

$$\boldsymbol{J_{k+1}} = \beta * \boldsymbol{J_k} + \tfrac{1}{2} \left((\underline{\mathbf{o}}-\underline{\mathbf{d}})^{t} * (\underline{\mathbf{o}}-\underline{\mathbf{d}}) \right)_{k+1}$$

A partir de este resultado puede ser obtenida la siguiente relación recursiva para el cálculo del gradiente :

$$\underline{\nabla}(\boldsymbol{J})_{k+1} = \beta * \underline{\nabla}(\boldsymbol{J})_k - \left([\Psi]^{t} * (\underline{\mathbf{d}}-\underline{\mathbf{o}}) \right)_{k+1}$$

Donde [**Ψ**] es la matriz jacobiana de **y** (salidas de la red) con respecto a los parámetros, que puede ser calculada por el método de Retropropagación .

La forma en que la matriz Hessiana inversa es determinada da origen a una variada serie de algoritmos de adaptación, llamados de segundo orden, tales como Gradiente Conjugado, Levemberg Marquardt, BFGS, etc.

3.4 ESTRUCTURAS USADAS EN INGENIERIA DE PROCESOS

RED NEURONAL FEEDFORWARD

En aplicaciones de ingeniería de procesos, que involucra fundamentalmente la tarea de aproximación funcional, la estructura más frecuentemente usada es la red neuronal tipo perceptron multicapa *Feedforward* (**FFNN**) con una capa oculta, producto interno como confluencia y sigmoides como funciones de activación, debido a que fue probado que es capaz de aproximar cualquier función no lineal contínua. En la arquitectura *Feedforward*, las salidas de los nodos de una capa alimentan en forma directa a los nodos de la capa adyacente. Tradicionalmente, este tipo de red es entrenado usando el método de Retropropagación (*Backpropagation*), sin embargo, por ser un método de primer orden, converge lentamente que lo hace prohibitivo para aplicaciones de control en tiempo real. Para mejorar la velocidad de entrenamiento de utilizan métodos de segundo orden de tipo batch o recursivos.

Las características y estructura de esta red se encuentran en la sección anterior.

Debido a la popularidad de este tipo de red, cuando en alguna aplicación de red neuronal no se especifica la estructura, se asume por defecto que se ha usado este tipo.

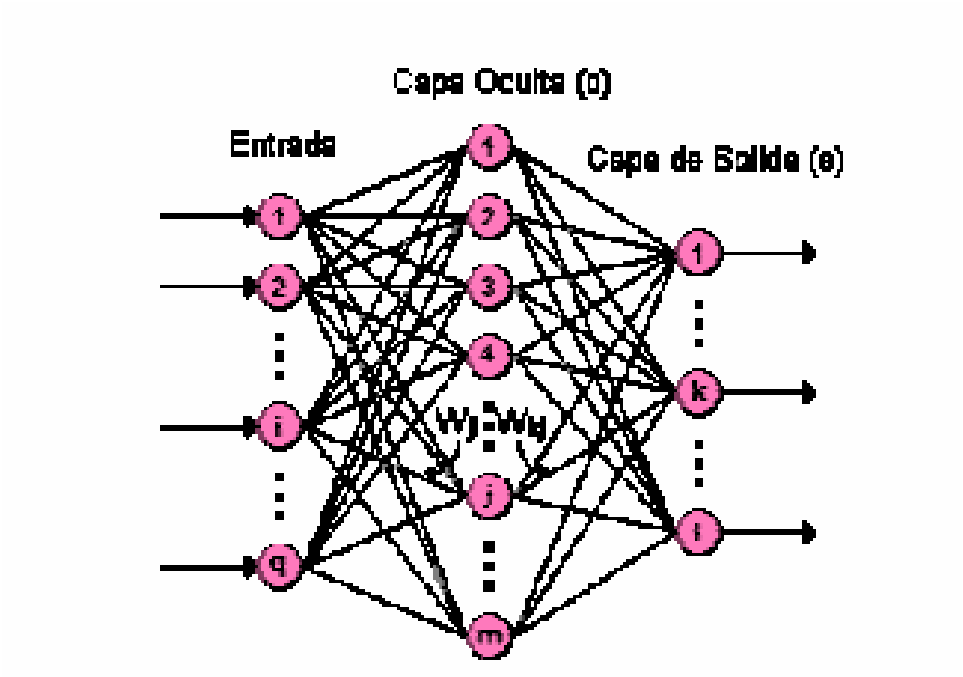


Figura Nº 7 : Red neuronal Feedforward con una capa oculta

REDES CON FUNCIONES DE BASE RADIAL (RBF)

TOPOLOGIA Las redes con funciones de base radial **RBF** (Radial Basis Functions) son antecesoras de las redes neuronales *Feedforward* y son usadas tradicionalmente como un método de interpolación de espacios multidimensionales. Actualmente, y dada su estructura, son consideradas como un tipo de red neuronal y han sido usadas exitosamente en identificación y control de procesos

Estas redes son una buena alternativa a las redes tipo multicapa en identificación de procesos, dado que las salidas de una red **RBF** son lineales con respecto a sus parámetros.

Las redes tipo **RBF**,consisten en una capa de vários nodos donde la función de activación actúa sobre la norma euclidiana de la diferencia entre el vector de entrada $\underline{\mathbf{x}}$ y un vector $\underline{\boldsymbol{c}}$ llamado "centro" , específico para cada nodo. La salida de la red resulta de una combinación lineal entre las salidas de los nodos y un vector de pesos $\underline{\boldsymbol{\theta}}$. La Figura 8 muestra un esquema de esta red.

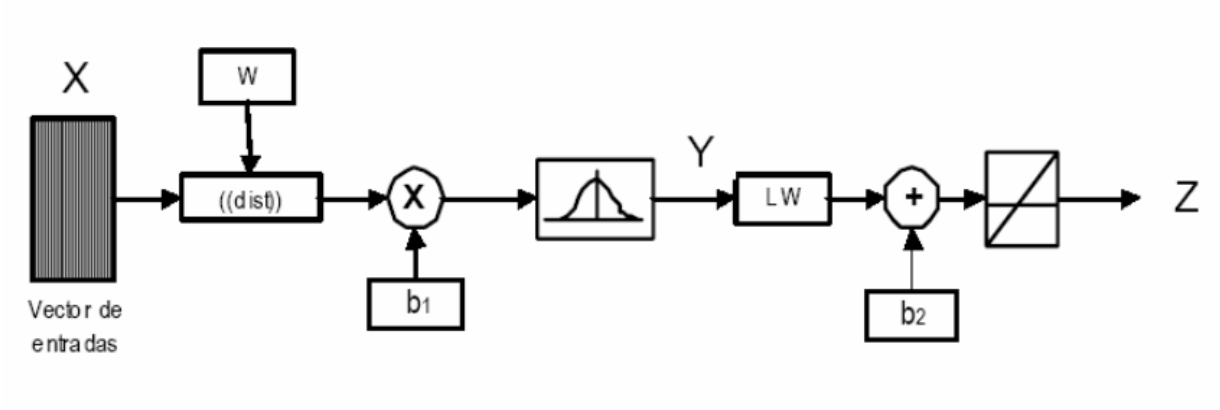


Figura Nº8 : Esquema de red de base radial

Las funciones de activación mas utilizadas son:

- Gaussiana $f(v) = \exp(-v/\mu)\ ;\ \mu > 0$
- Multicuadrática $f(v) = (v+\mu)^{1/2}\ ;\ \mu > 0$
- Multicuadrática recíproca $f(v) = (v+\mu)^{-1/2}\ ;\ \mu > 0$
- Spline grado n $f(v) = v^n \log(v)$

Donde v es el resultado de la operación de Confluencia, definido como la distancia euclidiana entre el vector de entrada $\underline{x}$ y el centro del nodo i, $\underline{c_i}$, y dada por:

$$v_i = \|\underline{x} - \underline{c_i}\| = [(x_1-\underline{c_{i1}})^2+(x_2-\underline{c_{i2}})^2+\ldots+(x_n-\underline{c_{in}})^2]^{1/2}$$

La salida de la red está dada por la suma ponderada de las salidas de las respectivas bases según:

$$\underline{y} = \sum_{i=1}^{nb} f(v_i)*\theta_i$$

La mayor dificultad para el uso de estas estructuras radica en una selección adecuada de su topología que comprende la selección y el número de centros así como el tipo y parámetros de la función de activación. En el área de identificación de procesos químicos se usa una metodología basada en test estadísticos para determinar automaticamente la topología de la red y los centros.

Históricamente, el número de bases ha sido determinado por análisis estadístico o correlación cruzada, mientras que los centros han sido seleccionados básicamente por valores aleatorios o valores extraídos del mismo conjunto de entrenamiento. Una interesante técnica para seleccionar los centros fue sugerida por Moody y Darken basada en un algoritmo que divide al conjunto de datos en ***n*** sectores y obtiene un conjunto de ***n*** centros, minimizando el error cuadrático medio incurrido en representar el conjunto de datos por los ***n*** centros. Estos centros pueden ser parte del mismo conjunto de datos o valores arbitrarios. Denominado ***n-mean clustering***, una de las características principales de esta forma de selección es que puede ser implementado recursivamente permitiendo una contínua adaptación de los centros.

REDES DINAMICAS O RECURRENTES

En este tipo de redes las salidas de las capas ocultas o de salida son retroalimentadas a las capas anteriores. Tambien estas retroalimentaciones pueden ser desfasadas en el tiempo (Delay) para darles un caracter dinamico a su comportamiento.

Las redes recurrentes se utilizan para simular comportamientos temporales como por ejemplo efectos de la visión, y simulacion de sistemas dinámicos. El entrenamiento de estas es mas complicado que las redes « estáticas » por lo que a veces se prefiere entrenar una red con retroalimentación cono una red de tipo FeedForward y despues simular su comportamiento retroalimentando las salidas simuladas.

La figura Nº 9 representa un esquema de red recurrente con atrasos.

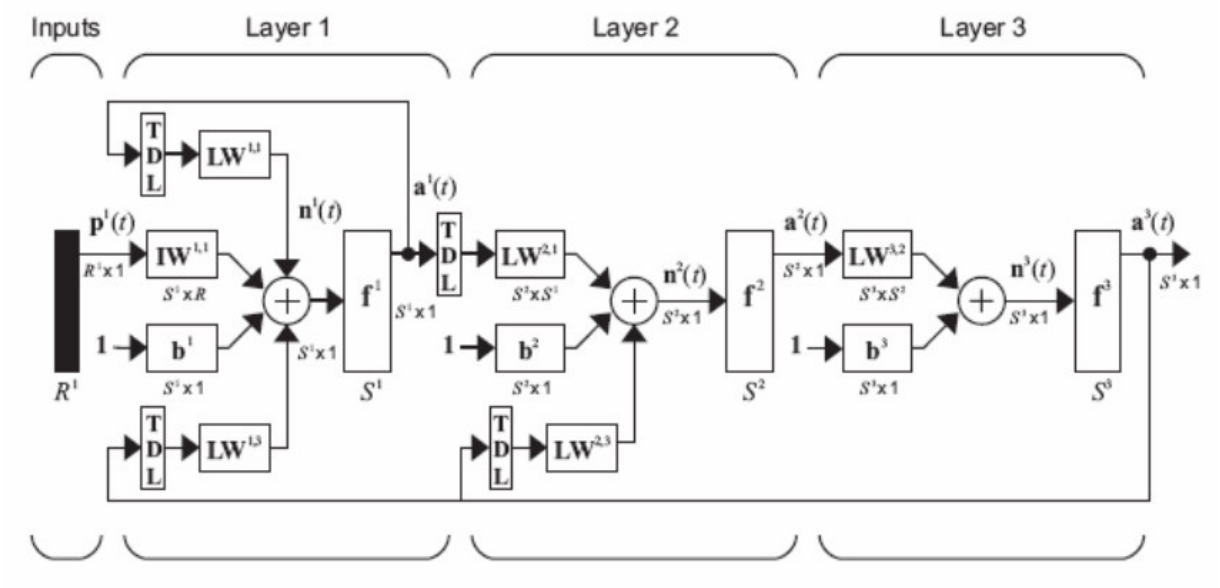


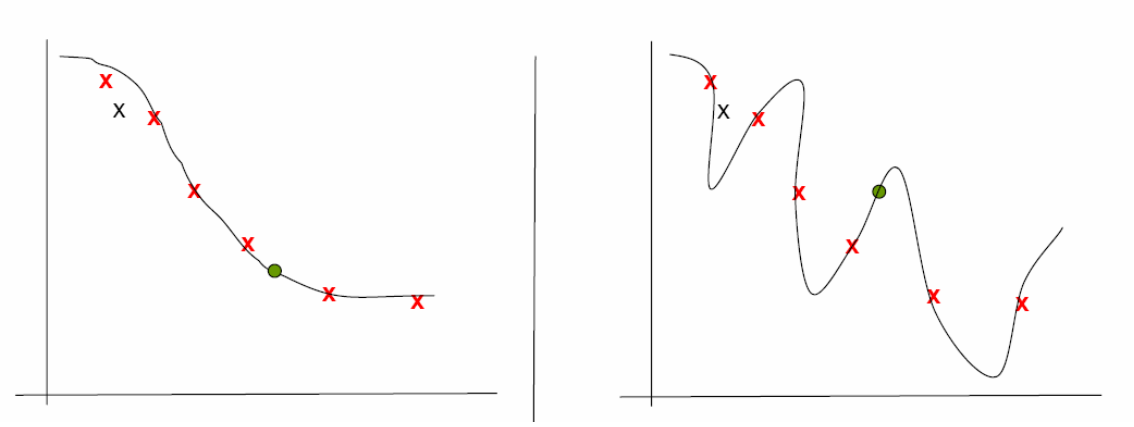
Figura Nº9 : Esquema de red recurrente

3.5 TECNICAS DE VALIDACION

La validación es una parte fundamental dentro del entrenamiento de una red neuronal para la resolución de un problema, esto es debido a que los patrones elegidos para el entrenamiento no tienen por qué representar exactamente como debieran al problema o incluso porque la red haya memorizado los datos siendo incapaz de generalizar la función buscada. Cuando se comienza a entrenar una red con un conjunto de ejemplos, la red cada vez disminuye más el error cometido hasta alcanzar un nivel aceptable (en caso de poseer capacidad de representación suficiente), una vez llegado a este punto se puede seguir bajando el error con entrenamiento, pero ello no nos asegura mejores rendimientos al usarla en casos reales. Esto ocurre porque aunque al principio la red va aproximándose cada vez más a la función objetivo, una vez alcanzada cierta cota al seguir entrenando la red se va particularizando cada vez más a los ejemplos produciéndose un “efecto memoria”. Como al entrenar una red tan sólo tenemos como criterio para seguir entrenando o no el error que comete red sobre los ejemplos, no es posible detectar el efecto anterior, por ellos es necesario incluir nueva información para tener una mejor referencia sobre en qué punto del entrenamiento nos encontramos. Para situarnos más concisamente en el marco del entrenamiento una técnica habitual es separar el conjunto de ejemplos en dos subconjuntos, el de entrenamiento y el de validación, el primero se usará para entrenar directamente la red, y el segundo para

comprobar la capacidad de generalización de esta, ya que el conjunto de validación solo se usa para comprobar la adecuación de la red, no para el entrenamiento.

En la mayoría de los casos al aplicar validación al entrenamiento de la red podemos obtener una medida clara de cuando la red se está sobre-entrenando o cuando generaliza de forma adecuada. Hay otras técnicas mas sofisticadas de validación, una de ellas es la validación cruzada, se dividen los datos en un número de conjuntos y se entrena con ellos dejando en cada iteración uno de ellos para validación, al final en la última iteración se usan todos los datos para entrenar.



Buena generalización sobre-entrenamiento

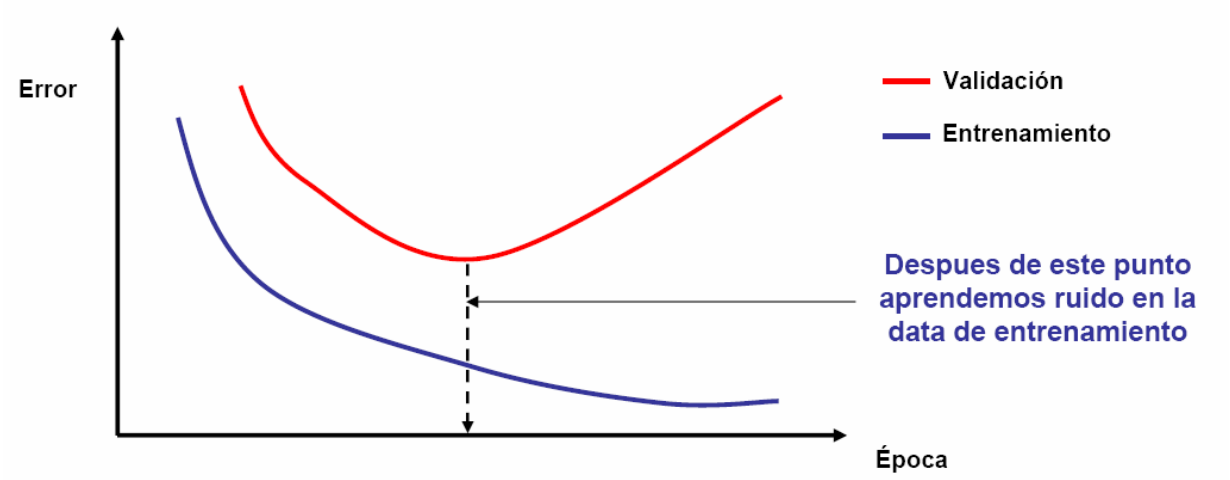


Figura Nº 10: Representación del efecto de sobre-entrenamiento

CAPITULO 4 SOFTWARES DE DESARROLLO PARA REDES NEURONALES

A partir de la codificación del algoritmo de retropropagación y la efectividad de los desarrollos basados en redes neuronales, la industria del software rapidamente comenzo a desarrollar aplicaciones tanto generales como específicas en el campo de las redes neuronales. En el anexo A se entrega un listado de los softwares que contiene el directorio de Google. En el se encuentran softwares gratutos, comerciales, de aplicación general o de aplicación en areas específicas como por ejemplo finanzas.

Entre los softwares mas usados, dos merecen atencion por su popularidad:

- "Neurosolutions" que es un sftware comercial a nivel de usuario que permite rápidos desarrollos en todas las areas, incluyendo un "adds-in" para excel.
- Neural Network Toolbox, Que es una colección de funciones que corren con la plataforma de análisis ingenieril Matlab.

En el ambito de este curso los ejemplos y desarrollo los realizaremos usando este ultimo software por lo que lo describiremos con más detalle.

4.1 TOOLBOX DE REDES NEURONALES DE MATLAB

Matlab es considerada hoy en dia como una de las principales plataformas de análisis y desarrollo de prototipos a nivel de ingeniería. Consiste en un interprete que ejecuta comandos y funciones propias o diseñadas por el usuario, ofrece además un conjunto de programas llamados "Toolbox" que contiene funciones propias de un campo específico. En este contexto matlab ofrece el Toolbox de redes neuronales para integrar los desarrollos de redes neuronales.

Con esta herramienta se puede crear, entrenar y simular una red neuronal y despues integrarla facilmente a rutinas de simulación optimización y diseño.

Una sesion de red neuronal en matlab consiste en ejecutar las funciones de creacion, entrenamiento y validacion de resultados mediante comandos, a traves de script (programa) o una interfase para hacerlo de forma sencilla para un usuario no familiarizado con matlab.

Tambien tiene la posibilidad de diseñar redes neuronales de tipo modular con una estructura determinada por el usuario a travez del comando "***network***"

El toolbox incluye una serie de funciones para pre y post procesamiento de los datros y un libro electrónico como tutorial de uso.

Para acceder a la interfase grafica se debe dar el comando **"*nntool*" ,** mientras que para ell ibro electronico el comando es **"*nnd*".**

La figura siguiente ilustra la interfase grafica de desarrollo

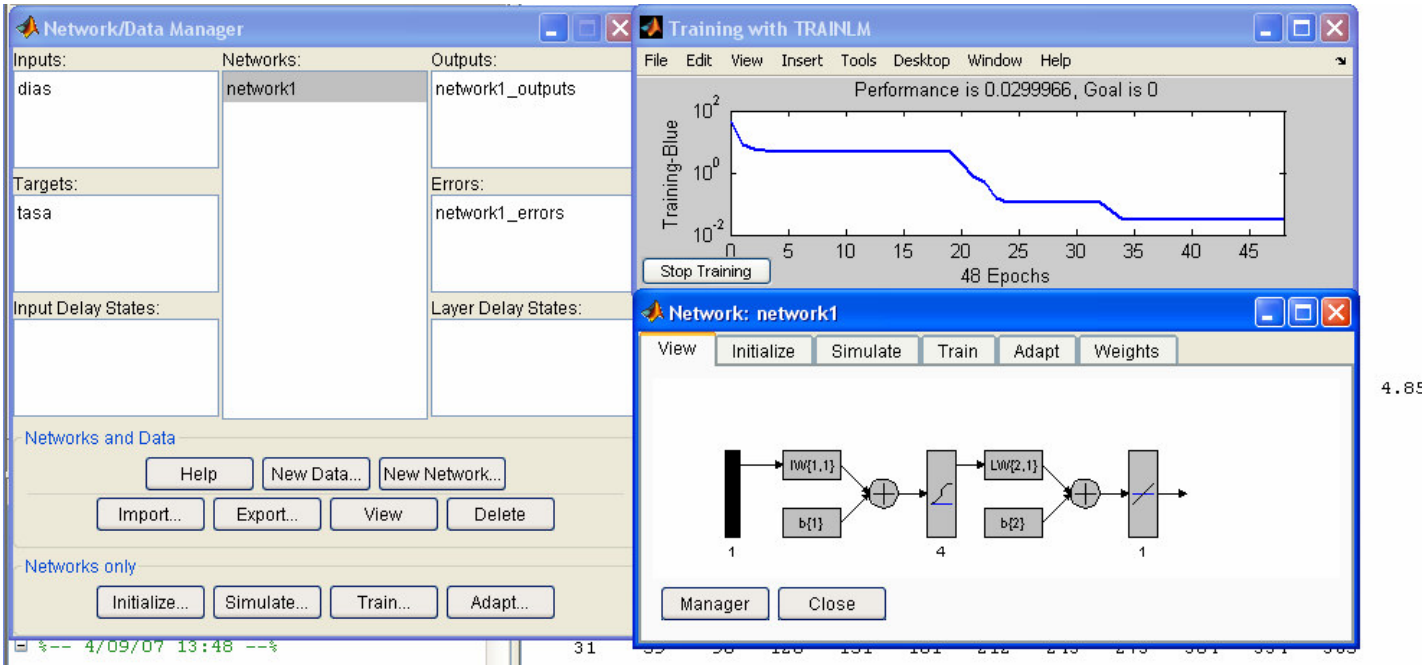


Figura Nº 11: Interfase grafica NNTOOL

4.2 EJEMPLOS A DESARROLLAR

Ejemplo de red estatica

A continuacion se presenta una sesión típica para el entrenamiento de una red neuronal estatica para la prediccion de la tasa de evaporacion como funcion de la fecha..

```
%script ejemplo uso redes neuronales estaticas
% carga archivo evaporacion
load evaporacion
who
% tasa = tasa de evaporacion segun fecha
% dias =  dias a partir de 1º de enero
plot(dias,tasa)
% crea  red bneuronal FF 1 capa oculta
netevap = newff([0 400],[3 1],{'logsig' 'purelin'});
% inicializa pesos
netevap=init(netevap);
%entrena la red
netevap.trainParam.epochs = 1000;
netevap=train(netevap,dias,tasa);
%simula salidas red entrenada
Salevap=sim(netevap,dias)
% grafico de salidas predichas y medidas
plot(dias,tasa,dias,Salevap)
```

El script denominado “ejeevap.m” define una red neuronal FF con 3 nodos en la capa oculta con funcion de activación sigmoidal y una funcion lineal para la salida.

La ejecucion del script arroja la curva de entrenamiento y un grafico de la predicción dela red entrenada.

Como los pesos son inicializados aleatoreamente, una sola sesión no garantiza el mejor resultado por lo que el procedimiento hay que repetirlo varias veces . La figura siguiente ilustra esta caracteristica.

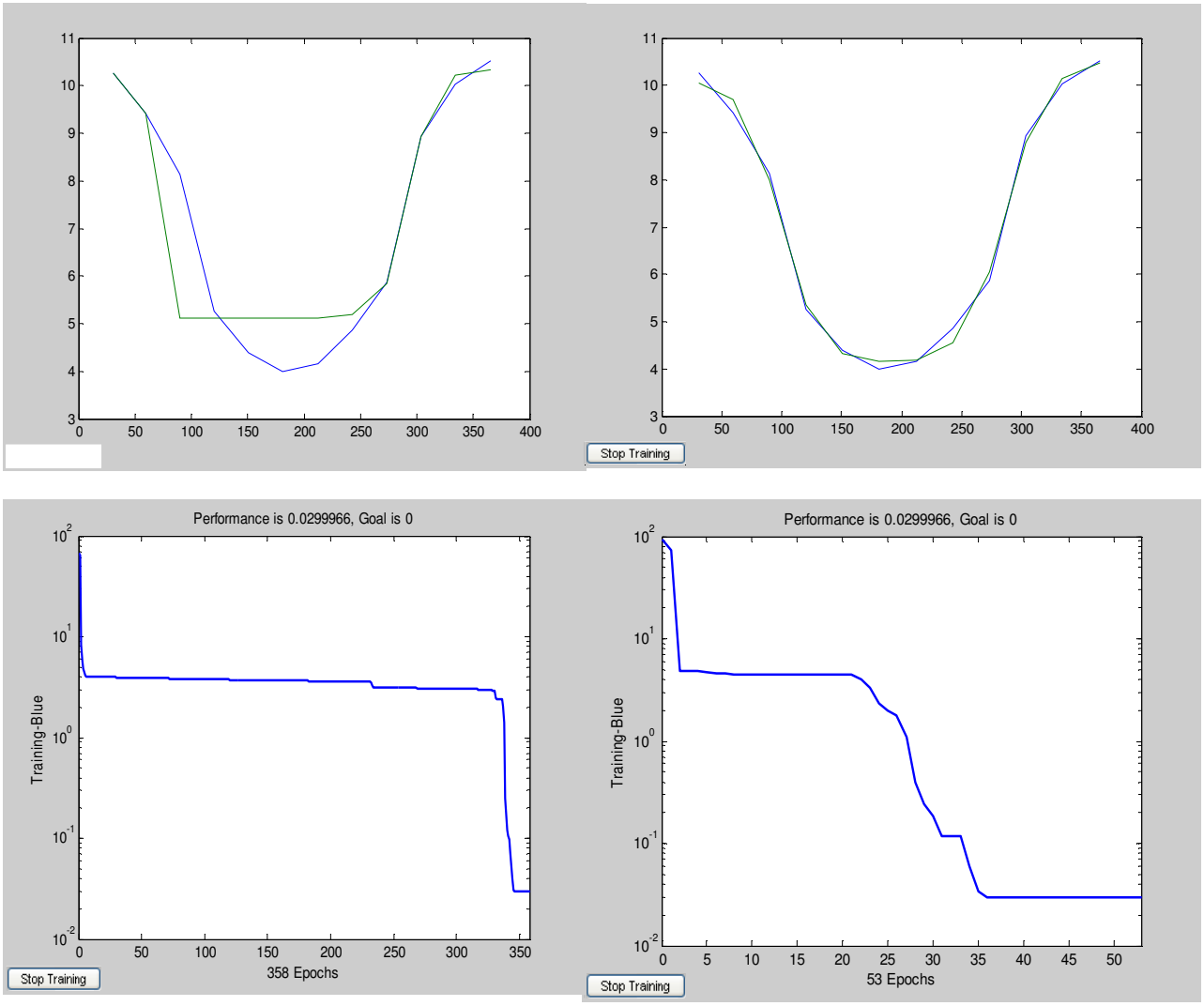


Figura Nº 12: Resultados tipicos de entrenamiento y validacion de la red: Entrenamiento parcial (izquierda), entrenamiento satisfactorio (derecha). La evolucion del entrenamiento se muestra en las figuras inferiores

Ejemplo de red dinamica

En este ejemplo se entrenará una red neuronal para modelar un proceso dinámico con un modelos tipo NARX (Ver Cap 5) . El ejemplo denominado ejeNARX.m contiene un archivo con 1000 datos historicos en secuencia de la entrada y salida de un calefactor de aire para secado. La entrada U2 es el calor añadido y la salida y2 es la humedad de salida del aire. La red tratara de predecir la salida en el instante actual y2(k) como funcion de los estados anteriores u2(k-1) , y2(k-1). Se usará una red neuronal tipo FF con una capa oculta .

```
% ejemplo modelo dinamico NARX
%carga datos secador dryer2
load dryer2
%grafico
subplot(2,1,1); plot(u2)
subplot(2,1,2); plot(y2)
%formato archivo de entrada red neuronal
S=[u2 y2]';
% numero de datos disponibles
size(S);
%creamos red con dos estradas u(k-1) y(k-1)
netnarx = newff([0 10;0 10],[2 1],{'logsig'
'purelin'});
netnarx=init(netnarx);
%entrenamiento con 300 datos
netnarx=train(netnarx,S(1:2,1:300),S(2,2:301));
%simulacion con todos los datos
saly2=sim(netnarx,S(1:2,1:999));
close
%grafico de salida
plot(2:1000,y2(2:1000),2:1000,saly2(1:999))
```

Ali gual que en el caso anterior es necesario repetir el procedimiento las veces que sean necesarias para obtener un entrenamiento satisfactorio. La figura siguiente ilustra dos patrones de entrenamiento diferentes, uno fallido y otro satisfactorio.

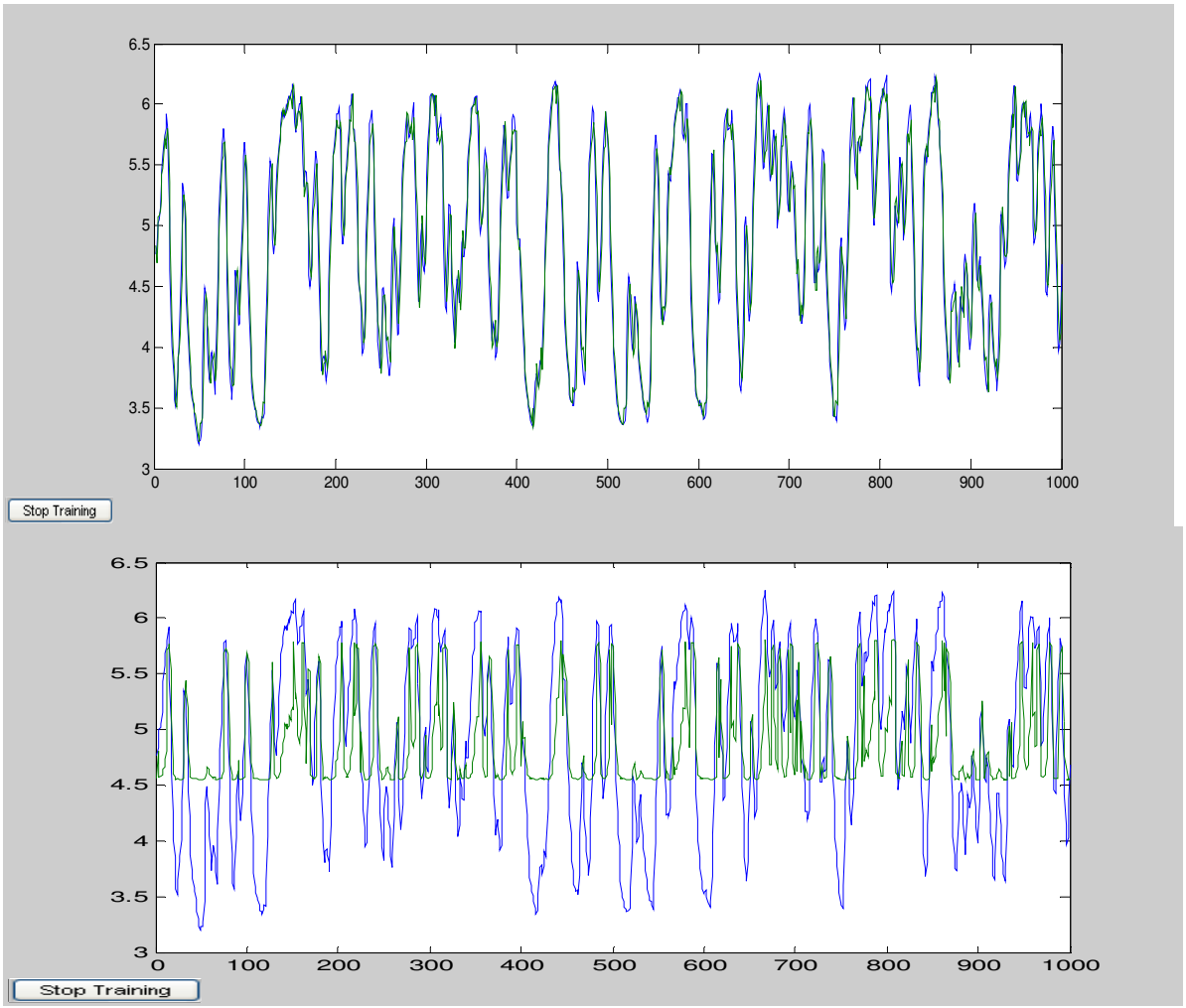


Figura Nº 13: Resultados para dos entrenamientos diferentes, solo los primeros 300 datos se usan para entrenar

CAPITULO 5: APLICACIONES EN INGENIERIA DE PROCESOS

Las aplicaciones típicas de las redes neuronales en el ámbito de la ingeniería de procesos comprende la estimación de patrones o funcionalidades estáticas en un esquema denominado de “caja negra” donde se persigue una mera regresión entre datos de entrada-salida. A continuación se presentan 3 esquemas de uso de redes neuronales que le dan una carácter mas especializado a los desarrollos y permiten su integración a sistemas de toma de decisiones. Estos tópicos son:

1. Modelos neuronales híbridos o de “caja gris” que permite la estimación de parámetros difíciles o imposibles de medir, lo que a veces se denomina un “sensor virtual”

2. Modelos dinámicos de procesos no lineales que son usados en predicciones futuras

3. Esquemas de control para procesos no-lineales, llamado a veces “control neuronal”

5.1 MODELACION HIBRIDA NEURONAL (CAJA GRIS)

La técnica de modelación híbrida-neuronal (**MHN**), consiste en la formulación de un modelo de proceso, compuesto por ecuaciones deducidas de princípios fenomenológicos - tales como balances de masa, energía, cantidad de movimiento, relaciones de equilibrio, etc - y redes neuronales, estas últimas encargadas de estimar aquellos parámetros inciertos o de difícil modelación.

Esta forma de representación es un intento por incluir un conocimiento previo a modelos neuronales tipo "caja negra", a fin de reducir su complejidad y mejorar sus propiedades de adaptación y predicción. Un esquema general para esta forma de modelo se muestra en la Figura 14.

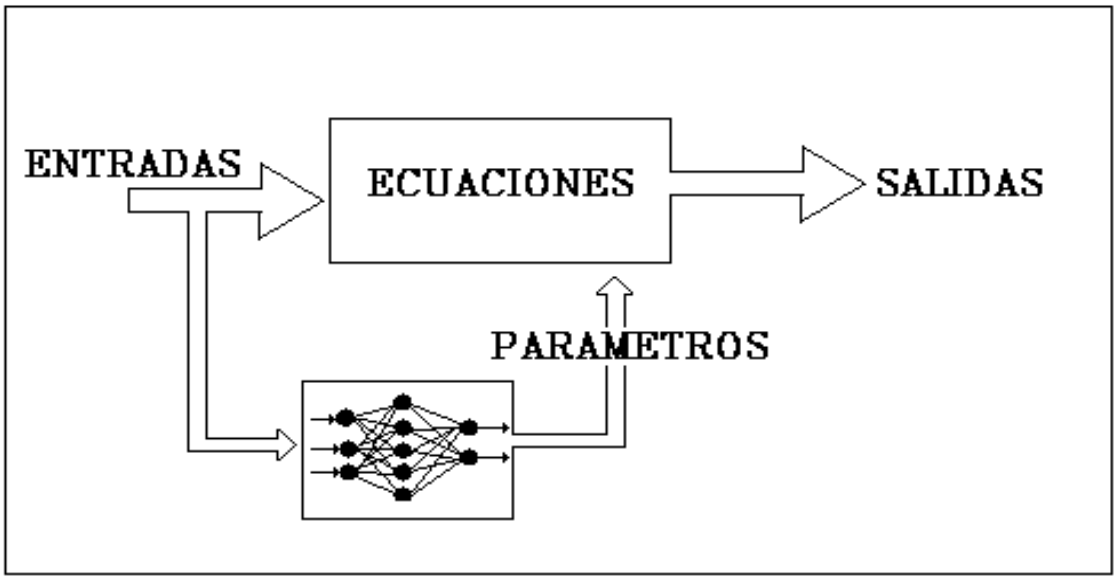


Figura 14 : Diagrama de un modelo híbrido-neuronal (MHN)

En 1992, Psichogios y Ungar aplicaron el esquema de modelación-híbrida a un bioreactor simulado, estimando las constantes de crecimiento bacteriano. Los resultados indicaron un excelente comportamiento de predicción y buena adaptación frente a datos contaminados con ruido. En todos los casos, el modelo híbrido-neuronal fué superior a su equivalente de "caja negra" y a otras formas de estimación de los parámetros cinéticos como Filtro de Kalman. Posteriormente los autores muestran las potencialidades de esta forma de modelación en un problema de optimización, cuyo objetivo fué determinar la mejor política de alimentación, a fin de maximizar la producción del sistema.

Desde un punto de vista fenomenológico, este tipo de modelación presenta el gran atractivo de que el conocimiento adquirido en torno a las restricciones y mecanismos de cada proceso, pueden ser efectivamente usados en el modelo, ya sea en forma de ecuaciones, o como información de entrada a la red neuronal. Mientras que la red neuronal se encarga sólo de la correlación entre los parámetros y los estados del sistema, focalizando el entrenamiento y disminuyendo la probabilidad de sobre-especificación de la red.

Debido a la forma en que los modelos híbridos neuronales son construídos, es posible deducir que la estructura produce mejores modelos que los "puramente empíricos" (basados solo en redes neuronales), debido a que en los primeros, las restricciones de conservación aparecen en forma explícita, resultando modelos más precisos en la predicción. Además el tamaño de la red neuronal es generalmente menor que en los modelos de caja negra, porque las dimensiones de las capas de entradas y de salidas son menores, facilitando así el entrenamiento, producto del reducido número de parámetros a estimar.

Otra ventaja de esta forma de modelos, sobre los no-paramétricos, es su flexibilidad, ya que cada parte del **MHN** tiene un significado físico y una tarea específica a realizar. Así, cada una de ellas puede ser mejorada, a medida que la experiencia adquirida aumenta, o alterada donde corresponda, para ser utilizada en algún proceso con aspectos fenomenológicos similares.

Por otro lado, la modelación híbrida neuronal presenta dos grandes ventajas frente a la modelación puramente fenomenológica: La primera se refiere a la capacidad de predecir los parámetros del modelo usando todas aquellas variables que los afectan, evitando tener que especificar en forma explicita, el tipo de correlación. Esta facilidad es particularmente útil en la modelación de procesos complejos, en medios heterogéneos o particulados, en donde una serie de mecanismos fisicoquímicos pueden actuar simultáneamente, dificultando enormemente su descripción mediante las técnicas tradicionales.

La segunda es la posibilidad de reducir la envergadura del modelo matemático introduciendo un error de modelación el que es absorbido durante el entrenamiento de las redes. El resultado es un modelo híbrido matemáticamente más simple que el original de primeros principios.

En síntesis, la modelación **MHN** se traduce en modelos relativamente simples de formular cuando existe un conocimiento previo del proceso, con una red neuronal fácil de entrenar, con buenas propiedades de generalización, matemáticamente simples y ampliamente flexibles y transportables.

METODOLOGIA

La metodología para la formulación de un modelo híbrido neuronal puede resumirse en la aplicación general de los siguientes pasos:

1) Establecer las ecuaciones de balances de masa, energía, cantidad de movimiento y relaciones termodinámicas, que relacionan las variables de salida con las variables de entradas y los parámetros del modelo.

2) Determinar cuales parámetros del modelo serán estimados vía redes neuronales y establecer la dependencia con las variables de entrada.

3) Determinación de la estructura y entrenamiento de las redes neuronales usando datos obtenidos del proceso.

4) Evaluación de la calidad de ajuste y capacidad de predicción del modelo final usando en lo posible datos no incluídos en el ajuste.

Los pasos anteriormente enumerados constituyen una guía general para el planteamiento de este tipo de modelos, por lo que existen muchas alternativas de **MHN** para un mismo proceso.

Como no existe una única estructura **MHN** para un dado proceso, quizas, la aplicación final determine su grado de detalle fenomenológico y complejidad matemática. Así, si el **MHN** será usado en tareas tales como control predictivo u optimización, este deberá ser matematicamente simple, a costa de supuestos gruesos de modelación (que deberían ser absorvidos por la red), para permitir una rápida solución de las variables de decisión.

Si el **MHN** será usado como un simulador del proceso, es conveniente detallar al máximo los mecanismos conocidos y dejar a las redes neuronales estimar los aspectos desconocidos de los fenómenos involucrados.

5.2 IDENTIFICACION DE SISTEMAS DINAMICOS

Las redes neuronales pueden ser usadas satisfactoriamente para identificación dinámica de procesos no lineales usando un esquema en paralelo como se muestra en la Figura xxx con la planta en lazo abierto, o mediante un archivo de datos históricos de entrada salida.

Para la identificación de procesos químicos, las redes neuronales tipo *Feedforward* (**FFNN**) con una capa oculta y funciones de activacion sigmoidales son las mas ampliamente usadas

Una atractiva variante a las redes *Feedforward* para identificación dinámica son las redes dinámicas o recurrentes (**RNN**), en donde las salidas de la red son retroalimentadas a los nodos anteriores para incluir una dinámica interna en la estructura. Su topología es mas compleja y el entrenamiento más complicado en relación a las redes **FFNN**

Las redes de base radial (**RBF**) también han sido exploradas para tareas de identificación dinámica de procesos. Su mayor ventaja es que los pesos de la red son lineales con respecto a las salidas y pueden ser estimados por métodos robustos de mínimos cuadrados.

Habitualmente, la identificación supone que la red neuronal aproxima al modelo de la planta en un esquema de predicción de un paso, siguiendo un esquema **NARX** (***N****onlinear* ***A****uto****R****egresive with e****X****ogenous*), que usa un número de entradas y salidas presentes y pasadas de la planta, para predecir la futura salida del proceso., según:

$\hat{y}$**(k+1)=*NN*[y(k),y(k-1),y(k-2)..y(k-n),u(k),u(k-1)..u(k-m)]**

en donde ***NN(x)*** es una red neuronal con vector de entradas ***x.***

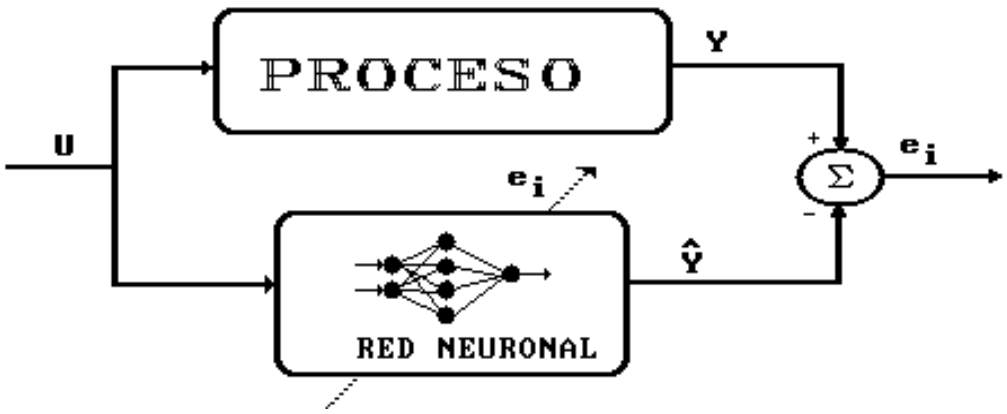


Figura 15 : Esquema de identificación con una red neuronal

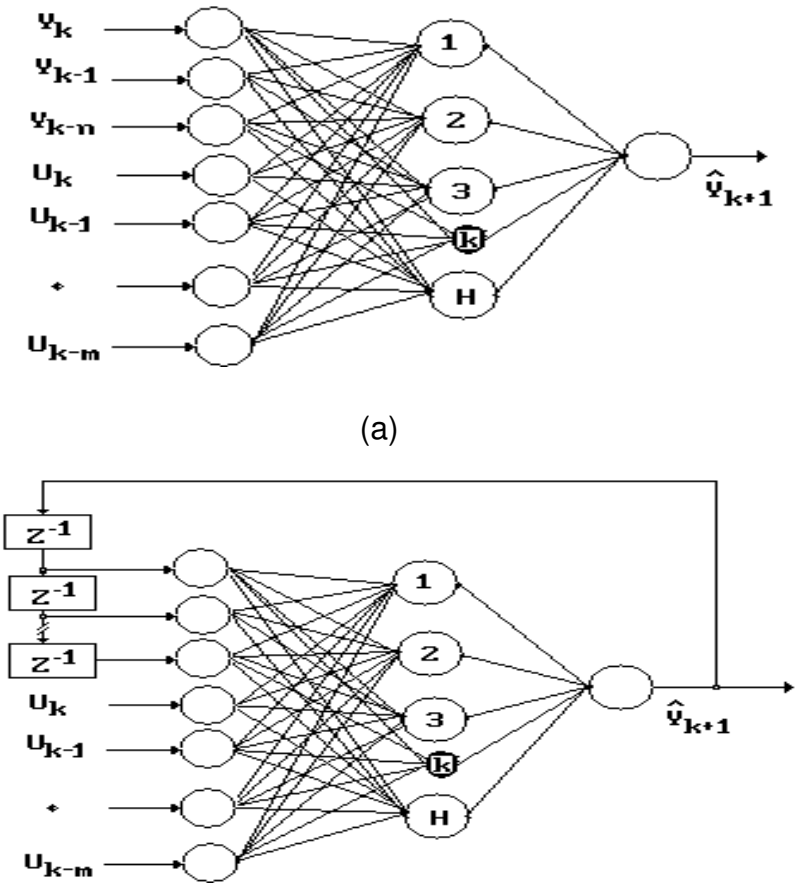


Figura 16: Estructuras típicas para identificación de procesos usando una red FFNN (Arriba) y una RNN (abajo) .

5.3 CONTROL DE PROCESOS CON REDES NEURONALES

Sin duda, la capacidad de las redes neuronales para identificar sistemas complejos, con no linealidades, y dinámica variable, ha producido un gran interés en incorporarlas a los métodos de control avanzado, principalmente de dos formas: como clasificador de padrones o como modelos de procesos no lineales.

La primera forma utiliza una red como un sistema clasificador de padrones, donde la red es capaz de reconocer un cierto evento en el sistema (una perturbación, cambio de propiedades, cambio en objetivo de control), y generar una acción correctiva, cambiando los parámetros o modificando la respuesta de un controlador convencional.

La segunda forma consiste en usar las redes neuronales como modelo de procesos, e integrarlas a algún esquema de control no lineal tipo **MBC** (Model Based Control), consiguiendo un mejor desempeño, debido a la calidad de la identificación y al uso de una estructura más simple y fácil de sintetizar, en comparación a los métodos tradicionales, que usan ecuaciones diferenciales o aproximaciones de éstas. Los métodos **MBC** con redes neuronales son los más ampliamente estudiados debido a que es un área bien establecida, con un abundante desarrollo teórico para sistemas lineales, aparte de tener una amplia aceptación en el ámbito industrial.

Los esquemas de control **MBC** pueden ser agrupados en dos grandes grupos:

-Esquemas de control Directos: Donde la red neuronal corresponde al controlador y la acción de control es computada en forma directa.

- Esquemas de Control Indirectos: Donde es preciso identificar primero la dinámica directa de la planta con la red neuronal, y después calcular la acción de control mediante algún procedimiento de diseño, que incluya al modelo identificado previamente. **MPC** (Model Predictive Control) e **IMC** (Internal Model Control) forman parte de este grupo.

CONTROLADORES DIRECTOS

En este tipo de controladores, la red neuronal es entrenada para predecir la dinámica inversa del proceso, por lo que la acción de control corresponde a la salida de la red, sin necesidad de un procedimiento de diseño del controlador.

La versión más simple es la denominada "*Input Matching*". En este esquema, la red es entrenada como la verdadera inversa de la planta usando las entradas del proceso en lugar de las salidas, según se muestra en la Figura 17 Aunque simple en esencia, este controlador sólo puede ser implementado de una forma adaptable, para compensar los errores de identificación y reyectar perturbaciones.

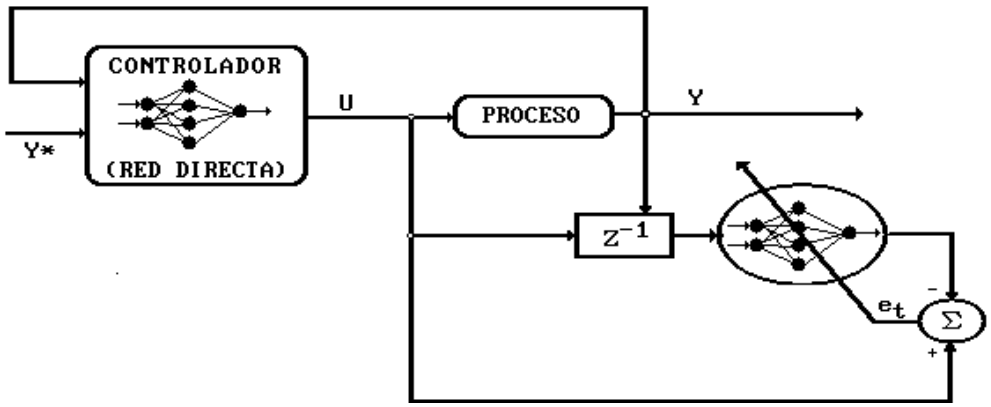


Figura Nº 17: esquema de síntesis directa

Para dar más robustez al diseño se propone un controlador directo en un marco **IMC**, a fin de incluir retroalimentación al sistema, según se muestra en la Figura 18.

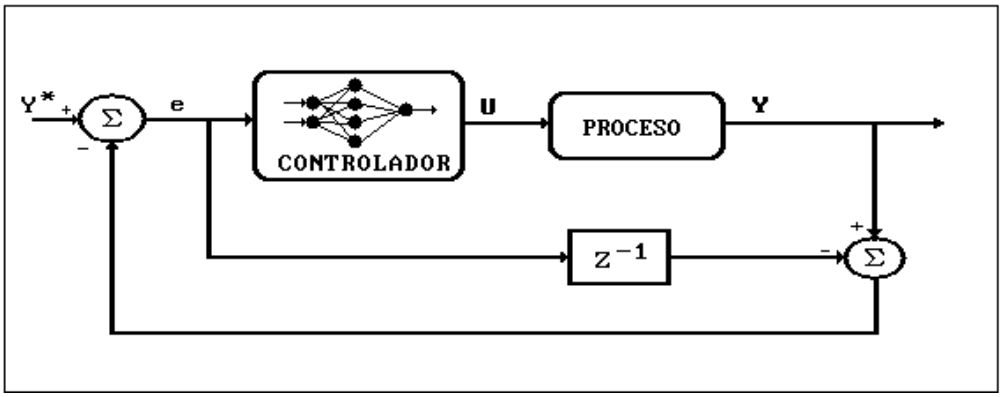


Figura Nº 18 Esquema de control directo en un marco IMC.

Otra forma de control directo, corresponde al esquema de control **IMC** en la cual, el block de la inversa del proceso es una red neuronal entrenada al igual que en el caso anterior, con la ventaja de ser robusta a los errores de modelación y frente a perturbaciones no medidas.

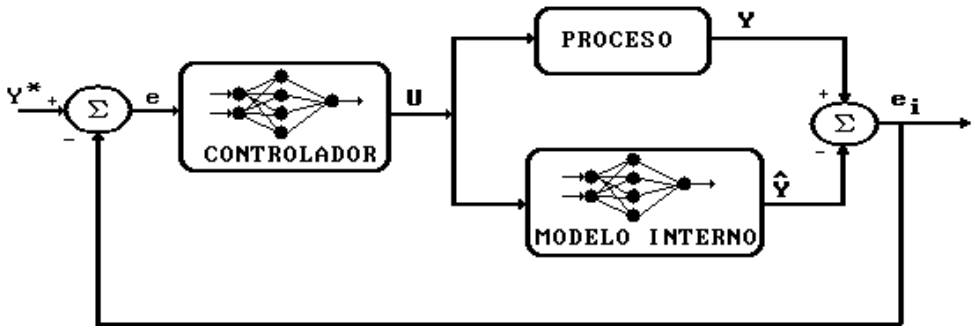


Figura Nº 19 : Esquema de control IMC ; En el esquema Directo, el controlador es una red neuronal entrenada como la inversa. En el esquema Indirecto el controlador es un algoritmo de diseño.

Aunque el esquema directo tiene la gran ventaja de entregar explícitamente el valor de las variables de control sin ningún procedimento de diseño, hasta el momento, el uso de las redes neuronales como modelos inversos es tratado con reservas por los investigadores, debido a que no existen garantías sobre su comportamiento en los puntos donde el proceso puede no ser invertible.

CONTROLADORES INDIRECTOS

Como fué establecido anteriormente, los controladores indirectos requieren de un modelo del proceso y de un algoritmo de diseño, que tome sus decisiones en base a las predicciones del modelo. De esta forma la red neuronal es usada como un modelo no lineal del proceso, que puede ser entrenada fuera de linea, usando un archivo de entrenamiento, o entrenada *on-line,* mediante un algoritmo recursivo de adaptación. La Figura 20 muestra un diagrama general de este esquema de control.

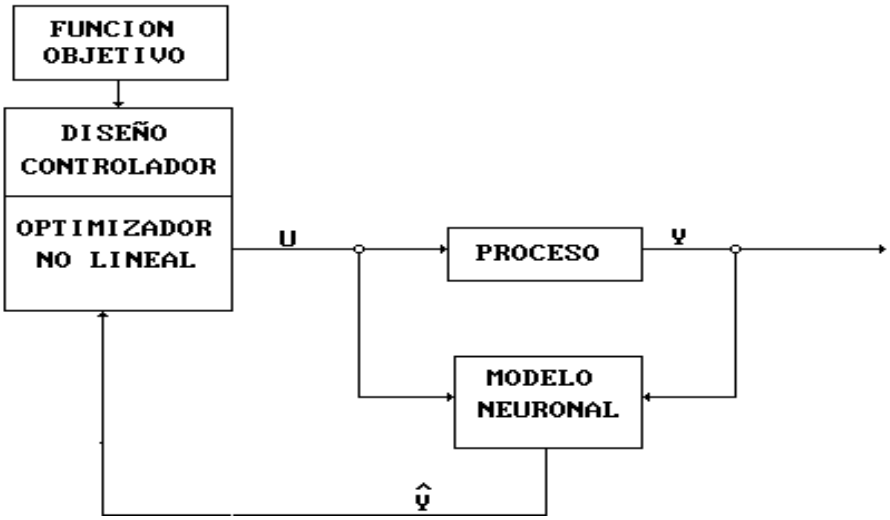


Figura 20 : Esquema general de un esquema de control indirecto

Una clasificación de utilidad se basa en el horizonte de predicción para el calculo de la respuesta del controlador. Así, tenemos los siguientes controladores.

- Predictivos o de "largo alcance", en donde las acción de control es calculada para satisfacer un objetivo a lo largo de un horizonte de predicción.

- De "un paso" en el que la accion de control se toma para regular al proceso en el próximo instante de muestreo.

- Predictivos de estado estacionario, en donde la acción de control es adoptada para regular al proceso en el estado estacionario.

Controladores de un paso

Este tipo de diseño, la acción de control es realizado sobre la predicción del modelo para el próximo instante de muestreo. La versión más sencilla es la denominada de "sintesis directa", en la cual se calcula el control, tal que el sistema alcance el *setpoint* en el próximo instante. Esta formulación es particularmente útil para sistemas adaptables o diseños especiales de redes, que permiten el calculo directo de la variable manipulada.

El diseño más utilizado en controladores de un paso es la versión indirecta de **IMC,** debido a sus características de estabilidad y robustez. En **IMC** indirecto 19, el modelo interno es una red neuronal que aproxima la dinámica del proceso, mientras que el block del controlador,

obedece a un diseño que invierte el modelo interno de la planta, ya sea resolviendo una ecuación no lineal o solucionando un problema de optimización.

CONTROLADORES PREDICTIVOS

Los controladores predictivos o **MPC** (*Model Predictive Control*) son actualmente los controladores avanzados con mayor éxito en el campo industrial. En ellos, las acciones de control son determinadas, tal de minimizar una función objetivo que es calculada usando las predicciones de un modelo de proceso, a lo largo de un horizonte de predicción. Puesto como un problema de optimización, esta formulación permite la inclusión explicita de las restricciones del proceso. En la práctica, este controlador es capaz de regular procesos inestábles y de dinámicas complejas.

Una formulación general de un controlador **MPC** está dado por:

$$\min_{u(k+i)} \sum_{i=0}^{P} \tau_i[\hat{y}(k+1+i) - \mathbf{r}^*]^2 + \lambda_i[\mathbf{u}(k+i) - \mathbf{u}(k+i-1)]^2$$

$$\mathbf{u}(k+i) = \text{cte} \quad \text{para } i = m \ldots p \; ; \; m \leq p$$

Donde **p** es el horizonte de predicción, **m** el horizonte de control, **r*** los set-points; τ y λ pesos positivos.

En esta formulación, las redes neuronales son usadas como modelos de predicción para ser usadas en control de procesos no lineales. La red puede ser diseñada para predicción de un paso o para predicción múltiple.

La selección de un conjunto particular de parámetros de la formulación MPC, genera una serie de sub-algoritmos, tales como, *Multi Step Predictive Control* , Horizonte Extendido o IMC de tipo indirecto. En todas las implementaciones es aplicado el principio de "horizonte movil", donde sólo el primer movimiento de la secuencia de acciones de control es aplicado al proceso, luego el problema es nuevamente resuelto en el próximo instante de muestreo.

Adicionalmente, un término de error de modelación puede ser incluído en la función objetivo, para dar la retroalimentación necesaria para reyectar perturbaciones y desviaciones de modelación.

La aplicación de redes neuronales en MPC fué inicialmente estudiada por Hernandez y Arkun (1990) que lo llamó de *Extended* DMC debido a su semejanza con *Dymanic Matrix Control*. Posteriormente los mismos autores desarrollan la versión equivalente de horizonte extendido (HE). Con este último algoritmo se han demostrado condiciones de estabilidad en lazo abierto y cerrado, así como la habilidad para controlar procesos con respuesta inversa, manipulando convenientemente el horizonte de predicción.

Cabe destacar que a partir de la versión de Matlab7 , se incluye en simulink una biblioteca con boques de control neuronal de tipo IMC y control predictivo (NMPC) . Estos bloques trabajan en conjunto con modelos de procesos elaborados en similink para el desarrollo de prototipos. La figura siguiente ilustra este concepto

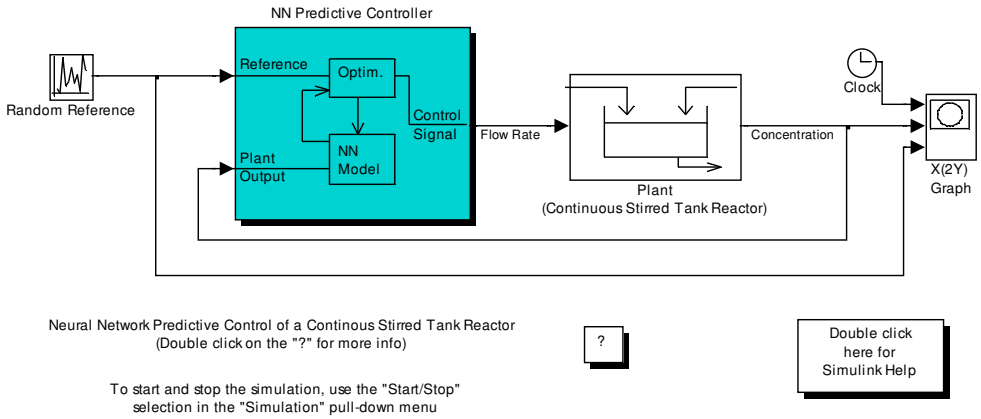


Figura 21: controlador NMPC de Matlab

BIBLIOGRAFIA

- "Neural Networks: A comprehensive Foundation", Simón Haykin, Segunda Edición, Prentince Hall, 1998.
- "Neural Networks Toolbox: User's guide. Version 5", H. Demuth and M. Beale The Mathworks
- “Neural Networks for Pattern Recognition”, Christopher Bishop,Oxford University Press, 1995
- Apuntes de Sistemas Inteligentes en Ingeniería de procesos, Francisco Cubillos, www.labcontrol.cl
- "Redes Neuronales Artificiales", José R. Hilera y Victor J Martinez. 2000. Alfaomega. Madrid. España

APENDICE A

SOFTWARES DE REDES NEURONALES EN INTERNET (Fuente GOOGLE)

Neural Network Toolbox for MATLAB - http://www.mathworks.com/products/neuralnet/

An environment for neural network research, design, and simulation within MATLAB.

Torch - http://www.torch.ch

A library of state-of-the-art machine learning algorithms. Licensed under the GPL, and designed for Unix and Linux environments.

Netlab - http://www.ncrg.aston.ac.uk/netlab/

A library of MATLAB functions for simulating neural network algorithms based on the book Neural Networks for Pattern Recognition by Chris Bishop.

DELVE - http://www.cs.toronto.edu/~delve/

A standard environment for evaluating the performance of learning methods. Includes a number of datasets and an archive of learning methods.

Pythia - http://www.runtime.org/pythia.htm

Software for simulation of back propagation neural networks. Evaluation version available.

Joone - http://www.jooneworld.com

Java Object Oriented Neural Engine is a free java neural net framework. Can be extended writing new modules.

NeuroSolutions - http://www.neurosolutions.com

Icon-based neural network development software. Supports several types of networks and training algorithms. Trial version is available.

Neuromat - http://www.neuromat.com

Model Manager for development of bayesian neural networks.

Alyuda NeuroIntelligence - http://www.alyuda.com/

Neural network software and Excel add-ins for forecasting and data analysis. Supports several algorithms. Trial versions are available.

PDP++ - http://www.cnbc.cmu.edu/PDP++/PDP++.html

A neural-network simulation system based on C++. This is the next generation of the PDP software originally released with McClelland and Rumelhart's PDP book.

NeuralWorks - http://www.neuralware.com/products.jsp

Professional II/PLUS is a comprehensive neural network development environment. Available for Windows and Unix. Predict is a neural network tool for solving prediction and classification problems. Available for Unix or as an Excel add-in for Windows.

FANN - http://fann.sourceforge.net

Neural network library implemented in ANSI C. Creates multilayer feedforward networks with support for both fully connected and sparse connected networks. Supports execution in fixed point, for fast execution on systems like the iPAQ.

Xerion - http://www.cs.toronto.edu/~xerion/

Neural network simulator based on C and Tcl. Made up of C libraries to build networks, and pre-built simulators.

Neural Networks at your Fingertips - http://www.neural-networks-at-your-fingertips.com/

Neural network simulators for eight different network architectures with embedded example applications coded in portable ANSI C.

Cobalt A.I. - http://www.cobaltai.com/

Provides programming tools for rapid application development of artificial intelligence systems.

EasyNN - http://www.easynn.com

Neural network software for Windows with numeric, text and image functions.

Simbrain - http://simbrain.sourceforge.net

A free java-based neural network simulation kit.

Tradecision - http://www.tradecision.com/

Neural network software for technical analysis and stock market trading. Demo is available.

NetMaker - http://www.ire.pw.edu.pl/%7ersulej/NetMaker/

Simulates MLP, RMLP and Cascade-Correlation models with dynamic size adjustment

algorithms. Includes various training patterns, error and activation functions.

NeuroXL - http://www.neuroxl.com

MS Excel add-ins based on neural networks. Designed for predicting, classification and financial forecasting.

Fann Neural Network for Mathematica - http://www.geocities.com/freegoldbar/

Free interactive environment for Mathematica includes pattern recognition and time-series prediction samples.

Neural Network Leaves Recognition - http://damato.light-speed.de/lrecog/

A neural network based system to recognize leaves written in Java. A Java-Applet is also available.

Annie - http://annie.sourceforge.net

Open-source neural network library for C++ (Windows and Linux). Support for MLP, RBF and Hopfield networks. Interfaces with Matlab's Neural Network Toolbox.

NNSYSID Toolbox - http://kalman.iau.dtu.dk/research/control/nnsysid.html

A set of MATLAB tools for neural network based identification of nonlinear dynamic systems.

The Neural Simulation Language - http://www.neuralsimulationlanguage.org

A simulation system for modeling large-scale general neural networks.

Tiberius - http://www.philbrierley.com

Neural network for classification and regression problems. Supports ODBC and Excel.

NeuroBox - http://www.cdrnet.net/projects/neuro/

An opensource .NET OOP library project written in C# to generate, propagate and train complex neural feedforward networks.

Amygdala - http://amygdala.sourceforge.net/

Open-source software for simulating spiking neural networks, written in C++.

Neural Network Models in Excel - http://www.geocities.com/adotsaha/NNinExcel.html

Neural network freeware for building prediction and classification models in Excel. Uses

backpropagation. Can handle missing values and categorical data.

NeuroDesigner - http://www.neurodesigner.com

A family of Java based computer products for neural network applications.

Temporal Difference Learning Project -

http://www.geocities.com/chen_levkovich/tdlearningproject.html

Java sources for temporal difference learning Random Walk and Tic Tac Toe.

N.A.R.I.A. - http://naria.karasuma.net

An open project about simulating human-like intelligence with the help of neural networks.

Neural Network Framework - http://www.nnfw.org

Class framework to create neural networks with arbitrary topology and mixed type of neurons, developed for research purpose. Includes technical information and discussion mailing-list.

Tom-ato'sOCR - http://www.thomastannahill.com/tom-ato/toc2.html

Showcases a java applet designed to recognize hand printed digits using a neural network. Digits can be drawn using the mouse and recognized by the applet in the browser.

Cortex - http://cortex.snowseed.com/cortex.htm

A back propagation neural network application.

Lightweight Neural Network++ - http://lwneuralnetplus.sourceforge.net/

Free software project. Implements a general feed forward neural network and some training techniques.

libF2N2 - http://libf2n2.sourceforge.net/

An open source neural network library. Implements feedforward neural network classes in multiple languages including C++ and PHP.

Java library - http://aydingurel.brinkster.net/neural

Open source Java implementation of feed-forward neural nets: multi-layer perceptrons, generalized and modular feed-forward networks.

Penguinwerks - http://www.penguinwerks.com

Open source neural network library to create multi-layer perceptrons. Written in C#.

LTF-Cimulator - http://rainbow.mimuw.edu.pl/~mwojnar/ltfcim/

LTF-C neural networks simulator for solving classification problems.

Neuropilot Project - http://freespace.virgin.net/michael.fairbank/neuropilot/

Showcases a java applet demo of a trained neural network piloting a lunar-lander type spacecraft over landscapes of various complexity.

Stuttgart Neural Network Simulator - http://www-ra.informatik.uni-tuebingen.de/SNNS/

Description of the features of the Unix and X11 based simulator, information about how to obtain the SNNS sources and an online user manual.

NeuroMine - http://www.neuromine.com/

Neural network COM+ components and development environment for forecasting and data analysis. Supports several algorithms. Trial version is available.

Neurak - http://www.gameroom.com/quaternions/

A freeware environment for development and application of artificial neural networks.

Genesis - http://www.genesis-sim.org/GENESIS/

A platform for simulating complex neural systems.

ECANSE - http://www.siemens.at/ecanse/

Provides a development environment for the design, simulation and testing of neural networks and their applications up to the production of an optimized software solution.

APENDICE B : FUNCIONES DE MATLAB

```
function net = newff(pr,s,tf,btf,blf,pf)
%NEWFF Create a feed-forward backpropagation network.
%
%  Syntax
%
%    net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
%
%  Description
%
%    NEWFF(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF) takes,
%      PR  - Rx2 matrix of min and max values for R input elements.
%      Si  - Size of ith layer, for Nl layers.
%      TFi - Transfer function of ith layer, default = 'tansig'.
%      BTF - Backprop network training function, default = 'trainlm'.
%      BLF - Backprop weight/bias learning function, default = 'learngdm'.
%      PF  - Performance function, default = 'mse'.
%    and returns an N layer feed-forward backprop network.
%
%    The transfer functions TFi can be any differentiable transfer
%    function such as TANSIG, LOGSIG, or PURELIN.
%
%    The training function BTF can be any of the backprop training
%    functions such as TRAINLM, TRAINBFG, TRAINRP, TRAINGD, etc.
%
%    *WARNING*: TRAINLM is the default training function because it
%    is very fast, but it requires a lot of memory to run.  If you get
%    an "out-of-memory" error when training try doing one of these:
%
%    (1) Slow TRAINLM training, but reduce memory requirements, by
%        setting NET.trainParam.mem_reduc to 2 or more. (See HELP TRAINLM.)
%    (2) Use TRAINBFG, which is slower but more memory efficient than TRAINLM.
%    (3) Use TRAINRP which is slower but more memory efficient than TRAINBFG.
%
%    The learning function BLF can be either of the backpropagation
%    learning functions such as LEARNGD, or LEARNGDM.
%
%    The performance function can be any of the differentiable performance
%    functions such as MSE or MSEREG.
%
%  Examples
%
```

```
%    Here is a problem consisting of inputs P and targets T that we would
%    like to solve with a network.
%
%      P = [0 1 2 3 4 5 6 7 8 9 10];
%      T = [0 1 2 3 4 3 2 1 2 3 4];
%
%    Here a two-layer feed-forward network is created.  The network's
%    input ranges from [0 to 10].  The first layer has five TANSIG
%    neurons, the second layer has one PURELIN neuron.  The TRAINLM
%    network training function is to be used.
%
%      net = newff([0 10],[5 1],{'tansig' 'purelin'});
%
%    Here the network is simulated and its output plotted against
%    the targets.
%
%      Y = sim(net,P);
%       plot(P,T,P,Y,'o')
%
%    Here the network is trained for 50 epochs.  Again the network's
%     output is plotted.
%
%      net.trainParam.epochs = 50;
%      net = train(net,P,T);
%      Y = sim(net,P);
%       plot(P,T,P,Y,'o')
%
%  Algorithm
%
%    Feed-forward networks consist of Nl layers using the DOTPROD
%    weight function, NETSUM net input function, and the specified
%    transfer functions.
%
%    The first layer has weights coming from the input.  Each subsequent
%    layer has a weight coming from the previous layer.  All layers
%    have biases.  The last layer is the network output.
%
%    Each layer's weights and biases are initialized with INITNW.
%
%    Adaption is done with ADAPTWB which updates weights with the
%    specified learning function. Training is done with the specified
%    training function. Performance is measured according to the specified
%

function [net,tr]=train(net,P,T,Pi,Ai,VV,TV)
%TRAIN Train a neural network.
```

```
%
%  Syntax
%
%    [net,tr] = train(NET,P,T,Pi,Ai)
%    [net,tr] = train(NET,P,T,Pi,Ai,VV,TV)
%
%  Description
%
%    TRAIN trains a network NET according to NET.trainFcn and
%    NET.trainParam.
%
%    TRAIN(NET,P,T,Pi,Ai) takes,
%      NET - Network.
%      P   - Network inputs.
%      T   - Network targets, default = zeros.
%      Pi  - Initial input delay conditions, default = zeros.
%      Ai  - Initial layer delay conditions, default = zeros.
%    and returns,
%      NET - New network.
%      TR  - Training record (epoch and perf).
%
%    Note that T is optional and need only be used for networks
%    that require targets.  Pi and Pf are also optional and need
%    only be used for networks that have input or layer delays.
%
%  Examples
%      net = train(net,p,t);
%      y2 = sim(net,p)
%      plot(p,t,'o',p,y1,'x',p,y2,'*')
%
%  Algorithm
%
%    TRAIN calls the function indicated by NET.trainFcn, using the
%    adaption parameter values indicated by NET.trainParam.
%
%    Typically one epoch of training is defined as a single presentation
%    of all input vectors to the network.  The network is then updated
%    according to the results of all those presentations.
%
%    Training occurs until a maximum number of epochs occurs, the
%    performance goal is met, or any other stopping condition of the
%    function NET.trainFcn occurs.
%
%    Some training functions depart from this norm by presenting only
%    one input vector (or sequence) each epoch. An input vector (or sequence)
%    is chosen randomly each epoch from concurrent input vectors (or sequences).
```

```
%    NEWC and NEWSOM return networks that use TRAINWB1, a training function
%    that does this.
%
%  See also SIM, INIT, ADAPT


function [Y,Pf,Af]=sim(net,P,Pi,Ai)
%SIM Simulate a neural network.
%
%  Syntax
%
%    [Y,Pf,Af] = sim(net,P,Pi,Ai)
%    [Y,Pf,Af] = sim(net,{Q TS},Pi,Ai)
%    [Y,Pf,Af] = sim(net,Q,Pi,Ai)
%
%  Description
%
%    SIM simulates neural networks.
%
%    [Y,Pf,Af] = SIM(net,P,Pi,Ai) takes,
%      NET - Network.
%      P   - Network inputs.
%      Pi  - Initial input delay conditions, default = zeros.
%      Ai  - Initial layer delay conditions, default = zeros.
%    and returns:
%      Y   - Network outputs.
%      Pf  - Final input delay conditions.
%      Af  - Final layer delay conditions.
%
%    Note that arguments Pi, Ai, Pf, and Af are optional and
%    need only be used for networks that have input or layer delays.
%
%
%  Examples
%
%    Here NEWP is used to create a perceptron layer with a
%    2-element input (with ranges of [0 1]), and a single neuron.
%
%      net = newp([0 1;0 1],1);
%
%    Here the perceptron is simulated for an individual vector,
%    a batch of 3 vectors, and a sequence of 3 vectors.
%
%      p1 = [.2; .9]; a1 = sim(net,p1)
%      p2 = [.2 .5 .1; .9 .3 .7]; a2 = sim(net,p2)
%      p3 = {[.2; .9] [.5; .3] [.1; .7]}; a3 = sim(net,p3)
```

```
%
%    Here NEWLIND is used to create a linear layer with a 3-element
%    input, 2 neurons.
%
%      net = newlin([0 2;0 2;0 2],2,[0 1]);
%
%    Here the linear layer is simulated with a sequence of 2 input
%    vectors using the default initial input delay conditions (all zeros).
%
%      p1 = {[2; 0.5; 1] [1; 1.2; 0.1]};
%      [y1,pf] = sim(net,p1)
%
%
%  Algorithm
%
%    SIM uses these properties to simulate a network NET.
%
%      NET.numInputs, NET.numLayers
%      NET.outputConnect, NET.biasConnect
%      NET.inputConnect, NET.layerConnect
%
%    These properties determine the network's weight and bias values,
%    and the number of delays associated with each weight:
%
%      NET.inputWeights{i,j}.value
%      NET.layerWeights{i,j}.value
%      NET.layers{i}.value
%      NET.inputWeights{i,j}.delays
%      NET.layerWeights{i,j}.delays
%
%    These function properties indicate how SIM applies weight and
%    bias values to inputs to get each layer's output:
%
%      NET.inputWeights{i,j}.weightFcn
%      NET.layerWeights{i,j}.weightFcn
%      NET.layers{i}.netInputFcn
%      NET.layers{i}.transferFcn
%
%    See Chapter 2 for more information on network simulation.


function gensim(net,st)
%GENSIM Generate a SIMULINK block to simulate a neural network.
%
%  Syntax
%
```

```
%    gensim(net,st)
%
%  Description
%
%    GENSIM(NET,ST) takes these inputs,
%      NET - Neural network.
%      ST  - Sample time (default = 1).
%    and creates a SIMULINK system containing a block which
%    simulates neural network NET with a sampling time of ST.
%
%    If NET has no input or layer delays (NET.numInputDelays
%    and NET.numLayerDelays are both 0) then you can use -1 for ST to
%    get a continuously sampling network.
%
%  Example
%
%    net = newff([0 1],[5 1]);
%    gensim(net)
```

APENDICE C : ARTICULOS SELECCIONADOS